Architectural Design of Digital Integrated Circuits
Prof. Indranil Hatai
School of VLSI Technology
Indian Institute of Engineering Science and Technology, Shibpur, Howrah

Lecture – 37
Multiplier Architecture (Contd.)

So, welcome back to the course on Architectural Design of ICs. So, in the last class, we have seen that squaring circuit how we can do, then again multiplication using squaring circuit, where it will be used when it will be beneficial; that means, what is the architecture that we have seen in the last class.

So, after that, we have discussed that the till now whatever I multiplier architecture we have discussed, that is based on this integer or that means, the considering the fact, that the number is not constant they are unknown to me, but in DSP system what we know, that the numbers are basically known from the beginning at the coefficient values there or the twiddle factors for FFT or the coefficient; that means, the filter coefficient values they are for FIR filters or IR filters or any of this filter the that means, the corresponding values, they are fixed for any particular specification.

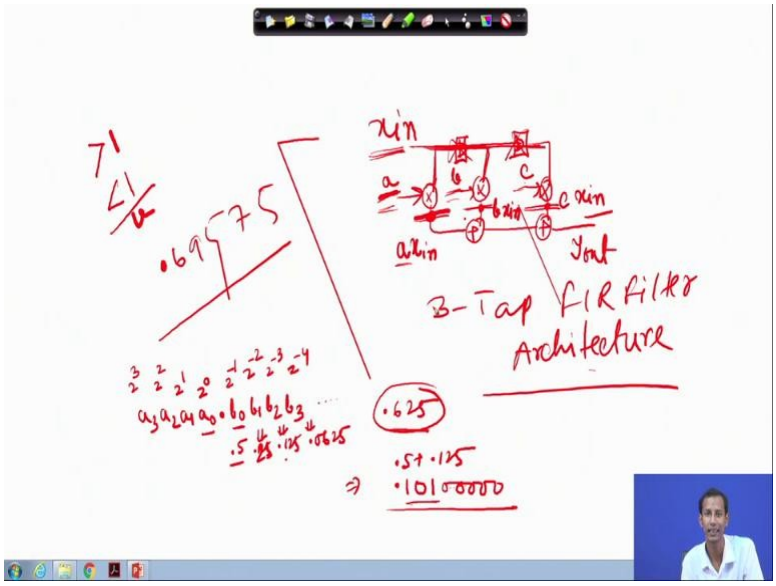
So, from the beginning we know. So, after that, how we can design one circuit which will be optimize as the number is, from the beginning we know. So, at the time can I get the benefit of that, whenever we will design the multiplication operation for those DSP system? So, that we will see in today's class.

(Refer Slide Time: 01:40)

So, to start with that ok, so, here the topic is basically Reconfigurable Constant Multiplier Design, based on this Common Sub Expression Elimination algorithm. So, to start with this reconfigurable architecture of this constant multiplier design I have to, I will discuss, what is constant multiplication, where it is used basically then we will come to the fact that what is the meaning of reconfigurability or this what is the meaning of reconfigurable architecture and why I need to design this reconfigurable constant multiplier and then, what is this common sub expression elimination algorithm and how it basically advantageous or how it used to design one that means, efficient, architecture for this reconfigurable constant multiplier design.

So, now let us first initially consider this fact that what I have seen or what I told you that suppose I am having one 3-TAP FIR filter sorry this is my x in ok.

(Refer Slide Time: 02:38)



So, this is these are the filter coefficient these are the registers and this is the y out. This is nothing but one 3-Tap F I R filter architecture ok. So, here you see, I need multiplier for doing these three multiplication. So, what is happening here? These are the coefficient values which is a, b and c.

So, in most of the case we know that this F I R filter coefficient values that cannot go beyond

1. So, they are basically less than 1 all the time. So, then my first point is that or the first understanding will be that how we represent the; that means, the fractional value, suppose 0.69575 something like this fractional value which is this values is where the values is less than 1, how we can represent in binary number?

So, how we can represent? Basically, we can represent in binary. How we represent? That is suppose I am having the decimal point in this particular location. So, then I am having this a 0 a 1 a 2 a 3 something like this and b 0 b 1 b 2 b 3, something like this. So, if I consider the decimal part at this particular the decimal point at this particular position so, at that time the weight for this, this will be 2 to the power 0, this is 2 to the power 1, this 2 to the power 2, this is 2 to the power 3. So, before to the decimal point whatever is the value or what sorry, whatever is the point or whatever is that means, corresponding bits, so, that corresponds to the weight in terms of in the power of 2.

But after the decimal point that the corresponding b position which is this b 0 to b 3, there also considering the fact that there also the values are in the power of 2, but here the values are varying in terms of 2 to the power minus 1, 2 to the power minus 2, 2 to the power minus 3, 2 to the power minus 4, something like this.

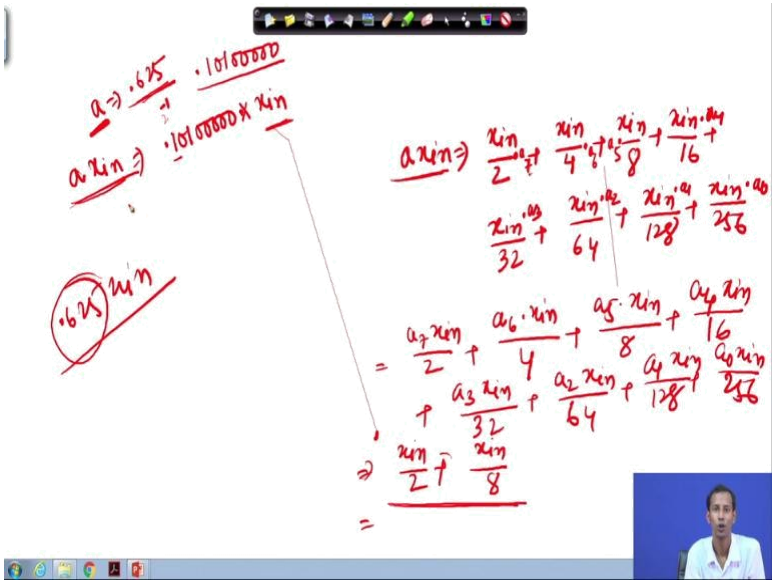
So that means, now if I want to suppose, if I have to represent one 625 data or the values which is basically 0.625, so, how I can represent that value? So this particular value corresponds to what? 0.5, this corresponds to what 0.125, this sorry, this sorry, 0.25. So, this corresponds to 0.125, these corresponds to 0.0625 something like this, ok. So that means, now if I have to get this 0.625 values, so, this basically says 0.5 plus 0.125. So that means, the values will be something like 0.101 and then if I consider rest of the bit will be 0 0 0, if I consider 8 bit to represent the corresponding value, ok. So, this is the fact how we represent the decimal sorry, this fractional value in binary, ok.

So, now this what I said or what we know that in DSP system, this corresponding filter coefficient values they are mainly less than 1; that means, they will be something like this, the values will be something like this, ok. And not only that the fact is that all the time it will be known depending on this 3 tap specification of 3 tap F I R Filter so, this values will be known from the beginning, ok.

So, if I known from the beginning so, here what is unknown? The unknown value is what will be my x in. So that means, if I just come to this particular point so, this will be the output for this particular output at this particular point the output will be a into x in, ok.

So, here the output is b into x in. If I just do not consider this, this register that suppose there is no register, ok. So, at the time this will be b into x in and this will be c into x in. So, these are the three multiplications which I have to do for this kind of architecture to implement. So, here you see one thing is that now a is basically represented by this kind of things, the same for b, the same for c.

(Refer Slide Time: 09:21)



So, can I write this particular terms, this a x in, in a different way? I can write, how can I write? So, suppose the values for a that are what that is 0.625. So, now, if I want to add a into x in, so, at that time how, what will be it is values? It will be 0.10100000 into x in, something like this. So, means what this means, what, how I can get corresponding a into x in value? If I just write this particular term I know that this is what is the that means, the power of or what is the that is this values of this particular position? That is 2 to the power minus 1, ok.

So; that means, if I just write this a into x in, in such a way so, that I can get it in a, I can get the results, but in a different the representation of that, that can I write it in a different way? I can. So, what it will be, a into x in can be written as x in by 2 plus x in by 4 plus x in by 8 x in by 16 plus x in by 32 plus x in by 64 plus x in by 128; 1 2 3 4 5 6 7 plus x in by 256, not only that.

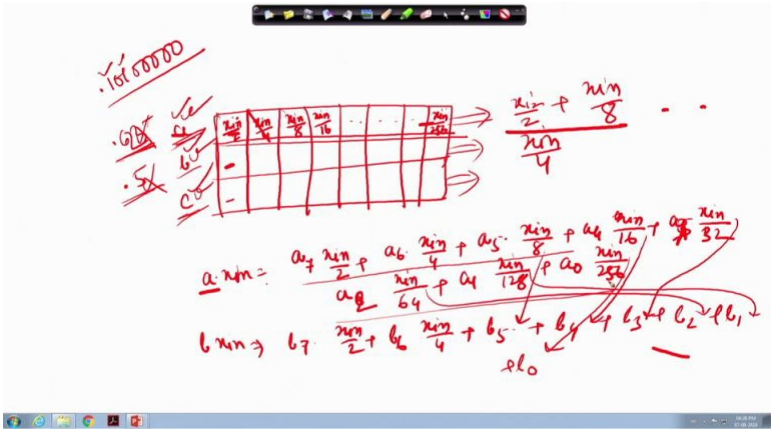
So, here along with that what I have to write, what is the bit corresponding bit position for this a. So, this is the a 7; so, this is a 6, this is a 5, so, this will be anding operation. Then again, so, x in will be anded with a 4, this will be with a 3, this will be with a 2, this will be with a 1 and this will be with a 0.

That means what? Now these particular a into x in; that means, this particular term or these can be written as this a 7 into x in divided by 2 plus a 6 into x in divided by 4 a 5 into x in divided by 8 a 4 into x in divided by 16 plus a 3 into x in divided by 32 a 2 into x in divided by 64 a 1 x in divided by 128 plus a 0 into x in divided by 256. Now, if I have to multiply with 0.625, if this corresponding a value is 0.625, so, at that time only this will come. So that means, it will be x in by 2 plus, so, this will be 0 because what is the values of this? That values is 1 0 1 and rest all are 0, right. So, this will be x in by 8, rest all are bit are 0, so; that means, all these particular terms will be 0. So, only this I require only one particular adder to do this 0.625 x in, ok. That means what? For these particular terms though I require how many adder? That means, for to do this 8 terms, I need 7 addition operation but at this number is fixed so, I require only 1 number of addition for this operation for this particular case.

So that means, rest of the 6 that I do not require to do. So that means, for this constant multiplication, I can get the area optimization as well as the power optimization on the higher side. As we know, this value from the beginning, ok. So, now, this is for a into x in.

So, if I just go back to the previous so, how many operation I am having; a x in, b x in and c x in. So, three particular terms, this a is different, b is different, and c is different, but x in that is common to each of these.

(Refer Slide Time: 14:51)



So, at that time what I can do? So, if I write this in a matrix form, considering the fact that each of this is basically 8, right 8 bit, each of this a b c of 8 bit. So, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 7 and then 8, right. So, this is for a, this for b and this is for c.

And each of this case, this is if I consider this is x in by 2, this is x in by 4, this x in by 8, x in by 16, something like this up to x in by 256 for each of this position, where this is basically multiplied with the corresponding bits of a. This particular position is multiplied the corresponding bit position of b, this particular position is being multiplied with or anded with bitwise, this is bitwise anded. So, that is anded with the each of the bit of c, ok.

So that means, these particular this basically now this forms one 2-dimensional matrix. So, now, if I consider the fact that I know this a b c from the beginning then for each of this multiplication operation that will be optimize as in the previous case for this if this value is 0.625, so, only one this x in by 2 plus x in by 8, I have to add to do this 0.625 into x in.

Suppose this value is, let us consider that is 0.5, ok. So, only x in by 4, I have to do. So, in this way the corresponding multiplication operation which is required here that will be optimized in a better way, ok. If I just do, if I just; that means, generic multiplier if we use, so, at that time it requires huge number of logic elements or the logic gates, but in this case the logical expression for each of these that are reduced in a great extent; that means, I can reduce the or I can improved the performance of this circuit in a better way.

Now, what I said? Whenever I know that these value are fixed at that time, this is, this can be done, but in some case what happens that I do not know the values, ok, I do not know the values of this a b c; that means, these values are not same, I do not know the values of this. So, at that time all the values of this a b c they can change dynamically.

So, at that time I need one or at that time these particular things needs for one reconfigurable architecture. Reconfigurable architecture means the same architecture can be accommodate the different values of a, different values of b, different values of c. So, how we can design that fact design one particular circuit where any on the values of a b c can be accommodated?

So, can I design one such kind of circuit where any of this values of this a b c can be accommodated to design the corresponding this multiplier architectures, ok? So, it can be done. So, it can be done how? It can be done based on the fact that so, when it will take so, at that time you consider the worst case. So, worst case means, edge for this 0.625, the values

are like only 101 rest all are 0. So that means, as only this two particular bit position is 1, that is why there is only this two-addition operation. If I want to; that means, do this multiplication operation in a shift and add base method, so, at that time when all this positions are 1, so, at that time I will get the maximum number of terms over here for the addition operation.

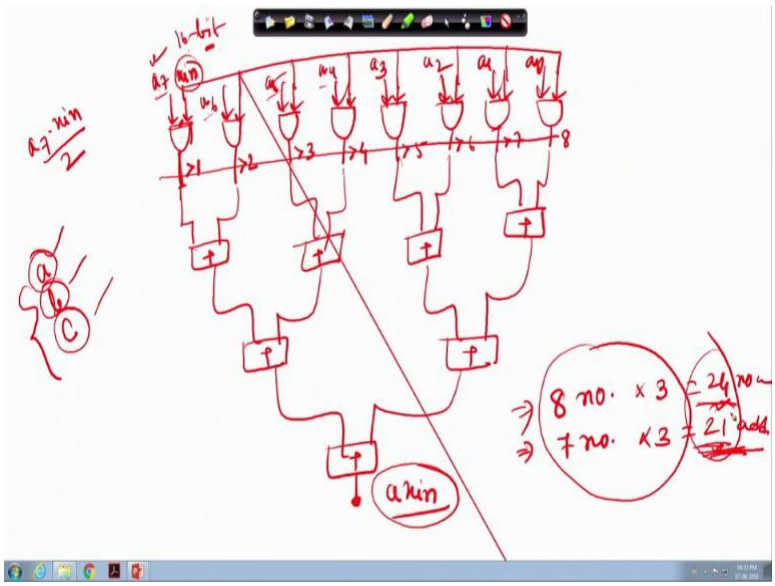
So that means, as this particular position is; that means, each of this is basically anded with the corresponded values of a, this particular position has been anded with the corresponding values of b. So that means, x in at these for a what is happening a into x in, what is happening? That is a 7 x in divided by 2. So, a 6 into x in divided by 4 a 5 into x in divided by 8 then a 4 x in divided by 16.

So, a 5 x in divided by 32, then a sorry, this is a 3, this is a 2 x in divided by 64 plus a 1 x in divided by 128 plus a 0 x in divided by 256, the same thing happening for b x in also. So, here what is the change, only this x in by 2 that is common, only this particular b is value is changing.

So, here it will be b 6 x in by 4 then b 5 then it will be b 4, then it will be b 3, then it will be b 2, then it will be b 1, then it will be b 0 and each bit multiplied with this corresponding terms, ok.

Each anded with this corresponding terms. That means, if I do not know the values of a, so, at that time I have to add each of this particular terms in this way. So, what will be the architecture for a into x in, then? So, a into x in then it will be something like, we requires this and gates, total 8 and gates.

(Refer Slide Time: 22:44)



So, this is a 7 x in. So, x in is basically common for all these, if I just say, ok. What is that mean? Then this bit is only different, this is a 6, this is a 5, this is a 4, this is a 3, this is a 2, this is a 1 and this is a 0. So, now, this is basically divided by 2; so that means, this will be left shift by 1 bit.

So, these will be left shift by 2 bit, this will be left shift by 3 bit, this will be left shift by 4 bit, this will be left shift by 5 bit, this will be left shift by 6 bit, this will be left shift by 7 bit, this will be left shift by 8 bit, why? Because what I am doing that is what I am doing? Whenever I am doing a 7 into x in plus divided by 2.

So, divided by 2 means that is right shift by 2, right shift by 2 means thus describe the LSB, left shift means increase the that means, power and that means, multiplied with 2, right shift means divided by 2, discard the LSB divide by 2, and that means, put one extra bit at the LSB means you are basically increasing the power of 2 increasing by power of 2. So, how many bit position in the LSB, you will put? So, that many that much in the power of 2 that much you will enhance the corresponding magnitude of the bit or the magnitude of the corresponding values.

So, now then for each of these, now so this is anding operation, this is not and gate, it will be one series of and gate, why? Because if I consider the x in of 16 bit, so, I require a 7 will be anded with each of this 16 bit of x, ok. So, this is not only one single and gate. So, this is the series of and gate which are basically dependent on that values of x; that means, the word length of x, ok.

So, now these particular 2 bit will be now I have to add in this manner. So, this is nothing but the addition operation. Then so, this here I will get finally, a into x in. So, why I require this addition operation? If I just go back, here you see each of this terms is basically finally, added to get this a into x in terms, right.

Here I am doing this final addition; that means these are the partial products generated from this particular anding operation. So, after that each of this these partial products has been added or summed up to find out the final a of x in. So, the same kind of this particular architecture, now it will be followed for b, the same architecture will be followed for c; that means, at that time only the change will be at this a 7, a 6, a 5, a 4 in shift for b, it will be b 7, b 6, b 5, b 4, for c it will be c 7, c 6, c 5, c 4 something like this.

So that means, for a into x in, for a into x in, so, how many of these and gate, I; that means, these series of and gate operation, I require, ok? So, they are of 8 numbers, right and for this summed up how many added operation, I required addition operation, I require? There are 7 numbers. If I consider that means, for a, b and c, three of these particular coefficient values.

So; that means, at that time how much require to implement this particular circuits, this particular circuit where I need to do this a x in b x in and c x in? So, how many of these addition operation and these anding operation I require? Into 3; that means, total 24 number of anding operation I require and 21 number of addition operation I require. So, this is to implement in a generic way; where and why is that? Here, what I considering, I do not know the values of the a, b and c.

So, at that time I require this 24 number of anding operation and 21 number of addition operation to perform this a x in, b x in and c x in on that particular circuit. Now my point is that and this kind of structure is basically known as reconfigurable structure, ok, where I do not know; that means, these value corresponding these, a, b, c values.

So, now my point is that can I optimize this circuit in such a way or optimize in these circuit means can I reduce this numbers more? So, reducing these numbers more means I can save the power as well as the area. So, can I reduce it? If I can reduce it then what will be the method or how, what will be the approach, how can I reduce the corresponding numbers in these particular terms? So, how can I do that?

So, in this regards people have developed different algorithms to do this optimization in these numbers, ok. So, one of them is this Binary Common Sub Expression Elimination Algorithm, ok. So, what happens there, what is the meaning of this common sub expression and then what is the meaning of Binary Common Sub Expression Elimination Algorithm? So, that we will see in the next class. So, thank you for today, if you have any doubts regarding this so, then please let me know via discussion forum.

Thank you.