UNIT – II

6. ARITHMETIC

6.1 Number Representation

Computer systems represent number in terms binary values. I.e. in terms of bit values 0's and 1's. With type of representation, it is classified into two types:

1. Unsigned number
2. Signed number

Unsigned Number: This type of number system is capable of representing only the positive numbers and there is no method to represent negative numbers. All n bits present in this system correspond to magnitude of a positive number.

Represents range from 0 to $2^n - 1$ values i.e. a 4 bit number can represent 0 to 15 different values.

Signed Number: This type number system can represent both positive and negative number. Based on the type of representation, it is again classified into three types:

1. Sign magnitude
2. 1's Compliment
3. 2's Compliment

Signed magnitude number: In this system n-1 bits are used to represent the values and 1 MSB is used to represent the sign of a number. If MSB is 0, it means it is a positive number. If MSB is 1, it means a negative number.

For n bits, value ranges from $-2^{n-1} - 1$ to $+2^{n-1} - 1$. For example if there are 4 bits, it can represent the values ranging from -7 to +7.

+7 → 0111 -7 →1111 +0 →0000 -0→ 1000

1's Compliment: In this system negative number is represented as a compliment to positive number.

For n bits, value of the number ranges from $-2^{n-1} - 1$ to $+2^{n-1} - 1$. For example if there are 4 bits, it can represent the values ranging from -7 to +7.

For example:

+7 →0111 -7 →1000 +3→0011 -3→1100

2's Compliment: In this number system, positive number is same as that of sign magnitude representation and negative number is obtained by adding 1 to 1's compliment of a positive number.

For n bits, value of the number ranges from -2^{n-1} to $+2^{n-1} - 1$. For example if there are 4 bits, it can represent the values ranging from -8 to +7.

For example:

1→0001 -1→1110 (1's compliment) + 1 = 1111

7→0111 -7→1000 (1's compliment) + 1 = 1001

6.2 Addition and subtraction of signed numbers

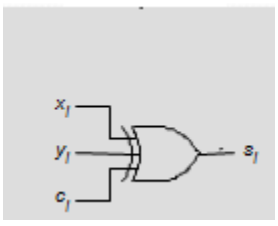
Below shown logical truth table gives a clear picture on the implementation of full adder unit that performs the addition on two operands X_i, Y_i and one carry C_i generating two outputs Sum Si and Carry C_{i+1}.

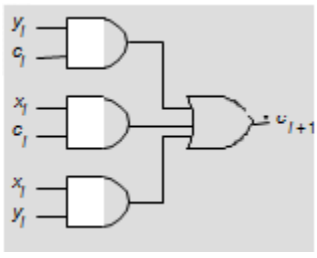
x_i	y_i	Carry-in c_i	Sums s_i	Carry-out c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

According to the truth table, following equations for Sum S_i and carry C_{i+1} can be generated as show below:

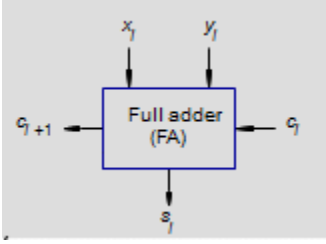
$$s_i = \bar{x}_i\bar{y}_i c_i + \bar{x}_i y_i \bar{c}_i + x_i \bar{y}_i \bar{c}_i + x_i y_i c_i = x_i \oplus y_i \oplus c_i$$
$$c_{i+1} = y_i c_i + x_i c_i + x_i y_i$$

Logical circuit for Sum and carry is represented as shown below:





A single stage Adder having three inputs and two outputs is represented as:

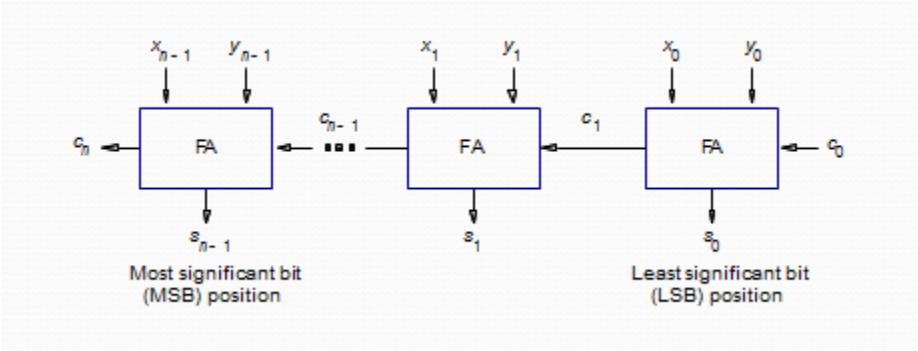


6.2.1 n-bit ripple carry adder

→It is a cascaded connection of n bit full adders put together which can add two n bit numbers with input values x, y, c and producing output S and C.

→For performing the operation, since carries must be passed from one stage to another stage in the sequence, it is called as **ripple carry adder.** Since n adder bits serially connected to one another, they are also called as **serial adders.**

Below is a typical representation of ripple carry adder which is an n bit adder:

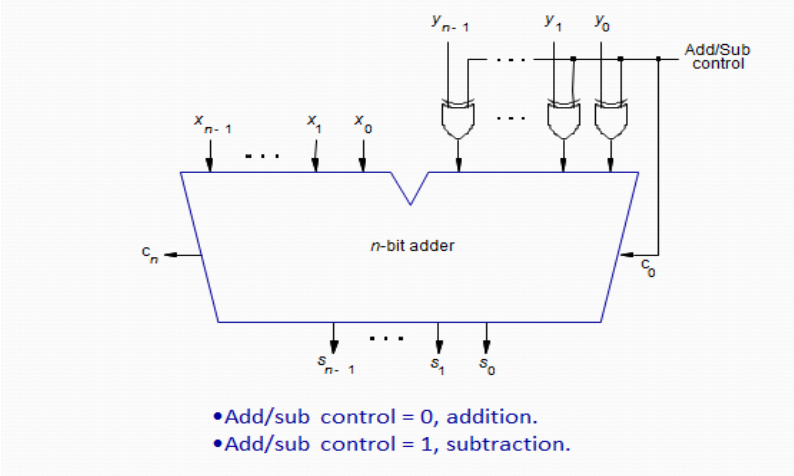


Delay in this type of adder to generate **sum** is 2n-1 and **carry** is 2n

Note: To design n bit subtractor, initial carry C_o in the above circuit is set to 1 and the compliment of second operand bits are passed so that each bit of the operand Y is converted to 2s compliment by adding 1 to 1s compliment.

6.2.2 Addition/ Subtraction Logic unit

Instead of designing two different circuits for doing addition and subtraction operation, both addition and subtraction is done in a single circuit which is called as adder/ subractor circuit. To get the clear understanding of the working procedure of this circuit, let us look into the circuit diagram that is mentioned below:



Above shown circuit is the one that performs both addition and subtraction operation. The circuit is again made of n full adders which facilitates the addition/subtraction operation of n bits

To perform both addition and subtraction in this single unit, circuit has some logics implemented:

1. There are n Full adders to perform n bit addition/subtraction. Each Full adder has three inputs: x_i, y_i and c_i and two outputs S_i and C_{i+1}. Initial carry is connected to first full adder on the LSB side
2. x_i is sent into one of the input of Full adder directly as it is and the other bit y_i is sent into another input of the full adder through two input xor gate. The other input for the xor gate is connected to carry C_0 in common.
3. When C_0=0, it performs addition of two operands. Whenever any input bit y_i is passed into xor gate along with another input value as 0, the output will be same as that of the input value y_i. When C_0 =1, the circuit performs subtraction by adding 2s compliment of operand Y with operand X. We know that 2s compliment is nothing but 1s compliment+1.So when input bit y_i is passed into xor gate along with another input as 1, output will be the compliment of input value y_i. Since C_0 =1, it adds 1 to the generated 1s compliment yielding 2s compliment of the operand.

Another important aspect of this circuit apart from performing addition/subtraction, it has to determine overflow occurrence in the performed operation. Overflow for any two operand addition/subtraction can occur only in two cases:

1. Both operands involved in the operation must be of same sign.
2. Logically it is determined in above circuit through a gate configuration $Overflow = c_n \oplus c_{n-1}$ i.e. whenever this equation gives the output 1 that means there is an overflow generated in the operation.

6.3 Design of fast adders

In this section we will be discussing about another type of adders called as fast adders. Before discussing fast adders in detail, let us know what is a fast adder?

Basically fast adders are the type of adders that perform addition/subtraction in a faster way when compared to the discussed convectional circuits. There are two approaches to build fast adders:

1. Fastest electronic technology that makes the functioning in very negligible amount of time.
2. Usage of large circuits that can work in parallel. Here a different logic is used for the existing circuit configuration so that it can make the gates to work in parallel in different stages.

Next we will see what is the disadvantage of the discussed ripple carry adder?

When we consider ripple carry adder there will be increasing delay as the number of bits in the operation increases. I.e. for **sum,** it has **2n-1 gates delay** and for **carry**, it has **2n gate delays** which is a very large number when a large number of bits are involved in the operation. To overcome this disadvantage, there has to be some method were in delays remain constant for any number of bits in the operand. Hence the intension of fast adder is to keep delay constant for any operation which basically overcomes the disadvantage of ripple carry adder.

6.3.1 Carry Look Ahead Adder (Parallel adder)

One of the types of adder circuit that belongs to the class of Fast adders implementing the technique that makes all the gates to work in parallel. Hence the adders are also called as **Parallel adders.**

The adder circuit overcomes the disadvantage of ripple carry adder were in a large number of delay is caused due to rippling of carries from one stage to another stage. In this technique carries are independently generated in parallel without having to ripple from one adder to another adder.

To fasten up the generation of carry signal and to reduce the delay, alterations to the logical expression are done. We know that below mentioned equations are the logical expressions for sum and carry respectively.

$$s_i = x_i \oplus y_i \oplus c_i \quad \text{——————} 1$$

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i \quad \text{——} 2$$

Factoring equation 2

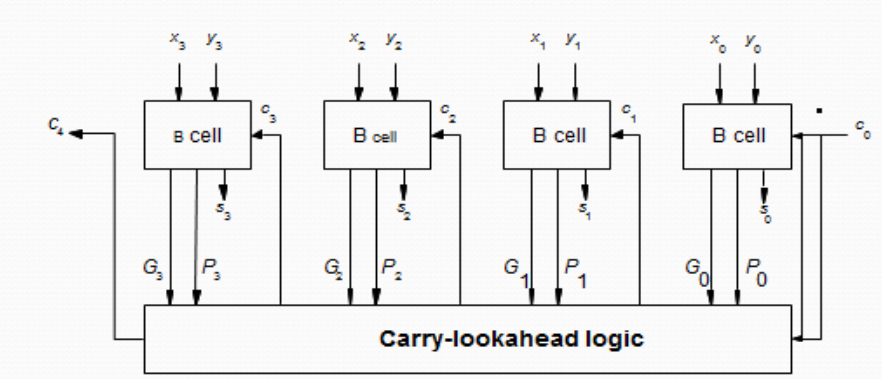
$$c_{i+1} = x_i y_i + (x_i + y_i) c_i$$

The factored equation mentioned in the previous page can be written as:

$$c_{i+1} = G_i + P_i c_i$$
$$where\ G_i = x_i y_i\ and\ P_i = x_i + y_i$$

G_i is called generate function and P_i is called propagate function. Generate equal to 1 means carry C_{i+1} is 1 independent of carry from previous stage. Propagate equal to 1 means C_{i+1} becomes 1 provided it may depend on the previous stage carry.

According to the derived equation, for each stage (Bit cell) a small change is made in terms of the circuit configuration were in each bit cell is having three inputs x_i, y_i and c_i and has three outputs one of them is sum s_i and the other two are generation G_i and propagation P_i. Below shown is the circuit configuration for the 4 bit carry look ahead accordingly which consists 4 adders:



In this circuit, all the bit values x_i, y_i and initial carry c_o are independent entities available. Since G and P are derived from x_i and y_i even G_i and P_i operate independently on all the B cell. Hence the key idea in this logic is to make all the B cell work independently without any dependence for the carry from previous stage (B cell) which makes all the B cell to work in parallel generating both sum and carry. So equation for all the B cell is derived of independent entities G, P and C_0.

Let us write the equation in terms of G, P and C_0 for the carry look ahead circuit that is shown in the above figure. The above circuit consists of four carries C_1, C_2, C_3 and C_4 respectively. Let us write the equation for all those carries in terms of G, P and C_0 as shown below:

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1 = G_1 + P1\ (G0 + P0C0) = G_1 + P_1G_0 + P_1P_0C_0$

$C_3 = G2 + P2C2 = G_2 + P2\ (G_1 + P_1G_0 + P_1P_0C_0) = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

$C_4 = G_3 + P_3C_3 = G_3 + P_3\ (G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0) = G_3 + P_3G2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$

Now when you observe above written equation very closely all the final equations generated for carry are in terms of independent elements G, P and C_0. Hence all those gates forming those equations have the ability to operate in parallel without waiting for any carry from the previous bit.

Next we will calculate the delay factor in this adder and lets compare it with the delay that may be caused in a serial adder:

According to the above written equation all the carries are And or Network of G, P and C_0. Hence the delay in the above case is caused in three different steps:

1. In the first step it generates all $G=x_iy_i$ and $P=x_i+y_i$ in parallel taking only **1** gate delay.
2. In the second step, once after getting G and P in parallel, it generates all the carries from C_1 to C_4 in parallel. The carries are formed of And- Or network which requires **2** gate delays by default. So all carries are generated within **3** gate delays in parallel.
3. Finally once after getting the all the carries, sum for all the bits can be generated in parallel with another **1** gate delay after generating carries. So Sum is derived in 4 gate delays for all the bits in parallel

Finally to conclude, Carry requires 3 gate delays and Sum requires 4 gate delays.

Comparison with ripple carry adder:

For ripple carry adder, it takes 2n-1 gate delays for n bit sum and 2n delays for n bit carry. Hence for 4 bit ripple carry adder, it requires **2*4-1 = 15** gate delays for sum and **2*4=16** gate delays for generating all the carries.

But in case of carry look ahead adder it is just **3** gate delay for all carry bits and **4** gate delay for all the bits of sum.

Problem for generating carries of higher order through this method:

Performing *n*-bit addition in 4 gate delays independent of *n* is good only theoretically because practically it gives raise to fan-in constraints. As you might have observed for generating C_4, the equation is:

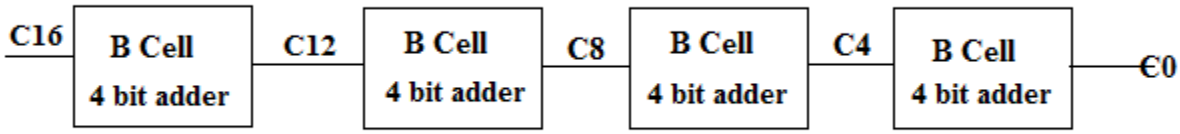
$$\mathbf{C_4 = G_3 + P_3G2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0}$$

As you can observe in the above equation that there is a Fan in of 5 gates for the last set of And gates which is practically maximum fan in and cannot be extended further. So at the maximum only 4 bit parallel adder can be implemented.

Hence to generate carry through this method, it requires block of 4 bits to be connected serially were in higher order carry C_4, C_8, C_{12} … etc. are rippled into next block. In that case it will be 2n+1 gate delays for all carries and 2n+2 gate delays for the entire sum because all G and P are

generated in one gate delay and 2 gate delays for generating the carries of each block since the higher block has to wait for the carry coming in previous block. So if there are n bits, there will be 2n gate delays. After this 1 gate delay of extra is required for generating the entire sum in parallel.

Typically a 16 bit adder of this type is represented in the schematic as shown below:



16 bit ripple carry adder using four 4 bit carry look ahead adder in block

The delay in this case is 9 gates for carry and 10 gates for sum.

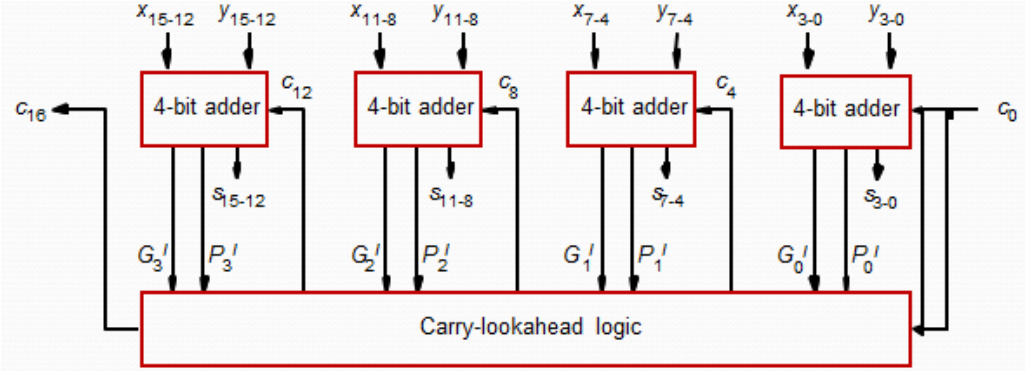
6.3.2 Blocked Carry-Lookahead adder

First let us know what is a blocked carry – lookahead adder and then we try to analyze its implementation logic and the advantage of it.

As we have seen in the last section that carries can be generated for every single bit of an adder in parallel. But due to a practical restriction of Fan in constraint the parallel adder scope was limited only to 4 bits. Because of this to generate a higher order adder other than 4 bit we have rippled the 4 bit blocks and connected serially one after the other. But the major disadvantage of this adder is again a delay factor were in higher block cannot start its functioning till it receives the carry from the lower block as the carry coming from the previous block acts as a initial carry for all the bits of each block.

To overcome this disadvantage, a logic of carry look ahead adder is used were in instead of generating the carries for bits in parallel, at the outset higher order carries are generated in parallel. I.e. the carries C_4, C_8, C_{12}... are generated in parallel at the first level. Later all other carries are generated. This is because we know that the higher order carries are fed as initial carry to the next block. So this is the one that is not generated in parallel and causing the delay.

To get the better understanding of this, let us directly look into the circuit configuration of 16 bit adder implemented through four 4 bit locked carry lookahead- adder that is shown in the next page.



The above circuit looks almost similar to the carry lookahead- adder discussed in the last section. The only difference is that the adder discussed in the last section was generating carry in parallel for each and every bits. But this circuit generates carry with respect to the blocks in parallel which is called as higher order carry for each block.

First we will know the configuration of the above circuit and let us understand its logic:

1. Above shown circuit diagram is a 16 bit adder having 16 sum and 16 carry outputs for 16 x and 16 y bits
2. There are four 4 bit block adders were each block has 4 Full adders giving rise to total of 16 Full adders in the connection.
3. Each block generates four sum, four carries, four generation and four propagation functions all of which are independent.
4. The last bit carries generated by each of the block is called as higher order carries. According to this figure higher order carries are C_4, C_8, C_{12} and C_{16} respectively.
5. In the last section we have seen that these higher order carries are the one that was causing delay. So the idea in this circuit is to generate those higher order carries C_4, C_8, C_{12} and C_{16} in parallel so that all the carries can be generated without having to wait for the block to complete its operation.
6. To implement the parallel generation of higher order carries logically, all the generations and propagation functions G and P of the block are clubbed together as $G^|$ and $P^|$ respectively which gives four $G^|$ functions $G_0^|$, $G_1^|$, $G_2^|$, $G_3^|$, $P_0^|$, $P_1^|$, $P_2^|$ and $P_3^|$ respectively as shown in the figure. We will see how exactly this $G^|$ and $P^|$ can be derived logically.

Consider the higher order carry C_4 and write the logical expression in terms of the value derived in last section which is:

$$C_4 = \underbrace{G_3 + P_3G2 + P_3P_2G_1 + P_3P_2P_1G_0}_{G_0'} + \underbrace{P_3P_2P_1P_0}_{P_0'}C_0$$

Substituting $G_0^| = \mathbf{G_3 + P_3G2 + P_3P_2G_1 + P_3P_2P_1G_0}$ and $P_0^| = \mathbf{P_3P_2P_1P_0}$ gives us the equation: $\mathbf{C_4 = G_0^| + P_0^| C_0}$ were in all $G_0^|$, $P_0^|$ and C_0 are all independent elements.

Steps for generating sum and carry in blocked carry- lookahead adder:

1. **Generation of higher order carry:**
   In step 1, all the higher order carries i.e. C_4, C_8, C_{12} and C_{16} are generated with respect to $G^|$, $P^|$ and C_0 all of which are independent elements as shown below:
   $\mathbf{C_4 = G_0^| + P_0^| C_0}$
   $\mathbf{C_8 = G_1^| + P_1^| C_4 = G_1^| + P_1^| (G_0^| + P_0^| C_0) = G_1^| + P_1^| G_0^| + P_1^| P_0^| C_0}$
   $\mathbf{C_{12} = G_2^| + P_2^| C_8 = G_2^| + P_2^| (G_1^| + P_1^| G_0^| + P_1^| P_0^| C_0) = G_2^| + P_2^| G_1^| + P_2^| P_1^| G_0^| + P_2^| P_1^| P_0^| C_0}$
   $\mathbf{C_{16} = G_3^| + P_3^| C_{12} = G_3^| + P_3^| (G_2^| + P_2^| G_1^| + P_2^| P_1^| G_0^| + P_2^| P_1^| P_0^| C_0)}$
   $\mathbf{= G_3^| + P_3^| G_2^| + P_3^| P_2^| G_1^| + P_3^| P_2^| P_1^| G_0^| + P_3^| P_2^| P_1^| P_0^| C_0}$
   In the above written equation, all the elements are independent elements hence they can operate in parallel without have to depend on any of the previous generated output. This is the very first step were in we generate all the higher order carries so that once after this step all the blocks can work in parallel.
   Delay factor:
   Next we will see what is the total delay factor for generating all the above mentioned carries in parallel:
   i. There is an and or network combination of $G^|$ and $P^|$ in all the expressions which require delay of **2 gates.**
   ii. To derive $G^|$ and $P^|$ which is again a combination of generations and propagations as shown:
   $G_0^| = \mathbf{G_3 + P_3 G2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0}$ and $P_0^| = \mathbf{P_3 P_2 P_1 P_0}$
   The above shown expression again has a and or network for G and AND gate for P which can work in parallel and require delay of **2 gates**
   iii. Finally G and P are the combination of bit value x and y which is derived in parallel in delay of **1 gate.**

   To conclude this all higher order carries are generated in 5 gate delays in parallel

2. **Generation of intermediate carry:**
   In step 2, once after getting higher order carries required for the block, all intermediate carries are generated in parallel in another delay of **2 gates** in parallel as discussed in previous section.

3. **Generation of sum:**
   Final step is generation of Sum which is generated in parallel with another delay of **1 gate** for all the bits.

Conclusion: To conclude, all carries are generated in 5+2=7 gate delays and all sum are generated in 5+2+1 = 8 gate delays.

6.4 Multiplication of Positive numbers (Multiplication of unsigned numbers)

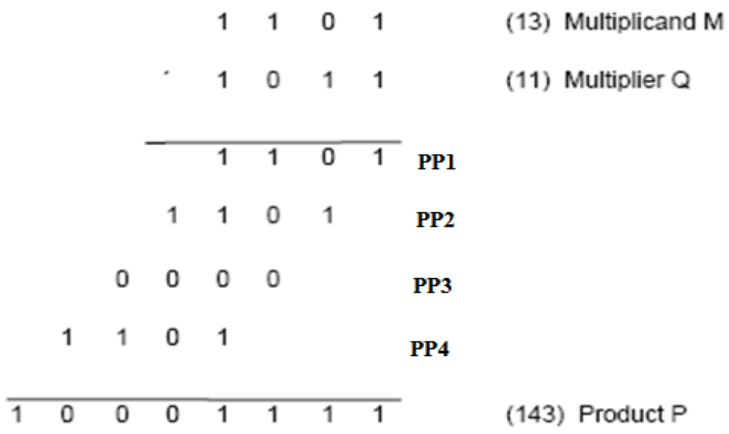
First we will understand how two unsigned numbers are multiplied manually for the binary numbers. Later we will extend this knowledge in terms circuit generation. When it is unsigned, there will be no sign bit and all n bits represent the magnitude of a number.

6.4.1 Manual Multiplication

Consider multiplication of two number 13X11 were 13 is Multiplicand represented by the symbol M and 11 is multiplier represented by the symbol Q.

When doing the multiplication in binary system, you need to do multiplication of two binary numbers of equal bit length **n** yielding the result of **2n** bits. To do this first you need to consider the number with higher value, represent it in binary which yields n bits. The lower number should also be represented in n bits itself. Then the final product will be in terms of 2n bits.

In our example 13 is greatest number, first represent 13 in binary which gives 1101 represented in 4 bits. Next represent 11 in 4 bits which is represented as 1011. Multiplication is done as shown below:



Observations:

1. When nxn bits are multiplied it results the product in terms of 2n bits. In our example for two 4 bit number multiplications, result is generated 8 bits product.
2. For nxn bit multiplication, its generates **n** partial products(pp). In the above example it's a 4 bit multiplication which gave rise to four partial products.
3. Each partial product is the combination of n bits generated by **and** operation of selected multiplier bit with all the multiplicand bits. This bits are called as summands. First partial product is selected multiplier bit from LSB, second partial product is the selected

multiplier bit which is next higher bit and so on up to MSB of the multiplier. In our example, it is a 4 bit multiplication which generates 4 partial products PP1, PP2, PP3 and PP4.

4. Any partial product generated after PP1 is shifted left by one bit. So when the final PP is generated, it is shifted by n-1 bits to the left.
5. To get the final product, all the partial products are added together which yields a result of 2n bit products as shown in the above calculations

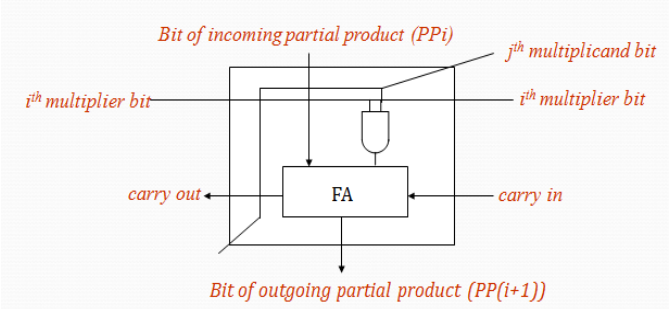
6.4.2 Array Multiplication of Positive operands

In the last section we have seen the method of the multiplication of unsigned binary operands manually. Based on the method that was discussed in the previous section, a similar kind of combinatorial multiplier circuit is being implemented which is called as **Array multiplication of positive operands.**

Following are the design properties of the circuit which gives its working method:

1. When you observed the manual multiplication method there are n partial products and nxn summand bits were each summand bit is a single cell performing some logical functioning. In the similar way, an n bit multiplication array circuit has nxn cells for producing nxn summands.
2. In the manual multiplication method, final product is achieved by adding all the generated partial product bits at the end. Instead of that in the array multiplication logic, partial products are added at each stage as and when it is generated, I.e. generated partial product bit is added to previous partial product bit.
3. Every generated partial product is handed over to the next stage partial product as an input value.
4. Initial value of the partial product will be zero when it is added to the very first generating partial product.

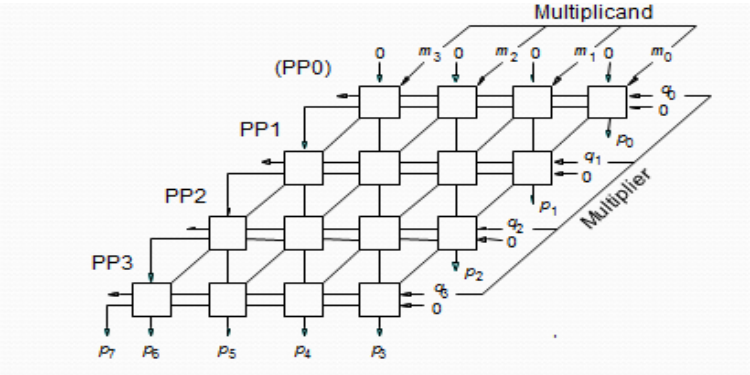
According to the properties mentioned, a single multiplication cell is represented as shown below:



The above cell consists of a Full adder to add the current partial product bit with previous partial product bit. Each Full adder takes three inputs, one is the previous partial product bit (PP_{i-1}),

another one is the current partial product bit(PP_i) and the third one is the carry rippled from the previous bit of partial product (PP_i) and produces two outputs which next level partial product bit (PP_{i+1}) and ripple carry C_{i+1}. An And gate is used to generate each bit of the summand.

A typical combination array multiplier of 4 bits is shown below according to the mentioned properties:



There are some major disadvantages in the implementation part of this circuit apart from its functioning on the positive operands:

1. One of them is there is a large number of gates arranged in the circuit and as the number of bits increases number of cells also increases. For example if it is a 64 bit multiplication there will be 4096 cells which is a very huge number and has a huge amount of delay.
2. Another disadvantage is that when the circuit becomes such a large unit it proves to be very inefficient.

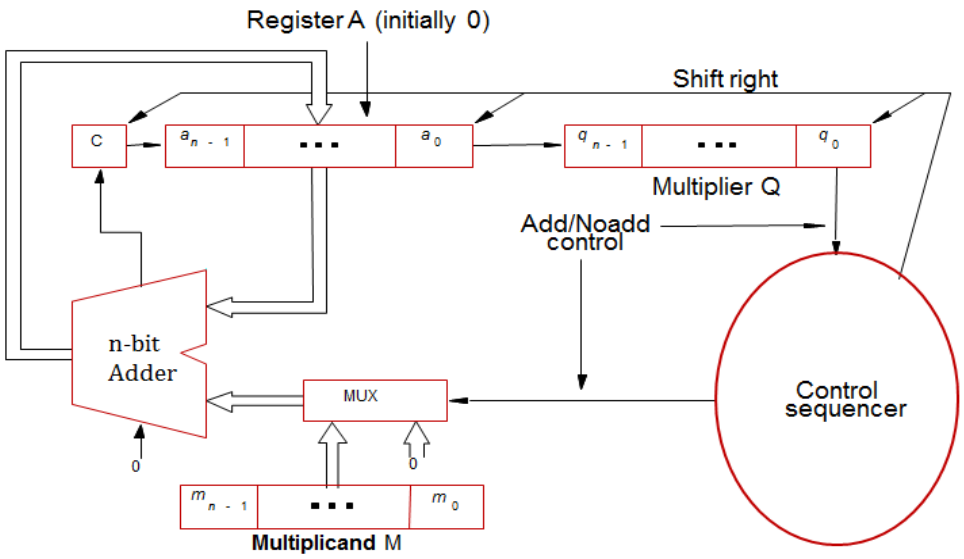
6.4.3 Sequential Multiplication for positive operands

To overcome the disadvantage of the array multiplication circuit discussed in the previous section, one of the techniques is the implementation of **sequential multiplier** circuit. Since the multiplication is done sequentially it is called so.

The sequential multiplier circuit has the following design properties:

1. Instead of using nxn adders, in this circuit, n adders are used n times to generate the summands.
2. In this logic, instead of shifting new partial product(PP_i) to the left and adding it to the un shifted previous partial product(PP_{i-1}), old partial product PP_{i-1} is shifted to the right and added to new un shifted partial product PP_i.

According to the above discussed properties, a typical sequential multiplier circuit is shown in the next page:



Sequential Multiplier Circuit

Following are the components of the circuit design and their functionality:

1. The circuit design has three registers and one flip flop:

i. Register A: It is an n bit register for n bit multiplication which has all zeroes stored in them initially indicates the very first partial product. This register is used to store the generated partial product every time when it is added with the previous partial product. This register is connected to n-bit adder. After the completion of computing product, this register stores the first half of the product.

ii. Register Q: It is an n bit register for n bit multiplication. This register is used to store n bit multiplier initially. After computing the product, this register stores second half of the product value. Register A and Register Q is used together to store **2n** bit product of **n** bit multiplication. This register is also connected to control sequencer.

iii. Register M: It is an n bit register for n bit multiplication. This register is used to store the n bit multiplicand value which is connected to multiplexer.

iv. Flip flop C: This flip flop is used to store one bit carry that is generated by the partial product. Initially this is loaded with the value 0.

2. Control Sequencer: This unit is the one that detects the multiplier bit value and sends the signal. If the multiplier bit is 1, it sends add signal. If it is 0, its sends no add signal. If the signal is add, it sends the information to the multiplexer to select multiplicand. Once after the

generating the required signal it even takes the responsibility sending right shift signal to C, A & Q registers.

3. Mux: It is a 2:1 multiplexer which selects multiplicand as adder input if the signal form control sequencer is 1 or else selects bit values 0 if the signal from control sequencer is 0.

4. n-bit adder: It is an n bit full adder that is used to add generated partial product with the previous shifted partial product sends the result into the register A. It takes one input from the register A and another input from the multiplexer in terms of either multiplicand or zero bits.

Working of the circuit: Circuit shown above performs the n bit multiplication in n number of steps which are called as cycles of multiplication with each cycle has a defined procedure and the components of the circuit works according to that resulting in 2n bit product in n cycles. For example if it is a 4 bit multiplication, there will be four different cycles to produce 8 bit product

Above represented circuit works according to the following steps:

1. Initially circuit loads register A with all zeroes, register Q with the multiplier value and register M with multiplicand value.
2. Next is it starts performing the operation from cycle 1 to cycle n and in every cycle, it performs two steps:
   i. In the first step, it detects the LSB of the multiplier bit and if the LSB of the multiplier bit is 1, it adds Multiplicand with the contents of register A and stores the generated result back in register A or else if it is 0, it doesn't perform addition function.
   ii. In the second step once after completing step1, it shifts the contents of register A & Q to the right exposing next multiplier bit into LSB pushing out the previous multiplier bit out of the register Q.

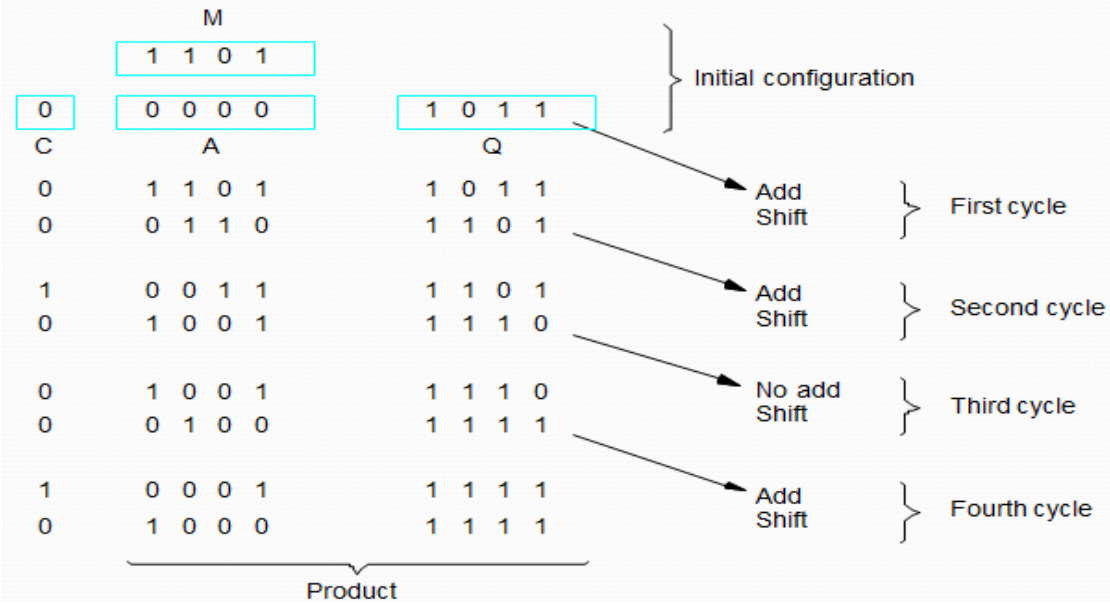
Working of the circuit according to the above mentioned step can be best understood through solving a problem mentioned below:

Example: Let us consider a problem of multiplying 13 and 11. I.e.13X11.

Here in this problem, 13 is multiplicand M and 11 is multiplier Q. First let us determine number bits in the multiplication process.

To select the number of bits, consider the largest value operand, here Multiplicand 13 is largest value. Hence represent it in binary value: **13(M)→1101(M)** which is four bits. So represent the other value in binary i.e. **11(Q)→1011(Q).** According to this, produced result should be in 8 bits.

Solution to this problem is shown in the next step which is done according to the above mentioned steps.



Exercise : Try to implement 15 X 14 using sequential multiplication method.