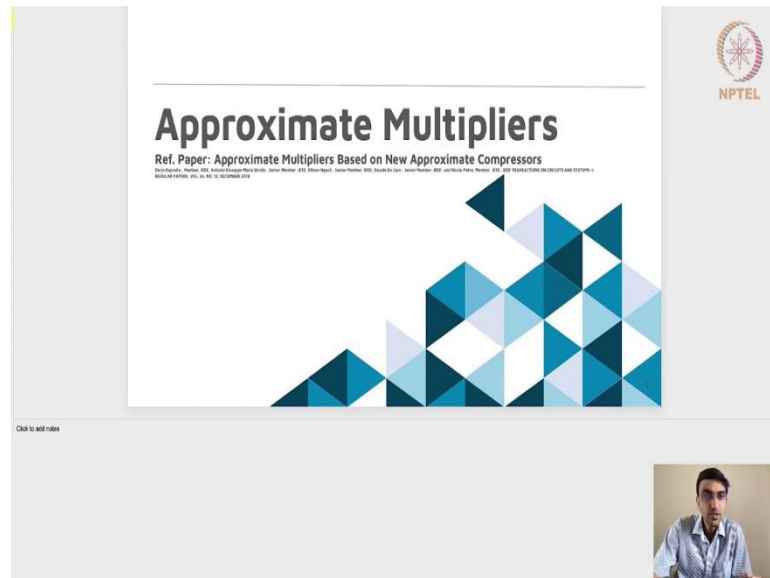


Design and Analysis of VLSI Subsystems
Dr. Madhav Rao
Department of Electronics and Communication Engineering
International Institute of Information Technology, Bangalore

Lecture - 92
Approximate Multipliers - Part 1

(Refer Slide Time: 00:16)



Hello students welcome to this particular lecture. In this particular module we will look into the Approximate Multipliers. It is basically a multiplier circuit and then we will try to understand how the multiplier blocks are designed and then try to go about approximating the multipliers.

There are certain advantages of using the approximate multipliers especially for today's image processing and signal processing applications and we will have a look at one such image processing application.

Most part of this particular lecture is actually taken from this particular reference paper called as approximate multipliers based on the new approximate compressors. The approximate multipliers will go through in achieving the approximate multipliers we will have to understand compressors and we will also try to understand, what does the approximate compressors mean alright.

(Refer Slide Time: 01:16)


NPTEL

Approximate Computing:

- > **What is it ?**
 - Computation technique that returns a possibly inaccurate result rather than a guaranteed accurate result, and that can be used for applications where an approximate result is sufficient for its purpose[1]
- > **Why is it done?**
 - To reduce area, delay and power consumption involved in the application
- > **Where it can be used?**
 - Error-Resilient Applications
 - application to produce acceptable outputs despite some of its underlying computations being incorrect or approximate [2]
 - Such as digital signal processing, image, audio, and video processing, graphics, wireless communications, web search, and data analytics
- > **How it can be achieved?**
 - Strategies
 - **Hardware: Approximate Circuits, Structures and Systems**
 - **Software: Software-level approximation**

[1] <https://www.nptel.ac.in/courses/6-034S19/lecture-10>
 [2] Analysis and Characterization of Shared Application Resilience for Approximate Computing: Vinay K. Chappell, Arvind T. Chakrabarti, Karthik Raj and Anand Raghunathan / School of Electrical and Computer Engineering, Purdue University / System Architecture Department, NCS Laboratory, Austin

Click to add notes



Moving ahead, approximate multipliers comes under this broad domain of approximate computing. It could be an approximate computing as such will involve multiplication or the multiplication and addition, it would also involve a division. You know there are several approximate arithmetic operations that has been useful now in this particular in today's applications.

In that sense it falls into the approximate computing domain. Now, what is it? It is a computation technique that returns a possibly an inaccurate results. You know expecting an accurate results instead of that we are expecting an inaccurate results and to I mean it helps in to reduce the area, delay and the power consumption that will be involved in the circuit design and which goes into the application on the chip or a system application on the chip design.

Now, where it can be used? It is heavily used in most of the error resilient applications where even if you have some kind of an error the it does not really matter that much. Some of those applications are really the digital signal processing and then the image and then the audio and video processing, some of them are also useful in the wireless communication.

In that sense it actually connects the VLSI design to that of the application side. We will talk about the subsystem design, but then eventually it will be useful for the application side. This error resilient application represents that where the product which in the sense whatever is the output of the arithmetic computation and it is kind of an acceptable outputs.

Despite some of the underlying computations being inaccurate or incorrect. The question is how can we achieve this? one of the old methods is to make it software level approximations. But, over the last 4, 5 years there has been a lot of focus on the approximate circuits and thereby getting the approximate outputs which we can eventually design in the hardware alright.

(Refer Slide Time: 03:27)

Binary Multiplication

Involves computing a set of *partial products*, and then summing the partial products together

A multiplier includes:

1. Partial products generation
2. Partial products reduction
3. Carry-propagate addition

Approximations can be introduced in any of these blocks

$$\begin{array}{r} a_3 a_2 a_1 a_0 \\ \times b_3 b_2 b_1 b_0 \\ \hline p_{30} p_{20} p_{10} p_{00} \\ p_{31} p_{21} p_{11} p_{01} \times \\ p_{32} p_{22} p_{12} p_{02} \times \times \\ p_{33} p_{23} p_{13} p_{03} \times \times \times \\ \hline s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0 \end{array}$$

Dot diagram

1st stage 4 to 3
2nd stage 3 to 2
final 2 to 1 bit

bit
half adder
full adder

NPTEL

Image courtesy: The Proposed Full-Depth Multiplier (DFM) - International Journal for Emerging Research in Science & Technology Volume 4, Issue 11, April 2019
DOI number: 2350-0275

3

Click to add notes

That was a very brief discussion about the approximate computing and in this particular lecture we will actually talk about the multiplication. How do we get the approximate multipliers and it is circuit design or the block design to be done. To understand the approximate multiplier one has to understand what is a multiplication, how do we go about the multiplication in the VLSI domain.

The binary multiplication is one such very standard method where it is nothing, but you have this if I consider the 4 bit inputs 2 of the multiplier and multiplicand and if I do a multiplication of that I will have the partial products and the partial products can be designed or it is deduced very very similar to how our manual multiplication method works.

Each of this bit will get doing an AND operation we will get this partial products in the first row. The second bit when it gets you know gets ANDed with the a_0 to a_3 will give us the second row here.

,
Of course, there will be an offset here and the third one will give us the third row here with 2 offsets here and then P_3 when it gets multiplied with the four of the A bits we will get the fourth row of course, with the 3 offsets. If I do an addition of all these things I will get the 8 bits here which will be the product s_0 to s_7 . If I actually look into this particular design in a dot diagram design it is very interesting to see that we call this particular the ANDed operation of b_0 to a_0 or b_0 to a_1 or b_0 to a_2 or b_0 to a_3 or b_3 to a_3 . All these are nothing, but the partial products.

In a multiplier definition all these are nothing, but the partial products each of these dots in this dot diagram represents the partial products. This first level of the dots however, it is arranged it represents this particular P_{00} to P_{33} . The offset here or an empty spaces here represents that it will eventually be filled with 0. Then when it gets added this dot which is nothing, but a partial product will get added to 0 here, it will have any effect. What we really want is while we are actually adding it up and then eventually getting the product results. We want to make the 1st stage here is the case you know this is the partial products that has been generated and then in the 1st stage we make this kind of an arrangement.

The 1st stage will get reduced to from this is the 1st stage. If I consider this particular level this particular three of this partial products can go up here, because it is anyways the addition here and these two can go this one can go here. That is what we will get here the 1st stage we can say that this represents the same thing here, but it is kind of rearranged to some extent. Once I have a rearranged 1st stage then we will apply some half adder or full adder so that we will get the 2nd stage which is of the length of 3, the maximum length of 3.

Here the maximum length is 4 here in the sense maximum length means if I consider any kind of a column here out of the 8 columns here 1, 2, 3, 4, 5, 6, 7 out of the 7 columns here. If I consider the 4th column, that turns out to be having the maximum height or the maximum length.

I am when trying to reduce this the length of 4 into 3 by applying some half adders here. This particular enclosure here which is enclosing the two of the bits is nothing, but representing the half adder. This particular enclosure representing the two dot partial products or the two dots here represents the half adder here. Eventually if I have the half adder here then I will get the 3 bits here, and then 1 bit here and then the 3 bits here, 3 bits

here, 3 bits here. I am actually reducing from 4 to 3 bits here, 3 dots here whatever we can say in the 2nd stage now we will do the further reduction and then get from 3 dots to 2 dots here.

Finally, if I have this particular stage with only 2 dots. That means, that I need to have some kind of an adder a subsystem block which can actually do an addition of 7 bits an addition of this 7 bits. I need a 7 bit addition and then finally, I will get this results.

Here I have written that it is a final carry propagate adder of 6 bit here which means that I am going to take apply the addition up till this particular dots or the bits here leaving this out. This could be anyways we can consider it to be directly as the first LSB of the product bits right, that is what we do.

We have this particular partial product. In an overview the binary multiplication can be recognized into three stages. One is a partial products generation and then kind of rearranging, the second is the partial product reduction which will be like 4:3, from the 1st stage 4 the length of 4 can be reduced to a length of 3.

Then the length of 3 can be reduced to a length of 2 by applying some half adders or full adder designs and then the last one is the carry propagate addition which we do it in the last stage. Once we get the length to be equal to 2, then we apply the carry propagate addition.

We can apply carry skip adder, we can apply carry select adder or increment adder whatever we can apply that to finally, get the eventually get the product bits. Notice that here this is one single dot these two dots represents the half adder and then these three dot represents the full adder design.

We can actually use the full adder and half adder design to convert to reduce to in fact, to in the 2nd stage we call it as the partial products reduction stage. What we say that this is the binary multiplication we can also say that the approximation can be introduced in any of these blocks. Whether it is the 1st stage or the 2nd stage or the 3rd stage or even in fact, we can say that during the partial product generation as well.

(Refer Slide Time: 09:54)

Exact multiplier: Dadda Multiplier

Maximum number of Partial products in a column of each reduction stage is derived from:

$$8 \times 8: 8 \rightarrow 6 \rightarrow 4 \rightarrow 3 \rightarrow 2$$
$$16 \times 16: 16 \rightarrow 13 \rightarrow 9 \rightarrow 6 \rightarrow 4 \rightarrow 3 \rightarrow 2$$

- Benchmark multiplier

Ref: <https://www.geogebra.org/m/9Q5D8m5m5Y>

Fig. Dot diagram for 8-bit Dadda multiplier

4

Here is an exact multiply here is a this is called as a conventional multiplier also called as a dadda multiplier, because it was kind of invented by the author Dadda himself. This is the way the dadda multiplier works where you have the 8 bit multiplication, giving us a 16-bit output.

If you consider the 8-bit multiplication these dots are nothing, but the partial products that are being generated. Then the two the boxes which are enclosing the 2 dots represents the half adder the boxes which are enclosing the 3 dots are the full adders. We apply in the dadda multiplier it keeps applying the half adder and full adder to bring it to a stage where it is only the length is 2 bits. Then finally, apply some the fast adders and then get the a product bits.

In all the stages of partial product reduction from reducing the length here to a length of 6 here to a length of 4 here to length of 3 and then finally, 2. We apply the half adders or the full adders, the sequence for an 8 x 8 bit multiplier is like this the length is 8 here or the height of this the partial product stack is actually 8. In the next stage it reduces to 6, in the next it will reduces to 4, 3 and 2.

$$8 \times 8: 8 \rightarrow 6 \rightarrow 4 \rightarrow 3 \rightarrow 2$$

Similarly, for 16 x 16 bit multiplication it actually reduces start from,

$$16 \times 16: 16 \rightarrow 13 \rightarrow 9 \rightarrow 6 \rightarrow 4 \rightarrow 3 \rightarrow 2$$

In this particular work I think the benchmark multiplier is considered to be the dadda multiplier.

(Refer Slide Time: 11:32)

EXACT COMPRESSORS

Half Adder

A ———> [Half Adder] ———> Sum
 B ———> [Half Adder] ———> Carry

| A1 | A2 | Sum | Carry | A1+A2 | Sum+2*Carry |
|----|----|-----|-------|-------|-------------|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 2 | 2 |

2-1 Compressor: $A1+A2 = \text{Sum}+2*\text{Carry}$

3-2 Compressor: $A1+A2+A3 = \text{Sum}+2*\text{Carry}$

3-2 Compressor: Full adder

A ———> [Full Adder] ———> Sum
 B ———> [Full Adder] ———> Carry
 Cn ———> [Full Adder] ———> Carry

| A1 | A2 | A3 | Sum | Carry | A1+A2+A3 | Sum+2*Carry |
|----|----|----|-----|-------|----------|-------------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 2 | 2 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 2 | 2 |
| 1 | 1 | 0 | 0 | 1 | 2 | 2 |
| 1 | 1 | 1 | 1 | 1 | 3 | 3 |

NPTEL

5

Click to add notes

Let us take a look let us take a quick look at the compressors what do we mean by compressors. When we apply this half adder or the full adder into our partial products so as to reduce the partial product stages we call it as a compressors. It is basically compressing from the higher length of the partial product stack into a lower length.

the very primitive compressors could be the half adder or a full adder here the full adder as we know already a 3 input generates the sum and carry and then half adder is nothing, but 2 inputs will generate the sum and carry. It could also be called as 2:1 compressor the half adder and 3:2 compressors if you use the full adder, here is the truth table for the half adder and full ladder. Moving ahead, let us take a look at the 4:2 compressors.

(Refer Slide Time: 12:23)

The slide is titled "EXACT COMPRESSORS". It features a central diagram of a 4:2 compressor. On the left, a block labeled "4-2" has four inputs x_1, x_2, x_3, x_4 and one carry input C_{in} . It produces a carry output C_{out} and a sum output. To the right, this is expanded into two full adders (FA). The top FA takes x_1, x_2 and C_{in} as inputs and produces a carry C_1 and a sum s_1 . The bottom FA takes x_3, x_4 and C_1 as inputs and produces the final carry C_{out} and the final sum. Handwritten annotations include a box on the left with "4-2" and "Cin = 0", and a box on the right with x_1, x_2, x_3, x_4 and "Sum". Below the diagram is the equation:
$$4-2 \text{ Compressor: } x_1+x_2+x_3+x_4+C_{in} = \text{Sum}+2(\text{Carry}+C_{out})$$

What do we mean by an exact 4:2 compressors? It is actually realized by the 2 full adders here. The 4:2 compressors will involve 4 inputs there will be 1 carry input and there will be a carry output and then there will be a carry here and then the sum here.

If I consider one segment one. Let me use the pointer here. If I have a stack of partial products if I have 4 and then in the neighboring also if I have 4 then I can use this 4:2 compressors.

Then it is going to generate the carry out here C_{out} it is going to generate the sum here and it may take a carry input assuming that is the LSB side. I will say that the C input is 0. I can say that the C input is 0 I can assume that and then there will be a carry out or rather I will say that carry. This carryout is actually going and it will fit it into as a carry input to this particular 4:2 compressors, this will be 4:2 compressors.

Now, whatever goes into carry input this particular output will be the carry input to the next 4:2 or the subsequent 4:2 compressors. This carryout will go in as the carry input in the next stage of the partial product. When we create the partial product this carry in will go into to the next stage of the carry input here.

The sum will be anyways be considered as a dot in the next stage as a dot in the next stage or rather as a partial product in the next stage hope this is clear. That is why the 4:2

compressors will have 2 carry outs, one going into the same stage the second one going into the next stage.

The 4:2 compressors was actually developed using the 2 full adder circuits. The x_4 is the LSB side, x_1 is the MSB side and if you have the 3 MSBs and then the last LSB is actually going into the second or rather the full adder side here along with the carry input.

Whatever is the sum that has been generated here will go into the to the full adder here. Remember that I think what we are doing is we are considering this particular 4 bits as an input and then we ultimately want to find out the sum and then the carry out. It is really not whether x_4 is the MSB, LSB or the MSB it is nothing but,

$$x_1 + x_2 + x_3 + x_4 + C_{in} = \text{Sum} + 2(C_{carry} + C_{out})$$

We will use one full adder for the x_1, x_2 and x_3 and another full adder for this sum whatever is the sum generated and then along with that sum x_4 and then we will have the C input and then create the overall sum and then overall carry alright. Hope this particular understanding is clear about the exact 4:2 compressor.

(Refer Slide Time: 15:56)

EXACT COMPRESSORS

5-2 Compressor: Realized using 3 full adders.

$x_1 + x_2 + x_3 + x_4 + x_5 + C_{in1} + C_{in2} = \text{Sum} + 2(\text{Carry} + C_{out2})$

Delay = 6.XOR

A compressor is a logic circuit that counts the number of "ones" in the input.



Similarly, we can have 5:2 compressors where there are 5 inputs and then there are 2 carry inputs, 2 carry outputs, 1 sum and 1 carry. This can be done using the 3 full adder circuits. I will have the first full adder will have 3 inputs, second full adder will have 3 inputs, third full adder will have 3 inputs.

The only thing here is the inputs are nothing, but the carry input of the first carry input and then the second or the third full adder will have the second carry inputs and it will generate the sum carry and C_{out} and C_{out1} . If I look into the full adder circuits here the full adder circuits can be represented as nothing but x_1 .

If I consider the sum here sum can be represented as an XOR gate, x_1 , x_2 and x_3 here. The Sum 2 can be considered as the S_1 XOR with that of x_4 with that of C_{in} and that to 1. Finally, this particular sum here will be considered as again S_2 XOR x_5 and then XOR with that of C_{in} of 2. If I consider the sum or the delay associated with us extracting the sum output it will be 1, 2, 3, 4, 5, 6, there will be 6 OR gates of delay.

Basically a compressor is nothing but a logic circuit that counts the number of 1's in the input, number of 1's in the sense x_1, x_2, x_3, x_4 and then carry input, carry input 1 and carry input 2.

(Refer Slide Time: 17:51)

The slide displays a grid of partial products for a multiplier. The inputs are $x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0$ and $y_7, y_6, y_5, y_4, y_3, y_2, y_1, y_0$. The grid shows the following partial products:

- x_7y_0 to x_7y_7
- x_6y_0 to x_6y_6
- x_5y_0 to x_5y_5
- x_4y_0 to x_4y_4
- x_3y_0 to x_3y_3
- x_2y_0 to x_2y_2
- x_1y_0 to x_1y_1
- x_0y_0

The summation formula is $S = \sum \{ p_0, p_1, p_2, \dots, p_{j-1} \}$. A red box highlights the partial products $x_3y_3, x_4y_4, x_5y_5, x_6y_6$ in the grid. A blue box highlights the partial products $x_2y_2, x_3y_3, x_4y_4, x_5y_5, x_6y_6$ in the grid. The NPTEL logo is visible in the top right corner.

Compressors are XOR-rich circuits and partial products reduction is a critical multiplier block. XOR gates tend to contribute to high area and delay

If I have this particular the partial products which are been generated. Now, the question is why can I not apply the 5:2 compressors, 3:2 compressors, 5:2 compressors here and then make a much more reduced partial product stage in the next stage.

Ultimately what we want is if I want to create a sum of this it is nothing, but the summation of this partial product, summation of this partial product, summation of this, this and this. The partial product summation starting from 0 to $j - 1$ in the j th column.

The compressors are XOR rich circuits and partial products reduction is a critical multiplier block. The XOR gates tend to contribute to the higher area and delay. If I have some kind of an approximate compressors then it will be highly beneficial in actually leveraging to that of the footprint, the delay and then the power.

We can greatly reduce the we can greatly benefit get benefited in terms of all these three parameters. Of course, we will achieve the inaccurate results, but for some of the applications it may not have that much of an impact.

(Refer Slide Time: 19:07)

APPROXIMATION

- How is the approximation done....???
- Proposed design: By using simple AND-OR gates and not use any XOR gates.
- Eg: To approximate the half-adder, XOR gate of Sum is replaced with OR gate.

$x_1 + x_2$
 $x_1 \cdot x_2$
 $(x_1 + x_2)$

Sum = $x_1 + x_2$
Carry = $x_1 \cdot x_2$

$S = \Sigma\{x_1, x_2\} = \Sigma\{x_1 \cdot x_2, x_1 + x_2\}$

This results in one error in the Sum computation.

| Inputs | | Exact Outputs | | Approximate Outputs | | Absolute Difference |
|--------|-------|---------------|-----|---------------------|-----|---------------------|
| x_1 | x_2 | Carry | Sum | Carry | Sum | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 |

Ref: S. Venkateshram and S. B. Ku, "Design of power and area efficient approximate multipliers," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 25, no. 5, pp. 1762-1768, May 2017.

Approximation how is the approximation done? This is the proposed design in this particular paper is nothing but you can represent the partial products in the form of an AND and OR gates and not use the XOR gates at all. Instead of an XOR gate which will give you an accurate sum right instead of an XOR gate why not use some kind of an AND gate or an OR gate.

Of course, we will get some error. But, if that error will not have that much of an impact then we will get reduce delay. We will also avoid the XOR gate which occupies more of an area and instead use that off an AND or an OR gates.

To approximate the half adder, let us say the sum = $x_1 + x_2$. If I consider the half adder it is only 2 bits, 2 input bits, 1 carry and 1 sum. The XOR gate of the sum is replaced, if I

replace that XOR gate of the sum. The accurate sum of an half adder is nothing but an XOR gate here.

But, instead of an XOR gate if I do an OR gate here and carry is retaining the same it is an AND gate. We will retain the same instead of the XOR here we will do an OR gate here turns out that in the truth table only one error is likely to be appearing.

This is the output here the carry is 1, sum is 0 here. For 1 and 1 for the inputs to be 1 and 1 here in the approximate output I will get 1 here and then 1 here. I will have a problem only in this particular case it is supposed to be 0, but we are getting a value of 1 here. The absolute difference out of the 4 cases is only 1 case.

What we can do is,

$$S = \sum (x_1, x_2) = \sum \{x_1, x_2, x_1 + x_2\}$$

This results in one error in the sum computation. What we are saying is if I consider $x_1 + x_2$ addition of x_1 and x_2 . What I can say that we can actually do an x_1, x_2 and then do an addition with x_1 or x_2, x_1 or x_2 is likely to give us one error here in the sum computation alright.

(Refer Slide Time: 21:44)

AND-OR Re-coding

Let us consider two partial products x_i and y_j of the $(i+j)^{th}$ column of the PPM and let us introduce the following two modified partial products:

$$A_{ij} = x_i y_j \text{ AND } x_j y_i \quad | \quad p_{ij} p_{ji}$$

$$O_{ij} = x_i y_j \text{ OR } x_j y_i \quad | \quad p_{ij} p_{ji}$$

It can easily be observed that:

$$A_{ij} + O_{ij} = x_i y_j + x_j y_i$$

Thus, in the $(i+j)^{th}$ column of the PPM, we can replace the couple of partial products $x_i y_j$ and $x_j y_i$ with the modified partial products A_{ij} and O_{ij} .

For future reference:

$$\sum (x_i y_j + x_j y_i) = \sum (p_{ij} p_{ji} + p_{ji} p_{ij})$$

NPTEL

OR -
AND -

Ref: A. C. Clark et al., "High speed speculative multipliers based on speculative carry-save trees," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 12, pp. 3426-3435, Dec. 2014.

If I proceed further now why not use this AND an OR re-coding for our different size of compressors. What here I have written is in our partial product generation. This is my 8

bit x input and 8 bit y input, if it gets multiplied I will get the partial products. The partial product would be of 16 the length here will be of 8 and 8 bits will make it 16 bits. The height here the largest height will be nothing, but the 8 bits.

Here what I have done is I have represented in different colors just to ensure that finally, this particular representation will make it to an OR gate and then the other color will be represented as an AND gate. What we really need is x_i and y_j x_i and y_j of the $i + j^{\text{th}}$ column. If I consider $i + j^{\text{th}}$ column to be somewhere around this alright.

It will be nothing, but $i + j^{\text{th}}$ column will represent $x_i \cdot y_j$ and it will also represent x_j and y_i , the dot product of without the partial products of x_i, y_j and then it will also present x_j, y_i . If I consider the $i + j^{\text{th}}$ column in the partial product alright. If I actually consider those two things, what it really means is if I consider $x_7 y_2$ in one of these columns and $x_2 y_7$.

If I pick these two partial products and if I want the sum basically we want to add all of them. Why not add these two together. If I want to add these two together. The addition what we can say is the addition of this can also be represented as an ANDing of x of this particular partial product. I can say that this is the partial product $i j$ and then this is the partial product $j i$.

I can actually do an ANDing of P_{ij} and P_{ji} and ORing of P_{ij} and P_{ji} , this actually the summation of these two product these two partial products can be represented as an AND an OR. That is why we say that this is an AND an OR re-coding, thus in the $i + j^{\text{th}}$ column of the PPM we can replace the couple of partial products especially $x_i y_j$ and $x_j y_i$ with a modified partial products of A_{ij} and O_{ij} .

That is what I have done here. I have represented this one as the OR gate and this one as the AND gate. Similarly, for all other 4 terms we will have the OR and then the AND gate. The same partial products what we have obtained from the binary multiplication of this 8 bit, 2 8-bit numbers. It is now represented in the form of an OR gate, the output of an OR gate and then the output of the AND gate.

For the future references what we will do is it is nothing, but $x_i y_j$ which is nothing, but the partial product and if it has to be summed with that of $x_j y_i$. It is nothing, but the partial product summation which will be nothing but the AND operation and then the OR operation and that has to be summed.

(Refer Slide Time: 25:22)

→ PROPOSED APPROXIMATE COMPRESSORS


Considerations:

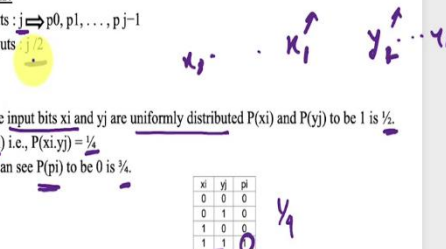
- No. of inputs : $j \Rightarrow p_0, p_1, \dots, p_{j-1}$
- No. of outputs : $\frac{j}{2}$

NOTE:


Considering the input bits x_i and y_j are uniformly distributed $P(x_i)$ and $P(y_j)$ to be 1 is $\frac{1}{2}$.
Therefore, $P(p_i)$ i.e., $P(x_i \cdot y_j) = \frac{1}{4}$
Similarly, we can see $P(p_i)$ to be 0 is $\frac{3}{4}$.

| x_i | y_j | p_i |
|-------|-------|-------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |





11



The output of the OR and then the output of the AND that has to be added together. The proposed approximate compressors what they have proposed in the paper is considering the input bits x_i and y_j that are uniformly distributed. If I consider the partial products or even the x_i .

I will consider x_1 , I will consider y_2 as an individual bit it has a probability, if it is uniformly distributed we say that the probability of x_1 to be 1 is 0.5 and probability of y_2 being 1 is also 0.5. That is what if we consider that uniformly distributed of this particular bit and if all these bits x_1 to x_8 . Similarly, y_1 to y_8 are independent of each other and are uniformly distributed then we can say that each of these bits x_8 to x_1 and y_8 to y_1 will have a probability of being 1 is half.

Therefore, the probability of p_i is probability of the each partial product will be half multiplied by half, because it is the probability of x_i ANDing with that of the y_i which will be 1 by 4th.

Similarly, we can see that the probability of p_i to be 0 is 3 by 4 and that is what is written in this particular table x , y , y_j and p_j . We will have only 1 here out of the 4 cases, that is why it is 1 by 4 of the partial product being 1. Ultimately what we want is if I have the number of inputs as j . I will have the partial products of p_0 to p_{j-1} . Finally, the number of outputs we want to reduce from one stage partial product generation stage to the next stage we want to have j by 2 length and then so on.

