

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

Lecture - 93
Approximate Multipliers - Part2

(Refer Slide Time: 00:16)

$P(p_0, p_1) = 1/4$
 $P(p_0) = 1/2$
 $P(X_i) = 0.5$
 $P(Y_i) = 0.5$


Approximate 2/1 Compressor


- Arithmetic sum of two partial products belonging to the same column:
 $S = \sum \{p_0, p_1\} = \sum \{p_0, p_1, p_0 + p_1\}$

| p_0 | p_1 | $p_0 p_1$ | $p_0 + p_1$ |
|-------|-------|-----------|-------------|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 |

- $P(p_0, p_1) \ll P(p_0 + p_1), \therefore p_0, p_1$ term could be ignored.
- Thus, $S_{\text{approx}} = \sum \{p_0 + p_1\}$

| p_1 | p_0 | s_1 | S | S_{err} | $P(\text{Err})$ |
|-------|-------|-------|-----|------------------|-----------------|
| 0 | 0 | 0 | 0 | 0 | - |
| 0 | 1 | 1 | 1 | 0 | - |
| 1 | 0 | 1 | 1 | 0 | - |
| 1 | 1 | 2 | 1 | 1/16 | 1/16 |





Let us talk about 2:1 compressor. The 2:1 compressor I can consider it to be partial product 1 0 and partial product 1, there were 2 bits. It can be easily represented as an AND and an OR operation here and then the summation of that. If I look into the AND operation here the p_0 itself will be $1/4$ p_1 the probability of p_1 will be itself be $1/4$ because it is a partial products.

$$S = \sum \{p_0, p_1\} = \sum \{p_0, p_1, p_0 + p_1\}$$

The X_i being 1 is the probability of X_i being 1 is 0.5 the probability of Y_i or y_j being 1 is 0.5. The probability of P of p_0 is $1/4$ the probability of p_0 and with that of p_1 will be actually be $1/16$ and probability of p_0 plus p_1 will be nothing, but $7/16$ and that is something we can get from this particular truth table. Remember that if I have this kind of a probability then in fact, if I can avoid this completely in my summation process, because it will have an error of only $1/16$. What we said was the summation of the partial product that is what we are interested in.

The summation of the partial product is also be rewritten as the summation of the OR products and then the AND products. In the AND products the probability of this being 1 is actually 1/16. Most of the cases it will be 0, in that sense if I actually make it ignore it then if I do only the OR operation it will actually have it can reduce the computation.

The AND product is kind of ignored and what happens to the approximate sum. One such truth table is given $p_1 p_0 w_1$ is given w_1 is nothing, but $p_0 + p_1$ is considered to be the w_1 and then the sum is given and then the approximate sum is given this is the accurate sum. If I have 0, 0 it is 0 approximate is also 0 if I have 0, 1 I will have the sum as 1, 1, 0 it will be 1, 1, 1 it will be nothing, but 2.

But whereas my approximate sum will give me the same values except this particular value. The error here is 1 and then the probability of error is nothing but 1/16 coming from straight from this particular value.

What we have done is in a 2:1 compressor instead of the regular sum 2:1 compressor is nothing but a half adder instead of an XOR gate why not represent just with an OR gate.

(Refer Slide Time: 02:59)

Approximate 3/2 Compressor

- $S = \Sigma\{p_0, p_1, p_2\}$

By AND-OR coding:

$$\Sigma\{p_0 p_1\} p_0 + p_1$$

$$\Sigma\{p_0 p_1 p_2\} p_0 + p_1$$

$$\Sigma\{p_0 p_1 p_2\} p_0 + p_1$$

- $S_{\text{apprx}} = \Sigma\{p_0 p_1 + p_2, p_0 + p_1\}$
 $= \Sigma\{w_2, w_1\}$

$\frac{1}{4} * \frac{1}{4} * \frac{1}{4} = 1/64$

| p_2 | p_1 | p_0 | w_2 | w_1 | S | S_{apprx} | Err | P(Err) |
|-------|-------|-------|-------|-------|-----|--------------------|-----|--------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | - | - |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | - |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | - |
| 0 | 1 | 1 | 1 | 1 | 2 | 2 | 0 | - |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | - |
| 1 | 0 | 1 | 1 | 1 | 2 | 2 | 0 | - |
| 1 | 1 | 0 | 1 | 1 | 2 | 2 | 0 | - |
| 1 | 1 | 1 | 1 | 1 | 3 | 2 | 1 | 1/64 |

Going ahead 3:2 compressor can we now approximate 3:2 compressor of course, we can do that summation of the partial products 3 partial products will be involved. In this case and if I consider p_0 and p_1 as 2 of the partial products I can rewrite it as p_0 and p_1 and p_0 or p_1 and leave that p_2 as it is.

Now if I actually do p_0, p_1 use this and this further use it as AND an OR recoding, I will have p_0, p_1, p_2 and then $p_0, p_1 + p_2$ and then this one will be our last stage. Finally, I need to evaluate what is the probability of this particular being 1 and what is the probability of this being 1 and this being 1. Turns out that this is nothing, but

$$S = \sum \{p_0, p_1, p_2\}$$

AND-OR

$$\sum \{p_0 p_1, p_0 + p_1, p_2\}$$

$$\frac{1}{4} \times \frac{1}{4} \times \frac{1}{4} = 1/64$$

The approximate sum for the approximate 3:2 compressors we can actually have as nothing, but the summation of only these 2 partial products. What is that $p_0 p_1$? or that of p_2 and $p_0 + p_1$. Instead of 3:2 compressors will be nothing, but a full adder and instead of 2 XOR gates what we can do is? We can have an OR gate here we can have an OR gate here and then we can have an AND gate here.

This will give me the partial products,

$$\begin{aligned} S_{\text{approx}} &= \sum \{p_0 p_1 + p_2, p_0 + p_1\} \\ &= \sum \{w_2, w_1\} \end{aligned}$$

If I actually use in the truth table here, w_2 and w_1 are actually and in the diagram here. The w_2 and w_1 can be generated by AND an OR gates and in the truth table we can see that the approximate sum and then the expected sum most of them are matching except the last case here which is expecting 3 and the approximate 1 is giving the answer of 2.

Once we get this w_2, w_1 we can actually do the summation of that which will give me an approximate 2 here summation of that and we will give me an approximation of 2 that. Here also we can actually use an XOR gate or we can use an AND an OR recoded adder.

(Refer Slide Time: 05:21)

Approximate 4/2 Compressor

• Arithmetic sum, $S = \Sigma\{p_0, p_1, p_2, p_3\}$

By AND-OR coding: $\Sigma\{p_0 p_1, p_0 + p_1, p_2 p_3, p_2 + p_3\}$

$\Sigma\{p_0 p_1 (p_2 + p_3), p_0 p_1 + p_2 + p_3, (p_0 + p_1)(p_2 p_3), (p_0 + p_1 + p_2 p_3)\}$

$\Sigma\{p_0 p_1 p_2 + p_0 p_1 p_3, p_0 p_1 + p_2 + p_3, p_0 p_2 p_3, p_0 + p_1 + p_2 p_3\}$

$\frac{1}{64} \quad \frac{1}{64} \quad \frac{1}{64} \quad \frac{1}{64}$

• $S_{\text{apprx}} = \Sigma\{p_0 p_1 + p_2 + p_3, p_2 p_3 + p_0 + p_1\}$
 $= \Sigma\{w_2, w_1\}$

14

Click to add notes

We can similarly extend it into 4:2 compressors also we had got 3:2 compressors. The 4:2 compressors will have 4 partial products and we can consider two each $p_0 p_1$ AND an OR recoding we can apply here.

$$S = \sum \{p_0, p_1, p_2, p_3\}$$

$$\text{AND-OR Coding: } \Sigma\{p_0 p_1, p_0 + p_1, p_2 p_3, p_2 + p_3\}$$

$$\sum \{p_0 p_1 (p_2 + p_3), p_0 p_1 + p_2 + p_3, (p_0 + p_1)(p_2 p_3), (p_0 + p_1 + p_2 p_3)\}$$

$$\sum \{p_0 p_1 p_2 + p_0 p_1 p_3, p_0 p_1 + p_2 + p_3, p_0 p_2 p_3, p_1 p_2 p_3, p_0 + p_1 + p_2 p_3\}$$

Eventually we will find it out that this is 1/64, and then this is 1/64, this is 1/64, 1/64 we ignore that. Finally, I think we can approximate it into this particular AND an OR recoded forms. We can ignore this we can ignore this. Finally, we will be left with,

(Refer Slide Time: 06:32)

Approximate 5/3 & 6/3 Compressors

Approximate 5/3 compressor:

- Arithmetic sum, $S = \Sigma\{p_0, p_1, p_2, p_3, p_4\}$
 $= \Sigma\{p_0 p_1, p_0 + p_1, p_2 p_3, p_2 + p_3, p_4\}$
- $S_{\text{apprx}} = \Sigma\{p_0 p_1 + p_2 + p_3, p_2 p_3 + p_4, p_0 + p_1\}$
 $= \Sigma\{w_1, w_2, w_3\}$

Approximate 6/3 Compressor:

- Arithmetic sum, $S = \Sigma\{p_0, p_1, p_2, p_3, p_4, p_5\}$
- $S_{\text{apprx}} = \Sigma\{p_0 p_1 + p_2 + p_3, p_2 p_3 + p_4 + p_5, p_4 p_5 + p_0 + p_1\}$

Click to add notes

Similarly, we can do the 5:3 and 6:3 compressors also eventually by having 5 partial products we can have 3 of them and 5 for having 6 of the partial products we can eventually reduce it further. You can see that it is actually having the 3 of the on the outputs here.

(Refer Slide Time: 06:57)

Higher-Order Approximate Compressors

- Higher order approximate compressors e.g. 7/4, 8/4, 9/5, 10/5 & so on, can be designed by extending the earlier explained method.
- But we can observe that its approximate arithmetic summation S_{apprx} is equal to the sum of the summation of multiple smaller sized compressors.
- E.g.: 7/4 could be realized using 4/2 & 3/2 compressors.
8/4 could be realized using two 4/2 compressors.

Click to add notes

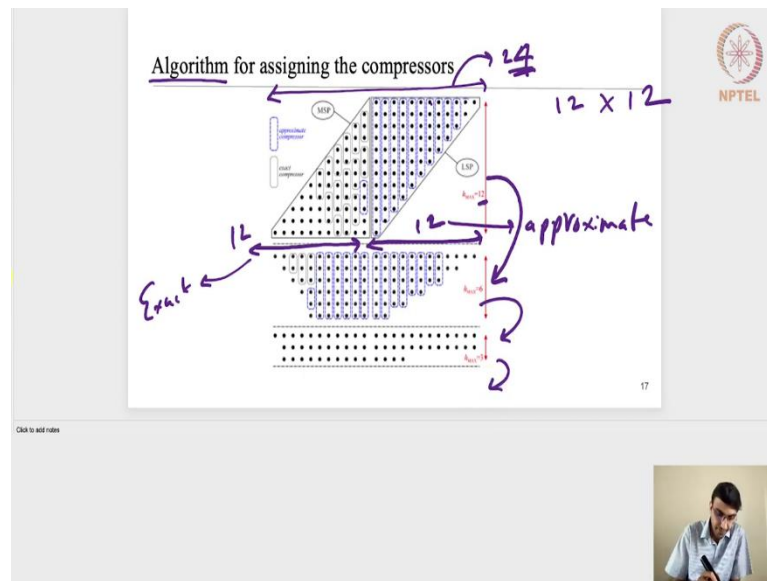
What we have tried here to design is we have approximated the compressors and compressors are actually used for adding up finally, counting the number of ones in the partial products and finally giving us the multiplier.

If you are approximating the compressors we are naturally going to get the approximate multipliers. The higher order approximate multipliers what we have said is generalizing

everything. The 7:4, 8:4, 9:5, 10:5 can also be designed by extending the earlier explained method.

But to keep it very very simple we can also say that 7:4 could be realized using two of the lower order compressors and 8:4 could be realized by having 2 of the 4:2 compressors 4:2 compressors. What we will do is we will restrict ourselves to a lower order compressors and then try to use apply that into the multiplier design and get the inaccurate results.

(Refer Slide Time: 07:52)



This is what we will be doing, this is the kind of a very generic simple algorithm we will be using for assigning the compressors. This is nothing, but 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, we have 12 bit cross 12 bit multiplier, overall I will have 24 columns, this is 24 columns and the length here will be nothing but 12.

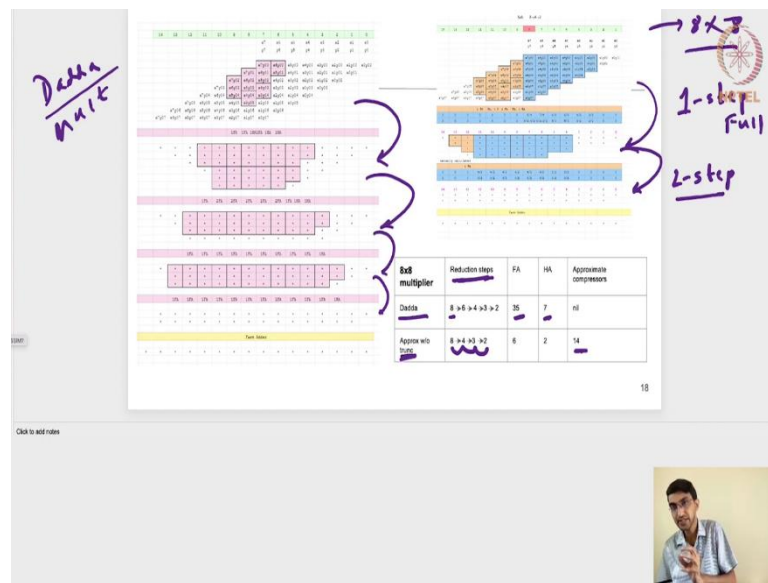
The first partial product generation stage will have 12 and for approximating what we will do is the LSP side the lower bit side in the 16 columns or in the 24 columns, 12 bit columns the 12 columns the starting 12 columns will be the approximate 1 approximate compressors. The last 12 columns will be the exact compressors. On the here I think we have we will be using the approximate 1, but rest of the cases it is the exact compressors. We will do the partial product reduction stage and then go to the next stage.

In the next stage also we will see that we will apply the approximate compressors to an extent to an extent where the height is maximum. Here the height was maximum here, we

divided the LSP side we divided the region two segments and then applied on one segment we applied approximate compressors on the other side we applied the exact compressors.

Here also, once we get this get into the second partial product reduction stage we will identify the maximum height here and once we get the maximum height here we will apply up till that particular maximum height we will going to apply the approximate compressors. And lower than that we will going to apply the accurate compressors. Finally, we will keep on reducing the number of stages.

(Refer Slide Time: 10:01)



Here is the overall design of an 8/8 x 8 bit multiplier and this is the regular data multiplier which uses the half adder and full adder right. Does not use any 4:2 compressors or 5:2 compressors. It uses only half adder and full adder, it is much more an accurate designs and here it is an approximate design. The approximation here is done is because on the LSB side we have approximated the compressors itself and on the MSB side there is an exact compressor, but while it is kind of reduced from here to here you can still see that maximum height on right of some maximum height everything is in approximate compressors left of that it is the exact compressors and then so on.

The data multiplier actually has the sequences from here to here from here to here from here to here and then finally, this one and then the fast adders. This one 8:8 bit multiplier with approximate compressors is giving us one reduced stage and then from one reduced stage to the next reduced stage and then you can directly achieve the fast adders.

Comparing with that of the data multiplier here I think this is much more efficient in terms of hardware because the number of stages has been reduced. The sequence of the reduction stages reduction steps for an 8 by 8 bit multiplier of the data multiplier type it is eight to 8 to 4, 3 and 2.

The number of full adders that has to be used is 35 half adders it is 7. Approximate I have written without truncation I will come to that later. The approximate multipliers which we have used here is 8:4 and then 3:2, there is the number of stages the reduction stages has drastically reduced and the approximate compressors that has been utilized is 14 alright.

(Refer Slide Time: 11:55)

| 16x16 multiplier | Reduction steps | FA | HA | Approximate compressor |
|-------------------|-----------------------------|-----|----|------------------------|
| Dadda | 16 → 13 → 8 → 6 → 4 → 3 → 2 | 195 | 15 | 14 |
| Approx with trunc | 16 → 8 → 4 → 2 | 27 | 3 | 55 |

Moving ahead, this is a 16 x 16 bit multiplier design and this is the data multiplier design. Using the half adders and full adders and then note there is a sequence here of reduction here the sequence of reduction is very very simple, 1, 2, 3, 4, 5, 6, 7. There are 7 stages here, it is only 4 stages the number of approximate compressors is 55 here.

Here the full adders and half adders are 195 and 15 here it is only 27 and 3. Remember that what we have done is we have applied on to the LSB side we have applied the approximate compressors and on the MSB side we have applied the exact compressors.

(Refer Slide Time: 12:45)

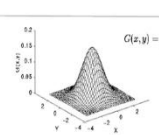
How Smoothing is done?

- Gaussian smoothing is 2D operator and is a spatial domain smoothing technique.
 - It uses a kernel that represents the shape of a Gaussian ('bell-shaped') hump [1]
- Kernel used:
$$\begin{bmatrix} 0.0947 & 0.1183 & 0.0947 \\ 0.1183 & 0.1478 & 0.1183 \\ 0.0947 & 0.1183 & 0.0947 \end{bmatrix}$$
- Obtained from MATLAB:

```
fspecial('gaussian', [3 3], 1.5)
```


$\sigma = 1.5$ and zero mean
- Image Filtering is done in Verilog using set of the implemented exact and approximate multipliers
- Output images are used in MATLAB to get the SSIM value

Image source and [1]: <https://www.researchgate.net/publication/261222222>



2-D Gaussian distribution with mean (0,0) and $\sigma=1$

The standard deviation, σ , determines the width of the filter and hence the amount of smoothing. A filter with a large σ will suppress much of the noise, but also smooth away the weakest edges.



Then we have applied just to understand or see the impact of on the application side this particular approximate compressor driven multiplier design is actually used for the Gaussian smoothing application. Gaussian smoothing application is one of the very regularly or very frequently used filtering method just to looking at the Gaussian shape here it gives us a kind of a blurring it applies a blurring on top of that particular on that particular pixel values. In this particular case if I consider this as an original image here and if I apply the Gaussian smoothening here on top of this. I am likely to get some kind of a blurred image and in kind of this blurring image is actually very very useful in some of the filtering process, one of the heavily used image processing application.

(Refer Slide Time: 13:35)

What is SSIM?

- The Structural Similarity Index (SSIM): Perceptual metric that quantifies image quality degradation
- It outputs the similarity of the given image with respect to reference image (assumed to be of perfect quality)
- Considered to be an apt measure of image quality than traditional Peak-Signal-to-Noise Ratio (PSNR) and Mean Squared Error (MSE) as it takes into account the structural information in that just absolute errors.
 - structural information is more relevant to images as spatial distribution of pixels holds a strong amount of information about the image

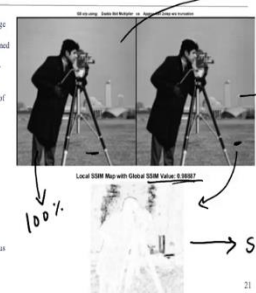
$ssim(I, R) \in [0, 1]$

1. Smoothed image that used Dadda multiplier
2. Smoothed image that used approximate multiplier

If both the images are exactly similar, SSIM = 1 else will be less than 1

Result expected:

- Depends on the image considered. SSIM evaluated pixelwise.
- Approximate multiplier with truncation deviates more from the outputs of Exact/Dadda multiplier, compared to approximate multiplier w/o truncation. Thus the Gaussian Smoothing that uses Approx. mul with truncation should be more structurally dissimilar than a lower SSIM value.
- I.e., $(SSIM)_{approx, \text{w/ truncation}} < (SSIM)_{approx, \text{w/ out truncation}}$



In this particular Gaussian smoothening application the multipliers are a more frequently used arithmetic operators. Instead of using an exact multiplier if we use the approximate multipliers how does it behave.

This particular slide talks about applying the approximate multiplier in the Gaussian smoothing filter and what is the output of that. If I have an original image of a cameraman original image and if you build the Gaussian smoothing that the filter using the exact or the data 8 bit multiplier here this is what the image it looks like.

It is compared to the original image it looks little bit blurred and that is what we expect from the Gaussian smoothening filter and this particular output on the right side is from the Gaussian smoothening filter when the filter is actually developed from this approximate 8 bit multiplier.

If I actually compare between these 2 images there is hardly any difference only thing you can see is there is little bit of that the data is kind of lost, but I think all the prolific information or all the relevant information which is kind of the image is rendering is still present. With respect to these 2 images we will still get the same kind of information that is required. That is why we say that the approximate computing is used especially for some of the error resilient applications and image processing happens to be one of them.

To find basically the statistical difference between these 2 images one it is output of the exact multiplier and then one is the output of the approximate multiplier there is something called as a structural similarity index that is a parameter which quantifies how much of the output is kind of degraded.

If I consider the output of the exact multiplier, this particular image and if we find out what is the SSIM value structural similarity index turns out to be this is what the benchmark is, that is why it is SSIM is 100%. If I find out the SSIM for this approximate 8 bit multiplier which is kind of adopted in the building the Gaussian smoothening filter the output of this is giving us around 98.88%. This is what we are getting the SSIM the structural similarity index value. If I can find out the difference quantifyingly or you know statistically it is only a degradation of around 1.12% which is very very small.

This particular image is about after running the output image into the SSIM module function we will get this particular image, and from this particular image we will be able to calculate or estimate the SSIM value.

If I consider the SSIM function on the output of the Gaussian smoothing I will likely to get 1 particular image which rounds about 200%. The SSIM of this particular output image turns out to be like this and this is giving us around 98.88% of SSIM.

(Refer Slide Time: 17:16)



This is the overall output of the images using the 8 bit multiplier, and then the 16 bit multiplier using the approximate compressors. Here, I have different types here I have said one step full one step truncated 2 step full and 2 step truncated. I need to go back to the 16 bit designs or the 8 bit designs and then try to explain what does that 1 bit and 2 bit represents right 1 step full and 2 step.

In this sense in this particular the partial products stages or the representation of the partial products and its reaction stages. The 1 step full represents, we are applying the approximate compressors only on the first stage and then from the second step onwards there is no approximate compressors at all. That becomes 1 step and it would be applied for an 8 bit multiplication or a 16 bit multiplication. The 2 step full represents we are applying the approximate compressors on 2 on both the stages not only in the first stage reduction, but also in the second stage reduction which will go into the third stage.

We will have 2 stage or the 2 step where we will apply the approximate compressors. There is one more term called as the truncation, truncation represents that in this the lower LSB side everything is truncated to 0 only we are considering only about the MSB side the higher order significant bits.

The first 8 bits we will truncate it to 0, whatever is the partial product we will for a 16 cross, 16 bit the first 8 bits for first 8 columns will all be truncated to 0, because we know that even if we make this the LSB side 0 the MSB sides will still have more weight age.

We can easily do the exact compressors or the approximate compressors and get the values get the product values, but having truncating the lower LSB side is not going to have that much of a difference in the values between the inaccurate results and then the exact results. That is what it is called as the truncation. Having this for an 8 bit and then a 16 bit we can actually have 1 step without truncation, 2 step approximate compressors without truncation, 1 step approximate compressors with truncation 2 step approximate compressors with truncation.

I can actually do it for 8 bit multiplier as well as an 16 bit multiplier and in the end we will actually get 4 designs for an 8 bit and 4 designs for the 16 bit. This particular row is for the 16 bit multiplier and this particular row is for the 8 bit multiplier.

The 8 bit multiplier we can easily see that the SSIM value turns out to be 99%, 99%, 98% 98% when for the 1 step if I apply the approximate compressors without doing the truncation this will be the highest. If I apply the truncation it will be slightly less than that of the previous value, 2 step if I apply the approximate compressors there it will go down to 98% 2 step application of the approximate compressors along with the truncation will give me 98% even lower than this.

If I use the 16 bit and do the 2 step full and then 3 step full. Instead of in the 16 x 16 bit it is not only the first stage second stage, but also third stage is available because the higher number of columns are there. Then the length is also higher, it will eventually reduce it to we can easily apply the approximate compressors even to the third stage.

In that particular case if I have 2 step full means 2 step 2 stage where we will apply the approximate compressors we will get the SSIM value as 0.99. But, if I have the truncation in a 16 bit multiplier this there is a significant loss here. The 2 step 2 stage truncation is

not good whereas, the 3 stage applying the approximate compressors is still better the 98 percent of the SSIM value is achieved.

But, 3 step again with a truncation making the LSB part to be completely 0, it also degrades very very low. In this particular lecture what we have seen is how do we do an approximate compressors using an AND an OR recoding and then actually neglecting the part which has a very less probability of being 1 and then use that for putting the approximate compressors and use that particular approximate compressors into the multiplier design and we have seen the 8 x 8 bit, 16 x 16 bit multiplier design and then if those multiplier designs if we use it in a Gaussian smoothing application what will be its impact. It is impact in terms of the SSIM is not it is almost close to the 100% it is giving us the results of 99 and 98%.

But overall, the area the footprint of the design of the multipliers on the chip the power consumption for the 8 bit or a 16-bit approximate multiplier as well as the delay the critical path reduces significantly. All these could be leveraged without causing much of an SSIM degradation in the final application side.