COE-608 - Lab 3a

I REQUIREMENTS:

1. VHDL CODE (for ALU & any inside entity)
2. WAVEFORM SIMULATIONS (any boundary conditions or average cases) -submit as many as you feel necessary
   (2 types (a) Functional Simulation (b) Timing Simulation)
3. SHORT REPORT (~1 page explaining your design & hierarchy)
4. DEMO (demonstration + questions)

II ARITHMETIC LOGIC UNIT (ALU):

(a) What is it ?

As CPU is the brain of a computer, ALU is the brain of the CPU. It executes the instructions which are of arithmetic or logical type.

Typical Arithmetic Instructions:

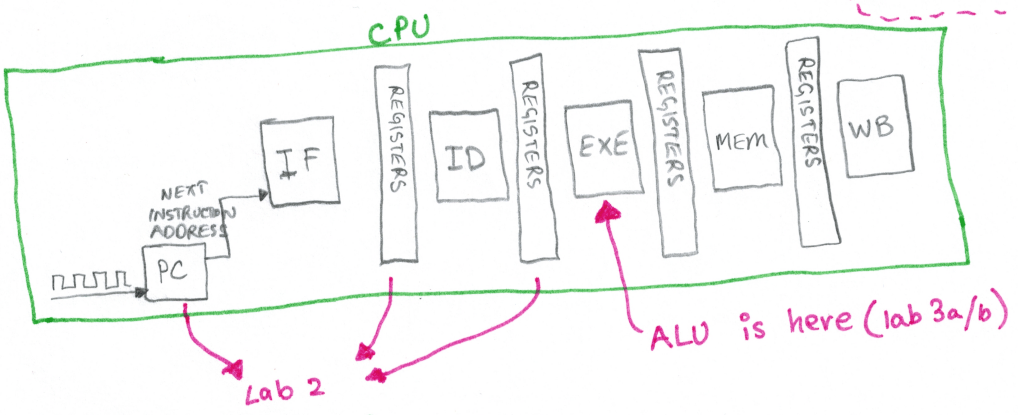
1. Addition (+)
2. Subtraction (-)
3. Multiplication (x)
4. Division (÷)

Typical Logical Instructions:

1. AND
2. OR
3. XOR
4. NOR

IMPORTANT

5. ROL (Roll over Left)
6. ROR (Roll over Right)



(b) ROL / ROR ?

ROL means Roll over Left. That is, ROL 1 ⇒ before ROL → 0b 0001 0000
after ROL → 0b 0010 0000

The bits move left 1 position. (Also known as LSL) → Logical shift Left

ROR means Roll over Right. That is, ROR 1 ⇒ before ROR → 0b 0001 0000
after ROR → 0b 0000 1000

The bits move right 1 position. (Also known as LSR) → Logical shift Right

(c) Why ROL/ROR ?

Eg. ①

In Decimal	In Binary	
5	0000 0101	Before ROL 1
10	0000 1010	After ROL 1

Just by moving the bits 1 position to the left, we multiplied the number by 2!!

∴ In general, value ROL n ⇒ value $\times 2^n$

5 ROL 1 ⇒ $5 \times 2^1 = 10$

5 ROL 2 ⇒ $5 \times 2^2 = 20$

Eg. ②

In Decimal	In Binary	
6	0000 0110	Before ROR 1
3	0000 0011	After ROR 1

Just by moving the bits 1 position to the right, we divided the number by 2!!

∴ In general, value ROR n ⇒ $\frac{\text{value}}{2^n}$

6 ROR 1 ⇒ $\frac{6}{2} = 3$

6 ROR 2 ⇒ $\frac{6}{2^2} = \frac{6}{4} = 1.5 \xrightarrow{\text{floored}} 1$

Important

In this lab, we assume that ROL/ROR functions are logical. This means that when the values are shifted left or right, the empty space is filled with a 0.

Reason: ROR & ROL instructions are used in computers to multiply or divide the numbers faster.

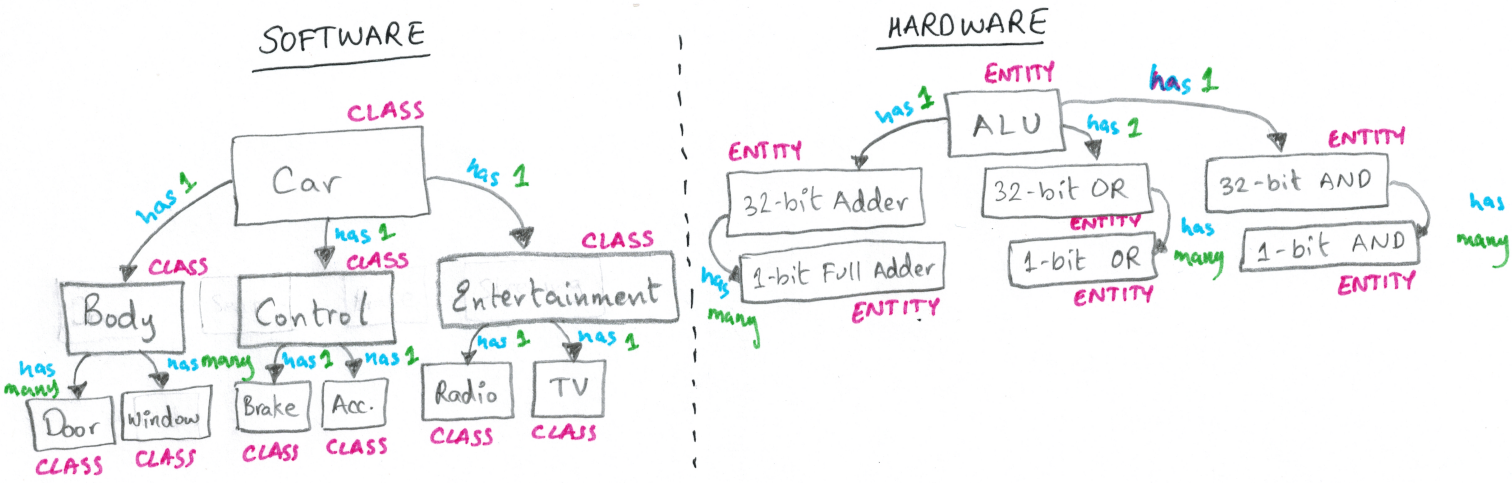
Want 7 × 4 ? ⇒ shift the bits to left by 2 positions

Want 18 ÷ 8 ? ⇒ shift the bits to right by 3 positions

What if 7 × 3 ?

Shift the bits to left 2 positions to get 7×4 = 28. Then subtract 7 from answer 28 − 7 = 21.

III DESIGN ABSTRACTION :



CLASS = ENTITY

OBJECT = COMPONENT

THREAD/FUNCTION ≈ PROCESS

Not exactly

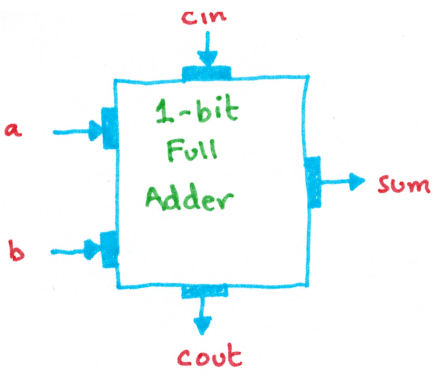
fullAdder.vhd

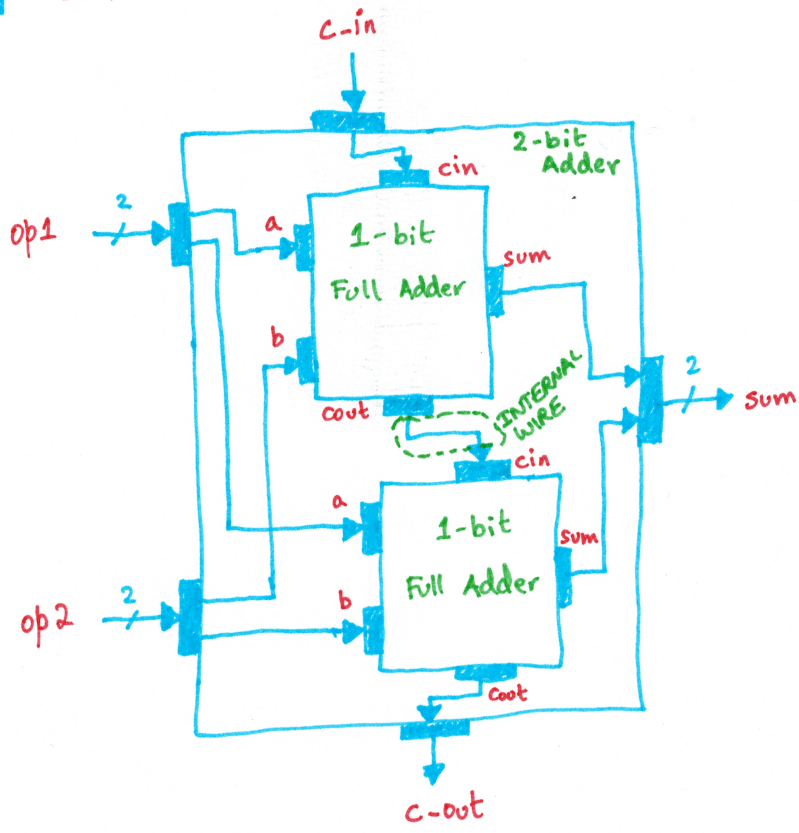
```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_arith.ALL;
use ieee.std_logic_unsigned.ALL;

entity fullAdder is
    port (
        a   : in  std_logic;
        b   : in  std_logic;
        cin : in  std_logic;
        sum : out std_logic;
        cout: out std_logic
    );
end fullAdder;

architecture fullAdder9mp of fullAdder is
begin
    sum <= (a xor b) xor cin;
    cout <= (a and b) or ((a xor b) and cin);
end fullAdder9mp;
```

adder2bit.vhd

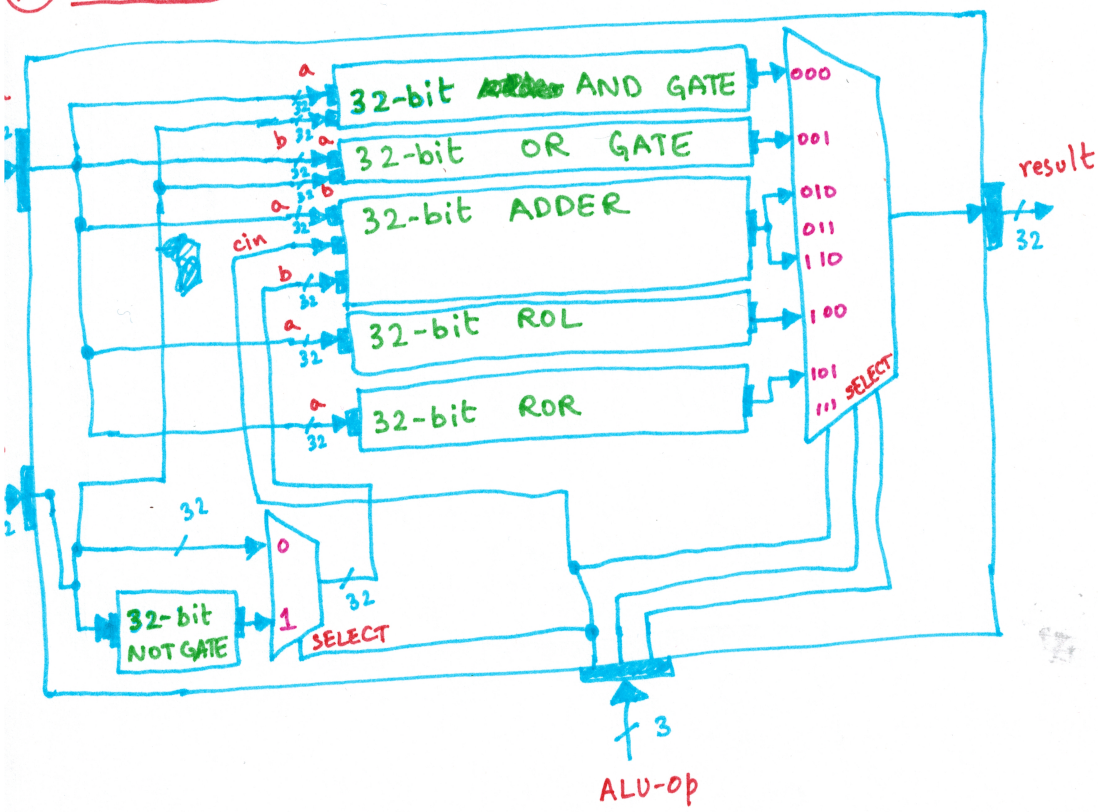
```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_arith.ALL;
use ieee.std_logic_unsigned.ALL;

entity adder2bit is
    port (
        op1 : in std_logic_vector(1 downto 0);
        op2 : in std_logic_vector(1 downto 0);
        c_in : in std_logic;
        sum : out std_logic_vector(1 downto 0);
        c_out : out std_logic
    );
end adder2bit;

architecture adder2bit9 of adder2bit is
    component fullAdder is
        port(
            a: in std_logic;
            b: in std_logic;
            cin: in std_logic;
            sum: out std_logic;
            cout: out std_logic
        );
    end component;
    signal carrywire : std_logic;
begin
    a1: fullAdder port map (op1(0), op2(0), c_in, sum(0), carrywire);
    a2: fullAdder port map (op1(1), op2(1), carrywire, sum(1), c_out);

end adder2bit9;
```

Internal wire to bind stuff up


cin
1-bit
Full
Adder
a
b
sum
cout


c_in
2-bit
Adder
cin
op1
2
a
1-bit
Full Adder
sum
b
cout
INTERNAL WIRE
2
sum
cin
a
1-bit
Full Adder
sum
op2
2
b
Cout
c_out

(IV) LAB HINT :


a
32-bit AND GATE
000
b
32
32-bit OR GATE
001
32-bit ADDER
010
011
110
cin
b
32-bit ROL
100
a
32
32-bit ROR
101
111
SELECT
result
32
32
0
32-bit
NOT GATE
1
32
SELECT
3
ALU-op