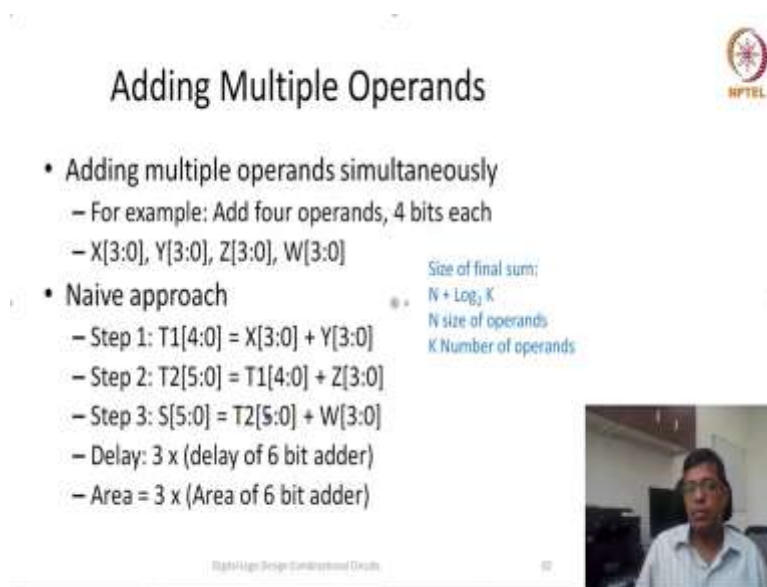



**Digital System Design**  
**Professor Neeraj Goel**  
**Department of Computer Science Engineering**  
**Indian Institute of Technology, Ropar**  
**Lecture – 36**  
**Multiple Operand Adder**

(Refer Slide Time: 00:15)






### Adding Multiple Operands

- Adding multiple operands simultaneously
  - For example: Add four operands, 4 bits each
  - $X[3:0], Y[3:0], Z[3:0], W[3:0]$
- Naive approach
  - Step 1:  $T1[4:0] = X[3:0] + Y[3:0]$
  - Step 2:  $T2[5:0] = T1[4:0] + Z[3:0]$
  - Step 3:  $S[5:0] = T2[5:0] + W[3:0]$
  - Delay:  $3 \times$  (delay of 6 bit adder)
  - Area =  $3 \times$  (Area of 6 bit adder)

Size of final sum:  
 $N + \log_2 K$   
 $N$  size of operands  
 $K$  Number of operands

Digital Logic Design: Combinational Circuits



Now, instead of adding just two operands, what do we want to do? We want to add multiple operandings simultaneously the same time. Why do we want to add multiple operands? So, multiple operands, you will see multiple such applications across. So, visible in our regular daily life, we have seen that whenever we do Excel kind of additions, so whenever you go and buy n number of things for the market, so you have to add all the n numbers together.

It is not that we can, we will only add two numbers and then we will give the output and then again add two numbers that is not the way. So, there would be a good number of scenarios where we would like to add n numbers simultaneously. Now, let us say we have to add 4 numbers and each of them are 4 bit. So, the numbers are X Y Z and W. So, whatever we have learned, based on that, if we want to add these 4 numbers, what would be our technique?

We will basically would do a step one where T1 would be generated where T1 is equal to X plus Y. So, what would be our naive approach? In the naive approach, we will add 4 bits of X and 4 bits of Y and we will put it in an intermediate let us say signal. So, that signal we will say what

would be the size of that. Now we have to consider that Cout is also part of the results so instead of calling it Cout, we will say that it is a 5 bit number.

Fifth bit is actually Cout. So, whatever is the carry generated that becomes the fifth bit. Similarly, now, whatever is the result to that result, now, we should add Z. Now, the first operand then is 5 bit. Another operand is of 4 bit, if we add both of them then in the worst case, it could be a 6 bit operand, 6 bit output. So, 0 to 5 that would be a T2. Now, we still have to add W, so in T2 W would be added and I am still writing my sum as a 6 bit number.

Why so because the other will see. So, essentially, if this is the method then overall delay would be equal to, I because I require 4 bit adder so it is not actually 4 bit adder it is actually 6 bit adder. Here, this was a 6 bit adder, here also this 5 bit adder. So, let us say on an average delay of 5 bit adder into three times. Because once this edition is done before this edition is done, we cannot start the second edition.

Similarly, till this edition is done, we cannot start the third edition delay would be. So, delay of this combination would be three times into delay of 6 bit adder. So, 6 bit we are it taking it conservatively. Although the step one would require only 4 bit addition, the step two would require 5 bit addition and step three would require 6 bit addition. So, we can otherwise you could also have taken average of delay of 5 bit adder, but let us say yeah, delay of 4 bit adder plus delay of 5 bit adder plus delay of 6 bit adder that way also we can say.

Similarly area also we can say delay area of 4 bit adder, plus area of 5 bit adder plus area of 6 bit adder. So, this is going to be area and delay of this. And yes, the question, how do we know what would be the number of some bits. So, this is a very generic formula here that if we have  $n$  bits in each operands, let us say here, it was only 4 and how many operands are they, let us say  $K$  operands are there, then the size of the final sum will be equal to  $n$  plus  $\log_2 K$ .

So, this formula can tell us that what would be the size of our final sum. So, here, total number of operands were 4, and the each of the operands was of the size 4. So, essentially, the sum size of my final would be final result would be 6 bit. So, that is why I have taken  $S$  equal to 5 to 1, 5 to 0 that mean 6 bits sum could be the output.

So, now if we had to do this intuitive way where we are, we are adding 1 by 1 what why it is taking so much of time because every time we are doing this carry propagation either using ripple carry or carry save adder carry skip adders or parallel prefix adders. But carry is being propagated in each and every addition and this delay could be huge if the number of operands are large as well as the number of bits in each operand is large. So, can there be a better way.

(Refer Slide Time: 05:36)

The slide is titled "Carry save adders" and features the NPTEL logo in the top right corner. It contains a bulleted list of points:

- Cause of delay: carry propagation
- Carry save adders:
  - Save the carry, propagate in the end
- Each addition: partial sum and partial carry
- Use full adder as building block

Below the text is a diagram showing two full adders. The first full adder has two inputs labeled  $2^1$  and  $2^1$ , and two outputs labeled  $2^1$  and  $2^2$ . The second full adder has two inputs labeled  $2^1$  and  $2^1$ , and two outputs labeled  $2^1$  and  $2^2$ . The slide also includes a small video inset of a man speaking and a footer with the text "Digital Logic Design - Compressed Decimals" and the number "11".

So, to understand this the real problem, first we have to understand the real problem. The real problem is again carry propagation. Carry propagation was our problem in our even in our standard addition of two operands also, but here the problem has aggravated more much more because the number of operands has increased for every operand for every two sum two operands we are propagating carry through the end and then again we are propagating for the next edition also.

So, what could we the alternative? Alternative is a carry save adder and carry save adder we do not we do not propagate the carry till end, but we save the carry and we propagate only in the end. So, once we have calculated, once we have saved all the all the carry's in our last step then we will go ahead and propagate. So, the idea here would be that each addition will generate some partial sum as well as some partial carry. So, these partial carry can also be added through.

So, this partial carry could also be added through another partial carry. So, finally, we can keep on adding all of them and in the last step when we have to do propagation is required then we will do the propagation and then final sum will be calculated. So, the overall advantage would be you would be able to save quite a good amount of time. Now, the basic unit here all we would be using our standard full adder or a half adder.

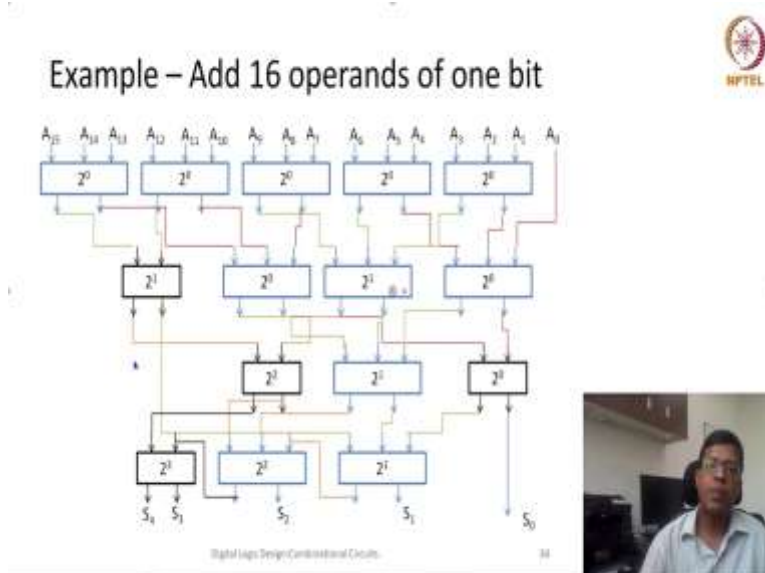
So, in the examples what we are going to show here, we would be representing an adder let us say I am representing a full adder using blue and inside I would write that so, in full adder will have three inputs. So, all of these three inputs, what power of 2, all of these three inputs belong to? So, we have to make sure that whenever we are adding any number of inputs in our full adder, they should be of similar power or similar places.

So, you remember when we do decimal addition or we do binary addition, we can add only those bits or those digits which are of same place. So, if these digits these inputs are belong to 1 place or also belongs to 2 place. Then we can add all of them and then the sum will also be of 2 place and our carry would be of 4 place.

Similarly, if all of them are of 1 place, this is also 1 place this is also 1 place. In that case my sum would be a 1 place, but carry would be off 2 place. So, this particular notion we have to remember and if we use this particular notion, then we can see that how we can use these full adders as a basic building block and then we try to pack all the computation in less number of delays and less number of basically area.

So, area as well as delay both can be optimized very efficiently in this carry save adders. So, to understand this carry save adder let us take a simple example where, yes, it is not so simple also, but let us try to make it simple.

(Refer Slide Time: 09:34)



So, let us take a simple example where we are adding 16 operands, all of them are of single bit. So, basically we are adding all these 16 operands. And all of them are of units place or all at 1's place value. So, what we can do is we can add all of them. We can use our full adders. So, if we use full adders, then we can group all the three bits at one time, three bits at another time. So basically, we can use 5 copies of my full adders.

Each of these full adders is adding units place. So, one particular bit is left behind, so let us keep it like this, let us try to adjust it in whenever we will get someplace or in another set of full adders. So, now in this five full adders, we are adding all of the unit place value or basically all the sixteen operands. Now, what we have to see is that the output here this is of units place this is of twos place.

Similarly, this is of unit this is 2's power 1 place, 2's power 0 place and this is of 2's power 1 place, so on so forth. So, now to add further we can have another set of we can have another set of full adders and on half adder would also be required. So, what we will do is that we will add all the values which are there at 2's 0 place and similarly, we also add all the values which are there at 2's power 1 place.

So, that means now for this particular full adder, I can start from I can add this A0, I can take A0 as one of the input. Similarly, this is also 2's power 0 place. So, I can add this also as a second

input and this is also 2's power 0 place so I can add this as a third input. Now, this is 2's power 0 place, this is 2's power 0 place this also 2's power 0 place so I can keep all of the three inputs in this particular this particular adder.

So, this is the input and then this and the third one. Now, this particular output is 2's power 1 place this is also 2's power 1 place, this is also 2's power 1 place so I can add all of them in using this particular full adder and yeah, this the second one and the third one, so I can add all of them. Similarly, the remaining two this is also 2's power 1 place this is also 2's power 1 place I can add both of them also.

So, using this I have now created the second stage. So, yeah finally we have to add whatever is the output generated here. So, we can we can finally add them also. Now, here I will still require 1 full adder 2 half adders. So, in the full adder I will take this, this is 2's power 0 place and this also 2's power 0 place. So, I will add both of these two inputs I will take both of them as input to my this particular adder, this as well as this.

Now, this is 2's power 1 place this particular output is 2's power one place, this is also 2's power 1 place this is also 2's power 1 place and this is also 2's power 1 place. I have 4 inputs which are all valued off 2's power 1 place. So, what should I select? So, I will select three of them. So, I will select this one, this one and this one. So, I will select this one and I will select this one and I will select this one.

So, if we see this particular output is of 2's power 2 place. So, I can add in another half adder which is of 2's power 2. So, I can add this I am taking an orange color for this and I am adding this 2's power 2 place output is this one. So, I will add this one. Now this is the third stage of my 16 operand addition. Now let us see one more stage. So, here so what are the out of these one two three four five six and seventh ones we have seven inputs which are left.

So, out of them which one is of 2' power 0 place. 2's power 0 place is only coming from this. This is 2's power 1 place this is 2's power 1 place, this is 2's power 2 place, this is 2' power 2 place, this is 2' power 3 place. So, only one input one output have 2's power 0 place so I can directly declared that as a result as 0. Now for 2' power 1 place I can add here in my addition which is getting raises 2's power' 1 place.

So, this is another 2's power 1 place which is spending here I can add this and the one which was remaining from a previous stage I can use that also. So, this particular output was 2's power 2 place. So, I can add it here, this is also 2's power 2 place, I can add it here. The other one will come from here. So, this particular addition when we are connecting this particular carry to the next adder this particular way of connection is called propagation.

So, in all other methods we are saving the carry we were not propagating within the same stage, but here within the same stage we are trying to propagate the carry to the another adder. So, then I can say this one is my sum one first bit of sum and similarly, this would be my second bit of sum. Now, 4th edition have 2's power 3 place. So, there would be two values of 2's power 3 place one is this and the other is which is propagated from this particular carry.

So, now we can say the output here is this is S3 and this is S4. So, as we know that if we are adding 16 operands if all of them are one then the output is going to be 5 bits long. So, I would require 5 bits to represent all that. Now, we see the last stage is the only stage where we are propagating the carry in all other stages we are saving and we are simply forwarding to the next stage. What is the delay of this particular whole adder?

So, the delay would be one full adder delay second full adder delay third full adder delay fourth full adder delay fifth full adder delay and sixth full adder delay. So, the old computation can be done using a using six full adder delay and the total area is also quite less one, two, three, four, five, six, seven, eight, nine, ten, eleven, twelve, thirteen, fourteen, fifteen. So, overall fifteen adders are used out of them eleven were full adder and four were half adders.

If we try to do the same thing known in a intuitive way or in a naive way, we would require sixteen additions one after another delay cannot be less than sixteen full adder delays. But in the later stages when we are adding let us say we have already added let us say seven operands after that my output is going to be my output is going to be 3 bit, sorry 4 bit. So, when the output is 4 bit then every addition would require a 4 bit edition delay.

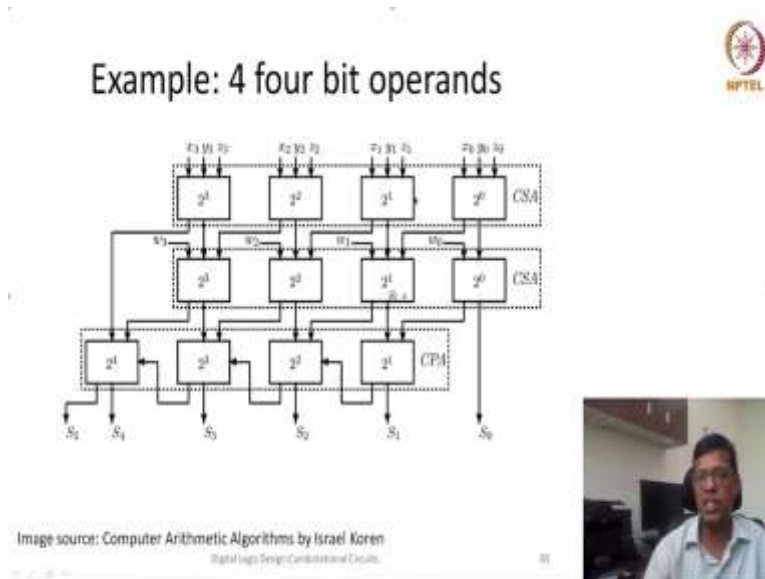
So, the delay in a nave approach is very, very large with respect to this carry save adder delay as well as the area is also quite efficient. So, the overall you may get confused at how do we create these combinations and how do we connect all of them the whole idea is that we have to always

keep in mind that a particular output is what is the place of that particular output, we cannot add two outputs which are of different places. So, in one full adder the all the inputs should be should belong to same place, then we can add them otherwise we cannot add them.

The other way to design this this kind of a tree structure is that we will try to see how many inputs are there. If number of input to a particular place is more than 3, or a multiple of 3, then we can use full adders. If it is less than 3, it is only 2 then we can use half adder. If it is more than 3 your half adder is not sufficient, you can also leave it like we have left this particular output here.

So, then we can keep on building from stage 1 to stage 2 and then stage 3. And then we can finally create a whole tree and we can see how this carry save adder could be created. So, let us try to see using one more example.

(Refer Slide Time: 19:52)



So, in this example what we will do is we will use 4 operands of 4 bit each. So, this particular example has been taken from this particular book computer arithmetic algorithm by Koren. So, yeah, I am directly explaining the example. So, the method is very similar. So, now therefore, 4 inputs X Y Z, all of them are of 4 bits.

So, let us say x is X0 X1 X2 X3. So, in first stage of carry save adder what we do is we take all the three operands X0 Y0 Z0 X1 Y1 Z1 X2 Y2 Z2 X3 Y3 Z3 and we add all of them. Now all of



them so, here you see a different pattern because  $X_0 Y_0 Z_0$  they were of  $2^0$  place these were all 1 of  $2^1$  place these were all  $2^2$  place and these are all  $2^3$  place. So, we can add them together, because their powers are same.

So, in this next level of carry save adder so we, whatever is the output of this  $2^0$  place we can add it here. Now,  $W_0$  was pending. So, this  $W_0$  also we can add in the next stage here the output of the second output that means carry output a belongs to  $2^1$  place. So, we can add in a carry which takes  $2^1$  place as input the some part of this particular adder is also of  $2^1$  place now  $W_1$  is also  $2^1$  place.

So, we can add them in a full adder which takes  $2^1$  place as input. So, similarly, for  $2^2$  power 2 place also the output of this particular edition carry part of this particular edition is  $2^2$  power 2 place. So, similar to this  $2^2$ ,  $2^3$  the second stage of CSA can be completed. Now, after the second stage, there is the only one output which is of  $2^0$  place we can declare that as  $S_0$ .

The next one this can be taken as the output here because this is the  $2^1$ . Now, this is the another input from boost power one. So, this is a half adder which we have used here some would be given here. Now the carry would be propagated to the next stage. Now, the next stage is taking carry from  $2^1$  stage here  $2^1$  adder here and carry of  $2^1$  adder and some of  $2^2$  adder.

Now, similarly this  $2^3$  adder is taking input from these three places carry of  $2^2$  stage here carry of  $2^2$  stage and sum of  $2^3$ . So, similarly  $2^4$  adder. You are also taking three inputs one from the carry of  $2^3$  adder another carry of  $2^3$  adder and carry of  $2^3$  adder which is there also so. Now you see in this last stage carry was propagated from here to here then here to here then here to here.

So, this is the last stage where we are doing carry propagation adder in the first two stages it was all carry save because carry was moving ahead. Within the same stage there were no two adders where we were passing on the carry input from one adder to the other adder. Now, delay of this whole 4 bit operates is equal to full the delay of full adder 1 delay full adder 2 plus delay full adder 3, 4, 5, 6.

So, total delay is equal to six full adder delays. While in our naive approach, we have seen the delay of adding four operands so 4 bit was equal to delay of 4 bit adder plus 5 bit adder plus a delay of 6 bits adder. But here it is actually 6 full adders delay which is going to be less. Similarly area is also quite efficient. The total number of adders we require full adders we require is only 12.

So, this carry save adder presents a good opportunity. It is a different kind of an optimization, where we are playing with the bits we are trying to identify which bits we can add together which belongs to the same place and this can give us an optimization which can save both area as well as the delay.

(Refer Slide Time: 25:03)

The image shows a presentation slide with the following elements:

- Title:** Summary
- Bullet Point:** • Carry select adders and carry save adders
- Logo:** NPTEL logo in the top right corner.
- Video Feed:** A small inset video of a man speaking, located in the bottom right corner.
- Page Number:** 98 at the bottom center.
- Footer:** Digital Logic Design / Combinational Circuits at the bottom left.

So, in our previous approaches like in carry select adder or parallel prefix adders or look ahead adder, we were compromising on area. So, we were reducing we are increasing the area to reduce the delay, but we have seen in this carry save adders, we are reducing both area as well as delay for a particular mutation, but this this case is very selective whenever we are adding multiple operates.

We will see in our next lecture that what is the most common use case of these carry save adder or when do we require to add multiple operands at the same time with this, let us close the lecture. Thank you very much.