Politecnico di Torino

Microelectronic Systems

DLX Microprocessor: Design & Development

Final Project Report

Master degree in Electronics and Computer Engineering

Referents: Prof. Mariagrazia Graziano, Giovanna Turvani

Authors: group 21

Dario Castagneri, Gianpietro Noto, Francesca Silvano

October 21, 2022

Contents

CHAPTER 1

Introduction

1.1 Abstract

The goal of the project is the design of a 32-bit RISC processor architecture based on the one developed by John L. Hennessy and David A. Patterson [1]. For its realization, the standard design flow has been followed. Starting from the DLX specifications, the RTL description of the architecture has been derived. The verification phase has been carried out by using testbenches dedicated to each unit in combination with *QuestaSim* by Siemens. Then, the architecture has been synthesised by using *Design Compiler* by Synopsis and a 45nm standard cell library. Finally, the physical design of the entire architecture is extracted along with the Layout and Place&Route descriptions by the means of *Innovus* by Cadence.

CHAPTER 2

Implementation overview

The DLX is a 32-bit fixed length instructions pipelined processor with an in-order execution. The pipeline is composed of 5 stages: Fetch, Decode, Execution, Memory and WriteBack. In our scenario, the top-level entity is made of four elements:

- **Datapath:** It implements the 5 stages pipeline.
- **Control Unit:** It is an HardWired Control Unit, it generates the control words that are used to manage the datapath elements accordingly to the operation to be performed.
- **DRAM:** it is used in presence of Load and Store operations in case of Store operations, the content of the destination register is saved in the DRAM which is accessed in write mode, while in case of Load operations the DRAM is accessed in read mode and the value is stored in the register of the Memory stage.
- **DRAM:** it is used by the Windowed Register File when performing the context switch; for spill and fill operation to saving all the register file of the current windows and restore them with the fill.
- **IRAM:** It is filled with the binary codes of the assembly instructions generated by the compiler. During the fetch operation, it is accessed by the program counter (PC) and it delivers back the instruction to the instruction register (IR) of the Fetch stage.

2.1 Supported instructions

Operation	Code
SLL	0x04
SRL	0x06
SRA	0x07
ADD	0x20
SUB	0x22
AND	0x24
OR	0x25
XOR	0x26
SEQ	0x28
SNE	0x29
SLT	0x2a
SGT	0x2b
SLE	0x2c
SGE	0x2d
SLTU	0x3a
SGTU	0x3b
SLEU	0x3c
SGEU	0x3d
MUL	0x3e
NAND	0x3f
NOR	0x40
XNOR	0x41
LHI	0x42
LLI	0x43

Table 2.1: Rtype instructions

Operation	Code
J	0x02
JAL	0x03
BEQZ	0x04
BNEZ	0x05
ADD	0x08
SUB	0x0a
AND	0x0c
OR	0x0d
XOR	0x0e
SLL	0x14
NOP	0x15
SRL	0x16
SRA	0x17
SNE	0x19
SLE	0x1c
SGE	0x1d
LW	0x23
CALL	0x3e
RET	0x3f

Table 2.2: General instructions

2.2 Branch and Jump management

The calculation of Jump and Branch target address is made in the decode stage in order to prevent the fetch of additional instructions that will not proceed in further pipeline stages.

2.3 Forwarding

Used to prevent data hazards. Four signals are managed:

- **alu forwarding 1:** to perform the forwarding from the end of the execute stage to the beginning inserting the value into the register one of the alu.
- **alu forwarding 2:** to perform the forwarding from the end of the execute stage to the beginning inserting the value into the register two of the alu.
- **mem forwarding 1:** to perform the forwarding from the end of the memory stage to the beginning of the execution inserting the value into the register one of the alu.
- **mem forwarding 2:** to perform the forwarding from the end of the memory stage to the beginning of the execution inserting the value into the register two of the alu.

CHAPTER 3

Design architecture

3.1 Datapath

In the following is shown the designed Datapath and its input signals coming from the Control Unit.

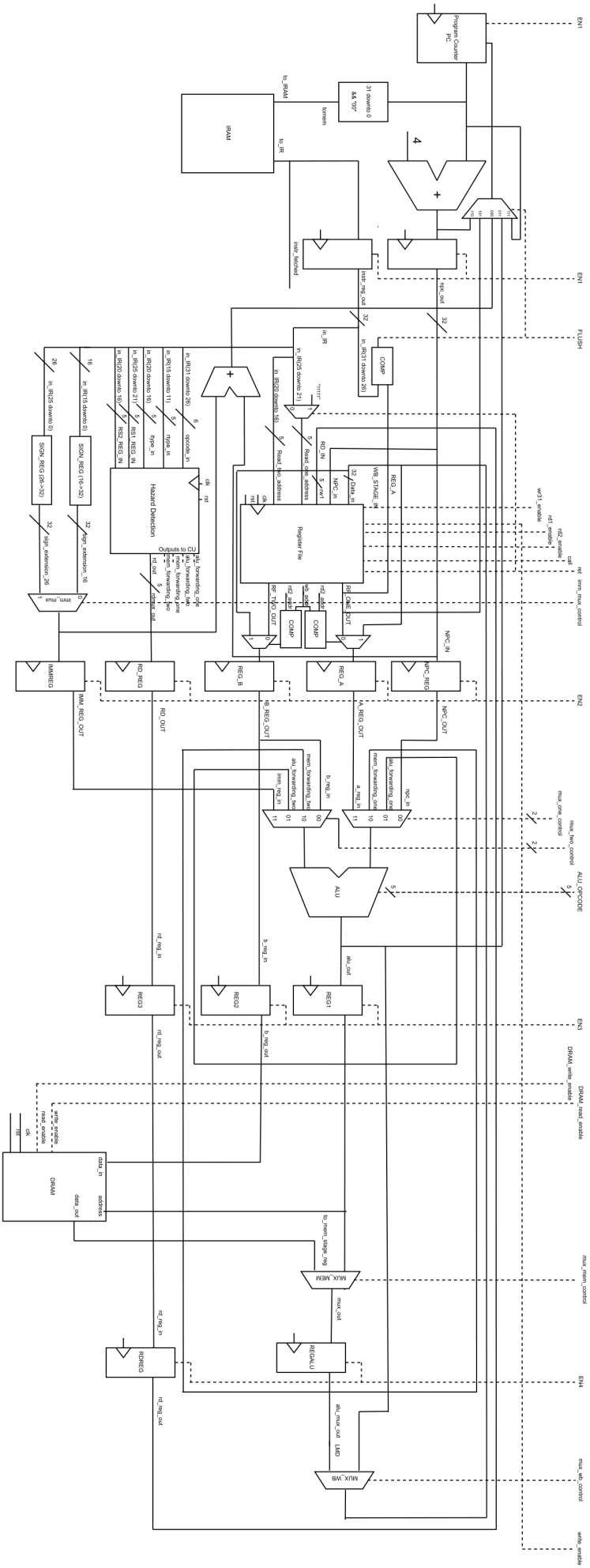

Figure 3.1: Datapath schematic

3.2 Fetch Unit

The Instruction Fetch Unit (IF) has the role of fetching the instructions from the IRAM and passing them to the following Unit (Decode Unit) in order to execute them. It is made up of several elements:

- **Program Counter** (PC): it is an adder that sum 4 to the current value of the PC; there is an input port which is used to set a value in presence of Jump or Branch instructions.
- **Pipeline Registers**: Instruction Register (IR) and Program Counter register (NPC) register store respectively the current fetched instruction and the value of the New Program Counter, are located at the output of the IF stage and are used as pipeline registers in order to divide the Fetch Unit from the Decode Unit.
- **Multiplexer** (MUX_PC): it is driven by a signal coming from the following stage (ID stage) which has the role of changing the value of the Program Counter when there is a Jump or Branch instruction.

Ideally the IRAM could be considered as a component inside the Fetch Unit, but in our component organization it is outside the Datapath. In the following figure can be observed the previously described components with their connection with the IRAM.

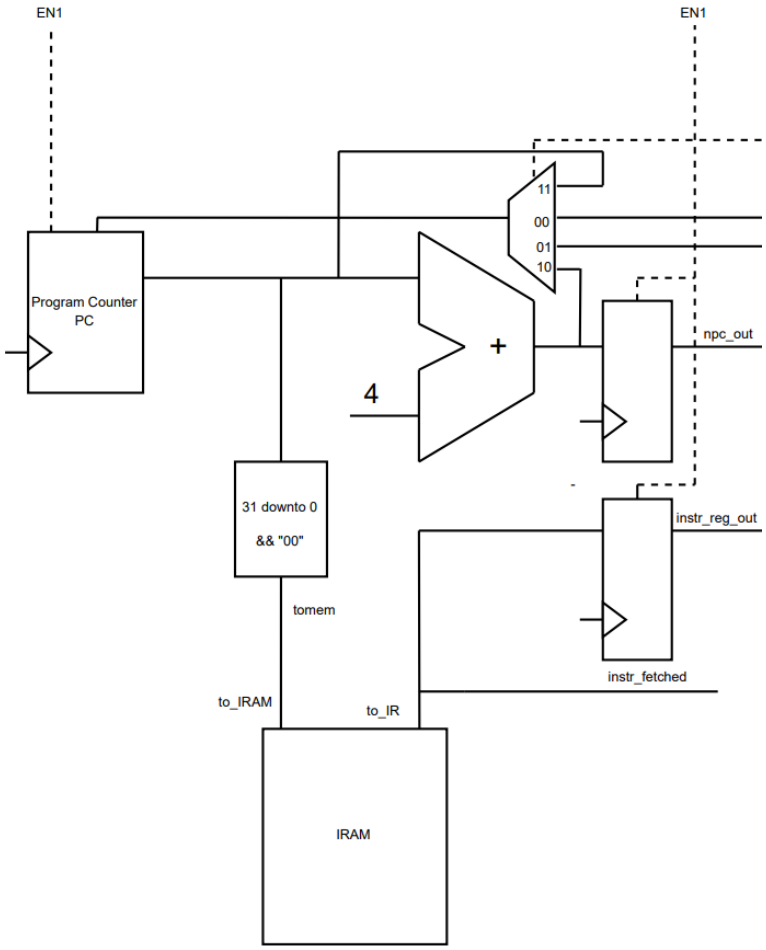

Figure 3.2: Fetch Unit schematic

3.3 Decode Unit

The Instruction Decode Unit (ID) has the role of decoding the fetched instruction, read the Register File (RF), and to compute the new program counter value in presence of a Jump/Branch instruction. It is composed by the following elements:

- **Windowed Register File**: a Register File is an array of registers used by the processor in a central processing unit (CPU). It is a 2 read 2 write Register File and is a fast access storage, the second write port is used only in presence of a Jump And Link or a Call instruction so that the current value of the Program Counter is written in the last register of the Register File (R31). Adding the windowing feature means allocating register space for more than one procedure, so that whenever a new one is called, a context switch is performed and that procedure get a new set of registers inside a new window.
- **A comparator block** (COMP): it has the OPCODE of instructions as input and it has the role of understanding wether there is a conditional Branch or a Jump instruction, then its output is used to drive the 4to1 Multiplexer of the Fetch Unit in order to select the right value of the Program Counter to use for fetching the next instruction.
- **New Program Counter Adder**: on one side it has the new program counter as input, while on the other side there are two possible inputs driven by a multiplexer that are the 16 LSB or the 26 LSB of the current instruction; it is used in presence of jump instructions.
- **Hazard detection block**: it has the role of selecting the right destination register R_D by looking at the OPCODE of the instruction and then to understand if there is a data hazard.
- **Pipeline registers**:
  - NPC register, stores the delayed value of the New Program Counter.
  - REG_A register, first output of the Register File.
  - REG_B register, second output of the Register File.
  - RD_REG register, stores the address of the destination register decoded by the MUX.
  - IMM_REG register, in presence of an I-type instruction it stores the 16-bit immediate converted in 32-bit.

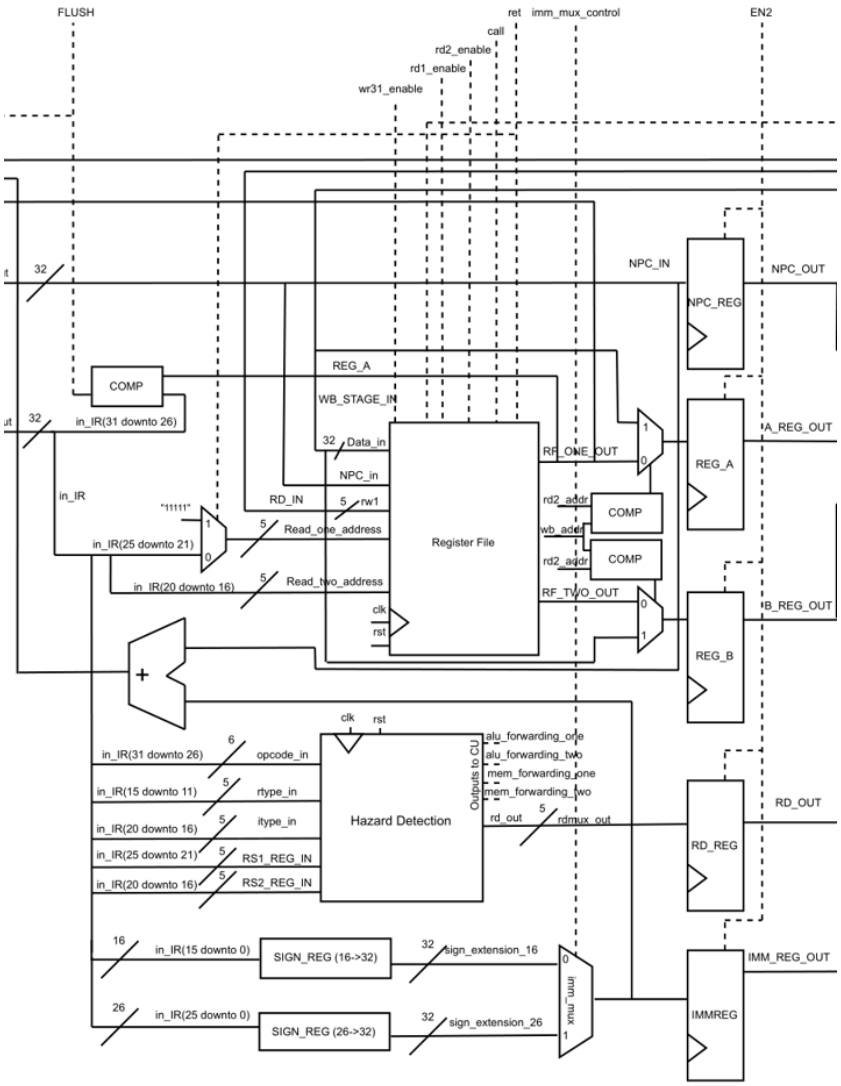

Figure 3.3: Decode Unit schematic

3.3.1 Windowed Register File

The windowed register file is a register file with an architecture around it that allows to handle calls to multiple subroutines. A simple register file once a subroutine has been called should block the cu for the time it takes to store the registers in the DRAM (one register per clock cycle for 32 registers = 32 clock cycle means a lot of time) or simply not handle the call to other subroutine.
The windowed register file allows the cu to see a normal register file and have all its registers available (black box) while inside, the physical register file is surrounded by a logic that allows it to be divided into blocks called windows.
We have modify it in such a way to be a register file with 2 write and 2 read, the second write is performed only when a branch is required and store in r31 the value of the new program counter. Each of these windows is assigned to a subroutine called by cu. In the base windowed register file each windows contains:

- Eight input registers: input to the subroutine from the previous one

- Eight local registers: registers dedicated for the subroutine
- Eight output registers: registers used to save the results of the subroutine

Each subroutine then has registers that are shared among all: eight global registers.
Input and output are strictly related because the input of a subroutine are the output of the calling one.

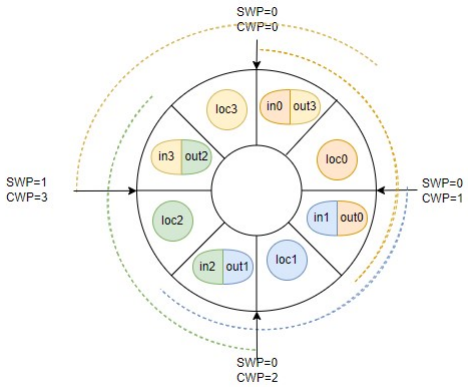

Figure 3.4: schematic windowed register file

A windowed register file is characterized by two pointers: the cwp which indicates the current windows and the swp which indicates the next windows to be saved in memory. Once a subroutine is called, when the physical register has no more space, the input and the local registers of the windows pointed to by swp are saved in memory to make room for the new subroutine called. When you have a return, the registers saved in memory are re-established in the register file in the windows pointed to by swp.

Structure of Windowed Register File The structure contains:

- **Physical Register file**: contains all the registers for all the windows (in the figure we have 4 windows so 72 registers)
- **Decoder**: It takes the address and generate all the enable for each register in the physical register file
- **Enable generator**: for the fill phase generate all the enable for the physical register file then shifted by the decoder
- **Current Window Pointer** (CWP): register with some logic that store the value of the current windows and update it when a call/return is performed.
- **Saved Window Pointer** (SWP): register with some logic that store the value of the windows that needs to be spill/fill
- **Selection block**: Windows selector for the outputs and spill
- **Address generator**: Is more like a counter for the selection of all the register to spill in the memory

Now we can look one block at the time.

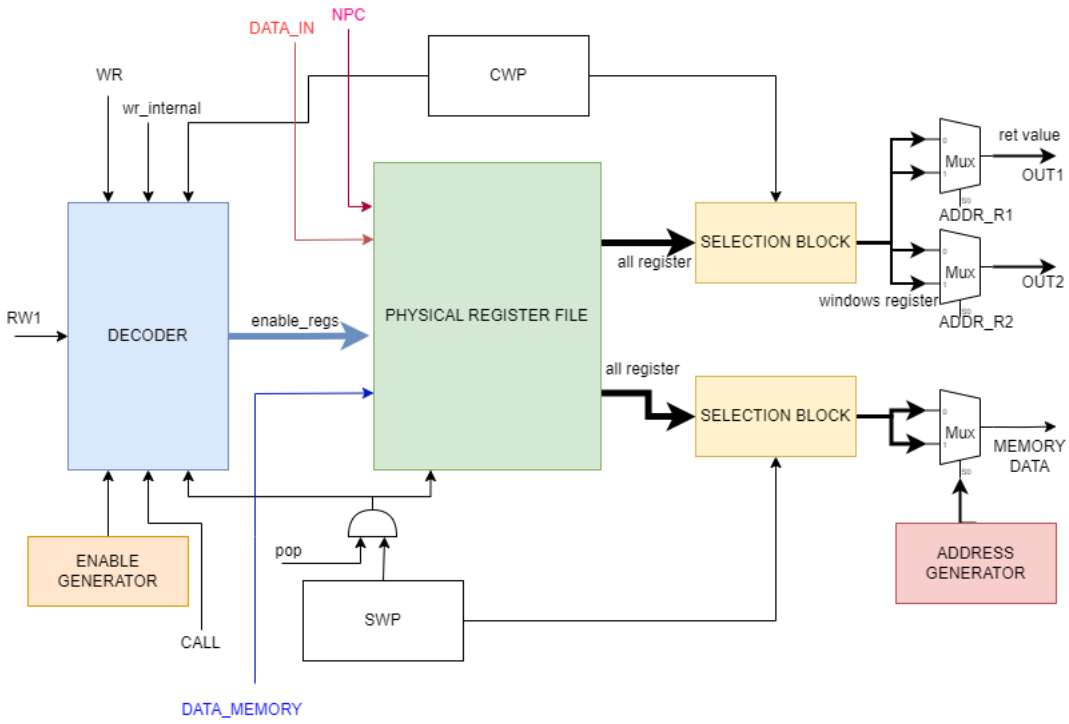

Figure 3.5: Structure of the windowed register file

Physical register File The Physical register file groups all the WRF registers.

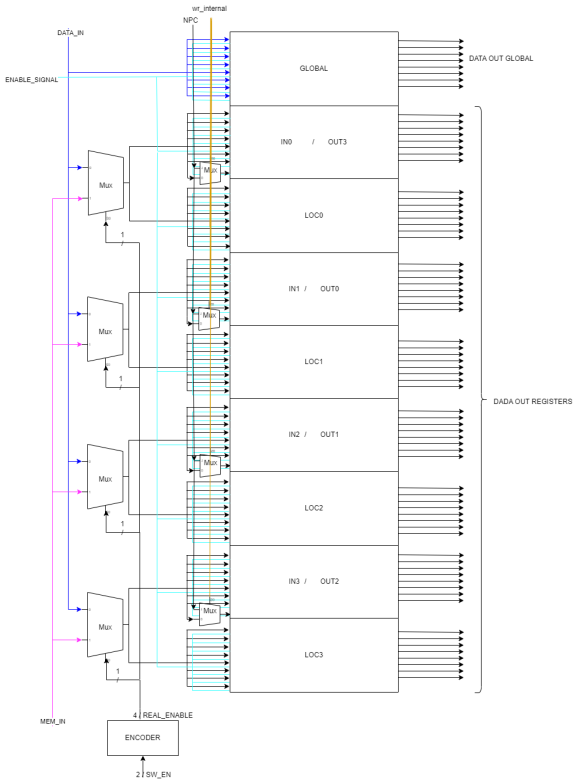

Figure 3.6: Structure of the Physical Register File

The encoder takes the SWP value and generates the mux selection values for the register input data. The SWP is at 0 when there is a write for any windows, while it activates the signals coming from the memory for a given windows when there is a fill action. The enable signal generated by the decoder sends a bit to each register of each windows. The register is written only when its enable is activated (one enable at a time). The only registers that do not have a multiplexer are the global registers because there is no need to take their values from memory. All the registers are in output. One register is important (and it is not possible to

CWP and SWP CWP and SWP are two registers with associated logic for calculating the windows. A mux selects the next windows based on the sel signal, when 01 goes to the next windows, when 10 goes back to the previous windows while 11 and 00 keeps the current windows.

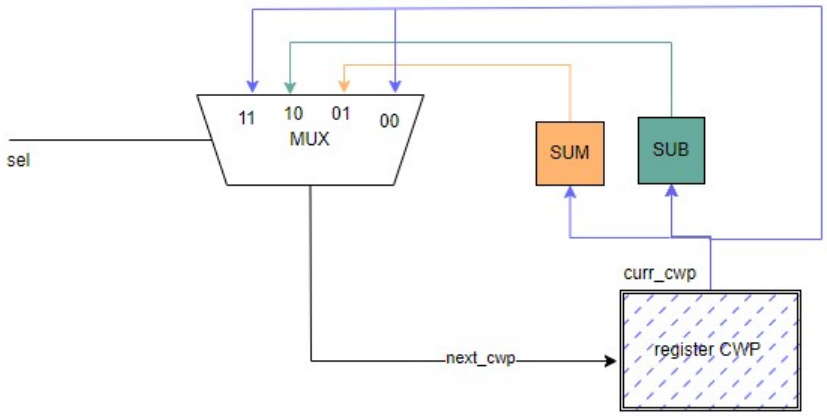

Figure 3.7: Structure of the CWP and SWP registers

Enable Generator and Address Generator Enable Generator and Address Generator are both based on similar FSMs, the only difference between these two blocks is that the enable generator takes care of generating the enables to restore the registers and therefore thinking of the memory as a stack you start from the last register of the locales up to the first register of the windows inputs in 32 bits while the address generator generates the addresses starting from the first address of the input block up to the last local and therefore in 5 bits.

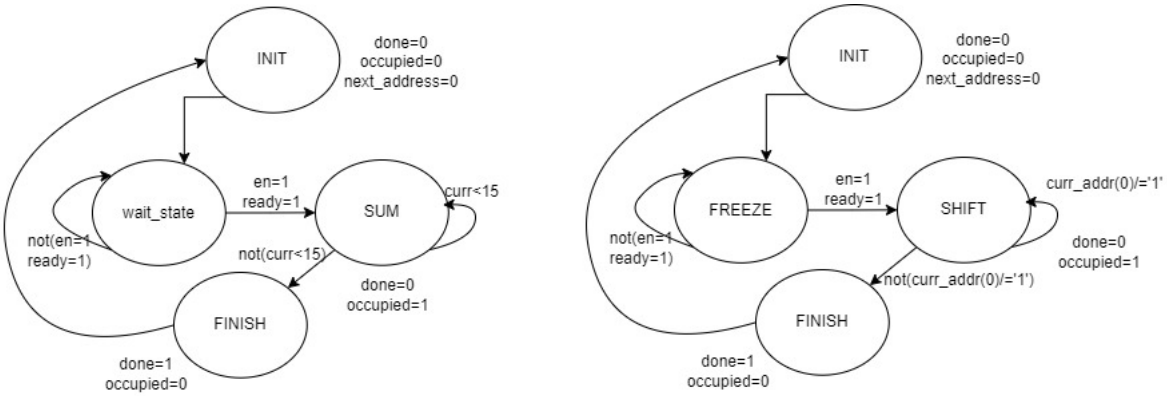

Figure 3.8: Structure of the Selection block and the output selection

Selection Block and output selection The Selection Block takes as input all the data registers from the physical register file and through the value of the CWP register (taken as input) it selects the registers of the current windows sending them to the muxes which, through the address taken from inputs ADDR_R1 and ADDR_R2, selects the register that is requested in output. The figure below describes the reading through a single mux but from the general structure it can be seen that there is a mux for each reading address and a Select block also for the spill operation which in this case takes the value as input of the SWP register.

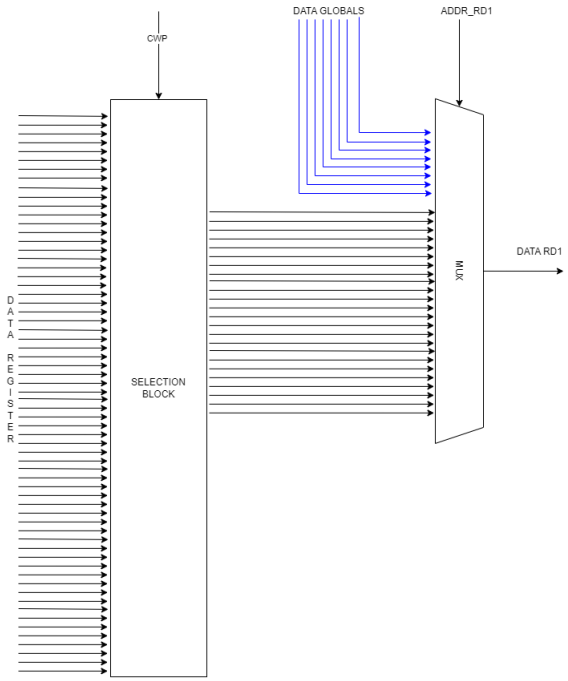

Figure 3.9: FSM of the AG and EG

Decoder The decoder manages the writing and filling of the register file, it gives precedence to the fill operation, checking when the input signal from the SWP is different from 0, the input signal is not the value inside the SWP register but the its value computed in AND with the pop signal. If the SWP signal is different from 0 then a fill operation is in progress and the decoder outputs the enable signal shifting correctly the enables coming from the enable generator. For the write operation, on the other hand, it is based on the CWP input signal, translates the incoming address RW1 into enable signal and then shifts it based on the current windows. It also detect when a call is performed enabeling the register 31 of the next windows to store the value of the new program counter, the signal wr_internal detect when a jump is performed and the decoder can enable the register 31 to store the npc.

Windowed FSM In order to save and read data from the memory we use the Windowed Register File Control. The WRFC through the data signals from the WRF (pop, push, done_fill, done_spill) generates the address for the memory, which is treated as a stack, and is used as data transmitter between register file and memory. For the spill as soon as push = 1 it starts generating the addresses starting from the first (all 0), when all the registers are saved in memory done_spill will have the value 1 and therefore the generation of the addresses is suspended leaving the address in freeze. For the fill (pop = 1) we start from the address left suspended by the spill, decreasing it every clock cycle and sending the data from the memory to the corresponding register.

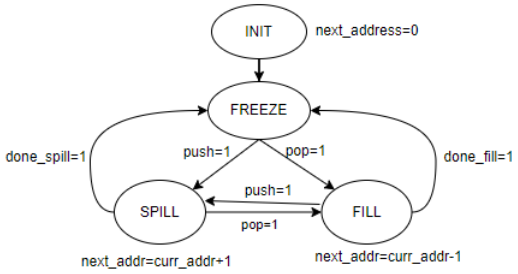

Figure 3.10: FSM of windowed control

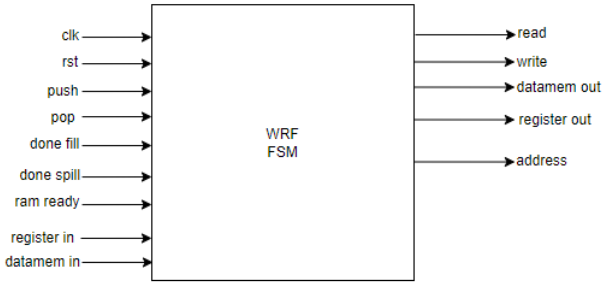

Figure 3.11: Structure of the windowed control

3.3.2 Hazard Control

During the execution of the instructions in the pipeline we can have various types of Hazard. In this section we will cover Data Hazards.
According to the order of writing and reading of the instructions we can have three types of data hazards:

- **WAR**: There is a Write After Read hazard when y tries to write to an operand before x read it, in this way the value taken by x is the one written by y, that is incorrect.

x	SW 0(R2),R1	IF	ID	EX	MEM1	**MEM2**	WB
y	SUB R1,R4,R5		IF	ID	EX	**WB**	

  This happens when the instructions write before reading the next one in the pipeline. Due to how the pipeline has been structured, all registers are read in the ID phase and then written in the WB phase, so these hazards are rare if not impossible.

- **WAW**: There is a Write After Write hazard when y tries to write to an operand before x, in this way the value written by x will remain in the register instead of the one written by y as it should be.

x	LW R1,0(R2)	IF	ID	EX	MEM	MEM2	**WB**
y	SUB R1,R4,R5		IF	ID	EX	**WB**	

  The pipeline presented, however, writes to the register file only during the Write Back stage, does not allow writing in other phases and does not allow writing by skipping the previous instruction

- **RAW**: The Read After Write hazards are the most common. They occur when the y instruction reads the value before it was written by the previous x, thus taking a wrong value in the register. In this case, the use of forwarding is useful

x	LW R1,0(R2)	IF	ID	EX	MEM	MEM2	**WB**
y	SUB R4,R1,R5		IF	**ID**	EX	WB	

In this section we will cover the dependencies that require a stall (hence a nope operation). For example the code:
LW R1,0(R2)
SUB R4,R1,R5
The load instruction does not have the correct value of R1 when the subtraction is performed so it is useless the forwarding we need to stall the pipeline for 1 clock cycle to allow the load to have the value of r1 in time for the decode phase of the subtraction. We have divided the Hazard control in three blocks:

- **Register Destination selector**: It selects the correct destination of the instruction based on the instruction opcode
- **Data Hazard1**: It checks the relationship between destination register and source register and depending on the instructions and the relationship defines three signals for alu forwarding, memory forwarding and a signal called nop add which warns the cu to add a nop in the instruction to avoid a result of the operation wrong
- **Data Hazard2**: Same checks but for the register destination and register source 2.

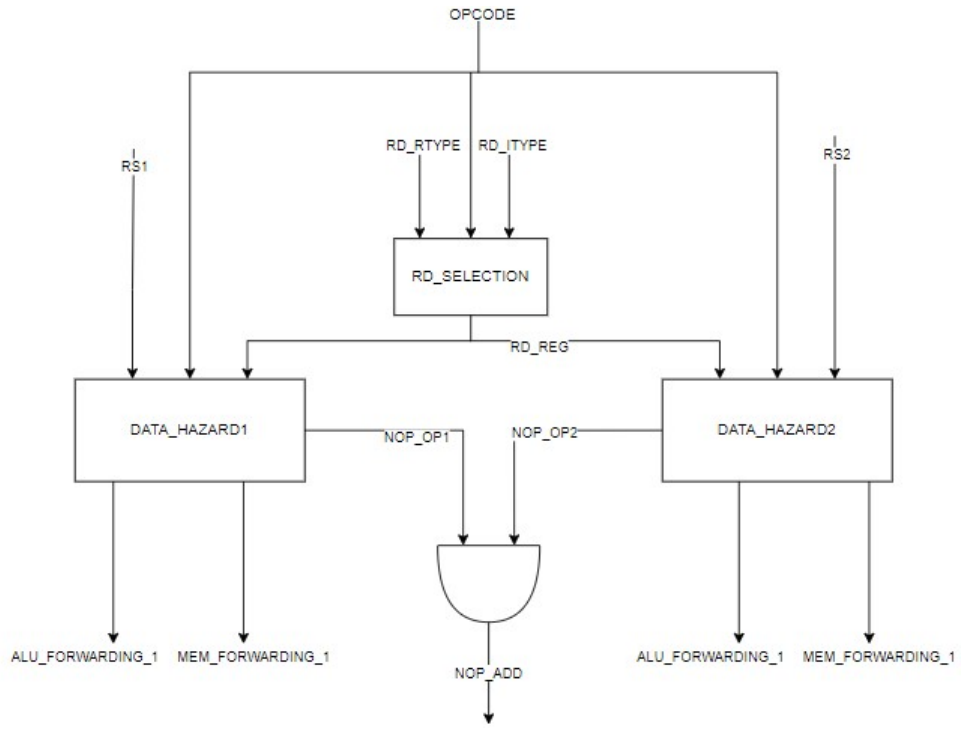

Figure 3.12: Structure of the data hazard unit

3.4 Execution Unit

The Execution Unit (EXE) performs the R-type or I-type operations supported by the Instruction Set, it receives inputs coming from the previous stage (ID) and using Multiplexers driven by the Control Unit send those inputs to the Arithmetic Logic Unit. It is composed by:

- **ALU**, which contains all the units able to perform all the operations of the Instruction Set.
- **Multiplexers**, used to send different data at the inputs of the ALU. They are also used to select data forwarded by the Memory Unit and the Write Back Unit in presence of data Hazards.
- **Pipeline registers**:
  - REG1, stores the output of the ALU.
  - REG2, stores the delayed content of the second output of the Register File, it is used as input data when a Store instruction must be performed.
  - REG3, stores the delayed content of the destination register R_D.

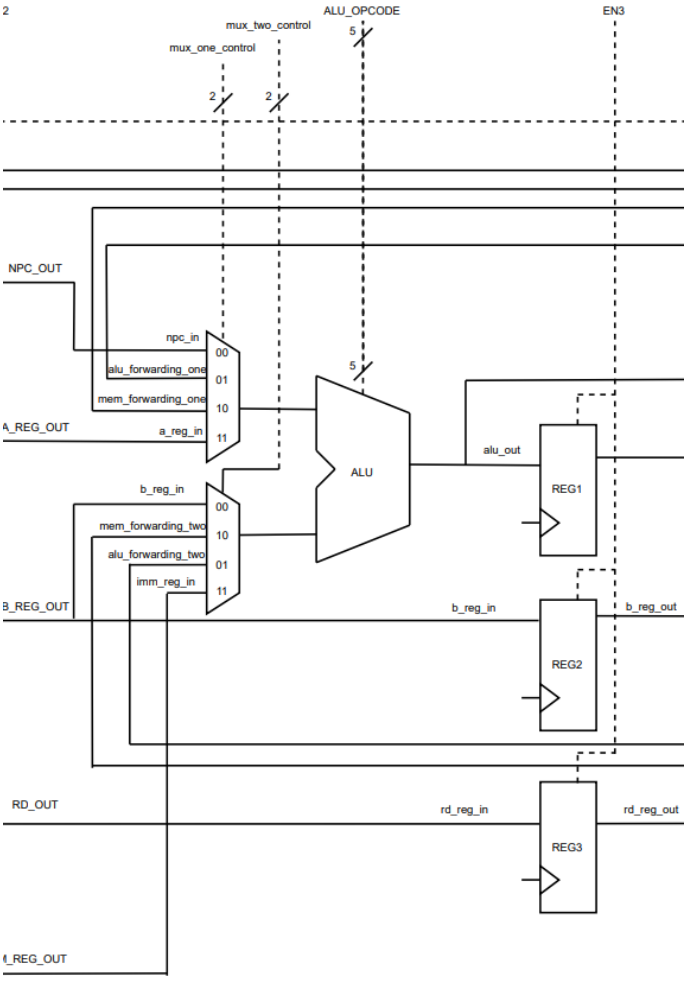

Figure 3.13: Execution Unit schematic

3.4.1 ALU

The ALU contains all the computational units used to implement the DLX supported instructions. The available units are:

- Addition and Subtraction unit
- Logic unit
- Shift unit
- Comparison unit
- Muliplication unit

The ALU is organised in such a way that the incoming operands are concurrently delivered to every computational units. Once the instruction is completed, a multiplexer delivers the result of the operation as the ALU output. In order to reduce the power consumption of this solution, the operands are copied internally in each unit. Every time the operands are changed, only the unit responsible for the current instruction will actually see the updated operand copies. This approach helps to reduce the switching activity and therefore is able to save some dynamic power.

Adder-Substractor The DLX addition and subtraction are executed recurring to the architecture used in the Pentium 4 processor. Its top level architecture is shwon in figure 3.14. The presence of a generic 32-bit XOR gate is used to handle subtraction operations. In particular, the second operand of the operation is XOR-ed along with the *carry in* signal which defines whether an addition (if '0') or a subtraction (if '1') is to be performed. The unit is based on two separate elements: the carry generator and the sum generator. The first, shown in figure 3.16, is based on a sparse tree structure and computes the carry to be used for the sum generation. Each block of the sum generator is based on two ripple carry adders that compute the sum related to a future carry equal to '0' and '1'. Its architecture is shown in figure 3.15. On the basis of the actual carry value, the correct sum is produced. The P4 adder architecture provides better performances in terms of a reduced propagation delay with respect to a standard adder/subtractor unit.

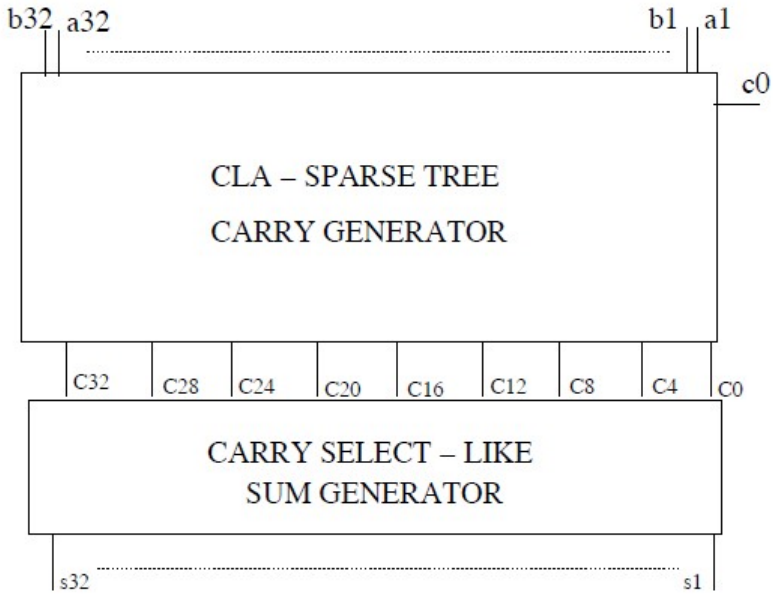

Figure 3.14: P4 adder top level architecture

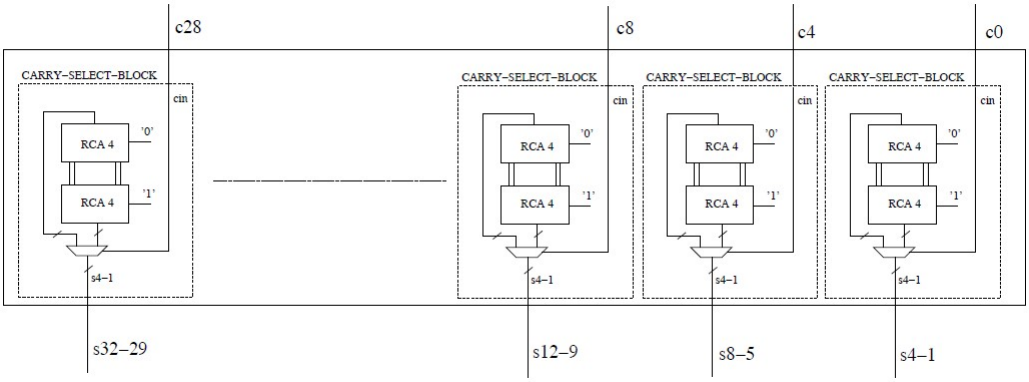

Figure 3.15: Sum generator architecture

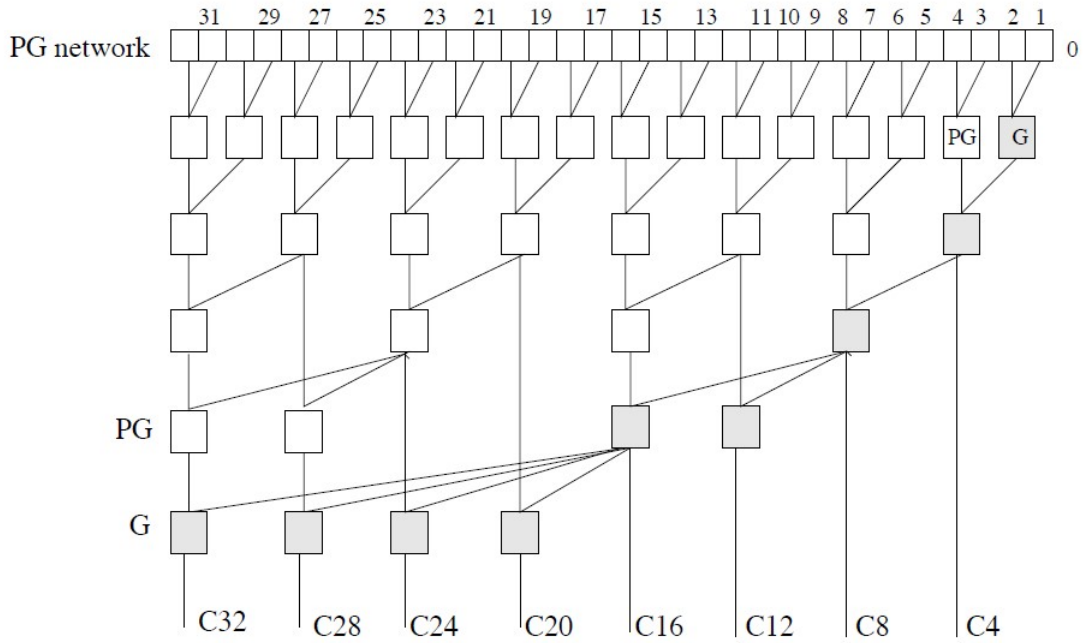

Figure 3.16: Carry generator architecture

Logic unit The DLX logic unit is based on the one available in the UltraSPARC T2. It supports six bitwise logical operations between two operands: AND, NAND, OR, NOR, XOR, XNOR. Its architecture is based on two NAND gates levels: four 3-inputs each NAND gates for the first level and a single 4-input NAND gate for the second level whose inputs are the outputs of the first level. NAND3 and NAND4 gates are described recurring to a structural implementation by connecting basic inverters and NAND2 gates. The complete architecture is shown in figure 3.17

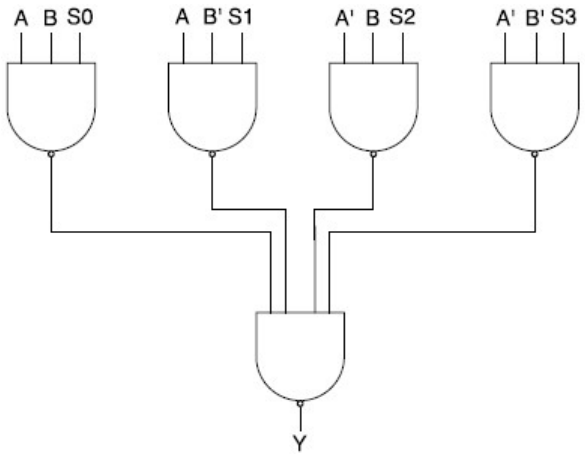

Figure 3.17: T2 logical unit

Shift unit The DLX shifter is based on the one available in the UltraSPARC T2. It supports multiple shift operations: logic left, logic right, arithmetic right and their immediate versions. Its architecture is based on three levels. The first level prepares multiple possible masks, each already shifted of 0,8,16.. left or right. The second level performs a coarse grain shift, which is about choosing the nearest mask among the available ones. The choice is operated using bit 5,4,3 of operand B. The

third level receives the selected mask and implements the fine grain shift according to bits 2,1,0 of operand B. A signal *sel* is used to select the type of shift operations.

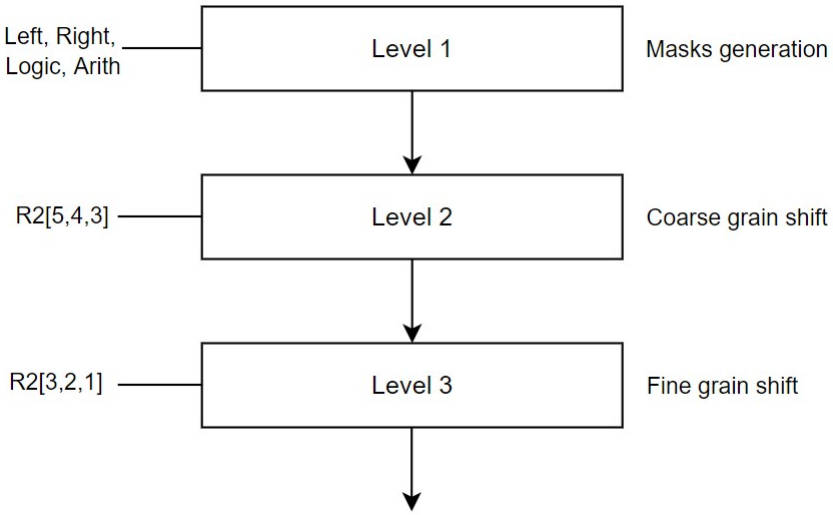

Figure 3.18: T2 shift unit

Comparator unit The DLX comparator is based on a structural architecture made of an adder/-substractor and a set of logic gates: 2-inputs OR, 3-inputs NAND, 2-inputs AND and a INV. It supports every possible comparison operations: $>, <, \geq, \leq, =, \neq$. The sign input is XOR-ed together with the *cout* signal to compute the correct comparison even if the two operands have a different sign. The sum between the two operands is computed by using the P4 adder. The complete architecture is shown in figure 3.19

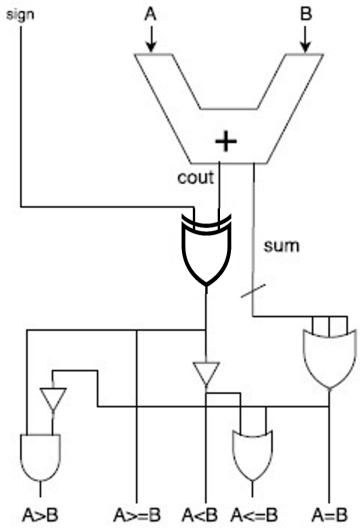

Figure 3.19: Comparator

Multiplier The DLX multiplier architecture is shown in figure 3.20. It is based on the Booth's algorithm which reduces the number of partial products with respect to a multiplication with an

array structure. Its operands are on 16 bits and the result is on 32 bits.

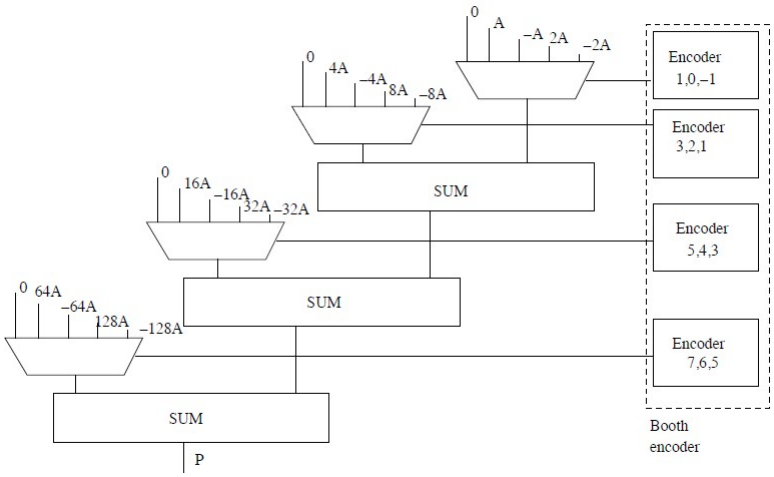

Figure 3.20: Booth multiplier architecture

3.5 Memory Unit

The Memory Unit (MEM) is mainly used in presence of Load and Store instructions where a DRAM access is expected, the DRAM is not present inside the MEM Unit, but it is connected with it from the outside through the output of the two pipeline registers from the previous EXE Unit. One output of the MEM Unit is the 32-bit DRAM address and the other output is the 32-bit DRAM input data; on the other side there is the data_out coming from the DRAM connected to a MUX inside the MEM Unit. Inside the MEM Unit the following elements are present:

- **MUX_MEM**, it is used to select which data has to proceed in the pipeline execution, either the ALU output or the DRAM output.
- **Pipeline registers**:
  - REG_ALU, contains the data that may be written into the Register File.
  - RD_REG, stores the delayed address of the Destination Register that may be used to write inside the Register File.

3.6 WriteBack Unit

The Write Back Unit (WB) is where the Register File is written in order to store the result of the ALU operations, or the value loaded by the DRAM. From this Unit a connection is made with the Write input and the Write address of the register file.

3.7 Control Unit

The control unit is developed using the Hardwired technique. It is based on two different look-up-tables, the first one is used when an R-type instruction is detected, the other one is used when the instruction to decode is an I-type or a Jump/Branch instruction.

Instructions are expressed on a 32-bit word. In order to recognise the different instructions, the six Most Significant Bits are read, they represent (OPCODE). With the OPCODE it is possible to recognize three different type of instruction:

- R-type instructions.
- I-type instructions.
- J-type instructions.

By looking at the OPCODE value we can distinguish between those type of instructions:

- If the OPCODE is made of all zeros it is an R-type instruction. In this case also the 11 Least Significant Bits are read; they represent the FUNC field and are used to detect the required operation. Figure 3.17.

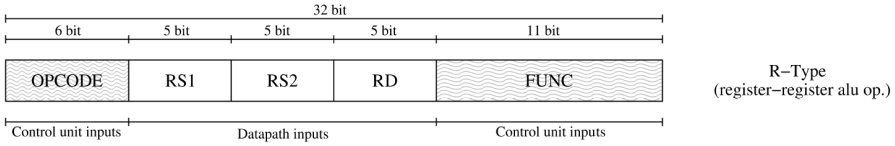

Figure 3.21: R-type instruction fields

- If the OPCODE is different from zero, it can be an I-type or a J-type instruction and the operation to be performed is detected directly from the OPCODE field. Figure 3.18 and Figure 3.19.

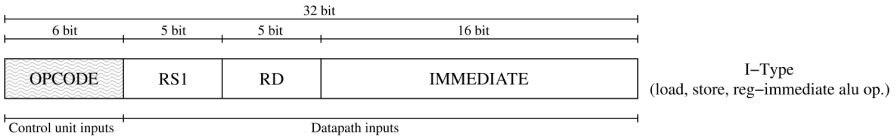

Figure 3.22: I-type instruction fields

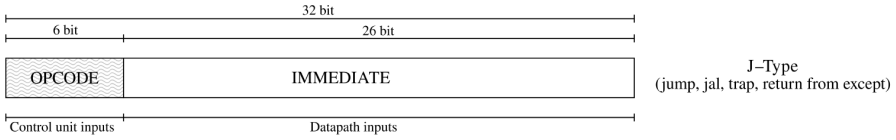

Figure 3.23: J-type instruction fields

Look-up-tables are addressed by the FUNC in case of an R-type instruction and by the OPCODE in case of I-type or J-type instructions. When we address the look-up-table we obtain a Control word that is used to drive all the input signals of the different blocks of our system which are: Datapath, IRAM and DRAM. In Figure 3.20 a representation of the Hardwired Control Unit is shown:

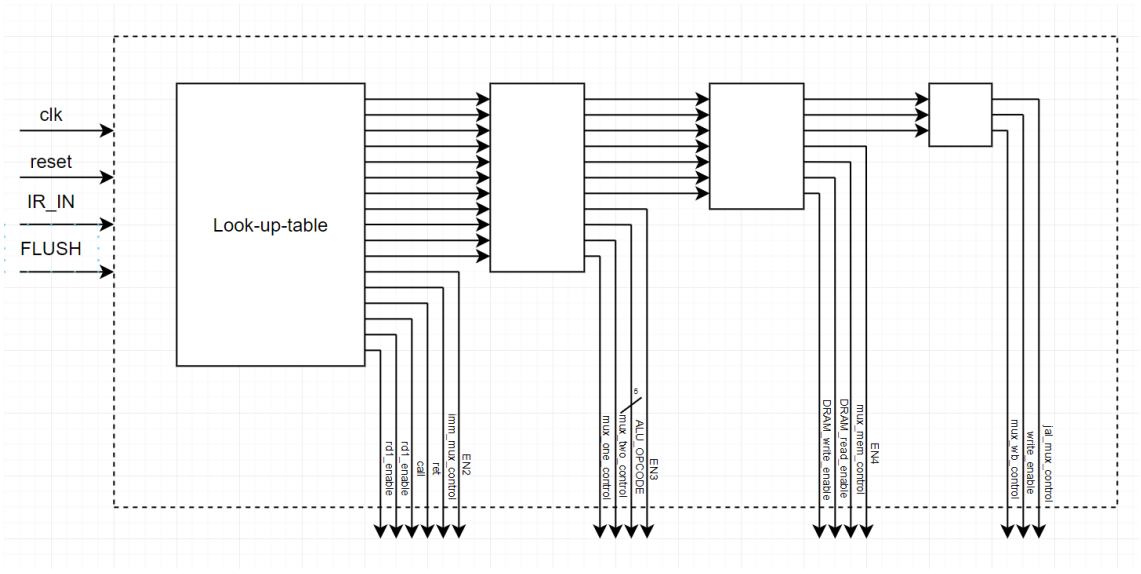

Figure 3.24: Hardwired Control Unit

CHAPTER 4

Simulation and testing

The DLX has been tested according to a bottom-up approach exploiting testbenches to be used with *QuestaSim* by Siemens. Starting from each low level entity, the testing moves to already tested higher and higher level entities and continues until the top level entity is reached. The DLX top level entity is tested along with memories. In particular, the instruction memory is loaded with a *mem* file created from a suitable *asm* file using the assembler script.

4.1 Script

In order to automate the simulation process of the DLX top level entity, the *simulation.tcl* script has been used. The script must be placed and run in the main source file folder. It contains two procedures.

compile_directory The procedure analyzes recursively in reverse alphanumerical order every file in every sub-directory of the main directory specified as input. It must be executed prior any simulation. It receives as inputs:

- the name of the main directory containing the files to be analyzed (the default value is ./).
- the extension of the files to be analyzed, vhdl or verilog (the default value is vhdl).

simulate_dlx The procedure performs a simulation of the entire dlx using a custom .asm file. It converts the incoming assembly test file into the correspondent .mem file.
It receives as inputs:

- the name of the .asm file to be converted in .mem file by the dlxasm (the default value is test.asm).
- the duration time of the simulation (the default value is 100 ns).

CHAPTER 5

Synthesis

The synthesis has been carried out by using *Design Compiler* by Synopsys.

5.1 Script

In order to automate the synthesis process, the *synthesis.tcl* script has been used. The script must be placed and run in the main source file folder.
It contains three tcl procedure.

analyze_directory The procedure analyzes recursively in reverse alphanumerical order every file in every sub-directory of the main directory specified as input. it must be executed prior any synthesis. It receives as inputs:

- the name of the main directory containing the files to be analyzed (the default value is ./).
- the extension of the files to be analyzed, vhdl or verilog (the default value is vhdl).

synthesize The procedure performs a basic synthesis with clock gating using the *NandgateOpenCellLibrary* (45 nm) as technology library.
It receives as inputs:

- the name of the top level entity to be synthesized (the default value is dlx).
- the maximum delay in nanoseconds between inputs and outputs (the default value is 2 ns).
- the name of the wire load model (the default value is 5K_hvratio_1_4).

It produces as outputs:

- the post-synthesis netlist.
- the sdc file.
- the reports for timing, power, area and clock gating.

synthesize_dual_vth The procedure performs a double voltage threshold synthesis with clock gating using the *STcmos65* as technology library.
It receives as inputs:

- the name of the top level entity to be synthesized (the default value is dlx).
- the maximum delay in nanoseconds between inputs and outputs (the default value is 2 ns).
- the names of the low and high threshold voltage libraries (the default values are CORE65LPLVT and CORE65LPHVT).

It produces as outputs:

- the post-synthesis netlist.
- the sdc file.
- the reports for timing, power, area, threshold voltage group and clock gating.

5.2 Reports

5.2.1 NandgateOpenCellLibrary

Timing

```
Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : dlx
Version: F-2011.09-SP3
Date   : Fri Oct 21 21:19:53 2022
****************************************

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

  Startpoint: DTP/PC/d_out_reg[6]
              (rising edge-triggered flip-flop clocked by Clk)
  Endpoint: DTP/PC/R_235_PC_d_out_reg[31]
            (rising edge-triggered flip-flop clocked by Clk)
  Path Group: Clk
  Path Type: max

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

  Point                                                   Incr       Path
  ------------------------------------------------------------------------
```

```
clock Clk (rise edge)                                  0.00       0.00
clock network delay (ideal)                            0.00       0.00
DTP/PC/d_out_reg[6]/CK (DFF_X1)                        0.00       0.00 r
DTP/PC/d_out_reg[6]/QN (DFF_X1)                        0.07       0.07 f
U10513/ZN (NOR2_X1)                                    0.06       0.13 r
U10512/ZN (INV_X1)                                     0.03       0.16 f
U10511/ZN (NOR2_X1)                                    0.06       0.22 r
U10510/ZN (AND2_X1)                                    0.06       0.28 r
U10509/ZN (INV_X1)                                     0.03       0.31 f
U10508/ZN (NOR2_X1)                                    0.06       0.37 r
U10507/ZN (AND2_X1)                                    0.06       0.43 r
U10506/ZN (AND2_X1)                                    0.05       0.49 r
U10505/ZN (INV_X1)                                     0.03       0.51 f
U10504/ZN (NOR2_X1)                                    0.05       0.57 r
U10503/ZN (INV_X1)                                     0.03       0.60 f
U10502/ZN (NOR2_X1)                                    0.06       0.66 r
U10501/ZN (AND2_X1)                                    0.06       0.72 r
U10500/ZN (INV_X1)                                     0.03       0.75 f
U10499/ZN (NOR2_X1)                                    0.06       0.81 r
U10498/ZN (AND2_X1)                                    0.06       0.87 r
U10497/ZN (INV_X1)                                     0.03       0.90 f
U10496/ZN (NOR2_X1)                                    0.05       0.95 r
U10495/ZN (INV_X1)                                     0.03       0.98 f
U10494/ZN (NOR2_X1)                                    0.06       1.04 r
U10493/ZN (AND2_X1)                                    0.06       1.10 r
U10492/ZN (INV_X1)                                     0.03       1.13 f
U10491/ZN (NOR2_X1)                                    0.05       1.18 r
U10490/ZN (INV_X1)                                     0.03       1.21 f
U10489/ZN (NOR2_X1)                                    0.05       1.26 r
U10488/ZN (INV_X1)                                     0.03       1.29 f
U10487/ZN (OR2_X1)                                     0.07       1.36 f
U10422/ZN (NOR3_X1)                                    0.07       1.43 r
U10486/ZN (INV_X1)                                     0.03       1.47 f
U10485/ZN (NOR2_X1)                                    0.05       1.52 r
U10484/ZN (INV_X1)                                     0.03       1.55 f
U10483/ZN (NOR2_X1)                                    0.06       1.61 r
U10651/ZN (AOI21_X1)                                   0.04       1.65 f
U10553/ZN (NOR3_X1)                                    0.09       1.74 r
U10842/ZN (NAND2_X1)                                   0.05       1.79 f
U10854/ZN (NOR3_X1)                                    0.09       1.88 r
U10857/ZN (OAI21_X1)                                   0.04       1.91 f
DTP/PC/R_235_PC_d_out_reg[31]/D (DFF_X1)               0.01       1.93 f
data arrival time                                                 1.93

clock Clk (rise edge)                                  2.00       2.00
clock network delay (ideal)                            0.00       2.00
DTP/PC/R_235_PC_d_out_reg[31]/CK (DFF_X1)              0.00       2.00 r
library setup time                                    -0.05       1.95
data required time                                                1.95
```

```
  --------------------------------------------------------------------------
  data required time                                                  1.95
  data arrival time                                                  -1.93
  --------------------------------------------------------------------------
  slack (MET)                                                         0.03
```

Power

```
Loading db file '/home/mariagrazia.graziano/do/libnangate/NangateOpenCellLibrary_typical_ecsm.db'
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   : Fri Oct 21 21:19:54 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  =   4.9849 mW   (77%)
  Net Switching Power  =   1.5203 mW   (23%)
                         ---------
Total Dynamic Power    =   6.5052 mW  (100%)
```

```
Cell Leakage Power     = 283.5328 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    155.1169         960.7856        3.0788e+03         1.1190e+03  (  16.48%)
register       4.5642e+03          84.6209        1.1676e+05         4.7655e+03  (  70.20%)
sequential        17.2175           6.7706        2.6151e+03            26.6032  (   0.39%)
combinational    248.4192         468.1527        1.6108e+05           877.6533  (  12.93%)
--------------------------------------------------------------------------------------------------
Total          4.9849e+03 uW     1.5203e+03 uW     2.8353e+05 nW     6.7888e+03 uW
```

Area

```
****************************************
Report : area
Design : dlx
Version: F-2011.09-SP3
Date   : Fri Oct 21 21:19:54 2022
****************************************

Library(s) Used:

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

Number of ports:                          66
Number of nets:                          860
Number of cells:                         770
Number of combinational cells:           556
Number of sequential cells:              212
Number of macros:                          0
Number of buf/inv:                        74
Number of references:                     31

Combinational area:        583.870001
Noncombinational area:    1000.159977
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:          1584.029978
Total area:                 undefined
```

Clock Gating

```
****************************************
Report : clock_gating
```

```
Design : dlx
Version: F-2011.09-SP3
Date   : Fri Oct 21 21:19:54 2022
****************************************
```

```
                    Clock Gating Summary
  ------------------------------------------------------------
  |    Number of Clock gating elements    |        2         |
  |                                       |                  |
  |    Number of Gated registers          |     72 (33.96%)  |
  |                                       |                  |
  |    Number of Ungated registers        |     140 (66.04%) |
  |                                       |                  |
  |    Total number of registers          |        212       |
  ------------------------------------------------------------
```

5.2.2 STcmos65

Timing

```
Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : dlx
Version: F-2011.09-SP3
Date   : Fri Oct 21 21:38:02 2022
****************************************

Operating Conditions: nom_1.20V_25C   Library: CORE65LPLVT
Wire Load Model Mode: enclosed

  Startpoint: DTP/cnt/i_reg[1]
              (rising edge-triggered flip-flop clocked by Clk)
  Endpoint: DTP/cnt/tc_reg
            (rising edge-triggered flip-flop clocked by Clk)
  Path Group: Clk
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  dlx                area_3Kto4K           CORE65LPLVT

  Point                                                   Incr       Path
  ------------------------------------------------------------------------
```

```
  clock Clk (rise edge)                                   0.00       0.00
  clock network delay (ideal)                             0.00       0.00
  DTP/cnt/i_reg[1]/CP (HS65_LH_DFPRQNX4)                  0.00       0.00 r
  DTP/cnt/i_reg[1]/QN (HS65_LH_DFPRQNX4)                  0.20       0.20 f
  sub_x_338/U85/Z (HS65_LL_OR2X9)                         0.07       0.26 f
  U10125/Z (HS65_LH_OR3X4)                                0.12       0.38 f
  U10100/Z (HS65_LL_NAND4ABX3)                            0.10       0.48 f
  sub_x_338/U45/Z (HS65_LL_OR2X9)                         0.07       0.54 f
  sub_x_338/U43/Z (HS65_LL_OR2X9)                         0.05       0.59 f
  sub_x_338/U41/Z (HS65_LL_OR2X9)                         0.05       0.64 f
  sub_x_338/U39/Z (HS65_LL_OR2X9)                         0.05       0.69 f
  sub_x_338/U37/Z (HS65_LL_OR2X9)                         0.05       0.73 f
  sub_x_338/U35/Z (HS65_LL_OR2X9)                         0.05       0.78 f
  sub_x_338/U33/Z (HS65_LL_OR2X9)                         0.05       0.82 f
  sub_x_338/U31/Z (HS65_LL_OR2X9)                         0.05       0.87 f
  sub_x_338/U29/Z (HS65_LL_OR2X9)                         0.05       0.92 f
  sub_x_338/U27/Z (HS65_LL_OR2X9)                         0.05       0.96 f
  sub_x_338/U25/Z (HS65_LL_OR2X9)                         0.05       1.01 f
  sub_x_338/U23/Z (HS65_LL_OR2X9)                         0.05       1.06 f
  sub_x_338/U21/Z (HS65_LL_OR2X9)                         0.05       1.10 f
  sub_x_338/U19/Z (HS65_LL_OR2X9)                         0.05       1.15 f
  sub_x_338/U17/Z (HS65_LL_OR2X9)                         0.05       1.20 f
  sub_x_338/U15/Z (HS65_LL_OR2X9)                         0.05       1.24 f
  sub_x_338/U13/Z (HS65_LL_OR2X9)                         0.05       1.29 f
  sub_x_338/U11/Z (HS65_LL_OR2X9)                         0.05       1.34 f
  sub_x_338/U9/Z (HS65_LL_OR2X9)                          0.05       1.38 f
  sub_x_338/U7/Z (HS65_LL_OR2X9)                          0.05       1.43 f
  sub_x_338/U5/Z (HS65_LL_OR2X9)                          0.05       1.48 f
  U10133/Z (HS65_LH_OAI12X2)                              0.07       1.55 r
  U10134/Z (HS65_LH_OAI13X1)                              0.13       1.68 f
  U10123/Z (HS65_LH_NOR3X1)                               0.12       1.80 r
  U10124/Z (HS65_LH_NOR2X2)                               0.07       1.87 f
  DTP/cnt/tc_reg/D (HS65_LL_DFPSQX4)                      0.00       1.87 f
  data arrival time                                                  1.87

  clock Clk (rise edge)                                   2.00       2.00
  clock network delay (ideal)                             0.00       2.00
  DTP/cnt/tc_reg/CP (HS65_LL_DFPSQX4)                     0.00       2.00 r
  library setup time                                     -0.09       1.91
  data required time                                                 1.91
  -----------------------------------------------------------------------
  data required time                                                 1.91
  data arrival time                                                 -1.87
  -----------------------------------------------------------------------
  slack (MET)                                                        0.03
```

Power

```
Loading db file '/home/ms22.21/FINAL_SYNTH/dual/tech/STcmos65/CORE65LPLVT_nom_1.20V_25C.db'
Loading db file '/home/ms22.21/FINAL_SYNTH/dual/tech/STcmos65/CORE65LPHVT_nom_1.20V_25C.db'
```

```
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   : Fri Oct 21 21:38:09 2022
****************************************


Library(s) Used:

    CORE65LPLVT (File: /home/ms22.21/FINAL_SYNTH/dual/tech/STcmos65/CORE65LPLVT_nom_1.20V_25C.db)
    CORE65LPHVT (File: /home/ms22.21/FINAL_SYNTH/dual/tech/STcmos65/CORE65LPHVT_nom_1.20V_25C.db)


Operating Conditions: nom_1.20V_25C   Library: CORE65LPLVT
Wire Load Model Mode: enclosed

Design        Wire Load Model            Library
------------------------------------------------
dlx                     area_3Kto4K       CORE65LPLVT
SNPS_CLOCK_GATE_HIGH_dlx_1
                        area_0Kto1K       CORE65LPLVT
SNPS_CLOCK_GATE_HIGH_dlx_2
                        area_0Kto1K       CORE65LPLVT


Global Operating Voltage = 1.2
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1mW


  Cell Internal Power  =   5.1563 mW   (76%)
  Net Switching Power  =   1.5879 mW   (24%)
                         ---------
Total Dynamic Power    =   6.7442 mW  (100%)

Cell Leakage Power     =   1.4821 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      0.1272           0.8976         1.1330e-06              1.0248  (  15.19%)
register           4.6665        7.1544e-02         2.6439e-04              4.7383  (  70.24%)
sequential     1.5939e-02        5.9238e-03         1.3497e-06          2.1864e-02  (   0.32%)
combinational      0.3466           0.6129         1.2152e-03              0.9607  (  14.24%)
--------------------------------------------------------------------------------------------------
Total              5.1563 mW        1.5879 mW      1.4821e-03 mW          6.7457 mW
```

Area

```
****************************************
Report : area
Design : dlx
Version: F-2011.09-SP3
Date   : Fri Oct 21 21:38:09 2022
****************************************

Library(s) Used:

    CORE65LPLVT (File: /home/ms22.21/FINAL_SYNTH/dual/tech/STcmos65/CORE65LPLVT_nom_1.20V_25C.db)
    CORE65LPHVT (File: /home/ms22.21/FINAL_SYNTH/dual/tech/STcmos65/CORE65LPHVT_nom_1.20V_25C.db)

Number of ports:                           66
Number of nets:                           930
Number of cells:                          859
Number of combinational cells:            655
Number of sequential cells:               202
Number of macros:                           0
Number of buf/inv:                        105
Number of references:                      96

Combinational area:       1935.439954
Noncombinational area:    1690.000005
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:          3625.439959
Total area:                 undefined
```

Threshold Voltage Group

```
********************************************************************************
                        Threshold Voltage Group Report

Vth Group                 All               Blackbox           Non-blackbox
Name                    cells                 cells                 cells
********************************************************************************
```

```
HVT                    66   (7.67%)          0   (0.00%)         66   (7.67%)
LVT                   795  (92.33%)          0   (0.00%)        795  (92.33%)
--------------------------------------------------------------------------------
Total                 861 (100.00%)          0   (0.00%)        861 (100.00%)

Vth Group                All                Blackbox           Non-blackbox
Name                  cell area            cell area            cell area
--------------------------------------------------------------------------------
HVT                 208.00   (5.74%)       0.00   (0.00%)     208.00   (5.74%)
LVT                3417.44  (94.26%)       0.00   (0.00%)    3417.44  (94.26%)
--------------------------------------------------------------------------------
Total              3625.44 (100.00%)       0.00   (0.00%)    3625.44 (100.00%)
```

Clock Gating

```
****************************************
Report : clock_gating
Design : dlx
Version: F-2011.09-SP3
Date   : Fri Oct 21 21:38:09 2022
****************************************
```

Clock Gating Summary

Number of Clock gating elements	2
Number of Gated registers	63 (31.19%)
Number of Ungated registers	139 (68.81%)
Total number of registers	202

5.2.3 Standard VS Dual V-th synthesis comparisons

Synthesis	Library	Clock period	Total power	Total area
Standard	NandGateOpenCellLibrary (45 nm)	2 ns	6.5 mW	1584 μm^2
Dual V-th	STcmos65 (65 nm)	2 ns	6.75 mW	3625 μm^2

A meaningful comparison between the presented reports is difficult since different libraries have been used. In any case a slight improvement in power consumption at the expense of a larger cell area can be appreciate despite the STcmos65 being an older technology. In both case, a 2 ns clock is compatible with the two timing requirements leaving a small margin to account for uncertainties and process variations.

CHAPTER 6

Physical design

The physical design has been carried out by using *Innovus 17.11* by Cadence starting from the post-synthesis netlist and sdc file generated from the synthesis. In the following Figure is possible to observe an overview of the place and route.

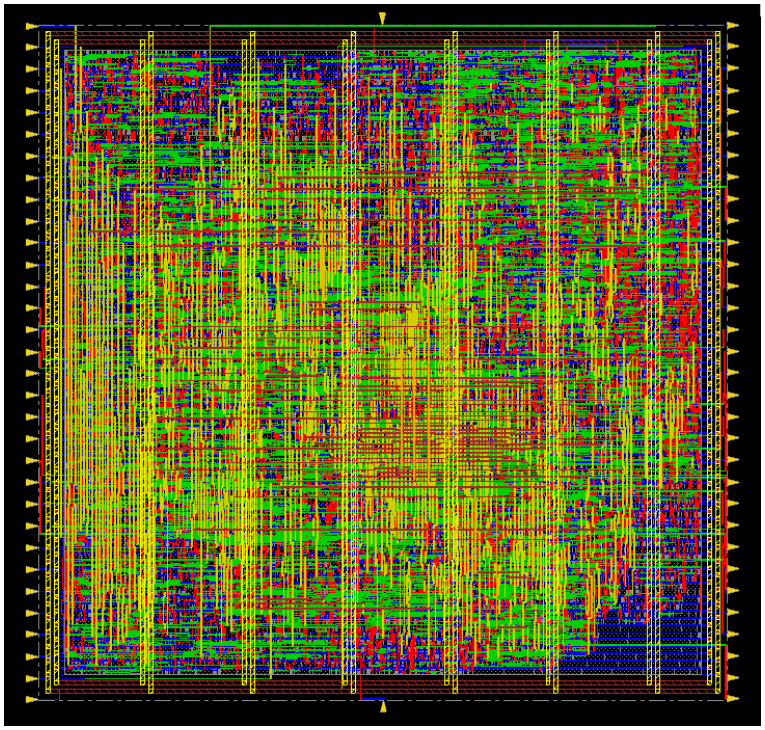

Figure 6.1: Physical design

After this operation is possible to observe informations about Timing and Geometry, in our case there were no violations. The process is divided in multiple steps:

- **Configuration** The post-synthesis DLX netlist and the layout view of the library of cells are imported.
- **Floorplanning** The area to be used for the cells and power supply rings are set.
- **Power Rings insertion** The metal rings for power (VDD) and ground (GND) are inserted.

- **Stripes insertion** The vertical VDD and GND stripes are introduced in order to complete the hierarchical signal distribution.
- **Routing** The horizontal connections between standard cells and vertical striped are realised.
- **Placement** The standard cells are placed along the available rows. I/O pins are introduced along the edges of the DIE.
- **Post clock-tree-synthesis** The design is optimised in order to achieve the required timing constraints.
- **Routing** The connections among the standard cells is performed by means of the available metal layers.
- **Timing and integrity analysis** The reports about timing, parasitics and power are extracted.
- **Design verification** The final design is verified for any design rule violations to ensure its correct behaviour prior to its actual realisation by the foundry.

CHAPTER 7

Conclusions

Further improvements
A list of possible additions or modifications to the DLXY may include:

- advanced branch prediction techniques
- interrupts and exceptions handling
- division (introducing multi-cycle operations)
- caches hierarchy

The DLX has been tested with different testbenches, in order to check the right behaviour of all the instructions. Instructions like jump, branch, arithmetical instructions, bitwise and shift operations and memory instructions have been tested. Everything has been developed in a modular approach in a way that further improvements can be done easily.

Bibliography

[1] Hennessy, J. L. and Patterson, D. A. *Computer Architecture: A Quantitative Approach.*, Morgan Kaufmann Publishers, Inc. San Mateo, CA, Fifth Edition, 2011.

[2] Graziano, M. *Microelectronic Systems Lecture Notes*, 2022.