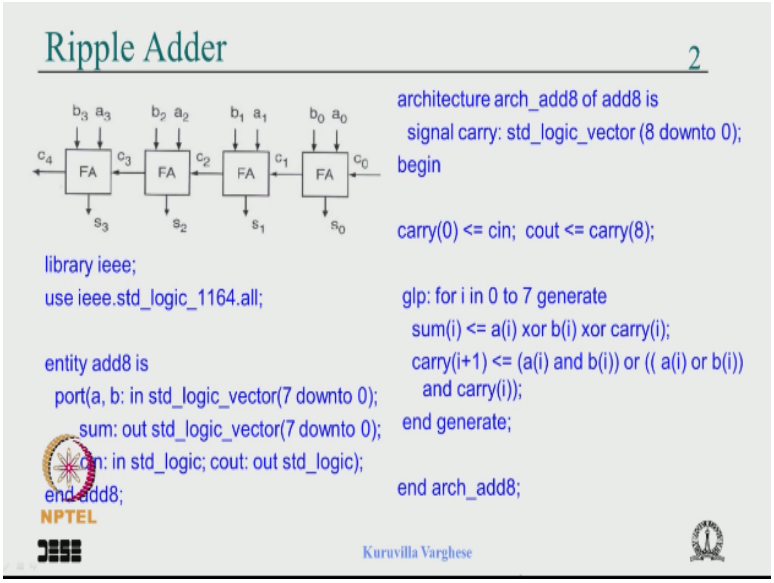
Digital Systems Design with PLDs and FPGAs
Kuruvilla Varghese
Department of Electronic Systems Engineering
Indian Institute of Science-Bangalore

Lecture-22
VHDL Examples, FSM Clock

So, welcome to this lecture on VHDL in the course digital system designed with PLDs and FPGAs. The last lecture we were looking at some examples of a given circuit how to write the code and given a VHDL code how to infer what is a circuit which could be synthesise from that code and so on. And we could not complete that, so I am planning to complete it and get to the digital systems designed for few lectures.

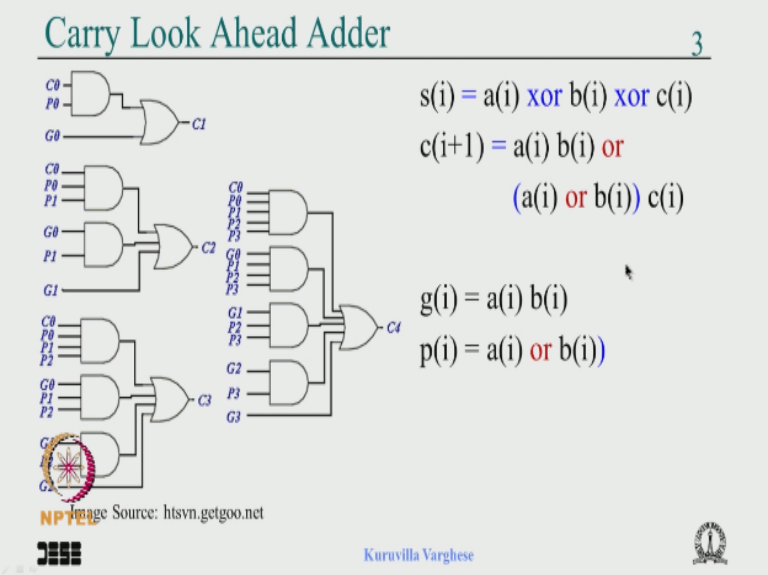
Before maybe we come back to the VHDL little bit again go to some other topics okay so, let us have a look at what we have done last class, last lecture so, last lecture we have looked at the ripple adder.

(Refer Slide Time: 01:12)



So, in the ripple adder it is simple you know you have full adder cascaded and the problem is delay and we said carry look ahead adder basically it is like you know writing the truth table and implement it, or working out from the full adder.

(Refer Slide Time: 01:31)

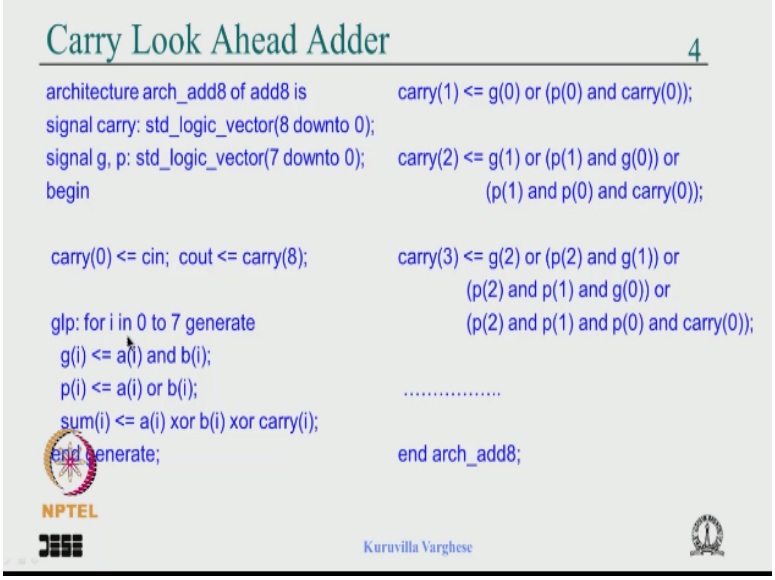


It means that you have this two equation one is for sum, one is for carry that trouble with the ripple adder is that the say ci + 1 is you know kind of derive from ai, bi and ci. Ci is you know coming of coming out of the previous stage so, it ripples you know it adds to the delay. So, what the carry look ahead adder does it that you take the and substitute here from the starting with the c0, like c1 is a0, b0 or a0 or b0 and c0.

And when it comes to c2 it is a1, b1and so on. So, c1 whatever is the c1 equation is substituted expanded at two level kind of structure only difference is that this is made common. You know ai, bi and ai or bi because that is used every where so, that is made common. So, there is a another kind of level of gates that is all what it so, you get as substitution goes it gets expanded. You know one product term becomes extra.

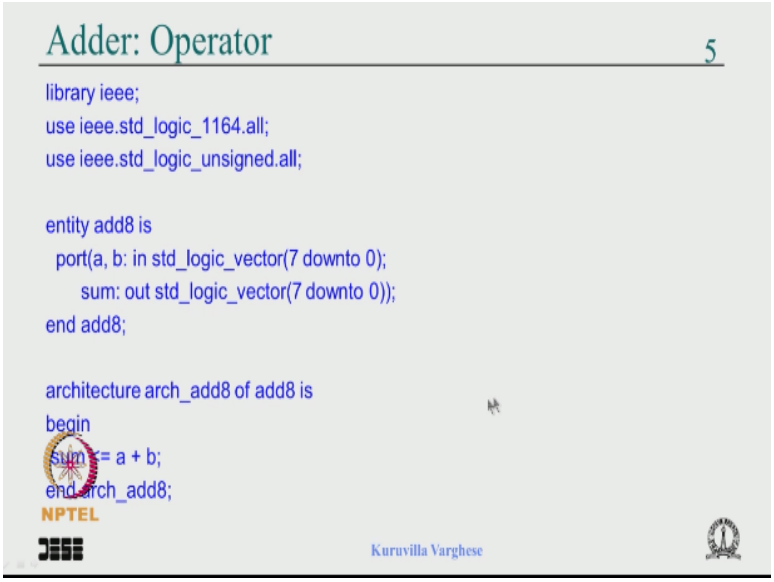
Because of this process, and you have you can see that this is c0, p0 at to a level of gates. So, you have three levels of gates for the carry and then the xor followed with the xor. So, you have around kind of five gates delay for all the stages. You know that is what is achieve by the carry look ahead adder.

(Refer Slide Time: 03:14)



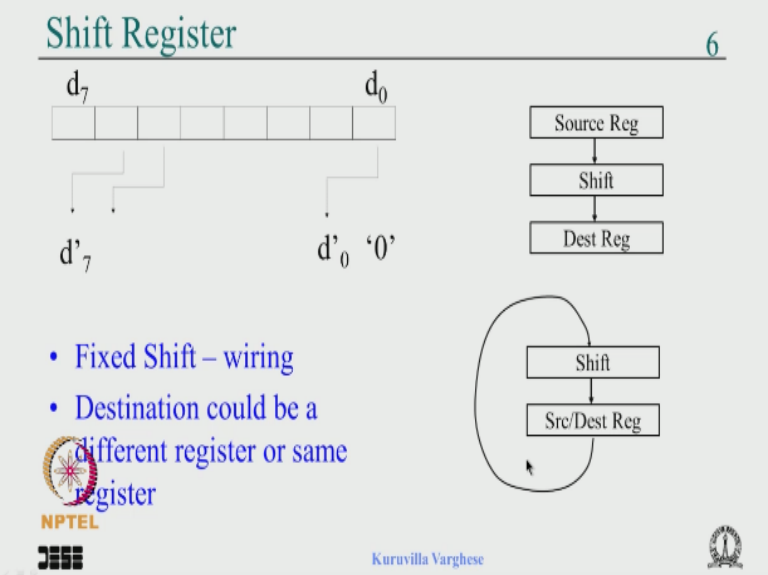
And we have a implemented the sum and gi, pi in a loop, and the carry equations are explosively written not a big deal. Because even if it is to a write a 32 bit it not a big problem, you can write it.

(Refer Slide Time: 03:32)



And ultimately at least in the case of + most of the time, we use the + operator depends how it is implemented and mostly it will be ripple adder in a FPGA it will result in built in adder resource which we will look at it or later when it when we come to the FPGA part.

(Refer Slide Time: 03:56)

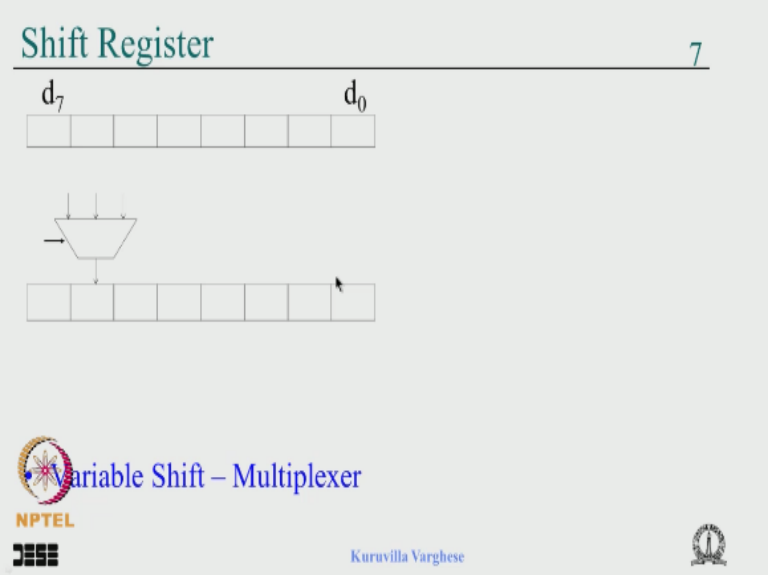


And we have looked at as in example the shift register and most of you would have learn some, how you have you would have couple the shift with the register. But you should know that there is a register, then there is a shift operation, and when you want to shift by kind of say 1 bit then, it is a matter of wiring, say d6 go to d7, d0 go to d1. This is d1 d dash 1 and you upon the 0 so, it is just a wiring and designation could be a different register.

In most practical applications it will be a different register you know only less very cases where the designation is same as the source, because you have a data path, where the data kind of you know goes through a pipe line. So, there is no way, you can push it back you know many a times so, you it goes all the way unless the algorithm itself as a iteration back then you iterate over the data path.

The example will be some kind of encryption schemes where you do it iterate it over which can be unfolded but then that is an easy that is a very costly kind of cooperation. So, it is iterated over back through the data path.

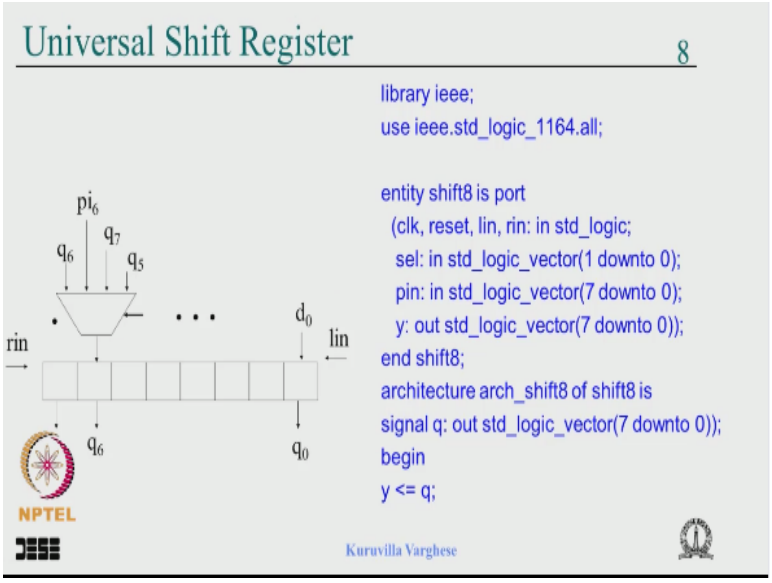
(Refer Slide Time: 05:18)



And but when you need some kind of variable shift say, you want a shift register with shift left, shift right are you shift by one bit shift by 2 bit. Then you need to have a configurable shift that is the achieve by a mugs so here I am showing shift left in that case the q6 will get the q5 for shift left and if it is shift right then q6 will get q5 and it is a parallel load it will be coming you know coming from the external input and so on.

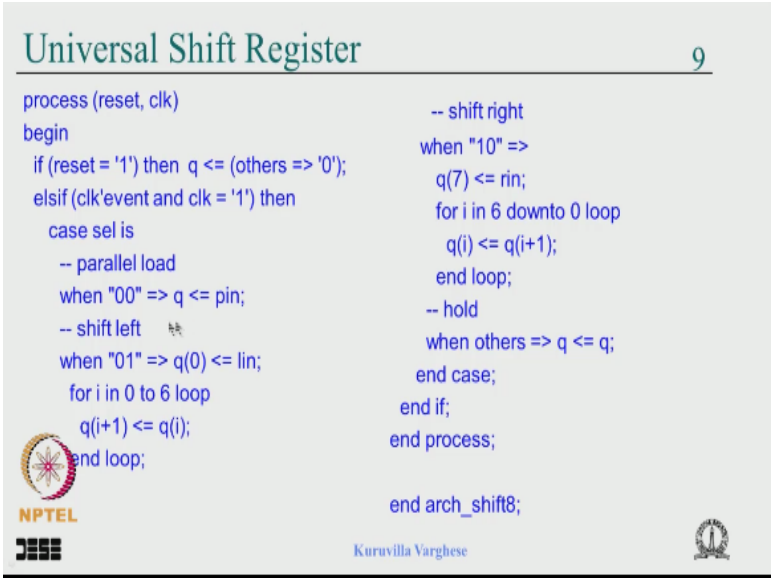
And there is a select line depending on the number of paths and this is replicated so, you have an 8 bit register then you have a 8 mugs as, and so, I said you can view it as 8 separate mugs or a huge mugs it depends, how you view it and the coding also can be different depending on that so, the variable shift is done through a multiplexer.

(Refer Slide Time: 06:16)



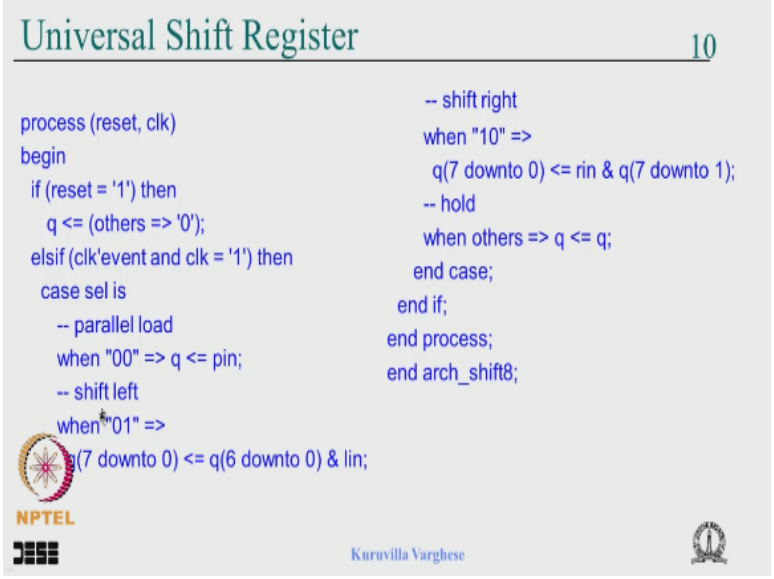
And we have a seen a universal shift register, where you have shift left, shift right and a parallel load and that is the code.

(Refer Slide Time: 06:26)



And we have seen we have written a case within the clock event, clock is equal to 1this one shows separate mugs as as a loop.

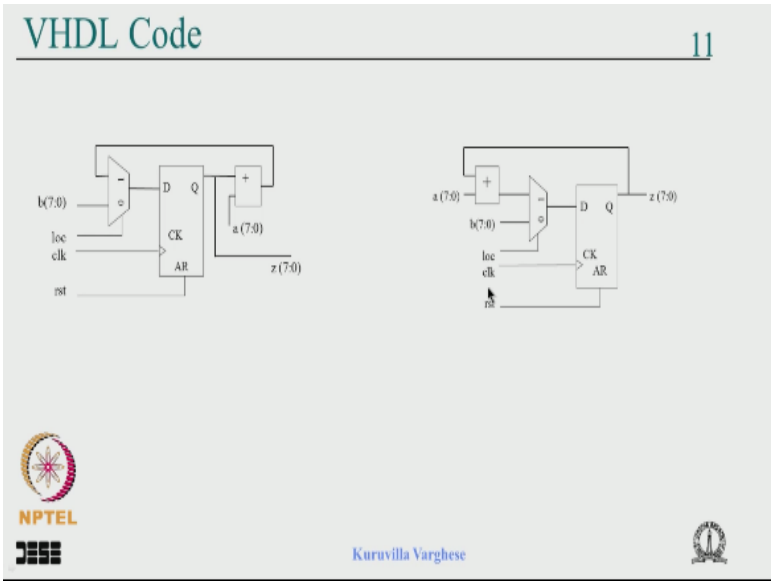
(Refer Slide Time: 06:37)



And the second code shows a kind of single mugsing essentially it is same but like I am talking in terms of the hardware now. But in terms of the syntax into VHDL this is kind of loop and this is kind of what is a vector assignment otherwise the circuit wise there no difference there could be difference, because it all depends, how the synthesis tool kind of inferred the structure and how it is map to the underlying hardware and all that.

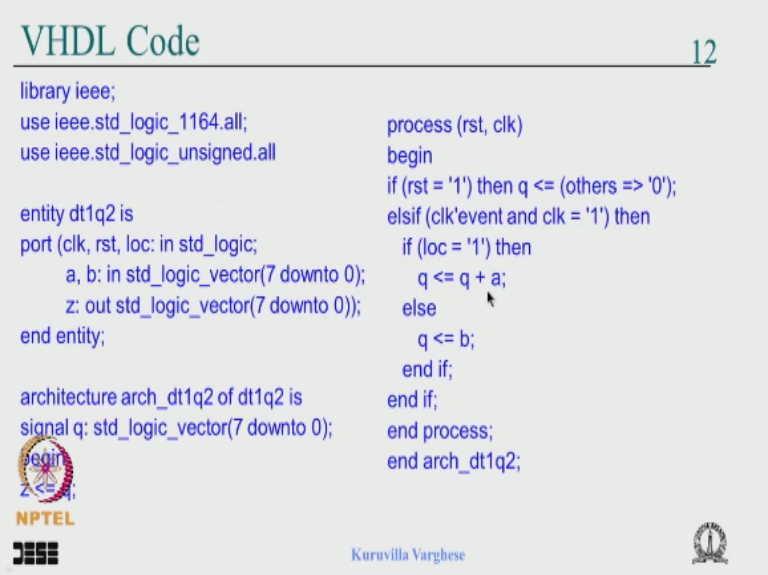
There could be different when as to try it out maybe when we play with the tool, we can take some example and have a look at how the synthesis tool handle the various type of coding and so on.

(Refer Slide Time: 07:23)



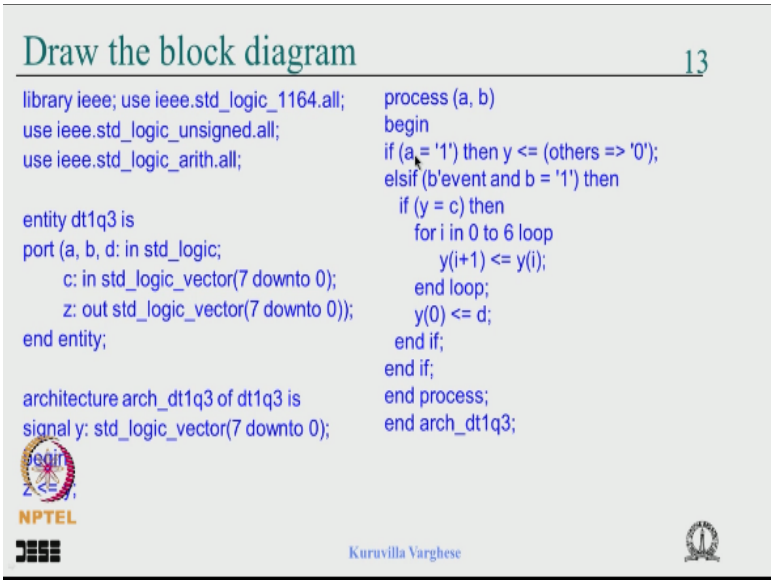
So, we have seen another example given a block diagram how to code as I said we can code a register with the proceeding logic using a single process. And here it looks like there is some logic proceeding it, and some logic following it, but this being a kind of you know cyclic this can come over here that is what we have done. And then you can write a process like clock and reset is there if reset is 1, then z is 0. Otherwise upon the clock if clock is 1 then it is q +a else it is b. So, that is what is shown here.

(Refer Slide Time: 08:04)



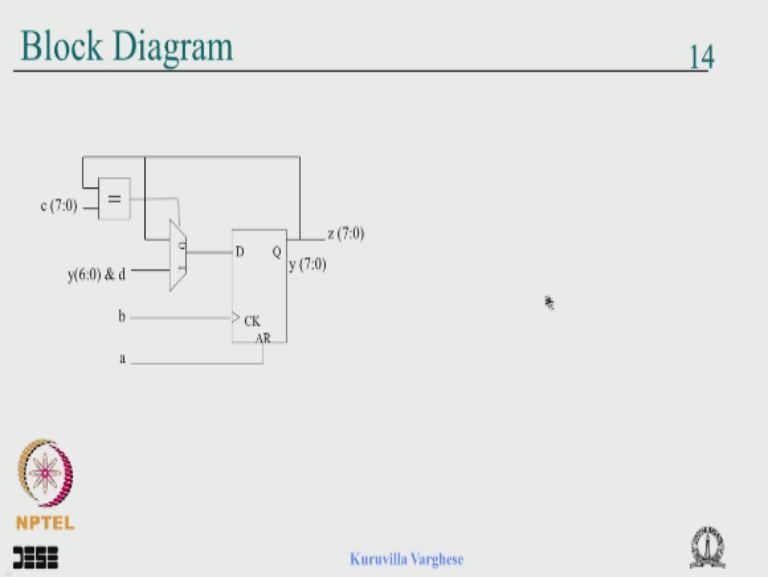
If reset is 1 q is other 0 upon the clock if loc is 1, then q gets q + a, else q gets b, it is very simple to write the code.

(Refer Slide Time: 18:17)



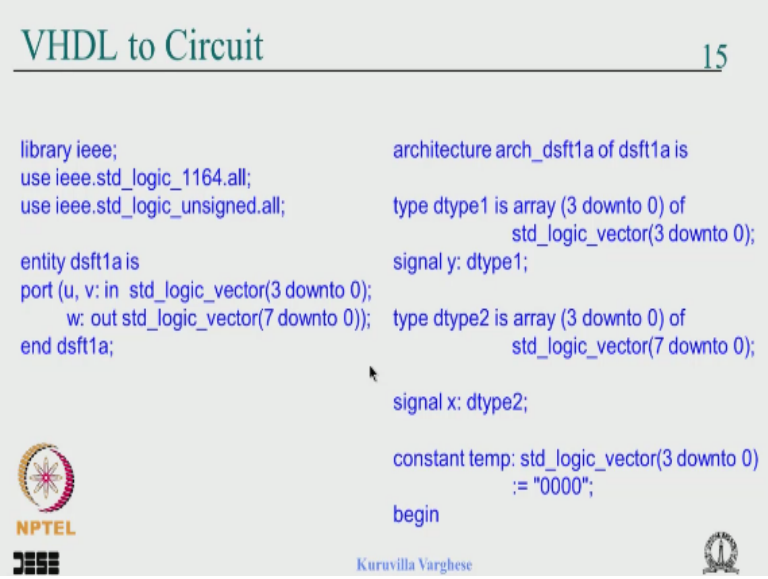
And this is the another example we have got the VHDL code and we kind of work back to find what is the circuit which is implemented by this and from this we could like you have an, a is equal to 1, then the output is 0, otherwise upon the b. So, a is a reset b is the clock so, this is a comparison y is combined the output is compared with some input and if it is 1 then the data is shifted you know the output is shifted version of the present output. The next output is shifted version of the present and that is it, otherwise it will hold the values

(Refer Slide Time: 08:59)



So, you get a as reset b as a clock, then the y is combined with the c if it is 1, then a shifted version of the y along with the d goes in here. Otherwise it is kind of re-circulated that is the hardware you get.

(Refer Slide Time: 09:17)



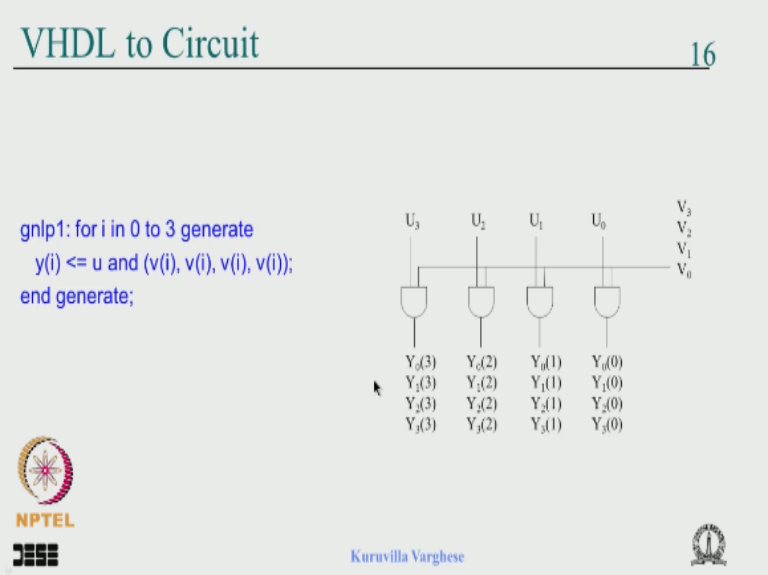
And this is where we have kind of stop okay this is another code a VHDL code our aim is to kind of find at least draw the block diagram of the circuit. And infer what is the function of the circuit okay so, at the entity level it is very simple. You have two inputs which are for bit u and v, and a single output which is 8 bit okay, and there is no clock or reset. So, you are sure at least have not shown the code.

But this being a practical code there is no mention of something kind of sounding like clock and reset. So, this is a combination circuit and which takes 2 for bit values and give an 8 bit value. Now coming to the architecture, before the begin, we have some declaration a d type1 it is a new data type which is an array of for locations. Each location is for bit and we are declaring a y of this type okay now that means there is a y3, y2 y1, y0.

Each of which is for bit okay now another data type which is again for location 8 bit and we declared x as that type that means you have x3, x2, x1, x0. And we have each of x3, x2, x1, x0 is 8 bit, and we have defined a constant that is for easy kind of coding maybe for kind of some symmetry, we will see in the code, which is temp which is of for bit vector which is all 0s okay so, 4 0s okay.

So, we will see the code so, remember this u and v are the inputs for bit, w is a output 8 bit, we have some intermediate signal y which is for location for ys, y3, y2, y1, y0. Each of 4 bit x3, x2, x1, x0 each of 8 bit okay now let us go to the code.

(Refer Slide Time: 11:42)



So, look at this you know there is a generate loop now, as I said u and v are the input, and y is for kind of values, for indexes y3, y2, y1, y0 so, the loop is going from 0 to 3. Now look at it y of 0 is u and v of 0 replicated for times so, this is kind of aggregating it. So we know that u is a for bit vector which is an input so, you have u3, u2, u1, u0 which is anded with u3 is anded with v0, u2 is anded with v0 and so on.

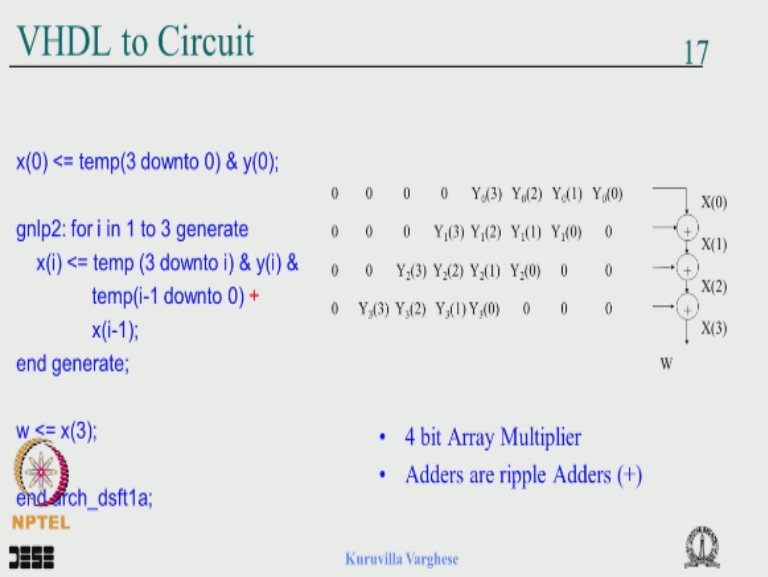
So, all of the for bit is kind of mask with v0 okay in the for y0 when it comes to y1 this same u is kind of masked with repetition of v1s, y2 is nothing but u and kind of sorry y y3 is u and v3 all v3s and so on okay. So, you gat 4y0, y1, y2, y3 which is nothing but u is mask by this bit okay so, I am going to show this pictorially, you have a for bit u vector u3, u2, u1, u0 the first operation is that.

You take there is a v which is going from v3, v2, v1, v0. So, initially the u is mask with all v 0s, a v0, v0, v0, v0 that is what is done here. And you get y0 which goes from 3, 2, 1, 0 then next is v1 then it is masking, and you get y of 1, 3, 2, 1, 0. V2 then you gety2, and you say v3 then you

get y3. So, it is going like this the arrow is going like this one by one, and here the arrow is going like that.

You know so, maybe I should have made it kind of opposite pattern I hope it is clear v0 make y0, v1 y1, v2, y2, v3, y3, okay now may be you get an idea what could be what possibly could be the circuit but let us wait you know let us look at go head see the next part of the code. So, this is a very simple code but this is how the circuit is coming up so, I think keep this in mind let us go to the next part of the code okay.

(Refer Slide Time: 14:33)



Now we are now here we form the ys from u and v that is the game, and in the second part we are going to make x from y and term, and you know that the temp is just 0s okay. So, let us look at it the x0 which is not in the loop probably which could have in the loop. But let us see x of 0 is temp 3 is down to 0, which is nothing but 4 0s, and y of 0, y of 0 is this. This for bit so, you have 4 0s, then appended with y of 0 okay.

Now we get to the loop, loop start with because 0 is already make loop is going from 1 to 3, and you say let us take the first index x of 1, because x of 0 is already there, x of 1 which is temp 3 down to y, that mean 3 down to 1, here it was 3 down to 0, 4 0s. But here it is 3 0s, then y 1 now so, the earlier was y0 was appended, now we are append the y1 it is appended with 3 0s y1, and temp i-1 that is 1-1 temp 0 down to 0, that means 0 only 1 bit.

So, you put a 0 here that mean 3 0s y1 and 10 okay, now you can guess when it comes to x2 this become 3 down to 2, the you have 2 0s, y2 and it is 2-1, 1 down to 0, 2 0s. And the next loop when it is x3, it is 1 kind of you know 3 down to 3, 10 y and 3 0s, and in addition you see the as loop goes initially, we have 4 zero and y, now when it comes to x1 it is 3 0 y, and 10 + this is the previous one so, this 2 are added.

When it comes to x2 whatever those 2 added is added with the new value and so on. And when you get x3, x3 is assigned as w which is the output okay. Now let us look at that picture so, we initially put okay now come back to this, we had u, we had v each of the first least significant bit of v0 is mask. You get y0 another bit is mask, then you get y1, next bit is mask with u, y2, y3 now you are forming all 0s then put the y0.

Then you shifted version of y1 is added together, then you shifted version of y2 is added to it, and shifted version of y3 is added to it, and you get w. So, it is you have getting the point so, you know that it is nothing but shift and add so, it is like it is multiplying and if you look at this. This is nothing but forming the partial product so, you have a multiplicate end which is for bit u3, u2, u1, u0. You have a multiplier which if for bit.

So, the first partial product is form by the least significant bit of the multiplier, anded with this multiplicate, you get the first partial product. Then the v1 is anded with this, you get the second partial product, v2 is anded with this then you get the third partial product. And then v3 is anded and you get the fourth partial product okay, that is what is done here so, you get 4 partial products and now.

When it comes to this structure it is nothing but partial products are shifted left you know one by one, and it is added together so, it is like you know the partial products are added to get the product okay. So, this though I am showing this adder there is this can be kind of the it is showing 3 adders and parallelly it is added. Because it is a generate loop it is not at the same adder is use.

Each time, a new adder is use, because it is replicated so, you get a parallel adder so, it is a multiplier but it is not a kind of iterative multiplier. It is a parallel multiplier. And the name of this multiplier is called array multiplier, because you form an array of the partial product and you add it parallelly. So, it is an array multiplier, it is a 4 bit unsigned array multiplier, only thing is that normally when you refer to the text book to avoid the rippling something called instead of ripple adder.

A carry say adders are use, here you know that when you add this there is rippling here. Then there is a rippling here it is possible to avoid the rippling kind of because the idea is that say, you are adding this two bit it does not matter, whether the carry comes here or carry comes here. So it is possible to put the carry of this stage along with this here, similarly carry of this stage can come here, the carry of this stage can come here.

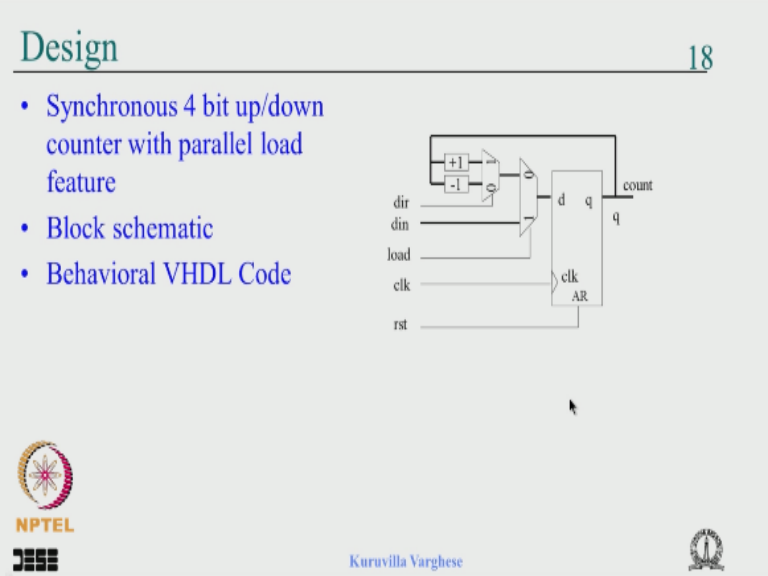
So, that gives a kind of to reduce the ripple otherwise it is ripples, ripples, ripples and you know so, you will get kind of 4+4 8 rippling, it is possible to avoid the rippling using carry safe adders. But this i s say kind of the idea is not to kind of design the array multiplier with the carry safe adder. It is a practice in kind of working out from the code written inferring the circuit which is implemented.

And as I have shown though the code looks little crazy it is very easily you know you can write and work out the underlying hardware okay so, only thing is that it has to be done systematically you have to work on the pen and the paper okay. When I say pen and paper definitely you can write on your tablet or the note tool or I do not know when this course is being recorded in November 2013 maybe somebody is watching in 2000 16 or 17, if it all it is done.

Then I do not know what is the technology may be, whatever maybe the technology, you can kind of you work it out okay, literally work it out on whatever tool, you have I would say on a paper and pencil. Then the things will be very clear there is no kind of ambiguity, you know I have I have taught you what the code means now it is a matter of patiently working systematically. Then you will get what is the circuit and in the process.

You can add like suppose somebody has given a cod like that definitely infer that the ripple adder is being use that is lot of rippling maybe we can replace this ripple adder with the carry safe adder. And make things faster the critical path delay can be reduced and so on. So, that is the advantage of kind of working the hardware from the VHDL code so, that should be kept in mind. So, let us move ahead one more example.

(Refer Slide Time: 22:28)



That has to be let us look at this example, so I want you to design a synchronous 4 bit up down counter with the parallel load feature okay. That means it is a synchronous counter 4 bit, it should count up when the direction is 1, down when the direction is 0,and this is a load signal and 4 bit input when the load is 1 it should load, the counter width the input value and depending on the direction when the load goes low depending on the direction.

It should go up or down okay and we will first we will do is that, we will write the block schematic. Then we will write the behavioural code, you know step systematically from this block schematic which is kind of drawn. So, you know that the 4 bit counter needs a for flip-flops, you have the clock and reset okay. So, like when the reset comes, the count is 0, and when the clock comes that the game starts the first thing is that.

You know that in a parallel load counter, we need to have the priority for the load. So, we will put a mugs here okay to the d and the select line is load, and when that is 1, we supply the input

okay. If it is 0 then we have to put a count part okay so, let us put that when so, we put it 221 mugs at the input. Because that is the highest priority, and we have seen that the highest priority will come near to the d because that is the one which gets effect first.

So, when the load is 1, this din gets loaded into the d, if it is 0 now the direction comes now we have two things to do if there is a direction is 1, then it up counts, direction is 0 it down counts. So, what we do is that we put a 2-1 muxs here, the select line is the direction, if direction is 1 then what we have to we have to increment this. Because it is q so, you take the q back put a +1 to the one input of mugs similarly if direction is 0, then q has to be decremented.

So, q is put through a -1 and put to the mugs okay so, let us put that so, we have a 2-1 mugs there is a direction signal. If it is 1 q is taken back to as a input then we have a +1 which goes to the mugs, and if it is 0, qi going through a -1. It goes to the 0 and it goes there okay. So, now that is simple we have 4 flip-flops reset clock which is common. We have the highest priority load which goes through a first 221 mugs.

And load is 1 this goes there load is 0 depending on the direction now. Direction is 1 then q is q+1 else q is q-1 okay, and that is simple now actually like the design is over now if you look at this you know that this is entity has say reset as input single bit, clock as a input single bit, load as a input single bit, direction is input single bit din is an input which is standard logic vector 3 down to 0 because 4 bit.

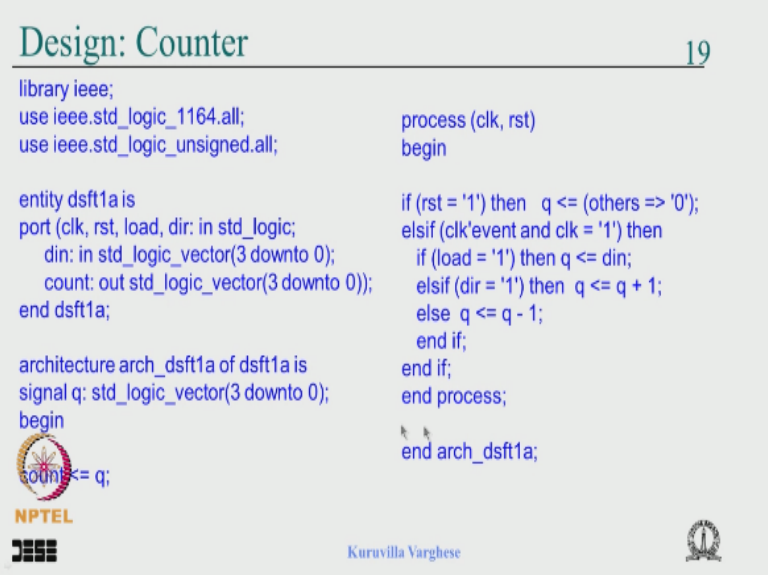
Then we have a count which is output which is 3 down to 0, 4 bit, that then we this structure shows that count is also input to it. And we know that the VHDL does not allow, we declare an internal signal called q, and then we use q as a to write the code and in the architecture statement region. We assign this q to the count okay, now if you look at the VHDL code simple process clock and reset.

If reset is 1 q gets other 0 else if clock event clock is equal to 1 under that if load is 1, q gets din else if direction is 1 q gets q+1, else q gets q-1 end if sorry yes end if that is it so, that is the whole code the moment you see this the code is there is no need to write a code. You now it is a

mechanical process so, it is possible to generate the code if you want. You know somebody draw a block schematic.

It is possible to generate the VHDL code out of the block schematic okay it is for a computer science student is very easy, you know from this kind of structure to generate a VHDL code. It is very straight forward like you have a tool to draw like the moment you put the draw, like you are only allowing the designer to draw. So, you know what is happening and you keep you know adding the templates you have a template code. Then you can keep you know generating the VHDL code it is very easy.

(Refer Slide Time: 28:09)



So, let us look at the kind of the code, you have the library and we have standard logic 1164. Then standard logic unsigned because we are going to use a + as I said clock reset load direction is 1 bit, din is 4 bit input count is 4 bit output then the q is the same as count which a signal 4 bit count gets q. Then we start a begin okay, in the begin count gets q then you write process clock reset begin.

If reset is 1, q gets others 0 else if clock event clock is equal to 1, then underneath all that is coming behind and this as highest priority next this as a priority. So, if load is 1 then q gets din else if direction is 1 then q gets q+1, else q gets q-1, end if okay. So, and this end if this is the

memory end if, this is the combinational circuit if this represent the combinational circuit with kind of priority and this represent the memory end if.

Because it no else and end process and the architecture. So, that is very simple and that is how you should write the code very straight forward, there is nothing to worry but if somebody say like you give this spec and you start like that straight away without any clue whatsoever about the circuit and most people try to do that you know they kind of some vague idea and you know start assigning things and things can go wrong.

Because you will be wondering you know something goes wrong then you will be playing with the if you say ok. Let us say instead of putting the if else you are you put something else right another nested if function. Because you have no clue on what is so circuit and you will waste lot of energy in going but if you know as we have learn you know what is the meaning of this syntax as far as the circuit is there concerned then it is very easy very very very very easy to write.

After practice these kind of structure will be in your mind you do not need to kind of expanded you can just shows a blog with low direction and some way to indicate which is which is kind of priority over the other and things like that some kind of notation you can use them right away you can call it it's very easy. So I think it is a some kind of logical place to stop for a while because we have taken quite a lot of VHDL.

But that is to put you in a strong putting, so that when we go to the to take a case study try to implement your kind of through. So in this class we have looked at the carry look adder, adder VHDL code then we have seen basically the ripple adder, carry look adder, adder then some examples of some random circuit how to code it. Then basically we have looked at the code of an array multiplier and worked out the underlying circuit.

And without even bothering about what is the functionality of course you are writing kind of exam and you are such a kind of question then you are your kind of finding like I know you have a shortage of time then definitely you can guess and jump to conclusion, but when you are doing this serious design in a design team we cannot do that we should do as I kind of have shown.

But when you are away so serious design you should not get anything you should work it out and in for what is going on and the last thing we have look at that is that given the spec of account how to draw the block diagram and how to write the VHDL code for 1 to 1 in a what is shown on the block diagram is a code return give it ant synthesis code, it is going to generate the same circuit.

Maybe it is map to some inbuilt blocked as a single unit or multiple unit, does not matter but then we are not sure maybe the multiplexer in there is implemented using some special resources within an FPGA, but that apart from that you know apart from the technology the block schematic which is generated by the synthesis tool will be exactly same as what you wanted and what have drawn on the paper or whatever return the code.

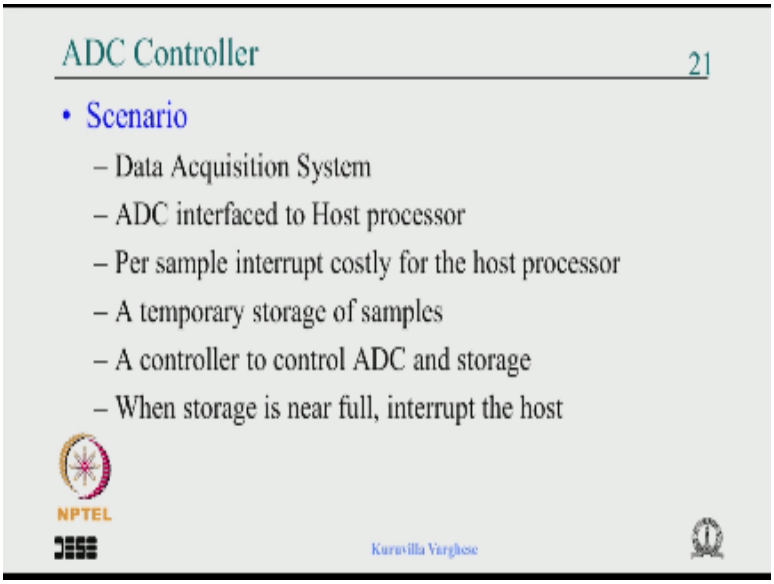
You are 100% sure about that. So that is a kind of thing I want you to do is a designer and do not think that a anything complex will have you know much more difficult than this because ultimately we have seen that you know you have a complex circuit then we are going to break it down hierarchically as data path in control then we are going to data path itself is going to be broken down do this level of detail.

And then you can write the code in a very very very straight forward way. You do not have to kind of do not have a very happy that you get a c code to VHDL compiler you can do that, if you are in short of time or software engineering is trying to do the hardware design and son but I do not see any see big point in doing that I definitely need to improve as we go along the standard.

But still it will be kind of template basic cannot be kind of Artificial Intelligence and things like a machine learning and things like that but maybe we will come like that but at least for the time being it is kind of you know there is underlying templates you map and the you in for the structure architecture and write a map to the standard structures and so on. So let us stop ahh this the VHDL part here.

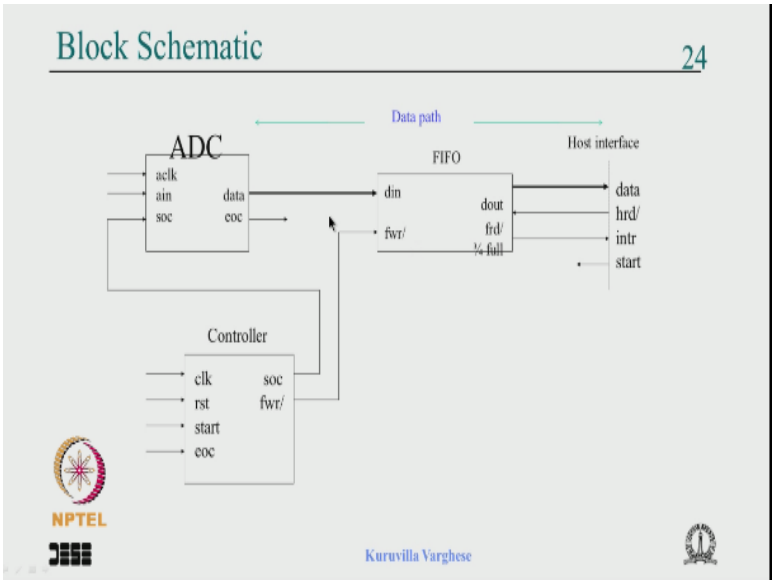
And let us for at least 2 lectures let us get back to the digital design. So that we can continue maybe with the VHDL little more. We will go hand in hand so let us move on so let us come back to the digital system design part now very quickly I want you to run through the last thing we have done, hopefully I am able to kind I am putting the thing correctly.

(Refer Slide Time: 35:08)



So the last thing we have discussed and digital design was a case study, if you have not familiar kind of if you if you forgotten or if you are not taken that part of the look at the part of the lecture go back and look at it ok.

(Refer Slide Time: 35:06)

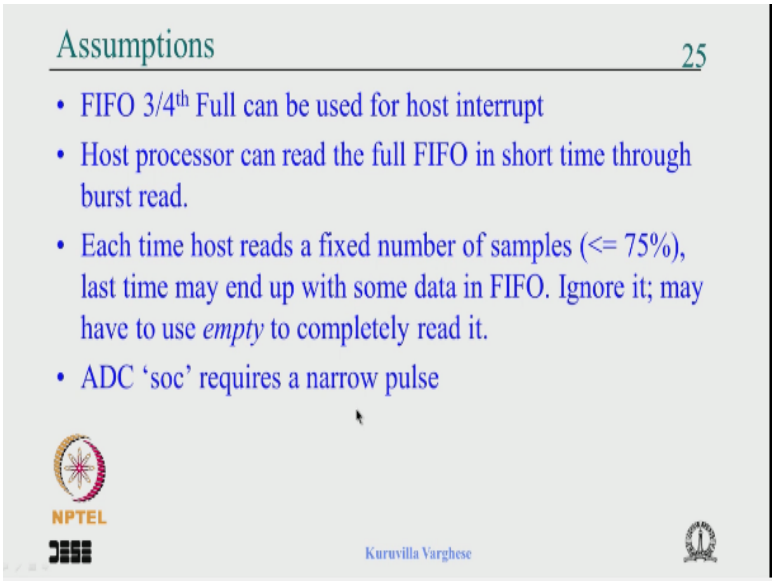


So what we have done was too we have ADC see what we have tried normally it is kind of interface to a host CPU or a microcontroller a processor what we are trying to do is that too kind of take the load of the process by having a controller controlling the ADC and putting the data to a temporary storage and when is kind of food the host will read it that was idea and we found that the best candidate for this is FIFO.

Because other the you know the you write the data in order you read the data in order. There is no and it is a one way data flow through you put the FIFO. So you put a controller to controller give the start of conversion end of conversion comes to the controller in addition controller has a clock and reset to use the fee for it and when this becomes full time is it in rats the host and host to read it.

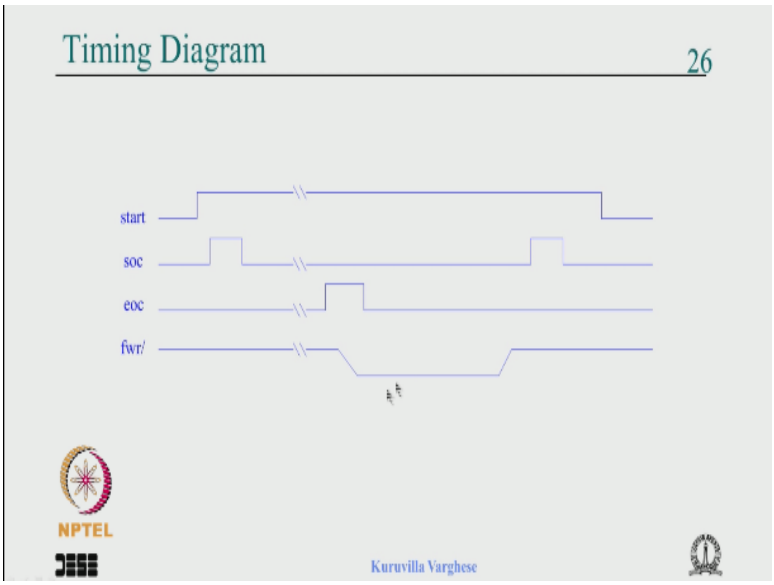
And host start for you know controlling the whole operation when it is started wanna keeps on doing it and when it is it stops and this is the data path FIFO and we are at least in this case we assume FIFO is there already and we have looked at the controller.

(Refer Slide Time: 36:46)



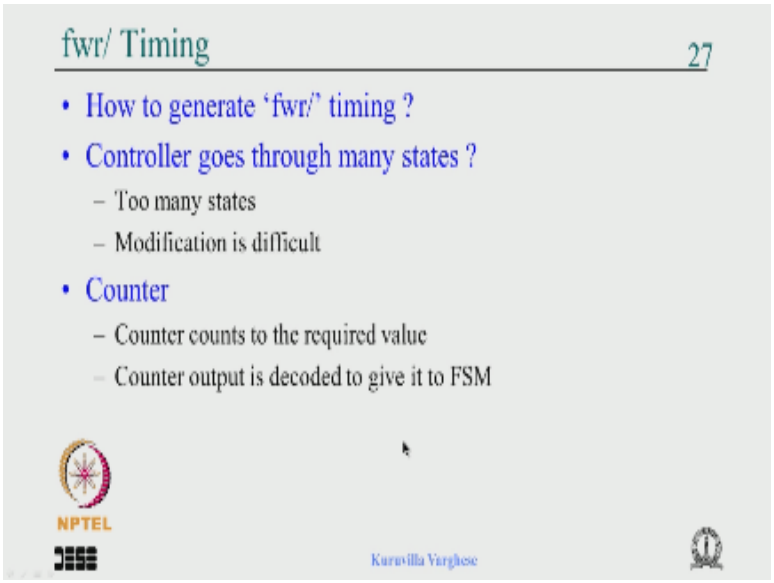
And first thing we made some assumptions you know that it is and we are gonna timing diagram wait for the start when they start come give SOC wait for an end of conversion give right signal.

(Refer Slide Time: 36:56)



Then which at that this is a can be a narrow pulse in one state you can give and output but this should be of some width and we need to generate that time properly.

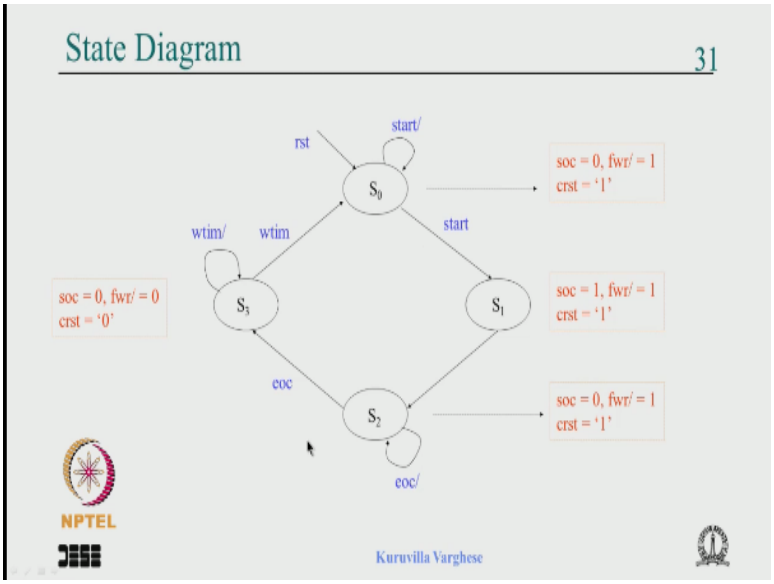
(Refer Slide Time: 37:11)



So we decided to put counter as a subsystem which is control the reset of which is control by the same controller or state machine and that is equal to decoder for particular value then gives the time. So idea is that when it when the FIFO right is given the rest is remove and it starts counting and wait for this time when it reaches the pulses, get a pulse, then you put that reset and terminate the FIFO.

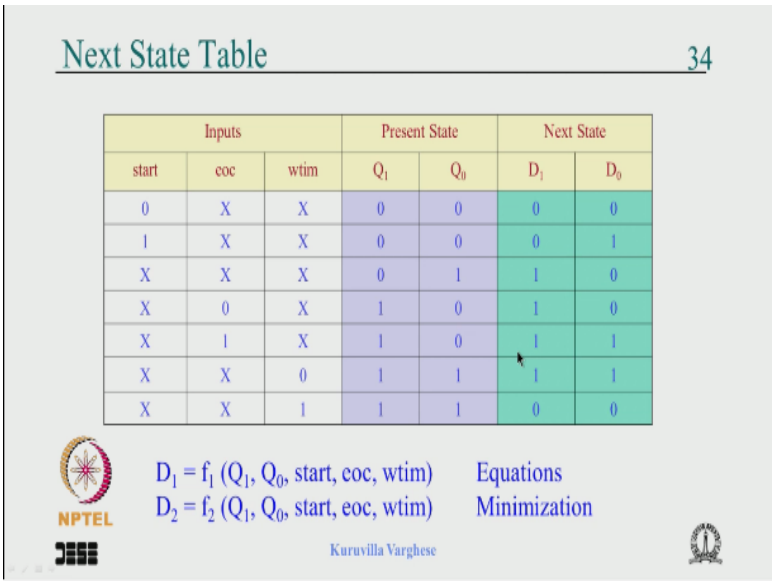
So have updated the timing diagram added this when you put this low the reset is removed wait for this and from this we have worked out an algorithm like you know wait for the start give start of conversion weight of an end of conversion, give FIFO right remove the research weight of wait for the signal and terminate everything go back to the same thing ok and that was the control algorithm.

(Refer Slide Time: 38:11)



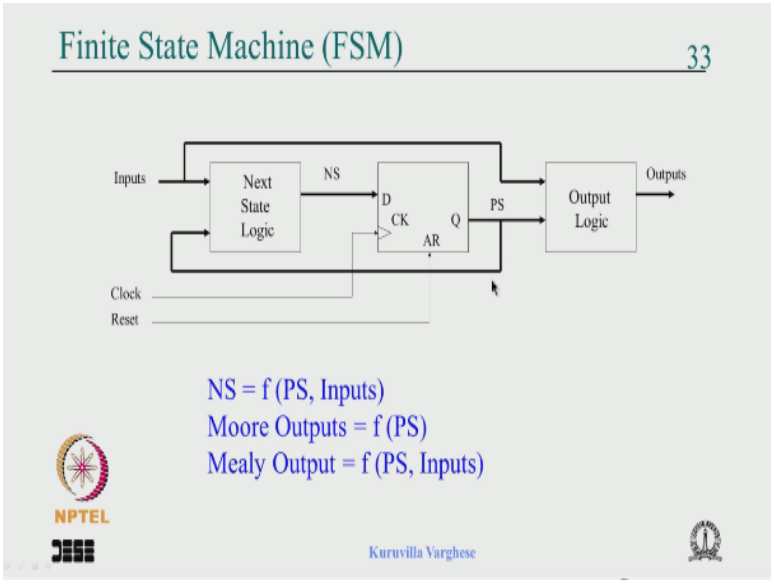
And we have returned IST diagram and exactly similar thing and so there are in boards transition states and output and we have decided use D flip flop for state binary encoding 4000 11011 and from this state the import transition you can work out the next table state output can work out output table.

(Refer Slide Time: 38:39)



So we have done that you have.

(Refer Slide Time: 38:41)

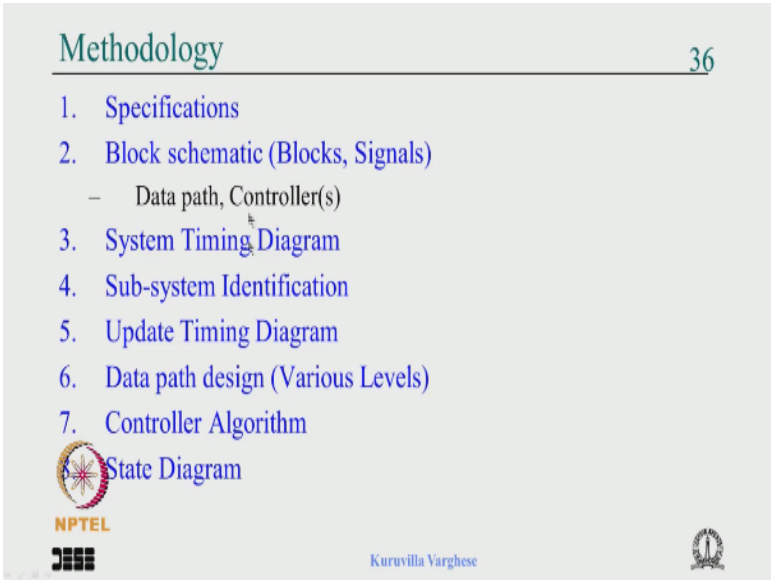


This is the controller we have basically the state variables or the flip flops which gives a present state, in our case to flip flop and that along with the input decode the next state and upon the clock next it is loaded and in each state the present state output is Decoder from the prostate as a function of the present state or as a function of input and the present state. If it is only the function of present state is Moore.

There is a functional present state and input then it is Mealy and we have returned as I said from the input and the present state and the transition this table then you get an equation for D1 and

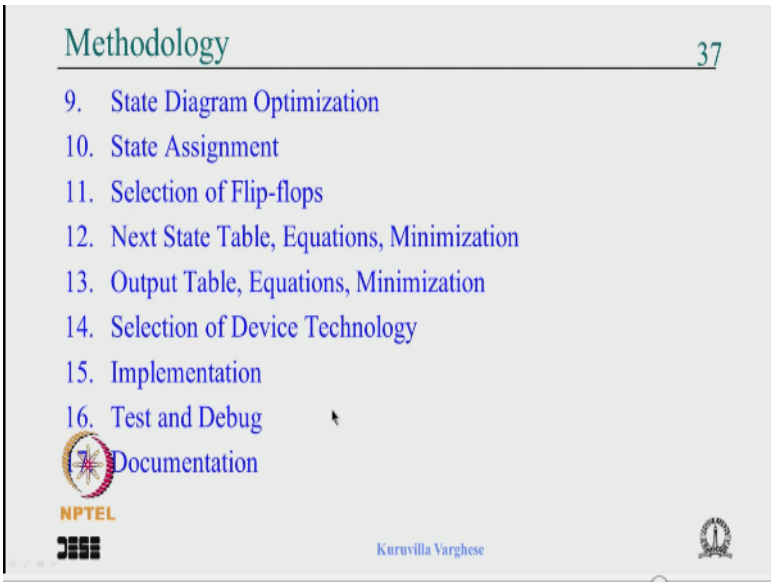
D0. In terms of the present state and input then you can know minimize and implemented similarly for the output you write the output in terms of the present state and you get output as a function of present state.

(Refer Slide Time: 39:39)



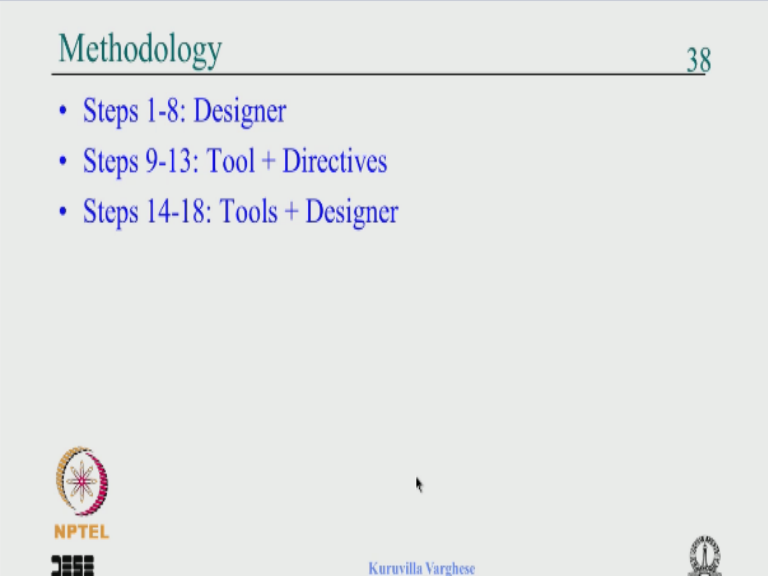
You from the equation and minimise it and that the old methodology and basically up to the state diagram is what are the designer has to put afford then the tools come into picture in all minimization and ultimately to test and debug you use some kind of equipment.

(Refer Slide Time: 40:00)



And a designers you have not design here designer give some kind of input and that is what we have done. So we have seen the whole process of designing of course the data path as we design.

(Refer Slide Time: 40:11)

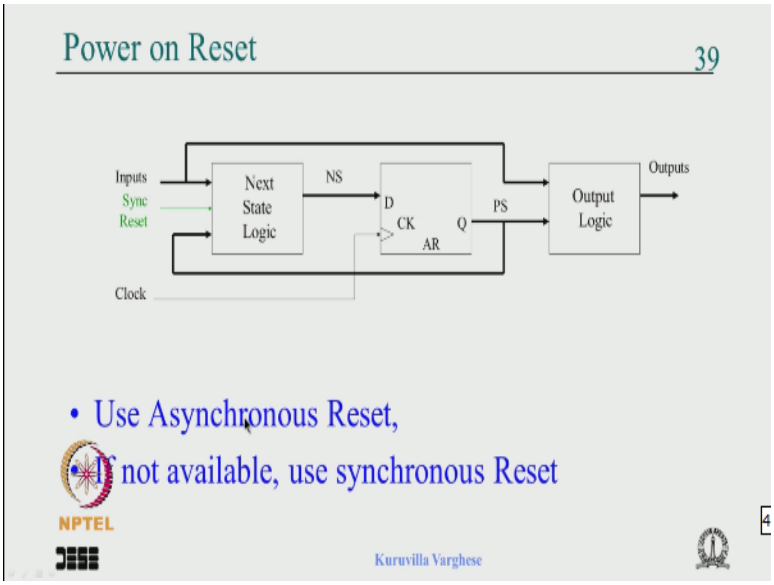


So let us now take some issues in the other state machine itself the controller. The first thing is that we know that we have happily put like when we discuss the state diagram you see here okay, the state diagram is here, so we have shown up on the reset that means at the beginning when the power on reset come, it should start with S0 ok. It can be 0, 0 can be any state any not be 00.

But it can be any state but their the game is that this reset has to be implemented, you should not forget that you know that is very dangerous you work out everything, but you forget to implement reset in state machine, then you power on it can come anywhere it is very dangerous we have only 4 states and 2 flip flop. But assume that we have say 6 states, then we need to use 3 flip flops.

And there are 8 states there will be some unused states and why you power at on without reset and if it goes to someone you state we do not even know what is the behaviour of the state machine very dangerous. So it has to be properly reset. So that is what we are going to see that.

(Refer Slide Time: 41: 45)



So let us look at that ok. This is the state machine we have the flip flops, present state next state logic and output logic. So very simple if you have reset asynchronous reset in the flip flop just use the connector power on reset that and the power on will become will come to the starting C, when it comes to the starting stay then you have the control, you know you are the one who is designing the state machine.

And you can check the end of the equation you can bring it back to the what are the state you require, you need not come back to the starting state depending on the application, it can go to some other state in wait for some input, but you have to at the beginning at the power on you have to bring it to us starting state and that is why it is used and at this is like in VHDL coding this is symbol because this is combined with the flip flop ok.

So you can treat this as a kind of process for a flip flop. Then this reset is implemented along with that flip flop you know as the synchronous reset of the flip flop. Suppose take the case where the flip flops in the technology whatever the technology used with ASIC or an FPGA assume that this reset is not available. Then you and there suppose there is no reason at all in the flip flop.

Then what we have to do is that we have to introduce asynchronous reset has a name suggest it is something to do with the d so it is very simple ahh you add an input reset here ok and change the

next state logic and we have seen the truth table can be changed or we are going to write the VHDL codes of in this summer we are going to reset is 1, then this next state is the starting state then that is done ok.

So if this is not available then use asynchronous reset but not both know I have removed that and if using rounds reset if not available for use or synchronous is not available use asynchronous reset. So that is done and from the experience what we have with the kind of data part design a commerce student can already in for what could be the circle in terms of inner we have we have shown for counter with the load signal.

This reset as the priority like that there is in what does present state but definitely we know that synchronous reset as the priority. So if you look at it, it is equal and kind of putting a mux here ok and the reset is the select line and when the reset is 1, a 0 goes there. Otherwise this next state logic comes here. So it like if you write a code for synchronous set and give it to the synthesis tool that is going to happen.

At the output of previously return code previously generator circuit you are going away 2-1 mark where the select line is reset and you get a zero ok. That is what that is what happening, so I am kind of reinforcing what we have learnt. Probably I have shown the picture, but you know it takes a lot of effort to draw this pictures and put it and I have already invest a lot of energy.

So but I think you can kind of imagine what is what is basic idea here. So that is about the power on reset and when it comes to the VHDL will go back to the VHDL coding of the state machine. There is nothing kind of nothing great because we know how to code flip flops or registers, we know how to code the combination circuit because this is a combination circuit.

This is a combination circuits or you could we have seen that you could combine the output logic with the next state logic, but after all combinational circuit and we know how to code the combinational circuit, but yet we will see that because the controller is one important part and we will see how to properly code it and it is also it is there are different ways of coding but it is very specific to maybe the tool in there maybe is 6 different ways of coding it.

But you are tool may not understand everything then there is a problem, so will see that, we will go back to the to the coding of the VHDL.

Let us look at the next topic ok now in this very conveniently we have shown a clock ok. And that is what all the text book do kind of they show a clock ok and say the clock is there. But what should be to stop ok. So what should be the frequency of the clock, that is the question ok. Many times did not address and you are kind of most of the time, when you are in an academic your answering some kind of most of the times of my exam.

Where it is enough that a line is drawn and you write the clock ok. So you are you are save by that god, but in real life that you have to give a clock ok and so we have to decide what be the clock. So definitely you must be getting some kind of idea saying that this probably can suffer or tolerate a range of frequencies. So let us assume let us try to find out what could be the maximum clock we can apply to this ok.

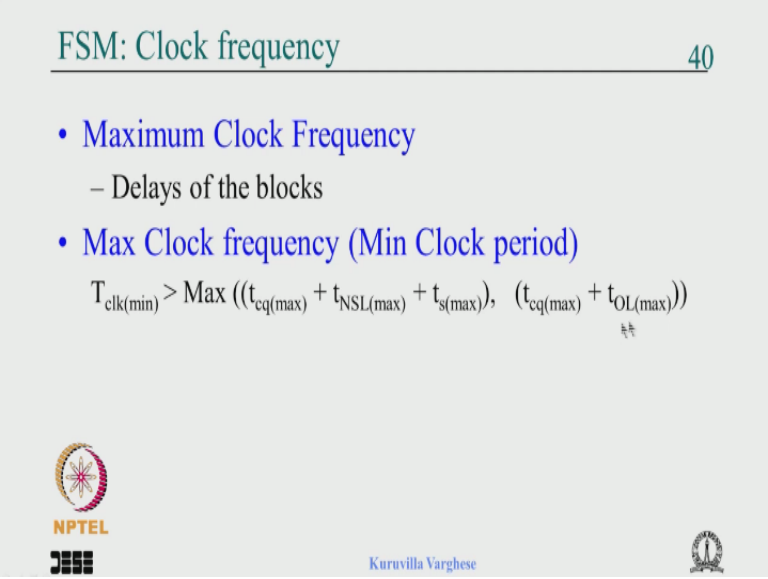
And what could be the minimum cock ok. So let us look at the maximum clock and let us look at the minimum clock ok. So the maximum clock you must be already getting the idea it has something to do with the delay of this logic delay of the flip flop and delay of this logic and we have already seen the maximum frequency of the state machine which is nothing but the maximum frequency of a data path or register to register part.

This is a sequential circuit but you have a data path as a pipeline, it is also you know you have some registers some flip flop its output is going through combinational logic and is going to some other in this case if you have Q1 and 0, Q1 output may go through some circuit good to D1 or or D0 and so on ok. So that before parts you know Q1, Q0, D1 D0, to by to your four parts.

So are you know that if you consider that the clock there is a tcq delay there is a delay through the combinational circuit and the data has to be here setup time before. So the clock period should be greater than tcq Max, tcom max-Tsetup max ok and we are also with the incomplete information we have this given clock output is kind of appear here after tcq and toutput logic.

So whichever is greater, whichever this is greater or this is greater which we choose definitely with goes through some other components required to register we have to consider that and that is what when you really design the tool is going to look at the old path you know this is just a picture but ultimately when you design this is going through a resistor output logic and combinational logic and ultimately it is going to register. And the tool is going to analyse that you know that you should keep in mind, So let us put the maximum frequency ok.

(Refer Slide Time: 50:08)



So we have a maximum torque frequency with depends on the day of the clock clock, maximum clock frequency or the minimum cock period is tclock means should be greater than Max of tcq Max+ tNFL max+tsetup max or tsetup max+tOL max and their could be kind of some more terms here but kind of with the with the information we have your putting I am putting this you know.

You can have that but suppose of design a state machine and you find the maximum frequency is a one gigahertz this kind of this can tolerate and is it a good idea to clock in at 1 gigahertz, maybe the system which is controlling which is the datapath which is controlled by se we are some data path which is which state machine controls. Suppose that is working at say 100 megahertz.

What is the point in a that is dissipating so less power but this is curiously talking at 1 gigahertz dissipating lot of power because it is sitting so there is no advantage in working at maximum frequency. We have to see how much minimum we can go because in terms of the participation that is the best thing to do like without like after all the state machine alone cannot do anything.

As long as it allows the data path to proceed at the desired throughput okay decide frequency. Then we should choose a frequency of the clock for the state machine such that the data path works properly, at the desired should give the decide performance. Then we can minimise power in the state machine. So let us think let us ask this question what should be the minimum frequency ok.

That is the great that right question to ask, so maybe I think you should think what is the what should the minimum frequency depend on ok. So is the question to ask and the answer is maybe I will put something we are in a basically controlling a data path. So these inputs are coming from the data path, some output and this outputs are going to the data path input.

And what we are trying to do that we are trying to look at various input, step through various sequences and control the data path ok. So essentially does got something do this input and something to do with output of because that is where the game is ok, then nothing extra, there is a clock, there is a reset, everything else is coming from the datapath. So it must got something to do the input something do probably, something to do the output ok.

So let us asked question ok. So what should be the minimum clock frequency ok now that I will kind of put some inputs and let us see that ok so awesome that we have three input our state machine okay that means we are looking at those say this inputs, 3 inputs from the data path, it is some output of data path in 1, in 2 and in 3 ok to make our life easy we are assuming they are all kind of you know square waves ok, which is which is not true in real life.

You do not get anything like this may or may not even find a scenario where very rare scenario where everything is kind of square wave, let us put this ok. So that is where so I am trying to product worked in on the bus, thing is that when you try to analyse and then you should put some

simpler state, simpler function, suppose in max you are trying to do something say a kind of matrix multiplication with which is a 10 by 10.

And you are trying to understand something maybe you are trying to understand something about Eigen values and Eigen vectors and so on. So it is very easy suppose you put a 5 by 5 matrix and trying to work out you will not get any kind of into idea but moment may be take a simple case of 2 by 2 Matrix and a kind of 2 by 1 vector and you try to in for these not draw a kind of graph and draw a picture then you will get some innovative idea.

So every time when you try to kind of understands of fundamental you putting simple thing, you know you do not make very complicated. So let us put a very simple picture which is not real life, then once you get the basic idea then we can bring it to them to the next level of reality we can bring back to the reality, close to real life. So what I am saying is that that I have shown three inputs.

And you know that they simply save some frequency maybe 100 into is 50 megahertz and in one is 25 megahertz ok. So now I put clock ok to see like we have to see what is the minimum kind of frequency, so I just put a clock ok, of course I want without manipulated for you to understand but look at this ok and this is working on the positive pocket okay like that.

And you see that when this clock comes say in IN1 is 0, the next clock 1 IN1 has changed, no problem it is detected with an appropriate again it is still 121, no problem, does going to zero that is detected you take the IN2 when the clock in IN2 was high it is detected then it when low ahh then this is detected and this change is detected. But you take a look at this IN3 and ahh you see that the clock edge comes and that is sample and this is 1 and there is a transition to 0 it comes back to 1.

But when the next clock comes to still 1, still 1, still 1, that means there are some changes in one of the input, but since we have chosen a particular clock frequency is not detected at all. So this shows that this clock frequency is not going to work maybe this is too low clock period ok. But suppose I put a clock like this now there is a positive clock edge yes it is detecting that it is zero.

And when the next positive at that is detecting that it is one, next positive and it is detecting that. So it shows that the clock frequency of the state machine has got something to do with the input changes ok. So definitely we know that there is a change here, so if a clock has to come the clock period has to come within this thing, so we can say as I as I how to start with why make a statement saying that the clock frequency should be twice that the maximum clock frequency okay.

So we are coming to probably the end of the lecture today's lecture I will continue, I will complete this portion, we have looked at 2 part, 1 is the reset use async reset and sync reset and maximum clock frequency depend on the delay of the blocks, minimum clock frequency has got something do with the rate of change of the input to the state machine which is output of the datapath. So please go through whatever we have covered in this lecture, revise it, learn well and I wish you all the best and thank you.