Politecnico di Torino

Microelectronic Systems

DLX Microprocessor: Design & Development

Final Project Report

Master degree in Computer Engineering

Master degree in Electronics Engineering

Referents: Prof. Mariagrazia Graziano, Giovanna Turvani

Authors: group_16

Battilana Matteo, La Greca Salvatore Gabriele, Pollo Giovanni

July 1, 2021

Feature

- Frequency - Slack - Area - Ecc

Grandes nacelles :

- Nacelle A318 PW
- Inverseur A320 CFM
- Inverseur A340 CFM
- Nacelle A340 TRENT
- Inverseur A330 TRENT
- Nacelles A380 TRENT900
- Nacelles A380 GP7200

Petites nacelles :

- Nacelle SAAB2000
- Inverseur DC8
- Inverseur CF34-8
- Inverseur BR710
- Nacelle F7X

Contents

Feature **i**

1 Introduction **1**
1.1 Abstract . . . 1
1.2 Workflow . . . 1

2 Hardware Architecture **2**
2.1 Overview . . . 2
2.2 Pipeline Stages . . . 3
2.3 Control Unit . . . 3
2.4 Memory Interface . . . 3
2.4.1 Signals and Timing . . . 3
2.4.2 Memory Addressing . . . 4
2.4.3 Memory Data Size: the MAS[1:0] signal . . . 4
2.5 Instruction Set . . . 4

3 Fetch Stage **5**
3.1 Instruction Register . . . 5
3.2 Program Counter . . . 5
3.3 Jump and Branch Management . . . 5

4 Decode Stage **6**
4.1 Instruction Decode . . . 6
4.2 Register File and Windowing . . . 6
4.3 Hazard Control . . . 6
4.4 Comparator . . . 6
4.5 Jump and Branch decision . . . 6
4.6 Next Program Counter computation . . . 6

5 Execute Stage **7**
5.1 ALU: Arithmetic Logic Unit . . . 7
5.1.1 Adder . . . 7
5.1.2 Multiplier . . . 7
5.1.3 Logic Operands . . . 7
5.1.4 Shifting . . . 8
5.2 Set-Like Operations unit . . . 8

6 Memory Stage **9**
6.1 Load-Store Unit 9
6.2 Address Mask Unit 9

7 Write Back Stage **10**

8 Testing and Verification **11**
8.1 Test Benches 11
8.2 Simulation 11
8.3 Post Synthesis Simulation 11

9 Physical Design **12**
9.1 Synthesis 12
9.2 Place and Route 12

10 Conclusions **13**

Listings

CHAPTER 1

Introduction

1.1 Abstract

1.2 Workflow

- Workflow used - git / github / pair programming

CHAPTER 2

Hardware Architecture

2.1 Overview

This DLX is a 32-bit RISC processor with a five-stage pipeline. The external interface is made mainly for memories connection (IRAM, DRAM and a DRAM for the Register File), and for the Clock and Reset signals. Inside we find the following blocks:

- Control Unit: it receives the fetched instruction from the IR register and starts to output the correct control signals towards all the pipeline stages. Moreover, it receives status signals from all other units about their working status like the comparator result (for branch decision), the status about possible hazards in the pipeline, Register File's Push & Pop operations under execution, and all the memories readiness. It's in charge of controling the entire pipeline and stop it in case of hazards or other situations that requires a stall.
- Decode Unit: part of the decode stage, it is in charge of keeping the status about all registers under use (for further hazard controls), computation of the new Program Counter (given a Jump or not), data comparison (for branches) and, the most important thing, the operation decode with the dispatch of all the operands towards the right ports of the DataPath.
- DataPath: the computational core of the processor. Made of 4 pipeline stages (Instruction Decode, Execution, Memory, Write Back) contains all the units capable of doing computation. In particular, we have the Register File (that manages all the registers of the core), the Arithmetic Logic Unit, the Load-Store Unit for data memory management, and other units useful for the correct operation of everything.
- IR and PC: two registers the compose the Instruction Fetch stage of the pipeline, they are in charge of keeping in memory the current instruction under execution and the address for the next instruction to execute, respectively.

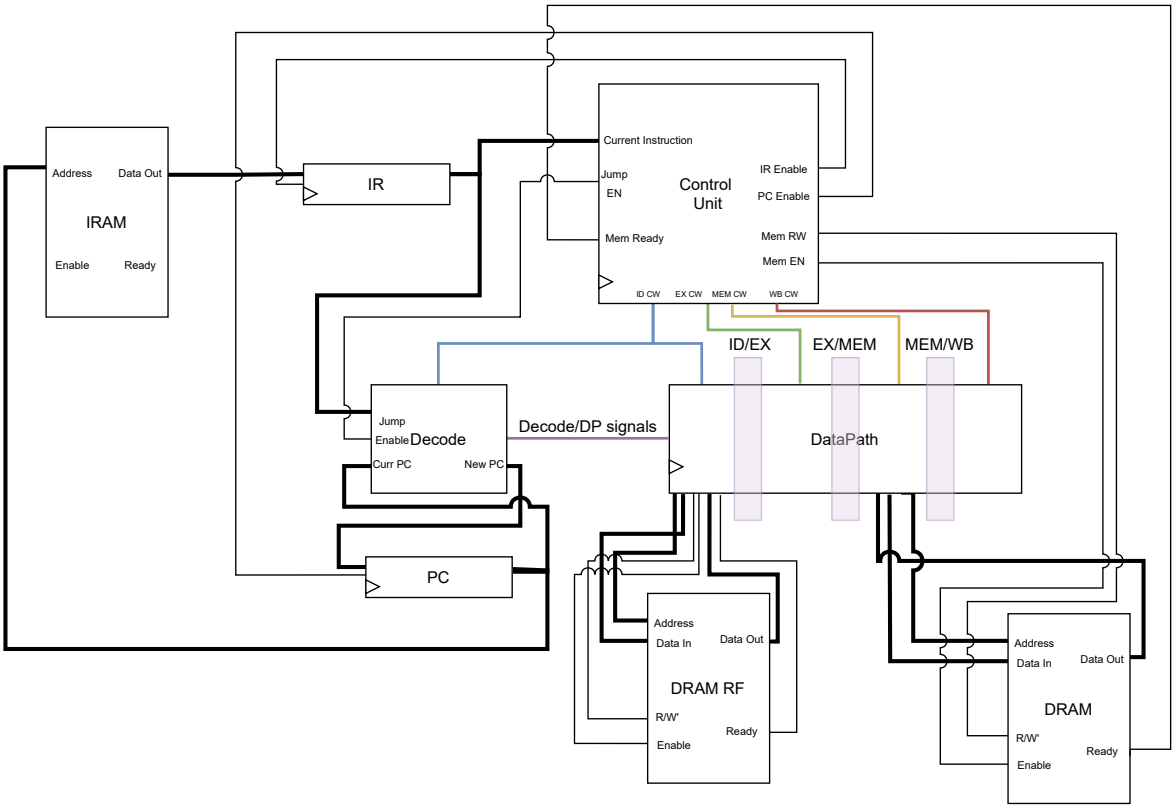


Figure 2.1: Schematic of the DLX

2.2 Pipeline Stages

2.3 Control Unit

2.4 Memory Interface

The DLX processor has a Harvard architecture, with two 32-bit data bus carrying instructions and data respectively. Only load and store instructions can access data from Data Memory. The Data are stored in memory in a Big-Endian format.

The third RAM required, the one for the register file, is not under the direct access of the User. It's managed as a Stack memory by the Register File for automatic sub routines management and it has a dedicated interface. It can be merged with the Data Memory with an external logic.

All the subsequent paragraph are referred in the same way to all the three memory types.

2.4.1 Signals and Timing

The signals in the DLX processor bus interface can be grouped into tree categories:

- Address class signals
- Memory Request signals
- Data class signals

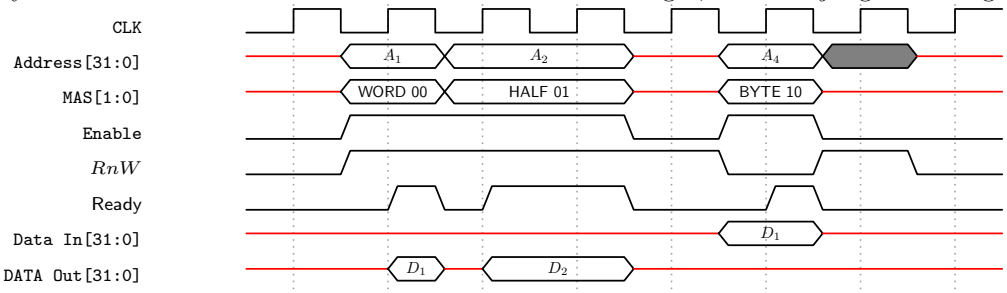
The *Address class signals* are:

- A[31:0]
- DATA_SIZE[1:0] *or MAS[1:0]*

The *Memory Request signals* are:

- Enable
- $R\overline{W}$
- Ready

Moreover, all the memories connected must agree on a certain protocol both for writing and reading operations. The most important thing to take under consideration is the *Ready* signal: it must be high only when the operation is really completed. For example after a data read, the *Ready* stays at 1 only when the data is valid. If meanwhile the address changes, the *Ready* signal must go off.



If the memory in use can't accomplish to this timing, an external *Memory Control Unit* must be placed between the CPU and the Memory.

2.4.2 Memory Addressing

A[31:0] is the 32-bit address bus that specifies the address for the transfer. All addresses are byte addresses, so a burst of word accesses results in the address bus incrementing by four for each cycle.

The address bus provides 4GB of linear addressing space, and this can be used externally in different manners like in a SoC with memory mapped peripherals that shares the same address space.

When a word access is signaled the memory system ignores the bottom two bits, A[1:0], and when a halfword access is signaled the memory system ignores the bottom bit, A[0]. Howewer, the core already masks the two LSBs when needed.

2.4.3 Memory Data Size: the MAS[1:0] signal

The *MAS[1:0]* bus encodes the size of the transfer. The DLX processor can transfer word, halfword, and byte quantities and the processor indicates the size of the transfer through this signal.

When a halfword or byte read is performed, a 32-bit memory system can return the complete 32-bit word, and the processor extracts the valid halfword or byte field from it. For 8 and 16 bit memories, the data must be placed on the right byte lanes in the data bus.

2.5 Instruction Set

CHAPTER 3

Fetch Stage

3.1 Instruction Register

3.2 Program Counter

3.3 Jump and Branch Management

CHAPTER 4

Decode Stage

4.1 Instruction Decode

4.2 Register File and Windowing

4.3 Hazard Control

4.4 Comparator

- Unsigned things

4.5 Jump and Branch decision

4.6 Next Program Counter computation

CHAPTER 5

Execute Stage

5.1 ALU: Arithmetic Logic Unit

5.1.1 Adder

5.1.2 Multiplier

5.1.3 Logic Operands

The basic and most simple implementation of a logic unit is based on single logic gates on N bits whose outputs are muxed, in order to generate the correct output. The problem with this solution is that the number of input signals to the multiplexer is extremely high; this implementation does not only suffer from the point of view of the delay but, since each logic function is implemented with a specific gate, the total area is huge.

In order to overcome the problems highlighted before, a more compact implementation has been chosen: the T2 logic unit.

This logic unit allows to perform AND, NAND, OR, NOR, XOR and XNOR using only 5 NAND gates, on two levels, and 4 selection signals. The schematic is the one in figure 5.1.

In order to compute one of the logical instructions, the select signals are properly activated as follow:

S_0	S_1	S_2	S_3	operation
0	0	0	1	AND
1	1	1	0	$NAND$
0	1	1	1	OR
1	0	0	0	NOR
0	1	1	0	XOR
1	0	0	1	$NXOR$

For example, in order to generate the AND logical operation, we have to select $S_3 = 1$, so that $out = R_1 \cdot R_2$; on the other hand, if we need NAND $S_0 = S_1 = S_2 = 1$ and $S_3 = 0$, so that $out = \overline{R_1} \cdot \overline{R_2} + \overline{R_1} \cdot R_2 + R_1 \cdot \overline{R_2} = \overline{R_1} \cdot \overline{R_2}$ that using the De Morgan law $out = \overline{R_1 \cdot R_2}$. This allows to obtain the best performances also because all paths work in parallel, compacting the area and the delay.

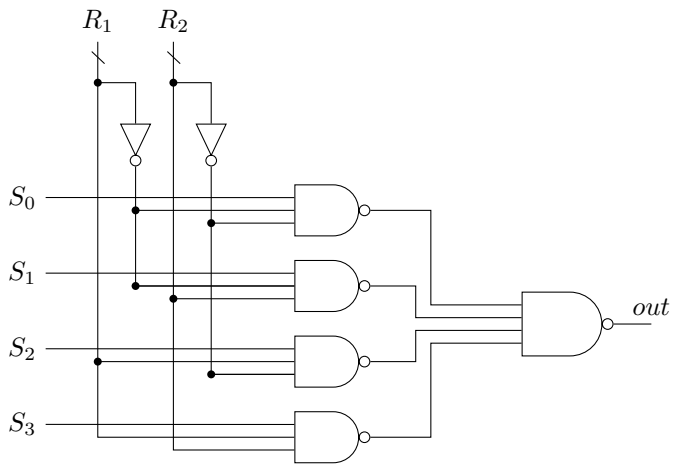


Figure 5.1: Logic unit

5.1.4 Shifting

5.2 Set-Like Operations unit

- setcmp

CHAPTER 6

Memory Stage

6.1 Load-Store Unit

- Unsigned things

6.2 Address Mask Unit

CHAPTER 7

Write Back Stage

Mux selects from Memory Output (LoadStore Unit) or ALU output.

Signal to enable register file write. Registers to delay the write register address

CHAPTER 8

Testing and Verification

8.1 Test Benches

8.2 Simulation

8.3 Post Synthesis Simulation

CHAPTER 9

Physical Design

9.1 Synthesis

9.2 Place and Route

CHAPTER 10

Conclusions