Politecnico di Torino

Microelectronic Systems

DLX Microprocessor: Design & Development

Final Project Report

Master degree in Electronics Engineering
Master degree in Computer Engineering

Referents: Prof. Mariagrazia Graziano, Giovanna Turvani

June 16, 2024

Contents

1 Introduction **1**
- 1.1 Architecture . . . 1
  - 1.1.1 RISC History . . . 1
- 1.2 Instruction Set . . . 1
- 1.3 Pipelining . . . 2
  - 1.3.1 Instruction fetch (IF) . . . 2
  - 1.3.2 Instruction decode/register fetch (ID) . . . 2
  - 1.3.3 Execution/Effective address cycle (EX) . . . 2
  - 1.3.4 Memory access/Branch completion (MEM) . . . 3
  - 1.3.5 Write-back (WB) . . . 3

2 Design **4**
- 2.1 Datapath . . . 4
  - 2.1.1 Instruction fetch (IF) . . . 4
  - 2.1.2 Instruction decode/register fetch (ID) . . . 5
  - 2.1.3 Execution/Effective address cycle (EX) . . . 6
  - 2.1.4 Memory access/Branch completion (MEM) . . . 7
  - 2.1.5 Write-back (WB) . . . 7
- 2.2 ALU . . . 7
  - 2.2.1 SUBSUM . . . 7
  - 2.2.2 COMPARATOR . . . 8
  - 2.2.3 LOGICALS . . . 9
  - 2.2.4 SHIFTER . . . 9
- 2.3 Control Unit . . . 9

3 Physical Synthesis **12**
- 3.1 Synthesis . . . 12
  - 3.1.1 Analyze and elaborate . . . 12
  - 3.1.2 Define constraints . . . 12
  - 3.1.3 Clock definition . . . 12
  - 3.1.4 VHDL netlist . . . 12
- 3.2 Place and Route . . . 12

CHAPTER 1

Introduction

1.1 Architecture

The DLX (pronounced deluxe) is a RISC processor architecture which has a 32-bit load/store architecture. L/S architecture describes most of the RISC microprocessors. All values must be loaded into registers before an execution can take place. The purpose of the RISC architecture is try to reduce the amount of complexity in the instruction set itself and regularise the instruction format so as to simplify decoding of the instruction. Overall it is possible to assume that implementations based on L/S instruction set would be faster in terms of clock rate and performance.

1.1.1 RISC History

In the late '70 DRAM memories were extremely expensive and some programs were not practical because systems with enough memory to run them were too expensive. Therefore CISC (Complex Instruction Set Computer) architecture was used since, thanks to a variable length instruction and the high number of addressing modes, requires smaller instruction memory. On the other hand, the side effect was a high circuitry complexity for fetching and decoding instructions and predicting the starting point of each within the memory. In the following years, thanks to the Moore's law, DRAMs become more affordable and minimise code size became less important, this RISC (Reduced Instruction Set Computer) architecture began to be studied and implemented. Nowadays CISC processors are still widely used because RISC architectures have not yet been enough to overcome, but recent CISC (e.g. Pentium 4) are internally similar to RISC, taking the RISC's advantages and maintaining compatibility.

1.2 Instruction Set

DLX uses only to addressing modes:

- Immediate
- Displacement

The instruction format is simple, with instructions on 32 bits grouped in three types:

- I-type
- R-type
- J-type

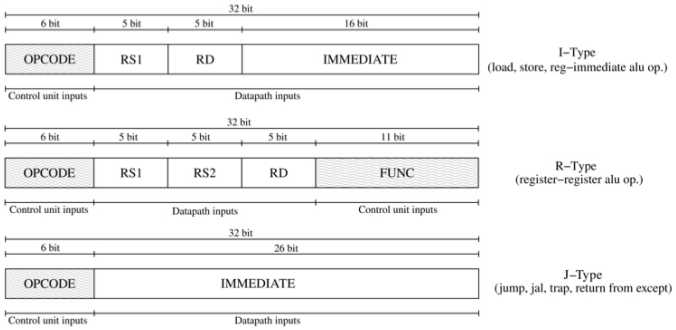

Figure 1.1: DLX Fixed format instruction types

1.3 Pipelining

Pipelining allows to process an instruction performing different independent phases. Thus they can be overlapped, processing multiple operations simultaneously with each operation at a different phase of its processing. With pipelining it is possible to achieve a better throughput than to a sequential processor, with the side effect of an higher latency of the single instruction itself, due to the register banks within stages. Overall with pipelining is possible to enhance the number of instructions executed per unit of time (e.g. clock period). Every DLX instruction can be implemented in at most five clock cycles.

1.3.1 Instruction fetch (IF)

The PC (Program Counter) is sent out and fetches the instruction from the memory into the IR (instruction Register) and it is incremented to address the next instruction.

1.3.2 Instruction decode/register fetch (ID)

The instruction is decoded by the CU (Control Unit), which is in charge of driving all the control signals to the following stages. Decoding is done in parallel with reading registers, which is possible because these fields are at a fixed location in the DLX instruction format This technique is known as fixed-field decoding. Within this stage is possible to read the RF (Register file) into two temporary registers (A and B), and sign-extend the immediate, storing temporary it into the IMM register.

1.3.3 Execution/Effective address cycle (EX)

Once the CU configures properly all the EX-stage control signals the ALU operates on the operands (either A, B and/or IMM) prepared in the previous cycle and the result is stored in the output register. Different ALU operations can be executed:

1. Memory reference
2. ALU operation
3. Resister-immediate operation
4. Branch address

1.3.4 Memory access/Branch completion (MEM)

Within this stage is possible to access the data memory if:

1. Instruction is a load, data return from memory and is placed in the LMD (Load Memory Data) register
2. Instruction is a store, the data from the B register is written into memory

In either case the address used is the one computed in the prior cycle and stored in the ALUOutput register.

1.3.5 Write-back (WB)

In the WB stage the result is written back into the register file, whether it comes from the memory system (LMD) or from ALU.

CHAPTER 2

Design

2.1 Datapath

The datapath is the nucleus of a computational system where calculations are executed.

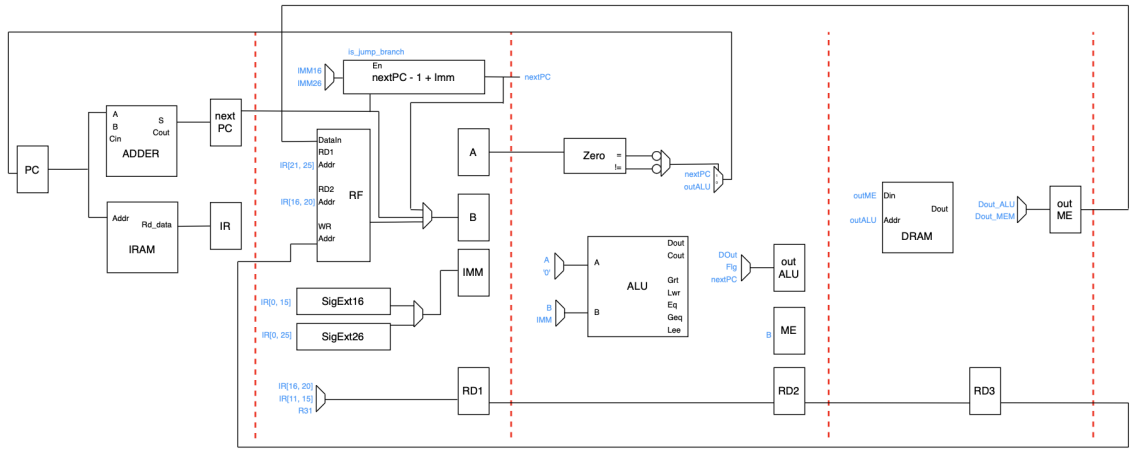

Figure 2.1: DLX datapath

2.1.1 Instruction fetch (IF)

The fetch stage has been implemented by the following components:

- PC

  The PC is a 32-bit register which addresses the instruction memory (IRAM).

- IRAM

  The IRAM is a 32-bit wide memory which reads a target file on the falling edge of the reset signal, which stores the compiled ASM script that has to be executed by the DLX.

- IR

  The IR stores the instruction fetched that will be available to the CU in the next clock cycle to compute the proper control words that as to be forwarded through the entire pipeline.

- PC incrementer by 1

  In order to compute the next PC an adder is used, which gets the current PC and a 1 as inputs. For this implementation, since the IRAM is 32-bit wide and the instructions are 32-bit wide too, the program counter has to be incremented by just 1 to refer to the next instruction that will be executed next. The incrementer has been implemented as a 32-bit P4 adder in order to reduce the delay due to the ripple of the carry.

2.1.2 Instruction decode/register fetch (ID)

The decode stage includes the following components:

- RF

  The RF has two read ports and one write port, allowing to handle up to two read and one write operation within the same clock cycle. The two read address ports are driven by a subset of the IR's bits, which corresponds to the register addresses of the two source addresses for the R-type operations. Moreover both the data in and write address ports are driven by inputs coming from the write back stage, the one in charge of commit the final result driving it back into the RF itself. In addition it contains 32 registers, on 32 bits each. In order to hit the correct timing behaviour of the whole pipeline, each stage has to compute the stage execution within one clock cycle. Therefore the RF is sensitive to the falling edge of the clock instead of the rising one. Thus the incoming data as to be driven as input to the RF by the end of the first half of the clock cycle, on the falling edge the RF will provides the requested output, then driven to the bank register.

- SIGEXT

  For both I-type and J-type instructions an immediate value has to be extracted from the IR and extended to 32 bits, taking into account the sign. Based on the instruction type the incoming immediate can be either on 16 bits or 26 bits for I-type and J-type respectively. Therefore two different SIGEXT components have been added to the datapath and both of them will compute the bit extension, but only one output will be driven into the IMM register, selected by the following multiplexer. As can be seen the SIGEXT will take place, independently on the instruction type. Will be CU's duty to enable or not the IMM register based on the instruction that is ongoing.

- A, B and IMM

  The register bank includes A, B and IMM registers. A and B are used to store the register file's outputs, while IMM to handle the immediate value extended on 32 bits as seen before. Based on the instruction both the enable and reset signals are driven properly to set up the stored values that will be computed by the following stages.

- RD1

  In order to save back the instruction result at the end of its execution we have to keep the destination address and propagate it through the pipeline up the last stage. Thus decode stage has a RD1 register which gets the address from the IR and stores it during the clock cycle. This register is 5-bit only since the register file has 32 registers and the source register filed in the fix-format of the instruction has 5 bits only. The RD1 has 3 possible different sources. Two of them are due to the fact that, based on the instruction type, the source register address can be from a different bit range. The third one is instead the address of the last register (register 31), needed because for the JAL instruction the next PC has to be stored within this position during the link phase.

- JUMP/BRANCH components

  In order to handle the jump and branch operations, the correct target PC is computed within this stage. Moving it into the decode stage from the execute one allows to reduce the penalty for the jump instruction (only) since only one "wrong" fetch will occur. For JAL operation this is true anymore, since the next PC has to be driven through all the stages before being written back. Branches are more critical too, since before updating the PC with the correct one, verify the truthness of the condition is needed. Different components are used to handle these different instructions. The multiplexer which is driving the register B input gets three different sources: B, next PC and the new target PC. Based on the control signals driven by the CU the proper value is loaded into the register. The other multiplexer is in charge of driving the correct immediate, either the 16-bit immediate or the 26-bit one if the instruction is a jump or a branch respectively. Lastly one component is used to compute the target PC, that is equal to PC + IMM. Therefore this component drives the nextPC - 1 + IMM if the enable is active, otherwise it simply forwards the input (next PC) itself.

2.1.3 Execution/Effective address cycle (EX)

The following components belong to the current stage:

- ALU

  The ALU (Arithmetic Logic Unit) is the unit that actually executes the operations, based on the inouts and the control signals. It has two inputs only, A and B, which both have different sources. A can be either zero or A, while B can be either IMM or B. As output some signals are driven, one for the outgoing result, while the others are signals coming from the comparison (not only equality) between the two selected inputs.

- OUT ALU

  One of the three registers of the register bank is the one that stores the ALU's output. It can be either the output itself, the output control signals (comparison) or the next PC. The last one is needed since may be needed to store the PC into the register file, for example when a JAL instruction is executed.

- ME

  In case of write operation the value is stored into the B register. Its output is then driven as input of the ME register, which will drive the data input of the DRAM in the next clock cycle, within the MEM stage.

- RD2

  The target address is still forwarded from RD1 to RD1.

- BRANCH components

  Some components have been added within the execute stage in order to verify the condition truths. With zero detector is possible to verify if the A values is either a zero or not. Then passing through a multiplexer, which simply chooses the output based on if a BEQ or a BNE is ongoing, drives the result as selection signal of another multiplexer. The expected behaviour is driving the next PC if the ZERO-enable signal is high or the output itself, driving as new PC the target one, driven as output by the ALU.

2.1.4 Memory access/Branch completion (MEM)

The memory stage includes the following components:

- DRAM

  The DRAM has a width of 32 bits and 1024 as depth and is addressed by the OUT ALU register. This memory is actually addressed is and only is a load or store instruction occurs.

- OUT ME

  The OUT ME register is a 32-bit register which handle the result of the current stage. It has two possible sources, the DRAM output in case of memory operation or the OUT ALU if read the memory is not needed.

- RD3

  The target address is still forwarded from RD1 to RD2. This is the last step, since the next stage (Write Back) is the one which drives the target address to the RF.

2.1.5 Write-back (WB)

The write back stage is simply in charge of commit the final result, driving back the write address and the value that has to be written into the RF. Over this clock period the CU will raise the proper control signal (Write enable) to the RF in order to guarantee that the write back takes places.

2.2 ALU

The inner implementation of the ALU includes four main components.

2.2.1 SUBSUM

The first component is a P4 adder, in which three blocks are defined:

- PG-logic

  Generates the P and G signals from inputs Generate: generates a true Cout independently on Cin Propagate: propagates a carry, thus produces a true Cout if and only if it receives a true Cin.

- Carry-logic

  The carry look ahead family is particularly rich of solution differing in most of the cases for the carry-logic. One of these solutions is called tree adder. Different tree implementations exist such as Kogge-Stone adder, Brent and Kung adder and Sparse tree adder. The main disadvantages of the first two are the large area required and the complexity in terms of interconnection system. Therefore the last one has been implemented. The sparse tree generates every fourth carry in the adder, instead of generate the carry for each bit (e.g Kogge-Stone adder). Therefore the idea of this architecture is to speed up the critical carry path by moving a substantial portion of the carry-logic from the main carry-tree to a noncritical side-path, the SUM-logic.

- SUM-logic

  The SUM-logic consists of a 4-bit conditional sum generator (carry select block) that produces sums assuming input carries of 0 and 1. Then through a control signal the correct addition is driven as output.

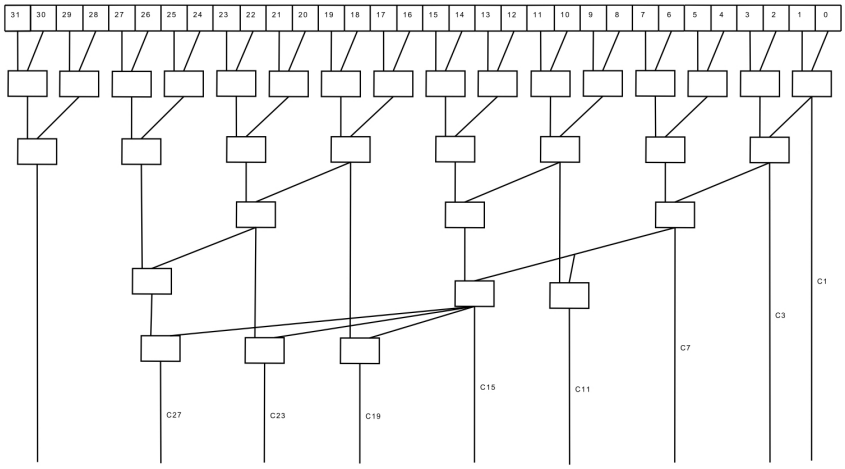

Figure 2.2: DLX Adder and Subtractor

The same structure has been used as a subtractor exploiting the XOR properties. All XOR are connected to the input Cin which is considered as a control signal to select either SUM or SUB. When Cin is 0 we get B itself (B XOR 0 = B), while when it is 1 B is complemented and the Cin is settled to 1. Therefore A + B' + 1, that is equivalent to A - B, will be computed.

Overall both the addition and subtraction are computed by one adder only.

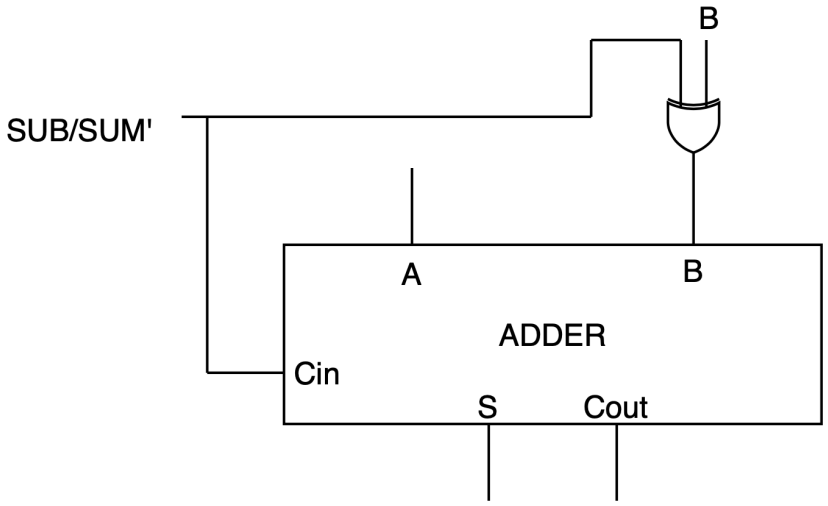

Figure 2.3: DLX Adder and Subtractor

2.2.2 COMPARATOR

The comparator is a Not only Equality Comparator, which exploit the ALU instead of use a specific hardware block. The subtraction is computed and the result and the carry out are driven to a set of logic gates which are in charge of recognise one specific condition.

2.2.3 LOGICALS

The LOGICALS block is needed since the instruction set contains some bitwise logic operation. One straightforward may be use six different logic gates and select among the results only one using a multiplexer. However this solution leads to high latency, due to the huge size of the multiplexer. One clever way to solve this problem is to implement the T2 logical block. This logic unit is implemented with two NAND-gate levels, the first one made of four 3-input NANDs and the second level with a 4-input NAND only. This block is driven by the data inputs (A and B) and by a selection signal. This control signal has to be driven properly in order to combine different contibutes together and get the final operation implemented only with NAND gates.

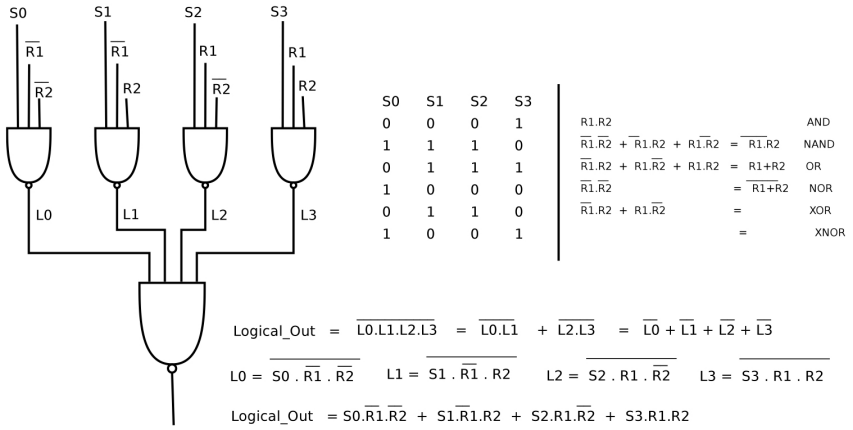

Figure 2.4: DLX logicals

2.2.4 SHIFTER

The shifter has been implemented at behavioral level and is able to perform a logical right/left shift. Some additional control signals are available for further implementation of other shift operations sush as arthmetical or rotate.

The final result it then selected and driven as output by a multiplexer which handles the SUMSUB, LOGICALS and SHIFTER output.

2.3 Control Unit

The CU (Control Unit) of a processor is the part which process the status of the system and generates the commands to have the datapath perform the needed operations. An Hardwired CU has been implemented. The hardwired approach uses the IR content as an encoding of an address to be fed to a look-up table containing all the control words.

As can be noticed the CU has two main blocks and some registers. The expected behavior is that as soon as the reset goes low the control fetch block is active and starts driving the control signals to the fetch stage. This block will always drive the same control signals to the IF stage in order to fetch the instruction and load it into the IR. On the other hand if its enable is off no fetch operation will take place. The remaining part of the CU has to be enabled only on the next clock cycle, thus a 1-bit register has been introduced to delay the enable signal by one clock cycle. Therefore only after one clock period the LUT and the other registers will be activated and the new IR content will address the LUT. The LUT's output will be then delayed by some clock cycle depending on the stage to which

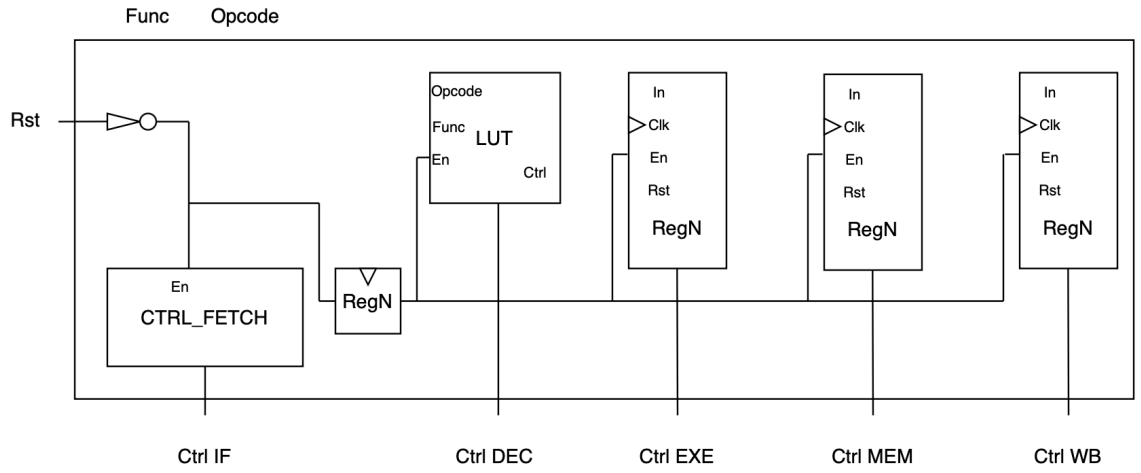

Figure 2.5: DLX Control Unit

the control signals belong. Therefore in order to feed the control word of each stage at the right clock cycle the LUT's output pass through a series of registers.

Instruction set coding					
General instructions			Register-register instructions		
Mnemonic	OP-CODE	Operation	Mnemonic	OP-CODE	Operation
J	0x02	PC <-- PC + imm16	SLL	0x04	R[C] <-- R[A] << R[B]
JAL	0x03	R[31] <-- PC + 4; PC <-- PC + imm16	SRL	0x06	R[C] <-- R[A] >> R[B]
BEQZ	0x04	if(R[A] == 0) PC <-- PC + imm16	ADD	0x20	R[C] <-- R[A] + R[B]
BNEZ	0x05	if(R[A] != 0) PC <-- PC + imm16	SUB	0x22	R[C] <-- R[A] - R[B]
ADDI	0x08	R[B] <-- R[A] + imm16	AND	0x24	R[C] <-- R[A] & R[B]
SUBI	0x0a	R[B] <-- R[A] - imm16	OR	0x25	R[C] <-- R[A] | R[B]
ANDI	0x0c	R[B] <-- R[A] & imm16	XOR	0x26	R[C] <-- R[A] xor R[B]
ORI	0x0d	R[B] <-- R[A] | imm16	SLE	0x2c	if(R[A] <= R[B]) R[C] <-- 1 else R[C] <-- 0
XORI	0x0e	R[C] <-- R[A] xor imm16	SNE	0x29	if(R[A] != R[B]) R[C] <-- 1 else R[C] <-- 0
NOP	0x15	Idle one cycle	SGE	0x2d	if(R[A] >= R[B]) R[C] <-- 1 else R[C] <-- 0
SNEI	0X19	if(R[A] != imm16) R[C] <-- 1 else R[C] <-- 0			
SLEI	0X1c	if(R[A] <= imm16) R[C] <-- 1 else R[C] <-- 0			
SGEI	0x1d	if(R[A] >= imm16) R[B] <-- 1 else R[B] <-- 0			
LW	0x23	R[B] <-- M[imm16 + R[A]]			
SW	0x2b	M[imm16 + R[A]] <-- R[B]			
SLLI	0x14	R[B] <-- R[A] << imm16			
SRLI	0x16	R[C] <-- R[A] >> imm16			

Figure 2.6: DLX Instruction Set

CHAPTER 3

Physical Synthesis

3.1 Synthesis

After the implementation and a simulation phase all blocks have been synthesize using Synopsys Design Compiler as synthesizer and Synopsys Design Vision as Graphical Interface.

3.1.1 Analyze and elaborate

During this step all files have been analyzed in order to check if the design is correctly synthesizable. After this the top-level entity is elaborated.

3.1.2 Define constraints

For the synthesis additional constraints have been settled using `set_max_delay` command. This command limits the maximum length of the paths.

3.1.3 Clock definition

A clock signal must be defined using the command `create_clock` -CLK "CLK" -period 2 Clk. This command forces a CLK generator at the input of the physical pin Clk.

3.1.4 VHDL netlist

Lastly the VHDL netlist of the whole design is generated together with its DLX.sdc file which contains all the constraints settled before.

3.2 Place and Route

After the netlists generated with the physical synthesis, we move to the final step of the project. The place and route is performed in Cadence Innovus. The final schematics of the design is obtained through a sequence of steps really similar to the one used during the last Microelectronic system's laboratory and all the cells have been placed and connected to each other. The power supply and GND distribution is achieved through a pair of rings around the die and some vertical lines on the two highest metal levels (metal M10 and metal M9), meanwhile all lower metals have been used to create interconnections between all cells.

```
Report : timing
         -path full
         -delay max
         -max_paths 1

  Startpoint: datapath_dlx/IR/state_reg[27]
              (rising edge-triggered flip-flop clocked by CLK)
  Endpoint: datapath_dlx/RF/REGISTERS_reg[5][1]
            (rising edge-triggered flip-flop clocked by CLK')
  Path Group: CLK
  Path Type: max

---------------------------------------------------------------------------
  data required time                                                  0.97
  data arrival time                                                  -0.97
---------------------------------------------------------------------------
  slack (MET)                                                         0.00
```

Figure 3.1: DLX Timing report

Further information about the place and route

From the Cadence Innovus tool, we had the opportunity to get a lot of useful and interesting information about our implementation. In addition to the information about power consumption and timing already discussed previously, we get informations about the total number of gates, with a total of 18717 gates (9383 cells) and an area of 14936.4 um^2 for the `DLX_DP_CU` module.

```
Total Power
-----------------------------------------------------------------------------------------------
Total Internal Power:        6.04187302        59.7517%
Total Switching Power:       3.77919405        37.3747%
Total Leakage Power:         0.29056036         2.8735%
Total Power:                10.11162742
-----------------------------------------------------------------------------------------------


Group                           Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------------
Sequential                         4.014      0.5346      0.1119       4.661       46.09
Macro                                  0           0           0           0           0
IO                                     0           0           0           0           0
Combinational                      1.843       2.097      0.1776       4.118       40.73
Clock (Combinational)             0.1842       1.147    0.001083       1.332       13.18
Clock (Sequential)                     0           0           0           0           0
-----------------------------------------------------------------------------------------------
Total                              6.042       3.779      0.2906       10.11         100
-----------------------------------------------------------------------------------------------


Rail                  Voltage   Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------------
Default                   1.1      6.042       3.779      0.2906       10.11         100


Clock                           Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------------
CLK                               0.1842       1.147    0.001083       1.332       13.18
-----------------------------------------------------------------------------------------------
Total                             0.1842       1.147    0.001083       1.332       13.18
-----------------------------------------------------------------------------------------------
Clock: CLK
Clock Period: 0.002000 usec
Clock Toggle Rate:  1000.0000 Mhz
Clock Static Probability:  0.5000
```

Figure 3.2: DLX Power report

```
Number of ports:                          10450
Number of nets:                           21625
Number of cells:                          12768
Number of combinational cells:             9839
Number of sequential cells:                1434
Number of macros/black boxes:                 0
Number of buf/inv:                         2924
Number of references:                         2

Combinational area:                9961.966019
Buf/Inv area:                      1847.369990
Noncombinational area:             6488.271765

Total cell area:                  16450.237784
```

Figure 3.3: DLX Area report

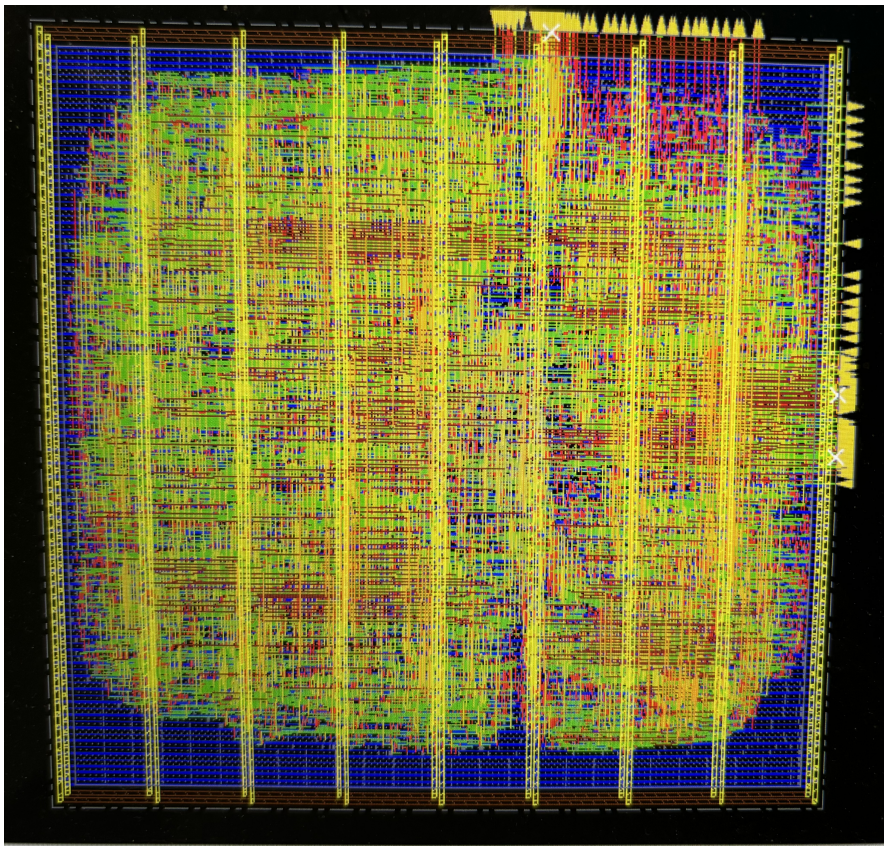

Figure 3.4: DLX after palace and route