
Politecnico di Torino

## Microelectronic Systems

# DLX Microprocessor: Design & Development

## Final Project Report

Master degree in Electronics Engineering

Referents: Prof. Mariagrazia Graziano, Giovanna Turvani

Authors: Group 15

Alberto Anselmo, Giulio Roggero

Wednesday 31st October, 2018

# Contents

# CHAPTER 1

# Adders

## 1.1 Full Adder

Adders are one of the most used digital components in computer processors. In general an adder is a digital circuit that implements the sum of two numbers expressed on N bits.
In particular, a full adder (FA) is characterized by three inputs and two outputs. If we consider a one-bit full adder, the three inputs are the two numbers to sum and the input carry. The outputs are the sum and the output carry. A schematic diagram of a one-bit full adder is shown in figure

The truth table of the one-bit full adder is shown in table 1.1.

| A B Cin | S | Cout |
|---|---|---|
| 0 0 0 | 0 | 0 |
| 1 0 0 | 1 | 0 |
| 0 1 0 | 1 | 0 |
| 1 1 0 | 0 | 1 |
| 0 0 1 | 1 | 0 |
| 1 0 1 | 0 | 1 |
| 0 1 1 | 0 | 1 |
| 1 1 1 | 1 | 1 |

Table 1.1: Truth table of a 1-bit full adder.

From the truth table we can easily derive the logic equations that describe the full adder:

$$S = A \oplus B \oplus C_{in} \tag{1.1}$$

$$C_{out} = AB + C_{in}(A \oplus B) \tag{1.2}$$

### 1.1.1 Area and power estimation

A simple Bash script has been written to automatise the calculus of area and power starting from the layout of the circuit. The script takes in input the SVG (which is the primary Inkscape's format) description of the layout and it extracts all the necessary information. The output of the script is a .txt file containing different data.
The output text file of the full adder is reported below.

```
Reference layout = planar_full_adder.svg
Number of layers = 1
Total number of magnets = 150
Clock zones = 5
Width [nm] = 2080
Number of vertical magnets =       11
Height [nm] = 1300
Area [nm2] = 2704000
Area [um2] = 2.704000
############ Cu wire case #############
Wire 1 resistance [Ohm] = .26010000000000000000
Wire 2 resistance [Ohm] = .32500000000000000000
Power consumption for one layer [W] = .00000526590000000000
Total power consumption [W] = .00001053180000000000
############ Ta wire case #############
Wire 1 resistance [Ohm] = 81.12642857142857142857
Wire 2 resistance [Ohm] = 101.36904761904761904761
Power consumption for one layer [W] = .00002919927619047619
Total power consumption [W] = .00005839855238095238
```

To summarize, the planar full adder is characterized by:

- Delay = 3 clock cycles;
- Area = 2.7 $\mu m^2$;
- Power = 10.53 $\mu W$.

## 1.2 Ripple Carry Adder

A Ripple-carry adder (RCA) is a more complex adder composed by a cascade of more full adders. It is used to add N-bit numbers and its name derives from the fact that the carry propagates from a full adder to the next one.

.....

.....

...

# CHAPTER 2

# Execution Unit

## 2.1 Overview

We have designed an execution unit that is capable of dealing with almost all the instructions present in the instruction set that has been given. Exceptions are the floating point operations, which requires speicific hardware that we decided not to include. It is worth mentioning that normally the *MUL* operations are executed by dividing them in subsequently pipelined stages. Here, differently, in order to keep the 5 stages in the pipeline and not to alter the normal execution flow, this doe not take place. As a consequence, the maximum operating frequency will be heavily affected by the propagation time needed by the multiplier to finish its job.

In the *Execution Unit* are therefore present the following components:

- **shifter**, used in order to shift and rotate the value contained in an input register
- **adder**, necessary obviously to perform addition and subtraction between values, but also used in the logic comparison operations
- **logic**, unit to perform logic bitwise operations, such as *OR, AND, XOR* and so on
- **comparator**, which purpose is to determine whether an input is greater, smaller or equal than another one
- **multiplier**, whose goal is to perform multiplications between inputs
- PC adder, a dedicated adder to increase by the value of the PC, useful in case of a *JMP* operation
- **registers**, to update the inputs and store the results
- **muxes**, to select the results coming from the required unit and correctly update the outputs of the stage

- **inverter**, needed to perform the *2'S complement* and perform the subtraction

For this stage, a bunch of bits of the CW are used. Here are reported:

- *ENEX*, a general enable for the components and especially for the registers
- *MUX1_SEL*, a selection signal for the mux that is able to select an immediate value or a register an input that as the first term for the other units
- *MUX2_SEL*, with a goal similar as the one above, but for the second input
- *UN_SEL*, 3 bits signal that selects the correct input, which is then sent to a register
- *OP_SEL*, 4 bits selection signal, used to indicate to each unit, with different encoding, the operation to be performed
- *PC_SEL*, selection signal to update the PC counter correctly, following a branch or jump.

## 2.2 Adder

For the adder architecture, we decided not to start from scratches. As long as our P4 adder, designed during the laboratories, has a general description, we resized it to our architecture on 32 bits. This means that we have included the components capable of producing the PG propagate and generate signals and the G ones used to produce only the generate and so the final carry. We consider that therefore the result, in terms of performance can satisfy us. At the input of the block, we have also a *C_IN* signal, which is used to create, in cooperation with the inverter, the 2'S complement of the input used (either immediate or coming from a register), in order to produce as a final result the signed sum. There are few control signals used for this unit.

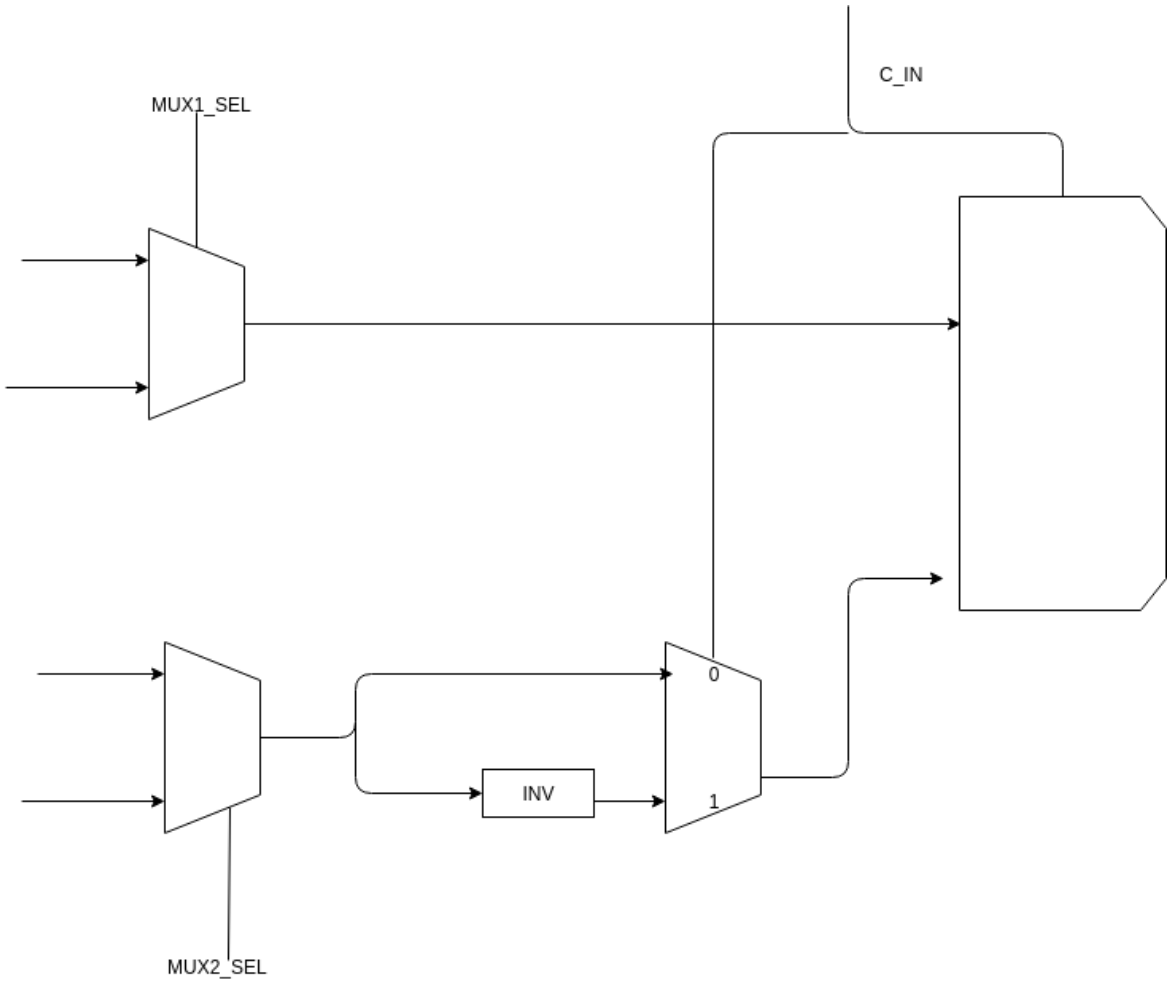


Figure 2.1: Adder with inverter and related selection signal

# APPENDIX A

# Adder behavioural VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ADDER_32 is
        generic (n: NATURAL:= 33);
        port (INA_ADD: in signed (n-1 downto 0);
                        INB_ADD: in signed (n-1 downto 0);
                        OUT_ADD: out signed (n-1 downto 0);
                        CTRL_ADD1, CTRL_ADD2: in STD_LOGIC);
end entity ADDER_32;

architecture Behaviour if ADDER_32 is
        begin
        discrimination: process (CTRL_ADD1, CTRL_ADD2) is
                begin
                        if CTRL_ADD1 = '1' and CTRL_ADD2 = '0' then
                                OUT_ADD <= INA_ADD - INB_ADD;
                        elsif CTRL_ADD1 = '0' and CTRL_ADD2 = '1'
                           then
                                OUT_ADD <= INB_ADD - INA_ADD;
                        else
                                OUT_ADD <= INA_ADD + INB_ADD;
                        end if;
                end process;
end architecture Behaviour;
```