UNIT-II: Arithmetic and Logic Unit

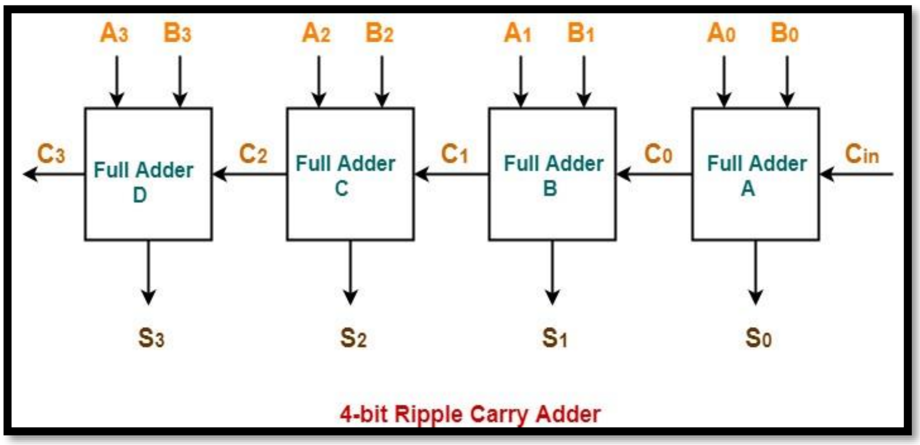
Table of Content

Look Ahead Carry Adder/Carry Look Ahead Adder (CLA)

In Ripple Carry Adder,

- Each full adder has to wait for its carry-in from its previous stage full adder.
- Thus, nth full adder has to wait until all (n-1) full adders have completed their operations.
- This causes a delay and makes ripple carry adder extremely slow.
- The situation becomes worst when the value of n becomes very large.

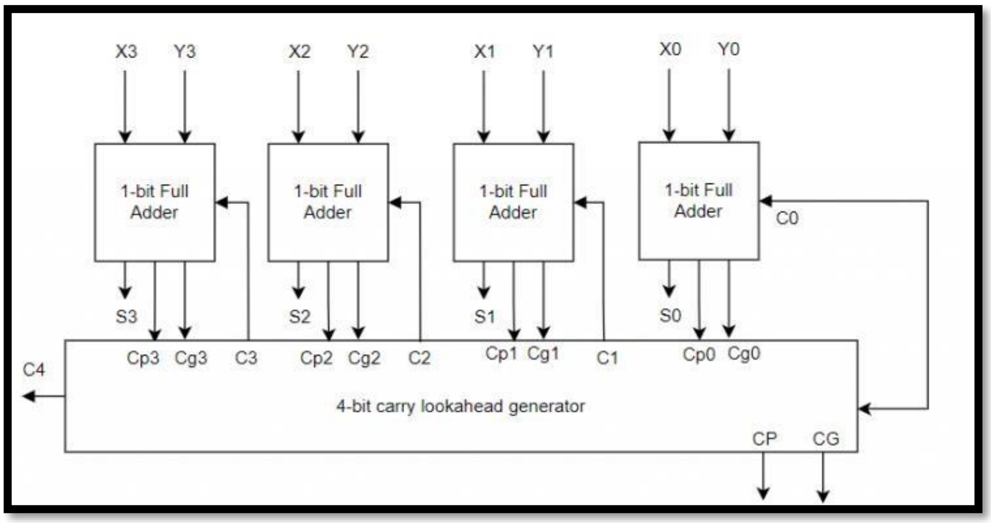
To overcome this disadvantage, Carry Look Ahead Adder comes into play.



Carry Look Ahead Adder-

- Carry Look Ahead Adder is an improved version of the ripple carry adder.
- It generates the carry-in of each full adder simultaneously without causing any delay. The carry-in of any stage full adder depends only on the following two parameters-
- **Carry Look Ahead Adder Working-**
- Bits being added in the previous stages
- Carry-in provided in the beginning
- The above two parameters are always known from the beginning.
- So, the carry-in of any stage full adder can be evaluated at any instant of time.
- Thus, any full adder need not wait until its carry-in is generated by its previous stage full adder. **4-bit**

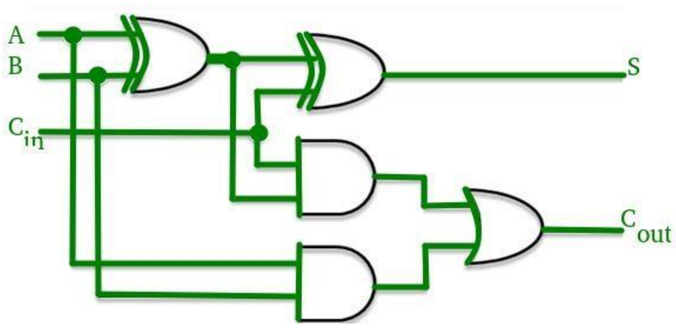
Carry Lookahead Adder Architecture

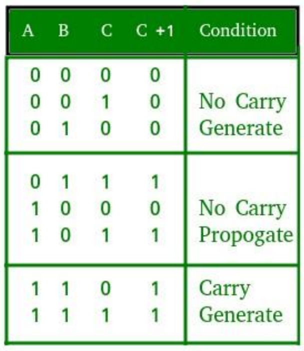


The total number of gate levels in the circuit for carry propagation can be known from the full adder circuit. From input Cin to output Cout, two gates are required which are AND and OR gates. As we are considering a 4bit circuit, the total number of gate levels will be 8. In the same way, for an n-bit parallel adder circuit, there is a 2n number of gate levels.

For the construction of carry lookahead adder, we need two Boolean expressions which are for carry lookahead adder formula for carry propagate Cp and carry generate Cg.

A carry look-ahead adder reduces the propagation delay by introducing more complex hardware. In this design, the ripple carry design is suitably transformed such that the carry logic over fixed groups of bits of the adder is reduced to two-level logic. Let us discuss the design in detail.





A	B	C	C +1	Condition
0	0	0	0	No Carry Generate
0	0	1	0	
0	1	0	0	
0	1	1	1	No Carry Propogate
1	0	0	0	
1	0	1	1	
1	1	0	1	Carry Generate
1	1	1	1	

Consider the full adder circuit shown above with corresponding truth table. We define two variables as ‘carry generate G_i and ‘carry propagate P_i’ then

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

The sum output and carry output can be expressed in terms of carry generate G_i and carry propagate P_i as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

The carry output Boolean function of each stage in a 4 stage carry look-ahead adder can be expressed as

$$C_1 = G_0 + P_0 C_{in}$$
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$
$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

From the above Boolean equations we can observe that C_{4} does not have to wait for C_{3} and C_{2} to propagate but actually C_{4} is propagated at the same time as C_{3} and C_{2} . Since the Boolean expression for each carry output is the sum of products so these can be implemented with one level of AND gates followed by an OR gate.

The implementation of three Boolean functions for each carry output (C_{2} , C_{3} and C_{4}) for a

carry lookahead adder circuit diagram-

Advantages and Disadvantages of Carry Look-Ahead Adder :

Advantages –

- The propagation delay is reduced.
- It provides the fastest addition logic.

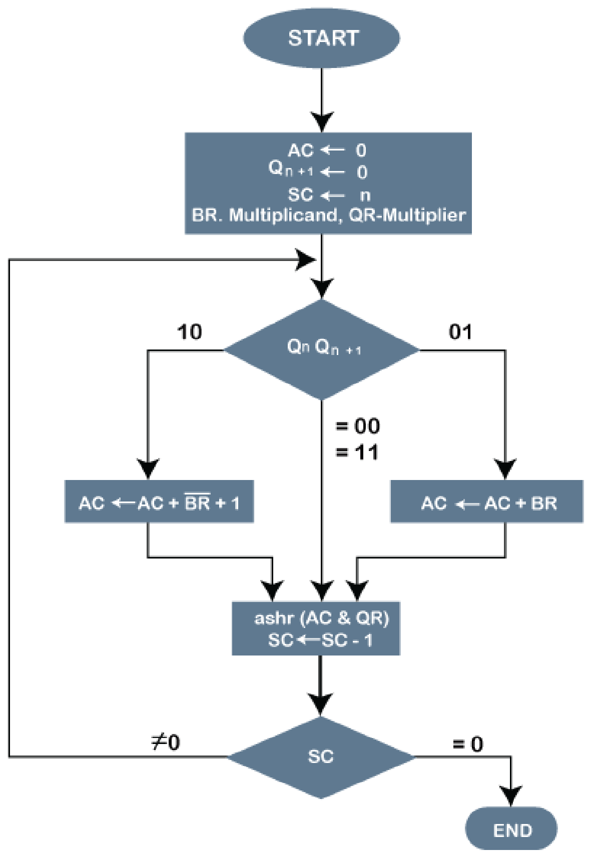
Disadvantages –

- The Carry Look-ahead adder circuit gets complicated as the number of variables increase.
- The circuit is costlier as it involves more number of hardware

Booth's Multiplication Algorithm

The booth algorithm is a multiplication algorithm that allows us to multiply the two signed binary integers in 2's complement, respectively. It is also used to speed up the performance of the multiplication process. It is very efficient too.

Following is the pictorial representation of the Booth's Algorithm:



Working on the Booth Algorithm-

1. Set the Multiplicand and Multiplier binary bits as M and Q, respectively.
2. Initially, we set the AC and Q_{n-1} registers value to 0.
3. SC represents the number of Multiplier bits (Q), and it is a sequence counter that is continuously decremented till equal to the number of bits (n) or reached to 0.
4. A Qn represents the last bit of the Q, and the Q_{n-1} shows the incremented bit of Qn by 1.
5. On each cycle of the booth algorithm, Q_n and Q_{n-1} bits will be checked on the following parameters as follows:
   i. When two bits Q_n and Q_{n-1} are 00 or 11, we simply perform the arithmetic shift right operation (ashr) to the partial product AC. And the bits of Qn and Q_{n+1} is incremented by 1 bit.

   ii. If the bits of Q_n and Q_{n-1} is shows to 01, the multiplicand bits (M) will be added to the AC (Accumulator register). After that, we perform the right shift operation to the AC and QR bits by 1.
   iii. If the bits of Q_n and Q_{n-1} is shows to 10, the multiplicand bits (M) will be subtracted from the AC (Accumulator register). After that, we perform the right shift operation to the AC and QR bits by 1.
6. The operation continuously works till we reached n - 1 bit in the booth algorithm.

7. Results of the Multiplication binary bits will be stored in the AC and QR registers.

Binary Numbers Representation

We can make the binary numbers into the following two groups − Unsigned numbers and Signed numbers.

Unsigned Numbers

Unsigned numbers contain only magnitude of the number. They don’t have any sign. That means all unsigned binary numbers are positive.

Signed Numbers

Signed numbers contain both sign and magnitude of the number. Generally, the sign is placed in front of number. So, we have to consider the positive sign for positive numbers and negative sign for negative numbers.

If sign bit is zero, which indicates the binary number is positive. Similarly, if sign bit is one, which indicates the binary number is negative.

Representation of Un-Signed Binary Numbers

The bits present in the un-signed binary number holds the magnitude of a number. That means, if the unsigned binary number contains ‘N’ bits, then all N bits represent the magnitude of the number, since it doesn’t have any sign bit.

Representation of Signed Binary Numbers

The Most Significant Bit MSB of signed binary numbers is used to indicate the sign of the numbers. Hence, it is also called as sign bit. The positive sign is represented by placing ‘0’ in the sign bit. Similarly, the negative sign is represented by placing ‘1’ in the sign bit.

If the signed binary number contains ‘N’ bits, then N−1 bits only represent the magnitude of the number since one bit MSB is reserved for representing sign of the number.

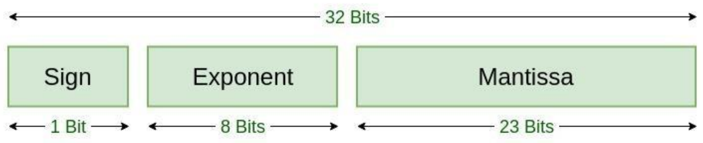
There are three types of representations for signed binary numbers-

- Sign-Magnitude form
- 1’s complement form
- 2’s complement form

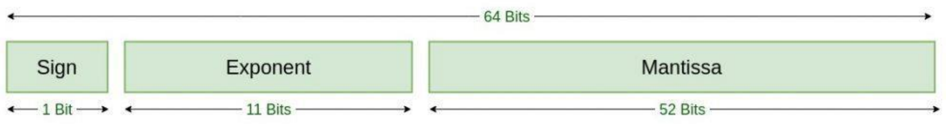
IEEE Standard 754 Floating Point Numbers

The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point computation which was established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE).

IEEE 754 numbers are divided into two based on the above three components: single precision and double precision



Single Precision
IEEE 754 Floating-Point Standard



Double Precision
IEEE 754 Floating-Point Standard

There are several ways to represent floating point number but IEEE 754 is the most efficient in most cases. IEEE 754 has 3 basic components:

1.The Sign of Mantissa –This is as simple as the name. 0 represents a positive number while 1 represents a negative number.

2.The Biased exponent –The exponent field needs to represent both positive and negative exponents. A bias is added to the actual exponent in order to get the stored exponent.

Exponent	**Biased by**
8-bit (single-precision)	127
11-bit (double-precision)	1023

3.The Normalized Mantissa –The mantissa is part of a number in scientific notation or a floating-point number, consisting of its significant digits. Here we have only 2 digits, i.e. O and 1. So a normalised mantissa is one with only one 1 to the left of the decimal.

TYPES	**SIGN**	**BIASED EXPONENT**	**NORMALISED MANTISSA**	**BIAS**
Single precision	1(31st bit)	8(30-23)	23(22-0)	127
Double precision	1(63rd bit)	11(62-52)	52(51-0)	1023

Example – 85.125

85 = 1010101

0.125 = 001

85.125 = 1010101.001

=1.010101001 x 2^6

sign = 0

1. Single precision:

biased exponent 127+6=133

133 = 10000101

Normalised mantissa =

010101001 we will add 0's to

complete the 23 bits

The IEEE 754 Single precision is:

= 0 10000101 01010100100000000000000

2. Double precision:

biased exponent 1023+6=1029

1029 = 10000000101

Normalised mantissa = 010101001

we will add 0's to complete the 52 bits

The IEEE 754 Double precision is:

= 0 10000000101 0101010010000000000000000000000000000000000000000000

Special Values: IEEE has reserved some values that can ambiguity.

1) Zero – Zero is a special value denoted with an exponent and mantissa of 0. -0 and +0 are distinct values, though they both are equal.
2) Denormalised –If the exponent is all zeros, but the mantissa is not then the value is a denormalized number. This means this number does not have an assumed leading one before the binary point.
3) Infinity –The values +infinity and -infinity are denoted with an exponent of all ones and a mantissa of all zeros. The sign bit distinguishes between negative infinity and positive infinity. Operations with infinite values are well defined in IEEE.
4) Not A Number (NAN) – The value NAN is used to represent a value that is an error. This is represented when exponent field is all ones with a zero sign bit or a mantissa that it not 1 followed by zeros. This is a special value that might be used to denote a variable that doesn't yet hold a value.

EXPONENT	**MANTISA**	**VALUE**
0 0 exact 0 255 0 Infinity		
0	not 0	denormalised
255 not	0	Not a number (NAN)

There are five distinct numerical ranges that single-precision floating-point numbers are not able to represent with the scheme presented so far:

1.Negative numbers less than – (2 – 2-23) × 2127 (negative overflow)
2.Negative numbers greater than – 2-149 (negative underflow)
3.Zero
4.Positive numbers less than 2-149 (positive underflow)

5.Positive numbers greater than (2 – 2-23) × 2127 (positive overflow)

Overflow generally means that values have grown too large to be represented.

Underflow is a less serious problem because is just denotes a loss of precision, which is guaranteed to be closely approximated by zero.

Restoring Division Algorithm for Unsigned Integer

Restoring division is usually performed on the fixed point fractional numbers. When we perform division operations on two numbers, the division algorithm will give us two things, i.e., quotient and remainder.

Here, register Q is used to contain the quotient, and register A is used to contain the remainder. Here, the divisor will be loaded into the register M, and n-bit divided will be loaded into the register Q. 0 is the starting value of a register. The values of these types of registers are restored at the time of iteration. That's why it is known as restoring.

Now we will learn some steps of restoring division algorithm, which is described as follows:

Step 1: In this step, the corresponding value will be initialized to the registers, i.e., register A will contain value 0, register M will contain Divisor, register Q will contain Dividend, and N is used to specify the number of bits in dividend.

Step 2: In this step, register A and register Q will be treated as a single unit, and the value of both the registers will be shifted left.

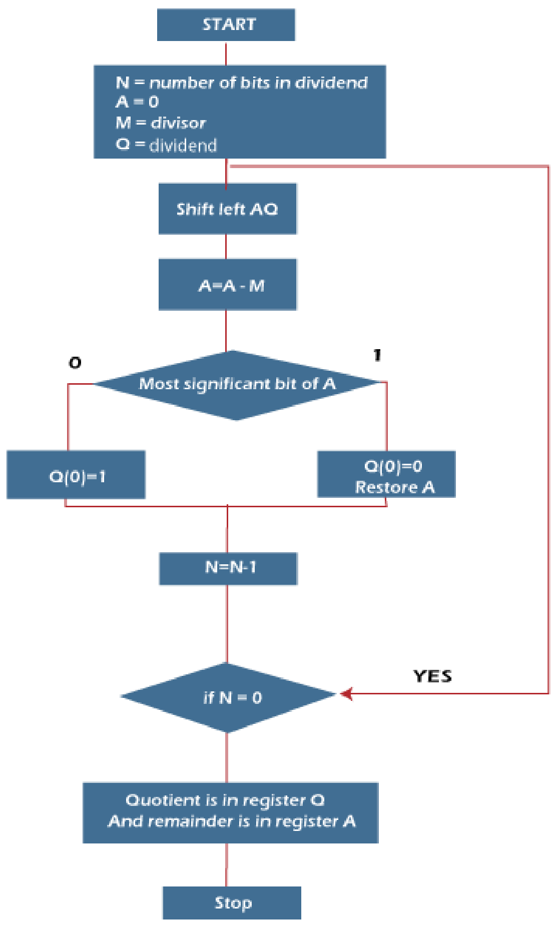
Step 3: After that, the value of register M will be subtracted from register A. The result of subtraction will be stored in register A.

Step 4: Now, check the most significant bit of register A. If this bit of register A is 0, then the least significant bit of register Q will be set with a value 1. If the most significant bit of A is 1, then the least significant bit of register Q will be set to with value 0, and restore the value of A that means it will restore the value of register A before subtraction with M.

Step 5: After that, the value of N will be decremented. Here n is used as a counter.

Step 6: Now, if the value of N is 0, we will break the loop. Otherwise, we have to again go to step 2.

Step 7: This is the last step. In this step, the quotient is contained in the register Q, and the remainder is contained in register A.

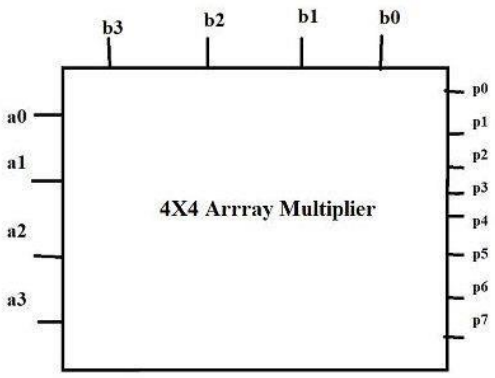

START
N = number of bits in dividend
A = 0
M = divisor
Q = dividend
Shift left AQ
A=A - M
Most significant bit of A
0
1
Q(0)=1
Q(0)=0
Restore A
N=N-1
if N = 0
YES
Quotient is in register Q
And remainder is in register A
Stop

Array Multiplier

An array multiplier is a digital combinational circuit used for multiplying two binary numbers by employing an array of full adders and half adders. This array is used for the nearly simultaneous addition of the various product terms involved. To form the various product terms, an array of AND gates is used before the Adder array. Array multiplication process for two 4-bit unsigned numbers a and b is shown below-

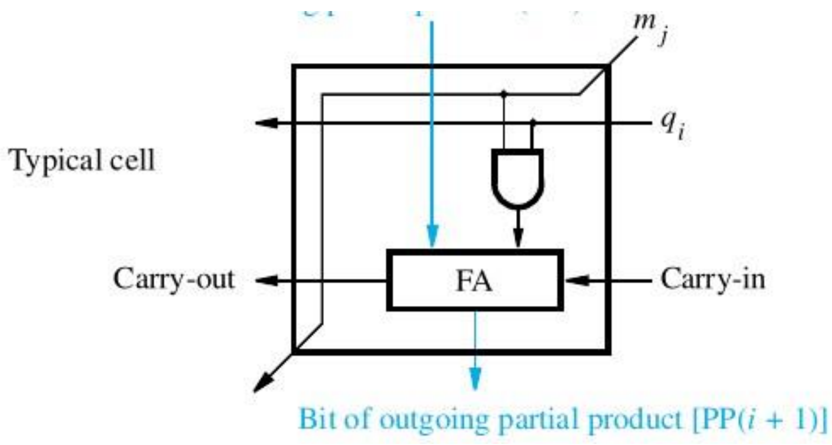
$$\begin{array}{cccccccc}
 & & & & a_3 & a_2 & a_1 & a_0 \\
 & & & \times & b_3 & b_2 & b_1 & b_0 \\
\hline
 & & & & p_{30} & p_{20} & p_{10} & p_{00} \\
 & & & p_{31} & p_{21} & p_{11} & p_{01} & \times \\
 & & p_{32} & p_{22} & p_{12} & p_{02} & \times & \times \\
 & p_{33} & p_{23} & p_{13} & p_{03} & \times & \times & \times \\
\hline
s_7 & s_6 & s_5 & s_4 & s_3 & s_2 & s_1 & s_0
\end{array}$$

Construction and Working of a 4×4 Array Multiplier

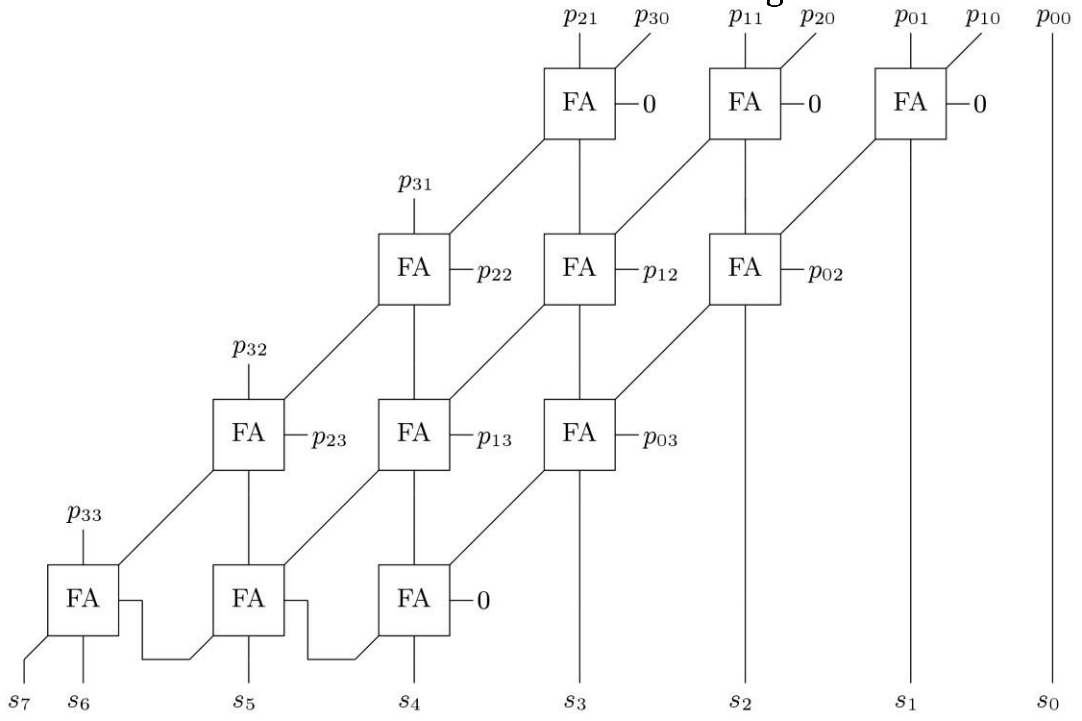


4-by-4-array-multiplier

Building Blocks of 4×4 Array Multiplier



On the contrary to the sequential multiplier, array multiplier is parallel. A array of full adders are used for the multiplication process. For n-bit data width, total n(n-1) full adders are used in this multiplier. Carry outputs of a stage is added in the next stage to form a systolic architecture. But in the last stage carry is used in the same stage to reduce hardware. The architecture of the array multiplier is shown below.



logic-diagram-of-4-by-4 – array – multiplier

The design structure of the array Multiplier is regular, it is based on the add shift algorithm principle.

Partial product = the multiplicand * multiplier bit………

where AND gates are used for the product, the summation is done using Full Adders and Half Adders where the partial product is shifted according to their bit orders. In an n*n array multiplier, n*n AND gates compute the partial products and the addition of partial products can be performed by using n* (n – 2) Full adders and n Half adders. The 4×4 array multiplier shown has 8 inputs and 8 outputs

For a 4×4 Array Multiplier, it needs 16 AND gates, 4 Half Adders(HAs), 8 Full Adders (FAs). Total 12 Adders.

Advantages of 4×4 Array Multiplier

The advantages of array multiplier are-

- Minimum complexity
- Easily scalable
- Easily pipelined
- Regular shape, easy to place and route

Disadvantages of 4×4 Array Multiplier

- The disadvantages of array multiplier are as follows,
- High power consumption
- More digital gates resulting in large areas.

Reference: Book/Websites

Computer System Architecture: Morris Mano

Computer Organization and Architecture: Sudam Pawar

www.geeksforgeeks.org

www.javatpoint.com

www.learncomputerscienceonline.com