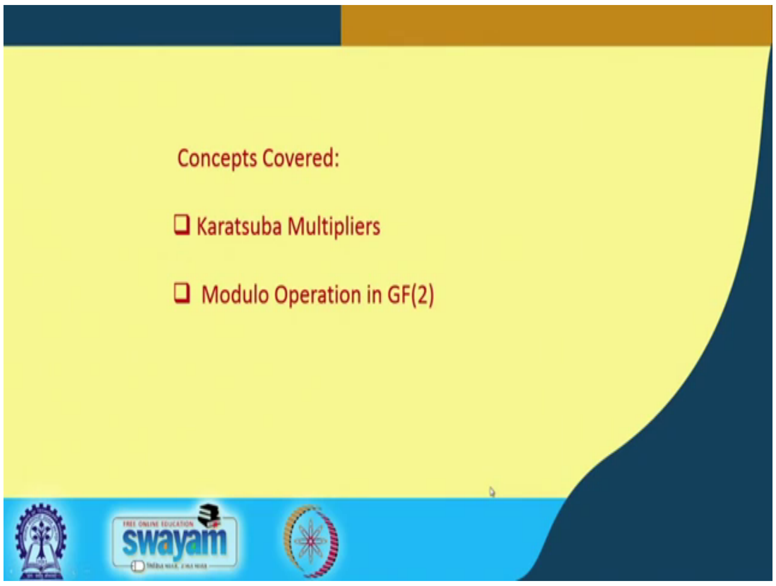
Hardware Security

Prof. Debdeep Mukhopadhyay
Department of Computer Science and Engineering
Indian Institution of Technology Kharagpur

Lecture – 05
Finite Field Architecture-1 (Contd.)

So, welcome back. So, today we will be again looking back into our Karatsuba multiplier. So, I will continue with what we were discussing in the last class.

(Refer Slide Time: 00:25)

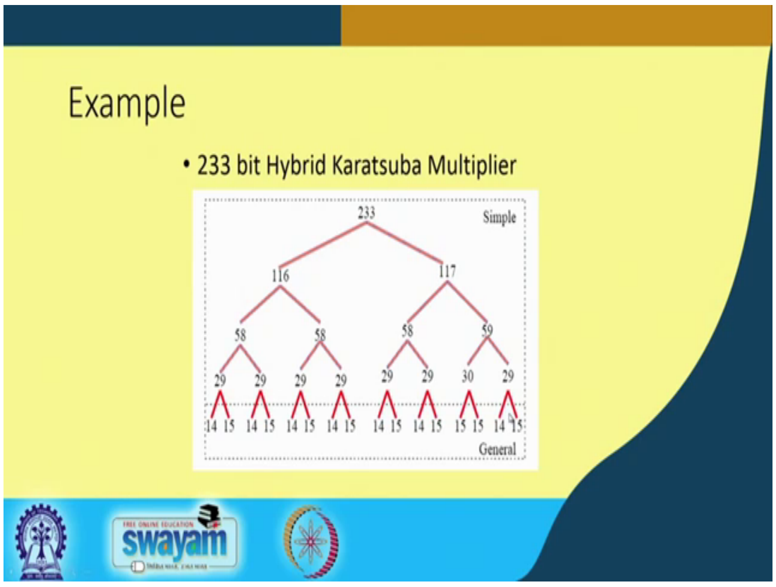


Concepts Covered:

- Karatsuba Multipliers
- Modulo Operation in GF(2)

And will be rather trying to cover these topics on how to implement a end-to-end Karatsuba multiplier. We will be also discussing another important operation which is called as a modulo operation. As we have seen in the last class that when we are doing multiplications, then we are splitting right I mean then we are spilling about the field. So, we need to again do a modulo operation, so that the result gets back into the field. So, there are very efficient ways of doing that, but in particular when you are talking about GF 2 arithmetic ok.

(Refer Slide Time: 00:53)



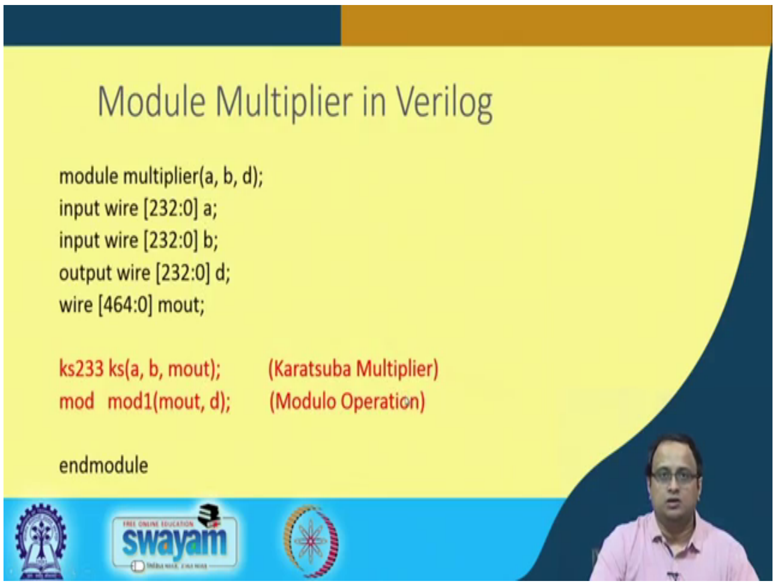
Example

- 233 bit Hybrid Karatsuba Multiplier

So, this is a reminder of what we where we stopped in the last class. So, we want to implement 233 bit multiplication, on 233 bit high bit Karatsuba multiplier. So, the strategy is that I take 233, and I start decomposing them like 116 and 117. So, I take two parts.

And then I can take 116, I can again decompose into say blocks of equal size of 58; and 58 117, I will again decompose at 58 and 59. And then I will do that gradually until I have got dimensions which are lesser than my threshold. The moment my dimension becomes lesser than the threshold, I switch my strategy, I apply either the general Karatsuba or if I may even apply my schoolbook algorithm. But the main point is that I need to switch a bit the proper choice of the threshold. So, one should vary the threshold and see for what kind of proper thresholding right for your platform you get an efficient performance.

(Refer Slide Time: 01:47)



Module Multiplier in Verilog

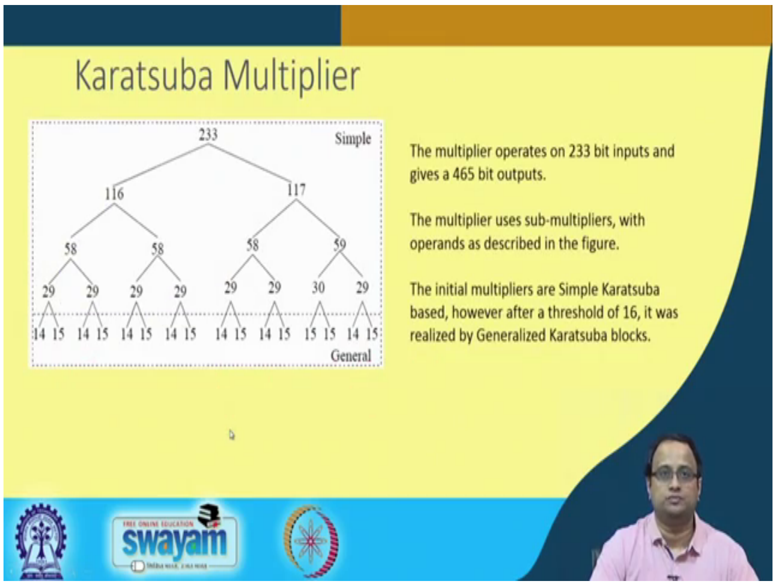
```
module multiplier(a, b, d);
input wire [232:0] a;
input wire [232:0] b;
output wire [232:0] d;
wire [464:0] mout;

ks233 ks(a, b, mout);        (Karatsuba Multiplier)
mod   mod1(mout, d);         (Modulo Operation)

endmodule
```

So, if I want to implement, so I will take an example from verilog program I mean verilog hardware description language or h d l. And we will discuss about how to implement the Karatsuba multiplier in hardware. So, therefore, what I say it is a it is a (Refer Time: 02:03) block, so I have got two inputs or dimensions 233 bits, because a and b are both GF 2 power of 33 elements. The final result is d which is again in the field. So, therefore, the result is also in to 233, it is ok.

Now, when I am doing a more normal multiplication with a and b, I know that my result can go up to 465 bits ok, because when I am multiplying two numbers then essentially can there can be an increase in the field size. So, therefore, in this case right when you are talking GF 2 part of 233, their degree is maximum 232. So, likewise for both for a and b. So, if I take a x and b x, and if I multiply them then my degree can go up to 232 into 2 which is 464.

So, therefore, right what I need to do is I need to do a modulo operation. On a modulo operation with my you know like with a specific primitive polynomial, so the result essentially gets back into the field. And I finally, get d which is again belonging to GF 2 power of 233 ok. So, how do I realize all these blocks?

(Refer Slide Time: 03:01)



Karatsuba Multiplier

233
Simple
116
117
58
58
58
59
29
29
29
29
29
29
30
29
14 15
14 15
14 15
14 15
14 15
14 15
15 15
14 15
General

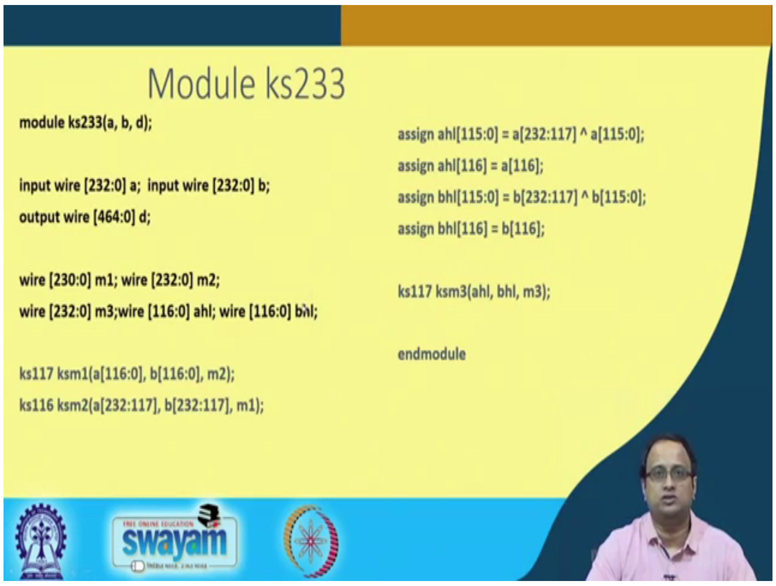
The multiplier operates on 233 bit inputs and gives a 465 bit outputs.

The multiplier uses sub-multipliers, with operands as described in the figure.

The initial multipliers are Simple Karatsuba based, however after a threshold of 16, it was realized by Generalized Karatsuba blocks.

So, as I said that when you are taking a Karatsuba multiplier and you would like to implement a Karatsuba multiplier, so this multiplier operates on 233 bit inputs and gives you a 465 bit output. And the multiplier uses sub multipliers with the operands as we have described in this figure. The initial multiplier is of course, a simple Karatsuba base, but however at threshold of I mean there has to be a specific threshold after which we are switching right, this is the overall strategy. So, when you are so basically like when I let us just first concentrate on the simple Karatsuba decomposition, when we have taken two arguments of dimensions 233 bits. How do we decompose them into dimensions of 116, 117, and do the Karatsuba operation?

(Refer Slide Time: 03:49)



Module ks233

```
module ks233(a, b, d);

input wire [232:0] a;  input wire [232:0] b;
output wire [464:0] d;

wire [230:0] m1; wire [232:0] m2;
wire [232:0] m3;wire [116:0] ahl; wire [116:0] bhl;

ks117 ksm1(a[116:0], b[116:0], m2);
ks116 ksm2(a[232:117], b[232:117], m1);

assign ahl[115:0] = a[232:117] ^ a[115:0];
assign ahl[116] = a[116];
assign bhl[115:0] = b[232:117] ^ b[115:0];
assign bhl[116] = b[116];

ks117 ksm3(ahl, bhl, m3);

endmodule
```

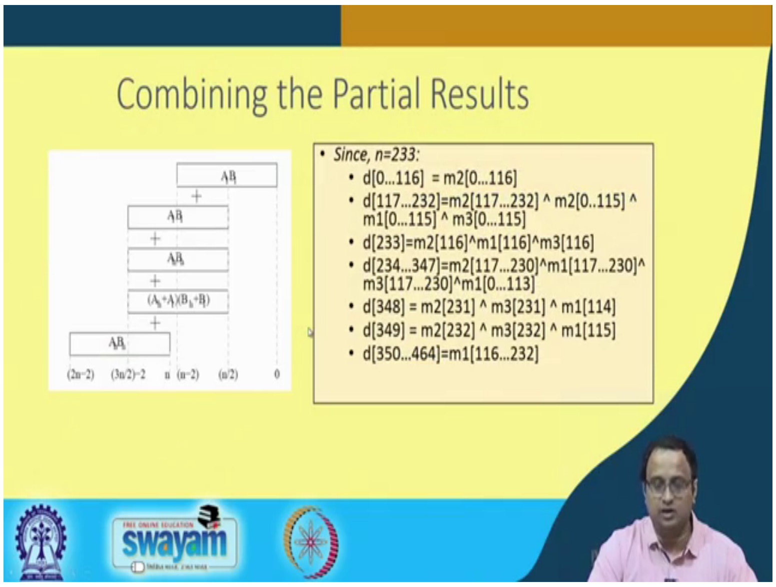
So, there are some you know like something that, so this is the overall structure of the operation. So, this is the code for doing the Karatsuba operation. And you see that I just take two elements like 233 bits each and I want to apply the Karatsuba operation. So, when I am doing a Karatsuba as I said right then I have got three multiplications that I am doing ok. So, therefore, the three multiplications are denoted as ksm 1, ksm 2 and also there is a ksm 3. So, these are the three instances of doing the Karatsuba operation.

So, as you see that the first thing that I have done is that I have split up into two parts. So, one which has got from 0 to 116 which is one part; and the other one which is like from 117 to 232 which is the second part. Likewise b is also split from 0 to 116 which is one part; and b 117 to 232 with is the other part. So, we often say that this part is the lower part, and this part is the higher part of a. Likewise this is the lower part of b, and this is the higher part of b.

So, when I am multiplying these two, I am getting the result in m 2. When I am multiplying these two, I am getting the result in m 1. So, when I want to do the other Karatsuba multiplication, then I have to take you know like the higher part and the lower part of a. And I have to do an XOR. So, therefore, I do an XOR of the lower part of a with the higher part of a; likewise I do an XOR of the lower part of b and the higher part of b. And then I essentially take the lower parts and the higher parts. And then finally, I do a Karatsuba essentially you know like between the ahl and the bhl part. So, you can observe here that I am essentially doing a multiplication between h plus al and bh plus bl.

So, therefore, you know like I am basically trying to multiply h plus l which is the higher part plus the lower part of a plus bh plus bl which is the higher part plus lower part of b. And I am getting the result finally, stored in m 3. So, now when we have done that, so when we have basically you know like got the result, then I should get the result which is a d, but d is now a 465 bit value. So, therefore, I of course, need to do a final modulo operation.

(Refer Slide Time: 05:57)

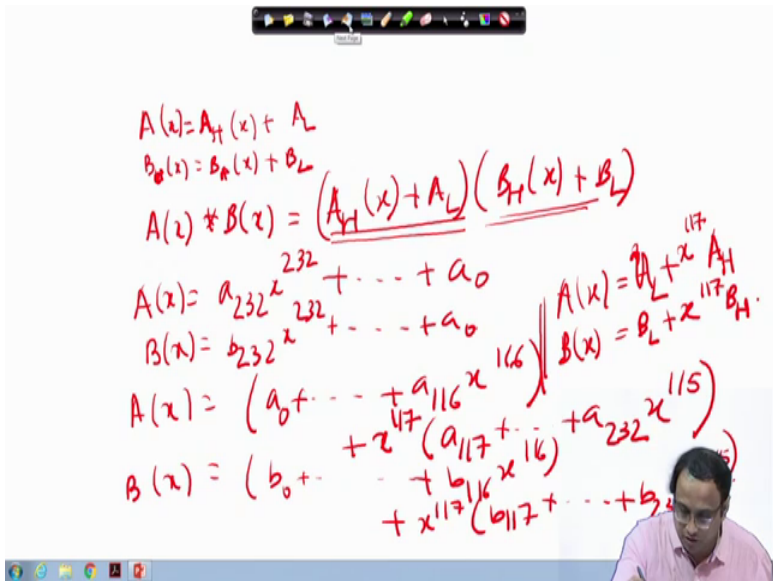


Combining the Partial Results

- *Since, n=233:*
  - d[0...116] = m2[0...116]
  - d[117...232]=m2[117...232] ^ m2[0..115] ^ m1[0...115] ^ m3[0...115]
  - d[233]=m2[116]^m1[116]^m3[116]
  - d[234...347]=m2[117...230]^m1[117...230]^ m3[117...230]^m1[0...113]
  - d[348] = m2[231] ^ m3[231] ^ m1[114]
  - d[349] = m2[232] ^ m3[232] ^ m1[115]
  - d[350...464]=m1[116...232]

So, now, let us see how we can do this operation, that means, how we can actually you know like do this 233 do basically do the cards (Refer Time: 06:05) operation. So, let us just focus on how we can combine the different components. So, for this right I mean we can actually again you know like we sort to we can actually see how the operation is done, and let us try to see how we can do this operation.

(Refer Slide Time: 06:25)



$A(x) = A_H(x) + A_L$

$B(x) = B_H(x) + B_L$

$A(x) * B(x) = \underline{(A_H(x) + A_L)}\underline{(B_H(x) + B_L)}$

$A(x) = a_{232}x^{232} + \cdots + a_0$

$B(x) = b_{232}x^{232} + \cdots + a_0$

$A(x) = (a_0 + \cdots + a_{116}x^{116}) + x^{117}(a_{117} + \cdots + a_{232}x^{115})$

$B(x) = (b_0 + \cdots + b_{116}x^{116}) + x^{117}(b_{117} + \cdots + b_{232}$

$A(x) = A_L + x^{117}A_H$

$B(x) = B_L + x^{117}B_H$.

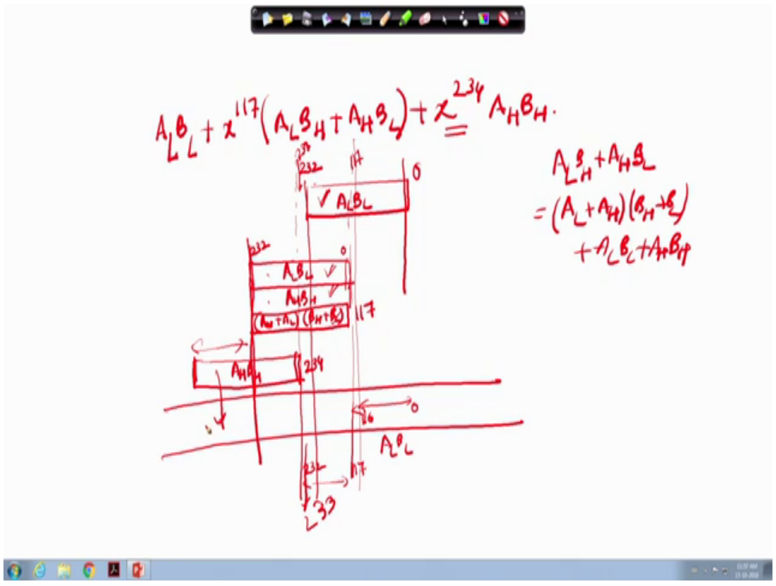
So, so let me first just take you know like. So, this is your A H plus bl. So, let us write just A H x plus A L. So, this is nothing, but A x. So, A x is my polynomial which has got two parts A H and A L. Likewise right I have got B H x which I have written as sorry B x which is written as B H x plus B L. So, this is my high part and this is my low part. And when I want to do A x into B x that means, I want to multiply A x with B x, then essentially write I mean we can observe that I need to multiply A H x plus A L with B H x plus B L.

So, let us take an example. So, in our case A x is nothing but a 232 x power of 232 plus so on till a 0. So, all these coefficients are 0 1 elements. Likewise I take B x which is b 232 x 232 plus so on till a 0. So, A x is therefore, equal to if I take a 0 plus so on till a 116 x part of 116 this is the lower part of a plus x power of 117 multiplied by a 117 plus so on till a 232 x 115 ok.

So, essentially right I mean whenever when you are doing an operation between A x and B x, so likewise right you can actually take B x and also write in this form which is b 0 till plus b 116 x 116 plus x power of 117 b 117 plus so on till b 232 x power of 115. So, therefore, right what you can actually do is in a compact manner, you can write this A x as A x equal to A L plus x power of 117 A H. And likewise you can write B x as B L plus x power of 117 B H ok.

So, let me correct this and write that this would be something like, so this is just to imply you know I do not take it as a multiplication with x, but this is just to indicate that I am combining the higher part with the lower part, but the combination is done in this way. So, therefore, right when we when we when we would like to multiply them, then you can easily observe then how the how the multiplication works ok. So, therefore, let me clear this, and just try to write the product ok.

(Refer Slide Time: 09:39)



So, this is easy. So, we can what we are basically doing is that the product is A L B L plus x 117 A L B H plus A H B L plus x power of 234 A H B H ok. So, this is my final result. So, now, what we have what now we can try to observe this in a figurative way it is way figure ok.

So, here what we see is that this is my lower part. So, this lower part is essentially aligned with the right, that means, this is aligned with 0 for example. So, this is my A L, B L. So, this starts in this case from 0 and continue still 232 bits. Likewise we see that this part that is A L B H plus A H B L. So, this part that is A L B H plus H B L which we have represented as A L plus A H into B H plus B L plus A L B L plus A H B H. So, this decomposed into three parts. So, this actually starts from 117.

So, therefore, if I draw this line, there is a line at 117 from which these three parts are essentially starting. So, essentially what I have got is three parts. And the three parts essentially are nothing but A L B L ok; likewise A H B H A H plus A L and multiplied with B H plus B L. So, this is my third part. So, this starts from 117 bit position and essentially right if you start or if you say that this is 0, then this is also 232 because it is 0 to 232 ok.

And now one can observe that there is another part which essentially starts from 234 ok, so that means, if I leave out some bits for that. So, essentially I start from here, and I get this part which is nothing but A H B H. So, A H B H again starts from 234 bit position and that you can observe here that is it is x to the power of 234. So, it starts here and essentially continues and comes here or comes over here.

So, now, what we have to do is we have to combine these terms; you have to combine these terms to get the final product. So, one you can one thing which you can easily observe is that from 0 to 116, that means, essentially before 117 this is just nothing but the copying of the A L B L. So, therefore, this part is very straightforward right I mean you just need to copy from 0 to 116. And that is nothing but copying A L B L which is nothing but essentially your you know like you have just copied the lower part actually.

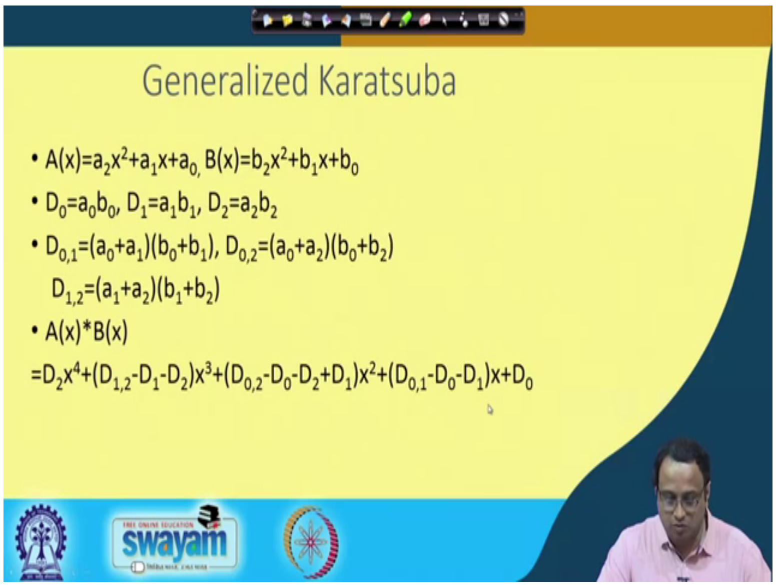
However, if you just observe the 117 bit, essentially the 117 bit will have or rather from 117 to 232, you see it is the addition of three components. These three components are being added up ok. So, that continues from 117, so this continues from 117 bit position to 232. So, we are basically adding up this m 1, m 2 and m 3 - three parts ok. The 233 bit right is something which is interesting, because it comes in between you are still adding up you are essentially so in this case when you are when you are doing this addition, yeah you would have actually you have got four parts, this part, this and this and this. So, we have got four parts which you are adding.

But for only one bit position, that means, the 233rd bit position you are only adding this, this and this. So, there are three bits which you are adding ok, so that gives you the 233rd bit. On the other hand right, for the remaining parts that is 234 till again the end of this, we again for certain position you have got or sudden bit positions, you have got three parts which you need to add; and for the remaining parts you just need to only add the you just need to copy the A H B H values.

So, this is the summary of how the overall computation looks like understand the correspondence here. So, here you can observe that as I say that the remaining parts are I mean the lower parts are just copying of one of the lower part which is m 2, we have just copied it further. From this part to this part you are that is 117 to 232, you have got four registers which you are XORing. For the 233rd third position, you have just three bits to XOR.

Again when you are going for the 234 to 347, now, these 347, you can easily calculate by the relative position. And these are here you have got again 1, 2 and 3 and 4 values to XOR. And then finally, you have got the remaining 348, 349 which is an XOR of three bit positions or three registers. And finally, you have got the higher part which you are just copying. So, this gives you our mechanism of how you can combine the partial results, and you can get the corresponding Karatsuba output.

(Refer Slide Time: 14:45)



Generalized Karatsuba

- $A(x)=a_2x^2+a_1x+a_0,\ B(x)=b_2x^2+b_1x+b_0$
- $D_0=a_0b_0,\ D_1=a_1b_1,\ D_2=a_2b_2$
- $D_{0,1}=(a_0+a_1)(b_0+b_1),\ D_{0,2}=(a_0+a_2)(b_0+b_2)$
  $D_{1,2}=(a_1+a_2)(b_1+b_2)$
- $A(x)*B(x)$
  $=D_2x^4+(D_{1,2}-D_1-D_2)x^3+(D_{0,2}-D_0-D_2+D_1)x^2+(D_{0,1}-D_0-D_1)x+D_0$

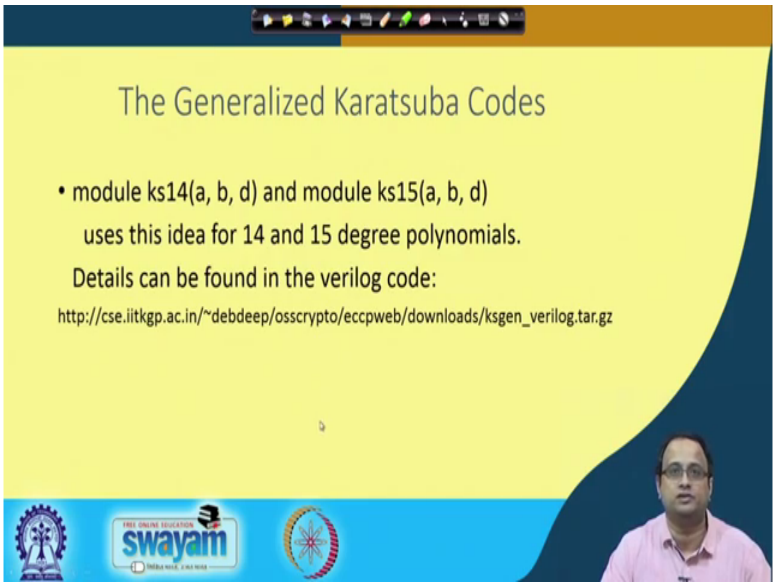
So, now we can actually go and we can describe the general Karatsuba. So, general Karatsuba, as I said is where you decompose instead of decomposing into 2 as we have seen in simple Karatsuba, you decompose it further. For example, if I take A x, I suppose and suppose that decompose A x into a 2, a 1 and a 0 that is three parts. Likewise I take B x and decompose into three parts. Then I can denote D 0 as a 0 b 0; D 1 as a 1 b 1; and D 2 as a 2 b 2. I can still write the final result A x and B x in terms of these coefficients ok.

So, you see that here I also define D 0, 1 as the Karatsuba of a 0 a 1 and b 0 b 1 just like as we did in the symbol Karatsuba. So, this is easy to check that this is your corresponding output result the interesting part is that when you have decompositions of this form, then you have got each of these terms which has got three arguments or four arguments ok.

So, now, if you observe or remember the old discussion that we had on the lookup table structure for the FPGA, it already has got fixed inputs that means, if you are you not using those four inputs for example, in a four input lookup table then you are basically underutilization the lookup table. And that is why general Karatsuba seems to be better utilizing the lookup tables because it always have got these number of input arguments for each of these terms.

So, therefore, you get a better implication in the general Karatsuba, but as I say that when you are doing or doing implementing a general Karatsuba, then it seems that the gate count increases. So, therefore, for higher dimensions it is not a good choice, but for lower dimensions it is a good choice.

(Refer Slide Time: 16:23)



The Generalized Karatsuba Codes

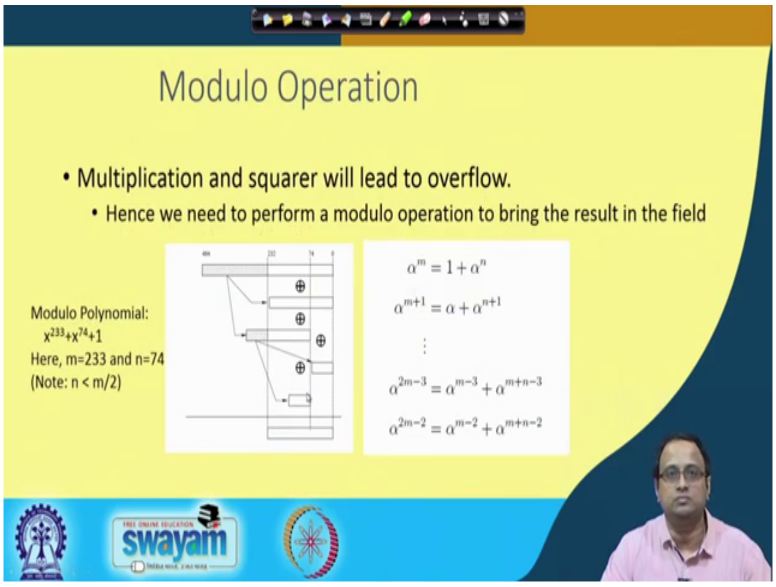
- module ks14(a, b, d) and module ks15(a, b, d) uses this idea for 14 and 15 degree polynomials.

  Details can be found in the verilog code:

  http://cse.iitkgp.ac.in/~debdeep/osscrypto/eccpweb/downloads/ksgen_verilog.tar.gz

So, therefore, what we can do is we can actually in the higher dimensions implement a normal simple Karatsuba, but when we switch back or when you go down to the lower field sizes, then we can switch to the general Karatsuba. So, therefore, what we do is you can actually do this and when you are going back going for the smaller degree like 14 and 15, you can actually switch to the general Karatsuba. So, here is a link which you can follow where you can get a complete synthesizable very low code for this design which you can where you can do further experimentation.

(Refer Slide Time: 16:41)



Modulo Operation

- Multiplication and squarer will lead to overflow.
  - Hence we need to perform a modulo operation to bring the result in the field

Modulo Polynomial:
$x^{233}+x^{74}+1$
Here, m=233 and n=74
(Note: n < m/2)

$$\alpha^m = 1+\alpha^n$$
$$\alpha^{m+1} = \alpha+\alpha^{n+1}$$
$$\vdots$$
$$\alpha^{2m-3} = \alpha^{m-3}+\alpha^{m+n-3}$$
$$\alpha^{2m-2} = \alpha^{m-2}+\alpha^{m+n-2}$$

There is another important structure which we also need to implement and that is called as a modulo arithmetic because you see that as we have said all are discussed already right that our field sizes have increased. And therefore, right we need to do a final reduction operation where we bring that the result into the find into the original field.

So, here is a very simple way of how we can do that. So, I will again take the example of my Karatsuba operation. So, I have got the final result from 0 to 464. And I want to bring the result back from 0 to 2 to 0 to 232. So, therefore, right my units I have to look into my in irreducible polynomial. So, in this case the irreducible polynomial is x to the power 233 plus x to the power of 74 plus 1. Note that 74 is in general lesser than n by 2.

So, now, what we do is that if I want to do this reduction, then a very common way of doing that is we replace x to the power of 233 by x to the power of 74 plus 1, so that means, right if I represent this x as alpha then alpha to the power of m becomes alpha to the power of n plus 1. So, therefore, right if you have got any element which is more than 232, that means, it is 233 for example, I can replace it by 1 plus alpha to the power of n ok. And likewise for all the elements which are more than that. So, therefore, right what I can do is that I will reduce it. And if I reduce it right then I will see that so these are corresponding reduced elements. So, these are all elements which are or positions which are outside my field and therefore, it goes from alpha to the power of m to alpha to the power 2 m minus 2 ok.

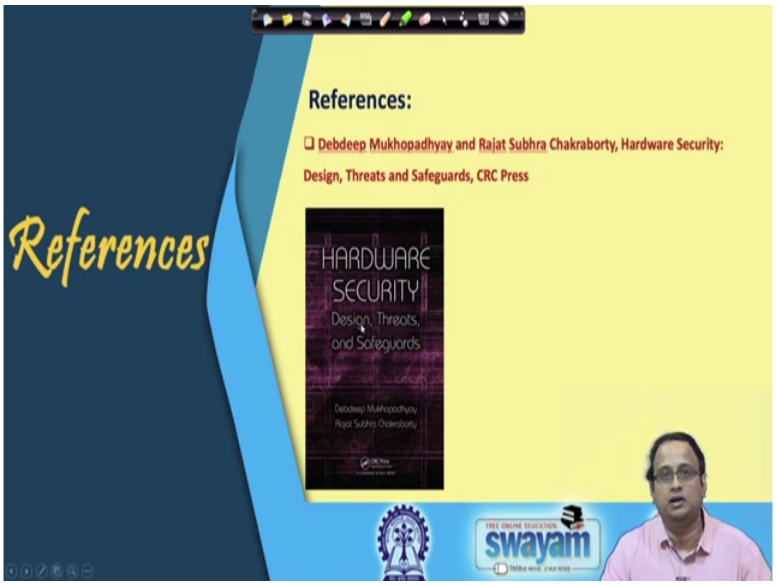
So, you can plug in for example, m equal to 233, and you note that two into 233 minus 2 is 464 that means, the maximum position here maximum value here. So, all of them are being reduced by the field, and you can see reduced by the irreducible polynomial. And you can observe that these terms like one alpha till alpha to the power of m minus 2 are now all inside the field ok. For example, m is 233; so therefore, m minus 2 will stand to 231 ok.

So, therefore, right you see that it becomes. So, you can see that is there is more gap here in this diagram to indicate that it is 231. So, therefore, right this is the corresponding part which is being shown by this white rectangle ok. So, I will reduced it. And therefore, this is my corresponding reduced result.

On the other end there is another component which is alpha to the power of n till alpha to the power of m plus n minus 2 which if you consider right, here it starts from alpha to the power of n, so n is 74. So, it starts here at 74 and continuous beyond. So, till a point and right that is it reaches to 232, it is fine, because it is inside the field. But beyond that again, it exceeds the field. So, you have to again reduce it. So, how do you reduce it again you take this elements and do a similar operation and then you reduce it. So, again you basically bring it here, and then you this is the part in this case right with just one reduction, it gets into the field and therefore, the final result is in the field ok.

So, you can observe that when you are doing this reduction right, and the final when finally, when you are giving the result, then the computations which you are doing are just employing by a Boolean XORs ok. So, therefore, the entire reduction operation can be done without any complicated division or finding out the demanded ok. You can just do it simply by using Boolean XORs. And therefore, the reduction operation or the modulo operation in GF 2 power of n arithmetic is very efficient ok. You can do it very in a very simple in a very simple manner.

(Refer Slide Time: 20:01)



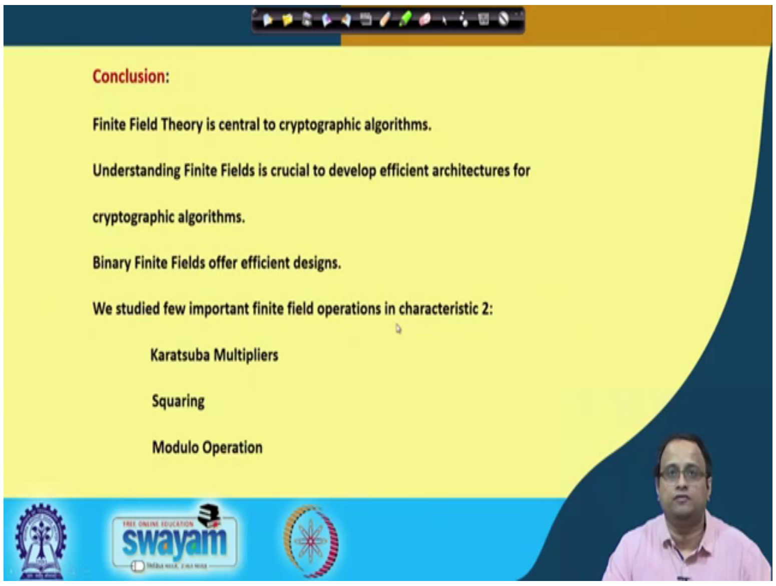
References

References:

❑ Debdeep Mukhopadhyay and Rajat Subhra Chakraborty, Hardware Security: Design, Threats and Safeguards, CRC Press

So, likewise right I mean you can find out more details in this book ok. So, this is like I mean this is just an overview about how you can implement the two fundamental structures like the Karatsuba multiplier both the simple and the general Karatsuba and also right how you can do the modular arithmetic ok. So, with this right you should be able to follow more discussions which are given in the book which essentially I am omitting for the sake of time.

(Refer Slide Time: 20:23)



Conclusion:

Finite Field Theory is central to cryptographic algorithms.

Understanding Finite Fields is crucial to develop efficient architectures for cryptographic algorithms.

Binary Finite Fields offer efficient designs.

We studied few important finite field operations in characteristic 2:

Karatsuba Multipliers

Squaring

Modulo Operation

And finally, right just to conclude what we discussed in both the previous lecture and this part is that finite field theory is very central to cryptographic algorithms. Understanding finite fields its crucial to develop efficient architectures for cryptographic algorithms. Binary fields offer special advantages in terms of efficient designs it is more compact, more optimal in terms of its representation. We studied few important finite field operations like in characteristic 2 in specifically I mean like the Karatsuba multiplication, the squaring operations and also the modulo arithmetic ok.

So, thank you. So, we will continue again in the next class with more discussions on finite field architectures and other subsequent topics.

Thank you.