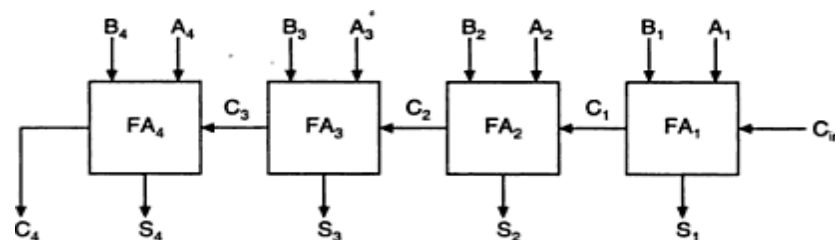


Parallel Adders

Binary Parallel Adder:

A binary parallel adder is a digital circuit that adds two binary numbers in parallel form and produces the arithmetic sum of those numbers in parallel form. It consists of full adders connected in a chain, with the output carry from each full-adder connected to the input carry of the next full-adder in the chain.

The interconnection of four full-adder (FA) circuits to provide a 4-bit parallel adder. The augends bits of A and addend bits of B are designated by subscript numbers from right to left, with subscript 1 denoting the lower-order bit. The carries are connected in a chain through the full-adders. The input carry to the adder is C_{in} and the output carry is C_4 . The S output generates the required sum bits. When the 4-bit full-adder circuit is enclosed within an IC package, it has four terminals for the augends bits, four terminals for the addend bits, four terminals for the sum bits, and two terminals for the input and output carries. An n-bit parallel adder requires n-full adders. It can be constructed from 4-bit, 2-bit and 1-bit full adder ICs by cascading several packages. The output carry from one package must be connected to the input carry of the one with the next higher-order bits. The 4-bit full adder is a typical example of an MSI function.



Logic diagram of a 4-bit binary parallel adder.

Ripple carry adder:

In the parallel adder, the carry-out of each stage is connected to the carry-in of the next stage. The sum and carry-out bits of any stage cannot be produced, until sometime after the carry-in of that stage occurs. This is due to the propagation delays in the logic circuitry,



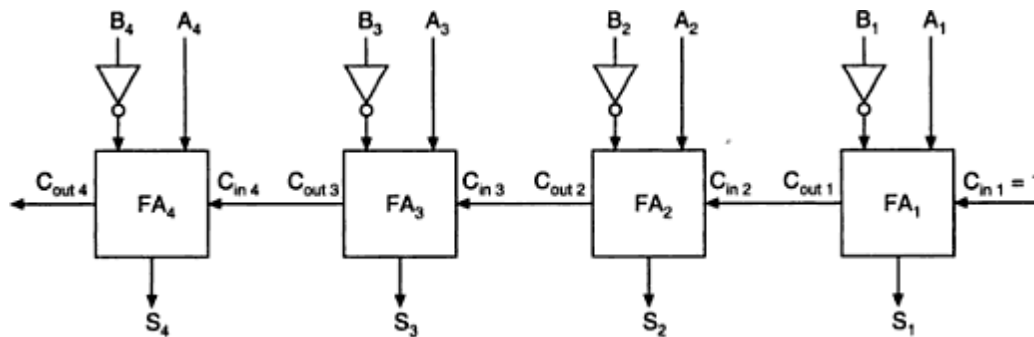
which lead to a time delay in the addition process. The carry propagation delay for each full-adder is the time between the application of the carry-in and the occurrence of the carry-out.

The 4-bit parallel adder, the sum (S_1) and carry-out (C_1) bits given by FA_1 are not valid, until after the propagation delay of FA_1 . Similarly, the sum S_2 and carry-out (C_2) bits given by FA_2 are not valid until after the cumulative propagation delay of two full adders (FA_1 and FA_2), and so on. At each stage, the sum bit is not valid until after the carry bits in all the preceding stages are valid. Carry bits must propagate or ripple through all stages before the most significant sum bit is valid. Thus, the total sum (the parallel output) is not valid until after the cumulative delay of all the adders.

The parallel adder in which the carry-out of each full-adder is the carry-in to the next most significant adder is called a ripple carry adder. The greater the number of bits that a ripple carry adder must add, the greater the time required for it to perform a valid addition. If two numbers are added such that no carries occur between stages, then the add time is simply the propagation time through a single full-adder.

4- Bit Parallel Subtractor:

The subtraction of binary numbers can be carried out most conveniently by means of complements, the subtraction $A-B$ can be done by taking the 2's complement of B and adding it to A . The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits. The 1's complement can be implemented with inverters as



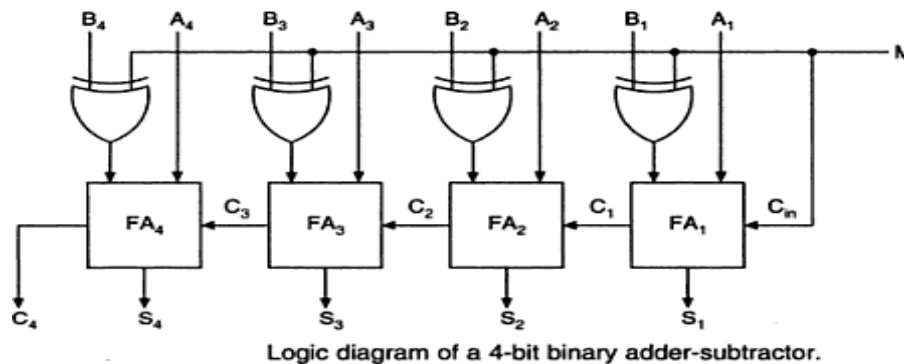
Logic diagram of a 4-bit parallel subtractor.

Binary-Adder Subtractor:

A 4-bit adder-subtractor, the addition and subtraction operations are combined into one circuit with one common binary adder. This is done by including an X-OR gate with each full-adder. The mode input M controls the operation. When $M=0$, the circuit is an adder, and when $M=1$, the circuit becomes a subtractor. Each X-OR gate receives input M and one of the inputs of B . When $M=0$, $B \oplus 0 = B$. The full-adder receives the value of B , the input carry is 0



and the circuit performs $A+B$. when $B \oplus 1 = B'$ and $C_1=1$. The B inputs are complemented and a 1 is through the input carry. The circuit performs the operation A plus the 2's complement of B.

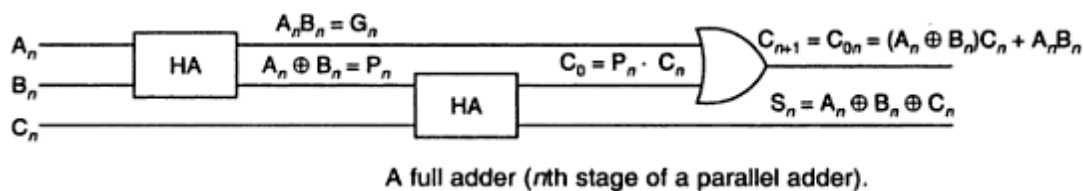


The Look-Ahead –Carry Adder:

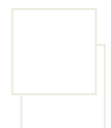
In parallel-adder, the speed with which an addition can be performed is governed by the time required for the carries to propagate or ripple through all of the stages of the adder. The look-ahead carry adder speeds up the process by eliminating this ripple carry delay. It examines all the input bits simultaneously and also generates the carry-in bits for all the stages simultaneously.

The method of speeding up the addition process is based on the two additional functions of the full-adder, called the carry generate and carry propagate functions.

Consider one full adder stage; say the n th stage of a parallel adder as shown in fig. we know that is made by two half adders and that the half adder contains an X-OR gate to produce the sum and an AND gate to produce the carry. If both the bits A_n and B_n are 1s, a carry has to be generated in this stage regardless of whether the input carry C_{in} is a 0 or a 1. This is called generated carry, expressed as $G_n = A_n \cdot B_n$ which has to appear at the output through the OR gate as shown in fig.



There is another possibility of producing a carry out. X-OR gate inside the half-adder at the input produces an intermediary sum bit- call it P_n –which is expressed as $P_n = A_n \oplus B_n$. Next P_n and C_n are added using the X-OR gate inside the second half adder to produce the final



sum bit and $S_n = P_n \oplus C_n$ where $P_n = A_n \oplus B_n$ and output carry $C_0 = P_n \cdot C_n = (A_n \oplus B_n) C_n$ which becomes carry for the (n+1) th stage.

Consider the case of both P_n and C_n being 1. The input carry C_n has to be propagated to the output only if P_n is 1. If P_n is 0, even if C_n is 1, the and gate in the second half-adder will inhibit C_n . the carry out of the nth stage is 1 when either $G_n=1$ or $P_n \cdot C_n=1$ or both G_n and $P_n \cdot C_n$ are equal to 1.

For the final sum and carry outputs of the nth stage, we get the following Boolean expressions.

$$S_n = P_n \oplus C_n \text{ where } P_n = A_n \oplus B_n$$

$$C_{on} = C_{n+1} = G_n + P_n C_n \text{ where } G_n = A_n \cdot B_n$$

Observe the recursive nature of the expression for the output carry at the nth stage which becomes the input carry for the (n+1)st stage .it is possible to express the output carry of a higher significant stage is the carry-out of the previous stage.

Based on these , the expression for the carry-outs of various full adders are as follows,

$$C_1 = G_0 + P_0 \cdot C_0$$

$$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$$

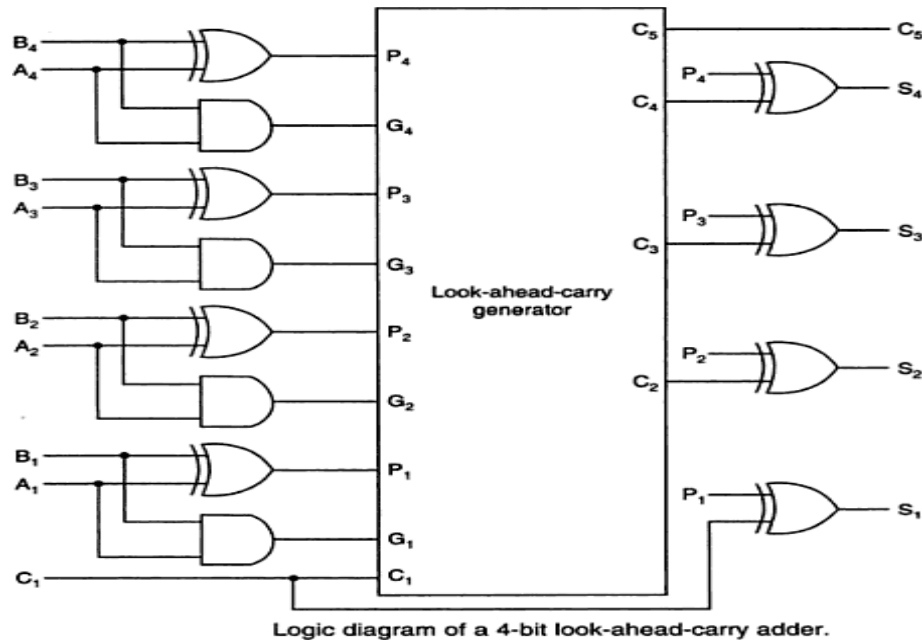
$$C_3 = G_2 + P_2 \cdot C_2 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

$$C_4 = G_3 + P_3 \cdot C_3 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

The general expression for n stages designated as 0 through $(n - 1)$ would be

$$C_n = G_{n-1} + P_{n-1} \cdot C_{n-1} = G_{n-1} + P_{n-1} \cdot G_{n-2} + P_{n-1} \cdot P_{n-2} \cdot G_{n-3} + \dots + P_{n-1} \cdot \dots \cdot P_0 \cdot C_0$$

Observe that the final output carry is expressed as a function of the input variables in SOP form. Which is two level AND-OR or equivalent NAND-NAND form. Observe that the full look-ahead scheme requires the use of OR gate with (n+1) inputs and AND gates with number of inputs varying from 2 to (n+1).



2's complement Addition and Subtraction using Parallel Adders:

Most modern computers use the 2's complement system to represent negative numbers and to perform subtraction operations of signed numbers can be performed using only the addition operation ,if we use the 2's complement form to represent negative numbers.

The circuit shown can perform both addition and subtraction in the 2's complement. This adder/subtractor circuit is controlled by the control signal ADD/SUB'. When the ADD/SUB' level is HIGH, the circuit performs the addition of the numbers stored in registers A and B. When the ADD/Sub' level is LOW, the circuit subtract the number in register B from the number in register A. The operation is:

When ADD/SUB' is a 1:

1. AND gates 1,3,5 and 7 are enabled , allowing B_0, B_1, B_2 and B_3 to pass to the OR gates 9,10,11,12 . AND gates 2,4,6 and 8 are disabled , blocking $B_0', B_1', B_2',$ and B_3' from reaching the OR gates 9,10,11 and 12.
2. The two levels B_0 to B_3 pass through the OR gates to the 4-bit parallel adder, to be added to the bits A_0 to A_3 . The sum appears at the output S_0 to S_3
3. Add/SUB' =1 causes no carry into the adder.

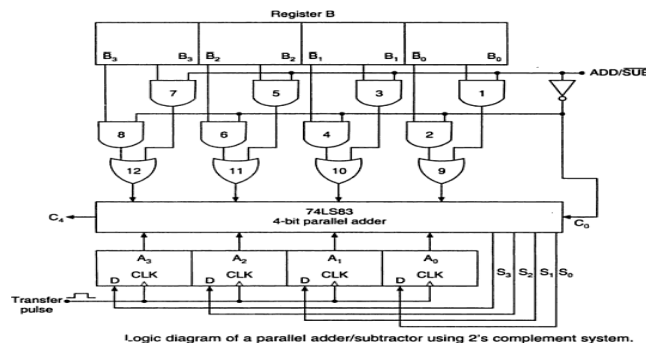
When ADD/SUB' is a 0:

1. AND gates 1,3,5 and 7 are disabled , allowing B_0, B_1, B_2 and B_3 from reaching the OR gates 9,10,11,12 . AND gates 2,4,6 and 8 are enabled , blocking $B_0', B_1', B_2',$ and B_3' from reaching the OR gates.



2. The two levels B_0' to B_3' pass through the OR gates to the 4-bit parallel adder, to be added to the bits A_0 to A_3 . The C_0 is now 1. Thus the number in register B is converted to its 2's complement form.
3. The difference appears at the output S_0 to S_3 .

Adders/Subtractors used for adding and subtracting signed binary numbers. In computers, the output is transferred into the register A (accumulator) so that the result of the addition or subtraction always ends up stored in the register A. This is accomplished by applying a transfer pulse to the CLK inputs of register A.



Serial Adder:

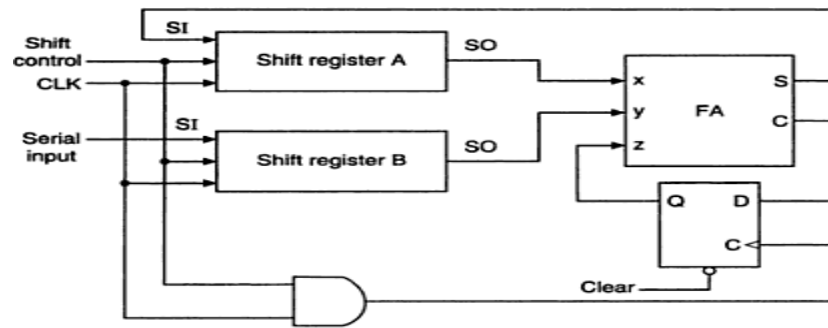
A serial adder is used to add binary numbers in serial form. The two binary numbers to be added serially are stored in two shift registers A and B. Bits are added one pair at a time through a single full adder (FA) circuit as shown. The carry out of the full-adder is transferred to a D flip-flop. The output of this flip-flop is then used as the carry input for the next pair of significant bits. The sum bit from the S output of the full-adder could be transferred to a third shift register. By shifting the sum into A while the bits of A are shifted out, it is possible to use one register for storing both augend and the sum bits. The serial input register B can be used to transfer a new binary number while the addend bits are shifted out during the addition.

The operation of the serial adder is:

Initially register A holds the augend, register B holds the addend and the carry flip-flop is cleared to 0. The outputs (SO) of A and B provide a pair of significant bits for the full-adder at x and y. The shift control enables both registers and carry flip-flop, so, at the clock pulse both registers are shifted once to the right, the sum bit from S enters the left most flip-flop of A, and the output carry is transferred into flip-flop Q. The shift control enables the registers for a number of clock pulses equal to the number of bits of the registers. For each succeeding clock pulse a new sum bit is transferred to A, a new carry is transferred to Q, and both registers are shifted once to the right. This process continues until the shift control is disabled. Thus the addition is accomplished by passing each pair of bits together with the previous carry through a single full adder circuit and transferring the sum, one bit at a time, into register A.



Initially, register A and the carry flip-flop are cleared to 0 and then the first number is added from B. While B is shifted through the full adder, a second number is transferred to it through its serial input. The second number is then added to the content of register A while a third number is transferred serially into register B. This can be repeated to form the addition of two, three, or more numbers and accumulate their sum in register A.



Logic diagram of a serial adder.

Difference between Serial and Parallel Adders:

The parallel adder registers with parallel load, whereas the serial adder uses shift registers. The number of full adder circuits in the parallel adder is equal to the number of bits in the binary numbers, whereas the serial adder requires only one full adder circuit and a carry flip-flop. Excluding the registers, the parallel adder is a combinational circuit, whereas the serial adder is a sequential circuit. The sequential circuit in the serial adder consists of a full-adder and a flip-flop that stores the output carry.

BCD Adder:

The BCD addition process:

1. Add the 4-bit BCD code groups for each decimal digit position using ordinary binary addition.
2. For those positions where the sum is 9 or less, the sum is in proper BCD form and no correction is needed.
3. When the sum of two digits is greater than 9, a correction of 0110 should be added to that sum, to produce the proper BCD result. This will produce a carry to be added to the next decimal position.

A BCD adder circuit must be able to operate in accordance with the above steps. In other words, the circuit must be able to do the following:

1. Add two 4-bit BCD code groups, using straight binary addition.



- Determine, if the sum of this addition is greater than 1101 (decimal 9); if it is, add 0110 (decimal 6) to this sum and generate a carry to the next decimal position.

The first requirement is easily met by using a 4-bit binary parallel adder such as the 74LS83 IC. For example, if the two BCD code groups $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are applied to a 4-bit parallel adder, the adder will output $S_4S_3S_2S_1S_0$, where S_4 is actually C_4 , the carry-out of the MSB bits.

The sum outputs $S_4S_3S_2S_1S_0$ can range anywhere from 00000 to 100109 when both the BCD code groups are 1001 (=9). The circuitry for a BCD adder must include the logic needed to detect whenever the sum is greater than 01001, so that the correction can be added in. Those cases, where the sum is greater than 1001 are listed as:

S_4	S_3	S_2	S_1	S_0	Decimal number
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18

Let us define a logic output X that will go HIGH only when the sum is greater than 01001 (i.e., for the cases in table). If examine these cases, see that X will be HIGH for either of the following conditions:

- Whenever $S_4 = 1$ (sum greater than 15)
- Whenever $S_3 = 1$ and either S_2 or S_1 or both are 1 (sum 10 to 15)

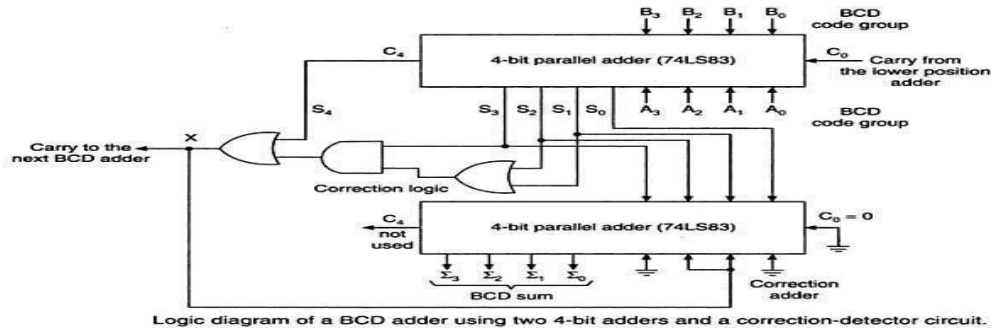
This condition can be expressed as

$$X = S_4 + S_3(S_2 + S_1)$$

Whenever $X = 1$, it is necessary to add the correction factor 0110 to the sum bits, and to generate a carry. The circuit consists of three basic parts. The two BCD code groups $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are added together in the upper 4-bit adder, to produce the sum $S_4S_3S_2S_1S_0$. The logic gates shown implement the expression for X. The lower 4-bit adder will add the correction 0110 to the sum bits, only when $X = 1$, producing the final BCD sum output represented by $\Sigma_3\Sigma_2\Sigma_1\Sigma_0$. The X is also the carry-out that is produced when the sum is greater than 01001. When $X = 0$, there is no carry and no addition of 0110. In such cases, $\Sigma_3\Sigma_2\Sigma_1\Sigma_0 = S_3S_2S_1S_0$.



Two or more BCD adders can be connected in cascade when two or more digit decimal numbers are to be added. The carry-out of the first BCD adder is connected as the carry-in of the second BCD adder, the carry-out of the second BCD adder is connected as the carry-in of the third BCD adder and so on.



EXCESS-3(XS-3) ADDER:

To perform Excess-3 additions,

1. Add two xs-3 codegroups
2. If carry=1, add 0011(3) to the sum of those two codegroups
If carry =0, subtract 0011(3) i.e., add 1101 (13 in decimal) to the sum of those two code groups.

Ex: Add 9 and 5

	1100	9 in Xs-3
	+1000	5 in xs-3

1	0100	there is a carry
+0011	0011	add 3 to each group
-----	-----	
0100	0111	14 in xs-3
(1)	(4)	

EX:

(b)	0 1 1 1	4 in XS-3
	+ 0 1 1 0	3 in XS-3

	1 1 0 1	no carry
	+ 1 1 0 1	Subtract 3 (i.e. add 13)

Ignore carry	1 1 0 1 0	7 in XS-3
	(7)	

Implementation of xs-3 adder using 4-bit binary adders is shown. The augend ($A_3A_2A_1A_0$) and addend ($B_3B_2B_1B_0$) in xs-3 are added using the 4-bit parallel adder. If the carry is a 1, then 0011(3) is added to the sum bits $S_3S_2S_1S_0$ of the upper adder in the lower 4-bit parallel



adder. If the carry is a 0, then 1101(3) is added to the sum bits (This is equivalent to subtracting 0011(3) from the sum bits. The correct sum in xs-3 is obtained

Excess-3 (XS-3) Subtractor:

To perform Excess-3 subtraction,

1. Complement the subtrahend
2. Add the complemented subtrahend to the minuend.
3. If carry =1, result is positive. Add 3 and end around carry to the result . If carry=0, the result is negative. Subtract 3, i.e, and take the 1's complement of the result.

Ex: Perform 9-4

1100	9 in xs-3
+1000	Complement of 4 in Xs-3

(1) 0100	There is a carry
+0011	Add 0011(3)

0111	
1	End around carry

1000	5 in xs-3

The minuend and the 1's complement of the subtrahend in xs-3 are added in the upper 4-bit parallel adder. If the carry-out from the upper adder is a 0, then 1101 is added to the sum bits of the upper adder in the lower adder and the sum bits of the lower adder are complemented to get the result. If the carry-out from the upper adder is a 1, then 3=0011 is added to the sum bits of the lower adder and the sum bits of the lower adder give the result.

Binary Multipliers:

In binary multiplication by the paper and pencil method, is modified somewhat in digital machines because a binary adder can add only two binary numbers at a time. In a binary multiplier, instead of adding all the partial products at the end, they are added two at a time and their sum accumulated in a register (the accumulator register). In addition, when the multiplier bit is a 0, 0s are not written down and added because it does not affect the final result. Instead, the multiplicand is shifted left by one bit.

The multiplication of 1110 by 1001 using this process is

Multiplicand	1110	
Multiplier	1001	
	1110	The LSB of the multiplier is a 1; write down the multiplicand; shift the multiplicand one position to the left (1 1 1 0 0)
	1110	The second multiplier bit is a 0; write down the previous result 1110; shift the multiplicand to the left again (1 1 1 0 0 0)
	0000	
	0000	



+1110000

The fourth multiplier bit is a 1 write down the new multiplicand add it to the first partial product to obtain the final product.

1111110

This multiplication process can be performed by the serial multiplier circuit, which multiplies two 4-bit numbers to produce an 8-bit product. The circuit consists of following elements

X register: A 4-bit shift register that stores the multiplier --- it will shift right on the falling edge of the clock. Note that 0s are shifted in from the left.

B register: An 8-bit register that stores the multiplicand; it will shift left on the falling edge of the clock. Note that 0s are shifted in from the right.

A register: An 8-bit register, i.e., the accumulator that accumulates the partial products.

Adder: An 8-bit parallel adder that produces the sum of A and B registers. The adder outputs S_7 through S_0 are connected to the D inputs of the accumulator so that the sum can be transferred to the accumulator only when a clock pulse gets through the AND gate.

The circuit operation can be described by going through each step in the multiplication of 1110 by 1001. The complete process requires 4 clock cycles.

1. Before the first clock pulse: Prior to the occurrence of the first clock pulse, the register A is loaded with 00000000, the register B with the multiplicand 00001110, and the register X with the multiplier 1001. Assume that each of these registers is loaded using its asynchronous inputs (i.e., PRESET and CLEAR). The output of the adder will be the sum of A and B, i.e., 00001110.

2. First Clock pulse: Since the LSB of the multiplier (X_0) is a 1, the first clock pulse gets through the AND gate and its positive going transition transfers the sum outputs into the accumulator. The subsequent negative going transition causes the X and B registers to shift right and left, respectively. This produces a new sum of A and B.

3. Second Clock Pulse: The second bit of the original multiplier is now in X_0 . Since this bit is a 0, the second clock pulse is inhibited from reaching the accumulator. Thus, the sum outputs are not transferred into the accumulator and the number in the accumulator does not change. The negative going transition of the clock pulse will again shift the X and B registers. Again a new sum is produced.

4. Third Clock Pulse: The third bit of the original multiplier is now in X_0 ; since this bit is a 0, the third clock pulse is inhibited from reaching the accumulator. Thus, the sum outputs are not transferred into the accumulator and the number in the accumulator does not change. The negative going transition of the clock pulse will again shift the X and B registers. Again a new sum is produced.

5. Fourth Clock Pulse: The last bit of the original multiplier is now in X_0 , and since it is a 1, the positive going transition of the fourth pulse transfers the sum into the accumulator. The accumulator now holds the final product. The negative going transition of the clock pulse shifts X and B again. Note that, X is now 0000, since all the multiplier bits have been shifted out.

