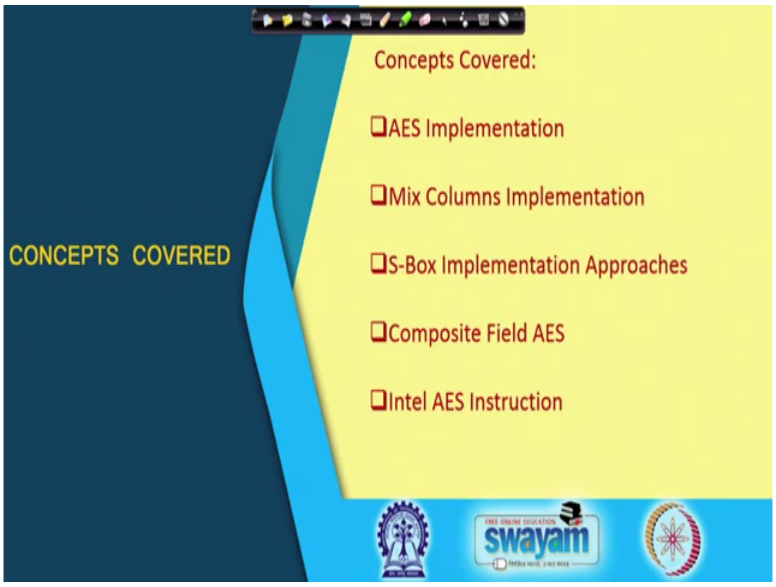
Hardware Security
Dr. Debdeep Mukhopadhyay
Department of Computer Science and Engineering
Indian Institutes of Technology, Kharagpur

Lecture – 12
Hardware Implementation of Advanced Encryption

So, welcome to this lecture on Hardware Security. So, today we shall be trying to understand about some new design techniques in perspective of the Advanced Encryption in perspective with the advanced encryption standard algorithm. So, to start with let me take go through the; like the topics that we shall be covering in today's class.

(Refer Slide Time: 00:33)



CONCEPTS COVERED

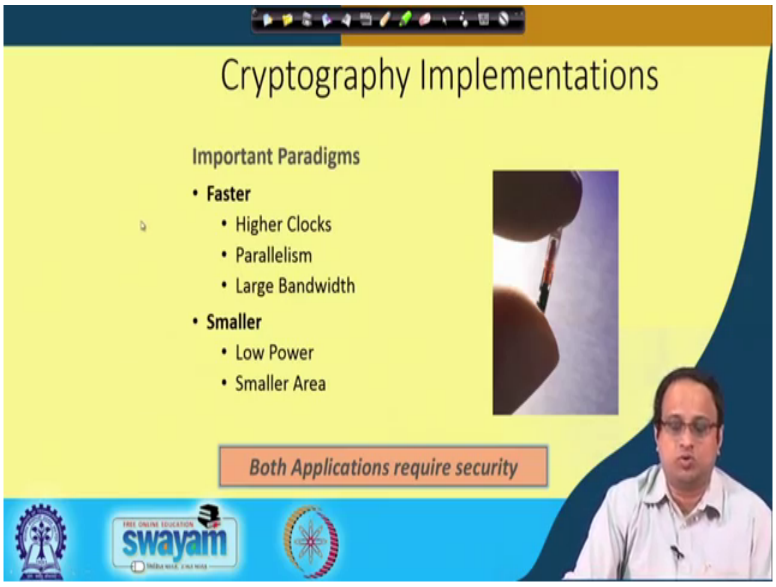
Concepts Covered:

- ❑AES Implementation
- ❑Mix Columns Implementation
- ❑S-Box Implementation Approaches
- ❑Composite Field AES
- ❑Intel AES Instruction

So, we shall be talking about AES implementation, we shall be talking with respect to the components of AES algorithm which is essentially the mix column operation and the sub box or the substitution box or the s-box implementation of AES. We shall be talking about you know like we shall be trying to apply the techniques of isomorphism that which we which we learnt in the last class. And try to see that how we can apply it for realizing a composite field AES.

So, and finally, we shall be concluding with some quick look into one of the most efficient AES implementations which is present in the modern the Intel machines in the form of his delegated instruction for accelerating AES operations. So, to start with we all know that cryptographic Implementations are important.

(Refer Slide Time: 01:26)



Cryptography Implementations

Important Paradigms

- Faster
  - Higher Clocks
  - Parallelism
  - Large Bandwidth
- Smaller
  - Low Power
  - Smaller Area

Both Applications require security

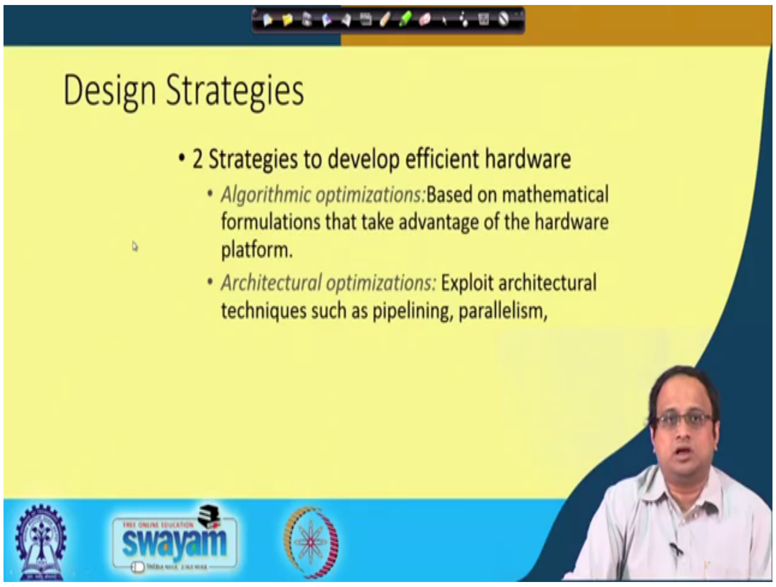
And we have got different paradigms for implementation for example, we need fast designs we need fast designs in the sense; we need like our designs to operate at higher clocks. We want to extract parallelism that is one of the main reasons why we are looking into hardware design. And we also want that our design should support larger bandwidth.

So, while this can be you know like one category of objectives you can also have another or you can have other categories on the other hand end of the spectrum; where you need smaller designs you need your designs to have low power to should consume you know like low power. And also you know like take less area on silicon. So, therefore, right I mean this is also very important in certain context for example, when you want to your designs to operate on lightweight platforms. For example, maybe your mobile phones or some resource constrained IOT node. So, therefore, right we would probably need these kind of design paradigms or various kind of design paradigms and we need various kinds of design techniques for that.

So, in one end one end we may require fast designs on the other end the requirement may be of making designs which are which are which are which are in which are lightweight and which essentially consume less energy and take place area for example. So, therefore, depending upon the perspectives your design choices may be different and we will try to see some of them some of these flavors when we take talk about the AES algorithm in particular.

So, so therefore both but one thing is for sure that we cannot compromise security and as we will be seeing this in more and more details in our class that you know like just implementing with respect to you know like these design choices may compromise security. So, therefore, at the end of the day we have to also ensure that our designs are secure and this we talking more and more details in during our course.

(Refer Slide Time: 03:26)



Design Strategies

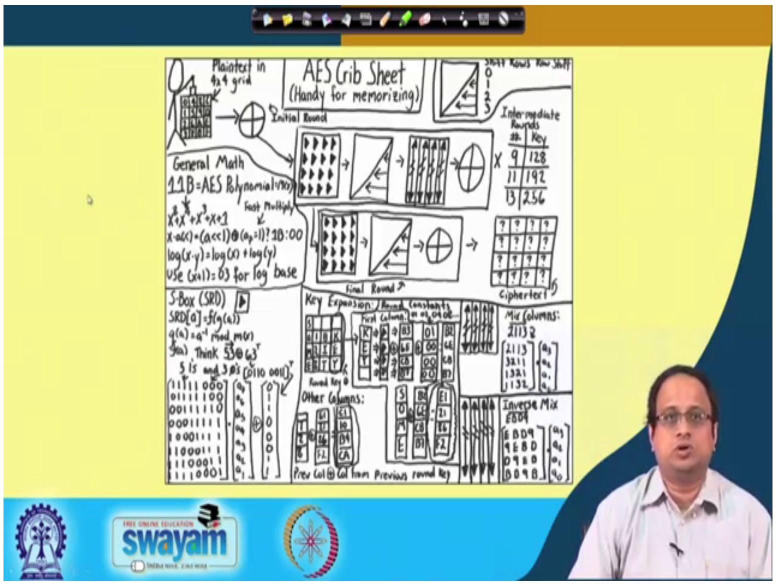
- 2 Strategies to develop efficient hardware
  - *Algorithmic optimizations:* Based on mathematical formulations that take advantage of the hardware platform.
  - *Architectural optimizations:* Exploit architectural techniques such as pipelining, parallelism,

So, to start with you know like we would like to discuss about some broad design strategies that we hardware designers kind of you know like they adopt when we want our designs for example, you know like we may want to do our optimizations which actually depend upon the mathematical formulations on which our ciphers are described. So, we would like to do something which is called as algorithmic optimizations which are based on mathematical formulations that take advantage of the hardware platform.

So, in order to you know like sort of to explain this part of the optimization or these ways of optimization; I will be taking the example of isomorphism that we which we discussed in the last classes. And we will try to see how we can adopt isomorphism to extract more compact designs of the AES algorithm ok. Like we will be trying to see how we can adopt it not only for components of the AES but in general over the I mean the overall AES hardware. How we can make it less consume less energy I mean how we can make it consume less resource ok.

So, on the other end right. We also adopt architectural optimizations where we try to exploit architectural techniques like pipelining and parallelisms. So, these are of course, like techniques which we have learned in the context of computer architectures. And we also try to adopt them to get an end to end high throughput design so.

(Refer Slide Time: 04:53)



So therefore, this is a. So, again you know like taken from the same source where I which I sited in the in some of my previous old classes; where we essentially this is this is just a summary of the various AES structures. So, therefore, if you remember like that in AES right you can actually represent the AES algorithm by a 4 cross 4 block. And each of these elements in this square matrix essentially are bytes. That means, that elements of GF 2 power of 8 that is Galois field 2 power of 8.

And then what you do is that you basically pass them through the several rounds for example, in AES 128 you pass them through the you know like 10 rounds. And each of the rounds are as follows like if you just recapitulate and remember I just remember what we already have discussed. So, initially you do an x oring with a key which is essentially the input key. And then you pass them through a step of through a certain sequence of steps which are called the rounds of the AES algorithm.

The first step is an s-box, so in the s-box you basically do a lookup; that means, every byte is essentially substituted by another byte and that is essentially denoted by this box where you are basically having say you know like 16 s-boxes to get the substitution done. And final and after that you do a shift rows in the shift row the first row you do not do any shift, the second row you do a cyclic left shift by one position.

The third row you do it cyclic left shift to the left by 2 positions or 2 byte positions. And in the final row you do a cyclic left shift by 3 byte positions. And once you have done that after that we have this mix column steps which recently it takes each column and performs a pre matrix multiplication with a fixed matrix as we have discussed in the previous classes and then you get another column. So, we basically do a column by column transformation and finally, the objective is that a combination of shift row and the mix column should essentially give me very fast diffusion in the cipher.

Then I do an x oring with the round key and finally, I get the output of the first round. And then I pass this through you know I kind of iterate this process through say 10 through 9 rounds in the case of AES 128 and then pass it through a final round. Now, note in the final round you have an s-box or substitution box layer and you also have the shift rows, but you do not have the mix columns. So, the mix columns is not present in the final round.

Then you do an x oring with the final round key and you get the cipher. So, while you know like AES 128 is just one parameter or one configuration of the AES algorithm you may have other configurations. For example, you know like you may have AES 128 AES 192 or AES 256 depending upon which you change your number of rounds as well ok. So, for example, in this case the number of rounds that you are doing is 9 plus 1, here you will be doing 11 plus 1 and here you will be doing 13 plus 1. So, this is the number of intermediate rounds which you have in your cipher.

So, finally, you get the cipher text and that essentially is what the you know like what an attacker should ideally see. That means, the attacker should be able to see the cipher text and maybe the plaintext, but from there we would try to know what is the key. So, that is the objective of what is called as cryptanalysis. So, how the s-box is defined or how the operations take place often in AES right we do the mathematics in GF 2 to the power of 8 as we have discussed. So, therefore, in GF 2 to the power of 8 or any extension field. We know that we need an irreducible polynomial.

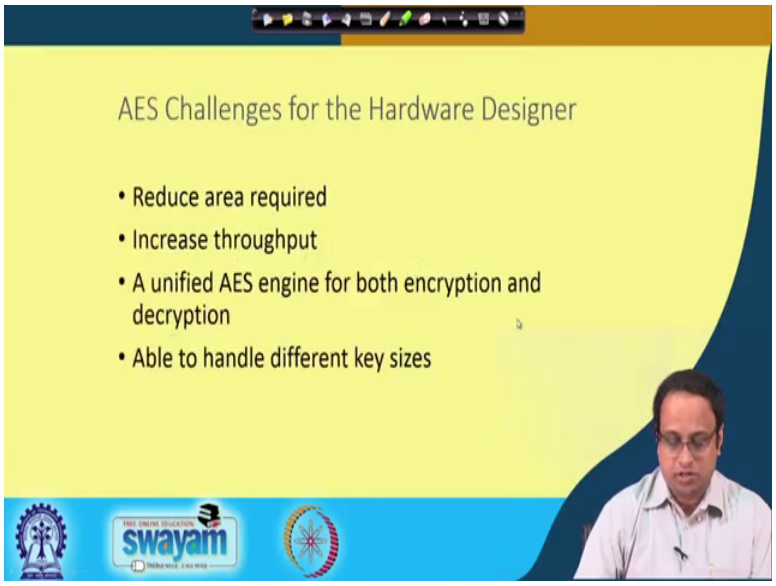
So, the irreducible polynomial which is used for AES is as follows x to the power of 8 plus x to the power of 4 plus x to the power of 3 plus x plus 1. And then you do your operations by doing module operations with this as we have discussed that in AES since it is based on composite fields there are many opportunities for developing efficient implementations. In particular just to remember the s-box what we do in an s-box computation is that it is a byte by byte substitution or it is a substitution operation. So, therefore, you take the byte which is denoted here as a and then you pass it through a series of transformations.

So, basically there are two transformations the first is the finite field inverse. So, you calculate a finite field inverse by performing a to the power of minus one modulo m x where m x stands for this irreducible polynomial. Finally, you give you get this result which is denoted as g a and then I do an affine transformation on g a. So; that means, like I take this matrix I mean I take this output of the of g a which is denoted as from as a vector from a 7 to a 1 or a polynomial you can also think of that to be a polynomial which would I actually should range from a 7 to a 0 actually because this is an eight bit element.

So, it is a 0 to a 7 which is eight bits and therefore, there is an element in GF 2 to the power of 8 we then do an affine transformation which means we basically multiply it by a fixed matrix and then you add an vector. And finally, get the result of my of the s box. So, therefore, right and so this is this is like the broad constitution of the AES operation and but, there is also another very important layer which we have not really discussed in details which is the key expansion. I will again not go into all the details on the key expansion, but rather just mention that in the key expansion you take an input key which is again 128 bit and from there you try to develop the remaining round keys.

So, for example, you develop 10 round keys in when you are talking about AES 128 likewise for AES 192 and AES 256 you would require to generate 128 bit keys or I mean every round will typically have still have 128 bit keys, but in AES 192 or AES 256 your input key is of dimension 192 bits or 256 bits respectively. So, likewise the key scheduling is also you know takes account of these parameters.

(Refer Slide Time: 10:48)


AES Challenges for the Hardware Designer

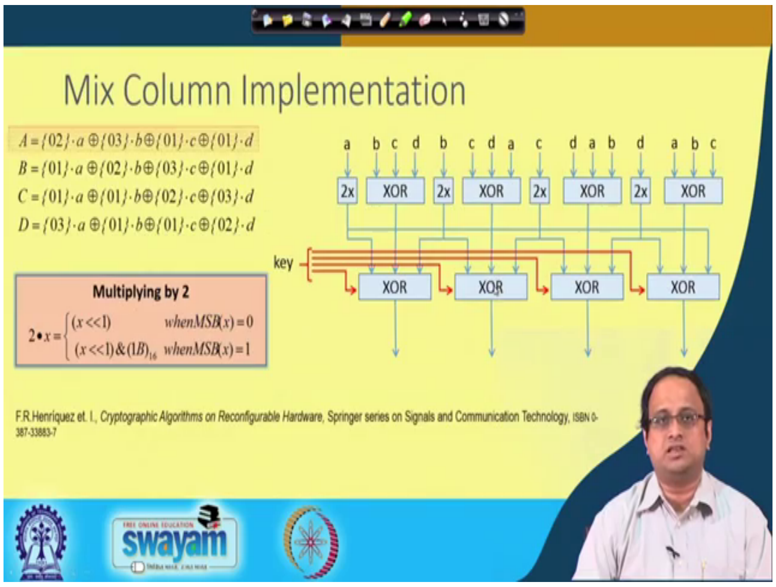
- Reduce area required
- Increase throughput
- A unified AES engine for both encryption and decryption
- Able to handle different key sizes

So, so now, the question is right we would like to develop a hardware design for AES. So, what are the motivations of our hardware design. So, for example, right one of the objectives could be that I would like my design to take less area. So, it would be you know like consume less you got it should be able we should be able to implement it with less than amount of gates. And therefore, what this is what is this particular you know like I mean how do we measure that whether our design is really consuming less area.

We often measured it by the by what is called as gate equivalent where we try to find out the number of two input NAND gates through which we can realize our design. So, that is often called as the gate equivalent or abbreviated as ge likewise we would also like to measure the throughput of our design. And therefore, we would like to make our design has having high throughput. So, I would like to increase my throughput of my design and for AES right one of the objectives could be that I want a unified AES engine. That means one single hardware which will work both as an encryption as well as well as a decryption.

Now you should note here that the AES algorithm right is not symmetric in terms of the encryption and decryption. That means, the encryption process and the decryption process are distinct. So, how do you really make a single hardware which essentially functions both as encrypter or as decrypter is a challenging proposition likewise of course. We should be able to handle different key sizes and this essentially is also an important could be an important design criteria where a single device should be we should be able to support 128 bit keys 192 bits or 256 bit keys. So, therefore, right I mean this could be there could be various such challenges for AES design.

(Refer Slide Time: 12:24)



Mix Column Implementation

$A = \{02\} \cdot a \oplus \{03\} \cdot b \oplus \{01\} \cdot c \oplus \{01\} \cdot d$

$B = \{01\} \cdot a \oplus \{02\} \cdot b \oplus \{03\} \cdot c \oplus \{01\} \cdot d$

$C = \{01\} \cdot a \oplus \{01\} \cdot b \oplus \{02\} \cdot c \oplus \{03\} \cdot d$

$D = \{03\} \cdot a \oplus \{01\} \cdot b \oplus \{01\} \cdot c \oplus \{02\} \cdot d$

Multiplying by 2

$$2 \bullet x = \begin{cases} (x << 1) & when MSB(x) = 0 \\ (x << 1) \& (1B)_{16} & when MSB(x) = 1 \end{cases}$$

F.R.Henríquez et. l., *Cryptographic Algorithms on Reconfigurable Hardware*, Springer series on Signals and Communication Technology, ISBN 0-387-33883-7

So, now, I will try to look into you know like some of them you know like during our discussions. So, I will start with the mix columns. So, if you remember in the mix columns you essentially do a pre matrix multiplication; that means, you take basically an element in GF 2 to the power of 8 and you pass it through a matrix; so this is how you do a mix column. So, therefore, here is a vector which is essentially nothing, but a column of the of my state matrix this is this element has got 32 bits which means that each element is a 2 to the power of 8 or GF 2 to the power of 8 element.

So, what we do is that for the mix column we pre multiply it with this matrix. So, which is showed here as 2 1 1 1 3, 3 2 1 1, 1 3 2 1, 1 1 3 2. So, this matrix essentially the 4 by 4 matrix and each element is also an element into 2 to GF 2 to the power of 8. So, when you are doing this multiplication of this matrix with this vector; remember that when you are multiplying set 2 with a 3 ok. So, when I am multiplying 2 with a 3 or 1 way or 3 with a 0 these multiplications are not integer multiplications they are multiplications in GF 2 to the power of 8. So, therefore, right I would represent these two as a polynomial. So, this polynomial in GF 2 to the power of 8 would be x.

So, therefore, when I multiply x with say a 3 a 2 a 1 and a 0 then remember that I mean when I am multiplying these two with a 3 then 2 is represented as x and likewise a 3 will also have an 8 bit representation or a polynomial representation. So, when I multiply this with this and whenever there is an overflow. That means, the degree of the polynomial exceeds 8 or exceed 7 rather, then I have to do a modulo operation. So, therefore, this modulo is again done by this mx polynomial which is the irreducible polynomial which I use in context to AES ah algorithm. So, so let us try to you know like look into the mix columns with this background.

So, as I said that when you are calculating the mix columns then your input here is denoted as a, b, c, d. So, these are the four bytes of my column. So, I have got one column and I am processing it through this mix column. And as I said that it is a 32 bit element the column is a 32 bit element where each element is a byte. So, therefore, a is a byte, b is a byte, c is a byte and d is a byte. So, when I do this multiplication then of course, like the first element is 2 into a plus 3 into b plus c plus d. And that essentially is my output element which is capital A. Likewise you have got capital B capital C and capital D, but they are multiplied with different elements.

Because each row as we can see right in mix columns is a cyclic shift of the previous row. So, therefore, is the same elements, but it is the arrangement is slightly different so finally, you get A, B, C and D. And again just what I said you know like just to recapitulate this point that when you are multiplying with 2 remember that 2 into a is not an integer multiplication. But rather is a multiplication in finite field which means that I will do typically what I do in normal arithmetic that is I will if I we want to multiply say this remember that a you can also represent as a polynomial whose degree is seven because it is an element on GF 2 to the power as 8.

So, when you multiply with two which essentially stands for x then you are basically doing a cyclic glyph shape ok. So, therefore, x into the polynomial a; that means, when I am calculating a into x then I am doing a cyclic left shift of the polynomial a or the or you can think of like the bit the bit representation of a. So, that is denoted here the you know like that when you are wanting to calculate 2 into x you are basically doing a cyclic left shift of x. So, that is denoted here, but remember that is fine when the MSB of x is 0 ok.

But when the MSB of x is 1 then basically there is an overflow. And therefore, once there is an overflow you need to do a modulo reduction with your irreducible polynomial. And that is denoted here by this step where you are basically doing an you know like doing a modulo reduction with your irreducible polynomial for AES. So, therefore, right I mean that gives you the how to compute 2 x and you can you know how to calculate 2 x remember that when you want to calculate 3 x or 3 b for example. And if you have got if you have got 2 a and if you have got a.

Then you just need to add 2 a with a to get 3 a ok; that means, you do not need to do an explicit multiplication with 3 a and that is why the mixed columns can be very efficiently implemented and that is shown here in this diagram. What you do here is that you denote. Suppose my input is a so I do it you know like multiplication with 2 as I have just now described. So, you calculate 2 into a and likewise you know like you can calculate 2 into b. But when you want for example, in this case if you want 2 a plus 3 b plus c plus d. You can note that when I am doing the final XOR then I am of course, passing in 2 a because 2 a is one of the components but I also need 3 b.

So, I need 2 XOR b 3 b, but how I am doing 3 b is interesting. So, I am doing 2 b, but I am not calculating 3 b explicitly. So, I am just calculating in 2 b and then I am x oring b with it. So, therefore, to be x or with b would stand for 3 b. And therefore, I get 3 b here without doing an explicit multiplication with 3. So, therefore, I have got 3 b and 4 c and d I just need to pass them and therefore, c and d are XOR and passed finally, this XOR output will give me 2 a plus 3 b plus c plus d or the where the plus essentially is nothing, but exclusive OR.

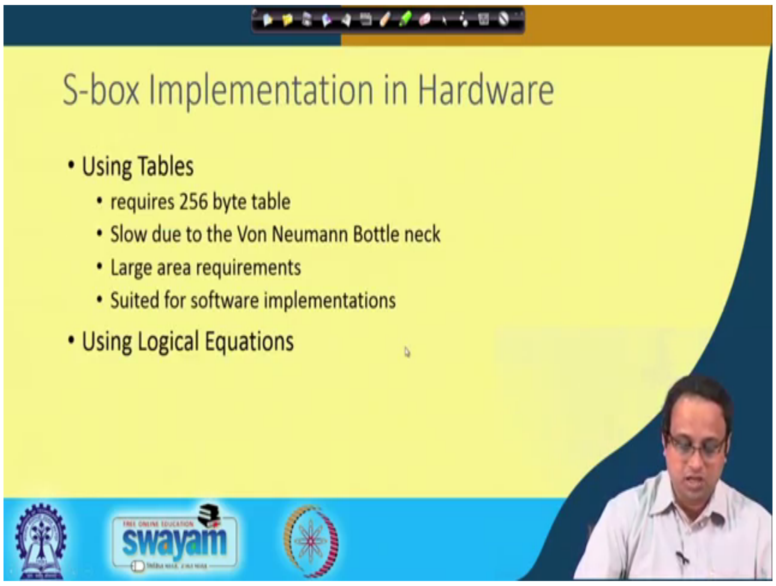
So, likewise you can actually calculate the other elements in the column. For example, you can calculate a plus 2 b plus 3 c plus d. So, that is here that is denoted here when you are calculating these components. And likewise for the remaining elements in the you know you can calculate capital A capital B capital C and capital D in this fashion. So, note that when you are you can leave it at this point, but you if you remember like in your round you also need to add the key ok.

So, therefore, that is denoted here by this layer. So, where key is again another byte. So, you can imagine that there are four bytes in the which you are taking from the key matrix which is the round key matrix and then you can actually x or them after the output of this layer. So, therefore, after the mix column you can do a separate dedicatedly you know different add round key, but there, but here you can do a slight trick when you are you know like in particular going for FPGA implementations and that is illustrated here.

So, remember that I told you that the lookup tables that you have in FPGAs often I have got fixed input structure. That means, they probably have got like four inputs or six inputs or so on, which means that if you are you know like not utilizing them properly then there are lot of wastages of the look up tables. So, what you can observe here is that if you combine the key addition layer with the mix column. Then if you observe this XOR for example, then this XOR has got 1 2 3 and 4 inputs.

That means, if you do not do the use this key and if you combine it later then you would probably require two equivalent look up tables. But if you combine them and do it at one shot then you require one single lookup table. So, therefore, you are basically doing the same operation, but you are saving on the resource that you need to make your design ok. And this is a small trick, but it can actually work nice because remember that this process is being repeated for all the 4 columns. And therefore, right the total amount of saving would be you know always magnified ok. And therefore, it is a useful trick that you can try to adopt in your design.

(Refer Slide Time: 20:04)



S-box Implementation in Hardware

- Using Tables
  - requires 256 byte table
  - Slow due to the Von Neumann Bottle neck
  - Large area requirements
  - Suited for software implementations
- Using Logical Equations

So, the other important layer which is there in AES implementations is the s-box. So, as I said that s-box is nothing, but a lookup table where you are basically substituting a byte with another byte. So, one way of implementing them would be by using tables. So, you can probably require a 256 byte table. And you can actually do your implementations by just writing you know like just developing a memory based implementation. But this design would be slow due to what is called as the Von Neumann Bottle Neck.

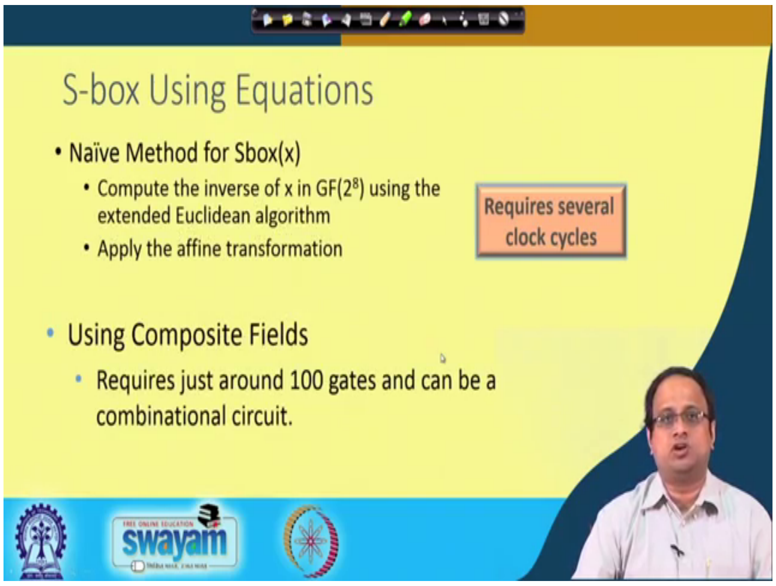
So, if you remember I told this in the first class that you know like there is a big gap between the processing speed that your CPU enjoys and the memory access time. Because, the memory access time is inherently much slower compared to your CPU processing your timing delay and other things. And that is why, I write what we have is what is called as a memory wall. And therefore, right if you do your memory computations then our if you do the s-box in memory then you are always kind of constrained by the memory access delay.

And therefore, your design becomes slow. More importantly like you are using more resource because you are using the memory components. And therefore, right while using s-boxes or using table base s-boxes is probably more suited for software implementations, for hardware implementations we will probably try to develop logical equations. But at the same time you also should kind of think that if you want to implement the AES s-box just by purely logic equations. Then that also would consume large amount of resource because simply because each of these s-box elements like if I take all the 8 bits of the are the all the 8 output bits of s-box.

Then this each of these output bits actually depends upon all the input bits of the of the s-box; which means the one of the reasons why it is done is because you want to make your design nice I mean you want to make your s-box strong against standard cryptanalytic attacks. And therefore, there are many designer criteria's which has led to the design of AES s-box which is very nice which essentially requires the AES s-box structure to be very complex.

And therefore, right if you would like to expand the output of the AES s-box in terms of its input then you would require a huge number of gates. You would require a large number of components to do these design. For example, if you want to write the equations maybe on a on a piece of paper then you would require a large amount of space to do that. And that means that your design is pretty complex. So, therefore, right I mean what people try to develop is try to develop tricks or techniques for doing so.

(Refer Slide Time: 22:32)



S-box Using Equations

- Naïve Method for Sbox(x)
  - Compute the inverse of x in GF(2^8) using the extended Euclidean algorithm
  - Apply the affine transformation

Requires several clock cycles

- Using Composite Fields
  - Requires just around 100 gates and can be a combinational circuit.

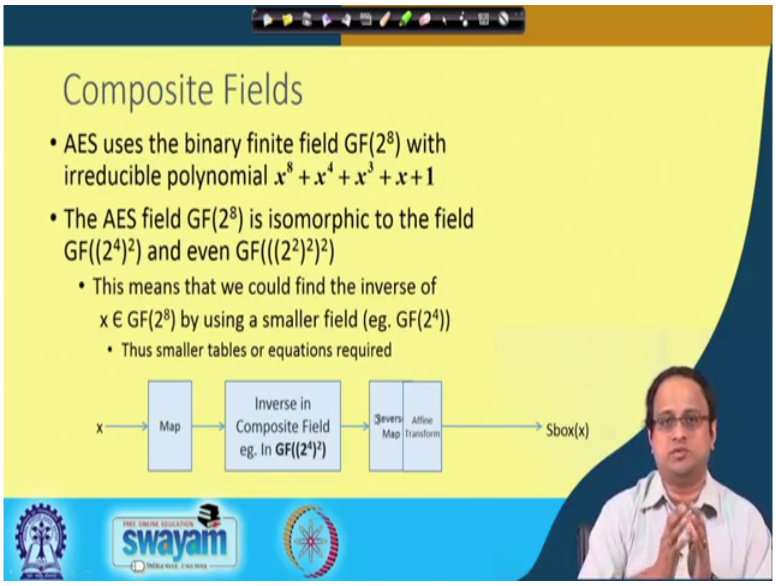
So, for example, like if I go for a naive way or naive method of implementing an s-box. I can actually try to adopt what is what we have already studied is that you know like I mean we can also go for these approaches where you try to apply maybe the Euclidean algorithm or the extended Euclidean algorithm for computing the AES output or the s-box output. But then you know like you would require a large amount of clock cycles for doing so ok.

So, therefore, I can of course, compute the inverse of x in GF 2 to the power of 8 using the extended Euclidean algorithm. And then apply the affine transformation, but if I do this in this process then several clock cycles will be required to get the output of a single s-box ok. And imagine that you have to do this for maybe 16 s-boxes. Of course, you can do them in parallel, but then you would require a large amount of resource. But there is a very interesting trick which where we try to apply the principles of isomorphism and is an example of algorithmic optimization ok.

As we have defined where we try to apply the techniques of isomorphism and use a principle or what are called as composite fields. So, as we have discussed like composite fields are fields of the form of GF 2 to the power of n whole power of m. So, if I try to you know like apply composite fields and apply it you know like the principle that all the composite fields are isomorphic.

If they are properly designed, then I can actually make very compact designs of the AES s-box which will probably require less than 100 gates or maybe around 100 gates. And we can essentially do the entire thing using a combinational circuit. That means, I can do it without expanding clock cycles to get the output of the s-box ok.

(Refer Slide Time: 24:11)



Composite Fields

- AES uses the binary finite field GF(2^8) with irreducible polynomial $x^8+x^4+x^3+x+1$
- The AES field GF(2^8) is isomorphic to the field GF($(2^4)^2$) and even GF($((2^2)^2)^2$)
  - This means that we could find the inverse of $x \in$ GF(2^8) by using a smaller field (eg. GF(2^4))
    - Thus smaller tables or equations required

So, therefore, let me take an example or let us try the example or you know like motivate why we need to go into this ok. Or try to understand the savings that we can essentially get out of this technique. So, this must remember that we in AES we use the binary finite field which is GF to the power of 8 and my modulo reduction polynomial or the irreducible polynomial is as x to the power of 8 plus x to the power of 4 plus x to the power of 3 plus x plus 1.

So, now, we will use this principle that is GF 2 to the power of 8 can also alternatively we represented by this composite field structure which is GF 2 to the power of 4 whole power of 2 or even you can break down GF 2 to the power of 4 further. And then you can also have fields like GF 2 square power of 2 power of 2 so even that is isomorphic ok.

So that means, now I can do my designs in a in a much smaller field. For example, if I want to calculate an inverse or do an inverse in GF 2 to the power of 8 then, I can transform my element into GF 2 to the power of 4 square which is this. And then do my inverse computation in GF 2 to the power of 4. So, what I can do is that I can you know like one way of doing this implementation is that I can just have a table based inverse computation.

For GF 2 to the power of 4 and that would be much more compact doing than doing an inverse in GF 2 to the power of 8 ok. And therefore, right I can have an overall compact design to get the output of GF 2 to the power of 8 inverse ok. So, therefore, I will implement this GF 2 to the power of 4 in a table based manner. And then use logical equations to derive the output in GF 2 to the power of 8 or GF to the power of 4 square.

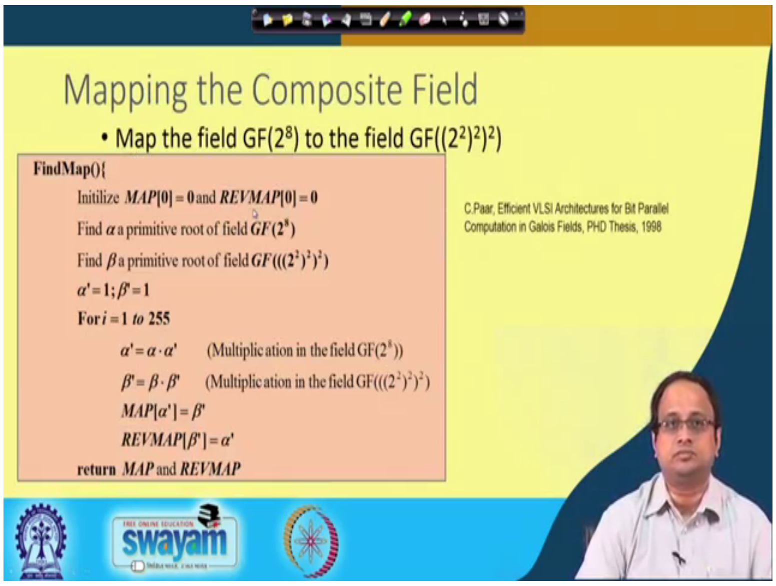
So, this transformation from GF 2 to the power of 8 to GF 2 to the power of 4 square and vice versa for this I need a conversion matrix. And as we have discussed in the previous class, there are different class of algorithms to which through which we can get these transformations being computed so. So, therefore, right if you are able to do that then what you will do is the as follows.

You will take x here and then you will map this x to an element in the composite field; That means, remember that x was in GF 2 power of 8. So, now, you will map this 2 GF to power of 8 into GF 2 to the power of 4 square ok. And then do the inverse computation in GF 2 to the power of 4 square; that means, this inversion will be calculated in GF 2 to the power of 4 square. And then you will do a reverse map to bring the result back to GF 2 to the power of 8. And then you can do an affine transformation. So, remember that affine transformation has to be done.

Because you final in one that your algorithm should be producing the correct result as per the specification of AES. So, then you can also do a very simple trick where this reverse map remember can also like remember that the map and the reverse map like based on the final algorithm. That I discussed in the last class can be represented by a linear matrix which is a 0 1 linear matrix right.

So, therefore, what I can try to do is I can try to you know like combine the inverse map matrix with the affine transformation matrix remember that these are two matrix operations. And therefore, I can combine them and I can get a one single matrix to do this transformation ok. And that probably would make my design more compact then you know like having two independent maps ok. Like the reverse map followed by the affine map we can try to combine them into one single map and that can save some gates.

(Refer Slide Time: 27:40)



Mapping the Composite Field

- Map the field GF(2^8) to the field GF($((2^2)^2)^2$)

```
FindMap(){
    Initilize $MAP[0] = 0$ and $REVMAP[0] = 0$
    Find $\alpha$ a primitive root of field $GF(2^8)$
    Find $\beta$ a primitive root of field $GF(((2^2)^2)^2)$
    $\alpha' = 1; \beta' = 1$
    For $i = 1$ to 255
        $\alpha' = \alpha \cdot \alpha'$    (Multiplication in the field GF($2^8$))
        $\beta' = \beta \cdot \beta'$    (Multiplication in the field GF($((2^2)^2)^2$))
        $MAP[\alpha'] = \beta'$
        $REVMAP[\beta'] = \alpha'$
    return $MAP$ and $REVMAP$
```

C.Paar, Efficient VLSI Architectures for Bit Parallel Computation in Galois Fields, PHD Thesis, 1998

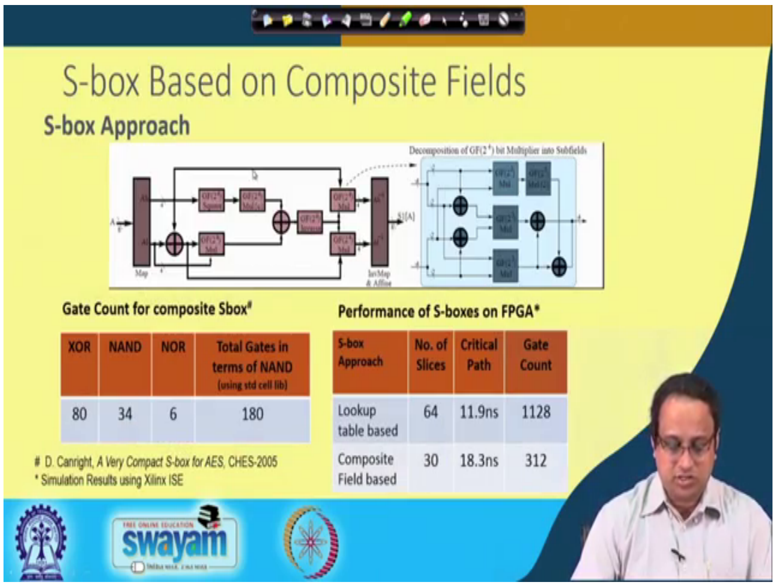
So, how we transform this element from to this field GF 2 to the power of 8 to this field GF two to the power of 2 power of 2, I will not go in details here because that is what we essentially covered in details in the last class, but maybe you can go through this thesis and it is a very nice you know like reference for these kind of discussions.

I just remind you that one way of doing that is we just map the 0 of this element to the 0 of the you know like the I mean the 0 of GF 2 to the power of 8 to the GF to the 0 of the composite field. And then find alpha which is a primitive root of the field GF 2 to the power of 8 you find beta which is a primitive root of the GF 2 power of 2 power of 2 and then you develop this map. So, therefore, you just try to map alpha to beta you try to map alpha square to beta square trying to map alpha cube to beta cube and so on.

So, therefore, you can essentially have a nice map between all these 256 elements which are there in GF 2 to the power of 8. As well as in GF 2 to the power of 2 power of 2 because they are equinumerous both of them have got the same number of elements only the way you are doing the computation is different. So, therefore, when you are doing isomorphism what you can do is you can actually calculate you know like you can perform a multiplication in the transform field.

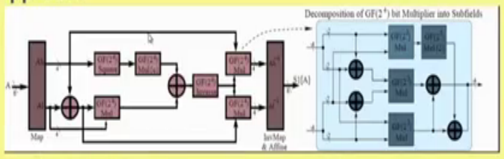
And then you can apply the reverse transformation to get the result in the original field that is what we have discussed in the previous class. But what may what we are trying to take advantage of is that the you know the computations that means, the multiplications and the inverse computations in the composite field should be more efficient than in the original field which is GF 2 power of 8.

(Refer Slide Time: 29:15)



S-box Based on Composite Fields

S-box Approach



Gate Count for composite Sbox#

XOR	NAND	NOR	Total Gates in terms of NAND (using std cell lib)
80	34	6	180

D. Canright, *A Very Compact S-box for AES*, CHES-2005
* Simulation Results using Xilinx ISE

Performance of S-boxes on FPGA*

S-box Approach	No. of Slices	Critical Path	Gate Count
Lookup table based	64	11.9ns	1128
Composite Field based	30	18.3ns	312

So, let us see that what are the you know like benefits of doing this and here is a simple you know like illustration of you know like how or what are the benefits of that. For example, I am not going so let us you know like not try to be bothered about this architecture right. Now, but just try to see the benefits for example, when you are for example, measuring the gate counts for composite s-box implementations.

So, so here are two some figures which have been provided to us ok. So, when you are calculating the s-box or composite s-box implementation then you can see the number of number of gates which are required to do this design ok. For example, the number of XORs is something like 80, the number of NAND gates is 34, number of NOR gates is 666 total number of gates in terms of NAND.

So, this is the you know what is called as the gate equivalent when you are representing the entire gate requirement in terms of 2 input NAND gates. And for standard cell libraries it turns out to be 180 which is like around 100 to 200 gates you can do it to do this design. So, so likewise right you can actually also try to look into this that suppose I want you know like to do these design rather than standard cells, but I would like to do this design on FPGAs.

So, here you can see that for a composite field based AES s-box implementation the number of slice requirements is 300 and the critical path of my design is 18.3 nanosecond. and the gate count is something like 312 ok. So, if you compare this with a pure lookup table based implementation you can see there the number of slices which is required is significantly reduced it was previously 64 in a lookup table based implementation.

But now it has been reduced to 30 which is almost like half. Now if we compare with the of course, there is there is a penalty which you pay for that. And you know like when you are doing your computations in logical circuits then of course, you can expect that the critical path of your design will increase. And that is shown here by these figures where you can see that the critical path increases from 11.9 nanoseconds to 18.3 nanoseconds ok.

So, but at the same time right also keep in mind that you know like you also have you know like when you are implementing in lookup table. Then and if you are for example, going for a dedicated memory based implementation then there is a memory access time which would also try to you know like I get accounted into this.

So, so therefore, right I mean of course, I mean you can say that you know like there is an increase in the critical path delay. But you know like I believe that we should be able to appreciate the fact that there is a significant amount of reduction in terms of the number of gates which are required to do this design ok. For example, the gate count here has diminished from say 1128 to only 312 ok.

So, that is a phenomenal amount of decrease in terms of gate counts. And therefore, right if that happens right then you can actually go for you know like parallel implementations of the AES s-box to alleviate the throughput issues and you can essentially get a very nice tradeoff between you know like throughput as well as area when you go for composite field implementations ok. And that shows you know like the minute of composite field implementations.

So, therefore, I will stop here and rather we will discuss in our next class on you know like how this architecture comes. And how we can actually get use composite fields to get an overall architecture for our for the s-box. So, we will start from there and again go into the mix columns and then try to see the overall picture so.

Thank you for your attention.