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 instruction 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, Decode, Execute, Mem, WriteBack.
For simplicity, we have not implemented a Branch prediction unit and a Forward Unit.
So in case of dependencies, our processor is less powerful.

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). Instruction and data of the entire procssor are represented on 32 bits. As specification says, Instruction and Data memory

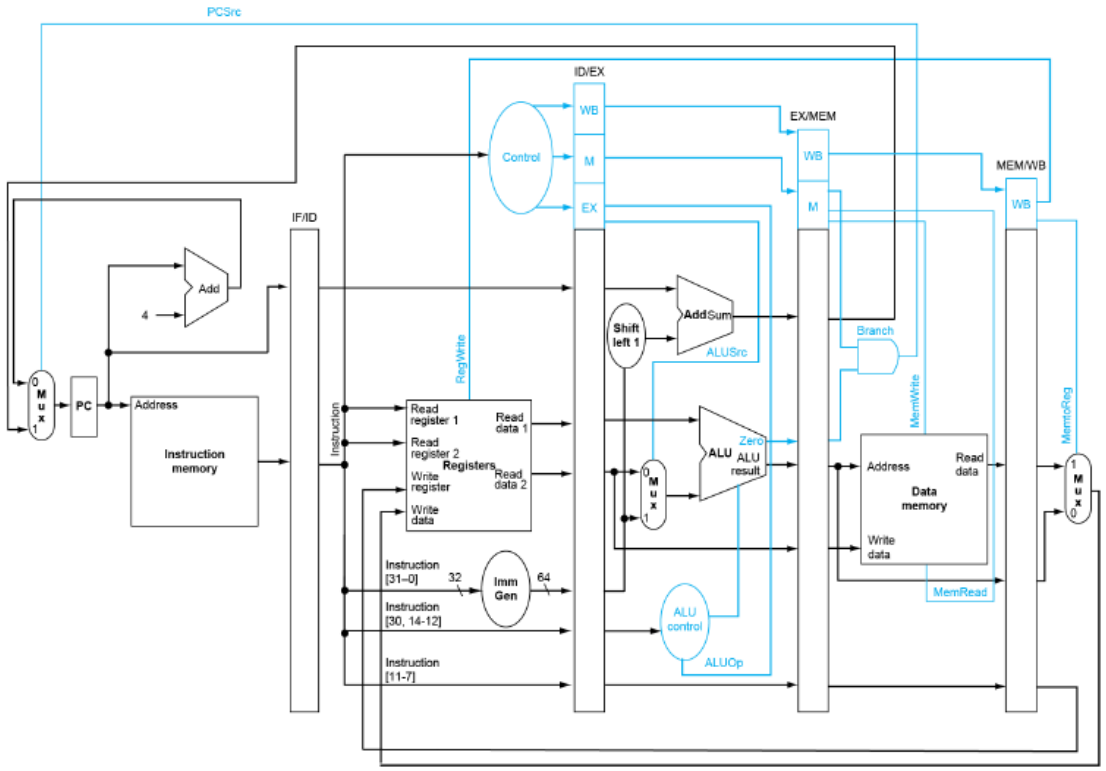

Figure 2.1: General structure

was not included. We have implemented it in the testbench.
Here we have tried to impement most of the single blocks in a Structural way.
Only Register file, alu and all filpflops are implemented as Behavioral.

On Figure 2.2 it is showed how we have organized the VHDL files. Now more details on singolar block implementation.

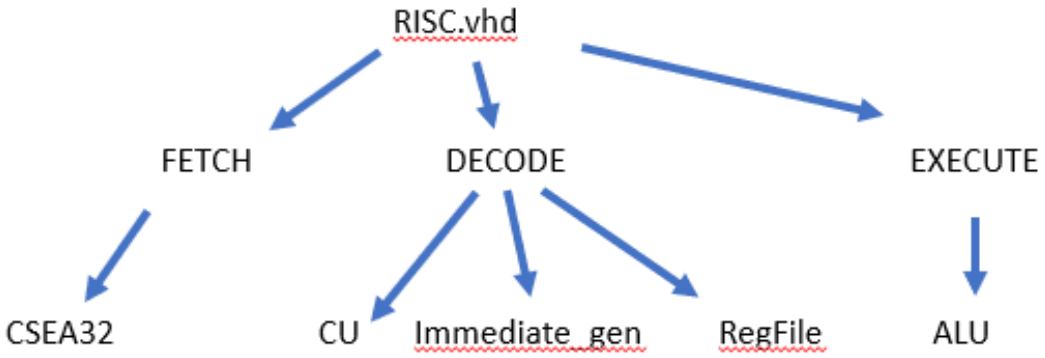

Figure 2.2: General VHDL structure

2.1 Fetch

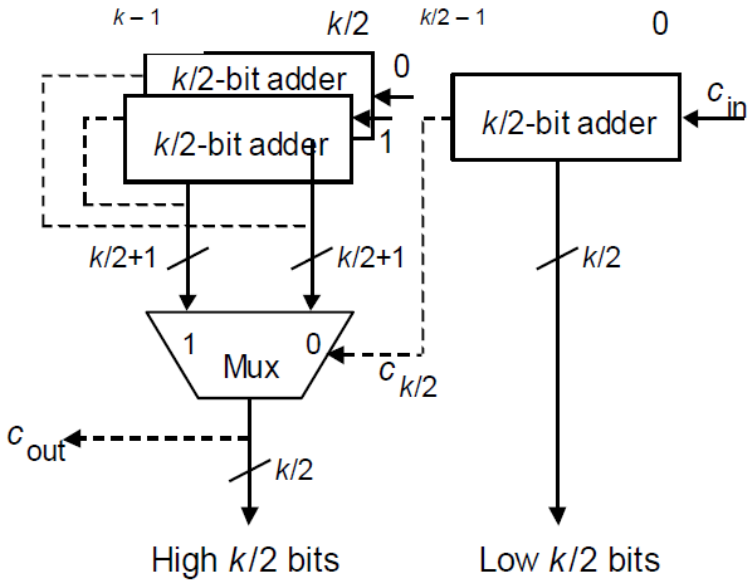

Figure 2.3: CSEA structure

To update the PC we have implemented a Carry Select Adder.
Each singolar blocks are compose of a 8 bit Ripple carry adder. In this way we have reduced the delay.

2.2 Decode

The register file is behavioral. It's composed of 32 registers on 32 bits.
At each rising edge of the clock, read two value given the addresses Register_read1 and 2.
If the write enable is 1, we write in the register location. It have this port:

- Write enable port;
- Register read1;
- Register read2;
- Write register;
- Write data;
- Read Data1;
- Read Data1;
- Clock;
- Reset;

The Immediate_generate block read the instruction and generate the Immediate value. So based

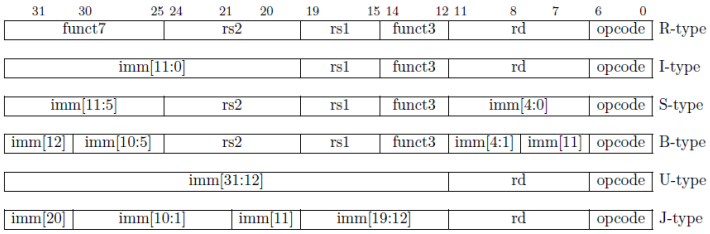

Figure 2.4: Instruction organization

on which instruction we are decoding, organize a 32 bits Immediate value. Here we have placed the Control Unit, see next chapter for further details.

2.3 Execute

The adder that create the new program counter is again implemented with a 32 bits CSEA. Then we have the Alu that receive two inputs value and the Opcode specify which instruction implement.

2.4 Memory

This module is implemented in the RISC top entity. Contain only the AND port that generate the selection signal for the multiplexer on the fetch phase.

2.5 Write back

This module is implemented in the RISC top entity. Contain a multiplexer of 4 inputs and decide 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. For 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, 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 it is a finite state machine that switches from a state to another at every clock cycle, generating a sequence of individual bits represent various control signals (i.e. the control word).

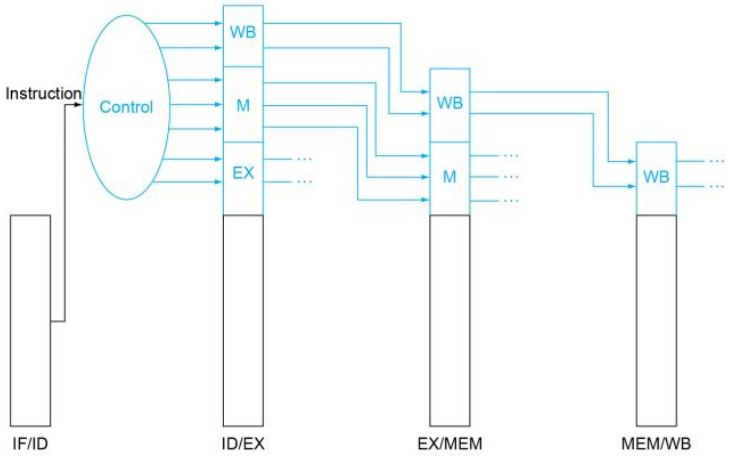

Figure 3.1: General CU structure

3.1 Control Word Generation

The component receives as input the instruction and based on the instruction's Opcode, it decides which CW it has to send to the output.
This selection is based on the bits in position 2, 4, 5, 6.

This because we have found that these are the bits which, once concatenated together, identify each instruction uniquely.
These bits are used to access to the matrix which generates the corrispondent CW.
Each instruction has its own control signals, but they are the same for instructions belonging to the same class (exept for some that belong to a class, but they have different control signals):

	REG1 LATCH EN	REGL 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	REGDL 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 is the one dedicated to the generation of the code destinated to the ALU, in order to determine its behavior. Its sensitivity list comprehend the instruction's OPCODE and the instruction's FUNCT and it follows the following scheme:

- 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";

CHAPTER 4

Testbench

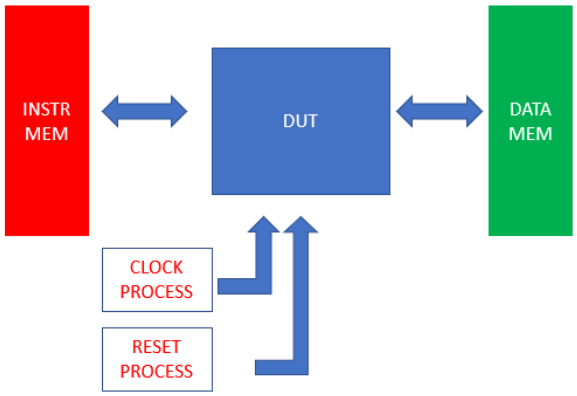

Figure 4.1: Testbench organizatiopn

The Clock process generate the clock signal for the entire processor.
We have decided to adopt a 20ns as a period for the testbench.
So our processor is tested at 50MHz.
Our goal was to verify that the processor works correctly, withouth considering the performance.
For this reason we have used a low frequency.
The reset process initialize all the component. This signal is active for 15ns then it is disabled.
For what concern the Instruction and Data memory we have implemented both like array of 32 bits.
This because both memory are not so big, we have decided to implement them without resorting to a specific structure.

So for the Instruction memory we have taken the program to test the architecture(Given in the specification)
and we have written all the instruction manually. All of them are interleaved by 3 empty instruction, to respect the
program counter update(PC+4). The results are written in the Register file or in the Data memory, depending on which instruction is executed.

```
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

CHAPTER 5

Synthesis AND Timing

We have synthetized the architecture and we have created the report for the area and the timing.

5.1 Area analysis

Firstly, after the synthesis, we have retrieved the area occupied by our processor.

```
****************************************
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

5.2 Timing analysis

We have run the synthesis with a clock of 0s. So looking the slack,
we can retrieve the maximal clock frequency that our architecture can reach. Then we have applied that contraint and we have run again the synthesis.
Finally we have implemented the routing phase. The result on Figure 5.6.

```
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

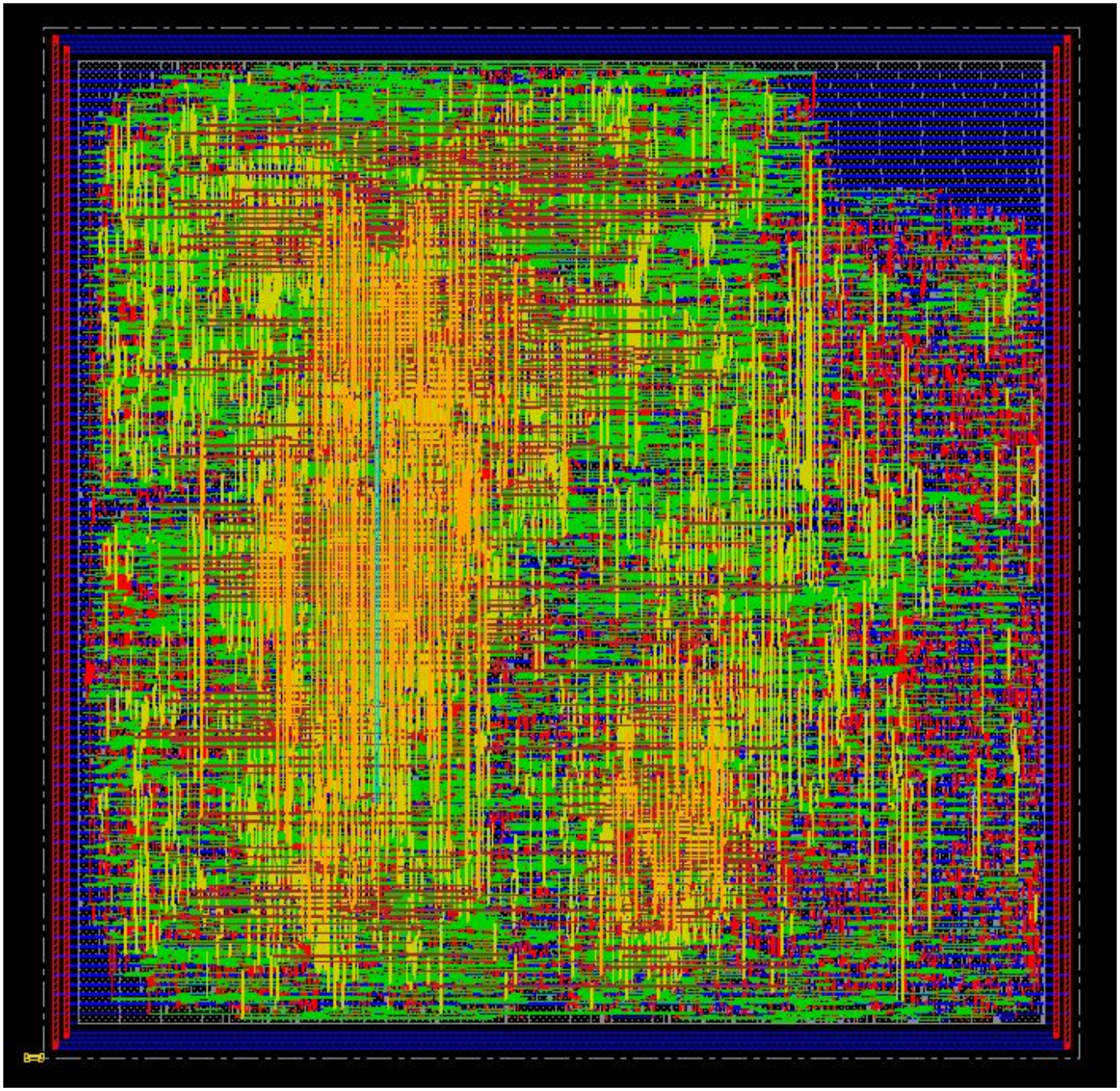

Figure 5.6: Routing