Hardware Security
Prof. Debdeep Mukhopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 23
Hardware for Elliptic Curve Cryptography -V

So, welcome to this class on Hardware Security. So, we shall be continuing our discussions on Elliptic Curve Cryptographic hardware design.

(Refer Slide Time: 00:23)

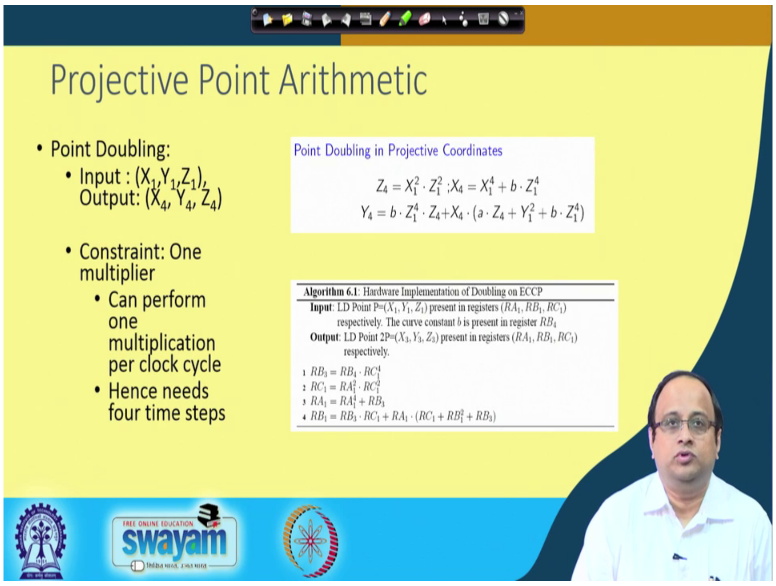
CONCEPTS COVERED

Concepts Covered:

- ❑ALU Design
- ❑Control Unit Design
- ❑Final Inversion
- ❑Top-level Scalar Multiplication Design & Verification

In particular today we shall we try to look into more details about the hardware that we started discussing in the last class. So, we shall we starting to look into the ALU design that is Arithmetic Logic Unit design. We shall be trying to take a more detail look into the control unit design and the final inversion along with quick discussion on the top level scalar multiplication and also how to verify the design that this methodology that we adopted for in our case.

(Refer Slide Time: 00:51)



Projective Point Arithmetic

- Point Doubling:
  - Input : (X_1, Y_1, Z_1), Output: (X_4, Y_4, Z_4)
  - Constraint: One multiplier
    - Can perform one multiplication per clock cycle
    - Hence needs four time steps

Point Doubling in Projective Coordinates

$$Z_4 = X_1^2 \cdot Z_1^2 \, ; X_4 = X_1^4 + b \cdot Z_1^4$$
$$Y_4 = b \cdot Z_1^4 \cdot Z_4 + X_4 \cdot (a \cdot Z_4 + Y_1^2 + b \cdot Z_1^4)$$

```
Algorithm 6.1: Hardware Implementation of Doubling on ECCP
Input: LD Point P=$(X_1, Y_1, Z_1)$ present in registers $(RA_1, RB_1, RC_1)$
       respectively. The curve constant $b$ is present in register $RB_4$
Output: LD Point 2P=$(X_3, Y_3, Z_3)$ present in registers $(RA_1, RB_1, RC_1)$
        respectively.
1 $RB_3 = RB_4 \cdot RC_1^4$
2 $RC_1 = RA_1^2 \cdot RC_1^2$
3 $RA_1 = RA_1^4 + RB_3$
4 $RB_1 = RB_3 \cdot RC_1 + RA_1 \cdot (RC_1 + RB_1^2 + RB_3)$
```

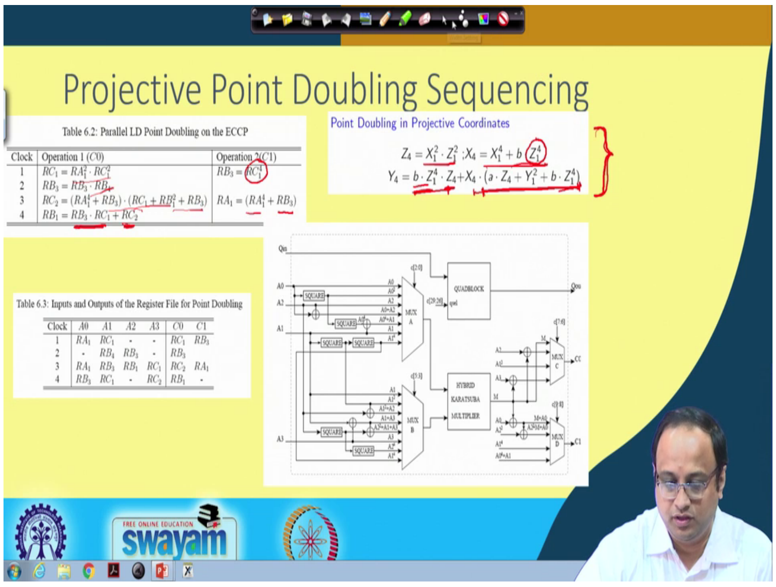
So, this is the quick recompilation of the projective point arithmetic. So, as we said already that the projective point arithmetic was introduced to reduce the number of costly inversions which are required in the underline finite field operations.

So, here are some equations on the point doubling; that means, when you are trying to double a point on the elliptic curve in projective coordinates. As you can see right that there are some multiplications which are involved in this step and in our design right if you remember that data path, we had a constraint of having only one multiplier.

So, therefore right we have to basically try to kind of multiply or essentially perform our operations with one multiplication at a time. So, our strategy will be that we will be ensuring that one multiplication is at least used every clock cycle; that means, the multiplier is not sitting idle in any clock cycle. So, this is just to reduce the number of clock cycles. So, in this case for example, we can see that if I schedule the operations ok; so one of the important steps that we need to kind of look into is how do you schedule this operation?

That means, how do you do this operation step by step. So, in this case we require 4 time steps to do the operation ok. And here are the corresponding steps which are detailed; so these are essentially nothing, but; so this algorithm is nothing, but essentially a depiction of how these steps would be performed.

(Refer Slide Time: 02:13)



Projective Point Doubling Sequencing

Table 6.2: Parallel LD Point Doubling on the ECCP

Clock	Operation 1 ($C0$)	Operation 2($C1$)
1	$RC_1 = RA_1^2 \cdot RC_1^2$	$RB_3 = RC_1^4$
2	$RB_3 = RB_3 \cdot RB_4$	
3	$RC_2 = (RA_1^4 + RB_3) \cdot (RC_1 + RB_1^2 + RB_3)$	$RA_1 = (RA_1^4 + RB_3)$
4	$RB_1 = RB_3 \cdot RC_1 + RC_2$	

Point Doubling in Projective Coordinates

$$Z_4 = X_1^2 \cdot Z_1^2 \,; X_4 = X_1^4 + b\, Z_1^4$$
$$Y_4 = b \cdot Z_1^4 \cdot Z_4 + X_4 \cdot (a \cdot Z_4 + Y_1^2 + b \cdot Z_1^4)$$

Table 6.3: Inputs and Outputs of the Register File for Point Doubling

Clock	$A0$	$A1$	$A2$	$A3$	$C0$	$C1$
1	RA_1	RC_1	-	-	RC_1	RB_3
2	-	RB_4	RB_3	-	RB_3	
3	RA_1	RB_3	RB_1	RC_1	RC_2	RA_1
4	RB_3	RC_1	-	RC_2	RB_1	-

So, more details look into this can be seen here in context with our data path for the arithmetic unit. So, you can see here that if you are interested to like for example, do these steps like.

Suppose these are the individual you know like successive 4 clock cycles clock 1, clock 2, clock 3 and clock 4. So, you see that in our arithmetic unit there are 2 output ports C 0 and C 1 this is for parallelism. So, we are trying to extract as much parallelisive as we can ok; given the constraint that we have got one multiplication or one multiplier.

So, here therefore, right when you are doing say R say when you are doing this computation like R A 1 square into RC 1 square. So, you are basically doing you occupying your multiplier, but at the same time you can see there in my data path. There are some other peripheral circuits like addition, squaring and like when 2 squarings are successive then you can also raise an element to an power of 4 ok.

So, therefore, these kinds of computations are kept so that you can essentially do it very much independent of the multiplier ok. And they are also less in terms of cost; so you can essentially provide dedicated support for them ok. So, you have to basically plane this entire circuit. So, essentially the cracks of the design lies in how you can know like nicely develop an architecture ok.

So, so that essentially is a sort of thing that we only gather out of trying few examples or through experience. But you can essentially get sudden you know you can try to understand the relationship between the operation that we need to do and the final circuit that we have at hand.

So, let me try to detail for example, like an at least try to understand how to read these tables basically ok. So, you can see that there is in this table basically detail the input and output of the register file for point doubling. Because the register file as we discussed yesterday works hand in hand with your arithmetic unit ok.

So, the arithmetic units basically receive some inputs from the register file; does some computation and the result is again stored back in the register file. So; that means, C 0 and C 1 and Q out that again stored back into the register file. So, if you see here the register file produces the register file had outputs like A 0, A 1, A 2 and A 3 ok. And it produces A 0 and A 1 in these 2 buses it floats a data of R A 1 and RC 1 ok.

So, now R A 1 and RC 1 are essentially the first registers of the first and the third register banks. So, R A 1 stores the value of X 1 and RC 1 and actually R A 1 stores the you know. So, R A 1 stores the coordinate X 1 and RC 1 stores the coordinate Z 1.

So; that means, it is basically what you are doing now is that when you are when you want to see multiply R A 1 square into RC 1 square; that means, this multiplier should receive through this multiplexer the inputs A 0 square and A 1 square basically ok. So, therefore, A 0 1 square and A 1 square should be passed in as an input; the multiplier will develop this result and if you see this output right needs to be multiplex out ok.

So, I will again come back to this when we talk about the control path or the control signals. But this is something that we need to do and this steering of information or data should be done by the control logic ok. That means the select lines of the multiplexers will be switched accordingly to the; our control logic.

And therefore, right we essentially have a clean separation of the data path and the control path which is desirable in any good architecture design. So, finally, right we have got this result; that means, for example, we calculate say; you know like X 1 square into Z 1 square. This is essentially performance of this step and in parallel you are also performing RC 1 to the power of 4; so that stands for Z 1 to the power of 4 ok.

So, therefore, right we have also calculating ah; if I may write you know like that you are also calculating RC 1 to the power of 4. And that is essentially nothing, but this operation that is Z 1 to the power of 4 ok. And this operation is essentially this operation that is X 1 square into Z 1 square. So, you see that these 2 things are performed in the same clock cycle in this architecture.

Likewise right in the next clock cycle we perform this computation that is RB 3 into RB 4. So, RB 3 is nothing, but Z 1 to the power of 4 and if you remember RB 4 was used to store the curve constant B ok. So; that means, B into Z 1 to the power of 4 is performed in this computation. And B Z 1 to the power of 4 is there in few other places also; so therefore this result is ready with us in RB 3.

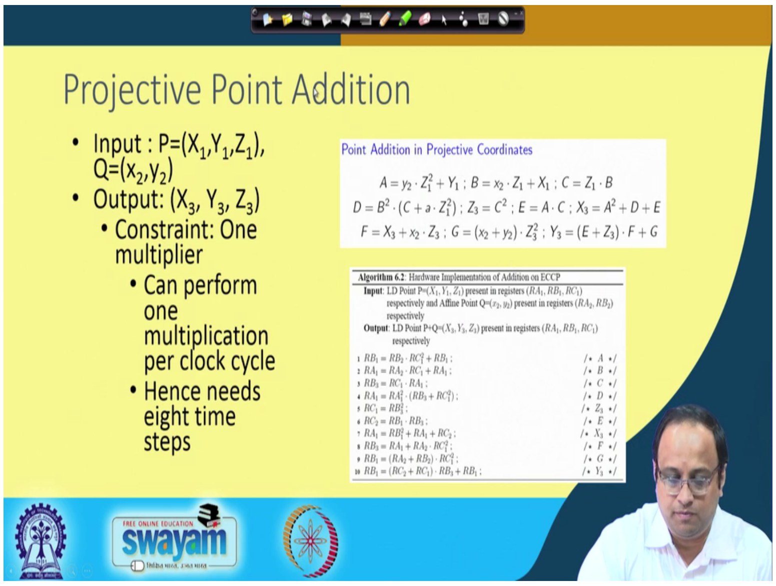
So, in the next clock cycle we basically kind of calculate this part that is we basically you know like in the first; in the in this in this computation we perform this part that is we add remember that A is 1; so we basically add Z 4 with Y 1 square with B Z 1 to the power of 4. So, B Z 1 to the power of 4 is already ready in RB 3 ok. So, therefore, the RB 3 essentially stores the value of B Z 1 to the power of 4. You add that with RB 1 square; so RB 1 square is nothing, but your Y 1 square and A Z 1 to the power 4; A being 1 is equal to Z 4 ok. So, therefore, this result is already ready with us and essentially this is stored in RC 1 ok.

So, so therefore, right you essentially have these 3 things ready and the other part is again you know like this part that is R A 1 to the power of 4 plus RB 3 ok. So, this is essentially trying to basically compute this part. So, therefore, essentially this nothing, but raising X 1 to the power of 4 and adding that with RB 3.

So, therefore, in the final clock cycle you just need to add this component. So, therefore, you need to add this component with this result ok. So, remember that we already had this RC 2 ready in the previous clock cycle; so we just add RB 3 into RC 1. So, RB 3 into RC 1 stands for this part of the computation because you already have RB 3 storing B Z 1 to the power of 4; you multiply with this component to get this part ok.

So, therefore, this pretty much kind of explain how the 4 clock cycles work and you essentially perform the doubling operation. So, I did not you know like derive these equations in the Lopez Dahab coordinates, but I leave it you as an exercise to do this to do this calculation. Rather I will be trying to you know like explain how the addition works, because that is slightly more complicated and we will try to see how the equations work in that case.

(Refer Slide Time: 08:59)



Projective Point Addition

- Input : $P=(X_1,Y_1,Z_1)$, $Q=(x_2,y_2)$
- Output: (X_3, Y_3, Z_3)
  - Constraint: One multiplier
    - Can perform one multiplication per clock cycle
    - Hence needs eight time steps

Point Addition in Projective Coordinates

$$A = y_2 \cdot Z_1^2 + Y_1 \; ; \; B = x_2 \cdot Z_1 + X_1 \; ; \; C = Z_1 \cdot B$$
$$D = B^2 \cdot (C + a \cdot Z_1^2) \; ; \; Z_3 = C^2 \; ; \; E = A \cdot C \; ; \; X_3 = A^2 + D + E$$
$$F = X_3 + x_2 \cdot Z_3 \; ; \; G = (x_2 + y_2) \cdot Z_3^2 \; ; \; Y_3 = (E + Z_3) \cdot F + G$$

Algorithm 6.2: Hardware Implementation of Addition on ECCP

Input: LD Point P=(X_1, Y_1, Z_1) present in registers (RA_1, RB_1, RC_1) respectively and Affine Point Q=(x_2, y_2) present in registers (RA_2, RB_2) respectively

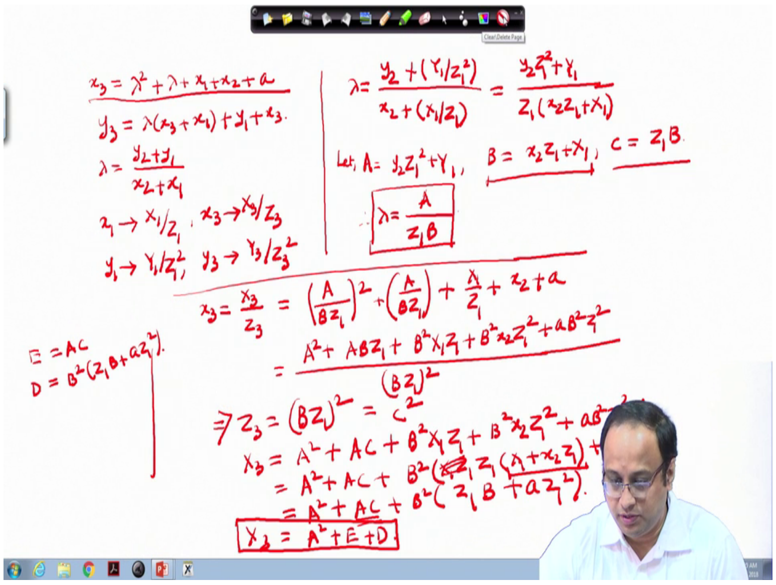
Output: LD Point P+Q=(X_3, Y_3, Z_3) present in registers (RA_1, RB_1, RC_1) respectively

```
1  $RB_1 = RB_2 \cdot RC_1^2 + RB_1$ ;            /* A */
2  $RA_1 = RA_2 \cdot RC_1 + RA_1$ ;              /* B */
3  $RB_3 = RC_1 \cdot RA_1$ ;                     /* C */
4  $RA_1 = RA_1^2 \cdot (RB_3 + RC_1^2)$ ;        /* D */
5  $RC_1 = RB_3^2$ ;                             /* $Z_3$ */
6  $RC_2 = RB_1 \cdot RB_3$ ;                     /* E */
7  $RA_1 = RB_1^2 + RA_1 + RC_2$ ;               /* $X_3$ */
8  $RB_3 = RA_1 + RA_2 \cdot RC_1^2$ ;           /* F */
9  $RB_1 = (RA_2 + RB_2) \cdot RC_1^2$ ;         /* G */
10 $RB_1 = (RC_2 + RC_1) \cdot RB_3 + RB_1$ ;    /* $Y_3$ */
```

So, these are my projective point addition operations and essentially this looks little bit more complex than the doubling equations ok. So, we will try to take a look into that derivation.

So, let me try to explain that and so therefore, right if you remember the original affine coordinate equations.

(Refer Slide Time: 09:23)



$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$
$$y_3 = \lambda(x_3 + x_1) + y_1 + x_3.$$
$$\lambda = \frac{y_2 + y_1}{x_2 + x_1}$$
$$x_1 \to X_1/Z_1, \quad x_3 \to X_3/Z_3$$
$$y_1 \to Y_1/Z_1^2, \quad y_3 \to Y_3/Z_3^2$$

$$\lambda = \frac{y_2 + (Y_1/Z_1^2)}{x_2 + (X_1/Z_1)} = \frac{y_2 Z_1^2 + Y_1}{Z_1(x_2 Z_1 + X_1)}$$

Let, $A = y_2 Z_1^2 + Y_1$, $B = x_2 Z_1 + X_1$, $C = Z_1 B$.

$$\boxed{\lambda = \frac{A}{Z_1 B}}$$

$$x_3 = \frac{X_3}{Z_3} = \left(\frac{A}{BZ_1}\right)^2 + \left(\frac{A}{BZ_1}\right) + \frac{X_1}{Z_1} + x_2 + a$$
$$= \frac{A^2 + ABZ_1 + B^2 X_1 Z_1 + B^2 x_2 Z_1^2 + aB^2 Z_1^2}{(BZ_1)^2}$$

$E = AC$
$D = B^2(Z_1 B + aZ_1^2).$

$$\Rightarrow Z_3 = (BZ_1)^2 = C^2$$
$$X_3 = A^2 + AC + B^2 X_1 Z_1 + B^2 x_2 Z_1^2 + aB^2 Z_1^2$$
$$= A^2 + AC + B^2(\cancel{X_1} Z_1 Z_1 (X_1 + x_2 Z_1) +$$
$$= A^2 + AC + B^2(Z_1 B + aZ_1^2).$$
$$= A^2 + AC +$$
$$\boxed{X_3 = A^2 + E + D}$$

So, the original affine coordinate equations where as follows you have x 3 which is equal to; so I will write or use lambda to store the slope in this case. So, lambda square plus lambda plus x 1 plus x 2 plus a ok. So, and likewise y 3 was equal to lambda into x 3 plus x 1 plus y 1 plus x 3 and lambda is the slope. So, in this case it is y 2 plus y 1; remember it is in characteristic 2; so divided by x 2 plus x 1 ok.

So, now for projective coordinates we will make this. So, remember it is a Lopez Dahab projective coordinates they are different projective coordinates. So, we will be using the Lopez Dahab projective coordinates; where small x 1 is mapped into capital X 1 by Z 1 and x 3 is mapped into capital X 3 by Z 3. And small y 1 is mapped into Y 1 by Z 1 square and small y 3 is mapped into Y 3 by Z 3 square.

So, now with this replacements or substitutions; you can again look into the value of lambda. So, lambda is now equal to; so remember what we will do is that, we will do this operation in affine coordinates ok. So, I will be keeping y 2 as it is. So, y 2 will remember remain in is affine coordinates rather this operation is done in mix coordinates. So, the y 2 is remaining in affine coordinates and the other one will be in projective coordinate.

So, we will write y 2 plus Y 1 by Z 1 square divided by x 2 plus X 1 by Z 1 ok. So, that turns out to with little bit of you know like simplifications y 2 Z 1 square plus capital Y 1 divided by Z 1 into x 2 Z 1 plus capital X 1.

So, therefore, right we will write here that let A hold the value of y 2 Z 1 square plus Y 1 and B hold x 2 Z 1 plus X 1. And let us C hold Z 1 B ok; so let us C hold Z 1 B. So, in that case therefore, lambda becomes nothing, but A divided by Z 1 B; A divided by Z 1 B ok.

So, therefore, right with this derivation we have got x 3; small x 3 equal to capital X 3 divided by Z 3 right. So, that is equal to ah; so we have basically having this equation for x 3. So, we are basically trying to calculate the value of x 3, but remember that x 3 is equal to lambda square plus lambda plus x 1 plus x 2 plus a. So, that is essentially given by this formula.

So, we basically can write now therefore, using this value of lambda because remember this is my expression for lambda. So, I can write this as A divided by B Z 1 square plus A divided by B Z 1 ok; plus X 1 by Z 1 plus X 2 plus a. Remember x 2 for small x 2 remains in the affine coordinates.

So, therefore this essentially is nothing, but B Z 1 whole square in the denominator. The numerator is A square plus A B Z 1 plus B square X 1; Z 1 plus B square small x 2 Z 1 square plus a B square Z 1 square. So, therefore, what we will say is at Z 3; this implies that since x 3 is equal to X 3 by Z 3; Z 3 is equal to B Z 1 whole square and that is equal to also C square because C is Z 1 B ok.

And capital X 3 is equal to A square plus ABZ 1 ok; this we can write as AC because C is Z 1 B. So, A square plus AC plus B square X 1; Z 1 plus B square small x 2 Z 1 square plus a B square Z 1 square. And that is equal to again A square plus AC plus see I can take B square common from all sides this.

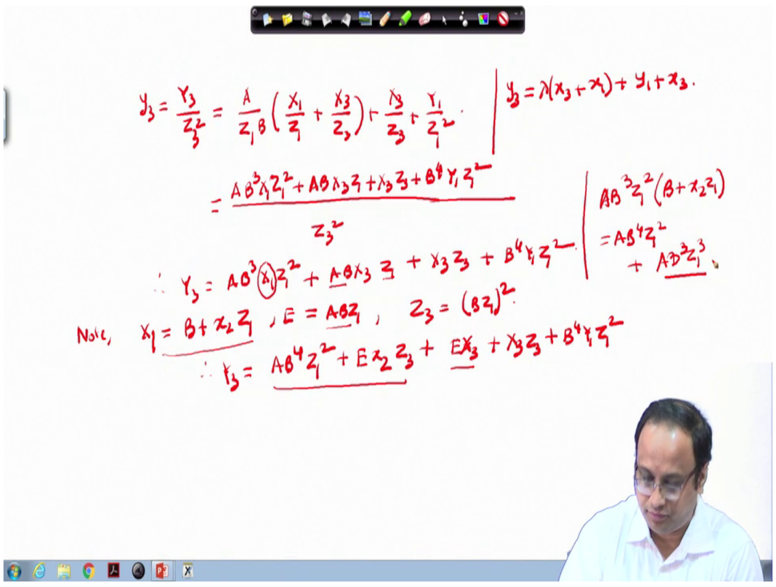
So, I can write this as B square X 1 Z 1; so actually I can probably you know like right this also as I can take Z 1 common and I can write X 1 plus x 2 Z 1 plus a Z 1 square ok. And this I can write again as A square plus AC plus B square Z 1 B. Because see that this part is nothing, but already B which is essentially as written here ok. So, this is my capital B; so I can write this is B square into Z 1 B plus a Z 1 square ok.

So, therefore, right I mean we have more or less compact equation and we have to kind of remember that these are all the intermediate steps which you are calculating when you are doing your addition operation ok. So, likewise you can derive the equation for y 3 ok; so and you can again do that. So, I will try to you know like derive the equation for y 3.

So, one thing we can observe that what I can also do few more substitutions here just to for example, this AC; I can write as say you know like E is equal to AC E equal to AC. And D is equal to B square Z 1 B plus a Z 1 square ok. So, if I write this then X 3 becomes equal to A square plus E plus D right; this is a more compact expression for X 3.

Likewise you can derive for y 3 ok; so let us do this after this. So, I will be clearing this part and again looking into how to derive for y 3.

(Refer Slide Time: 16:43)



$$y_3 = \frac{Y_3}{Z_3^2} = \frac{A}{Z_1 B}\left(\frac{X_1}{Z_1} + \frac{X_3}{Z_3}\right) + \frac{X_3}{Z_3} + \frac{Y_1}{Z_1^2}.$$

$$y_3 = \lambda(x_3 + x_1) + y_1 + x_3.$$

$$= \frac{AB^3 X_1 Z_1^2 + ABX_3 Z_1 + X_3 Z_3 + B^4 Y_1 Z_1^2}{Z_3^2}$$

$$\therefore Y_3 = AB^3 \boxed{X_1} Z_1^2 + \underline{A}BX_3 Z_1 + X_3 Z_3 + B^4 Y_1 Z_1^2.$$

$$AB^3 Z_1^2 (B + x_2 Z_1) = AB^4 Z_1^2 + \underline{AB^3 Z_1^3}.$$

Note, $\underline{X_1 = B + x_2 Z_1}$, $E = \underline{ABZ_1}$, $Z_3 = (BZ_1)^2$.

$$\therefore Y_3 = \underline{AB^4 Z_1^2 + E x_2 Z_3} + \underline{EX_3} + X_3 Z_3 + B^4 Y_1 Z_1^2$$

So, y 3 is; so is equal to Y capital Y 3 divided by Z 3 square and that is equal to A by Z; Z 1 B. So, divided into X 1 by Z 1 plus X 3 by Z 3 plus X 3 by Z 3 plus Y 1 by Z 1 square. So, this actually follows from the equation of y 3. So, if you remember that y 3 was equal to lambda into x 3 plus x 1 plus y 1 plus small x 3 ok.

So, now you are transforming them into the projective domain and using the value of lambda and the other things that we already saw. So, you can essentially write y 3 in this form and therefore, right you can again do a similar thing that we did for x 3; you can write the denominator as you know like.

So, this is a nothing, but writing the denominator as Z 3 square and so, Z 3 square. And then writing this as AB cube; X 1 Z 1 square plus AB X 3 Z 1 plus X 3 Z 3 plus B to the power of 4 Y 1 Z 1 square right.

So,; so therefore, right I mean if you have these components then now you can write therefore, Y 3 as AB cube; X 1 Z 1 square plus AB; X 3 Z 1 plus X 3; Z 3 plus B to the power of 4 Y 1 Z 1 square ok. So, now, note that we already had this in the previous derivations that capital X 1 was equal to B plus x 2 Z 1 and E was equal to AB Z 1 and also that Z 3 was equal to B Z 1 whole square ok.

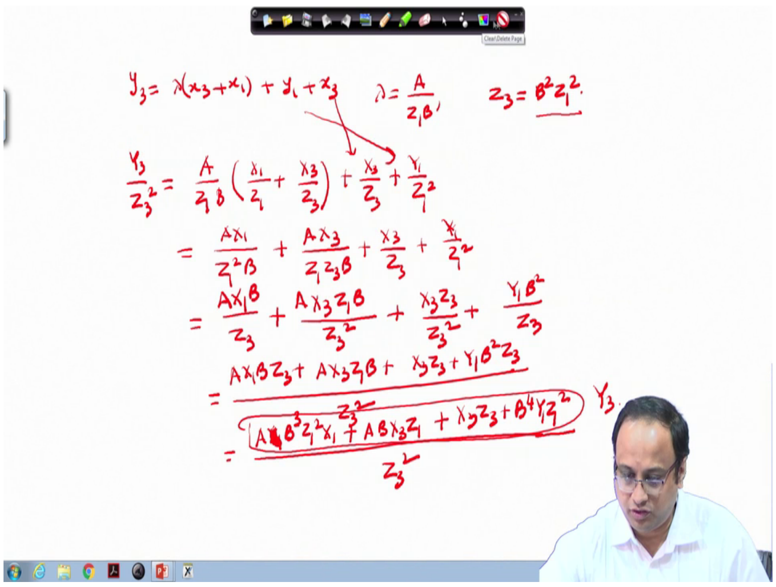
So, therefore, write Y 3 is equal to A B to the power of 4 Z 1 square plus E small x 2; Z 3 plus capital E X 3, capital E X 3 plus capital X 3 Z 3 plus B to the power of 4; Y 1 Z 1 square ok.

So, this is just substituting this AB Z 1; we are substituting E here and we have got E X 3 because of that and the other term is you know like. So, that is E x is a E x 2 is Z 1 and you also have got here this part. So, this part you are getting by from this equation; so capital X 1 you can actually substitute as B plus x 2 Z 1 ok.

So, if you substitute B plus x 2 Z 1; then you will have B to the power; AB to the power of 4 Z 1 square coming up as one part. And likewise; so you can just verify this that is if I for example, write AB cube Z 1 square and instead of capital X 1 that is this part; I write B 1 B plus x 2 Z 1, then I have got AB to the power of 4; Z 1 square plus AB cube.

And I have got AB cube; Z 1 to the power of 3 ok, so this part essentially right I mean yeah. So, this part should get cancelled out right with your; with your second term right that is this term. So, like we derive the steps for x 3 we will be deriving the equation for y 3 in a similar manner ok.

(Refer Slide Time: 21:39)



$$y_3 = \lambda(x_3 + x_1) + y_1 + x_3 \qquad \lambda = \frac{A}{Z_1 B} \qquad Z_3 = \underline{B^2 Z_1^2}.$$

$$\frac{Y_3}{Z_3^2} = \frac{A}{Z_1 B}\left(\frac{X_1}{Z_1} + \frac{X_3}{Z_3}\right) + \frac{X_3}{Z_3} + \frac{Y_1}{Z_1^2}$$

$$= \frac{AX_1}{Z_1^2 B} + \frac{AX_3}{Z_1 Z_3 B} + \frac{X_3}{Z_3} + \frac{Y_1}{Z_1^2}$$

$$= \frac{AX_1 B}{Z_3} + \frac{AX_3 Z_1 B}{Z_3^2} + \frac{X_3 Z_3}{Z_3^2} + \frac{Y_1 B^2}{Z_3}$$

$$= \frac{AX_1 B Z_3 + AX_3 Z_1 B + X_3 Z_3 + Y_1 B^2 Z_3}{Z_3^2}$$

$$= \frac{\boxed{AB^3 Z_1^2 X_1 + ABX_3 Z_1 + X_3 Z_3 + B^4 Y_1 Z_1^2}\ Y_3}{Z_3^2}$$

So, so if you remember right I mean the equation for y 3 that we had in the previous previously when we stated equation for y 3, we had y 3 equal to lambda into x 3 plus x 1 plus y 1 plus x 3 ok. And we derive that lambda we wrote in a compact format was A by Z 1 B ok.

And also we had you know like seen that Z 3 in the when we derive equation for X right we are Z 3 equal to B square Z 1 square. So, therefore, right now we can write Y 3 in a similar way we can write Y 3 in the projective coordinate as Y 3 divided by Z 3 square and that is equal to A by Z 1 B into X 1 by Z 1 plus capital X 3 by Z 3 plus X 3 by Z 3.

So, that is from these term and plus Y 1 by Z 1 square; so that is from this term ok. So, therefore now we essentially can break this I mean 2 parts. So, we can write this as A X 1 divided by Z 1 square B plus A X 3 divided by Z 1 Z 3 B plus X 3 by Z 3 plus Y 1 by Z 1 square.

So, now we can multiply the numerator denominator here Y B. And you see that if I do that then I get B square Z 1 square and that is equal to Z 3 ok. So therefore, I can write the denominator as Z 3 and that means, that I have multiplied the numerator also by B; so that I have got A X 1 B; plus I have got A X 3 and I multiply again the numerator and the denominator by in this case Z 1 B ok.

So, if I multiplied by Z 1 B; then I have got Z 1 B whole square in the denominator. And Z 1 B whole square means Z 3; so I have got another Z 3. So, this becomes Z 3 square plus again I multiply X 3 Z 3 by and I get Z 3 square here plus Y 1; B B square and this become Z 1 B 0. So, that is again equal to Z 3.

So, now we can take Z 3 square in the denominator and therefore, the numerator becomes AX 1 B Z 3 plus AX 3; Z 1 B this does not change plus X 3 Z 3; this also does not change. This becomes multiplied with Z 3; Y 1 B square Z 3.

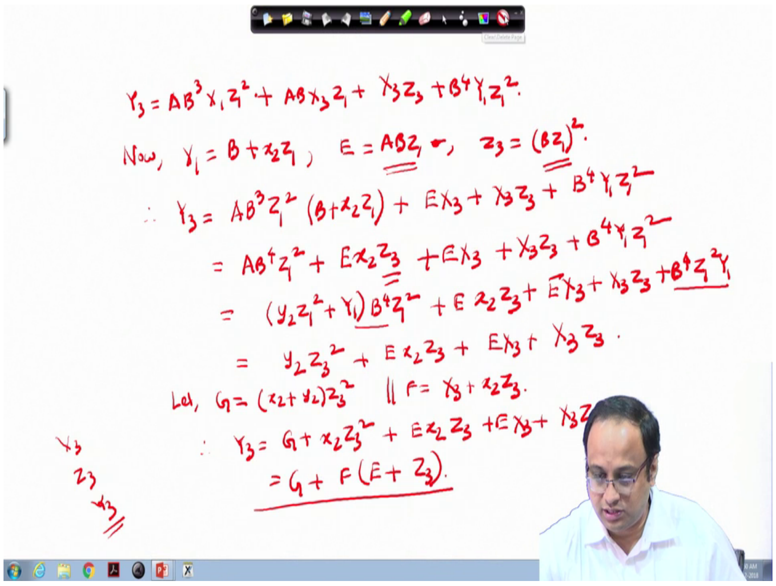
And that is equal to if I write in the numerator again remember that I have got Z 3 as equal to B square Z 1. So, if I plug that back then I have got A X 1 and A X. So, if I just plug plug in back Z 3 as B square Z 1 then I have got A; I can write here this part as A B to the power of 3 Z 1 square X 1 ok.

So, just writing the X 1 the end plus A B X 3 Z 1 plus this part again if I bring in. So, this part let me write as X 3 Z 3 and for this part I have got; if I bringing and substitute in place of 3 B square Z 1; then I have got B to the power of 4 Y 1 Z 1 square divided by Z

3 square. So, therefore, right I have got I have got this numerator right essentially is nothing, but is essentially Y 3; so, this is my equation for Y 3 ok.

So, so let me just use it and try to see that whether we can; let us try to simplify this equation and for that I will be using another sorry. So, let me just clean this; 1 second . So, let me just clean this up and write it has clean the equation.

(Refer Slide Time: 26:29)



$$Y_3 = AB^3X_1Z_1^2 + ABX_3Z_1 + X_3Z_3 + B^4Y_1Z_1^2.$$

Now, $Y_1 = B + x_2Z_1$, $E = \underline{\underline{ABZ_1}}$, $Z_3 = \underline{\underline{(BZ_1)}}^2$.

$$\therefore Y_3 = AB^3Z_1^2(B + x_2Z_1) + EX_3 + X_3Z_3 + B^4Y_1Z_1^2$$

$$= AB^4Z_1^2 + Ex_2\underline{\underline{Z_3}} + EX_3 + X_3Z_3 + B^4Y_1Z_1^2$$

$$= (y_2Z_1^2 + \underline{Y_1)B^4}Z_1^2 + E\,x_2Z_3 + EX_3 + X_3Z_3 + \underline{B^4Z_1^2Y_1}$$

$$= y_2Z_3^2 + E\,x_2Z_3 + EX_3 + X_3Z_3.$$

Let, $G = (x_2 + y_2)Z_3^2$ || $F = X_3 + x_2Z_3$.

$$\therefore Y_3 = G + x_2Z_3^2 + Ex_2Z_3 + EX_3 + X_3Z_3$$

$$= \underline{G + F(E + Z_3).}$$

X_3

Z_3

Y_3

So, we have got Y 3 is equal to AB cube X 1; Z 1 square ok. So, AB cube X 1 Z 1 square plus AB X 3 Z 1 plus X 3; Z 3 plus B to the power of 4 Y 1 Z 1 square ok.

So, now we will make few substitutions. So, for example, note that X 1 or capital X 1 is equal to B plus x 2 Z 1 and E is equal to a B Z 1 E equal to AB Z 1. And we also have as already we noted that Z 3 is equal to B Z 1 whole square ok.

So, therefore, right Y 3 is equal to if I do this then Y 3 is equal to A B to the power of 3 AB to the power of 3 Z 1 square and I instead of X 1; I can write B plus X 2 Z 1 plus plus E; capital X 3 plus X 3, Z 3 plus B to the power of 4 Y 1, Z 1 square. And this is equal to A: B to the power of 4 Z 1 square plus plus E X 2 Z 3 ok.

So, E X 2 X 3 because you know that Z 3 is equal to B squared Z 1 square. So, therefore, you can get it from here and you can also plug in E equal to a B Z 1 ok. So, you will get E X 2 Z 3 plus E X 3 plus X 3 Z 3 plus B to the power of 4 Y 1 Z 1 square. And that again you can write as this you can; you can write this as y 2 Z 1 square plus Y 1; B to the power of 4 Z 1 square being taken common ok.

So, so that essentially comes from the equation that we have for A and this plus E x 2 Z 3 plus E X 3 plus X 3 Z 3 plus B to the power of 4 Z 1 square Y 1. So, this becomes equal to y 2 Z 3 square plus E X 2 Z 3 plus E X 3 plus X 3 Z 3 ok. So, so this essentially right I mean you can see that B to the power of 4 Z 1 square and Y 1 square. So, these 2 term cancels of and this is essentially the term that I have finally.

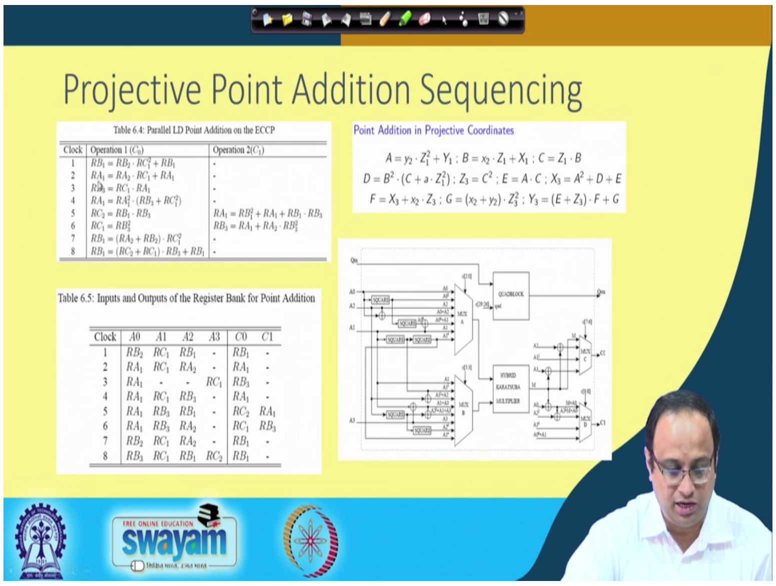
So, then I can also right few are few more substitutions like say G is equal to x 2 plus y 2 into Z 3 square and F is equal to X 3 plus x 2 Z 3. With this I can write therefore, that Y 3 is equal to G plus X 2 Z 3 square plus E x 2 Z 3 plus E X 3 plus X 3 Z 3 and that is equal to G plus F into E plus Z 3 ok.

So, that gives you the final equation for Y 3. So, we already had X 3, we had Z 3 and now we also can calculate Y 3. So, we are basically all the components for the addition equation. So, now let us get back to the you know like slide and try to see that whether these are the equation. So, this essentially the starting from the bottom; so, you can see that this is your Y 3 equation; there is exactly what we define here ok. So, E plus Z 3 into F plus G ok.

Likewise we already had Z 3 as C square and also you can verify the other equations step by step essentially it tallies with our derivation.

So, therefore, we have now equations now once we have this equations right; we have to see like how many clock cycles are required given the constant that we have got one multiplier. It turns out that in this case we require if you do at scheduling right.

(Refer Slide Time: 31:59)



Projective Point Addition Sequencing

Table 6.4: Parallel LD Point Addition on the ECCP

Clock	Operation 1 (C_0)	Operation 2(C_1)
1	$RB_1 = RB_2 \cdot RC_1^2 + RB_1$	-
2	$RA_1 = RA_2 \cdot RC_1 + RA_1$	-
3	$RB_3 = RC_1 \cdot RA_1$	-
4	$RA_1 = RA_1^2 \cdot (RB_3 + RC_1^2)$	-
5	$RC_2 = RB_1 \cdot RB_3$	$RA_1 = RB_1^2 + RA_1 + RB_1 \cdot RB_3$
6	$RC_1 = RB_3^2$	$RB_3 = RA_1 + RA_2 \cdot RB_3^2$
7	$RB_1 = (RA_2 + RB_2) \cdot RC_1^2$	-
8	$RB_1 = (RC_2 + RC_1) \cdot RB_3 + RB_1$	-

Point Addition in Projective Coordinates

$$A = y_2 \cdot Z_1^2 + Y_1 \ ; \ B = x_2 \cdot Z_1 + X_1 \ ; \ C = Z_1 \cdot B$$
$$D = B^2 \cdot (C + a \cdot Z_1^2) \ ; \ Z_3 = C^2 \ ; \ E = A \cdot C \ ; \ X_3 = A^2 + D + E$$
$$F = X_3 + x_2 \cdot Z_3 \ ; \ G = (x_2 + y_2) \cdot Z_3^2 \ ; \ Y_3 = (E + Z_3) \cdot F + G$$

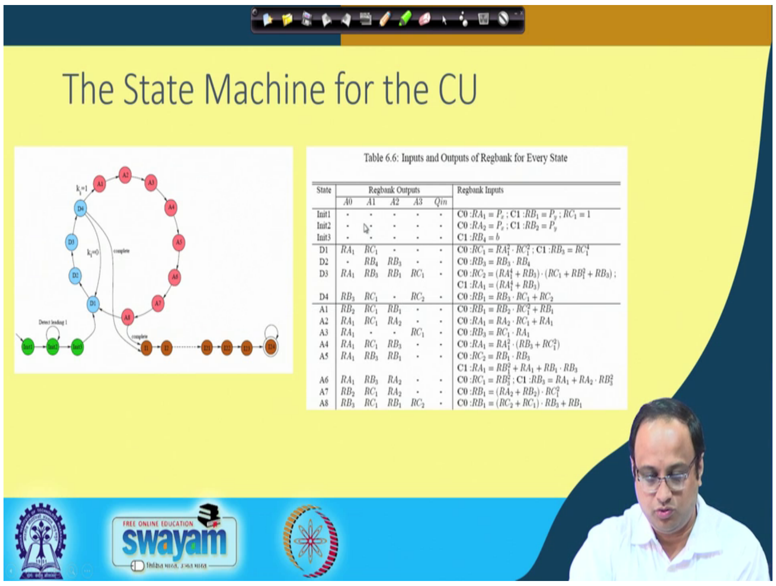
Table 6.5: Inputs and Outputs of the Register Bank for Point Addition

Clock	$A0$	$A1$	$A2$	$A3$	$C0$	$C1$
1	RB_2	RC_1	RB_1	-	RB_1	-
2	RA_1	RC_1	RA_2	-	RA_1	-
3	RA_1	-	-	RC_1	RB_3	-
4	RA_1	RC_1	RB_3	-	RA_1	-
5	RA_1	RB_3	RB_1	-	RC_2	RA_1
6	RA_1	RB_3	RA_2	-	RC_1	RB_3
7	RB_2	RC_1	RA_2	-	RB_1	-
8	RB_3	RC_1	RB_1	RC_2	RB_1	-

We require 8 clock cycles to perform this operation compared to your doubling where you at 4 clock cycle requirements. So, here you can see again you know you can verify that if I have got 2 outputs in parallel which I can produce. So, every clock cycle I am using one multiplication; the multiplied is always used in every clock cycle.

So, you can see that every time it is used the other component or other output is often used to do peripheral computations ok. So, in that case you have got 8 time steps which are required.

(Refer Slide Time: 32:31)



The State Machine for the CU

Table 6.6: Inputs and Outputs of Regbank for Every State

State	Regbank Outputs					Regbank Inputs
	$A0$	$A1$	$A2$	$A3$	Qin	
Init1	-	-	-	-	-	C0 :$RA_1 = P_x$; C1 :$RB_1 = P_y$; $RC_1 = 1$
Init2	-	-	-	-	-	C0 :$RA_2 = P_x$; C1 :$RB_2 = P_y$
Init3	-	-	-	-	-	C1 :$RB_4 = b$
D1	RA_1	RC_1	-	-	-	C0 :$RC_1 = RA_1^2 \cdot RC_1^2$; C1 :$RB_3 = RC_1^4$
D2	-	RB_4	RB_3	-	-	C0 :$RB_3 = RB_3 \cdot RB_4$
D3	RA_1	RB_3	RB_1	RC_1	-	C0 :$RC_2 = (RA_1^4 + RB_3) \cdot (RC_1 + RB_1^2 + RB_3)$; C1 :$RA_1 = (RA_1^4 + RB_3)$
D4	RB_3	RC_1	-	RC_2	-	C0 :$RB_1 = RB_3 \cdot RC_1 + RC_2$
A1	RB_2	RC_1	RB_1	-	-	C0 :$RB_1 = RB_2 \cdot RC_1^2 + RB_1$
A2	RA_1	RC_1	RA_2	-	-	C0 :$RA_1 = RA_2 \cdot RC_1 + RA_1$
A3	RA_1	-	-	RC_1	-	C0 :$RB_3 = RC_1 \cdot RA_1$
A4	RA_1	RC_1	RB_3	-	-	C0 :$RA_1 = RA_1^2 \cdot (RB_3 + RC_1^2)$
A5	RA_1	RB_3	RB_1	-	-	C0 :$RC_2 = RB_1 \cdot RB_3$ C1 :$RA_1 = RB_1^2 + RA_1 + RB_1 \cdot RB_3$
A6	RA_1	RB_3	RA_2	-	-	C0 :$RC_1 = RB_3^2$; C1 :$RB_3 = RA_1 + RA_2 \cdot RB_3^2$
A7	RB_2	RC_1	RA_2	-	-	C0 :$RB_1 = (RA_2 + RB_2) \cdot RC_1^2$
A8	RB_3	RC_1	RB_1	RC_2	-	C0 :$RB_1 = (RC_2 + RC_1) \cdot RB_3 + RB_1$

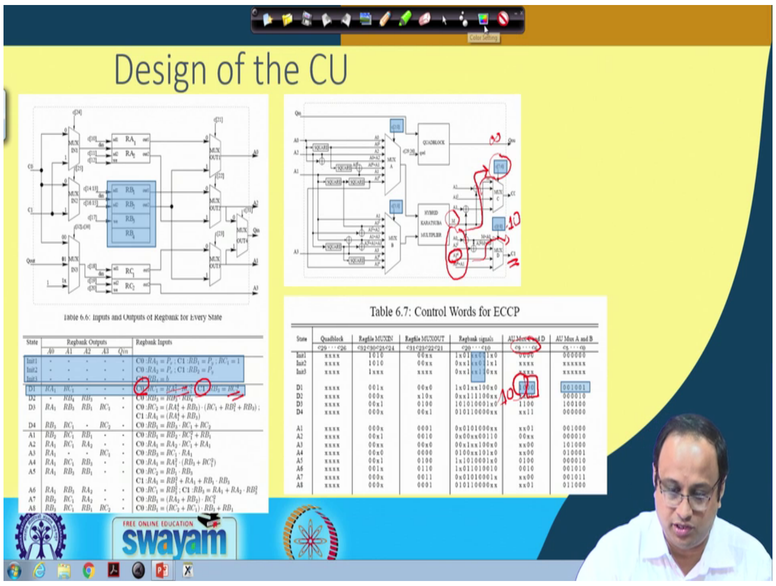
So, let us take a look at the control unit, in the control unit we will see that we basically have got you know like the data flow essentially the right for doing the entire scalar multiplication, we will have 4 distinct parts ok.

The first part will be the initialization part where initialize your like the base points and the constants for the curve. In the second part you do the doubling operation depending upon doubling you always do. So, doubling you expand like 4 clock cycles so that are shown by this blue circles and that is followed by the 8; 8 clock cycles which are required for doing addition operations ok, but this is conditional in this architecture ok.

Later on we will see it is not a very secured way of doing it, but at least from performance point of view this looks fine. So, you do basically 8 take 8 clock cycles for doing the addition. And once you have done all these things right then done or process the entire scalar; you go for the final result where you get the affine coordinates and you get back the ultimate result and for that you do an inversion.

So, that essentially shown by these brown circles and for the inversion we will see that you have got 24 clock cycle requirements ok. That it is done by the Ethos Uzi inversion algorithm. So, these are the different components for doing the operation; so, you have got various steps.

(Refer Slide Time: 33:49)



Design of the CU

Table 6.6: Inputs and Outputs of Regbank for Every State

State	Regbank Outputs					Regbank Inputs
	A0	A1	A2	A3	Qin	
Init1	·	·	·	·	·	C0 : $RA_1 = P_x$; C1 : $RB_1 = P_y$; $RC_1 = 1$
Init2	·	·	·	·	·	C0 : $RA_2 = P_x$; C1 : $RB_2 = P_y$
Init3	·	·	·	·	·	C1 : $RB_4 = b$
D1	RA_1	RC_1	·	·	·	C0 : $RC_1 = RA_1^4 \cdot RC_1^4$; C1 : $RB_3 = RC_1^4$
D2	·	RB_4	RB_3	·	·	C0 : $RB_3 = RB_3 \cdot RB_4$
D3	RA_1	RB_3	RB_1	RC_1	·	C0 : $RC_2 = (RA_1^4 + RB_3) \cdot (RC_1 + RB_1^2 + RB_3)$; C1 : $RA_1 = (RA_1^4 + RB_3)$
D4	RB_3	RC_1	·	RC_2	·	C0 : $RB_1 = RB_3 \cdot RC_1 + RC_2$
A1	RB_2	RC_1	RB_1	·	·	C0 : $RB_1 = RB_2 \cdot RC_1^2 + RB_1$
A2	RA_1	RC_1	RA_2	·	·	C0 : $RA_1 = RA_2 \cdot RC_1 + RA_1$
A3	RA_1	·	·	RC_1	·	C0 : $RB_3 = RC_1 \cdot RA_1$
A4	RA_1	RC_1	RB_3	·	·	C0 : $RA_1 = RA_1^2 \cdot (RB_3 + RC_1^2)$
A5	RA_1	RB_3	RB_1	·	·	C0 : $RC_2 = RB_1 \cdot RB_3$; C1 : $RA_1 = RB_1^2 + RA_1 + RB_1 \cdot RB_3$
A6	RA_1	RB_3	RA_2	·	·	C0 : $RC_1 = RB_1^2$; C1 : $RB_3 = RA_1 + RA_2 \cdot RB_1^2$
A7	RB_2	RC_1	RA_2	·	·	C0 : $RB_1 = (RA_2 + RB_2) \cdot RC_1^2$
A8	RB_3	RC_1	RB_1	RC_2	·	C0 : $RB_1 = (RC_2 + RC_1) \cdot RB_3 + RB_1$

Table 6.7: Control Words for ECCP

State	Quadblock	Regfile MUXIN	Regfile MUXOUT	Regbank signals	AU Mux C and D	AU Mux A and B
	$c_{29} \cdots c_{26}$	$c_{32}c_{30}c_{25}c_{24}$	$c_{31}c_{23}c_{22}c_{21}$	$c_{20} \cdots c_{10}$	$c_9 \cdots c_6$	$c_5 \cdots c_0$
Init1	xxxx	1010	00xx	1x01xx001x0	0000	000000
Init2	xxxx	1010	00xx	0xx1xx011x1	xxxx	xxxxxx
Init3	xxxx	1xxx	xxxx	0xx1xx110xx	xxxx	xxxxxx
D1	xxxx	001x	00x0	1x01xx100x0	1000	001001
D2	xxxx	000x	x10x	0xx111100xx	[illegible]	000010
D3	xxxx	00x1	0100	101010001x0	1100	100100
D4	xxxx	000x	00x1	010110000xx	xx11	000000
A1	xxxx	000x	0001	0x0101000xx	xx01	001000
A2	xxxx	00x1	0010	0x00xx00110	00xx	000010
A3	xxxx	00xx	00x0	00x1xx100x0	xx00	101000
A4	xxxx	00x0	0000	0100xx101x0	xx00	010001
A5	xxxx	00x1	0100	1x1010001x0	0100	000010
A6	xxxx	001x	0110	1x011010010	0010	001010
A7	xxxx	000x	0011	0x01010001x	xx00	001011
A8	xxxx	000x	0001	010110000xx	xx01	011000

So, let us try to see at least how to read the control logic ok; I will not go into too more details about that, but rather leave to you as an exercise. But you can see that pretty much you essentially have got all the components which are required. So, let us explain the control unit which is the essentially the you know the centered stone of the architecture. So, the control unit has got 4 distinct parts; so that is showed by this FSM or Finite State Machine

So, you can see that I have got at we have drawn like the use in the green colour; we have written the initialization portion where you load the initial curve constants and the initial base point. Followed by the operation which is shown by the blue circles and the doubling is done always.

But that either of 4 clock cycles we are require for doubling because of the reason that I said. And subsequently right you have got 8 clock cycles required for doing the addition operation and the addition is done conditionally in this architecture. Later on we will see that is not a very secured way of implementing things for at least from performance point of view this looks fine. And then that is the final step of inversion where you get the final result; you bring the result back from the projective to the affine coordinates and also get the result.

So, therefore this is shown here in the brown line and there are transfer to be 24 clock cycles which are required when you are doing Ethos Uzi inversion algorithm or implying the ethos uzi inversion algorithm as we have studied previously. So, here snapshot of the of the control logic; you can see that there are 3 initial blocks cycles, there are 4 you know like the doubling steps and there are like 8 addition steps ok.

So, let us see how the control logic is decided; so with this example. So, for this you have to see or keep in perspective the entire data path. So, here we show first of all in this cycle we in this the architecture; we show the register file and this is the arithmetic unique ok. And then these are the corresponding control words and also the corresponding logic for doing the computations ok.

So, let us try to take some examples like for example, the initialization block ok. If you take the initialization block and in that case you will see and I am in particular I am concentrating on the second register file. So, you will see that the second register file is stored or storing the essentially the; is processing on the Y coordinates ok. So, so if you observe the steps here then you get RB 1 equal to P Y; in the second clock cycle you do RB 2 equal to P Y ok. So, therefore, that means, you are here you are writing into RB 1, here you are writing into RB 2 ok.

So, now if you see the logic here; that means, when you are wanting to write into RB 1. So, in when you are say as I said that the first address or address 1 is used to choose the write address and address 2 is used to choose the read address ok. So, address 1 is essentially control by C 14 and C 13 and the read and write is stored by C 16 and C 16 and C 15.

So, that is shown here by this small think and if you can observe here; then you can see that the corresponding control initially write is essentially C 14 and C 13 and that shows 0 0 ok. If it shows 0 0; that means, you are basically writing into the first register here that is RB 1 and that is exactly what you are doing in the equation.

In the second clock cycle you are writing into RB 2 and that is; that means, that this control should be changed into 0 1 and that is what is done here ok. And in the third step you are basically writing into RB 4; so RB 4 should be 1 1 ok. So, that is what is done here is 1 1, but in all these things right you can see that the read address is do not care because you are not reading anything.

Likewise, if you see the doubling the first clock cycle for doubling; so in this case right when you are doubling then the operation which were doing is R A 1 square into RC 1 square ok. So; that means, right the data has to be passed into the multiplier and; that means, these select lines will become functional ok. So; that means, now if you see right I am basically taking the input from R A 1; so, that is essentially stored here.

So, what I do is; I basically write and this is the corresponding control for MUX A and MUX B. So, these are the 2 input multiplexers to the multiplier circuit. So, here I need to choose therefore, if I want to multiply this I have to choose the corresponding. So, these are the responding output results which have being generated from the arithmetic unit ok.

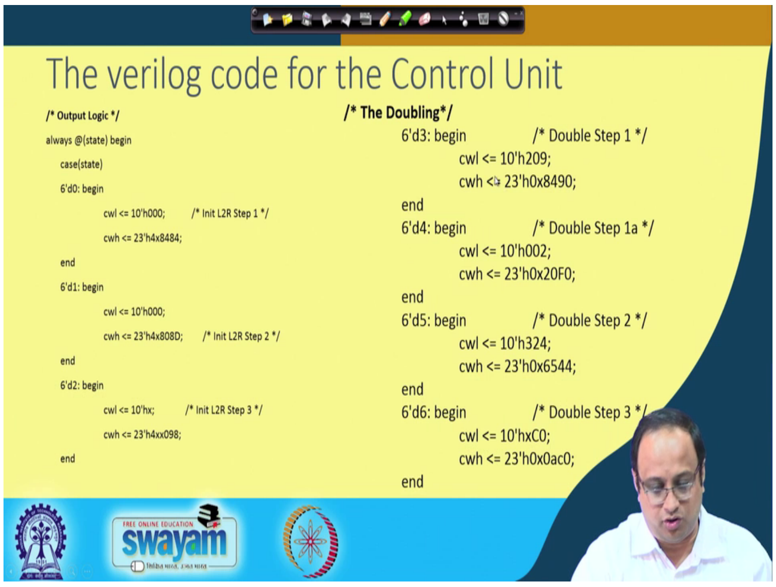
So, now you can see that what I want to multiply is R A 1 square into RC 1 square and that is here essentially you are A 0 square and A 1 square ok. So, therefore, you will see that the control logic here that is the MUX A and MUX B their select lines are 001 and 001. So, 001 means the second line of the multiplex of the inputs ok. So, you are basically selecting if you can see the input is A 0 square and here it is A 1 square; so; that means, A 0 is coming and A 1 is coming, they are getting squared by the square circuit in the arithmetic unit and I am passing A 0 square and A 1 square to the multiplier.

So, the multiplier multiplies them and the result is ready in the bus M ok. So, this is the corresponding bus M; where the result is stored. So, if you can observe right that this is your final result where you are getting the output.

So, this output needs to be multiplex out. So, therefore, right; that means, this select line should be set to 00 at that point ok. And that is precisely which is which is being done here if you observe then; you will see that C 6 C 6 and C 7 and C 8 and C 9; that means, these are the corresponding control logic. So, these 2 bits write are 0 0 here which means that the multiplier is getting passed. So; that means, this output is getting computed and getting passed out is getting is basically passed in C 0 bus; the other output is RC 1 to the power of 4 which you need to pass 2 C 1 that is C 1 is this bus ok.

So, if you see the inputs here are out of them right A 1 to the power of 4; this input is what I need basically and therefore, I need to pass this out as an output. So, therefore, write 01 and 2; so therefore, this line needs to be made 1 0 and that is essentially done by this control where you make it 1 and 0 ok. So, this kind of explains how you basically set the control. So, that you basically configure and all these steps essentially are done in a similar fashion which I leave it to you an exercise; you can verify each of these steps and you can try to understand their correlations.

(Refer Slide Time: 40:41)



The verilog code for the Control Unit

```
/* Output Logic */
always @(state) begin
    case(state)
    6'd0: begin
            cwl <= 10'h000;      /* Init L2R Step 1 */
            cwh <= 23'h4x8484;
    end
    6'd1: begin
            cwl <= 10'h000;
            cwh <= 23'h4x808D;     /* Init L2R Step 2 */
    end
    6'd2: begin
            cwl <= 10'hx;        /* Init L2R Step 3 */
            cwh <= 23'h4xx098;
    end
```

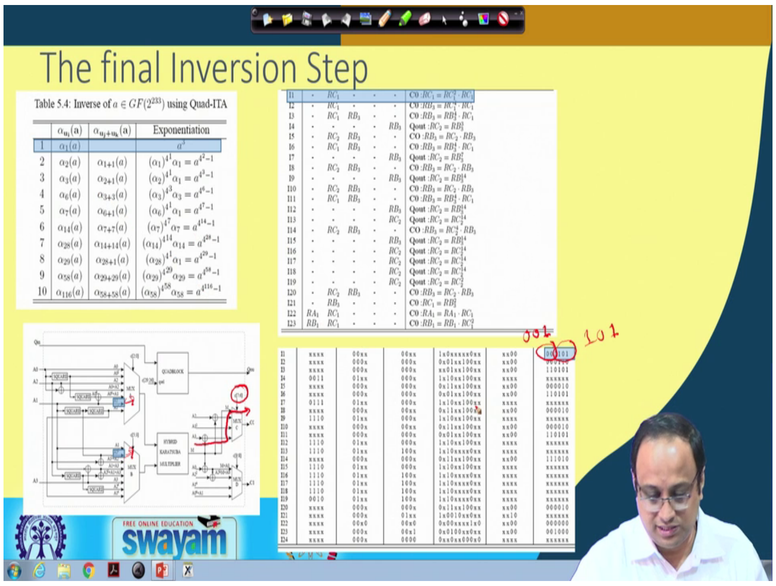
```
/* The Doubling*/
    6'd3: begin              /* Double Step 1 */
            cwl <= 10'h209;
            cwh <= 23'h0x8490;
    end
    6'd4: begin              /* Double Step 1a */
            cwl <= 10'h002;
            cwh <= 23'h0x20F0;
    end
    6'd5: begin              /* Double Step 2 */
            cwl <= 10'h324;
            cwh <= 23'h0x6544;
    end
    6'd6: begin              /* Double Step 3 */
            cwl <= 10'hxC0;
            cwh <= 23'h0x0ac0;
    end
```

So, with this background right we now can go into are just at I will look into the verilog code without going into too much details. So, this is nothing, but a depiction of how the controller works in. So, you can see that this is the output logic because the control are essentially generates an output which is nothing, but the control signal.

Now that depends upon 2 things; it depends upon typically on the state. So, essentially you basically of the state; so you have basically encoded all the steps in states and depending upon the state you produce and control word ok. And control word is nothing, but the binary value of these 1 0 configurations ok. So, that is essentially stored by this funny looking x characters ok; this is nothing, but they are binary representations.

So, likewise you essentially can write for all of them like this is these are the initial parts; these are the doubling parts and this is the corresponding addition states. So, all of them their control logic has been created or model in the verilog code.

(Refer Slide Time: 41:37)



The final Inversion Step

Table 5.4: Inverse of $a \in GF(2^{233})$ using Quad-ITA

	$\alpha_{u_i}(a)$	$\alpha_{u_j+u_k}(a)$	Exponentiation
1	$\alpha_1(a)$		a^3
2	$\alpha_2(a)$	$\alpha_{1+1}(a)$	$(\alpha_1)^{4^1}\alpha_1 = a^{4^2-1}$
3	$\alpha_3(a)$	$\alpha_{2+1}(a)$	$(\alpha_2)^{4^1}\alpha_1 = a^{4^3-1}$
4	$\alpha_6(a)$	$\alpha_{3+3}(a)$	$(\alpha_3)^{4^3}\alpha_3 = a^{4^6-1}$
5	$\alpha_7(a)$	$\alpha_{6+1}(a)$	$(\alpha_6)^{4^1}\alpha_1 = a^{4^7-1}$
6	$\alpha_{14}(a)$	$\alpha_{7+7}(a)$	$(\alpha_7)^{4^7}\alpha_7 = a^{4^{14}-1}$
7	$\alpha_{28}(a)$	$\alpha_{14+14}(a)$	$(\alpha_{14})^{4^{14}}\alpha_{14} = a^{4^{28}-1}$
8	$\alpha_{29}(a)$	$\alpha_{28+1}(a)$	$(\alpha_{28})^{4^1}\alpha_1 = a^{4^{29}-1}$
9	$\alpha_{58}(a)$	$\alpha_{29+29}(a)$	$(\alpha_{29})^{4^{29}}\alpha_{29} = a^{4^{58}-1}$
10	$\alpha_{116}(a)$	$\alpha_{58+58}(a)$	$(\alpha_{58})^{4^{58}}\alpha_{58} = a^{4^{116}-1}$

I1	·	RC_1	·	·	·	$C0 : RC_1 = RC_1^2 \cdot RC_1$
I2	·	RC_1	·	·	·	$C0 : RB_3 = RC_1^4 \cdot RC_1$
I3	·	RC_1	RB_3	·	·	$C0 : RB_3 = RB_3^4 \cdot RC_1$
I4	·	·	·	·	RB_3	$Qout : RC_2 = RB_3^3$
I5	·	RC_2	RB_3	·	·	$C0 : RB_3 = RC_2 \cdot RB_3$
I6	·	RC_1	RB_3	·	·	$C0 : RB_3 = RB_3^4 \cdot RC_1$
I7	·	·	·	·	RB_3	$Qout : RC_2 = RB_3^7$
I8	·	RC_2	RB_3	·	·	$C0 : RB_3 = RC_2 \cdot RB_3$
I9	·	·	·	·	RB_3	$Qout : RC_2 = RB_3^{14}$
I10	·	RC_2	RB_3	·	·	$C0 : RB_3 = RC_2 \cdot RB_3$
I11	·	RC_1	RB_3	·	·	$C0 : RB_3 = RB_3^4 \cdot RC_1$
I12	·	·	·	·	RB_3	$Qout : RC_2 = RB_3^{14}$
I13	·	·	·	·	RC_2	$Qout : RC_2 = RC_2^{14}$
I14	·	RC_2	RB_3	·	·	$C0 : RB_3 = RC_2^4 \cdot RB_3$
I15	·	·	·	·	RB_3	$Qout : RC_2 = RB_3^{14}$
I16	·	·	·	·	RC_2	$Qout : RC_2 = RC_2^{14}$
I17	·	·	·	·	RC_2	$Qout : RC_2 = RC_2^{14}$
I18	·	·	·	·	RC_2	$Qout : RC_2 = RC_2^{14}$
I19	·	·	·	·	RC_2	$Qout : RC_2 = RC_2^2$
I20	·	RC_2	RB_3	·	·	$C0 : RB_3 = RC_2 \cdot RB_3$
I21	·	RB_3	·	·	·	$C0 : RC_1 = RB_3^2$
I22	RA_1	RC_1	·	·	·	$C0 : RA_1 = RA_1 \cdot RC_1$
I23	RB_1	RC_1	·	·	·	$C0 : RB_1 = RB_1 \cdot RC_1^2$

001 101

I1	xxxx	00xx	00xx	1x0xxxxx0xx	xx00	000101
I2	xxxx	000x	000x	0x01xx100xx	xx00	000110
I3	xxxx	000x	000x	xx01xx100xx	xx00	110101
I4	0011	01xx	000x	1x10xx100xx	xxxx	xxxxxx
I5	xxxx	000x	000x	0x11xx100xx	xx00	000010
I6	xxxx	000x	000x	0x01xx100xx	xx00	110101
I7	0111	01xx	000x	1x10xx100xx	xxxx	xxxxxx
I8	xxxx	000x	00xx	0x11xx100xx	xx00	000010
I9	1110	01xx	000x	1x10xx100xx	xxxx	xxxxxx
I10	xxxx	000x	00xx	0x11xx100xx	xx00	000010
I11	xxxx	000x	000x	0x01xx100xx	xx00	110101
I12	1110	01xx	000x	1x10xx100xx	xxxx	xxxxxx
I13	1110	01xx	100x	1x10xxxx0xx	xxxx	xxxxxx
I14	xxxx	000x	000x	0x11xx100xx	xx00	111010
I15	1110	01xx	000x	1x10xx100xx	xxxx	xxxxxx
I16	1110	01xx	100x	1x10xxxx0xx	xxxx	xxxxxx
I17	1110	01xx	100x	1x10xxxx0xx	xxxx	xxxxxx
I18	1110	01xx	100x	1x10xxxx0xx	xxxx	xxxxxx
I19	0010	01xx	100x	1x10xxxx0xx	xxxx	xxxxxx
I20	xxxx	000x	000x	0x11xx100xx	xx00	000010
I21	xxxx	000x	01xx	1x0010xx0xx	xx10	xxxxxx
I22	xxxx	00x0	00x0	0x00xxxx1x0	xx00	000000
I23	xxxx	000x	00x1	0x0100xx0xx	xx00	001000
I24	xxxx	000x	0000	0xx0xx000x0	xxxx	xxxxxx

The finally, finally, we have an inversion step because after you have done doubling addition and the entire thing is ready; you need to do the inversion to get back the fine results ok. So, I will just quickly recapitulate you about that Ethos Uzi inversion algorithm using the quad logic which we studied previously.

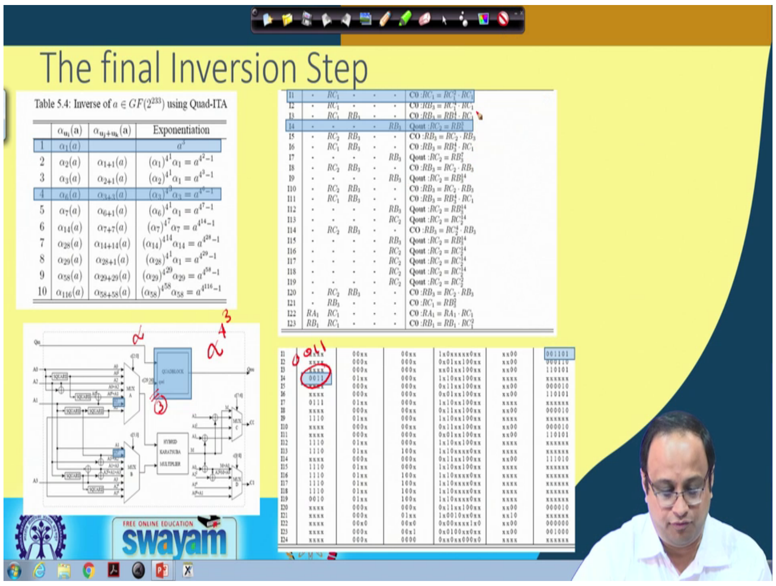
So, this is the quick flowchart about how this computation works in and if you remember like I will just show you like one part. For example, this is the initial state stay where you need to do A cube ok. So, in A cube basically I take the input, but remember I do not have any cubing circuit, but I have got a squaring circuit. So, what I do is I take A 1 I pass it just as it is here; I take A 1 and now I square it in pass it here A 1 square. I now use this hybrid Karatsuba multiplier to do this multiplication ok.

So, therefore, right when I am multiplying this I have to choose the fifth block; fifth input here and I have to again count 1, 2, 3, 4, and like so on; 0, 1, 2, 3, 4 and 5; so that is the fifth inputs. So, it should be 1 0 1 and that is essentially exactly what is here.

So, therefore, if you observe this in this part it is 1 0 1 and the first part is 0 0 1; which means it is the second input from here. So, you choosing the fifth input here and the second input here and they are being multiplied by the multiplier. So, therefore you need to again pass it through the output by accordingly choosing this control logic.

So, that is exactly how these operations are done and finally, you get your result. So, you can essentially perform all the successive steps in a similar fashion and get the entire computation being done.

(Refer Slide Time: 43:11)



The final Inversion Step

Table 5.4: Inverse of $a \in GF(2^{233})$ using Quad-ITA

	$\alpha_{u_i}(a)$	$\alpha_{u_j+u_k}(a)$	Exponentiation
1	$\alpha_1(a)$		a^3
2	$\alpha_2(a)$	$\alpha_{1+1}(a)$	$(\alpha_1)^{4^1}\alpha_1 = a^{4^2-1}$
3	$\alpha_3(a)$	$\alpha_{2+1}(a)$	$(\alpha_2)^{4^1}\alpha_1 = a^{4^3-1}$
4	$\alpha_6(a)$	$\alpha_{3+3}(a)$	$(\alpha_3)^{4^3}\alpha_3 = a^{4^6-1}$
5	$\alpha_7(a)$	$\alpha_{6+1}(a)$	$(\alpha_6)^{4^1}\alpha_1 = a^{4^7-1}$
6	$\alpha_{14}(a)$	$\alpha_{7+7}(a)$	$(\alpha_7)^{4^7}\alpha_7 = a^{4^{14}-1}$
7	$\alpha_{28}(a)$	$\alpha_{14+14}(a)$	$(\alpha_{14})^{4^{14}}\alpha_{14} = a^{4^{28}-1}$
8	$\alpha_{29}(a)$	$\alpha_{28+1}(a)$	$(\alpha_{28})^{4^1}\alpha_1 = a^{4^{29}-1}$
9	$\alpha_{58}(a)$	$\alpha_{29+29}(a)$	$(\alpha_{29})^{4^{29}}\alpha_{29} = a^{4^{58}-1}$
10	$\alpha_{116}(a)$	$\alpha_{58+58}(a)$	$(\alpha_{58})^{4^{58}}\alpha_{58} = a^{4^{116}-1}$

I1	·	RC_1	·	·	·	$C0: RC_1 = RC_1^2 \cdot RC_1$
I2	·	RC_1	·	·	·	$C0: RB_3 = RC_1^4 \cdot RC_1$
I3	·	RC_1	RB_3	·	·	$C0: RB_3 = RB_3^4 \cdot RC_1$
I4	·	·	·	·	RB_3	$Qout: RC_2 = RB_3^3$
I5	·	RC_2	RB_3	·	·	$C0: RB_3 = RC_2 \cdot RB_3$
I6	·	RC_1	RB_3	·	·	$C0: RB_3 = RB_3^4 \cdot RC_1$
I7	·	·	·	·	RB_3	$Qout: RC_2 = RB_3^7$
I8	·	RC_2	RB_3	·	·	$C0: RB_3 = RC_2 \cdot RB_3$
I9	·	·	·	·	RB_3	$Qout: RC_2 = RB_3^{14}$
I10	·	RC_2	RB_3	·	·	$C0: RB_3 = RC_2 \cdot RB_3$
I11	·	RC_1	RB_3	·	·	$C0: RB_3 = RB_3^4 \cdot RC_1$
I12	·	·	·	·	RB_3	$Qout: RC_2 = RB_3^{14}$
I13	·	·	·	·	RC_2	$Qout: RC_2 = RC_2^{14}$
I14	·	RC_2	RB_3	·	·	$C0: RB_3 = RC_2^4 \cdot RB_3$
I15	·	·	·	·	RB_3	$Qout: RC_2 = RB_3^{14}$
I16	·	·	·	·	RC_2	$Qout: RC_2 = RC_2^{14}$
I17	·	·	·	·	RC_2	$Qout: RC_2 = RC_2^{14}$
I18	·	·	·	·	RC_2	$Qout: RC_2 = RC_2^{14}$
I19	·	·	·	·	RC_2	$Qout: RC_2 = RC_2^2$
I20	·	RC_2	RB_3	·	·	$C0: RB_3 = RC_2 \cdot RB_3$
I21	·	RB_3	·	·	·	$C0: RC_1 = RB_3^2$
I22	RA_1	RC_1	·	·	·	$C0: RA_1 = RA_1 \cdot RC_1$
I23	RB_1	RC_1	·	·	·	$C0: RB_1 = RB_1 \cdot RC_1^2$

0 011

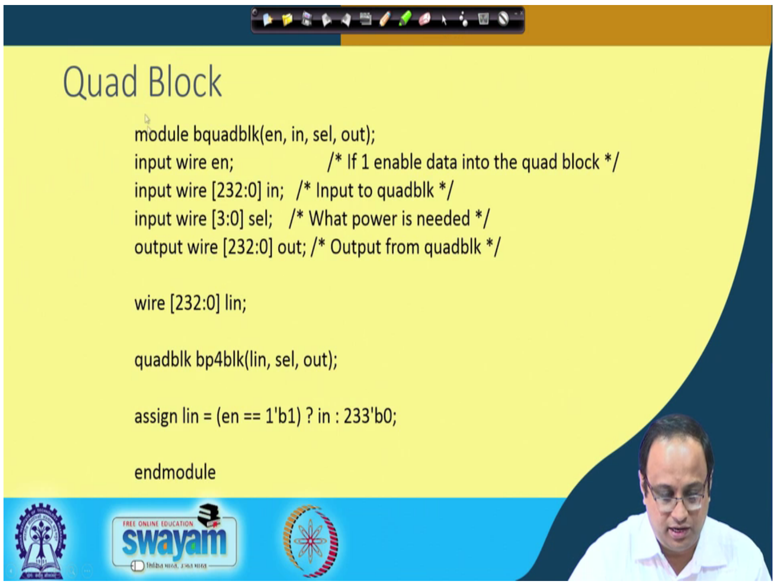
I1	xxxx	00xx	00xx	1x0xxxxx0xx	xx00	001101
I2	xxxx	000x	000x	0x01xx100xx	xx00	000110
I3	xxxx	000x	000x	xx01xx100xx	xx00	110101
I4	0011	01xx	000x	1x10xx100xx	xxxx	xxxxxx
I5	xxxx	000x	000x	0x11xx100xx	xx00	000010
I6	xxxx	000x	000x	0x01xx100xx	xx00	110101
I7	0111	01xx	000x	1x10xx100xx	xxxx	xxxxxx
I8	xxxx	000x	00xx	0x11xx100xx	xx00	000010
I9	1110	01xx	000x	1x10xx100xx	xxxx	xxxxxx
I10	xxxx	000x	00xx	0x11xx100xx	xx00	000010
I11	xxxx	000x	000x	0x01xx100xx	xx00	110101
I12	1110	01xx	000x	1x10xx100xx	xxxx	xxxxxx
I13	1110	01xx	100x	1x10xxxx0xx	xxxx	xxxxxx
I14	xxxx	000x	000x	0x11xx100xx	xx00	111010
I15	1110	01xx	000x	1x10xx100xx	xxxx	xxxxxx
I16	1110	01xx	100x	1x10xxxx0xx	xxxx	xxxxxx
I17	1110	01xx	100x	1x10xxxx0xx	xxxx	xxxxxx
I18	1110	01xx	100x	1x10xxxx0xx	xxxx	xxxxxx
I19	0010	01xx	100x	1x10xxxx0xx	xxxx	xxxxxx
I20	xxxx	000x	000x	0x11xx100xx	xx00	000010
I21	xxxx	000x	01xx	1x0010xx0xx	xx10	xxxxxx
I22	xxxx	00x0	00x0	0x00xxxx1x0	xx00	000000
I23	xxxx	000x	00x1	0x0100xx0xx	xx00	001000
I24	xxxx	000x	0000	0xx0xx000x0	xxxx	xxxxxx

So, likewise there is another step which we have shown here it is the final quading step; where you see that in one step I do I raise something to the power of 4 to the power of 3 ok.

So, if you remember the quad block in the quad block there is a select input ok; the idea is that if the select input is made 3; that means, if I made these 3 and if there is alpha this output is alpha to the power of 4 power of 3 ok. So, therefore, right here what we do is exactly that. So, you see the select value the select values is made 3 it is 0 0 1 and 1. So, it becomes 3 and therefore, you calculate alpha to the power of 4 power of 3 ok.

So therefore, right I mean that explains all the individual components.

(Refer Slide Time: 43:55)

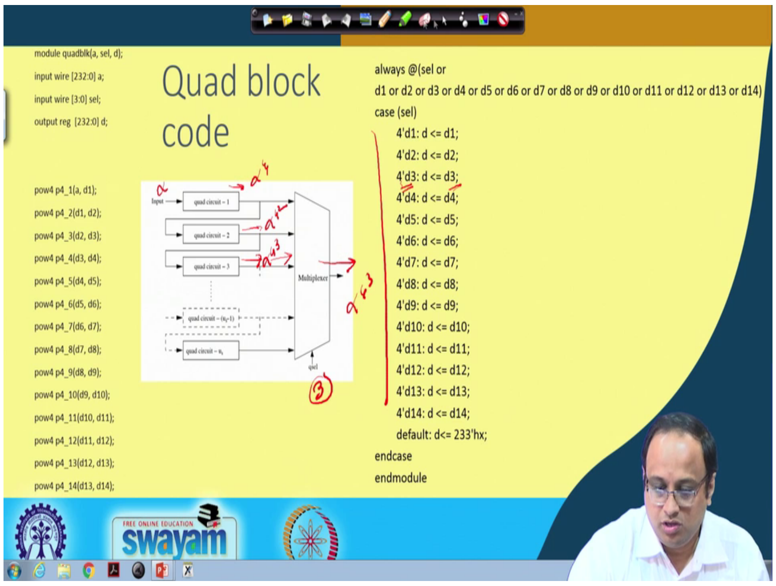


Quad Block

```
module bquadblk(en, in, sel, out);
input wire en;                /* If 1 enable data into the quad block */
input wire [232:0] in;  /* Input to quadblk */
input wire [3:0] sel;   /* What power is needed */
output wire [232:0] out; /* Output from quadblk */

wire [232:0] lin;

quadblk bp4blk(lin, sel, out);

assign lin = (en == 1'b1) ? in : 233'b0;

endmodule
```

And finally, right this is an example of how you can of the verilog program or the verilog model for the quad block. So, you see that there is the select line and depending upon the select line you have an internal quad block through which you calculate this.

(Refer Slide Time: 44:09)



Quad block code

```
module quadblk(a, sel, d);
input wire [232:0] a;
input wire [3:0] sel;
output reg  [232:0] d;


pow4 p4_1(a, d1);
pow4 p4_2(d1, d2);
pow4 p4_3(d2, d3);
pow4 p4_4(d3, d4);
pow4 p4_5(d4, d5);
pow4 p4_6(d5, d6);
pow4 p4_7(d6, d7);
pow4 p4_8(d7, d8);
pow4 p4_9(d8, d9);
pow4 p4_10(d9, d10);
pow4 p4_11(d10, d11);
pow4 p4_12(d11, d12);
pow4 p4_13(d12, d13);
pow4 p4_14(d13, d14);
```

```
always @(sel or
d1 or d2 or d3 or d4 or d5 or d6 or d7 or d8 or d9 or d10 or d11 or d12 or d13 or d14)
case (sel)
    4'd1: d <= d1;
    4'd2: d <= d2;
    4'd3: d <= d3;
    4'd4: d <= d4;
    4'd5: d <= d5;
    4'd6: d <= d6;
    4'd7: d <= d7;
    4'd8: d <= d8;
    4'd9: d <= d9;
    4'd10: d <= d10;
    4'd11: d <= d11;
    4'd12: d <= d12;
    4'd13: d <= d13;
    4'd14: d <= d14;
    default: d<= 233'hx;
endcase
endmodule
```

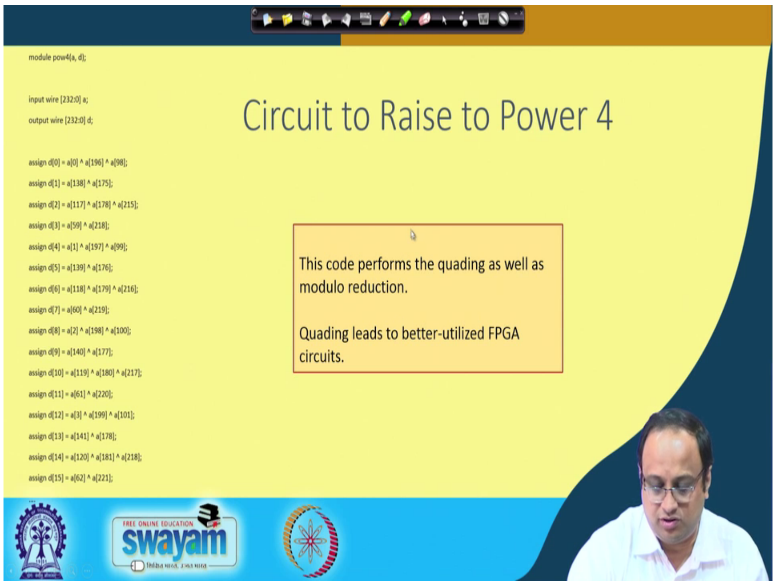
You can verify later on and if you see the quad block; just remember the architecture. So, in this the verilog model we just place these quad circuits one after the other. If you remember that we had 14 cascade stages; so these are the 14 cascade power blocks which have been placed.

All of them calculates power to the power of 4 ok; depending upon the select you basically multiplex any one of them out. For example, if I want to you know like select something intermediate then I can just choose one thing from the; for example, if I want to calculate alpha to the power of 4 power of 3 right then; that means, that I have to basically you know like take this; so this, so this is my alpha this becomes alpha to the power of 4.

This becomes alpha to the power of 4 square, this becomes alpha to the power of 4 power of 3 and now I make this 3; so therefore, these gets selected out. So, I becomes alpha to the power of 4 power of 3. So, now that exactly is essentially written here and this is the final selection module; so which says that how you are essentially selecting.

So, therefore, if your select line is 3 then you basically select this line; which you can implement easily through a multiplexer routine. So, finally, right I mean that essentially explains pretty much the logic and essentially right.

(Refer Slide Time: 45:27)



```
module pow4(a, d);

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

assign d[0] = a[0] ^ a[196] ^ a[98];
assign d[1] = a[138] ^ a[175];
assign d[2] = a[117] ^ a[178] ^ a[215];
assign d[3] = a[59] ^ a[218];
assign d[4] = a[1] ^ a[197] ^ a[99];
assign d[5] = a[139] ^ a[176];
assign d[6] = a[118] ^ a[179] ^ a[216];
assign d[7] = a[60] ^ a[219];
assign d[8] = a[2] ^ a[198] ^ a[100];
assign d[9] = a[140] ^ a[177];
assign d[10] = a[119] ^ a[180] ^ a[217];
assign d[11] = a[61] ^ a[220];
assign d[12] = a[3] ^ a[199] ^ a[101];
assign d[13] = a[141] ^ a[178];
assign d[14] = a[120] ^ a[181] ^ a[218];
assign d[15] = a[62] ^ a[221];
```

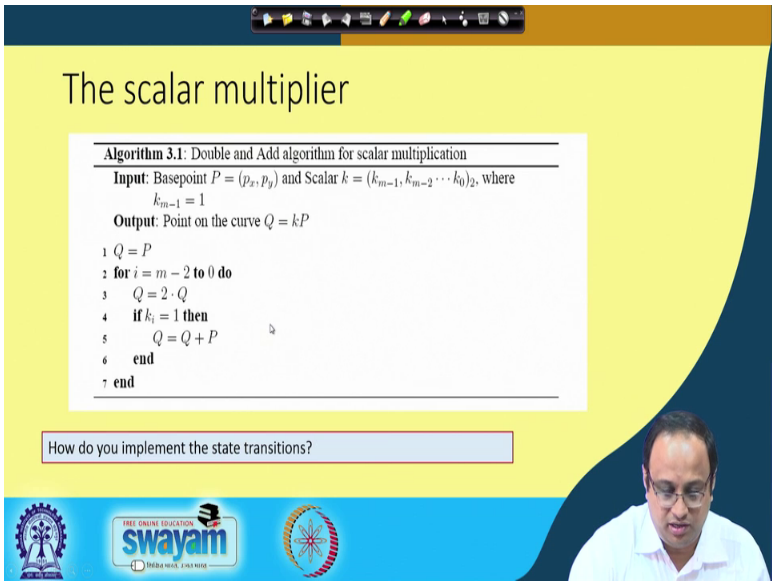
Circuit to Raise to Power 4

This code performs the quading as well as modulo reduction.

Quading leads to better-utilized FPGA circuits.

Finally this is your power 4 circuit which is exactly analogous to your squaring routine. It also does the modular reduction in one go inside the inside this code ok. And you can also observe that the entire architecture just have exhorts because power of 4 in characteristic 2 field is a linear operation. So, essentially you do not need any explicit AND gets you can do the entire operation just by using exclusive powers.

(Refer Slide Time: 45:51)



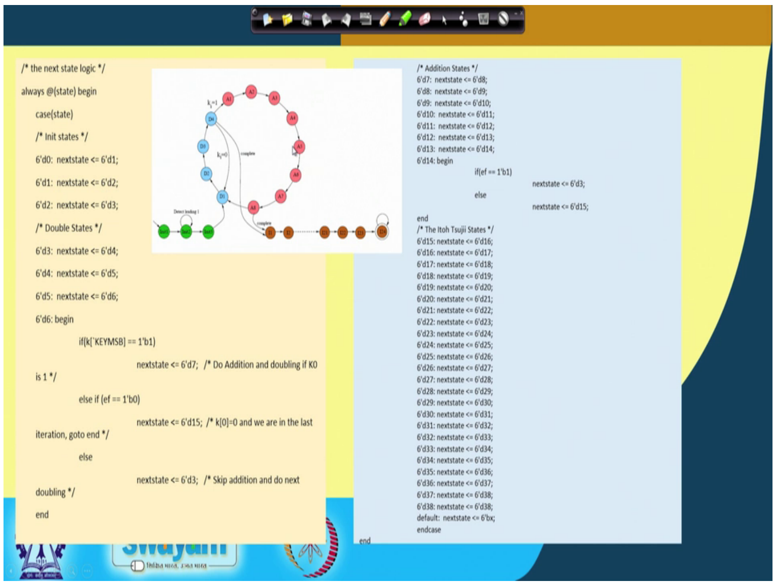
The scalar multiplier

```
Algorithm 3.1: Double and Add algorithm for scalar multiplication
  Input: Basepoint $P = (p_x, p_y)$ and Scalar $k = (k_{m-1}, k_{m-2} \cdots k_0)_2$, where
         $k_{m-1} = 1$
  Output: Point on the curve $Q = kP$
1 $Q = P$
2 for $i = m-2$ to $0$ do
3     $Q = 2 \cdot Q$
4     if $k_i = 1$ then
5         $Q = Q + P$
6     end
7 end
```

How do you implement the state transitions?

So, finally, you have got the scalar multiplier. So, in the scalar multiplier you have got these essentially is the recapitulation of your double an add algorithm.

(Refer Slide Time: 45:59)



```
/* the next state logic */
always @(state) begin
    case(state)
    /* Init states */
    6'd0:  nextstate <= 6'd1;
    6'd1:  nextstate <= 6'd2;
    6'd2:  nextstate <= 6'd3;
    /* Double States */
    6'd3:  nextstate <= 6'd4;
    6'd4:  nextstate <= 6'd5;
    6'd5:  nextstate <= 6'd6;
    6'd6: begin
              if(k['KEYMSB] == 1'b1)
                        nextstate <= 6'd7;   /* Do Addition and doubling if K0
    is 1 */
              else if (ef == 1'b0)
                        nextstate <= 6'd15;  /* k[0]=0 and we are in the last
    iteration, goto end */
              else
                        nextstate <= 6'd3;   /* Skip addition and do next
    doubling */
    end
```

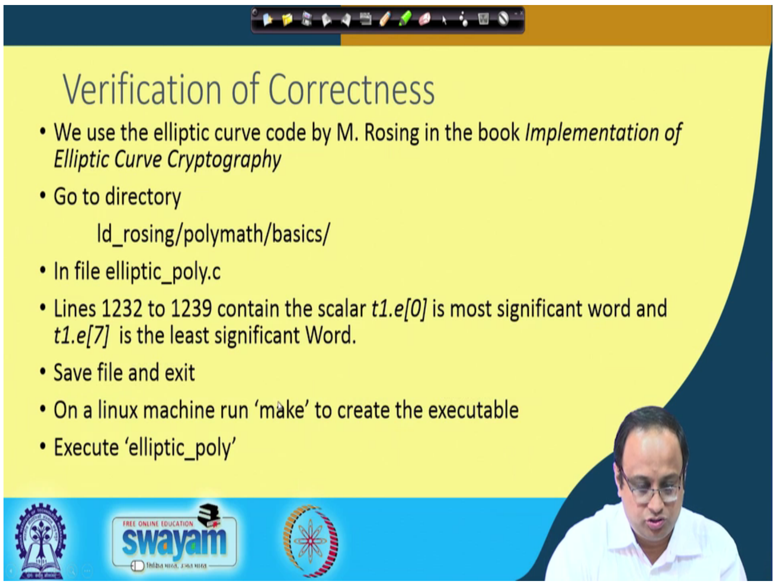
```
/* Addition States */
6'd7:  nextstate <= 6'd8;
6'd8:  nextstate <= 6'd9;
6'd9:  nextstate <= 6'd10;
6'd10:  nextstate <= 6'd11;
6'd11:  nextstate <= 6'd12;
6'd12:  nextstate <= 6'd13;
6'd13:  nextstate <= 6'd14;
6'd14: begin
            if(ef == 1'b1)
                        nextstate <= 6'd3;
            else
                        nextstate <= 6'd15;
end
/* The Itoh Tsujii States */
6'd15: nextstate <= 6'd16;
6'd16: nextstate <= 6'd17;
6'd17: nextstate <= 6'd18;
6'd18: nextstate <= 6'd19;
6'd19: nextstate <= 6'd20;
6'd20: nextstate <= 6'd21;
6'd21: nextstate <= 6'd22;
6'd22: nextstate <= 6'd23;
6'd23: nextstate <= 6'd24;
6'd24: nextstate <= 6'd25;
6'd25: nextstate <= 6'd26;
6'd26: nextstate <= 6'd27;
6'd27: nextstate <= 6'd28;
6'd28: nextstate <= 6'd29;
6'd29: nextstate <= 6'd30;
6'd30: nextstate <= 6'd31;
6'd31: nextstate <= 6'd32;
6'd32: nextstate <= 6'd33;
6'd33: nextstate <= 6'd34;
6'd34: nextstate <= 6'd35;
6'd35: nextstate <= 6'd36;
6'd36: nextstate <= 6'd37;
6'd37: nextstate <= 6'd38;
6'd38: nextstate <= 6'd38;
default:  nextstate <= 6'bx;
endcase
end
```

So, how do you implement the state transitions? Again this is the recapitulation of your final state machine. So, you can observe that the state transition that is the next state logic can be easily observed or written in verilog by this always encapsulation.

So, what you can do is that depending upon your current state; you essentially depend determine your next state ok. So, in all these entire thing it is pretty much that it goes into the next state depending upon your current state; except for this key logic where you have a jump ok. So, that essentially if this part where you have a conditional key statement and depending upon that from this clock cycle; I mean you either you know like you know you do a doubling, but are you follow a doubling with a doubling are you going to the addition step basically.

So, you either go from here to here or you go from here to here and that depends upon your key bit.

(Refer Slide Time: 46:51)



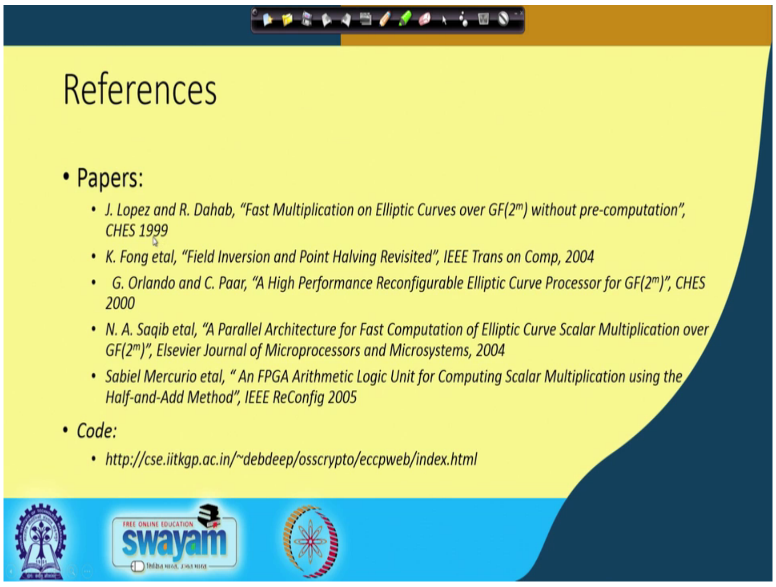
Verification of Correctness

- We use the elliptic curve code by M. Rosing in the book *Implementation of Elliptic Curve Cryptography*
- Go to directory
  ld_rosing/polymath/basics/
- In file elliptic_poly.c
- Lines 1232 to 1239 contain the scalar *t1.e[0]* is most significant word and *t1.e[7]* is the least significant Word.
- Save file and exit
- On a linux machine run 'make' to create the executable
- Execute 'elliptic_poly'

So, so finally right when you make a complicated design you have to verify your design. So, it is very important that you check that your result is correct; you should basically verify it with different kind of implementation. So, typically we write a software program and then we verify that our circuit is correct with respect to the software program.

In this case you can use this reference that we used from by a book by M. Rosing; it is called the, is Implementation of Elliptic Curve Cryptography. And from there you can get sudden references and if you look into it you can get an elliptic curve code on different fields and you can verify your result with respect to it.

(Refer Slide Time: 47:27)

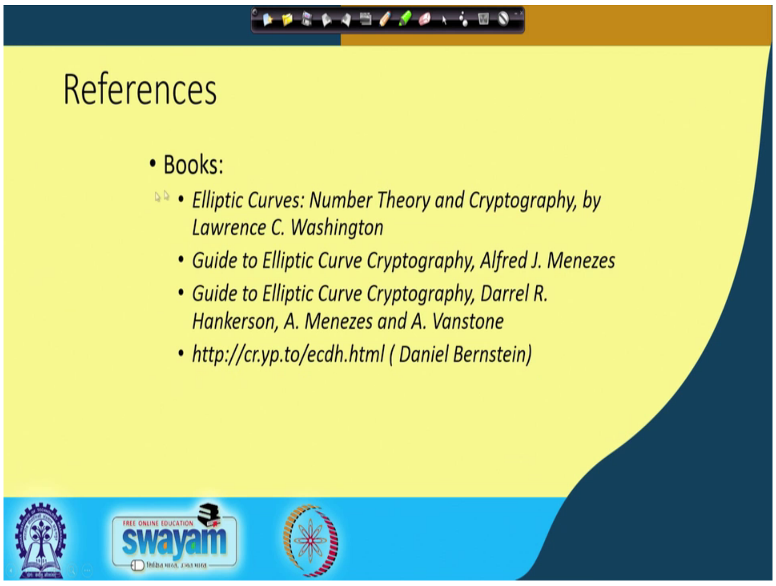


References

- Papers:
  - J. Lopez and R. Dahab, "Fast Multiplication on Elliptic Curves over GF(2^m) without pre-computation", CHES 1999
  - K. Fong etal, "Field Inversion and Point Halving Revisited", IEEE Trans on Comp, 2004
  - G. Orlando and C. Paar, "A High Performance Reconfigurable Elliptic Curve Processor for GF(2^m)", CHES 2000
  - N. A. Saqib etal, "A Parallel Architecture for Fast Computation of Elliptic Curve Scalar Multiplication over GF(2^m)", Elsevier Journal of Microprocessors and Microsystems, 2004
  - Sabiel Mercurio etal, " An FPGA Arithmetic Logic Unit for Computing Scalar Multiplication using the Half-and-Add Method", IEEE ReConfig 2005
- Code:
  - http://cse.iitkgp.ac.in/~debdeep/osscrypto/eccpweb/index.html

So, finally, right to conclude this part these are some of the references that we have used in particular this is a nice paper by Lopez and Dahab which was written in CHES in 1999. And there are some other interesting text also which I have kind of summarized here. And this entire program; that means, the entire verilog code for the elliptic curve processor; you can get in this link. So, you can just download this link and you know looking more details about the; you know like about what we probably could not discuss in the class.

(Refer Slide Time: 47:57)

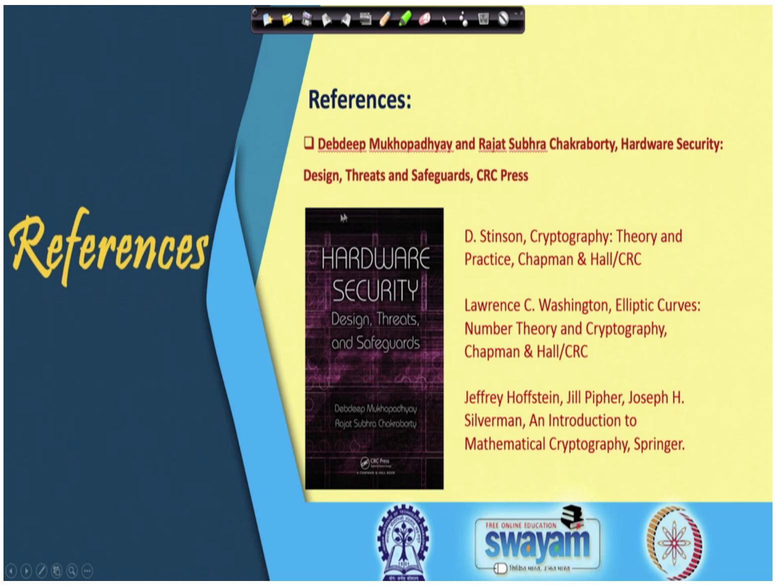


References

- Books:
  - Elliptic Curves: Number Theory and Cryptography, by Lawrence C. Washington
  - Guide to Elliptic Curve Cryptography, Alfred J. Menezes
  - Guide to Elliptic Curve Cryptography, Darrel R. Hankerson, A. Menezes and A. Vanstone
  - http://cr.yp.to/ecdh.html (Daniel Bernstein)

And these are again some of the references that we have used in particular; they are very nice book on elliptic curves. And this is also very standard textbooks on elliptic curve implementations an elliptic curve discussions.

(Refer Slide Time: 48:09)



References

References:

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

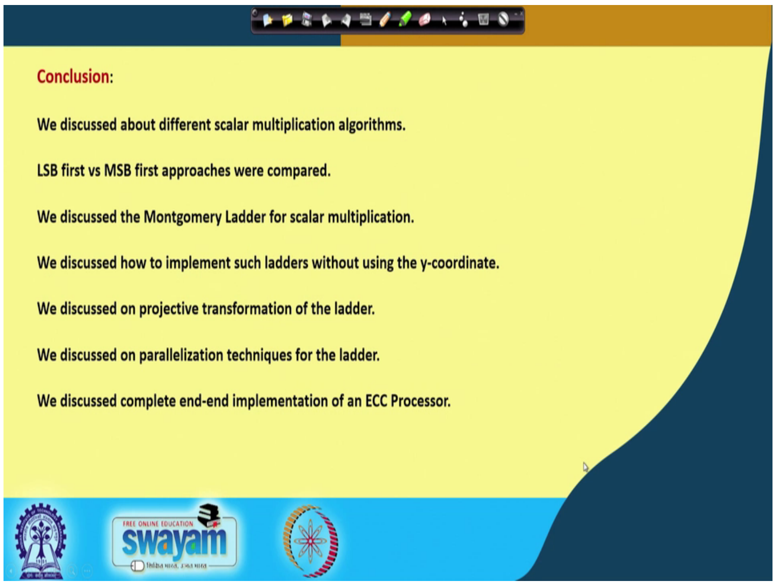
D. Stinson, Cryptography: Theory and Practice, Chapman & Hall/CRC

Lawrence C. Washington, Elliptic Curves: Number Theory and Cryptography, Chapman & Hall/CRC

Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman, An Introduction to Mathematical Cryptography, Springer.

You can also get into look into this book where this design is detailed in much more; you know like in much more content. And these are all some nice books on elliptic curves for example, this is also very nice book on elliptic curve which you can have a look into.

(Refer Slide Time: 48:27)



Conclusion:

We discussed about different scalar multiplication algorithms.

LSB first vs MSB first approaches were compared.

We discussed the Montgomery Ladder for scalar multiplication.

We discussed how to implement such ladders without using the y-coordinate.

We discussed on projective transformation of the ladder.

We discussed on parallelization techniques for the ladder.

We discussed complete end-end implementation of an ECC Processor.

So, what we into conclude; what we discussed is about difference scalar multiplication algorithms. We discussed about an LSB first verses MSB first approaches and we compared them. We discussed the Montgomery ladder for scalar multiplication; we discussed how to implement such ladders you without using the y coordinates explicitly.

We discussed on projective transformation on the ladder, we discussed on parallelization techniques for the ladder, we discussed complete end to end implementation of an ECC processor. So, that roughly takes us to the end of the; you know like discussions on elliptic curve hardware.

So, in the next class we shall be try to see that performance is not the only factor; there are many other things like security of implementations which we need to take care. So, that will bring us to the topic of what we call as side channel attacks which we will try to take of from the next class ok.

Thanks; thanks for your attention.