Digital Circuits and Systems

Prof. S. Srinivasan

Department of Electrical Engineering

Indian Institute of Technology, Madras

Lecture # 11

Arithmetic Circuits

So today we will look at arithmetic circuits, computers of course are the most important digital systems at least it is the most visible digital system. Whenever you say digital immediately what comes in your mind is the computer and computer consists of all these arithmetic units, of course there is a lot of control also, arithmetic units have to be controlled, when does an arithmetic unit work, where does it take the input from, where does it write the output, where does it send the output to, so all those things are there, the control aspect of computing.

We are not discussing that here but it will be in a future course. But in this course we will see some of the basic circuits which perform arithmetic in computers. When we say computer we need not imagine a big Pentium P3 P4 but it can be anything even a much simpler computing circuit.

So far we used logic gates in order to implement some of those circuits like parity generator for example code converters etc. whatever is the circuits definition input output specification we finally reduce it a Boolean equation minimum sum of products or product of sums implemented using gates and gates are basically logic devices, logic circuits wherein you have your true or false inputs and true or false outputs. So when some combination of true or false inputs gives rise to a specific output being a true or false.

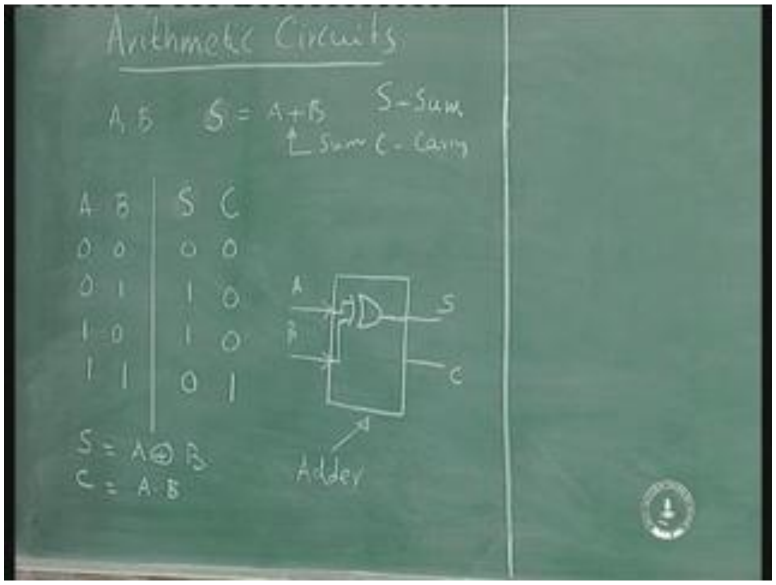
How you translate this logical operation into arithmetic operation is what is required in a computer. In computer we give numbers as inputs and we want numbers as the outputs so it is how you convert or how you transform these logical gates into arithmetic circuits. Again it is simple because of the binary nature of the numbers we are representing. Because we talk about binary representation in numbers, however big the number is we have to translate it into binary format binary representation 0s and 1s. So as long as of 0s and 1s are inputs of circuits we can have an output which is also 0 or 1.

You can interpret it as logical signals and logical inputs logical outputs or interpret it as arithmetic circuits whose values can be 0 and 1 and the output as 0 or 1. So it is a question of interpretation of the same logic gates whether it is an arithmetic circuit or a logical circuit the simple example will be a gate in which I have two inputs have 0 and 1 if I want to sum of these I want to sum of 0 and 1 so the output can be 1 and if both are 0 the output has to be 0. So this is the type of arithmetic operation we will perform using the same logic gate. So the simplest of arithmetic operation is an adder, of course all of us to know that.

Let us say we have two numbers A and B and we want C which is sum of A and B. this is the sum operator not the OR operator, the arithmetic sum. Since the numbers are binary each of them can only take the value of 0 or 1. So I have four values if both are 1 what should be the output? The output sum is 2 which cannot be represent with one binary digit because binary digit can only represent 0 or 1 so I know the number will be 0 sum is 0, it is truncating it is something like I am trying to add two single digit numbers and I have only a single digit to represent my result and if the number exceeds the single digit number naturally it will only show the first digit and the second digit will be discarded.

Suppose I have a single digit decimal representation both inputs and outputs as long as the number is less than 9 I can get the actual value but if the sum is more than 9 then the first digit only will appear the second digit will be discarded. Suppose I have two numbers added 5 and 5 and the answer is 10 but 10 cannot accommodate my single digit output but only represent the first digit 0 and the second digit 1 will be discarded. so that way when we look at it that the sum is 0 but we need to take care of fact that the sum is 0 not because both were 0 but because both were 1 so I need to have one more output so I will not even call this C now I will call it S as a sum.

(Refer Slide Time 09:28)



So S is a sum and I will call this C as the carry so S stands for sum and C for carry. The carry is when you have a digit which is overflowing that overflow has to be accommodated elsewhere in the next computation stage. The next time you compute you have to take into account this overflow before you can proceed with the addition. So I need a carry which is only one in this case but in all other cases it is 0. It is a simple adder so I have two inputs and two outputs so this is going to be my adder circuit with two inputs A and B, two outputs S and C adder and you can very easily see that sum is Exclusive OR of A and B and C is AND of A and B. So without having to go through the Karnaugh Map and Boolean algebra, reduction and everything I can look at the truth table and write the sum as A Exclusive OR B C as this (Refer Slide Time: 9:13). So basically this circuit is nothing but this, and the carry is nothing but a gate like this.

So an Exclusive OR takes this A and B and gives the sum and the AND gate takes this A and B and gives a carry. Now this is one bit addition, suppose no number is small enough to be represented by one bit in that case you don't need a computer, you don't need a calculator; you don't need an electronic circuit.

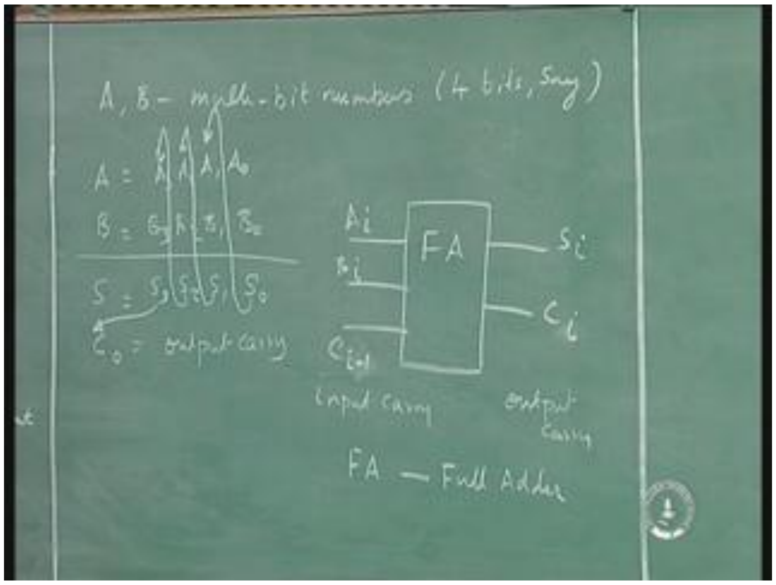
Suppose all you can do with a computer is add 1 and 1 and nothing more than that so you don't need a computer so I need to naturally represent the numbers by larger number of bits so the whole range of decimal number I want to operate from 0 to infinity as infinity is only a fixatious quantity any large number you want to deal with you want to add only then you need a computer you need a calculator you punch these numbers to get the output when the number is large and not when the number is very small. somebody asks to get three and four do you need a calculator for that you don't, may be after sometime but right now at least people can add single digit without having to resolve through calculator, earlier it used to be large numbers.

But now we have to depend on computers and calculators so much but for single digits we still don't need calculators but then we don't need these circuits. So naturally this one bit is only one of the several bits that will be in your number of representation. Hence this A and B will be really multi-bit representation of a decimal number. we have gone through the decimal to binary representation so when I have a given number A and B are both decimal numbers of several digits the first thing you do is convert them into binary and then add those numbers so reasonably assume that A and B are both multi-bit numbers. That means when I add A and B the first digit let us say I have multi-bit A and B, A and B are multi-bit numbers so to start with I will call them 4-bits. Now I represent A as A_0 the lowest significant bits A_1 A_2 A_3 and B as B_0 B_1 B_2 B_3 and I will have the sum as S_0 S_1 S_2 S_3 and then I may have a carry from A_3 and B_3 so there will be a carry out also in addition to this.

Therefore this is sum, A, B, sum and a carry C_o is for output carry. So when I consider A_0 and B_0 first two bits of A and B first bit of A and first bit of B naturally there is no carry because it is the very first bit from A and very first bit from B so you can use this hardware. But when I try do the addition of A_1 and B_1 I need to consider not only A_1 B_1 but I need to also consider the carry from A_0 B_0. So the carry from here has to flow into this and carry from here has to flow into this carry from here has to flow into this and the final carry is what we say here. So this is the ‘carry in’ and this is the ‘carry out’. So this carry is a carry out the output carry which has to be fed as the input carry in the next bit addition. In the next stage of the binary arithmetic I need to take the output carry from the previous stage and feed it as the input carry. So C out would be C_{in} for the next stage so this adder cannot do the job this is having limitation and it can only take two numbers and not the third number and such an adder is called a half adder.

A half adder is the one which can only take two digits and add the sum and get the carry but not take into account the carry from the previous position of the addition.

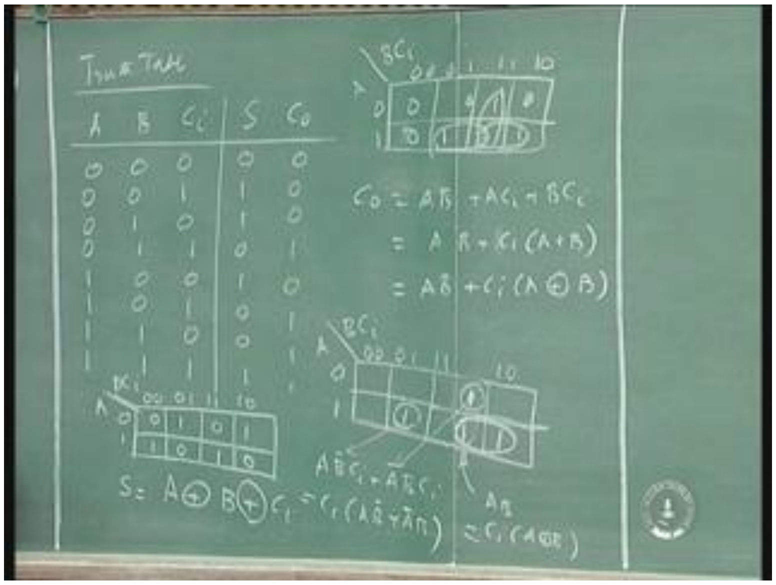
(Refer Slide Time 16:29)



I need to expand my circuit into a circuit in which I will take my A B and C_{in} the C_{in} being the carry from the previous bit position and get my sum Co C_{in} is the input carry and Co is the output carry. Input carry comes from the previous position if this is the nth bit this is n minus 1th carry out of the previous bit. So, if I generalize it C_i A_i and B_i are the ith bit of A and ith bit of B this is C_{in} which is nothing but Co C_i minus 1 from the previous position. I get C_i minus 1 and I get sum ith bit carry ith bit and this C_i will become in the next stage carry in. Such an adder which can take into account the carry from the previous position is called a full adder which is only of practical use. This is not a practical use except for the very first bit position. The very first bit position in a large multi-bit number the very first bit can be added using half adder but all other bits have to be added with a full adder. For a full adder you can connect the carry to C_o so I don’t want to have two types of circuits suppose I have 16-bit numbers I will have sixteen As and sixteen Bs so instead of having fifteen full adders and one half adder I can have sixteen full adders for uniformity sake, the very first full adder I will make the carry input as 0 then I will have uniformity.

So what is the behavior of this full adder? How does it look like? Full adder truth table: We will call A and B C input sum and C output. Sum is 0 1 1 0, 1 0 0 1, carry is 0 0 0 1, 0 1 1 1. You have look at this and write it very easily. For example, if you take this there are two bits 1 and the third bit is 0, two bits will lead to 2 and 2 is 1 0 so sum is 0 carry is 1. all you have to do this count the number of 1s A B C. in A B $_{Ci}$ count the number of 1s if it is a 0 you put a 0 if it is 1 you put 1 and if it is 2 you put a 0 and the one gets the next one, if it is 3 you put 1 and 1 in the next one that's all. So the number can be all the three can be 0s all the three can be 1s, this only a binary representation of the number of 1s.

(Refer Slide Time 29:35)



The output is nothing but the binary representation of the number of 1s in the truth table. The only thing is this if all are 0s we have 0 0, take this for example there are two 1s and the output is 1 0 binary representation of 2 is 1 0. This is 3 and the binary representation of 3 is 1 1. This is how we write the truth table, it is very easy number of 1. If it is 0 you put a 0, if it is 1 you put 1, if it is 2 you put a 0 1, if it is 3 you put 1 1.

Therefore now I can write the truth table for this. If the truth table is there I can put the Karnaugh Map for this and get the simplification and draw the logic for my full adder. I am going to give this an exercise, I am go to draw the Karnaugh Map, simplification I am going to leave it as an exercise to you which is very easy it is there in all the books, we have the truth table A B C_i this is sum (Refer Slide Time: 20:14) so first row is 0, C_i is 0 1 0 1 and 1 0 1 0 this is the truth table and the Karnaugh Map. This is the K' map for S, the first row is 0, second row is 1, third row is 1, fourth row is 0, fifth is 1, sixth is 1, seventh is 0 and eight is 1.

We know how to draw. We can simplify it apparently there are no prime implicants but remember we had this type of pattern in our parity generator, we had this pattern also in one of those converters gray code converters so this is the Exclusive OR solution. I will leave it as an exercise to you to get this expression. I will give the expression to you it is

A Exclusive OR B Exclusive OR C_i please get this expression as a homework, this is very easy, the pattern is there. The pattern has been identified earlier in our parity checker parity generator. And the carry out is 0 0 0 1 there are four 1s. If you try to join them it will be three prime implicants all essential three essential prime implicants we can ADD them or OR them really not ADD them. Co would be AB ACi BCi or AB C_i A OR B this is the carry output. If C_i is the carry input Co is the carry output and A and B are the bits. it can be applied to any bit Ai or Bi it can be the third bit or fourth bit or fifth bit and if it is the fifth bit then it is the fifth bit input of A, fifth input of B, fourth bit output of C, and this will be the fifth bit output of C. C_i is the one bit before the current bit, Co is that current bit output for carry.

I want to give this as an assignment. I can also write it as C_i A Exclusive OR B. We will use it sometimes not now. Occasionally we will use this expression for carry and not this expression. That means this should be equivalent that means this should also be right. I can tell you intuitively without even going through Boolean algebra how this can be written, when you have A B you are taking these two terms into account, this one and this one have been considered in getting this expression, A B is same as this and this, I have this one extra 1 and this extra one and in order to make my prime implicants smaller my sum of product expressions smaller what did I was to combine this 1 with A1 and combine this 1 with the 1. Remember this one is already been considered this expression so what I am saying again I will do it here one more time. Consider this carefully.

I have already considered this as AB BC_i, AB is what? Co is 0 0 0 1, BCi is 0 1 1 1 so when we consider this as BCi AB we will first take AB which is AB? AB let us say this has already been considered as AB I have these two 1s left I combine this 1 with the 1 2 make my product terms less by one literal. Whenever I combine two 1s my product term gets reduced by one literal. Every time I combine a factor of 2 one literal goes out, 4 if I combine two literals go out, if I combine eight terms three literals go out. So that was the reason why I have to make this and this even though this was not necessary to have been combined this with this and this because these has already been take into account, now if I consider these two terms separately what is this? This is A B bar C_i and this is A bar B C_i. So, if I take C_i out it becomes AB bar or A bar B these two terms (Refer Slide Time: 27:13) is this which is equal to C_i times AB bar A bar B which is nothing but A Exclusive OR B. So this is same as C_i A Exclusive OR B that is this. so I can either write it this way by grouping the 1s more than once, grouping this one three times actually, if I group this one three times I get this, if I do not do it if I only group this here for which I get AB and these two 1s I separately write as a sum of product term it reduces to an Exclusive OR.

In other words my Co can be AB or C_i and A OR B or A Exclusive OR B. I will use this to my advantage again as I said hardware advantage. Suppose I am asked to give an Exclusive OR solution I want more Exclusive OR gates and less OR gates for some reason may be I have an Exclusive OR array and I want to use as many Exclusive ORs as possible in my solution I can use this. Sometimes I may say give me the simplest possible solution then in that case I have do go for an OR solution. Exclusive OR gate is definitely a more complex gate than a simple OR gate. So this is the philosophy of minimum sum of product expression. Minimum sum of product expression not always give you the most optimum solution in terms of practicality. We saw it in the case of gray code converter, we saw it in the case of parity generator, if I did not use this Exclusive OR solution I will have four min terms each min term with three literals, there will be four AND gates with three inputs each and one OR gate with four inputs and each of these A B C is inverted so there are three invertors. Three invertors for A B C_i, four AND gates with three inputs each, one OR gate with four inputs. As again said I am now having two Exclusive OR gates with no invertors; A need not be inverted, B need not inverted, C_i need not be inverted, all I need is two Exclusive.

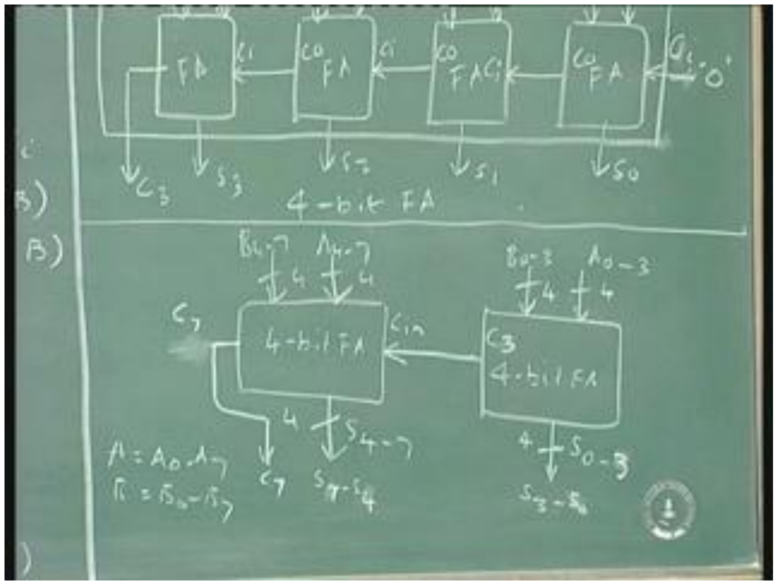
Suppose I have two Exclusive OR spare I am having a big hardware, board in which I am doing several things and I happened to have a couple of these Exclusive ORs unutilized so I will use it. So after the minimum sum of product expression all these Karnaugh Map mapping and minimum sum of product expression, trying to get as many 1s as possible to be grouped together is all fine I am not saying you should not do it but after that also you be a sort of little bit vigilant after that some little manipulation can lead to a very simple or elegant circuitry, that's what we did in the case of Exclusive OR gate, in parity check generator, that is what we did here in design gray code and that is exactly what I am doing here now.

Sometimes A OR B can be replaced by Exclusive OR if necessary I am not saying do it every time. Suppose an Exclusive OR solution is needed for some practical reason because probably you have stocked Exclusive OR gates a large number you want to use it rather than go to a market and buy an OR gate that is a good enough reason or I have an Exclusive OR gate unused in my board I use it rather than going and finding an another OR gate and finding a place for it and connecting it together and all that.

All I am trying to say is these two expressions are equivalent. Depending on whichever you want you can use it but generally people stop with this. And this you can see but this we will use later on, we are going to use this later in our consideration for simplification of circuitry. We will use that some other time but meanwhile I thought it is worthwhile to say this to you.

What we have done so far is to do a half adder two bits both A and B to be added with the carry, two input and two outputs with a half adder, not of much practical use except in the least significant bit where there is no carry input but generally when you are doing in a multi-bit addition one bit not having a carry is not significant at all so you can as well forget about the half adder and do a full adder in which there are three inputs A and B and previous input, previous carry output as the input of the carry, and two outputs sum and carry and sum is coming in the Exclusive OR format and carry can come in this format and this format. This is for a single bit. As I said we have to do multi-bit addition. Suppose I have 4-bit number as I assumed here A and B are multi-bit additions so how will it look like? A 4-bit adder will look like, I have 4-bits Ai A_0 B_0, A_1 B_1 A_3 B_3 now the carry from the previous input, let us assume all of them to be full adders for the sake of uniformity this carry in I will ground it make it 0 because in the very first bit there is no carry.

(Refer Slide Time 39:31)



This carry out goes into carry in, this carry out goes into this carry in, this carry out goes into this carry in (Refer Slide Time: 33:48) so we will have sum 0, sum bit 1, sum bit 2, sum bit 3 and the final carry bit will be my C3 so my sum will be two 4-bits adder and I can have a sum of five bits because the last bit can produce a carry. So two 4-bits when I add the sum, when I say sum that includes carry the total number I get at the output will be 4 or 5 depending on whether there is an output carry or not so I should allow for five bits. Whether I have it or not depends on the numbers, if both the numbers are 0s the output is 0 so there is no need for a carry out B, if I knew the numbers A and B why should I add, so if I don't know the numbers A and B it is a possibility that can generate a carry at the output so when you add two 4-bit numbers the result can have up to five bits.

The fifth bit is nothing but the carry out of the fourth stage. So when you are going to the shop to buy a 4-bit adder you have only the As and Bs to be fed in A_0 B_0, A_1 B_1, A_2 B_2, A_3 B_3 outcomes S_0 S_1, S_2, S_3 and C_3 and you have to give the Cin as the input as 0. This is called a 4-bit full adder as one package. Whatever I have drawn inside the rectangle is a one package of IC integrated Circuit wherein I can feed four A bits, 4-bits of B and carry from the input. Why do we need the carry input? Of course in the first stage the carry is 0 but suppose I want an 8-bit adder what will I do for 8-bit adder, I will use these 4-bits full adder.

I will use another 4-bit full adder, this is A0 to 3, B0 - 3 this arrow with a cross means there is more than one bit. I write here 0 to three bits that means it is 4. A0 to 3 means 4-bits of A, B0 to 3 means 4-bits of B but one 4-bits of B a single arrow cannot show 4-bits so you put a cross across it. If you know this means it is more than one input that's what it means. An arrow cross in a digital drawing indicates it is more than one bit input. If you know the number of inputs you can write 4. You know more than one but you are not sure how many then you put a cross and stop.

Similarly I get here A5, A4 to 7 4-bits this is my 4-bit, B4 to 7 so I have two numbers now A is equal to A0 to A7, B is equal to B0 to B7 8-bit numbers. A is an 8-bit number B is an 8-bit number the numbers are A0 to A7 B0 to B7 these are the bits. So the sum can be 8-bits or 9-bits, the eighth bit can have a carry so I will feed the output Co which is actually C3, C3 of the previous block as the input here. So don't assume that every time the first bit will have a 0 input, don't assume that the first bit of the 4-bit adder will always have a carry input as 0.

If it is an intermediate stage in a multi-bit addition then my input carry would be the output of the previous block. This block of 4-bits, the output feeding into a block of another 4-bits, the output carry C3 goes as input carry C_{in} and output of this is C is 8 C7 so this would be sum 0 to 4, (Refer Slide time: 38:56) sum 0 to 3 4-bits, 4 to 7 4-bits so this is four, 4-bits, 4-bits, C7. So C7 S4 to S7 S3 to S0 if you want to write S7 to S4 S3 to S0 first 4-bits, second 4-bits and the last bit the carry bit is the ninth bit. Hence like that you can add. You can cascade any number of 4-bits.

The 4-bit is generally available as a slice. The 4-bit adder is the component which is available as readymade circuits, sometimes 8-bit adders are available but not 16-bit adder you can't buy a 16-bit adder, you can't buy a 32-bit adder. if you want to build a 32-bit adder you have to buy either 4-bits or 8-bits, if 8-bits are available you buy 8-bits otherwise buy 4-bit adders and cascade them, first 4-bit feeding into the second 4-bit carry, carry of the second 4-bit feeding into the carry in of the third 4-bit etc each so each of these will give you 4-bit sum, 4-bit sum, 4-bit sum, 4-bit sum and the last carry would also be considered and intermediate carries will be fed into the next stage and the very first carry in is 0 and four As and Bs are fed into each of these blocks as usual. This is a multi-bit adder in 8-bit full adder (Refer Slide Time: 40:35). Like that you can go on building any number of 4-bit adders to make higher bit adders like 32-bits, 64-bits but let us examine whether it is a very efficient way of doing it.

Any questions so far on the half adder the full adder, how do you combine full adders to 4-bit adders and 4-bit adders into multi-bit adders of any number of bits, the concept of carry in and carry out, sum. So individually each of these blocks will have A Exclusive OR B Exclusive OR C as the sum and AB or C A Exclusive OR B as the carry circuits so there will be so many gates inside.

Take it as an exercise; find out how many gates will be there in a 16-bit full adder. Assuming the first stage also is a full adder. Let us not assume a half adder. Let us assume for uniformity sake there are 16-bits, I am adding using this scheme that means I will have four of this 4-bit adder and in turn a 4-bit adder will be like this (Refer Slide time: 42:12) full adder, full adder, full adder, full adder that means there is going to be sixteen full adders and you know each full adder is having the sum circuit like this and carry circuit like this. So find out and list using this equation and using this equation. For sum use this equation always (Refer Slide Time: 42:37) for carry you can use both these equations first this and this. In this case list the number of gates, whenever you are asked to give the number of gates you should give the type of the gates you can't say seventeen gates, you will not to say eight gates of three input AND gates, eight three input AND gates, six two input OR gates, four four input OR gates like that we should give a list of types and the number of inputs only then it becomes complete. As if we can go to a market with that and to a shop with this and then buy those. Make a shopping list of the number of gates required to build a 16-bit full adder.

Now having said that let us quickly see whether it is the most efficient way of doing the job. When I say efficiency we talk of many things, I already said in the beginning. The speed is one thing, the power consumption, the number of gates so that the size is reduced so you can make all in one IC it will be very fine so I can save on the size. For a moment you forget about that size assume that it is all that one integrated circuit. We will not even make 4-bit, 4-bit, 4-bit, 4-bit and in a 16-bit gate we will try to integrate sixteen of those units inside one small IC block so that size can be saved we will not talk about power saving now because I have not told you how to compute the power requirement of a gate. But one of the parameter I am stressing on from the beginning is the speed.

How much time it takes to compute a 16-bit addition? Do you have any idea how much time does it take? Suppose I have A and B, A is 16-bit, B is 16-bit all of them are available to start with. I am giving you a 16-bit number for A, another 16-bit number for B and say go and we are asked to do the addition. Can you do it instantaneously? No, because the gate themselves will take time. For example, even if a single bit adder or let us take this half adder, the moment I gave A and B the output S and C are not available because however fast these circuits work the gates work there is a finite time delay.

So, there is a time delay involved in propagation of these values for the switching operation Exclusive OR gates and switching operation of this AND gates before we get the result. Since we don’t have an idea of the gate delays of individual Exclusive OR gate and individual AND gate the delay will depend on the type of the gate and number of inputs both decides the delay. The gate delay depends on the number of inputs of the gate as well as the type of the gate. Since we don’t know anything of all those things we will only simply say the time it takes for the output to come after the input is given we will call it propagation delay.

The delay of propagation of the inputs to the outputs is called propagation delay. There is only one Exclusive OR gate and one AND gate. When you go to a full adder there are two Exclusive OR gates and a two level gate one AND gate feeding into a NOR gate, whether you use this model or this model let us use this model it is one AND gate feeding into another gate so there is going to be some delay there.

There is going to be a delay into these two stages of gate thing, here also there are two stages of a gate thing A Exclusive OR B has to be fed into this C, first Exclusive OR gate will find A Exclusive OR B, second Exclusive OR gate will take that and put it into C so two levels. Here is one level of gating some propagation delay, here there are two levels of gating propagation delay. Since we have taken the full adder as a unit of our consideration because half adder is used as only the first stage so we will forget about it for the moment for all purposes of calculation and standardization we will assume full adder as the basic building block.

A single bit full adder 1-bit full adder is a basic building block we will call the propagation delay of the full adder TP that consists of two parts the carry and the sum whether the sum comes first or carry comes first because one path carries another path whichever path activates first will come out first. We will not worry about, we are only interested in the carry now because the carry has to come from here (Refer Slide Time: 48:04) go here, go here. Even if the sum comes with a slightly more delay sum and carry may not have equal propagation delays. we don't know but we will have to look at the gates structure to see which is faster, sum can be faster carry can be faster let us not talk about it for a moment.

For a moment we will only consider on carry propagation because carry propagation is more important the carry has to go from here to here, here to here, here to here, here to here and then from here (Refer Slide Time: 48:30). The 16-bit full adder which I asked to do as an exercise carry has to propagate through sixteen stages, first stage has to go to the second stage, second has to go to the third and finally it has to go to the sixteenth stage so sixteen carry delays have to be accounted for. Meanwhile the sum will come because as soon as A and B are known and once the carry comes the carry triggers the sum, this carry trigger the sum. It is this something like the domino effect. As soon as the carry comes sum comes but then carry goes on, carry goes from here to here leaving sum behind, goes from here to here leaving sum behind. The sum may be slightly delayed than the carry but then before the carry goes to the next stage the sum could have been settled. So you don't worry about the minor difference between the sum delay and the carry delay let us only worry about carry delay because it is more significant of the two because it has to travel a wide long distance so the total propagation delay is sixteen carry propagation delays, carry of the full adder is carry propagation delay sixteen of them has to be used so it is a 16-bit adder, it could be a 32-bit adder, 64-bit adder you can imagine.

Therefore we have to think of, if you want to talk about one Giga hertz, Pentium II Giga hertz processor, what is a Giga hertz in terms of clock frequency? It is nano seconds. Giga is 10 power 9 reciprocal of 10 power 9 is nano. We are talking of nano seconds, the whole operation. When you say computer works at nano seconds the whole thing has to be done, some addition process, some multiplication process or some other process and we are talking about full adder here and how many such things will be there in a computer, adding two numbers is the least complicated thing you can think of in computing.

In a computing environment there are so many other more complex things that need to be done. That means we have to definitely try to do something to reduce the carry delay. There are several techniques and to exhaust all of them will be beyond the scope of this lecture series. But I thought at least some of those concepts are important it has to be drilled in the beginning. When you are learning the first course in digital you need understand to certain parameters.

At least you should know a problem exists. Later on you have opportunity to learn about solutions of these. So we will take one technique of improving the speed of carry. One technique of reducing the propagation delay of a 4-bit full adder we will see in the next lecture. But that is not the only way to do it there are several other methods. We will not be talking about all those methods. We will take one simple method of reducing the propagation delay of the 4-bit full adder in the next lecture.