Politecnico di Torino

Microelectronic Systems

DLX Microprocessor: Design & Development

Final Project Report

Master degree in Electronics Engineering

Referents: Prof. Mariagrazia Graziano, Giovanna Turvani

Authors: Group 7

Luigi Galasso, Rafael Campagnoli

October 16, 2019

Contents

1 Introduction **1**
- 1.1 Project Description . . . 1
- 1.2 Objectives . . . 1
- 1.3 Specification . . . 1
  - 1.3.1 DLX Basic . . . 1
  - 1.3.2 Instruction Set . . . 2
- 1.4 Functionality . . . 3
  - 1.4.1 Data Path . . . 3
  - 1.4.2 Hardwired Control Unit . . . 5
  - 1.4.3 Memories . . . 6

2 Functional Schema **7**
- 2.1 Data Path . . . 8
  - 2.1.1 Fetch Unit . . . 8
  - 2.1.2 Decode Unit . . . 8
  - 2.1.3 Execution Unit . . . 10
  - 2.1.4 Memory Unit . . . 11
  - 2.1.5 Writeback Unit . . . 12
- 2.2 Control Unit . . . 13

3 Implementation **14**
- 3.1 Synthesis . . . 14
  - 3.1.1 Description . . . 14
  - 3.1.2 Results . . . 14
- 3.2 Physical Design . . . 15

4 Discussion and Conclusions **16**

References **17**

A Synthesis Script **18**

List of Figures

1.1 Different types of instructions. 2

2.1 Schematic of the DLX. 7
2.2 Schematic of the Fetch Unit. 8
2.3 Schematic of the Decode Unit. 9
2.4 Schematic of the Execution Unit. 10
2.5 Schematic of the ALU 11
2.6 Schematic of the Memory Unit. 12
2.7 Schematic of the Write-Back Unit. 12
2.8 Schematic of the Control Unit. 13

3.1 Screen Dump of the physical implementation of the DLX 15

List of Tables

3.1 Critical paths achieved in the final synthesis optimization. 14
3.2 Power and Area results of the DLX synthesis. 15
3.3 Critical Path, Gate Count and Area obtained in the physical design. 15

CHAPTER 1

Introduction

1.1 Project Description

The purpose of this report resides in the portrayal of the project of a basic pipelined DLX processor. The project was carried out describing the processor using the VHDL description language. After the description and simulation, a refinement from RTL (Register Transfer Level) down to synthesis and physical design was performed. In this sense, this report elucidates the description of the essential components needed to assemble the DLX processor and the main choices made to accomplish the objectives.

1.2 Objectives

The Objectives of this project are, first the VDHL description of a basic pipelined DLX processor exploiting a set of simple components and a finishing phase consisting in synthesis and physical implementation that were constrained in order to optimize the performance in terms of timing. The reports of timing, power and area obtained should also be discussed.

1.3 Specification

1.3.1 DLX Basic

The specification used for the basic DLX configuration used is:

- **Pipeline**: The architecture is organized with a five staged pipeline data path and an hardwired control unit.
- **Instruction Set**: A set of 27 basic instructions to be implemented.
- **Data Path**: A VDHL description at RT level of a data path capable of execute all of the instructions in the instruction set defined.
- **Control Unit**: A VDHL description at RT level of a control unit capable of organize and manage the correct behavior.
- **Synthesis**: A synthesis of both the control unit and the data path with the generation of the respective performance reports and a final optimization for frequency.
- **Physical Design**: The placing and routing of the synthesized design.

1.3.2 Instruction Set

The instructions added to the set are the basic configuration of instructions, which are: add, addi, and, andi, beqz, bnez, j, jal, lw, nop, or, ori, sge, sgei, sle, slei, sll, slli, sne, snei, srl, srli, sub, subi, sw, xor and xori.
The instruction set can be divided in 3 different types of instructions that can be implemented, the R-type, the I-type and the J-type instructions. The type of instruction can be identified by the group of first 6 most significant bits of the instruction. This group of bits is called the **OPCODE**.

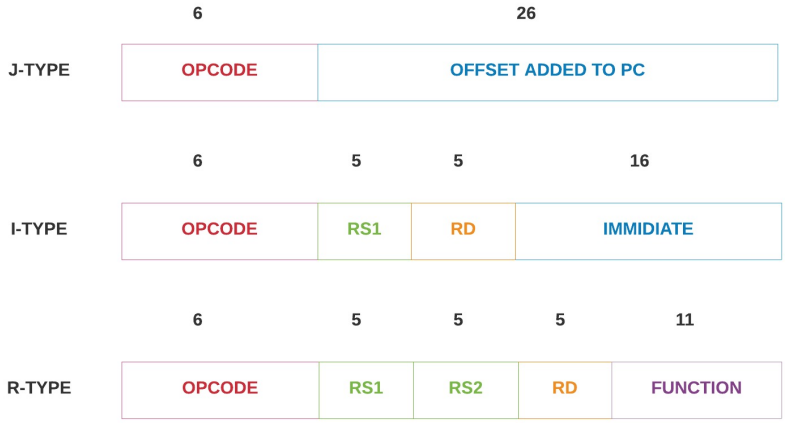

Figure 1.1: Different types of instructions.

Using as reference the Figure 1.1 in the following the different instructions types are analyzed.

R-type

R-type instructions are characterized by the fact that the correspondent ALU operations are between two registers with the final destination in another third register. The R-type instructions are then organized by an OPCODE equals to six zeros, 15 bits to address the 3 registers to be used and 11 bit assigned to the **FUNC** that is a set of bits used by the control unit to identify the mathematical operation to be executed, as the null OPCODE only says that a R-type instruction must be executed without specifying it.

I-type

The I-type instructions are, instead related to ALU operations between a register and an immediate number, with another register used as final destination. Their OPCODE gives information about which arithmetic operation must be performed, followed by 10 bits used for the addressing of the registers and a 16-bit set which represents the immediate number to be used in the calculation. Load and Store instructions are also I-type instructions, with the same configuration of the instruction word used for the ALU ones. However, the immediate is used as an offset to be added to the first register in order to calculate the address to be used in the memory. The branch kind of instructions are also identified as I-types. The registers addressed are used for the comparison requested by the instruction, while the immediate is used as the address to which the program should branch to.

J-type

J-type instructions are the instructions related to the unconditional jumps. In our project, the J-type instructions are the J (Jump) and the JAL(Jump And Link) instructions. The J-type instructions are organized as a 6 bits OPCODE and a 26 bits address(offset) to which the program should jump.

1.4 Functionality

In this section is described the main functionality of the complete DLX, while in the Chapter 2 all the schemes are analyzed. To describe the functionality of the DLX, it is first necessary to describe it's high-level functional blocks. As it is shown in figure in the following chapter, the DLX's blocks are:

- **Data Path**: the main functional block for identifying the instruction and executing it.
- **Control Unit**: the "brain" of the DLX, which command how the data path should work due to one specific instruction and how to manage the pipe pipeline functionality between instructions.
- **Instruction Memory**: Basically a LUT (Look Up Table) with the all the instructions of the code programmed inside, which are addressed by the PC (Program Counter).
- **Data Memory**: An external memory for storing and loading data necessary for executing the program.

1.4.1 Data Path

The data path is the main core of the system and it is divided in five pipelined stages. The five stages are fetch, decode, execute, memory and write-back stage. The reason why it is pipelined is to increase the throughput of instructions. This goal is accomplished by inserting registers between the stages so the latency of the whole data path increases. By doing that, the clock frequency can be increased, limited only by the stage with the highest latency plus the set-up time of the registers.
The functionality of the stages are described below.

Fetch Unit

The purpose of the fetch unit is to "insert" in the datapath an instruction from the instruction memory. The instruction memory is basically a LUT (Look-Up Table), which contains the programmed code instruction by instruction aligned in its addressable space. The instruction memory is addressed via the PC (Program Counter). Every clock-cycle, the PC is incremented by one and fetches the next instruction, which is then sent to the decode stage.
In the case that there is a jump or branch instruction, the PC is updated to the address of the of the instruction to which it is needed to jump to.

Decode Unit

The purpose of the decode unit is to take the instruction given by the previous stage and sent it to the control unit, which then defines it's control word to control the data path. The decode stage contains, mainly, the RF(Register File), a sign-extension block and a branch unit.
The RF is a small memory with 32 registers used for the calculations executed by the processor. It contains two reading ports and one write port. The reading ports are selected in the decode stage accordingly to the decoded instruction and then the addressed registers are sent to the next stage. The write port is used by the write-back unit to write in RF operations results.

The sign-extension block is used in case a I-type or a J-type instruction is decoded. It takes the immediate (26-bit in J-type case and 16-bit in I-type case) value given by the instruction, selects its most significant bit and extend it to 32 bits.
The Branch unit evaluates if there is a jump to be taken or not, depending on the instruction, and then sends a branch-taken signal to the fetch unit with a correspondent address. If there is a J or a JAL instruction, the branch is taken unconditionally, with the value contained in the last 26 bits of the instruction used as the new address. If there is a BEQZ or a BNEZ instructions, the given register is compared if it is or not equal zero and the branch-taken signal is sent accordingly if the condition asked by the instruction is met.

Execute Unit

The execute unit is the main block of the processor. It is where the instruction is executed using the right data. The execution block is made of an ALU (Arithmetic Logic Unit) and of multiplexers that select the inputs for the ALU. In our project, the ALU is capable of performing basic bitwise logic operations (OR,AND,XOR,SLL,SRL), sum and subtraction with a P4ADDER and comparison operations with a dedicated comparator (SEQ,SNE,SGE,SLE).
The multiplexers select three different configurations for the two inputs of the ALU. One is that both are the outputs of the RF used for R-type instructions; the second is for I-type instructions, in which one input is the first output of the RF and the other is the immediate sign extended number coming from the instruction and the last one is for JAL instruction for wich one input is the NPC (Next Program Counter) previously calculated and the other is the number 1 used to calculate the correct PC that has to be properly saved in R31 register.
The P4ADDER was inserted in the ALU according to the configuration written in a previous laboratory. It is used to allow a small timing optimization. Its functionality resides in two blocks: a carry generation block, i.e. SPARSE TREE configuration, and a carry select adder. The Sparse Tree calculates the proper carries in an optimized way, i.e. calculating each of the 8 carries concurrently, while the carry select adder, divided in 8 blocks, in its blocks performs a sum of a group of 4 bits each blocks with and without the carry-in. In eache blcks, both the sums are then multiplexed and selected according to the signal coming from the carry generation block. By using this architecture, the sum can be performed much faster than other architectures, without the delay of the first carry up until the last bit.
The Comparator, instead, used for set-instructions compares the content of the first ALU-input data with the second and set its output to 1 or 0 according to the executing configuration required.

Memory Unit

The memory unit is the data path's interface with the external memory. Is basically used in the SW and LW instructions, in which the data path provide the address for the memory and interact with it. In the case of SW, it sends data from a register to the memory at the given address. In the case of LW, it reads a word from the memory and writes it to the RF in the write-back stage.

Write Back Unit

The write-back unit's purpose is to update the register file after a instruction is executed. It consists basically of a multiplexer that selects the output of the memory and the output of the ALU, forwarding it to the input of the RF back in the decode unit. After an instruction is executed and a value of a register is altered, the multiplexer selects the output of the ALU. If the instruction is a LW, it selects the loaded data from the memory. The address of the destination register is known in the decode stage, and it's forwarded until the write-back stage for correct functioning.

It is important to note that the delay between the execute unit and the write-back unit may cause data hazards, so a correct number of "bubbles"(a NOP instruction) must be inserted in the pipeline if there is a data dependency between two instructions.

1.4.2 Hardwired Control Unit

The control unit is the "brain" of the DLX. It's the unit that controls all of the signals that control the data path, and also how to pipeline it. The configuration of the architecture we selected to implement was the hardwired one, which is basically a LUT with the OPCODE and FUNCTION of the instruction as inputs and a control word as an output. To control the datapath in a pipelined way, the control word is delayed for 4 stages, with the signals respectively connected to each stage on where they are used.

Control Word

In the instruction decode stage, the instruction contained in the instruction register is sent to the hardwired control unit, which then translates it in a control word. The control word used is 17 bits long and contains the 17 bits needed for the correct functionality of the datapath. The control word can be described, from MSB to LSB, as:

- **EN1**: Enable signal for the registers in the fetch stage
- **EN2**: Enable signal for the registers in the decode stage
- **SignSelect**: Signal used for the sing-extension
- **RD1**: Enable signal for the read port 1 of the RF
- **RD2**: Enable signal for the read port 2 of the RF
- **JMP**: Unconditional Jump signal
- **BranchCondSel**: Signal to indicate which condition the branch unit must evaluate
- **BRANCHenable**: Enable signal for the output of the branch unit
- **RegDestination**: Used to select the type of destination address to select
- **EN3**: Enable signal for the registers in the execute stage
- **MUX1SEL**: selection bit used for the selection of the ALU inputs
- **MUX2SEL**: selection bit used for the selection of the ALU inputs
- **ALUCODE**: 4-bit code to select the operation that must be performed by the ALU
- **EN4**: Enable signal for the registers in the memory stage
- **MemoryEnable**: Enable signal for the operation of the memory
- **ReadNotWrite**: Signal to the define which operation should be executed by the memory
- **SelWB**: Selection signal for the multiplexer that selects the output of the ALU or the Output of the memory
- **WR**: Enable signal to write-back into the register file

1.4.3 Memories

The memories present in the project are the instruction memory and the data memory.

Instruction Memory

The instruction memory is a ROM (Read Only Memory). Its purpose is to store the code of the program in the form of bit words, which are later decoded in by the control unit in the decode stage. For this project we used the given VHDL code for the memory, that basically takes an assembly program (test.asm.asm) previously converted to hexadecimal and alignes it in a LUT. The memory is then addressed by an index of 1, which means that from one instruction to the next, the address is should be incremented by 1. The datapath block provides the PC to address the memory and fetches the instruction to the instruction register. The instruction memory is controlled by an enable signal coming from the control unit.

Data Memory

The data memory is a external memory used for load and store. The one we described in VHDL is very similar to the register file, for simulation purposes. It's functionality can be described as a memory where one can read or write some data at a given address. It is controlled by an enable signal and a read-not-write signal that selects the type of operation to be done. Both signals come from the control unit.

CHAPTER 2

Functional Schema

The purpose of this chapter is to describe the schematics of the main blocks of the system and to explain its main way of working and interconnections. In figure 2.1 the functional schema of the all DLX is illustrated. Two main blocks represent the core of the microprocessor. In particular there is a Data Path, colored black, consisting of 5 stages (FETCH, DECODE, EXECUTION, MEMORY and WRITE BACK) and a Control Unit interconnected as shown in the illustration. The Control Unit is connected with the FETCH UNIT output, i.e. the instruction to be decoded, and the other four stages. Each Data Path stage is then connected with the next one, except for the DECODE UNIT which is connected to the FETCH STAGE to perform JUMP operations if required and the WRITE BACK UNIT which is connected to the Register File of the DECODE UNIT to perform write operations. In the schema are also present two external memory: an INSTRUCTION MEMORY and a DATA MEMORY.

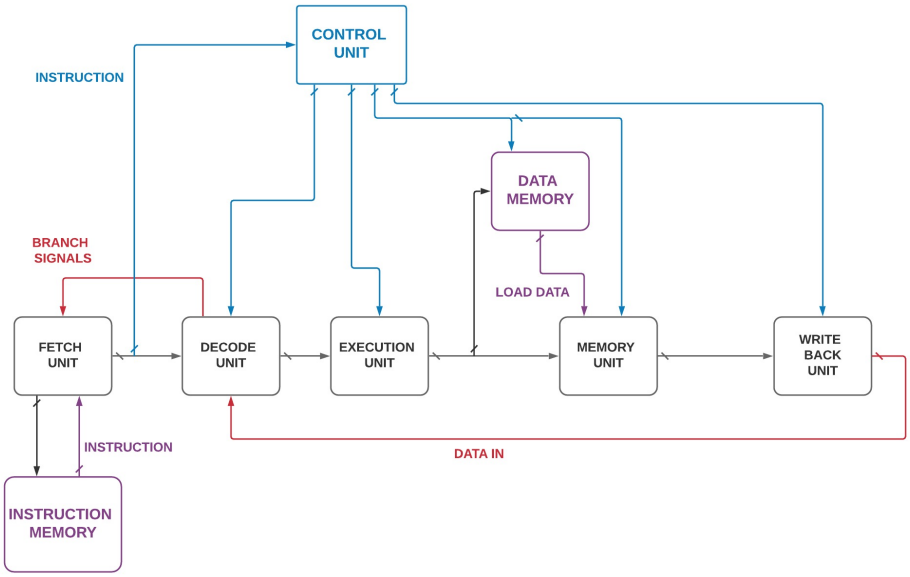

Figure 2.1: Schematic of the DLX.

2.1 Data Path

The Data Path units are composed by several different blocks. In order to simplify the discussion it is pointed out here that each stage contains some 32-bits pipeline registers. In the following, each block will be examined.

2.1.1 Fetch Unit

The fetch unit is assembled as described previously and it is illustrated in figure 2.2. The general functionality is that the program counter register addresses the memory and it's incremented by one every clock cycle. In this way the fetch unit always fetch one instruction after the other in the order in which they are stored in the Instruction Memory. The signals that pass to the next stages through registers are the instruction and the NPC.
It can be noticed from the figure the signals *BRANCH FROM DECODE*, which represents that a branch or a jump must be taken, and the *BRANCH PROGRAM COUNTER*, which represents the program address that should be fetched in the next clock cycle. The *BRANCH FROM DECODE* is the selection bit of the multiplexer whose output is connected to the program counter register and also enters in a or port with the reset signal to the reset port of the Instruction register. This second configuration is due to the need to implement a pipeline flush in the case of a branch is taken.

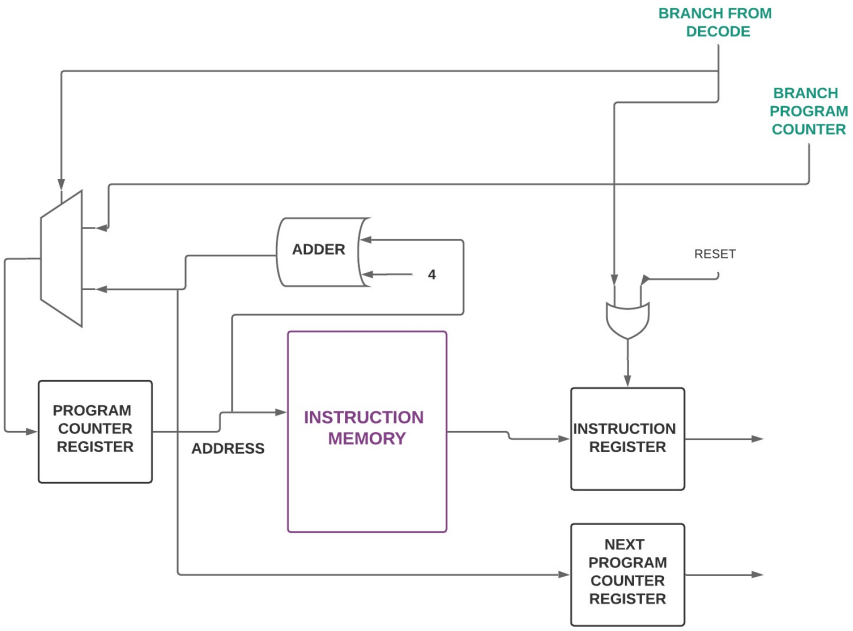

Figure 2.2: Schematic of the Fetch Unit.

2.1.2 Decode Unit

The DECODE UNIT can be observed by the Figure 2.3. It is assembled with the register file, the branch unit, some logic gates, the sign extension unit and the write-back address multiplexer.

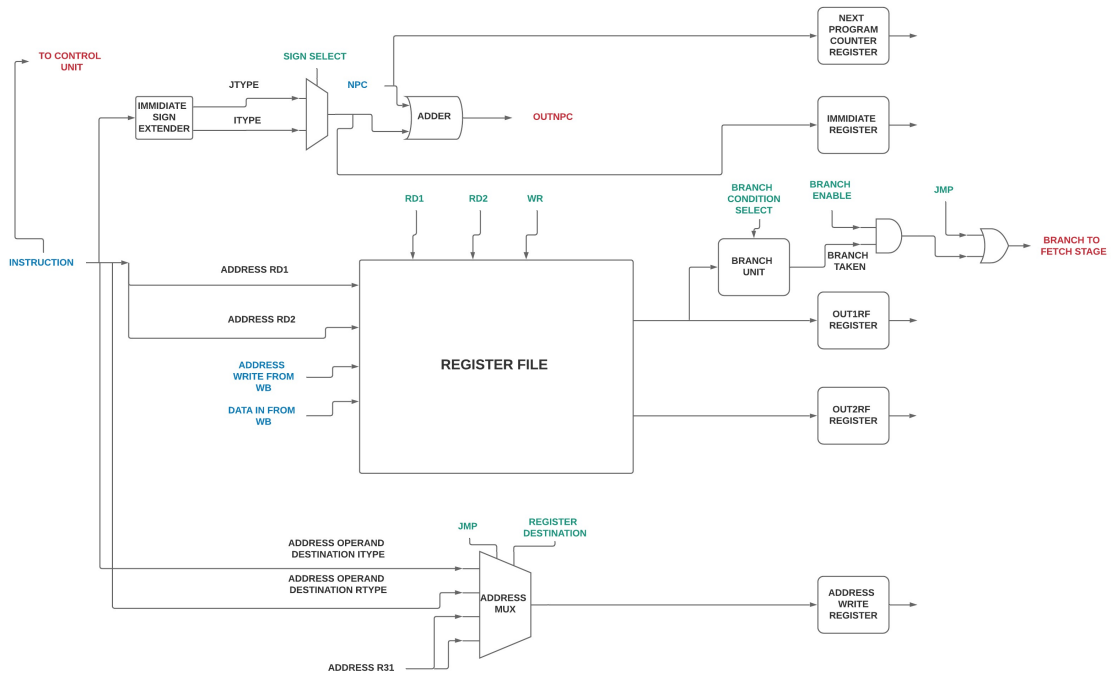

Figure 2.3: Schematic of the Decode Unit.

The register file can be read through two ports concurrently and written through one port. The read operation is asynchronous, while the write operation is only concluded after one clock cycle. It is important to notice that due to this configuration, data hazards may be observed if no bubbles are inserted. Instruction's operands addresses(R-type case) are used to address the Register File outputs. The Control signals delivered by the Control Unit to the Register File are in this stage RD1, RD2 and WR.
The Branch unit detects if the Branch condition defined by the BRANCH CONDITION SELECT delivered by the Control Unit is verified or not; if the condition is verified and the BRANCH ENABLE signal from Control Unit is high, through a logic of an AND port and an OR port, the BRANCH TO FETCH signal is sent to Fetch Unit. Otherwise, in the case of a Jump instruction, the JMP signal from Control Unit is high and the Jump operation has to be done unconditionally. In order to deliver the correct Program Counter to the Fetch Unit it is implemented an addition of the Next Program Counter from Fetch stage and the correct sign extended immediate corresponding to the Jump displacement. A SIGN SELECT signal defined by the Control Unit is used for this purpose.
As it is shown, it can be observed the presence of an address multiplexer(4 to 1 multiplexer) that is configured by a REGISTER DESTINATION signal from the Control Unit which selects if the operand is an I-type or a J-type according to this requirement a different set of bits of the Instruction is selected. Otherwise if the JMP signal is high the constant address of register 31 is selected. In other words, in both JUMP and JAL instruction cases the address of the operand destination is selected as R31, but only in JAL case the PC is then saved in RF. Five pipeline registers have been finally inserted. They store respectively: Next Program Counter, Immediate, OUT1RF (first Register File Out),OUT2RF (second Register File Out) and Address Write (Operand destination address).

2.1.3 Execution Unit

The EXECUTION UNIT can be observed in the Figure 2.4. It was assembled with the ALU(Arithmetic Logic Unit), three multiplexers and a set of pipeline registers.

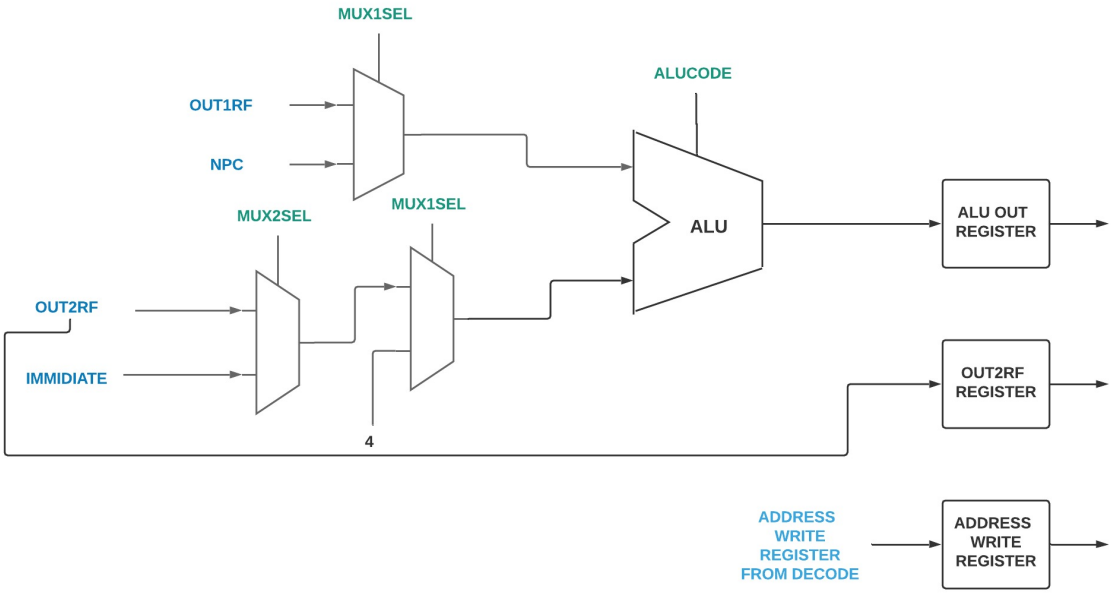

Figure 2.4: Schematic of the Execution Unit.

The two inputs of the ALU are selected by the Control Unit control signals MUX1SEL and MUX2SEL through multiplexers. In particular in R-type instruction case the the two Register File's outputs are selected, in I-type instruction case the first Register File's output and the Immediate are selected and in J-Type instruction case the NPC(Next Program Counter) and the constant 4 are selected. The Output of this stage are stored in three Pipeline registers corresponding to the ALU's output, the second Register File's output and the address of destination.
The ALU is carefully analyzed here. As it can be seen in Figure 2.5 a 10-input multiplexer is exploited to select the correct ALU output. This is determined by the 4-bits ALUCODE signal delivered by the Control Unit. It can be observed the presence of a SHIFTER, a P4ADDER and a COMPARATOR in addition to three simple logic gates(AND,OR,XOR). All the blocks are able to implement 32-bits parallel operations. The ALU CODE is also exploited to select the right configuration of the blocks through simple 1-bit parallel logic gates. In particular the P4ADDER is configured in sum or subtract mode by the CARRY IN definition, the SHIFTER mode direction is determined by the SHIFT DIRECTION and finally the COMPARATOR mode(equal,not equal, lower or equal, higher or equal) is determined by two comparison bits.

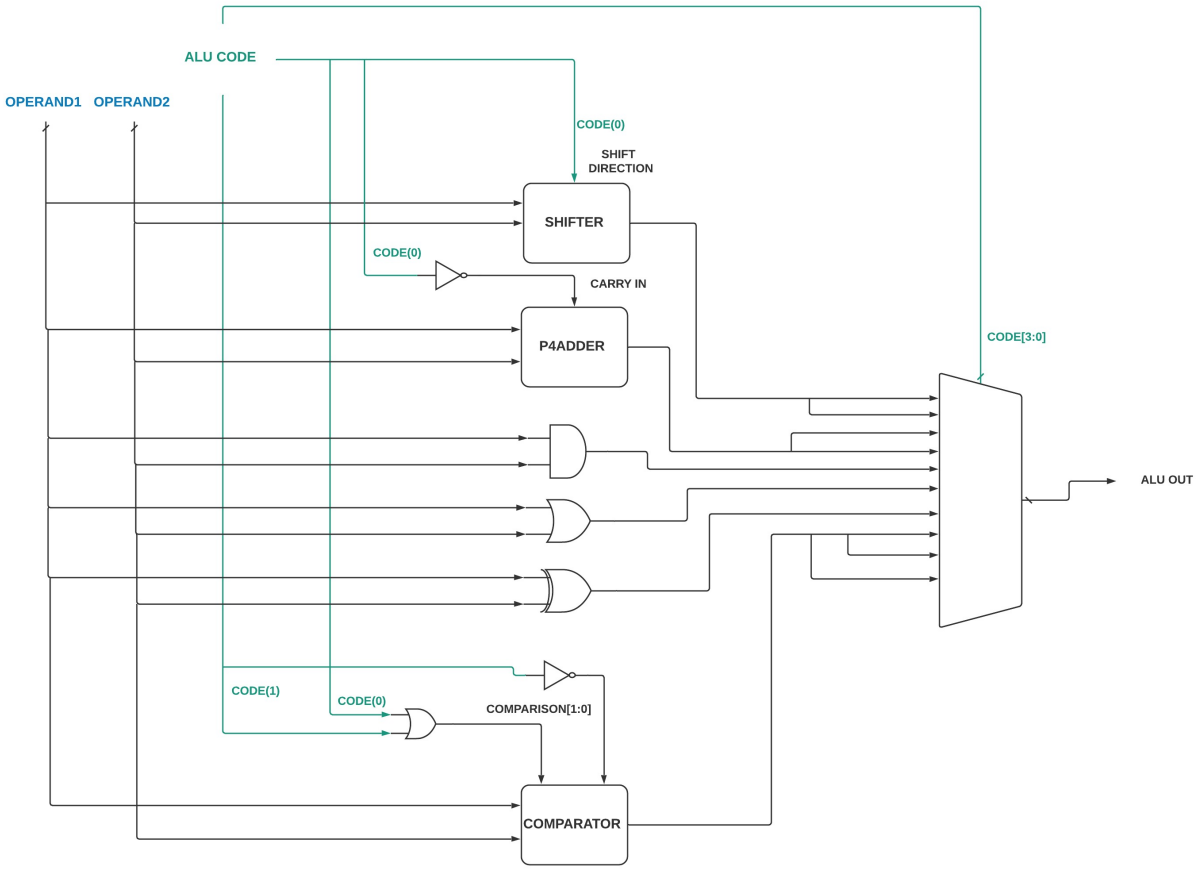

Figure 2.5: Schematic of the ALU .

2.1.4 Memory Unit

The MEMORY UNIT is composed of three pipeline registers and an external DATA MEMORY. As it can be observed by the Figure 2.6, there are three inputs. Two of them, i.e. the ALU OUT and the OUT2RF, are used respectively as data input(to be written) and address, for load/store operations of the DATA RAM which is a read and write memory. Memory operations are managed by two control signals from the Control Unit: DATA MEMORY ENABLE (which activates the memory) and READ NOT WRITE (which configures the operating mode of the RAM). The output of Memory is stored in a LMD (Dad Memory Data) register, whereas the ALU OUT register stores the ALU output and the ADDRESS WRITE register delays, of one clock cycle, the writing operation in Register File address pointed by the ADDRESS WRITE FROM EXECUTE.

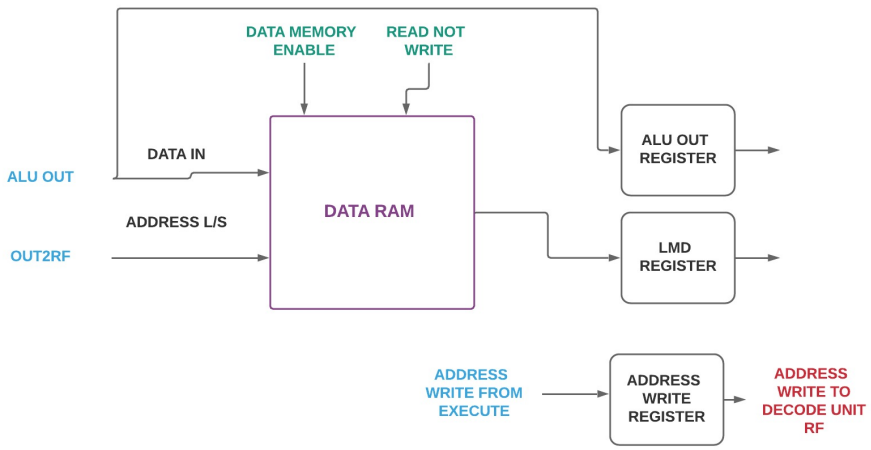

Figure 2.6: Schematic of the Memory Unit.

2.1.5 Writeback Unit

The WRITE-BACK unit is composed of a simple multiplexer 2 to 1.

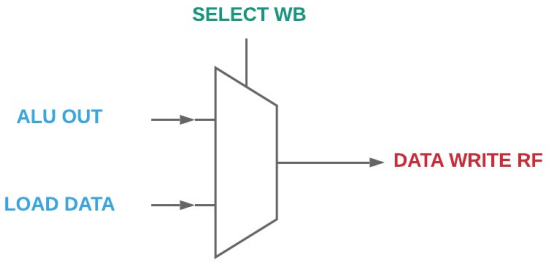

Figure 2.7: Schematic of the Write-Back Unit.

As it can be noticed by the Figure 2.7 the two inputs of the write-back multiplexer (the ALU OUT and the LOAD DATA from MEMORY STAGE) can be selected as data to be written in Register File in DECODE UNIT through a SELECT WB signal from the CONTROL UNIT.

2.2 Control Unit

The implementation chosen for the CU (CONTROL UNIT) is the HARDWIRED one. As it is illustrated in Figure 2.8, the INSTRUCTION taken as input is decoded by its two major parts, i.e. the OPCODE and the FUNCTION, which determine through a LUT(Look Up Table) a proper Control Word and the correspondent ALUCODE. Some registers are then exploited to store them and allow the pipeline to work in the correct way. The 16-bits Control Word is then delivered to the different stages, delayed by the right number of CLOCK periods, for the correspondent Control Word bits. For what concerns the 4-bits ALUCODE, it is first calculated during the Decode Phase, by the CU and then delayed by one CLOCK cycle before properly reach the ALU in the EXECUTION UNIT.

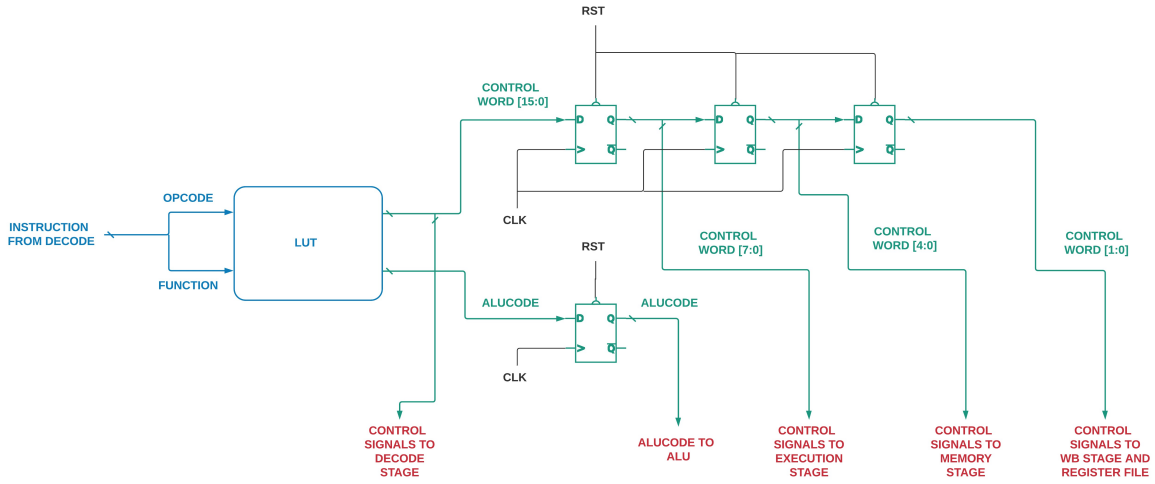

Figure 2.8: Schematic of the Control Unit.

CHAPTER 3

Implementation

After the correct assembly of the complete processor system, several simulations were performed using Modelsim to verify the correct behaviour of the described DLX. To this purpose the group created some assembly programs that are included in our benchmark that attests this verification.
After the simulation phase, a synthesis using the software Synopsys Design Vision followed by a physical design using the software Innovus were performed. The purpose of this chapter is to describe the results of the optimizations performed. The scripts used for the synthesis can be found in the Appendix A.

3.1 Synthesis

3.1.1 Description

In the Synthesis script the entire system (the microprocessor) , apart from the memories, is analyzed from bottom to top level. An elaboration and a compilation with no map effort were performed and a timing report was generated using **Synopsys Design Vision**. After that, a clock constrain (previously obtained after several synthesis) was created and the block was compiled again, this time with an high map effort. A timing report of the optimized design was finally generated.
This procedure has been carried out for the datapath, the Control Unit and for the DLX, in this order. Furthermore for the DLX entity, power and area reports were also obtained. At the end of the used script, some saving files were written, and an .sdc file was generated for the physical implementation phase.

3.1.2 Results

The results of the optimized timing reports can be observed by the Table 3.1

Data Path	Control Unit	DLX
1.14 ns	0.11 ns	1.24 ns

Table 3.1: Critical paths achieved in the final synthesis optimization.

It can be noticed that the DLX's critical path is more or less the sum of the Data Path and Control Unit critical paths, which demonstrates that the synthesis was executed correctly, even if the path may not be the same due to optimization algorithms performed by the software. As the timing reports already considers the set-up time, it can be calculated the maximum clock frequency, which should

be the inverse of the "worst-time" of the DLX. In this project, it was possible to achieve a maximum clock frequency of approximately 800 MHz. The results of the power and area reports of the DLX timing-optimized can be observed by the Table 3.2.

Internal Power	Switching Power	Leakage Power	Total Power	Total Cell Area(μm^2)
8.4080e+03 uW	609.4587 uW	3.4865e+05 nW	9.3661e+03 uW	15877.540156

Table 3.2: Power and Area results of the DLX synthesis.

3.2 Physical Design

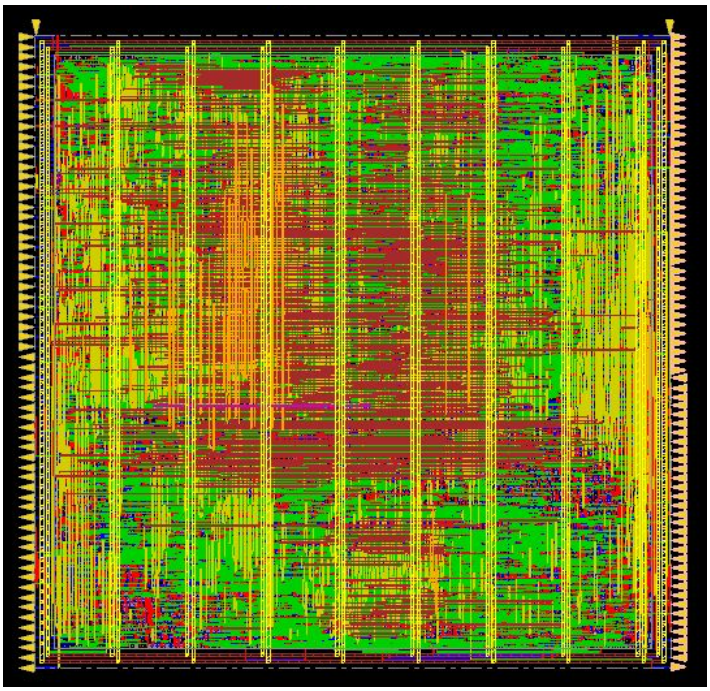

Figure 3.1: Screen Dump of the physical implementation of the DLX

For the physical design, the standard procedure given in the Lab 6 pdf was followed. The verilog file generated in the synthesis part was used for place and route using the software **Innovus**, through an optimization for routing. The result of the placing and routing can be observed in the Figure 3.1 of the screen dump generated. Some post Routing results are reported in the following Table 3.3. As it is shown in the Table, the Critical Path delay is equal in post synthesis and post routing phases; in addition, the area occupation and the gate count are reported.

	Data Path	Control Unit	DLX
Critical Path (ns)	1.14	0.11	1.24
Gate Count	17939	289	18228
Area (μm^2)	14315.6	230.6	14546.2

Table 3.3: Critical Path, Gate Count and Area obtained in the physical design.

CHAPTER 4

Discussion and Conclusions

The results obtained at the end of this project confirm that the main objective was achieved. The group was able to describe the microprocessor DLX at the RT level and refine it from a synthesis up to a physical design successfully. At the same time, a wide range of decisions in the design phase was taken resulting in various trade-offs. In the case of this project, the choices taken were aligned to the objective of keeping a simple DLX with a few timing optimizations.
One of the main problems encountered in the development of the architecture was the pipeline alignment between the Control Unit and the Data Path. In order to solve this issue, the solution implemented is to define the pipeline starting at the decode stage, due to the fact that it is in this pipeline stage that the instruction is actually decoded by the control unit. The fetch stage register's enable should always be set to 1, except in the case of a reset or a jump instruction, so that the fetch unit doesn't fetch the wrong instruction.
As a result of these few changes, the synthesis reveals that a maximum clock frequency of 800 MHz is obtainable and it can be considered satisfactory for the low complexity of the system.
For testing the correct behaviour of the system, some benchmarks have been implemented. The given benchmarks are mainly separated depending on the type of instructions tested. They are, one for the Jump and Jump and link instructions, one for the branches, one for the arithmetic operations and one for the comparison operations. All the benchmarks confirm the expected behaviour.
For future improvements, it might be thought to implement an hazard detection units, so that the data and control hazard have not be predicted by the programmer. Another possibility would be the implementation of multiplication and division units inside the ALU, so that the DLX could be more versatile with a wider range of instructions.

References

[Gra13] M. Graziano. Microelectronic systems lecture notes, 2013.

[Gra19] M. Graziano. Laboratory notes - physical design: standard-cells based layout, 2019.

[GST17] M. Graziano, G. Santoro, and G. Turvani. Desing and development of dlx microprocessors, 2017.

[HP12] J. L. Hennessy and David A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 2012.

APPENDIX A

Synthesis Script

```
#
analyze -library WORK -format vhdl {000-globals.vhd}
#Fetch
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.a-reg.vhd}
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.b-adder.vhd}
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.c-
    InstructionRegister.vhd}
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.d-mux21.vhd}


#Decode
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.e-BranchUnit.
    vhd}
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.f-mux41.vhd}
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.g-registerfile.
    vhd  }
#Execute
#P4ADDER
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.h-ALU.core/a.b.
    h.a-fa.vhd}
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.h-ALU.core/a.b.
    h.b-rca.vhd  }
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.h-ALU.core/a.b.
    h.c-CSb.vhd  }
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.h-ALU.core/a.b.
    h.d-CSA.vhd }
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.h-ALU.core/a.b.
    h.e-Pg.vhd  }
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.h-ALU.core/a.b.
    h.f-G.vhd  }
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.h-ALU.core/a.b.
    h.g-PGnet.vhd   }
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.h-ALU.core/a.b.
    h.h-sparsetree.vhd  }
```

```
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.h-ALU.core/a.b.
    h.i-P4adder.vhd }
#
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.h-ALU.core/a.b.
    h.j-comparator.vhd}
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.h-ALU.core/a.b.
    h.k-shifter.vhd}
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.h-ALU.core/a.b.
    h-alu.vhd}

#Memory Unit
#WB Unit

#DATAPATH
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.i-fetchUnit.vhd
    }
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.j-DecodeUnit.
    vhd}
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.k-executeUnit.
    vhd}
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.l-memoryUnit.
    vhd}
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b.m-WBunit.vhd}
analyze -library WORK -format vhdl {a.b-DataPath.core/a.b-dataPath.vhd}

#Control Unit
analyze -library WORK -format vhdl {a.a-HWCU.vhd}

#DLX
analyze -library WORK -format vhdl {a-DLX.vhd}

#########################################################################
# elaborating the top entity



elaborate DLX -architecture DLX_RTL -library WORK -parameters "M = 32, C
    = 4, N = 5"
compile
report_timing  > Report_DLX_time.txt
report_power > Report_DLX_pow.txt
report_area > Report_DLX_area.txt
create_clock -name "Clk" -period 1.28 Clk
set_max_delay 1.28 -from [all_inputs] -to [all_outputs]
compile -map_effort high
report_timing > Report_DLX_TOPT_time.txt
report_power > Report_DLX_TOPT_pow.txt
report_area > Report_DLX_TOPT_area.txt
#############################################
```

```
# saving files
write -hierarchy -format ddc -output DLX_TOPT.ddc
write -hierarchy -format vhdl -output DLX_TOPT.vhdl
write -hierarchy -format verilog -output DLX_TOPT.v
write_sdc DLX_TOPT.sdc
```