JAYPEE UNIVERSITY OF ENGINEERING & TECHNOLOGY, GUNA
DEPARMENT OF COMPUTER SCIENCE & ENGINEERING
Course: Computer Organization & Architecture Lab
Course Code: 18B17CI474
B. Tech. (CSE VI Sem.)

Experiment # 3

Aim: Design of 4-bit adder-subtractor circuits.

Arithmetic operations like addition, subtraction, multiplication, division are basic operations to be implemented in digital computers using basic gates like AND, OR, NOR, NAND etc. Among all the arithmetic operations if we can implement addition then it is easy to perform multiplication (by repeated addition), subtraction (by negating one operand) or division (repeated subtraction). Half adder can be used to add two 1-bit binary numbers and one 1-bit full adder, which uses previously generated carry as third input bit, can be implemented using two half adders. It is also possible to create logical circuits using multiple full adders to add ***n*-bit** binary numbers as following:-

- **Ripple Carry Adder:** A ripple carry adder (RCA), also known as parallel adder, is a digital circuit that produces the arithmetic sum of two binary numbers. It can be implemented with full adders connected is cascaded, with the carry ripples from each full adder connected to the carry input of the next full adder in the chain. Therefore, in the RCA, the sum of the most significant bit is only available after the carry signal has rippled through the adder from the least significant stage to the most significant stage. For an n-bit RCA, the propagation delay for sum and carry is $2n + 4$ and $2n + 3$ respectively**.** The layout of RCA is simple, which allows for fast design time; however, this adder is relatively slow, since each full adder must wait for the carry bit to be calculated from the previous full adder. The block diagram of 4-bit ripple carry adder is shown here below –

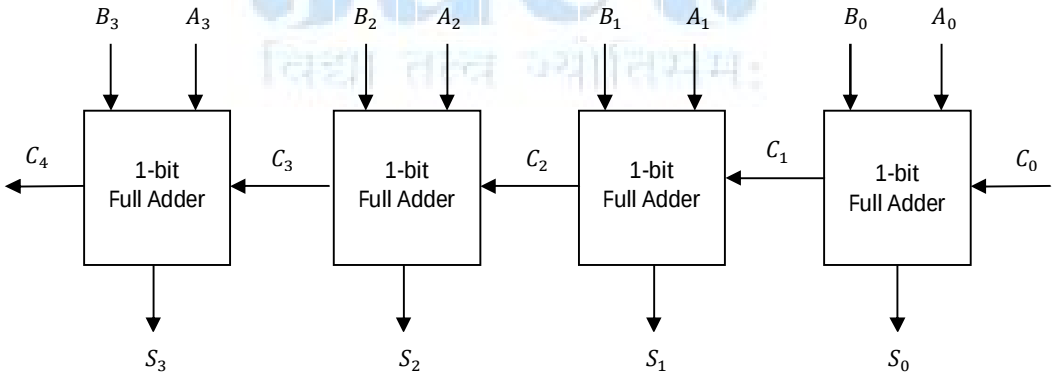

Figure 1: 4-bit ripple carry adder

Boolean Expressions

$$Sum: \quad S_i = A_i \oplus B_i \oplus C_{i-1}$$

$$Carry: \quad C_{i+1} = A_iB_i + A_iC_i + B_i C_i \qquad for\ i = 0, 1, 2, 3$$

- **4-bit Binary Adder-Subtractor:** The subtraction of binary numbers can be done most conveniently by means of complements i. e. A – B can be done by taking the 2's complement of B and adding it to A which can be written as A+$\overline{B}$+1. Both addition and subtraction operations can be combined into one common circuit by including an exclusive-OR gate with each full-adder. A 4-bit adder-subtractor circuit is shown in Fig. 2. The initial carry C_0 controls the operation. When C_0 = 0 the circuit is an adder and when C_0 = 1 the circuit becomes a subtractor. Each exclusive-OR gate receives input C_0 and one of the inputs of B. When C_0 = 0, we have B $\oplus$ 0 = B. The full-adders receive the value of B, the initial carry is C_0, and the circuit performs A plus B. When C_0 = 1, we have B $\oplus$ 1 = $\overline{B}$ and C_0 = 1. The B inputs are all complemented and a 1 is added through the input carry.

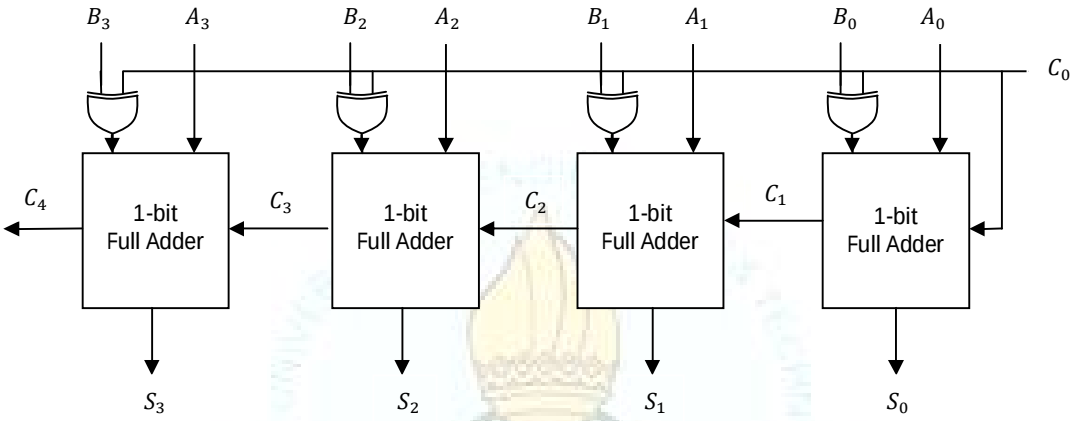

Figure 2: 4-bit adder-subtractor

Exercise#1: Design 4-bit ripple carry adder shown in Fig.1 using

(i) structural style of architecture

(ii) **generic** and **for-generate** statements in structural style of architecture

Exercise#2: Design 4-bit adder-subtractor shown in Fig.2 using structural style of architecture.

Note: (i) In order to parameterize an entity/component, VHDL provides the **generic** clause. In the entity declaration, all the values that have to be customized can be passed using **generic** clause. For example, entity of 4-bit adder (shown in Fig.1) using **generic** statement can be declared as following:-

```
entity Adder_4bit is
generic (N:  integer := 4);
port (A,B: in std_logic_vector (N-1 downto 0);
C : inout std_logic_vector (N downto 0);
S : out std_logic_vector (N-1 downto 0));
end Adder_4bit;
```

(ii) Syntax of **"for loop/generate"** statement with respect to a full adder as component in Figure 1.

```
F: for i in 0 to N-1 generate
        FA: fulladder port map (A(i), B(i), C(i), S(i), C(i+1));
end generate;
```

Note: Replace **generate** with **loop** in case of program is written in dataflow and behavioural architecture.