Politecnico di Torino

Microelectronic Systems

DLX Microprocessor: Design & Development

Final Project Report

Master Degree in Electronics & Computer Engineering

Referents: Prof. Mariagrazia Graziano, Giovanna Turvani

Authors: Group `ms22.14`

Fulvio **Castello**, Lavinia **Comerro**, Marcos Ricardo **Lawrie**

July 2022

Contents

List of Figures

CHAPTER 1

Introduction

The aim of this project was to create a fully-functional DLX microprocessor, entirely designed using the VHDL hardware design language. A team of three people contributed to the development of a custom architecture that would reflect what was taught during the "Microelectronic Systems" course, including all the essential features plus some additional ones that came from necessity and/or simple personal interest.

The whole block consists of a pipelined structure made by five contiguous stages, each contributing to a different functionality in order for a single instruction to take place during the same amount of clock cycles:

1. **IF** — Instruction Fetch
2. **ID** — Instruction Decode/Register Fetch
3. **EXE** — Execute/Effective Address
4. **MEM** — Memory Access/Branch Completion
5. **WB** — Write Back.

Each piece of information is represented by a 32-bits long `std_logic_vector`, which is also the required format for firmware commands. There's a total of 32 general-purpose registers, and the Datapath has been extended in order to include all the functionalities indicated in the advanced instruction subset. The Arithmetic Logic Unit was adapted as to be capable of performing all the related operations in a behavioral way as well.

All the rest of the Instruction Set is still compatible with the current implementation (even Floating Point operations), but all the "unknown" instructions will just resolve to a *nop*: this way, it would become much easier to add the support for those ones as well if the project was to be resumed at a later time. As a matter of fact, after creating the required blocks, it becomes almost a matter of plug-and-play.

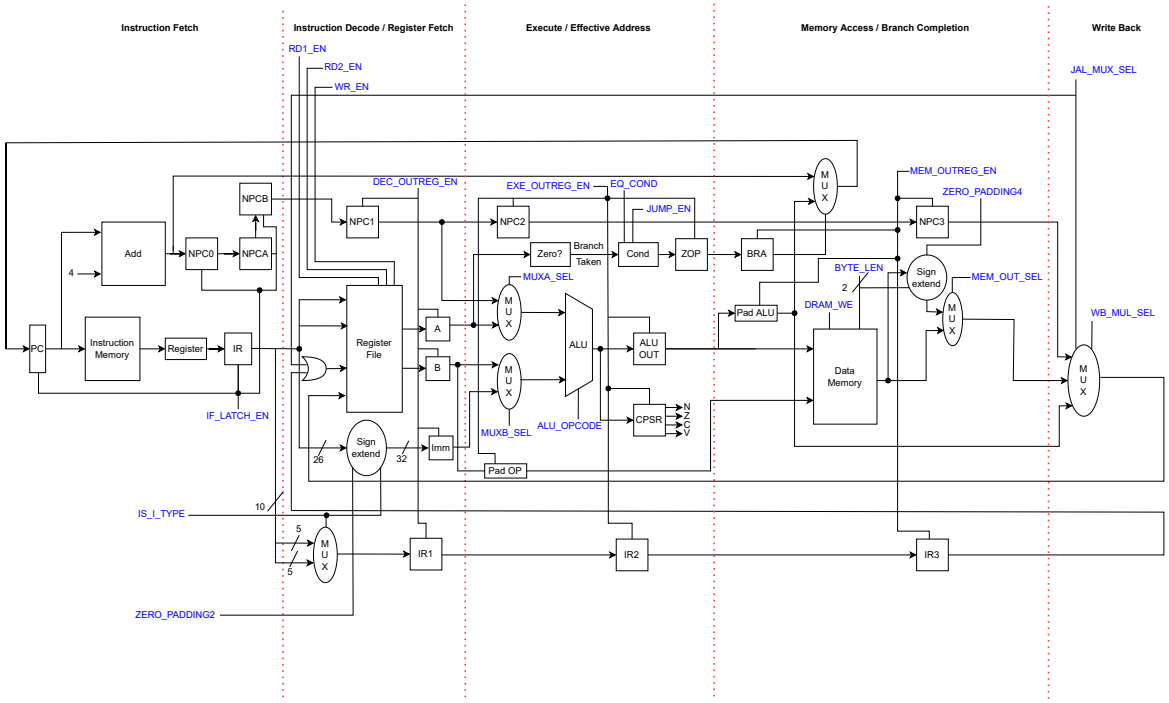

Figure 1.1: Overview of the main blueprint showing the current structure of the DLX.

CHAPTER 2

Implementation

The required constants, generics and types for this project are contained in one main package called `dlx_utils`, which must always be compiled as the first entry. All the entities described below inherently depend on it, while also making use of the standard `IEEE` library (with the exception of Memories, which include `STD` as well).

2.1 Fundamental blocks

The implementation of the DLX followed a bottom-up approach in which the lowest level building blocks were to be instantiated first. In the following, they are listed in alphabetical order.

2.1.1 `alu`

This behavioral unit mainly consists in one prominent `CASE` statement, whose branches are managed by an `ALU_MSG` special signal. The latter is an enumerated type representing all the possible operations linked to the Instruction Set. An internal variable with a 33rd additional bit is used to compute and handle both the result and some condition code flags. Of course, the block in question is wholly fault-tolerant.

It's important to notice that two operations were added (`R_mult` and `R_multu`) in order for them to work with an integer register file: their OPCODE was switched from `0x01` to `0x00` while the corresponding FUNC remained tha same, as to effectively convert them from FP to R-Type. In this context, both input operands had to be (equally) truncated to Half Word length in order to generate a result on 32-bits only.

2.1.2 `cond_branch`

A combinational set of two logic gates used to determine whether to perform a branch or not, with a dataflow description that directly optimizes the intended behavior.

2.1.3 `cpsr`

The Current Program Status Register was inspired by ARM architectures and contains the four main condition code flags generated by any given mathematical operation:

- **N** — Negative
- **Z** — Zero

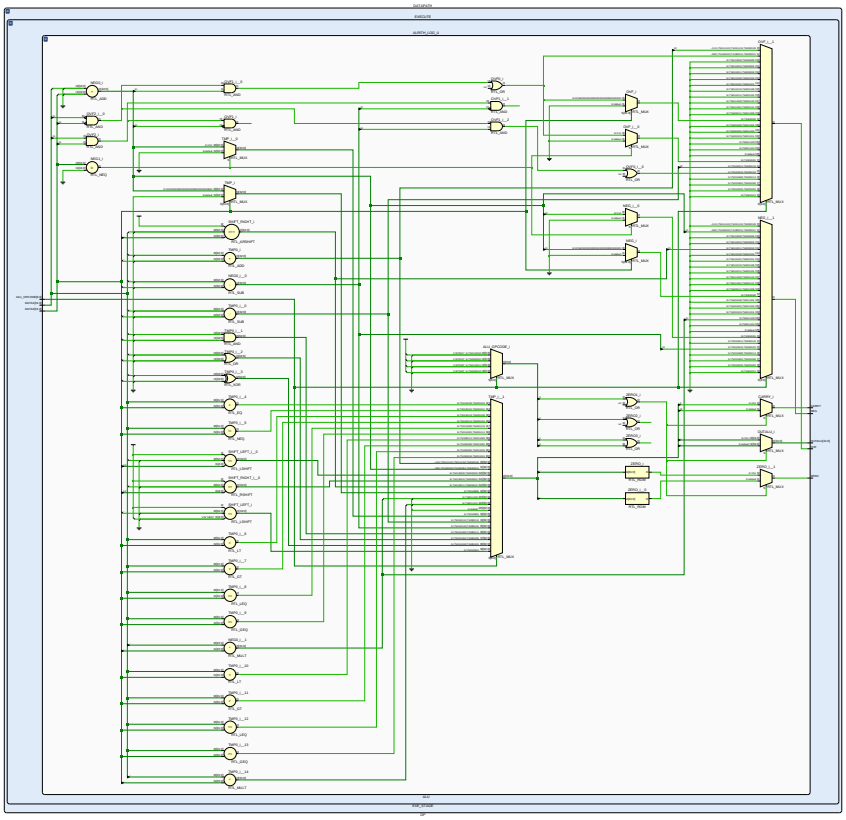

Figure 2.1: Schematic of the behavioral `alu`.

- **C** — Carry
- **V** — Overflow.

Its outputs only reach the Datapath level, dying off before reaching the actual I/O of the microprocessor itself: as a matter of fact, they are intended for internal use only, and possibly provide an additional tool for implementing new *conditional* instructions. They are left unused in the current state of this project.

2.1.4 `gen_mux21`

A dataflow description of the classic **2 to 1** multiplexer with variable-width operands.

2.1.5 `gen_mux41`

A slightly more complex revision of the above component, this **4 to 1** multiplexer requires a considerably larger amount of logic gates with two selectors to drive them. In the current state of the Datapath, its last two inputs ("10" and '11") are short-circuited, making the second selection bit a "don't care" whenever the first one is set to '1'.

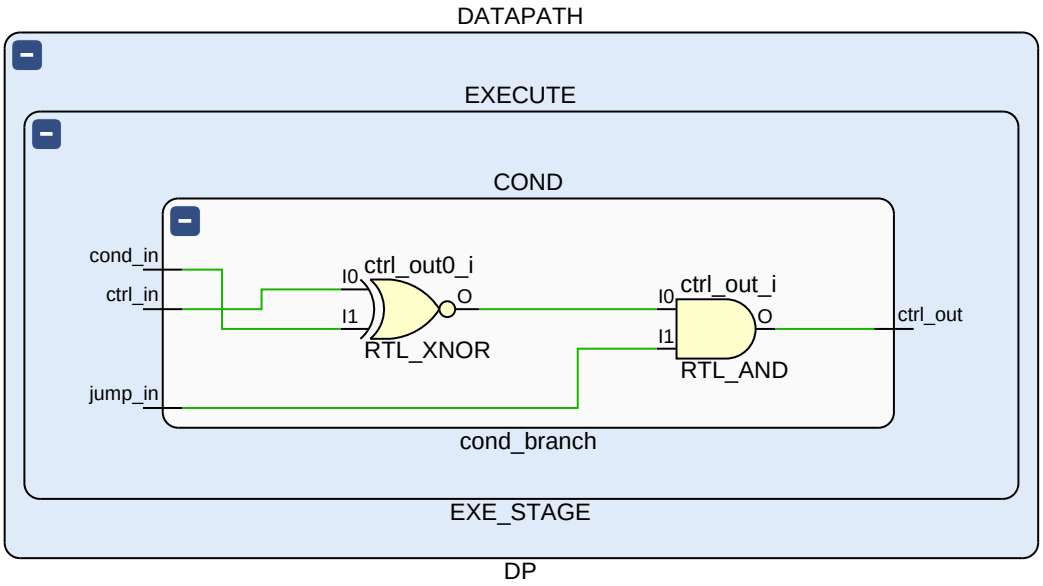

Figure 2.2: Schematic of `cond_branch`.

2.1.6 `gen_reg`

A behavioral implementation of a variable-width register, sensitive to the rising edge of a clock signal, including an asynchronous reset and separate loading capabilities.

2.1.7 `pc_add`

A behavioral adder used to compute the Next Program Counter with a fixed offset—the second operand is a constant equal to 4 in this context. Computation is **signed** in order to include backward jumps as well.

2.1.8 `reg_file`

This set of registers consists in a straightforward process that manages an internal memory, which is properly structured and accessed in compliance with the needs of the given ISA. Register `r0` is read-only, and set to an all-zero value during reset (asynchronous) in a pure MIPS-like fashion.

One interesting characteristic is the direct bypass of the internal signals in case of simultaneous reading and writing operations on the same address: the corresponding location will of course take on the provided value, but the latter will immediately be forwarded to the output port instead of still being gathered from that same cell. This mechanism was implemented in an effort to avoid causing undefined or unstable events during concurrent operations on the same register, providing an even

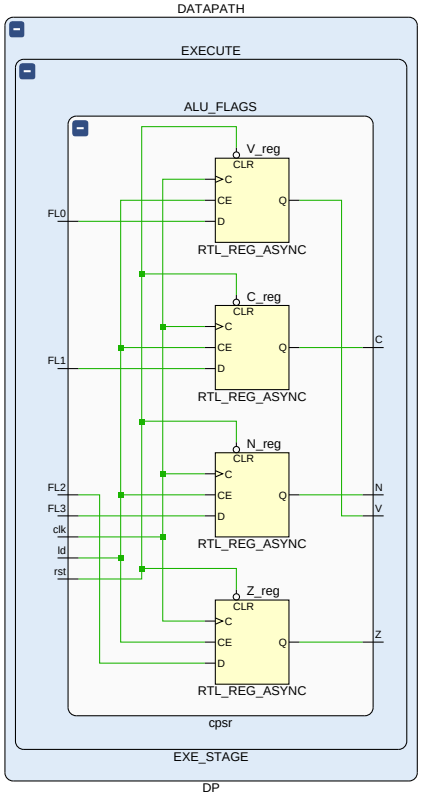

Figure 2.3: Schematic of the `cpsr`.

faster solution to such issue in terms of delay.

2.1.9 `sign_ext_alt`

An alternate, more complex version of the subsequent component that provides sign extension and zero padding capabilities. This is a highly specialized block that is only used inside the Memory pipeline stage, and the performed operations are as follows:

`ctrl_in`	`zero_padding`	Action
0	0	Extend byte sign
0	1	Zero pad byte
1	0	Extend half-word sign
1	1	Extend actual sign

2.1.10 `sign_ext`

This is a simplified version of the previous block, used inside the Decode stage this time. The implemented functionalities are again quite specific:

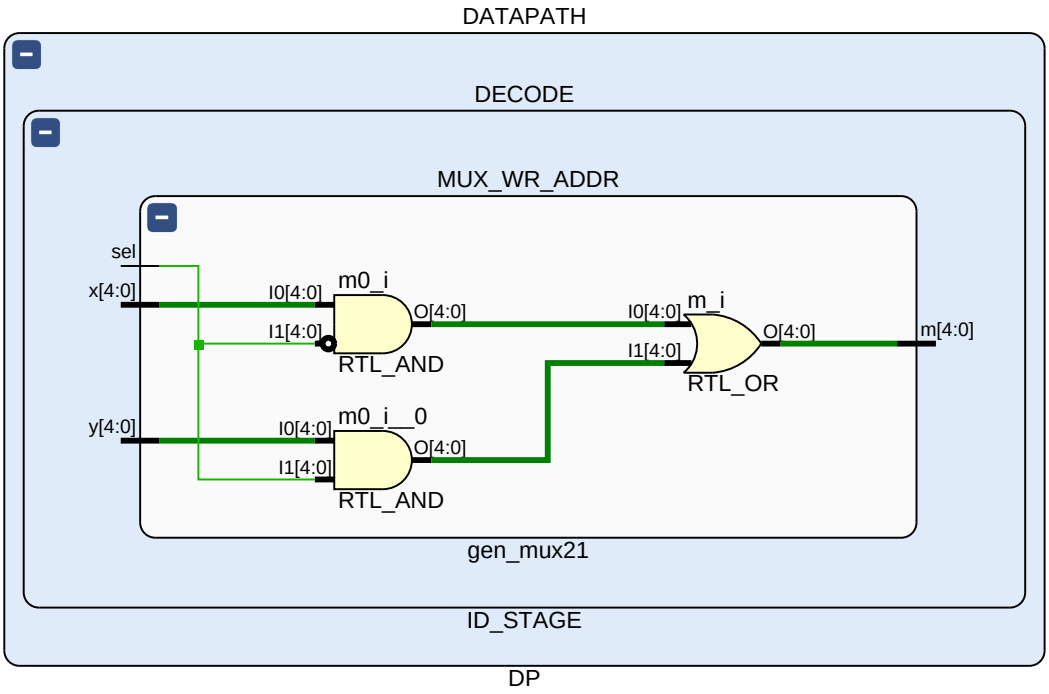

Figure 2.4: Schematic of a `gen_mux21`.

`ctrl_in`	`zero_padding`	Action
0	-	Extend actual sign
1	0	Extend internal sign
1	1	Zero pad actual length

2.1.11 `zero_check`

A trivial combinational unit aimed at recognizing whenever its variable-width input vector is made of only zeros, used in conjunction with `cond_branch` to generate the selector (subsequently made synchronous) of the multiplexer responsible for branches.

2.2 Pipeline stages

These five macro-structures represent the main building blocks of the Datapath, with the purpose of combining and linking all the previously described low-level components in order to create a unique and coherent unit. The latter would have to perform any given functionality during the span of five clock cycles, making timing synchronization of paramount importance.

Because of this last aspect, these mid-level stages are here listed following their temporal order inside the microprocessor pipeline.

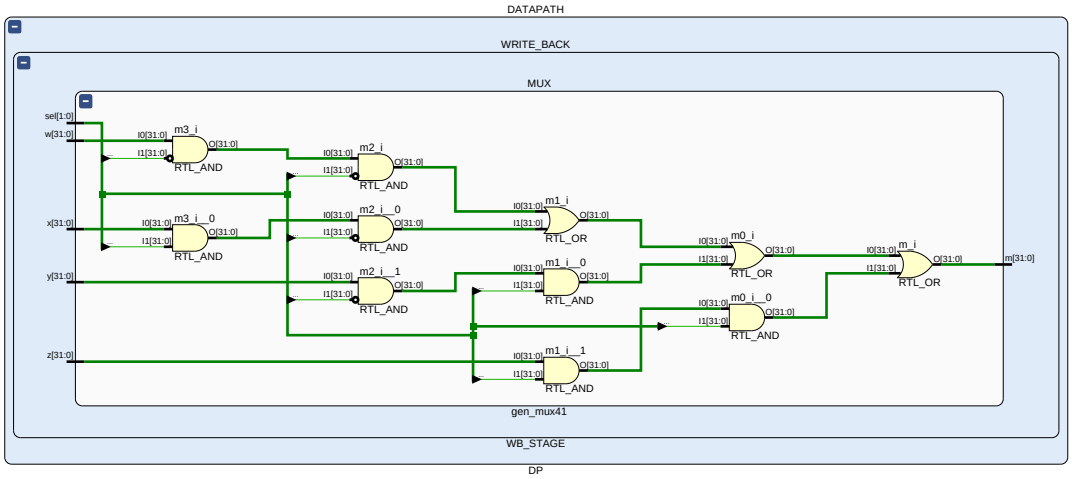

Figure 2.5: Schematic of a `gen_mux41`.

2.2.1 `IF`

A very straightforward structural architecture that includes the interface with the external Instruction Memory. It generates the address to the next operation that will be fed to the whole pipeline, whilst also taking in the current one that needs to be tackled with.

An array of synchronization registers had to be included in the path related to the Next Program Counter as to align it the one related to actual instructions instead (this concept will become clearer during the explanation of the `DLX` layer). This will obviously cause an initial delay between the arrival time of every command and its actual execution, but the timing interval between subsequent ones will still comply to the **5-cycle** pipeline of choice.

2.2.2 `ID`

A collection of blocks that are needed to store the actual data that travels throughout the whole design in a very fast internal location. Many synchronization registers are included, while the first sign extension block performs the required operation on the Immediate field depending on the type of instruction that is being analyzed.

One multiplexer is dedicated to selecting the correct field of R-Type versus I-Type instructions that would indicate the destination register RD. Moreover, a simple OR port in front of the writing address of the register file was introduced to easily place the NPC inside `r31` during “Jump and Link” operations (as per specifications).

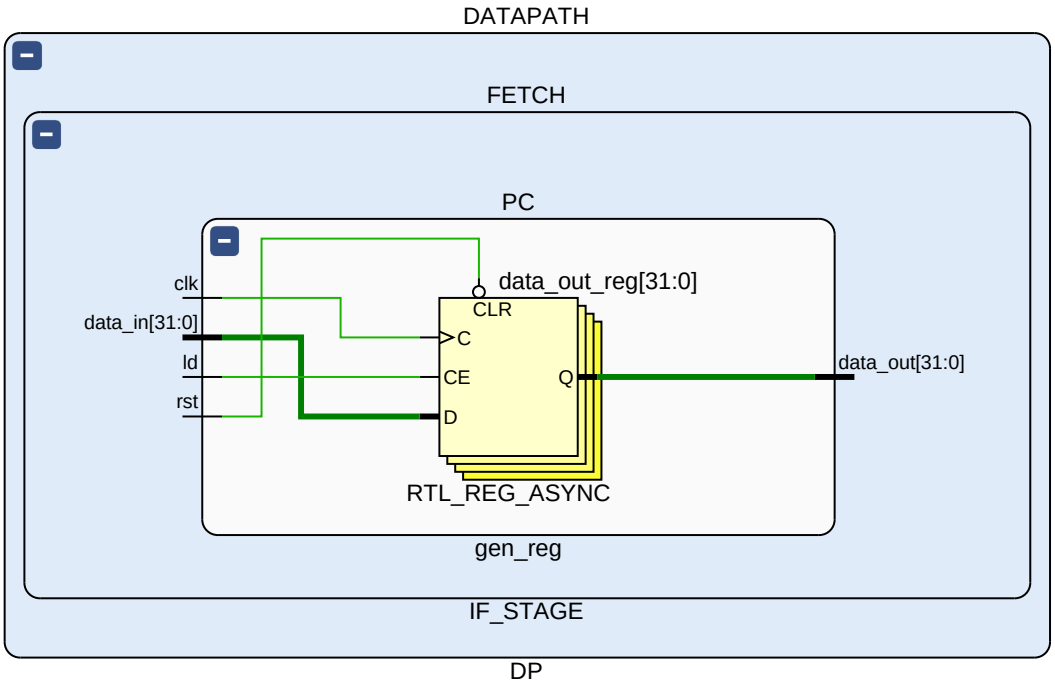

Figure 2.6: Schematic of a `gen_reg`.

2.2.3 `EXE`

The core of computation of the DLX microprocessor, it works on integer values only. Two multiplexers select the correct operands to be sent to the ALU, while a pair of combinational blocks generates the correct control signals during branch evaluations.

A noticeable amount of synchronization register are instantiated alongside the register dedicated to storing the status flags computed by the same ALU. The latter has, in turn, a dedicated internal signal aimed at selecting the correct operation to perform during the execution of any given instruction.

2.2.4 `MEM`

This stage contains another external interface, this time with the external Data Memory. It is in fact capable of providing both addresses and data to the input ports of such peripheral block, whilst directly forwarding the required control signals to the same from the Control Unit itself.

Read values are instead selected by means of an output multiplexer, after possibly going through the alternate sign extension module, which is used whenever a value shorter than 32-bits has to be loaded. A second multiplexer picks the correct Program Counter that needs to be sent to the Instruction Fetch stage.

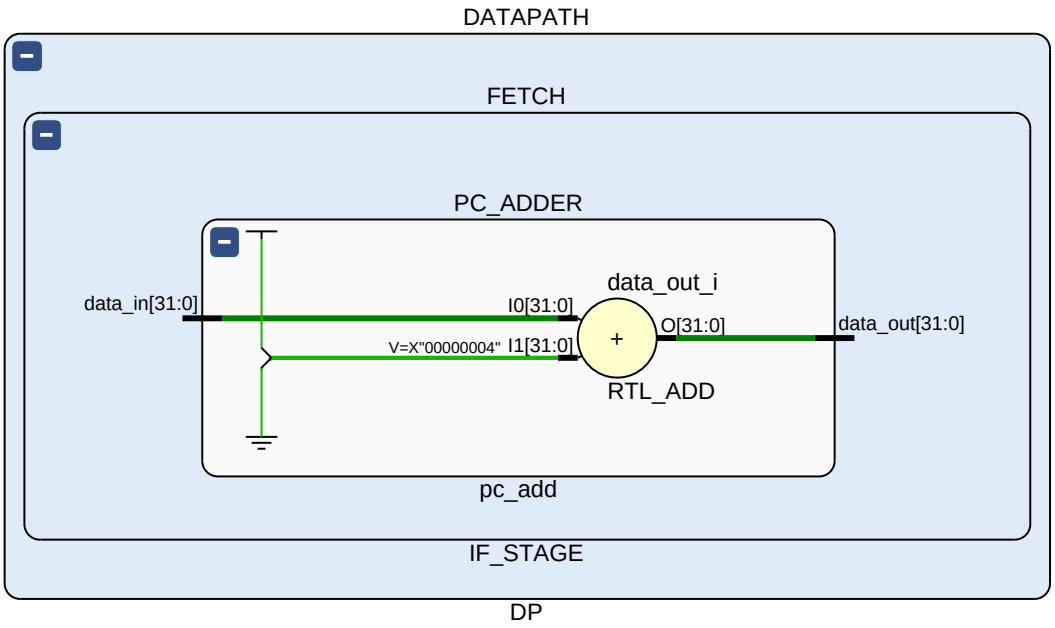

Figure 2.7: Schematic of `pc_add`.

2.2.5 `WB`

The simplest among all the pipeline stages, its only purpose is that of selecting the correct outcome that needs to be sent to the Register File. The `IR` signal goes through this stage untouched, while the 4-input multiplexer can only actually pick between three different alternatives:

`JAL_MUX_SEL`	`WB_MUX_SEL`	Selection
0	0	ALU Output
0	1	Memory Output
1	-	Next Program Counter

2.3 Higher level entities

Two main design units are eventually connected to each other in order to achieve the final architecture of the DLX microprocessor: in the following, the culminating blocks of the entire model are listed using a topological order.

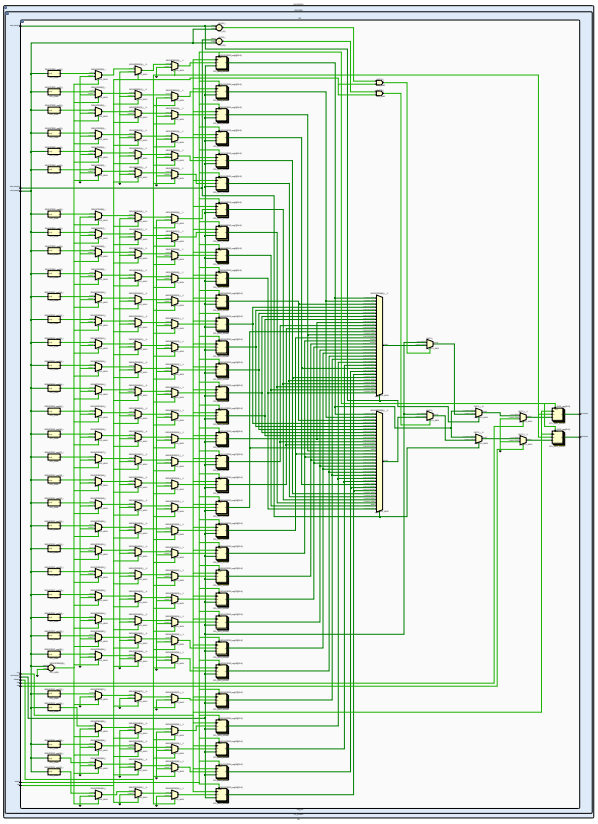

Figure 2.8: Schematic of the `reg_file`.

2.3.1 DP

The Datapath of this design consists in nothing more than an extensive set of connections between the five aforementioned pipeline stages, with a multitude of internal signals aimed at creating a clean and direct interface between them.

This composite structure is also responsible for communicating with both external Memories, actually generating the required I/O that has to later be handled by the top-level entity.

2.3.2 CU_HW

The Control Unit of this microprocessor is the module that actually manages the functionalities provided by the Datapath, feeding the latter with the correct combination of control bits at any given time. Its reset is still asynchronous, and it of course has to comply with the same 5-stage pipeline.

The chosen approach for this structure was a **hardwired** implementation, in which the control word codings are contained inside an internal LUT (read-only memory element). Starting from the input instruction, this module has to parse the two fields of interest, i.e. OPCODE and FUNC, in order to select the corresponding operation, ultimately driving all the other components located inside the architecture. Messages directed to both the ALU and a possible FPU are taken care of by means of custom VHDL types and dedicated processes.

Control words are picked from a dictionary of known values thanks to their unique identifier (whose hexadecimal values were provided inside the official specifications) in such a way that the

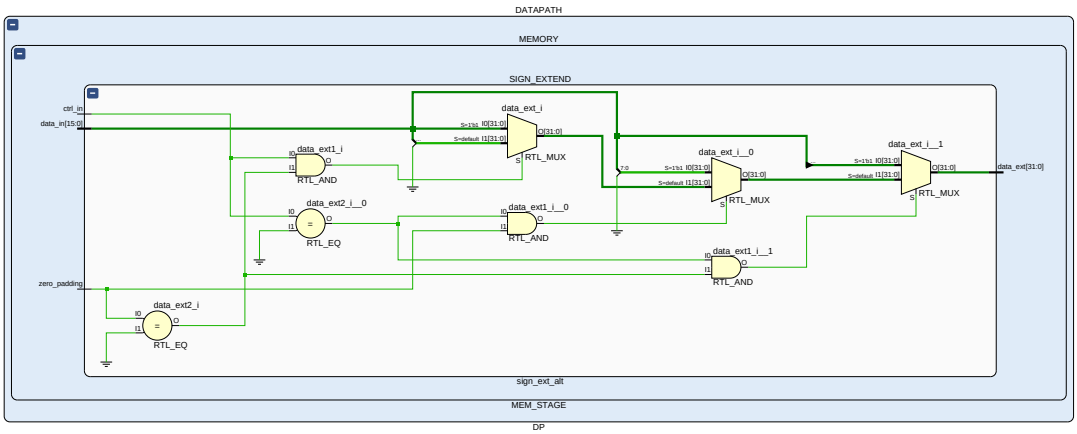

Figure 2.9: Schematic of `sign_ext_alt`.

output assignment gets reduced to a few subsequent steps:

1. Parsing and identification of the information coming from the Instruction Memory
2. Search operation inside the internal LUT (with several "holes" representing invalid entries)
3. Pipelined output assignment and message generation for the execution unit(s).

In case of an out-of-range request, or even a command that is yet to be implemented, this fault-tolerant architecture automatically returns a *nop*.

2.3.3 `DLX`

This component sits at the highest level of abstraction of the whole design space, combining the contributions of both the previously described modules. It is the main block that would later be simulated, synthesized and taken through the required physical design process.

At this point, it's important to highlight the essential role of an additional synchronization register that needed to be inferred (indirectly, through a VHDL process) between the Instruction Memory and the Datapath inside this layer—the Control Unit must be exactly **one** clock cycle in advance with respect to the latter in order to issue the correct coding at the right time.

The port list of this structure contains the main I/O of the DLX itself, while the whole set of generics needed inside all the previously mentioned modules are grouped right at this level.

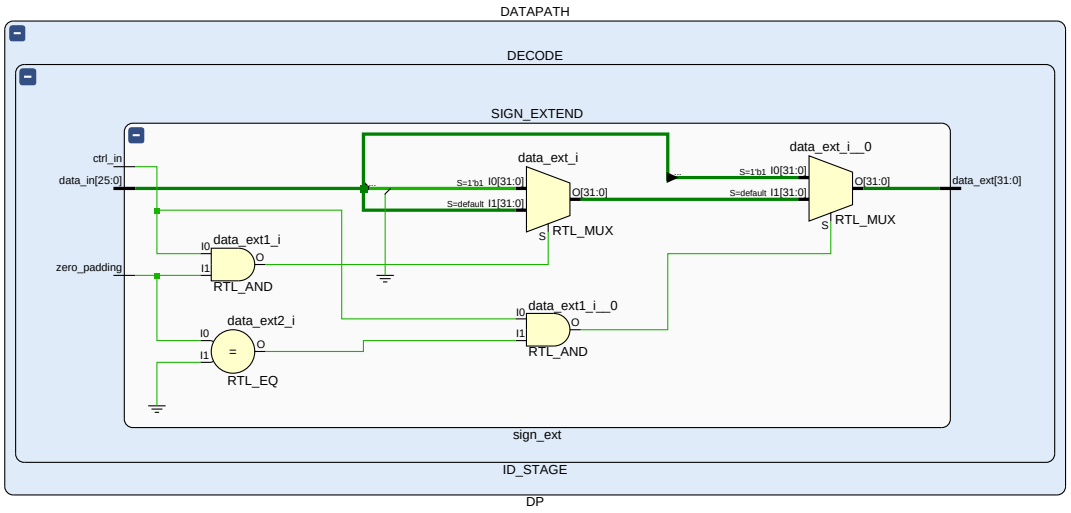

Figure 2.10: Schematic of `sign_ext`.

2.4 Memories

Two external Memories were used to test the design: an Instruction Memory and a Data Memory. They were implemented by strictly adhering to what's already included within the provided documentation.

2.4.1 `romem`

Used to instantiate a read-only memory that contains the set of instructions that will be fed to the microprocessor. The reset is active low and asynchronous, while reading operations are synchronous and can be delayed by any desired amount of clock cycles. The input firmware is gathered from an external text file, which must contain hexadecimals values only (once for each line).

2.4.2 `rwmem`

Instead represents a random access memory featuring the same characteristics of the previous module plus some additional capabilities. Reading is also synchronous, but now a dedicated control bus selects one format for data exchange amongst three possibilities:

An output file is finally generated with the same typesetting as the input one by means of a specialized VHDL procedure with the following prototype:

```
rewrite_content(data, path_file)
```

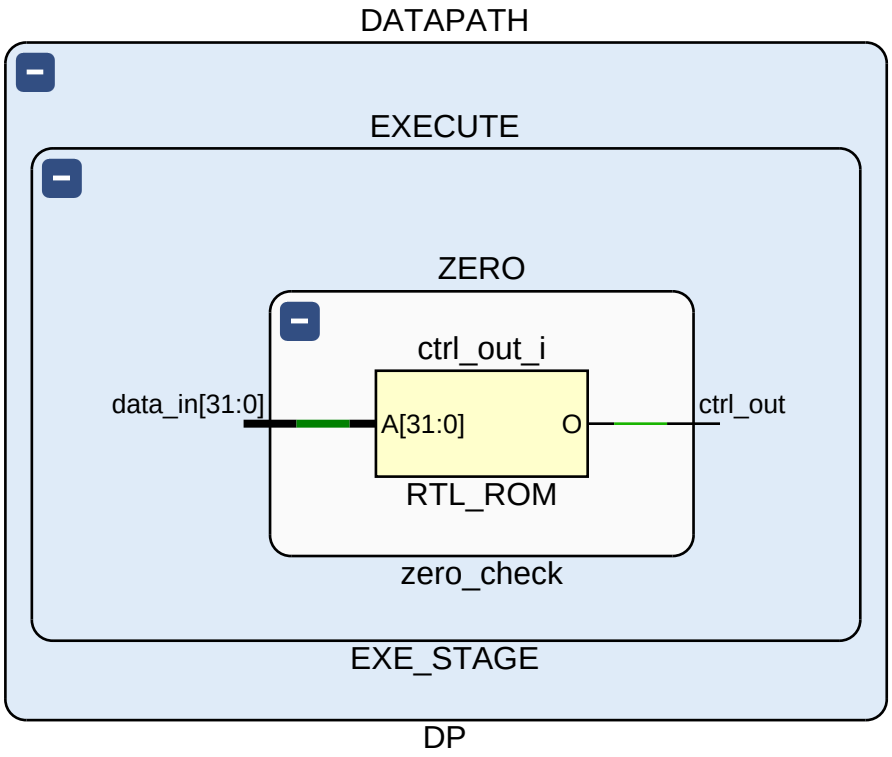

Figure 2.11: Schematic of `zero_check`.

BYTE_LEN	Unit	Length
00	Byte	8 bits
01	Half Word	16 bits
1-	Word	32 bits

where `data` is the piece of information being saved, while `file_path` is the location where those values have to be written into.

2.5 Testbenches

All the V&V steps were handled by means of a very extensive and complete set of testbenches located inside the "./test_bench/" directory. The whole assortment of components was thoroughly tested with the same bottom-up approach that was being used during the course of the implementation process.

The lower-level, more basic tests were manually compiled and carried on inside "Mentor Graphics QuestaSim", whereas everything starting from the ALU up was made handily available by means of a dedicated script. All these `.do` files can be found inside "./test_bench/scripts/" and have to be started inside the simulator by means of the following command:

```
do sim_MODULE.do
```

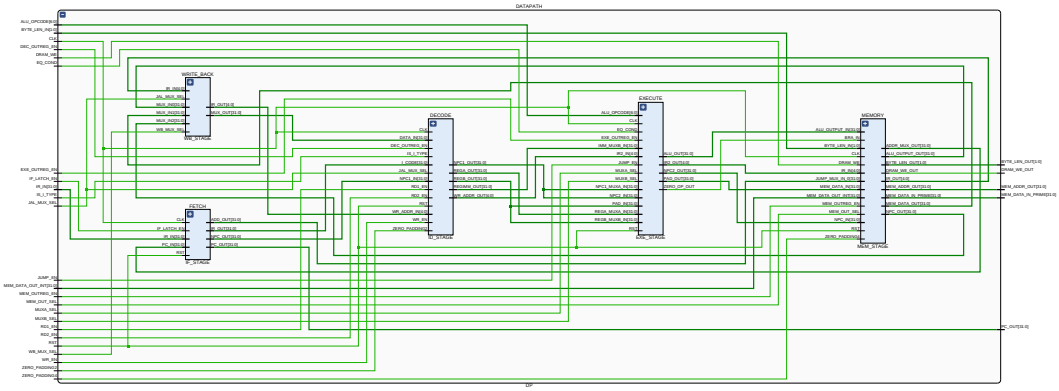

Figure 2.12: High-level view of the `DP`.

where `MODULE` has to be replaced with the name of the entity in question. Additional console logs and relevant details are provided by means of assertions directly inside the GUI of the above software directly during runtime.

The only high-level block that didn't need to be tested alone was the Datapath, as its checks were directly incorporated inside *TB_DLX* after making sure that the Control Unit was 100% functional. On that note, some relevant assembly instructions were converted into valid hexadecimal encodings by means of the provided shell script to be subsequently tested inside the aforementioned testbench.

In order to perform a specific test suite, the user is required to copy the contents of "./test_bench/memories/assembler/TEST_NAME/TEST_NAME_dump.txt" (where 'TEST_NAME' is a self-explanatory descriptor of the requested firmware) and paste them inside "./memories/romem/hex.txt" (possibly leaving `RWMEM_DEPTH=128` lines in total), before running the usual "sim_DLX.do" script. The corresponding human-readable mnemonics are located inside the `.asm` file found within the same directory as the source dump, recognized by the same test number. On the other hand, the final content of the Data Memory can be gathered from "./test_bench/memories/rwmem/hex.txt" after any successful simulation.

2.5.1 Known issues

A bug inside the Memory stage was found during the final testing phase for some Load/Store instructions, and could not be fixed in time. The interface with the external Data Memory is performed in an

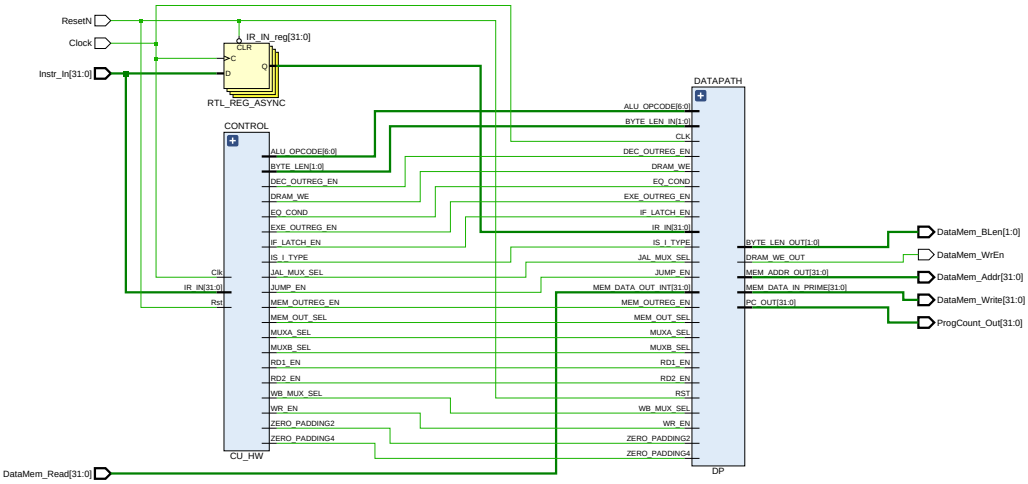

Figure 2.13: Top-level view of the `DLX` as a whole.

erroneously inverse way, as the pipeline registers are located *before* the combinational paths included within the same instead of synchronizing everything at the *end* of it.

This causes the data to be available one clock cycle too late with respect the bits coming from the Control Unit—as such, tests `load_store_basic_test` and `load_store_test2` cannot complete a successful run.

The best way to solve this issue would be to move the `sign_ext_alt` block inside the Write Back stage, handling its control signal accordingly. This would probably even improve the regularity of the structure in question, as the subsequent `gen_mux21` could be compacted into the final `gen_mux41`, which in turn would actually be used to its full capacity.

CHAPTER 3

Results

The whole microprocessor was synthesized inside "Synopsys Design Compiler" thanks to the "./synthesis/DLX_synth.scr" dedicated script (together with the provided `.synopsys_dc.setup` configuration file). There, an unconstrained cycle is executed first, then synchronous, combinational and low-power constraints are applied in order to perform a subsequent optimization process.

The specific values that are required to perform this step were found by hand, after a streak of *trial and error* synthesis runs, and the ensuing outcomes were automatically generated in "./synthesis/report/". The output netlists resulting from this process were finally used as input for the physical design step, that was ultimately carried out inside "Cadence Innovus Implementation System".

Note that Memories were excluded from the above operations, as indicated by the guidelines of this project.

3.1 Area

The following results were extracted after the optimization phase of the top-level entity:

```
****************************************
Report : area
Design : DLX
Version: F-2011.09-SP3
Date   : Wed Jul 13 19:02:12 2022
****************************************

Library(s) Used:

    NangateOpenCellLibrary (File: /home/mariagrazia.graziano/do/libnangate/NangateOpenCellLibrary_typical_ecsm.db)

Number of ports:                          165
Number of nets:                           224
Number of cells:                           34
Number of combinational cells:              0
Number of sequential cells:                32
Number of macros:                           0
Number of buf/inv:                          0
Number of references:                       3

Combinational area:       9448.320024
Noncombinational area:    8487.262273
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:          17935.582298
Total area:                 undefined
1
```

3.2 Timing

After applying the following clock period:

```
Information: Updating graph... (UID-83)
Warning: Design 'DLX' contains 1 high-fanout nets. A fanout number of 1000 will be used for delay calculations involving these nets. (TIM-134)

****************************************
Report : clocks
Design : DLX
Version: F-2011.09-SP3
Date   : Wed Jul 13 19:02:12 2022
****************************************

Attributes:
    d - dont_touch_network
    f - fix_hold
    p - propagated_clock
    G - generated_clock
    g - lib_generated_clock

Clock          Period   Waveform            Attrs     Sources
--------------------------------------------------------------------------------
Clock            3.00   {0 1.5}                       {Clock}
--------------------------------------------------------------------------------
1
```

the most critical path was found to be the one going from an internal flip-flop of the `CU_HW` (more specifically, bit number 12 of the control word dedicated to managing the Execute stage) till the flip-flop that manages the Zero flag inside the `cpsr`:

```
Information: Updating design information... (UID-85)
Warning: Design 'DLX' contains 1 high-fanout nets. A fanout number of 1000 will be used for delay calculations involving these nets. (TIM-134)

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : DLX
Version: F-2011.09-SP3
Date   : Wed Jul 13 19:02:12 2022
****************************************

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

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

  Startpoint: CONTROL/cw3_reg[12]
              (rising edge-triggered flip-flop clocked by Clock)
  Endpoint: DATAPATH/EXECUTE/ALU_FLAGS/Z_reg
            (rising edge-triggered flip-flop clocked by Clock)
  Path Group: Clock
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  DLX                5K_hvratio_1_4        NangateOpenCellLibrary

  Point                                                   Incr       Path
  --------------------------------------------------------------------------
  clock Clock (rise edge)                                 0.00       0.00
  clock network delay (ideal)                             0.00       0.00
  CONTROL/cw3_reg[12]/CK (DFFR_X1)                        0.00 #     0.00 r
  CONTROL/cw3_reg[12]/QN (DFFR_X1)                        0.07       0.07 r
  CONTROL/U108/ZN (INV_X1)                                0.03       0.10 f
  CONTROL/MUXB_SEL (CU_HW_MICRO_SIZE154_FUNC_SIZE11_OPCODE_SIZE6_IR_SIZE32_CW_SIZE20)
                                                          0.00       0.10 f
  DATAPATH/MUXB_SEL (DP_N_BITS_DATA32_N_BYTES_INST4_RF_ADDR5_N_BITS_JUMP26_N_BITS_IMM16)
                                                          0.00       0.10 f
  DATAPATH/EXECUTE/MUXB_SEL (EXE_STAGE_N_BITS_DATA32_RF_ADDR5)
                                                          0.00       0.10 f
```

```
DATAPATH/EXECUTE/MUXB/sel (gen_mux21_N32_3)             0.00       0.10 f
DATAPATH/EXECUTE/MUXB/U12/Z (BUF_X1)                    0.05       0.15 f
DATAPATH/EXECUTE/MUXB/U63/ZN (AOI22_X1)                 0.06       0.21 r
DATAPATH/EXECUTE/MUXB/U35/ZN (INV_X1)                   0.03       0.24 f
DATAPATH/EXECUTE/MUXB/m[11] (gen_mux21_N32_3)           0.00       0.24 f
DATAPATH/EXECUTE/ALRITH_LOG_U/DATA2[11] (ALU_N32)       0.00       0.24 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/B[11] (ALU_N32_DW02_mult_0)
                                                        0.00       0.24 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/U80/ZN (INV_X1)       0.04       0.29 r
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/U463/ZN (INV_X1)      0.03       0.32 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/U5/ZN (INV_X1)        0.10       0.42 r
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/U207/ZN (NOR2_X1)     0.06       0.49 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/U75/ZN (XNOR2_X1)     0.07       0.55 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/U73/ZN (XNOR2_X1)     0.06       0.62 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/S2_4_10/S (FA_X1)     0.14       0.75 r
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/S2_5_9/S (FA_X1)      0.12       0.87 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/S2_6_8/S (FA_X1)      0.14       1.01 r
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/S2_7_7/S (FA_X1)      0.12       1.13 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/S2_8_6/S (FA_X1)      0.14       1.27 r
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/S2_9_5/S (FA_X1)      0.12       1.38 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/S2_10_4/S (FA_X1)     0.14       1.52 r
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/S2_11_3/S (FA_X1)     0.12       1.64 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/S2_12_2/S (FA_X1)     0.14       1.78 r
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/S2_13_1/S (FA_X1)     0.12       1.89 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/S1_14_0/CO (FA_X1)
                                                        0.10       1.99 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/S4_0/S (FA_X1)        0.14       2.13 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/U26/ZN (NAND2_X1)     0.04       2.17 r
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/U23/ZN (NAND3_X1)     0.04       2.21 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/FS_1/B[14] (ALU_N32_DW01_add_0)
                                                        0.00       2.21 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/FS_1/U52/ZN (INV_X1)
                                                        0.04       2.25 r
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/FS_1/U105/ZN (NOR2_X1)
                                                        0.03       2.28 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/FS_1/U130/ZN (OR2_X1)
                                                        0.06       2.34 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/FS_1/U217/ZN (NAND2_X1)
                                                        0.03       2.37 r
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/FS_1/U63/ZN (NAND4_X1)
                                                        0.05       2.42 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/FS_1/U24/ZN (NAND2_X1)
                                                        0.04       2.46 r
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/FS_1/U61/ZN (NAND3_X1)
                                                        0.04       2.50 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/FS_1/U60/ZN (AND2_X1)
                                                        0.05       2.55 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/FS_1/U97/ZN (NAND2_X1)
                                                        0.03       2.58 r
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/FS_1/U166/ZN (NAND2_X1)
                                                        0.03       2.61 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/FS_1/U165/ZN (XNOR2_X1)
                                                        0.06       2.67 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/FS_1/SUM[23] (ALU_N32_DW01_add_0)
                                                        0.00       2.67 f
DATAPATH/EXECUTE/ALRITH_LOG_U/r79/PRODUCT[25] (ALU_N32_DW02_mult_0)
                                                        0.00       2.67 f
DATAPATH/EXECUTE/ALRITH_LOG_U/U570/ZN (AOI22_X1)        0.06       2.72 r
DATAPATH/EXECUTE/ALRITH_LOG_U/U143/ZN (AND4_X1)         0.06       2.78 r
DATAPATH/EXECUTE/ALRITH_LOG_U/U208/ZN (AND2_X1)         0.04       2.83 r
DATAPATH/EXECUTE/ALRITH_LOG_U/U141/ZN (NAND4_X1)        0.04       2.87 f
DATAPATH/EXECUTE/ALRITH_LOG_U/U146/ZN (NOR2_X1)         0.04       2.91 r
DATAPATH/EXECUTE/ALRITH_LOG_U/ZERO (ALU_N32)            0.00       2.91 r
DATAPATH/EXECUTE/ALU_FLAGS/FL2 (cpsr)                   0.00       2.91 r
DATAPATH/EXECUTE/ALU_FLAGS/U4/Z (MUX2_X1)               0.05       2.96 r
DATAPATH/EXECUTE/ALU_FLAGS/Z_reg/D (DFFR_X2)            0.01       2.97 r
data arrival time                                                  2.97

clock Clock (rise edge)                                 3.00       3.00
clock network delay (ideal)                             0.00       3.00
DATAPATH/EXECUTE/ALU_FLAGS/Z_reg/CK (DFFR_X2)           0.00       3.00 r
library setup time                                     -0.03       2.97
data required time                                                 2.97
---------------------------------------------------------------------------
```

```
  data required time                                                 2.97
  data arrival time                                                 -2.97
  ---------------------------------------------------------------------------
  slack (MET)                                                        0.00


1
```

3.3 Power

After the low-power synthesis step, the imposed constraint was finally conformed to, and the main contributions to this physical quantity were found to be as follows:

```
Information: Propagating switching activity (low effort zero delay simulation). (PWR-6)
Warning: Design has unannotated primary inputs. (PWR-414)
Warning: Design has unannotated sequential cell outputs. (PWR-415)

****************************************
Report : power
        -analysis_effort low
Design : DLX
Version: F-2011.09-SP3
Date   : Wed Jul 13 19:02:14 2022
****************************************


Library(s) Used:

    NangateOpenCellLibrary (File: /home/mariagrazia.graziano/do/libnangate/NangateOpenCellLibrary_typical_ecsm.db)


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

Design        Wire Load Model            Library
------------------------------------------------
DLX                   5K_hvratio_1_4    NangateOpenCellLibrary


Global Operating Voltage = 1.1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  =   3.9233 mW   (93%)
  Net Switching Power  = 290.1907 uW    (7%)
                         ---------
Total Dynamic Power    =   4.2135 mW  (100%)

Cell Leakage Power     = 352.6860 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000           0.0000  (   0.00%)
memory             0.0000           0.0000            0.0000           0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000           0.0000  (   0.00%)
clock_network      0.0000           0.0000            0.0000           0.0000  (   0.00%)
register       3.8095e+03          26.6191        1.2780e+05        3.9639e+03  (  86.81%)
sequential         0.0000           0.0000            0.0000           0.0000  (   0.00%)
combinational    113.8139         263.5720        2.2489e+05         602.2712  (  13.19%)
--------------------------------------------------------------------------------------------------
Total          3.9233e+03 uW       290.1911 uW     3.5269e+05 nW     4.5662e+03 uW
1
```

3.4 Physical Design

The post-synthesis Verilog netlist was used as input for the entire physical design process, which would in turn generate, among many other valuable resources, the post place and route one. Please note that all files with the "DLX_PRO" title are just the starting point of this step (deriving from the previous one), while all its actual outputs were saved with the "DLX" tag only.

All the produced dumps, archives and reports are located inside the "./physical_design/" directory, including all its subfolders for timing information. Many intrinsic characteristics of the final circuit can be found in there, including parasitics, geometry and connectivity characterization.

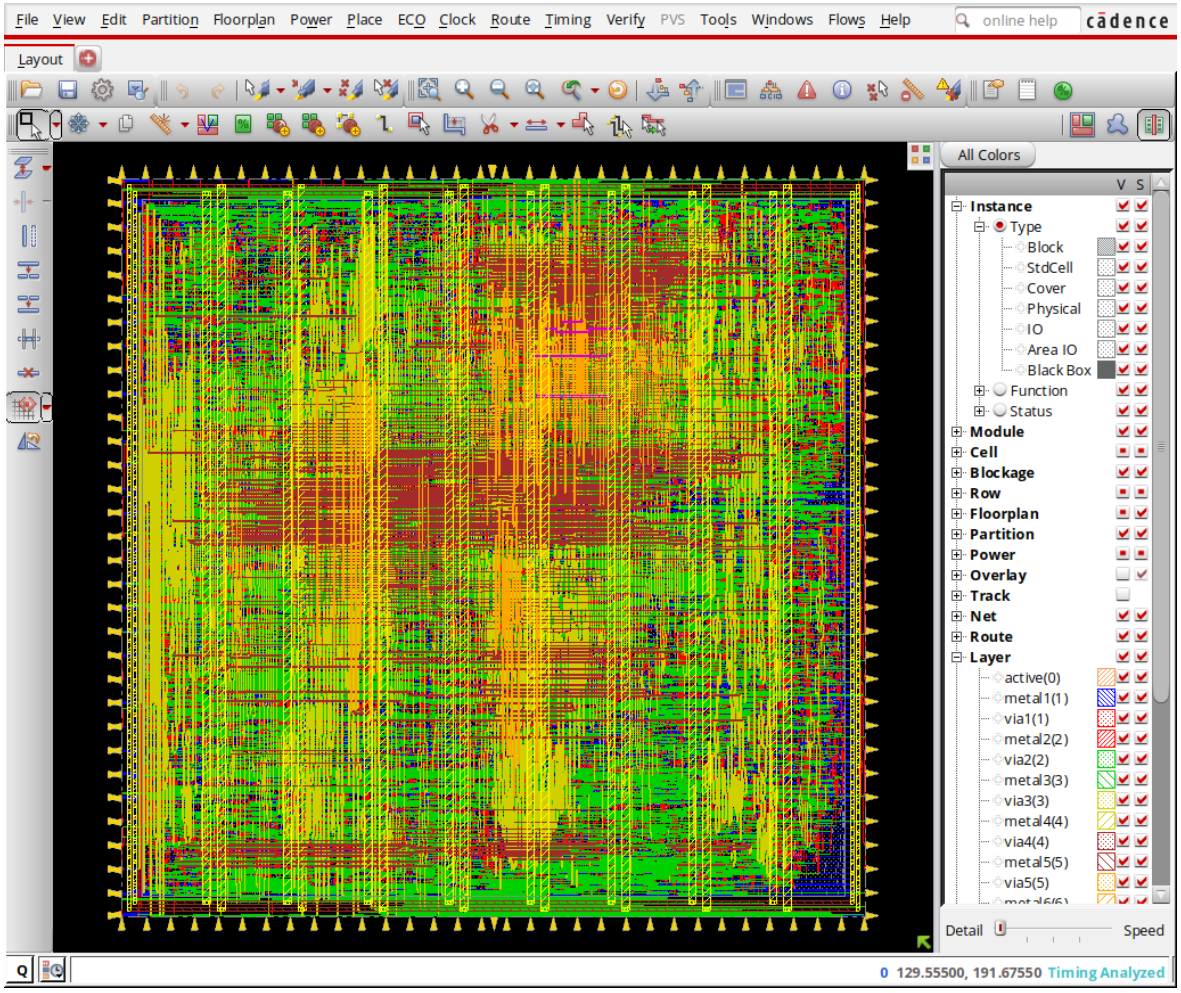

Figure 3.1: Obtained image dump of the circuit layout.

CHAPTER 4

Conclusion

This set of deliverables represents the outcome of more than half a year worth of team meetings, discussions, group work and overlaps with many external obligations, being them other courses/exams or just personal matters.

The required progress was student-driven in its entirety, and represented a valuable lesson in terms of both hard and soft-skills. Of course, the implementation followed a very deep reasoning phase in which all the theoretical knowledge previously available to the authors was exploited and expanded on, in order to make a considerable amount of arbitrary design choices. On the other hand, team effort was crucial during the whole duration of this project—code sharing, calendar planning, hand picking features and discussing many different solutions to the same problem, *all* represented a series of obstacles that had to be overcome in order to eventually shape an acceptable product.

Finally, the fact of trying to make everything as easy to use as possible required an increased effort into extensively commenting the code itself and writing it in a clear and concise way. All this would in turn require the authors to unequivocally grasp every single detail of their own implementation, therefore mastering the whole architecture and eventually being able to present it in first person.

Note: all the schematics included in this report were generated after a synthesis process that took place in a local instance of "Xilinx Vivado Design Suite". They do **not** represent the actual end result of the steps performed inside the provided environment with the correct custom library, and just serve as a reference for the explanations included in Implementation.