Submitted by

WWW.ASSIGNMENTPOINT.COM

In computing, an Arithmetic Logic Unit (ALU) is a digital circuit that performs arithmetic and logical operations. The ALU is a fundamental building block of the central processing unit (CPU) of a computer, and even the simplest microprocessors contain one for purposes such as maintaining timers.

Mathematician John von Neumann proposed the ALU concept in 1945, when he wrote a report on the foundations for a new computer called the Electronic Discrete Variable Automatic Computer (EDVAC). Research into ALU remains an important part of computer science, falling under arithmetic and logic structures in the ACM computing classification system.

1.2 Methodology:

An Arithmetic logic unit system has been developed by sequence of operation. To achieve a successful ALU design we use following methodologies:

- Studying literature on different types of bit and their implementation.
- Studying the existing method for ALU arithmetic and logic operation.
- Analyzing and designing for the proposed system.
- Implementation of the desiring design of Arithmetic Logic Unit.

www.AssignmentPoint.com

CHAPTER-2

How to Design an ALU

www.AssignmentPoint.com

2.1 Arithmetic Logic Unit:

An arithmetic logic unit (ALU) is a multi-operation, combinational-logic digital function. It can perform a set of basic arithmetic operations and set of logic operations. The ALU has a number of selection lines to select a particular operation in the unit. The four data inputs form A are combined with the four inputs from B to generate an operation at the F. The mode select inputs S_2 distinguish between arithmetic and logic operation. The two function select inputs S_1 and S_0 Specify particular arithmetic or logic operation to be granted.

2.2 Block Diagram of ALU:

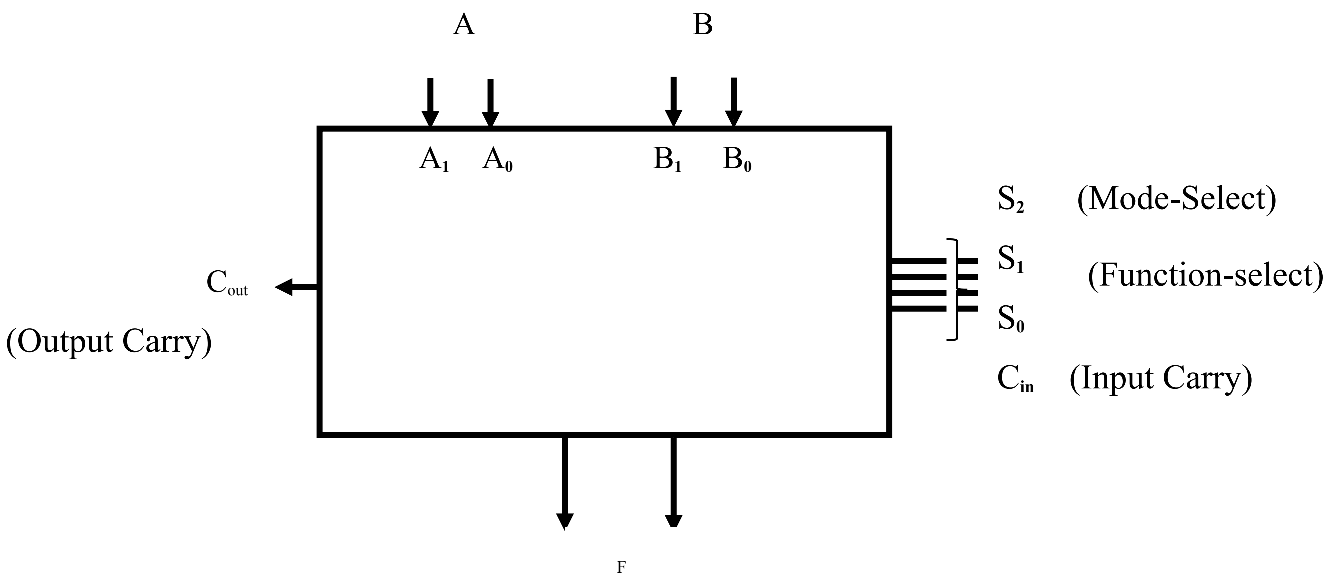


Fig: 2.1 Block Diagram of a 2 bit ALU

Basic features of a 2-bit ALU:

1. This 2-bit ALU has been designed based on 8 arithmetic operations and four logic operations. At first, the circuit for 2-bit ALU has been simulated using Pspice software and then the simulated circuit is finally implemented on bread board.

2. Our purpose was to reduce the time delay.

3**.** We took a new attempt by using this 2-bit ALU by using the latest version of Pspice software where the full-adder is also designed by logic gate.

4. Despite their complexity, they are only logic.

www.AssignmentPoint.com

Logic Gates:

2.3 NOT Gate:

(1) Instruction Format: NOT Operand A

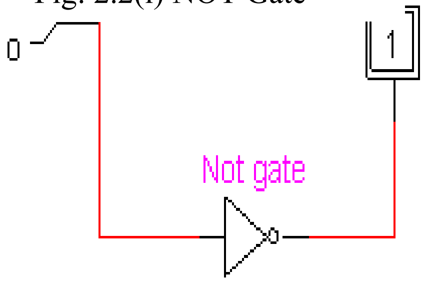
(2) Function: Operand A is a input. This operator performs the bitwise NOT operation on Operand A.

The truth table defines the behavior of each bit operation shown below:

INPUT	OUTPUT —
0	1
1	0

Fig: 2.2(ii) NOT gate Truth Table

Fig: 2.2(i) NOT Gate



1

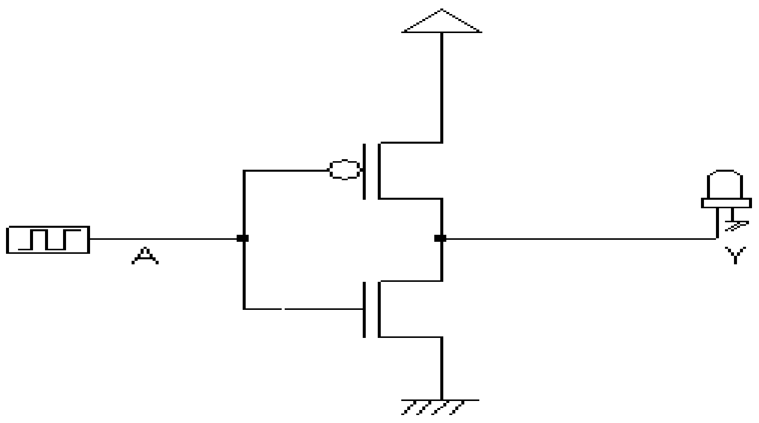


Fig: 2.2(iii) Inverter Switch

Timing diagram of NOT gate:

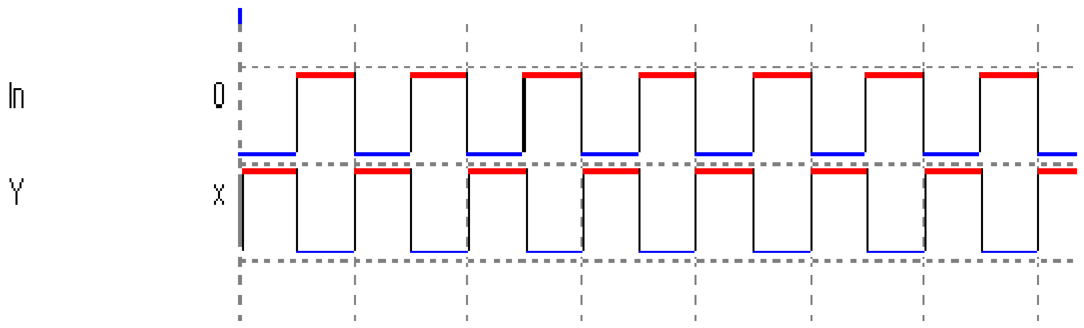


Fig: 2.2(iv) Timing Diagram of NOT Gate.

www.AssignmentPoint.com

2.4 AND Gate:

(1) Instruction Format: AND Operand A Operand B

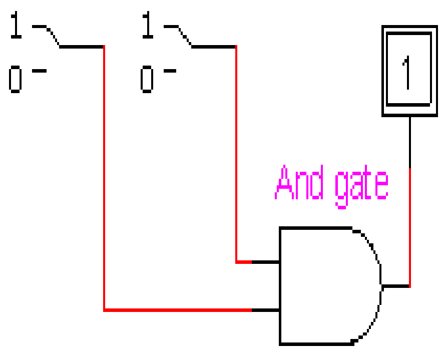
(2) Function: Operand A and Operand B are two inputs. This operator performs the bitwise AND operation, and put the result in F = X. Y

The circuit symbol and truth table defines the behavior of each bit operation shown below:

INPUT		OUTPUT
X	Y	X*Y
0	0	0
0	1	0
1	0	0
1	1	1

Fig: 2.3(i) AND Gate

Fig: 2.3(ii) AND Gate Truth Table



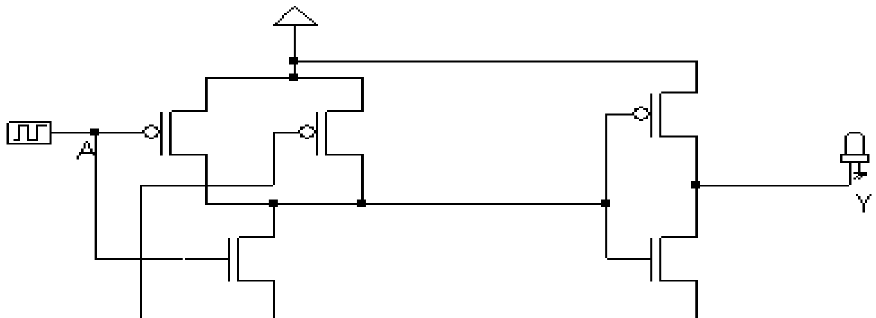


Fig: 2.3(iii) CMOS AND Gate

Timing diagram of AND gate:

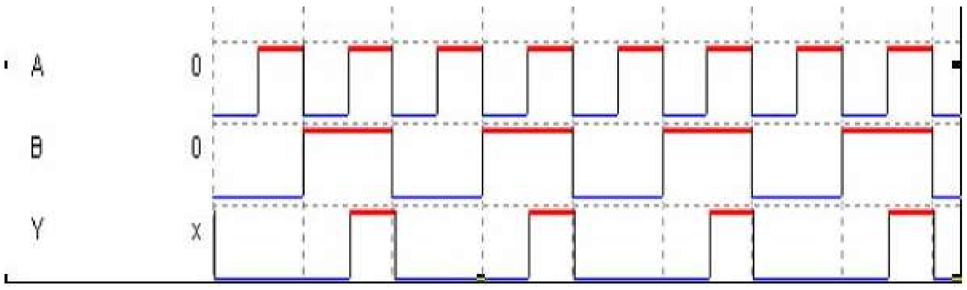


Fig 2.3 (iv) Timing Diagram of AND gate

2.5 OR Gate:

(1) Instruction Format: OR Operand A Operand B

(2) Function: Operand A and Operand B are two 16 bits register inputs. This operator performs the bitwise OR operation, and put the result in F = X +Y.

The truth table defines the behavior of each bit operation shown below:

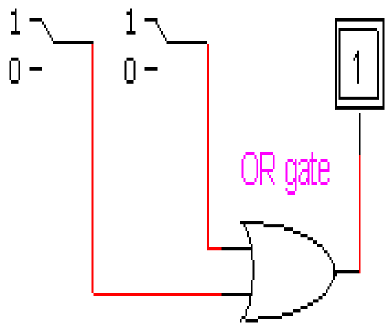


Fig: 2.4(i) OR Gate

INPUT		OUTPUT
X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

Fig: 2.4(ii) OR Gate Truth Table

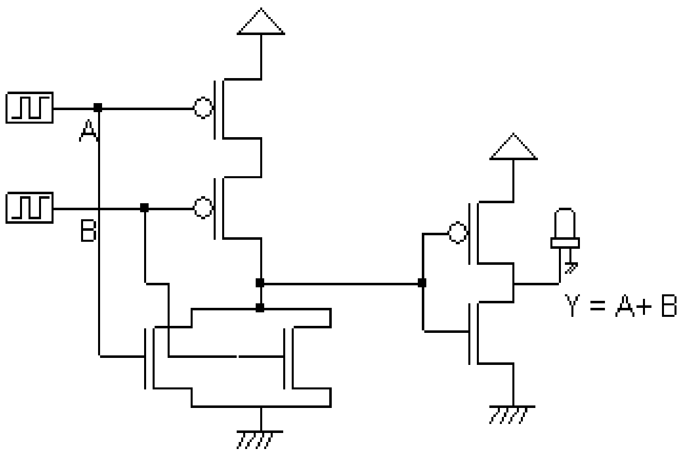


Fig: 2.4(iii) CMOS OR Gate

Timing diagram of OR gate:

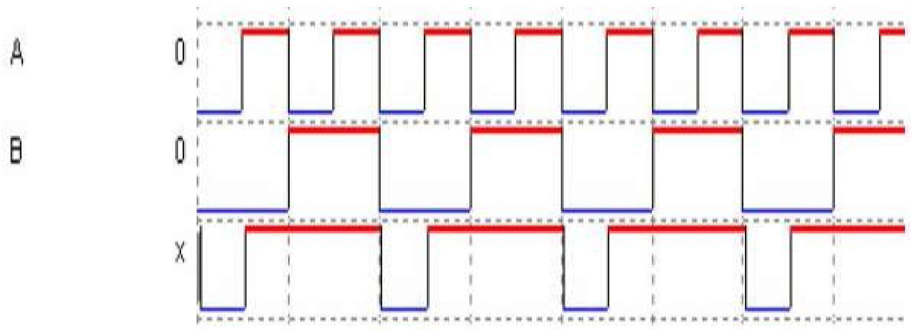


Fig: 2.4(iv) Timing Diagram of OR Gate

2.6 XOR Gate:

(1) Instruction Format: XOR Operand A Operand B

(2) Function: Operand A and Operand B are two 16 bits register inputs. This operator performs the bitwise XOR operation, and put the result $F = X \oplus Y$

The truth table defines the behavior of each bit operation shown below:

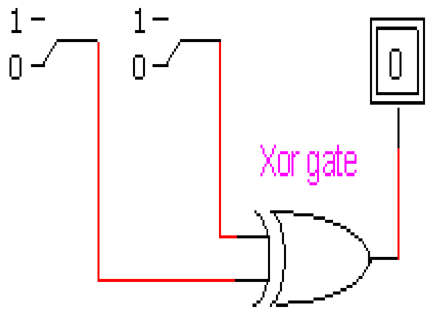


Fig: 2.5(i) XOR Gate

INPUT		OUTPUT
X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Fig: 2.5(ii) XOR Gate Truth Table

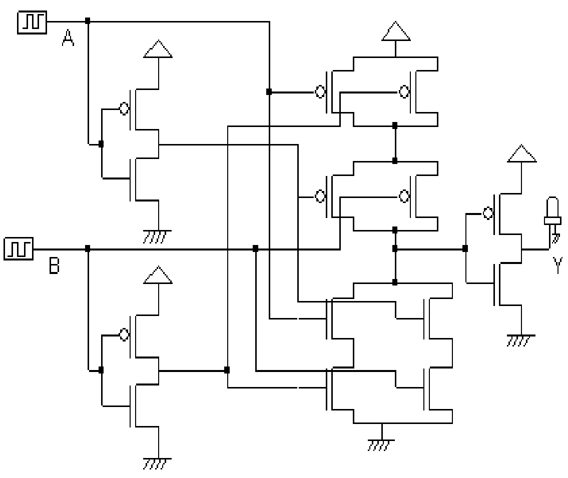


Fig: 2.5(iii) CMOS XOR gate

Timing Diagram of XOR gate:

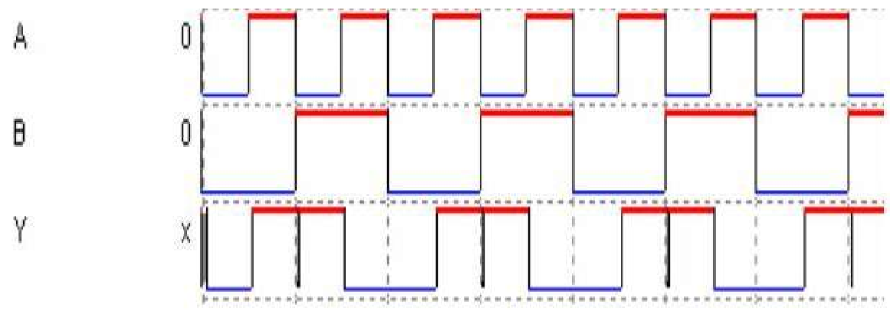


Fig: 2.5 (iV) Timing Diagram of XOR Gate.

2.7 Full adder:

A logic circuit that accepts three digit binary numbers and produce two outputs, a sum output (S) and a carry output (C_{out}).

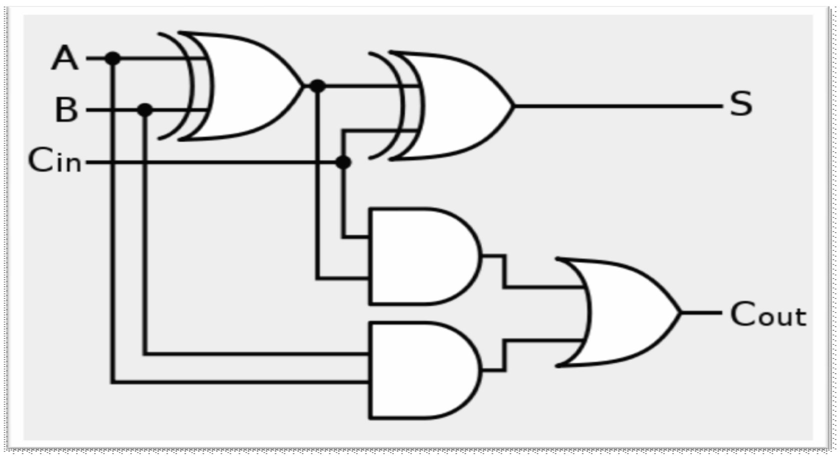


Fig: 2.6(i) Full-adder Circuit Diagram.

Full adder truth table:

A	B	C_{in}	C_{out}	F
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Fig: 2.6 (ii) Full-adder Truth Table

Chapter -3

Design of Arithmetic Circuit

www.AssignmentPoint.com

3.1 Addition:

The arithmetic addition is achieved when one set of inputs receives a binary number A the other set of inputs receives a binary number B, and the input carry is maintained at 0 in fig3.1(i) By making $C_{in} = 1$, it is possible to add 1 to the sum shown in fig 3.1(ii)

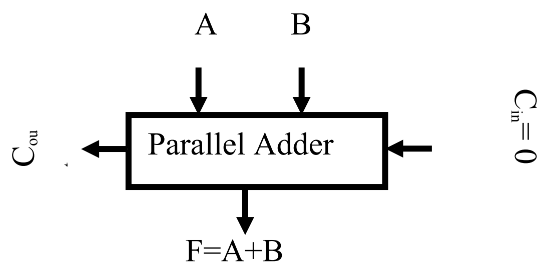


Fig3.1(i)Addition

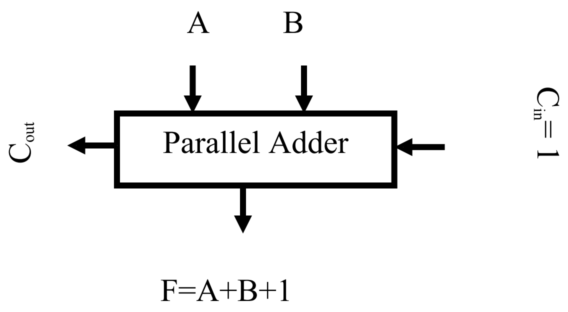
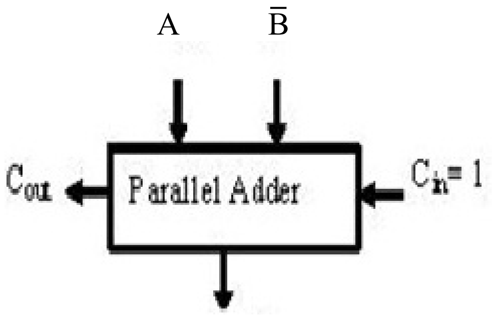


Fig: 3.1(ii) Addition with Carry

3.2 Complement and Subtraction:

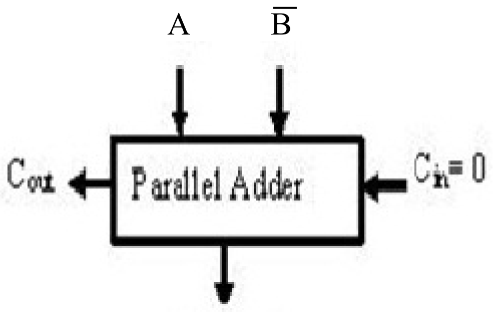
Consider the effect of complementing all the bits of input B, with $C_{in} = 0$, the output produces in fig 3.2(ii).

Which is the sum of A plus the 1's complement of B in fig 3.2(i) adding 1 to this sum by making $C_{in} = 1$, we obtain $F = A + \overline{B} + 1$



$F = A + \overline{B} + 1$

Fig: 3.2(i) Substraction of B

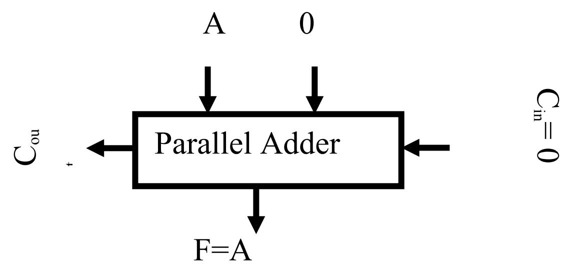


$F = A + \overline{B}$

Fig: 3.2(ii)Addition plus 1's complement

3.3 Transfer A and Increment A:

Force all 0's into the B terminals, we obtain F = A + 0 in fig 3.3(i), which transfer input A output F. Adding 1 through C_{in} as in fig3.3(ii) below, we obtain F = A+1, which is the increment operation.



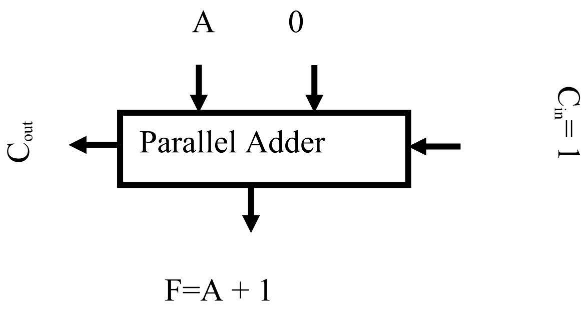


Fig: 3.3(ii) Increment A

3.4 Decrement A:

The condition illustrated in figure that shown bellows. Inserts all 1's into the B terminals. The produces the decrements operation F = A – 1 in fig3.4. To show that this condition is indeed a decrement operation, consider a parallel adder with n full-adder circuits. When the output carry is 1,

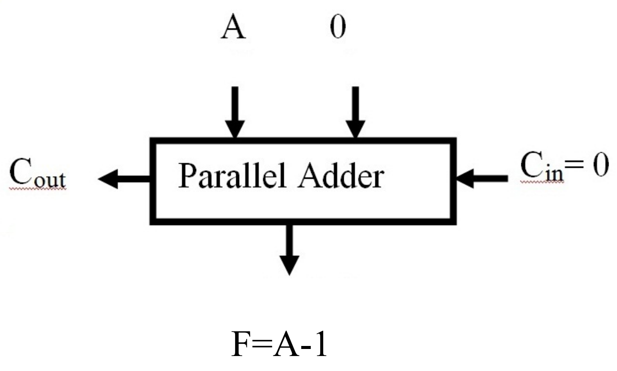


Fig: 3.4-Decrement A

3.5 Basic Operation of 2-bit ALU:

There are two types of ALU operation.

1. Simple operation.
2. Complex operation.

Simple operations:

Most ALU can perform the following operations:

- Bitwise logic operations (AND, NOT, OR, XOR)
- Integer arithmetic operations (addition, subtraction, and sometimes multiplication and division, though this is more expensive)
- Bit-shifting operations (shifting or rotating a word by a specified number of bits to the left or right, with or without sign extension). Shifts can be seen as multiplications and divisions by a power of two.

Complex operations:

Engineers can design an Arithmetic Logic Unit to calculate any operation. The more complex the operation, the more expensive the ALU is, the more space it uses in the processor, the more power it dissipates. Therefore, engineers compromise. They make the ALU powerful enough to make the processor fast.

CHAPTER 4

Design of Logic Circuit

4.1 Design of Logic Circuit:

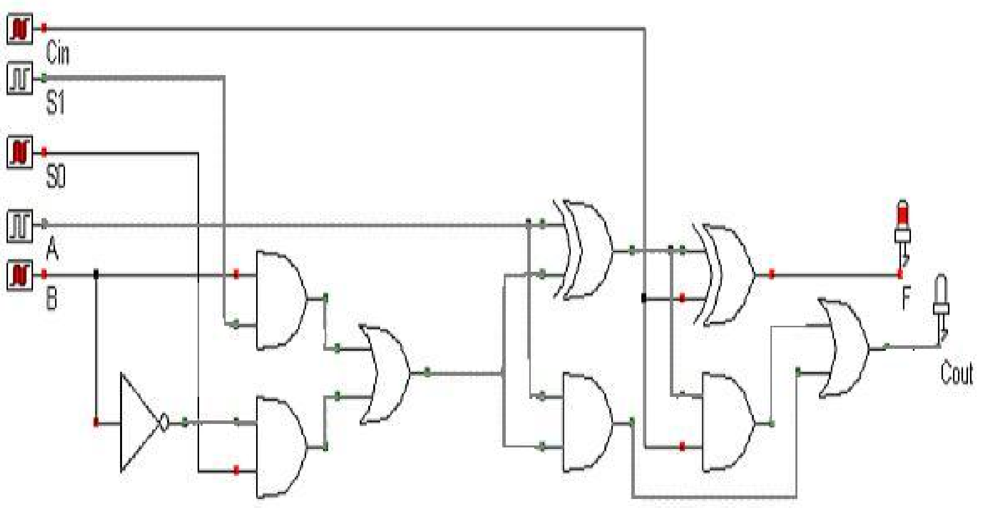
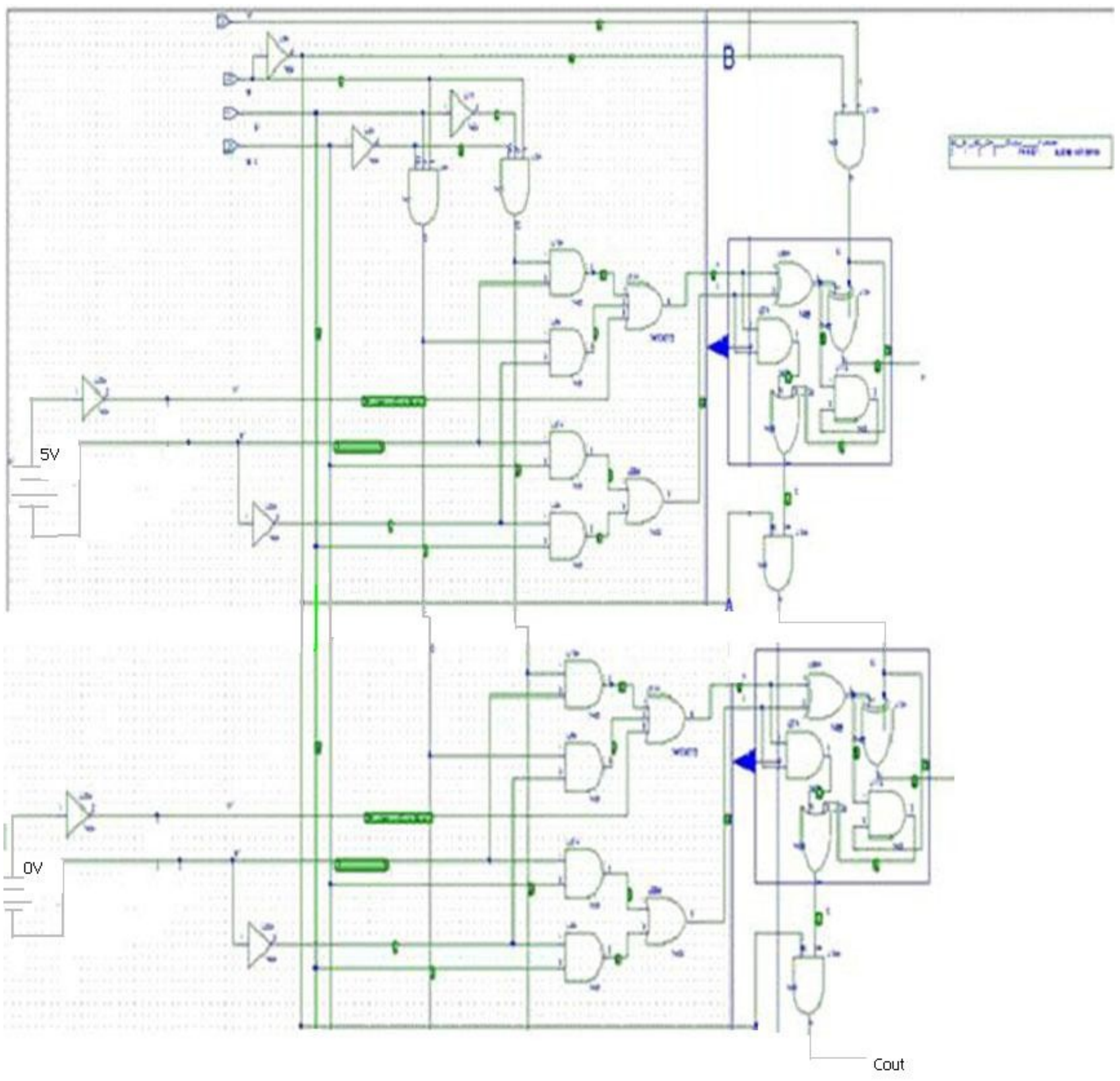


Fig: 4.1 Design of Logical Circuit.

4.2 Pspice View of project:

PSpice View

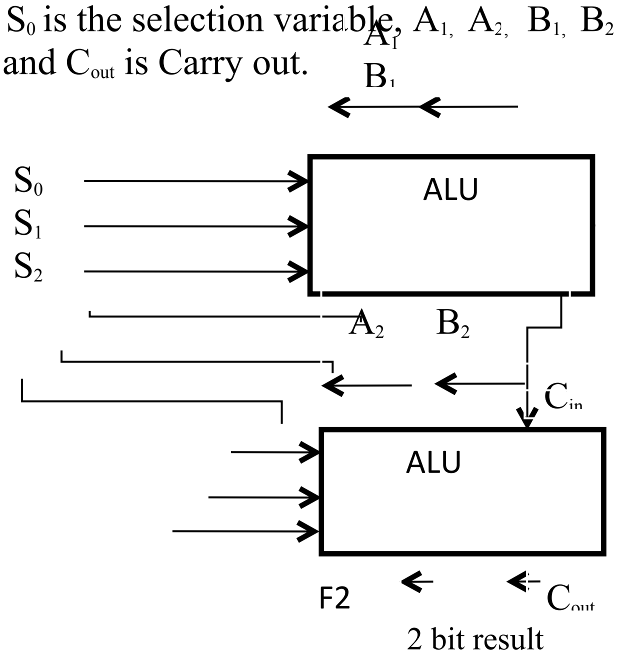


FFig: 4.2 PSpice view of 2-bit ALU

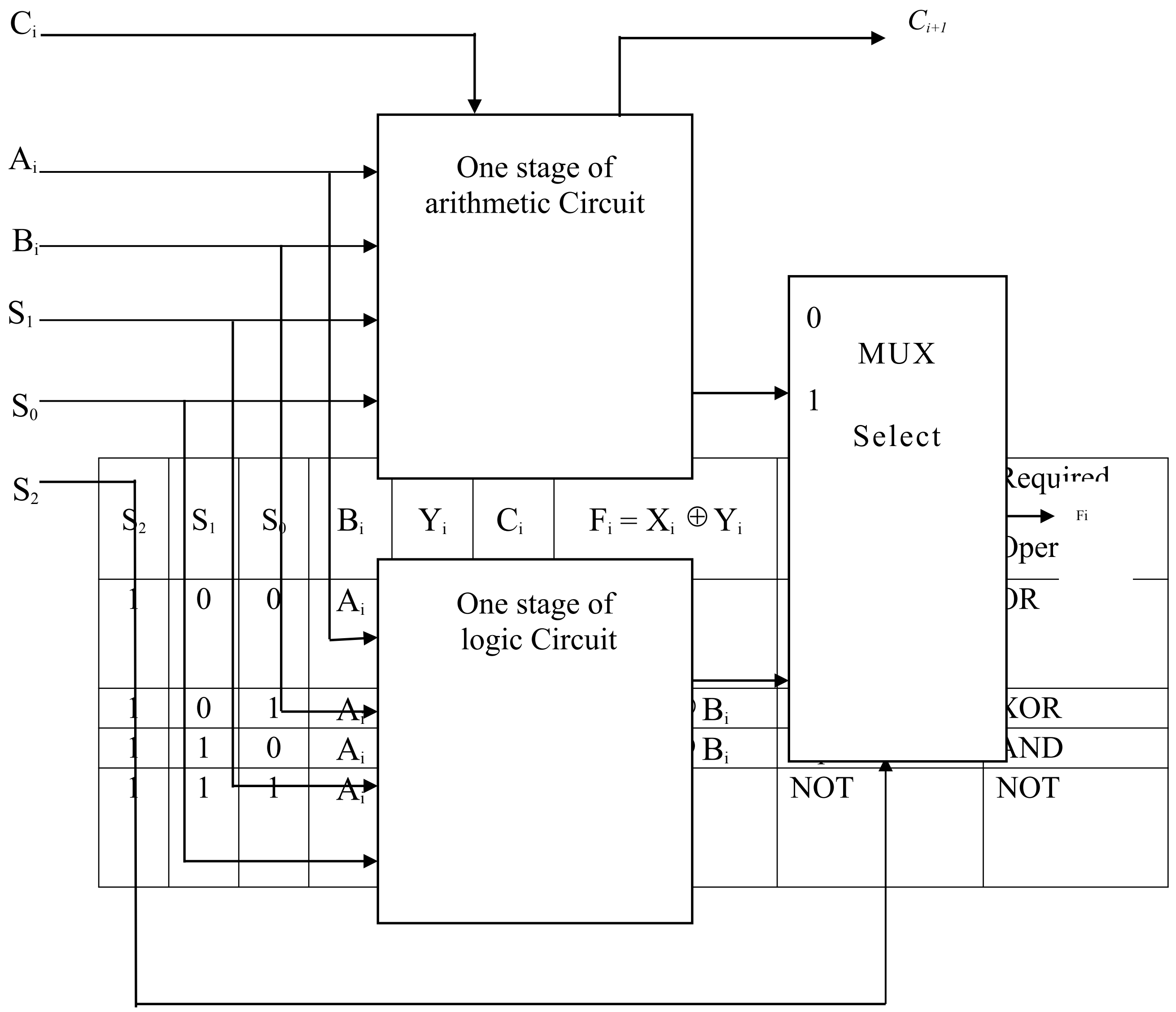
4.3 Block Diagram of 2-bit ALU:

Block Diagram of a 2-Bit ALU

Here S_2 S_1 S_0 is the selection variable, A_1, A_2, B_1, B_2 is the input and C_{in} is Carry in and C_{out} is Carry out.



4.4 Combining logic and Arithmetic Circuit:



S_2	S_1	S_0	B_i	Y_i	C_i	$F_i = X_i \oplus Y_i$		equired per
1	0	0	A_i	[illegible]	[illegible]	[illegible]	[illegible]	R
1	0	1	A_i	[illegible]	[illegible]	B_i	[illegible]	OR
1	1	0	A_i	[illegible]	[illegible]	B_i	[illegible]	ND
1	1	1	A_i	[illegible]	[illegible]		NOT	NOT

Fig-4.4 Combining Logic and Arithmetic circuit

4.5 Logic Operations in One stage of arithmetic Circuit:

Fig: 4.5 Logic Operations in 1 Stage of Arithmetic Circuit

4.6 IC required producing a 2-bit ALU:

We used 4 ICs. They are listed below:

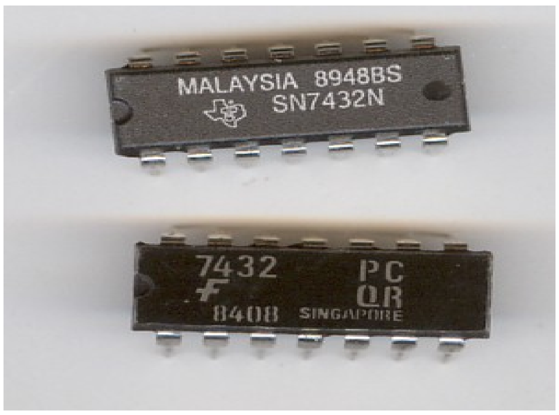


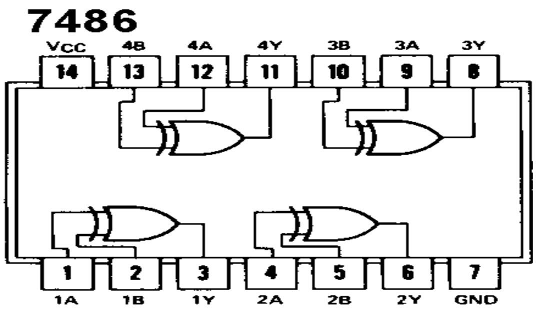
Fig: 4.6(i) 7432 IC

Fig: 4.6(ii) 7408 IC

Fig:4.6(iv)7404IC

Fig: 4.6(iii)7486 IC

4.7 Internal view of using IC:



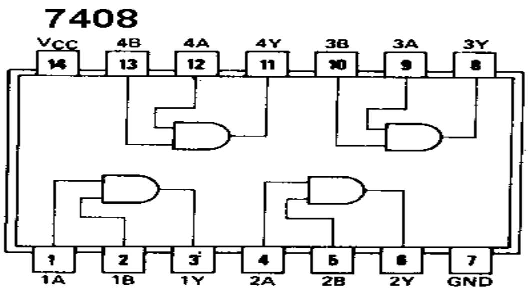
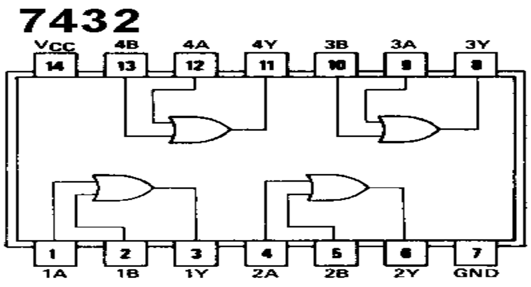


Fig: 4.7(i) 7486IC

Fig: 4.7(ii) 7408IC



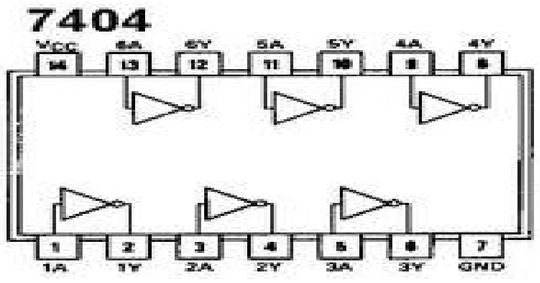


Fig: 4.7(iii) 7432 IC

Fig: 4.7(vi) 7404 IC

4.8 Hardware Implementation:

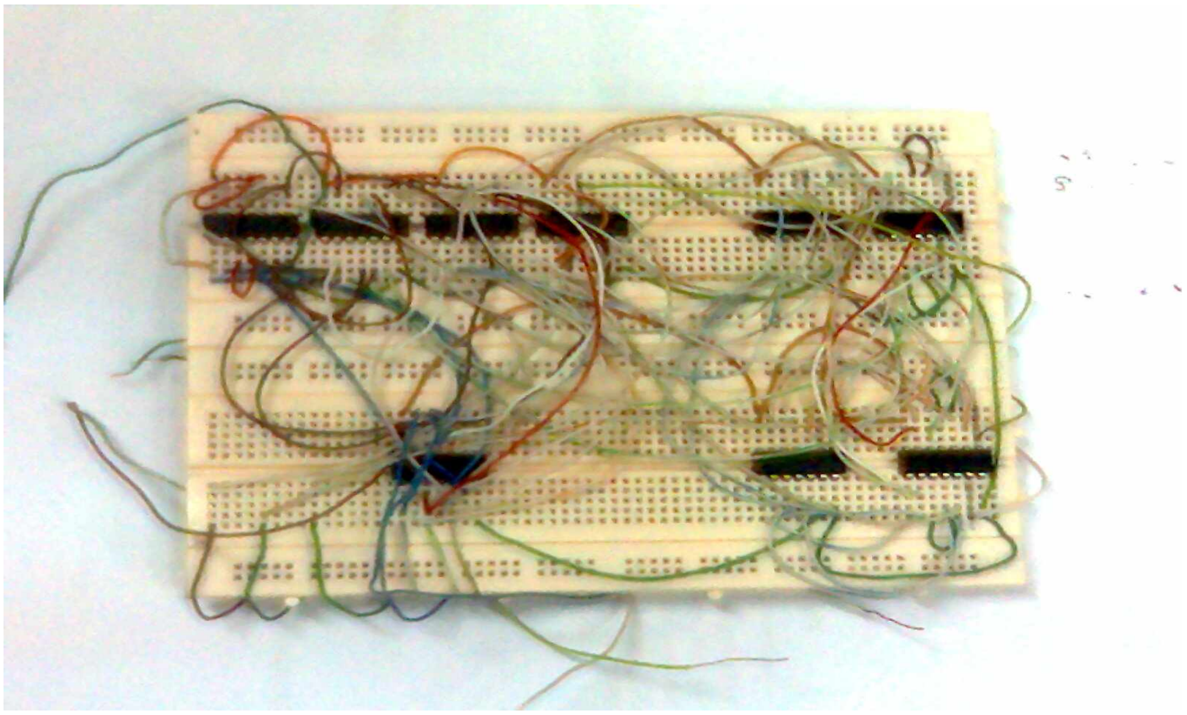


Fig: 4.8 Hardware Implementation of 2 -bit ALU.

CHAPTER-5

Overview of project work

www.AssignmentPoint.com

5.1 Introduction:

The Project topics “Arithmetic Logic Unit (ALU)” is concerned with the processor unit of digital computers. It discusses a typical arithmetic login unit (ALU) is presented and procedure is developed for the design of any other ALU configuration.

5.2 Background and Motivation:

Arithmetic (8-bit and 16-bit) and are highly data parallel.

- Goal was to build a high performance single chip processor.
- Small enough and cheap enough.

5.3 Designing the Arithmatic Logic Unit:

In this section, we design an ALU with eight arithmetic operations and four logic operations. There selection variables S_2, S_1 and S_0 select eight different iperations, and the input carry C_{in} is used to select four additional arithmetic operations. With, $S_2=0$ selection variables S_1 and S_0 together with C_{in} will select the eight arithmetic operations. With $S_2 = 1$, variables S_1 and S_0 will select the four logic operations OR, XOR, AND, NOT .

The design of an ALU is a combinational-logic problem. Because the unit has a regular pattern, it can be broken into identical stages connected in cascade throuth the carries. We can design one stage of the ALU and then duplicate in for the number of stages required. Ther are six inputs to each stage. A_i, B_i, C_{in}, S_2, S_1 and S_0. There are two outputs in each stage: output F_i and the carry out C_{i+1}. One can formulate a trouth table with 64 entries and simplify the two output functions.

5.4 Involved in the design of an ALU:

The steps involved in the design of the logic section.

1.Design the arithmetic section independent of the logic section.

2.Determine the logic operations obtained from the arithmetic circuit in step 1, assuming that the input carriers to all satges are 0.

3.Modify the arithmetic circuit to obtain the required logic operations.

The third step in the design in not a straightforward procedure and requires a certain amount of ingenuity on the part of the designer.

It must be realized that various ALUs are available in IC packages. In a practiocal situation, all that one must do is search for a suitable ALU or processor unit among the IC that are available commercially. Yet, the internal logic of the IC selected must have been designed by a person familiar with logic desing tchniques.

When S_2 = 1, the input carry C_{in} each stage must be 0. with S_1S_0 = 00, each stage as it stands generates the function $F_i = A_i$. Total change of the output to an OR operation, we must change the input to each full-adder circuit from A_i when $S_2S_1S_0$ =100. The unit is to generate an output of F_i = A_i,

To change the output to an OR operation, we must change the input to each full-adder circuit from A_i to $A_i + B_i$. This can be accomplished by OR, B_i and A_i when $S_2S_1S_0$ = 110;

The other setlection variables that give an undesirable output occur when $S_2S_1S_0$ = 110. The unit stands to generate an output Fi = A_i , but We want to generate the AND preartion $F_i = A_iB_i$. Let us investigate the possibility of Oring each input Ai with some Boolean function Ki. The Function so obtained is then used for X_i when $S_2S_1S_0$ = 110;

$F_i{=}X_i \oplus Y_i = (A_i + B_i) \oplus B_i' = A_iB_i + K_iB_i + A_i'K_i'B_i'$

Careful inspection of the result reveals that if the variable $K_i = B_i'$, we obtain an output:

$F_i = A_iB_i + B_iB_i' + A_iB_iB_i' = A_iB_i$

Two terms are equal to 0 because $B_iB_i' = 0$. The result obtained is the AND operation as required. The conclusion is that, if A_i is OR with B_i' when $S_2S_1S_0$ = 110, the output will generate the AND operation.

5.5 2-bit Arithmetic logic unit (ALU) block Diagram:

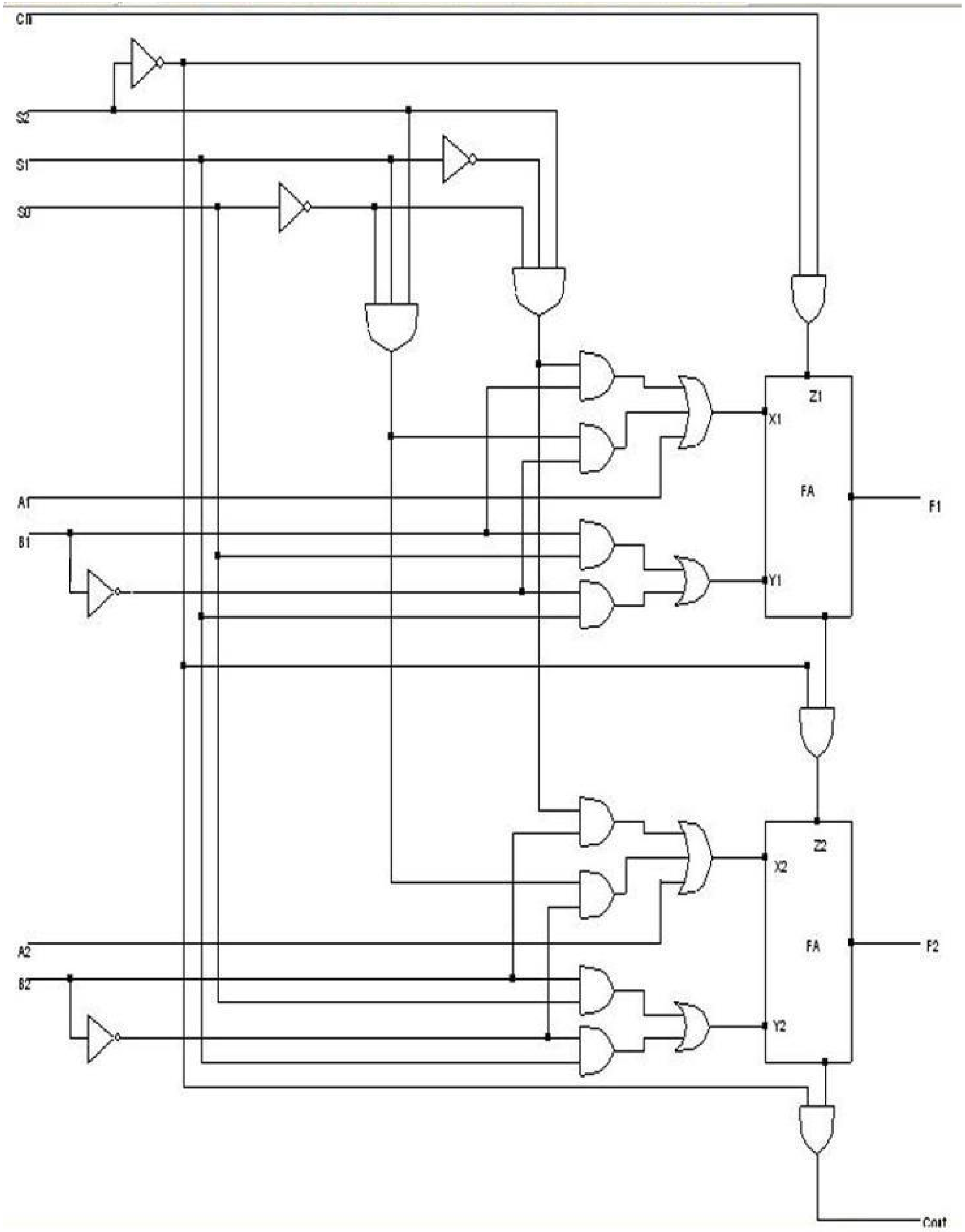


Fig 5.5: Block diagram of 2-bit Arithmetic Logic Unit designed.

www.AssignmentPoint.com

The final ALU is show in Fig 5.5. Only the first two stages are drown, but the diagram can be easily extended to more stages. The inputs to each full-adder circuit are specified by the Boolean functions:

$$X_i = A_i + S_2S_1'S_0'B_i + S_2S_1S_0'B_i'$$

$$Y_i = S_0B_i + S_iB_i'$$

$$Z_i = S_2'C_i$$

When S2 = 0, the three functions reduce to :

$$X_i = A_i$$

$$Y_i = S_0B_i + S_1B_i'$$

$$Z_i = C_i$$

Output Fi is then equal to $Xi \oplus Yi$ and produces the exciluseve-ORwhen $S_2S_1S_0$ = 100, each A_i is OR with B_i to provide the OR operation as discussed above. When $S_2S_1S_0$ = 100, each is OR with B_i' to provide the AND operation as explained previously.

The 12 operations generated in the ALU are summarized.The particular function is selected through S_2, S_1 , S_0 and Cin. The arithmetic operations are indentical to the ones listed for the arithmetic circuit. The value of Cin for the marked with don'tcare X's.

5.6 Function of 2 -Bit ALU:

The designed a ALU can perform eight arithmetic operations and four logic operations. Three selection variables S_2, S_1 and S_0select eight different operations and the input carry C_{in} is used to select four additional arithmetic operations. With S_2=0, selection variables $S_1$and S_0 together with C_{in} will select the eight arithmetic operations. With S_2 =1, variables $S_1$and S_0 will select the four logic operations OR, XOR, AND and NOT.

There are six inputs to each stage A_i ,B_i ,C_i ,S_2, S_1 and S_0. There are two output in each stage: output F_i and the carry out C_{i+1}

Table: Function table for arithmetic circuit:

Function Select			Y equals	Output equals	Function
S_1	S_0	C_{in}			
0	0	0	0	F=A	Transfer A
0	0	1	0	F=A+1	Increment A
0	1	0	B	F=A+B	Add B to A
0	1	1	B	F= A+B+1	Add B to A plus 1
1	0	0	$\overline{B}$	F=A+$\overline{B}$	Add 1's complement of B to A
1	0	1	$\overline{B}$	F=A+$\overline{B}$ + 1	Add 2's complement of B to A
1	1	0	All 1's	F= A-1	Decrement A
1	1	1	All 1's	F=A	Transfer A

Fig 5.6: Function table of 2 bit ALU

CHAPTER- 6

Result and Discussion

www.AssignmentPoint.com

6.1 Final view of the 2-bit ALU by using functional table:

In this table we used selection function as S_2,S_1 ,S_0 ,C_{in} ,and provide 2-bit input as A,B[A_1 ,A_2 , B_1 , B_2] ,output as [F_1 ,F_2] and dictinct funtion like addition, subtraction,increment, decrement, transfer,AND, OR, XOR. After doing the simulation we got that function table fig 6.1 has shown below as a chart.

A_2	B_1	B_2	F1	F2	Function			
1	x	x	0	1	Transfer 'A			
1	x	x	1	1	Increment A			
1	1	0	1	1	Addition			
1	1	0	0	0	Add with carry			
1	1	0	0	0	Subtract with borrow			
1	1	0	1	0	Subtraction			
1	x	x	1	0	Decrement A			
1	x	x	0	1	Transfer A			
1	1	0	1	1	OR			
1	1	0	0	0	XOR			
1	1	0	0	0	AND			
1	1	0	1	0	Complement A			

Function selection Input Output Distinct Function

Fig 6.1:- Final view of the 2-bit ALU by using functional table

6.2 ADVANTAGES & DISADVANTAGES:

ADVANTAGES:

- 2 bit ALU has minimum delay time to implementation.
- Minimize the logic gate.
- Less expensive due to using minimum gate.

DISADVANTAGES:

- Complex circuit diagram.
- To implement 2 bit ALU we need 4 bit input, but according to 4 bit being just logic, ALU require all the inputs to be present at once.
- They have no memory. We will look at adding some next time.
- 2 bit ALU works by 4 bit ALU.
- Output only 0 to 15.
- Input we can implement 4, 8 bit ALU.

Chapter -7

Conclusion and Future work

www.AssignmentPoint.com

7.1 Conclusion:

In this project we designed and implement 2-bit ALU. We have used Pspice software [version].For simulation we implemented this project on bread board and found satisfactory result on different type of simulation like AND, OR, XOR, transfer, increment, subtraction, decrement etc.

7.2 Future plan:

1. We implemented 2-bit ALU, but we want to work on 8-bit ALU, later on we will try to work on 32-bit or 64-bit as much we can.
2. Our purpose is to reduce the delay time.
3. To make the circuit complex free and less expensive.
4. To implement the circuit by applying latest version of the renown software.
5. Minimizing the logic gate as much it's possible.

www.AssignmentPoint.com