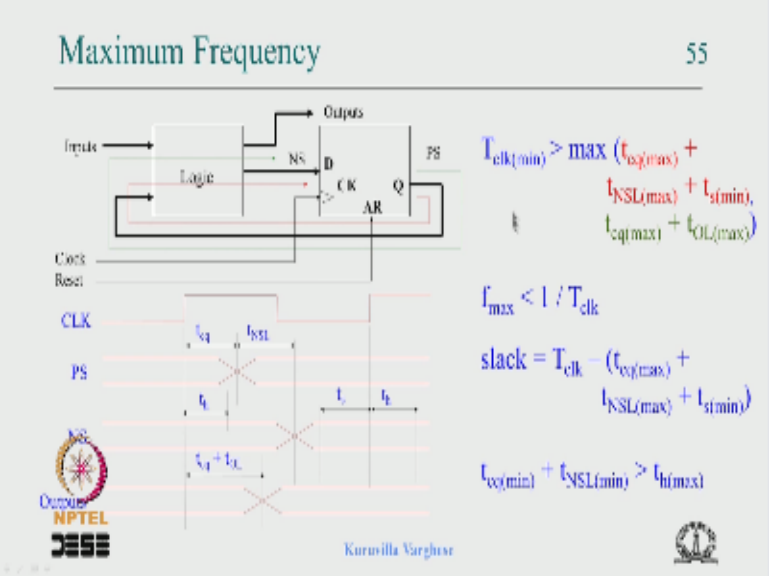
Digital Systems Design with PLDs and FPGAs
Kuruvilla Varghese
Department of Electronic Systems Engineering
Indian Institute of Science – Bangalore

Lecture-08
Case study 1

So welcome to this lecture of digital system design with PLDs and FPGAs. We will have a quick run through the last lectures part. The last lecture we have seen the timing analysis of the controller, how to go about designing the control algorithm using a state diagram and so on. So quick run through the slide then we will start todays part.

(Refer Slide Time: 00:56)

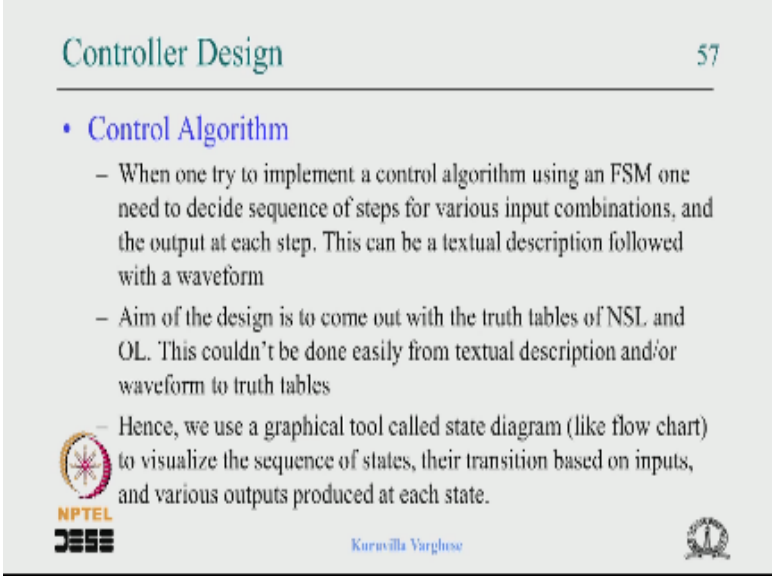


And we have looked at the maximum clock frequency we have to analyse 2 part, 1 is a register to register path and the register to output path and we choose the maximum as the minimum clock period and as I said that this output can go to some other register through a combinational circuit then one has to account the whole part from this register to register. But in the absence of that knowledge I have shown up till this point but you have to keep that in mind.

And similarly exactly like the synchronous counter case the whole time violation is this is a inequality the TCQ+Tlogic should be greater than the whole time, otherwise there will be whole time violation and if there is a violation then you have to increase the delay of the logic, so that because as I said everything is we are analysing with the same edge, so it got nothing do the clock frequency.

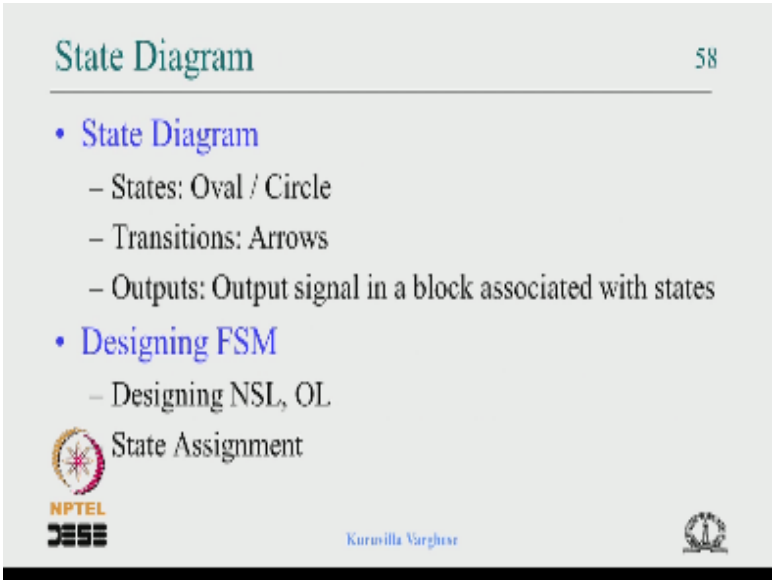
But in the first case we are our analysis is from one clock edge to the next clock edge, so if there is an issue with this is not satisfied you can increase the clock period or reduce the clock frequency can solve it. So whenever there is a slack is negative, the margin is negative you can decrease the clock frequency or increase clock period. But when there is a whole time violation we have to see how to increase combination delay in the path or do something else which we will see maybe some other techniques may be as we go along.

(Refer Slide Time: 01:06)



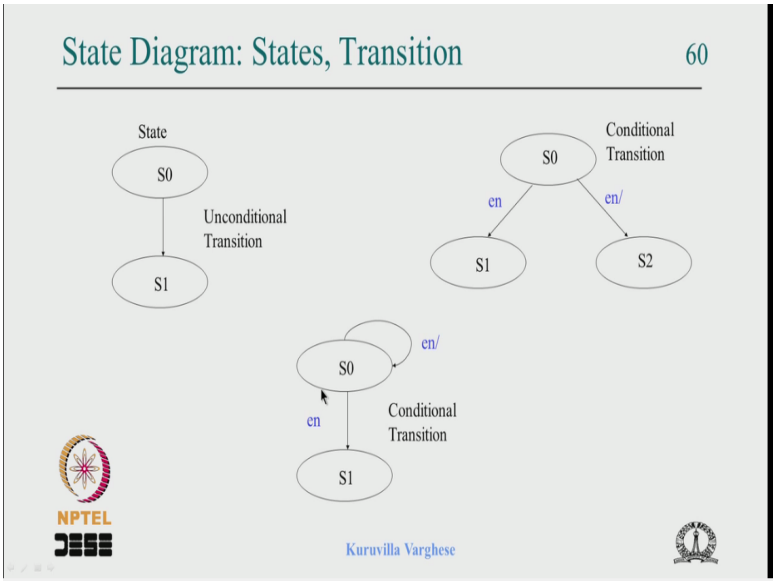
And we have looked at the controller design and many times when we design a controller we have a text kind of specification and from there we will have to drive from set of sequences then some waveform, but then there that may be quite a lot of possibilities we have work out then always humans are good at using graphical tools, so we use a graphical tool like flowchart called state diagram which captures the various steps in sequence of a stage.

(Refer Slide Time: 03:26)



And depending on the input it translation in each step of state what are the output, so that is basically what we are going to do so in any state diagram states are represented by oval or circle, transitions by arrows and output is associated in a clock with this oval, essentially will be able to once we draw that, will be able to come out with the truth table of the next state logic and output logic after the state assignment.

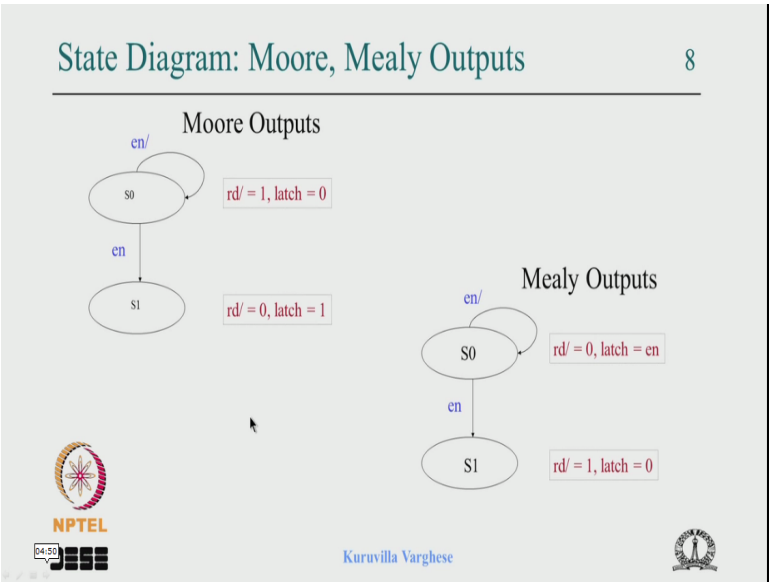
(Refer Slide Time: 03:40)



So we have seen some kind of unconditional transition going from one state to other state on next clock edge and here we have a transition depending on an input that means if the machine is in this state of step when input is one in goes here input 0 it is a branch in condition and very other useful conditioners at waiting for some signal to become active. So here as long as there is enable is not active you remain there the machine remain the present state is 0.

And when that becomes active it goes to next state, both are conditional transitions, so let us we have looked at output we normally specified output associated with the state and again in textbook cases sometime there is a habit of writing the output you know like input/output and all that. This works for very kind of symbol cases where there is only few inputs and few outputs.

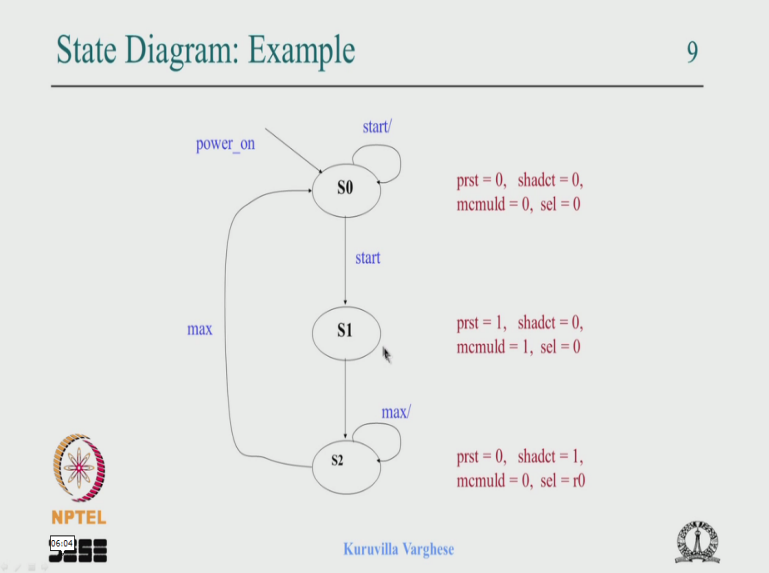
(Refer Slide Time: 04:56)



But when there are lot of output that cannot be done. So it is better to write separately, so this what is done here. So in this state these are the output signals as a direct decode of the state and in the state read bar has become 0 and the latches become 1 and in the case of Mealy output we write the output is the function of the input. In this case the input is same input used for the transition condition.

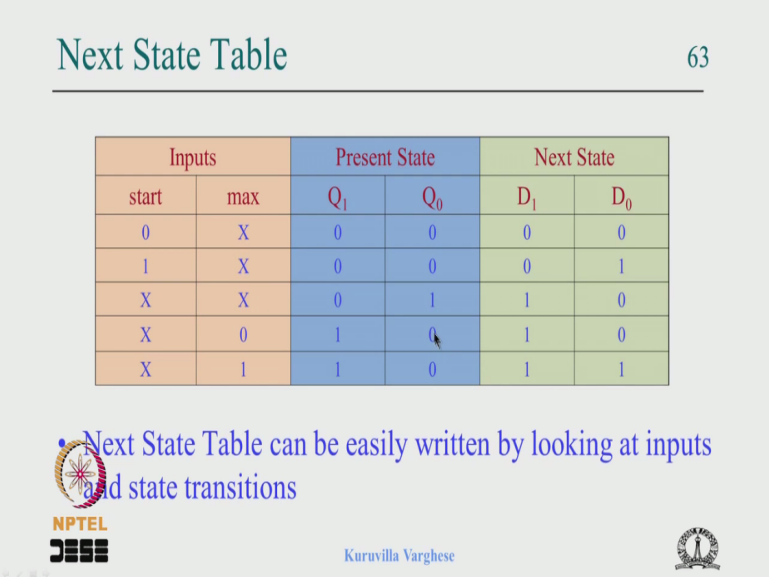
But as I said earlier you need not be can be another signal which is not an input to the state machine, so that captures the kind of the state diagram basics and we have looked at multiplayer kind of state diagram for multiplier data park and as I said normally like in software you come to a state where everything is inactive then wait for some signal, when you come to another state initial is everything.

(Refer Slide Time: 05:56)



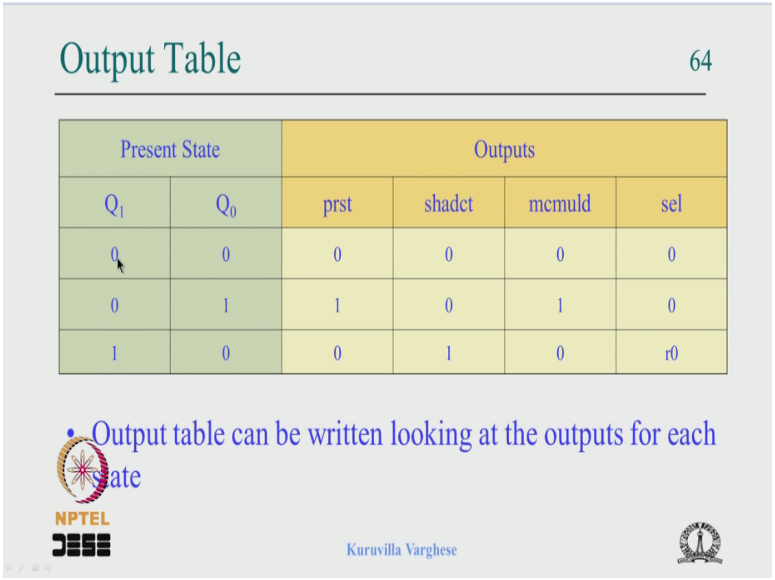
Then come to a main computation state and its waited 2 minute clock cycles and when the computation is done you go back and wait for the next operation and we have also seen that from the input and we have also seen that from the input and transition you can write the next state table and from this state and the output each output we can write the output table.

(Refer Slide Time: 06:16)



So from this we have seen how to write the next state table and moment you write that is confirm equation for d1, d0 minimise it and so on. But in our case we write this diagram in some hardware description language and the tool does everything, you know tool does the state assignment, tool does all these minimization and all that.

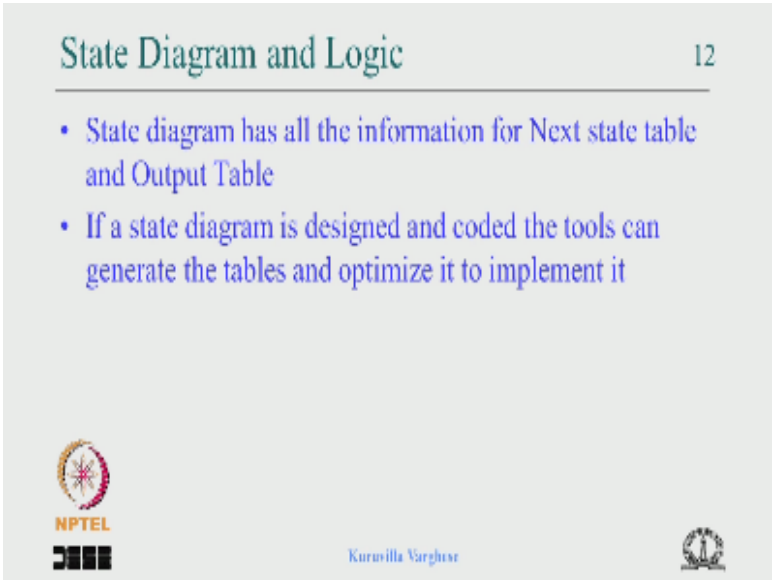
(Refer Slide Time: 06:39)



And output table in our case is a function of the present states for output we write as a function of the present state and then you can form equation for each output minimise it implemented again in our case we will write the state diagram in describe the state diagram then the tool will take over and do all the kind of minimization and implemented. And that a second.

And I also said that this is this next table and output table is kind of equal and to the state diagram because all the information for this is there. So the moment you draw the state diagram the game is over, the design is done, rest is a kind of automatic process which the tools can do very well.

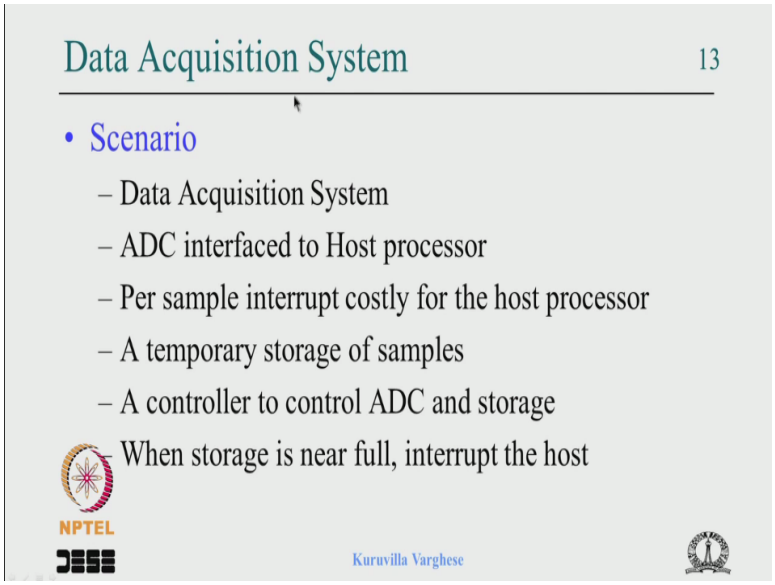
(Refer Slide Time: 07:36)



So with that I think we will move to a case study to basically this case study is too kind of illustrate the controller design than the data part design. We have seen at least the data path of the CPU we will take the data part design in another case study when we kind of little more proceed further. At this point the case study I will give more importance to the controller.

And as I said it is a simple case study, but it is realistic to it, it can be used in real life, it is not that you know something very simple I took up and most often most text book show something like a traffic like controller is a kind of favourite example for most people because everybody experience that case and one knows all the specification and the clearly from the kind of day to day life experience.

(Refer Slide Time: 08:54)



But whatever I am going to discuss also for an engineer is a familiar scenario very simple scenario. So let us go to that case study. So the case study have taken as a data acquisition system so in a normal like you want to capture some analogue signal and take the digital samples and do some kind of say Digital Signal Processing that is our system ok. So normally you have a ADC maybe single channel multichannel.

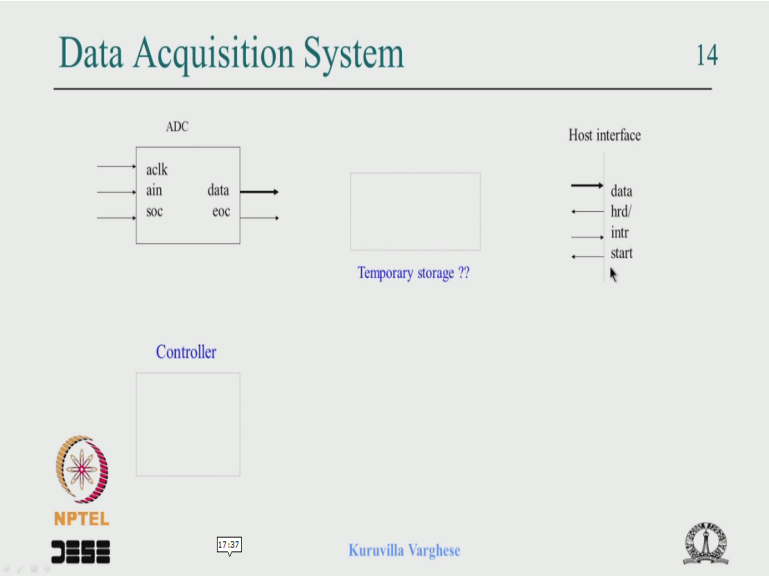
We do not care much about the number of channels, but we assume that is quite of fast kind of ADC which captures high frequency analogue signal ok. So now normally the ADC interfaced to a microprocessor or microcontroller, some kind of microcontroller or microprocessor and what is done at the processor you know tell the ADC to start the conversion.

Then when it is done it is read through this example I workout is little bit told maybe in the present day interfaces are not parallel interface like I to see or SPI interface but I am showing a parallel data from an ADC. In this case but this can be translated to the whatever is the latest interface is available. But we are assuming that this particular process is used for it is an embedded system and processor is kind of working on many things.

And now if ADC acquire sample and interrupt the processor every now and then like suppose it is an 12 bit ADC, everytime 1 sample is acquired or few samples are required is processor is kind of interrupted then everything slows down. So our idea is that we design a controller or design a system when our controller will give the tell the ADC to do the start of conversion and when are the ADC say this done and conversion is done.

We will store the sample somewhere in a temporary it will be stored in some place and then that storage is full we will interrupt in microprocessor and or the controller will interrupt the host CPU and the host CPU come and in a bust okay read the whatever was in storage. So that basically 2 kind of free the process from kind of pew sample overheads ok because if there is a real time OS running on the process this interrupts can be costly. Because there is a task switching to make this act even read and then continue with the other task. So all that is avoided. So that the basic scenario, what we are going to do.

(Refer Slide Time: 11:56)



So I will put a picture. So this is the end is always better to write a picture but draw a picture which brings in clarity ok, so we have an ADC as I said, I am showing a bit old ADC with a kind of parallel interface and not a command through SBI or I2c. So here you have an

unlocked, analogue input and there is a start of conversion to the ADC and when ADC is completing the you know the conversion it gives an end of conversion.

Then the idea is that we put this data into a temporary storage and this is a host interface host will give a start signal to start all operation, then we start this business, you know continuously keep on converting and putting into the storage and when the storage is becoming full we interrupt, the system will interrupt the host and host will read the data, the output of the storage read the data using this host read signal.

So that the basic idea, so we have to our data path, you know data path is very simple, we need a temporary storage. So we need to read the data from here, put into a temporary storage and the output of the temperature goes to the host and all these control signals you know the start of conversion, looking at the end of conversion, giving an interrupt and so on should be done by the control, all look at the start signal and so on ok.

So the question is what is the best kind of temporary storage and definitely it has to be a memory ok that is very clear, but you see that this memory and normal static RAM if you put it has 1 address bus and one data bus okay, but this ADC side has to write and the host has to read ok. So a single port may not suffice in the case like if you have only one address bus and 1 data bus that has to be switch between both sides you need some kind of multiplexing and it can become complex.

So normally one would think that ok why not use a dual port Ram, so in a dual port RAM you have 2 data buses and 2 address bus. So we can be used one kind of bus here to right and 1 kind of bus here the second port to read ok. But if you think for a while a RAM will allow a random access okay that means this input side can write any location and output side can read from any location.

And it allows a kind of other way I like it from this side you can write and this side you can read. But in our case if you look I read the data flows only in 1 direction, it does not flow in another direction and we do not care about the random access and all the light. We are getting the analogue sample you put it in a sequence the first sample comes a second here, third sample here and so on and hosting to read in the same sequence.

There is no point in reading the fifth sample, seventh sample, second sample and so on. There is no random access which is a one way flow, so BP RAM is a costly solution most features are not required. So if you again if you know the various memory technology is the best solution for is that if u is a one-way flow and its first in first out kind a read like you write first and you whatever is return first is read first.

So what is required is a FIFO here ok, that ideal solution because FIFO you write in one side and read from the other side and there is no addressing like undressing is implicit in a FIFO because there is a right address register on read address register or you can say right pointer or read pointer. At the beginning both is the first location. So you start writing then writing pointer increment, you start reading read pointer increment.

And definitely the read should not overtake the write and the write should not kind of come below like you should not overwrite. So there are some control signal like NTfull and all that FIFA give two kind of synchronise that that means when it is empty you do not read, when it is full you do not write and so on ok. But there is a FIFO, so FIFO is a best solution in this case, so let us try to put it.

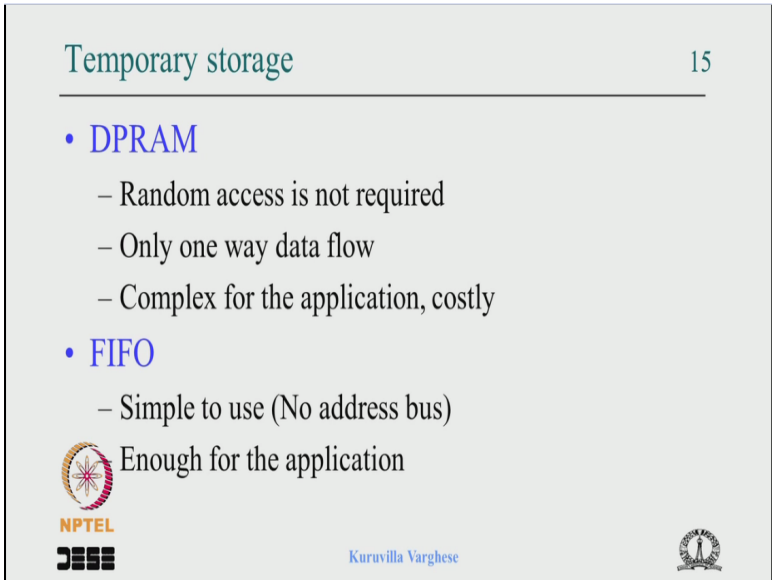
And you know that the controller get a clock and the reset. The controller has to give the start of conversion and when the end of conversion comes the controller should know that and the start signal from the whole should come to the controller to start the whole operation. And this FIFO need a right signal that has to be given by the controller and the FIFO this data can be connected to the data input of the FIFO.

This data can be connected to the data output of FIFO and the rich signal of the host will come to the FIFO read and how to give this interrupt signal ok. Now the FIFO has various signals like full, half full and 3 fourth full and all that ok. Now like if you have a full signal and if you connect the to interrupt maybe dangerous in the sense that when it becomes full is give the interrupt.

And the CPU has a OS it does all the task switching at least it has to before going to the interrupt service routine event with symbol system it takes care of you clock cycle by the time it comes to read it over it because we are continuously doing this operation suppose the host is not popping it. So it is very safe to use a 3 fourth full signal as interrupt, so by the time

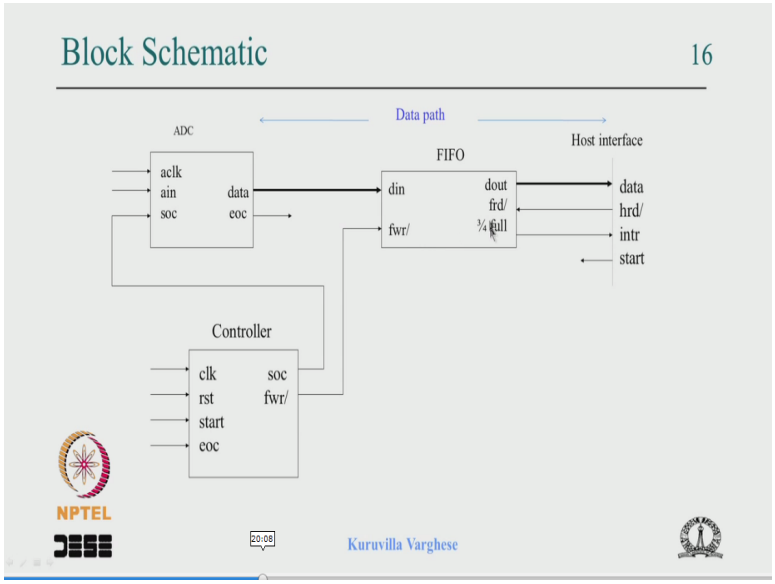
the hose comes here. Then the data is not know the temporary storage is not yet full and nothing get lost in the process. So that the basic kind of block diagram.

(Refer Slide Time: 18:31)



So let us look at it you know no that is this is what I declare I mean I have told that DP Ram you can put but the Random Access we do not require one way data flow, DP RAM allows to a data flow and it is quite costly because of multiple ports and it is too complex for the application the ideal solution is a FIFO, it is very simple to use addressing is implicit and it is enough for the application.

(Refer Slide Time: 18:57)



So I have put whatever I have told ok. So you have a controller which need a clock and reset. The start signal comes from the host start, so when the host start whole operation will start. The start of conversion is given by the controller, the FIFO right signal is given by the

controller because controller knows when is the end of conversion coming here the end of conversion is coming to the controller then we will give the FIFO right.

So this data is connected to the data input. This output data is connected to the host in the face the the host read is connected to the FIFO read and three fourth full is given to the interrupt. So that when it is three fourth full the host can come and read the signal ok. Now we will make some assumptions of so one thing is that like when the interrupt come and the host comes and try to read it.

By the time maybe that we have the controller has written or the ADC has written quite a bit of data more than the three fourth. So the question is that like we are not giving any other handset signal like NT or full to the host make it keep it simple. So the one question is that how much when an interrupt comes, how much the host should read okay. So we are not sure like but be sure that when the three fourth come over 75% is full.

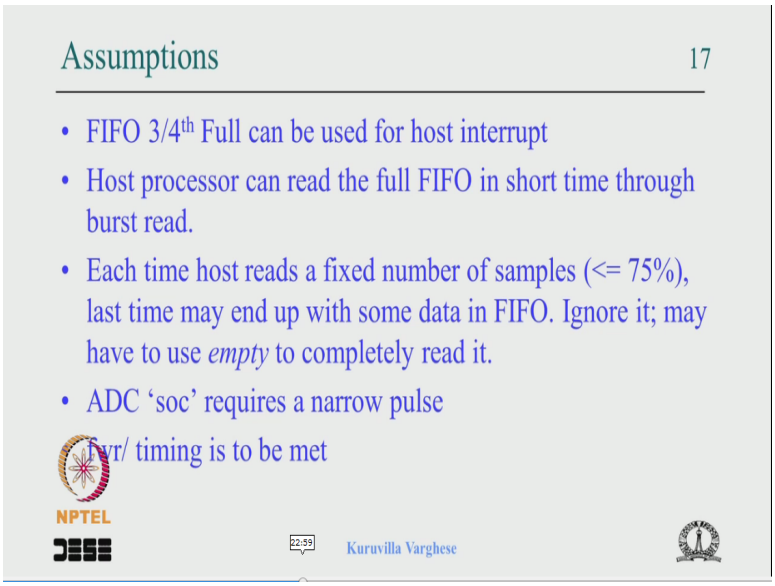
So let us assume when an interrupt come the host will come and read less than or equal to 75%, but fixed like agreed 70% or it be 70% does not matter ok. So read somewhere around less than 75 all the everytime the fix the more ok. And any way that there will be something remaining which gets pushed when the new data comes again it gets full and it comes and read the 75% ok.

And maybe the last time you know the the when the whole stop it and may be some there are some data remaining there we do not know how to empty it unless there is you know other hand side signal, so maybe that is okay like you say stop, you remove the start then the interrupt comes or interrupt does not come, some data may remain there it is okay like we use some data there.

The last cycle once it is known this can be taken care, but no issue. So that is so first assumption we make when an interrupt comes the host need something fixed amount less than or equal to the 75%. Second assumption we make is that we will not impose any constraint on the start of conversion pass that means that we say some kind of narrow path is given to the ADC that work.

You know that one assumption we make and but when you write to a memory write to a FIFO we need a definite with also to it, some minimum pulse would required for write we will try to meet that ok. So these are the assumption we make, one is regarding when the interrupt comes how much to read, the timing of the SOC pulse and the timing of the FIFO write signal ok.

(Refer Slide Time: 22:46)

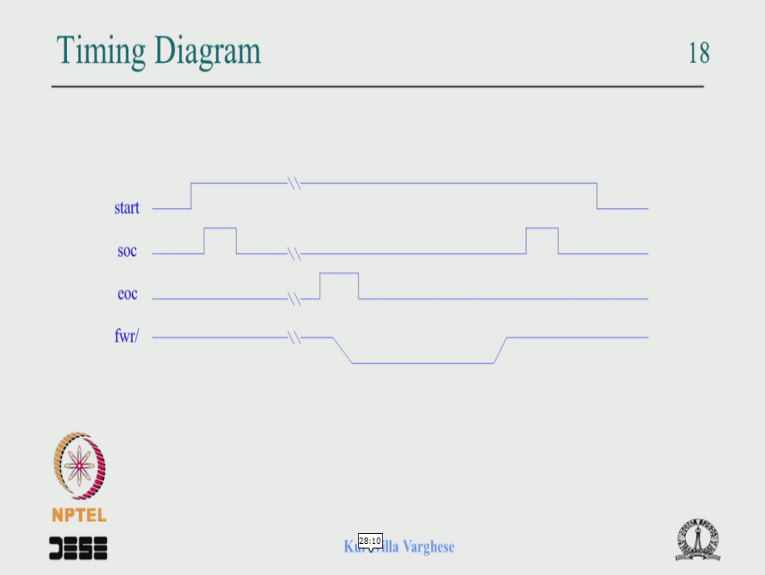


There is no much constraint on this, some narrow pulse will do, but here you need a definite password for FIFO to write ok. So that assumption we make now, so that is written here like host possible read something less than or equal to 75%, the SOCs are narrow pass and from right before right signal timing has to be met correctly minimum it has to be maintained ok. Now what is the next step you know like we have put a block diagram.

So the question is that what is the next step and can we go to the kind of control algorithm, the state diagram, no I think very important thing is that we have to draw the timing diagram, so the input and output relation from this description and referring to the various systems we use, maybe you have to refer to the ADC data sheet to get the ADC timing.

We have to read the FIFO data sheet, or FIFO we have to know like if you are designing the FIFO yourself you should know what are the timing parameters of the FIFO. Then we draw a timing diagram showing the input, output timing waveforms. So that is the next step.

(Refer Slide Time: 24:02)



So let us write the timing diagram so I have shown the timing diagram in the slide. So you are you know you have at the beginning the start is low or inactive. So all the signals are inactive, SOCs 0, end of conversion is kind of 0 and the FIFO right is high because it is an active low signal normally the right signals are active low it is kind of inherited from the TTL kind of logic because where the you know the active voice signal does not so smart current.

And active low signal is the one which shows in the current show most of the time the right is inactive to reduce the kind of current this was made in active but with the CMOS that is not big deal, but anyway this is kind of inherited from the previous technology. So if you look at that the timing diagram when the start goes high then the controller detects it and give an narrow start of conversion.

The moment the start of conversion is given and this shows the continuity you do not know how long it take the end of conversion to come. So this show some kind of continuity maybe it is continuing and then the end of conversion comes the moment and of conversion comes the controller has to be right signal of certain way to the FIFO and the FIFO gets return and we terminate.

The controller terminate the FIFO right and after that again look at the start signal and if it is one then give the start of conversion and repeat the whole process, so this is the kind of, so that is the iteration we do this old process starting here it is continued if the start signal is at this point so that is a basic kind of way for waveform and where we derive the control algorithm ok.

But then next question to ask is that how do we generate very precise timing for this before right ok. So here this start of conversion is a simple thing what we do is that we come to a state at the beginning SOC was 0, we when the start signal come we can come to a state make SOC1 and transit to another state and make a SCO 0. So for 1 clock period duration when that state machine is in that state, this SCO will be 1 and you get a narrow pulse ok.

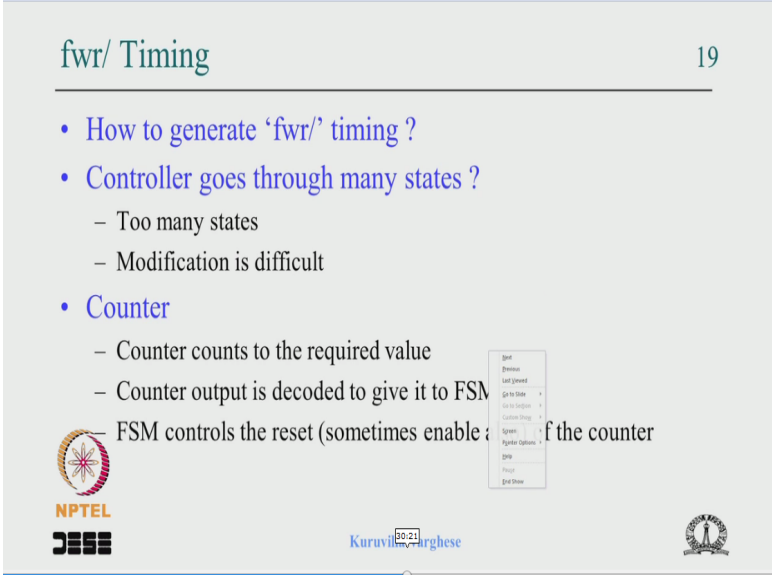
It depends on the clock frequency which we use for the control but this false can be off 1 clock period duration if you want to clock period duration make a SOC 1 for 2 states when you get 2 clock period duration. But we assume that there is no great constrain one state duration is good enough for this case but the question is that how to generate a pulse of certain size ok.

So I think you from what I have described 1 ways like when the end of conversion comes go to 1 state make it low, now if you know the clock period of the controller suppose it is say the its 5 nanoseconds and you need 25 nanosecond bit for the FIFO right and one thing to do is at the first come to a state where the FIFO right is low, transit next state where FIFO right it is low and go on doing that for the 5 o'clock period or 5 states.

Then you will get a clean pass, but then this is quite kind of compress ad suppose we kind of update this design with a faster FIFO then the state machine has changed, the controller has to change maybe only 3 states are required are 5, so the question is that can we remove this kind of timing from the implicit state which we add and move it elsewhere okay.

Subsystem we add that will keep track of the kind of width of the timing pulse then he'll be ideal because if there is a change in the width of the past week and modify that subsystem easily maybe there is some flexibility to do that, so that is what are shown here so can I have a doing it is at controllers going through many states which match with the width of the right timing.

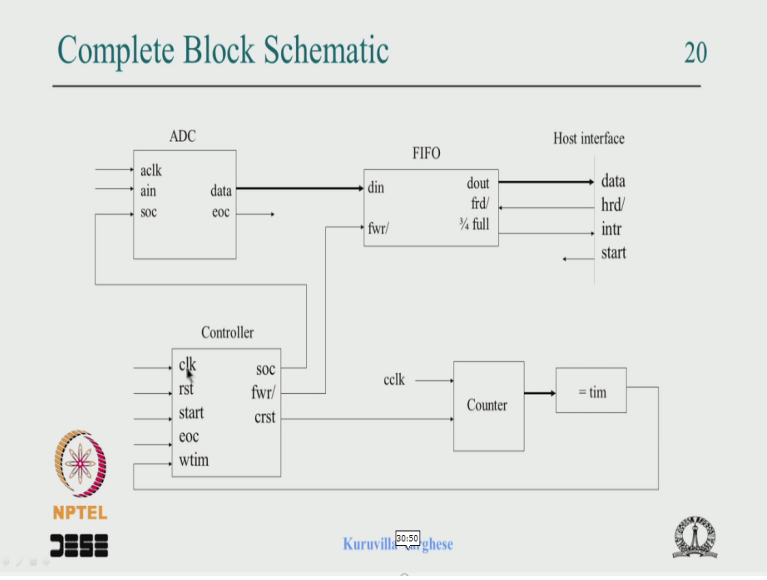
(Refer Slide Time: 28:40)



I might occupied too many state and modification is difficult I like you get a slower FIFO then you have to modify the state diagram and redesign the controller. So why not we use a counter, we keep a counter ok. So what we do is that we keep a counter and when the end of conversion comes it goes to stay and make the FIFO right low and start the counter ok. Now wait for the counter to reach the particular count and that we can decode using a decoder.

And that signal is given to the controller and when the signal comes we go to the state machine go to a particular state, new state and terminate this**,** this FIFO right to be 1, so this is more elegant because the counter maybe likely be used 8 bit counter, so you can choose way to various kind of a count can be decoded ok. So your flexibility of choosing many counts, many values and only the decoder need to change ok. If you at the beginning itself depending on some the range of the access time of the FIFO.

(Refer Slide Time: 30:26)



You choose a counter with to accommodate kind of all kinds of possible rangers in future then only the decoder need to change it is very simple and decoders many times and AND gate with inverter so it is very simple to redesign. So that is what we're going to do that. So now I will show the modified diagram with the subsystem. So everything else remains same, now we have account which is outside our data path.

And which stock by or maybe the same clock or a derive clock from here are normally there is no reason to use any other clock then the controller, maybe I will touch upon and why it is. So but yeah there is no need to use a low clock then control o'clock because unless it is available in the system ok. If there were lower clock then the controller is available you can use it.

Otherwise I suppose this is 10 megahertz if you plan to use 1 megahertz anyway you need a clock divider which is nothing but a counter so that division can itself happened here ok not a big deal. So and there is no point in using a high clock then this because this cock is lower which not be able to catch again we will discuss that business later and you see the counter after reset which goes from the output of the controller.

So normally this counter is kept reset by the controller and we can use an enable for the count also since we have not kind of learned how to bring enable analog I just keep very simple scheme that is capitalism and when the controller need this reset is removed, that reset is made 0, then start counting and this is a decoder equal to time when you say equal to sum fix value it is a decoder.

And equal to a variable is a comparative, so when you see equal to some numerical value do not think that it is a comparator because you know the comparator is an exclusive OR gate comparing but when you say equal to 3 equal to 10 is nothing but I like you say 4 lines are there is a equal to 10 means that it is decoding 1,0,1,0 pattern. So you can imagine that one then there is a inverter.

Then one inverter going to an AND gate that output is going to the controller. So the idea is that the controller gives the start of conversion it is waiting for end of conversion when it comes it does two things, it makes this FIFO right low, the counter reset is made low, it was 1, it is kept reset and the counters start counting and when it reaches that this particular kind of pattern.

And that pattern is chosen so that it matches the FIOF right width and discuss the signal to the controller this goes high this is 0 and then it terminates this FIFO right and weight for the start again. So that the basic business we added subsystem now so what next step up to do is that we have to kind of update the timing diagram ok. That is very important all these are kind of most people kind of hate this kind of very systematic approach.

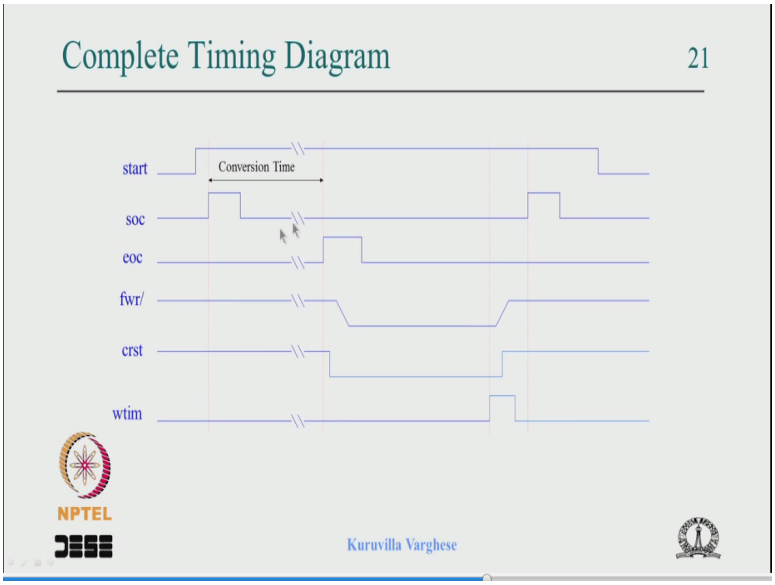
They think that is very pedantic to draw but you do not do you can make mistake and in a complex design where multiple teams are working on kind of on a project then ah if things small things can make a difference you integrate everything then you find some silly mistakes cropping up many times in complex design verification is done by somebody else and it goes to full cycle goes to the verification then come back with minor correction.

And lot of time is wasted if you do not plan things properly, so anything complex you plan it properly take time to plan take time to go systematically then there is less kind of debugging, less kind of errors cropping up but only thing is that you may not see many things concrete other than the paperwork like till towards end of the end part of the project which can keep some people tense stop.

But you need not worry anything complex the most activity will happen when towards the end of the project then once you do everything plan properly bring things together at the end it becomes quick need and easy in the rating than you know kind of going through tunnel

cycles of debug and iteration and things like that not that you don't have to debug but then if you plan properly there will be less trouble waiting for you at the end. So let us move on let us see the updated timing diagram.

(Refer Slide Time: 35:34)



So this is the latest on to the slide so everything remains same you have the when the start comes start of conversion is given by the can dollar, wait for end of conversion and essentially this is the conversion time from the start of conversion the end of conversion and then the controller does two things FIFO right is made low and the reset is made low and the counter starts counting.

And the control of wait for this right time signal ok, the right width time signal when it goes from the Decoder the reset is put back so that the counter is reset and inactive and FIFO right is made in active by making it 1. Then again check for the start and if it is at is ones at the start of conversion and whole thing repeat ok. So we have come to a kind of good point to drive the control algorithm.

And looking at this waveform you can derive the control algorithm, it is very easy now let us derive the control algorithm by looking at the timing diagram. So like let the state machine come to a starting state okay and all the signal, all the output signals are inactive SOC 0 this is a input signal before it is 1, reset is one and this these two signals are the input to the other state machine.

So which is what is look for transaction. So I given the start end of conversation and this right time is input SOC to FIFO right and the sea counter reset is are the outputs maybe I could put different colour but then anyway you understand that. Now so we are in a starting stage with this inactive condition SOC FIFO right and counter reset is inactive and at the start state we are waiting for the start signal.

So that means the start signal is low remain in the same state starting state and if the start signal is one transit to the next it up on the clock period clock active clock head, transit to the next day and make SOC 1 ok. Now we said we need a narrow pulse so way to generate is that next clock period transit to third state and make a SOC 0, so you get out of the controller you get pulse SOC1 clock period duration pulse as SOC pulse.

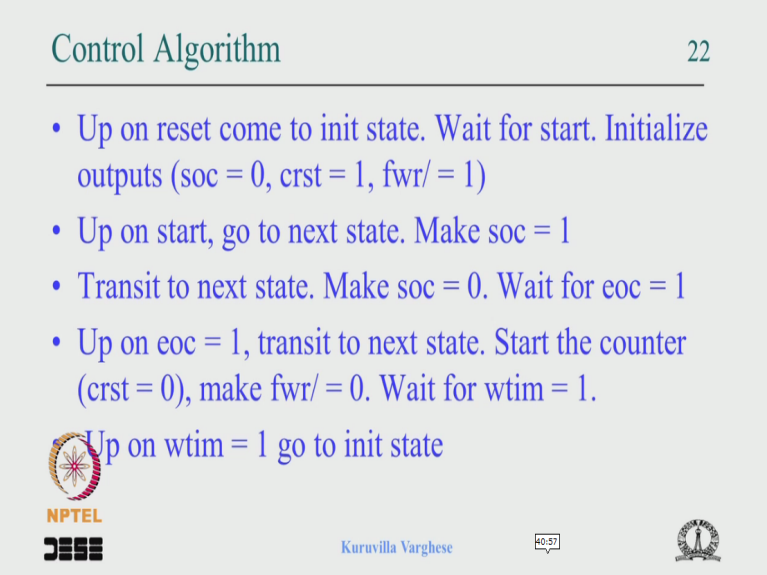
Then ADC start you know converting then the state machine is in the third state where the SOC was made 0 that the same state in wait for the end of conversion to become high that means remain there like a wait and go remain there as long as end of conversion is low when and of conversion comes high go to another state ok and make this FIFO right low and the counter reset low.

Now the counters start pounding, so understate what can do is that it remains as long as this signal is low. So it is waiting for right time signal to go high. So if it is low remain there and when it comes I go to the next state and make it 1 and FIFO right is 1 but you know that that next state can be the starting state itself because if you look at in this state the start of conversion is 0.

FIFO right is kind of one, reset is one so which is equal and which is same as starting state, so we can go back to the starting stage. So that the control algorithm, so looking at the waveform one can drive the control algorithm can write it in words once again starting stage make the output in active, wait for the start signal when the start signal come translate to a new state make the SOC 1, transit to another state make a source is 0.

Wait for end of conversion when the end of conversion comes transit to another state make the FIFA right low, the counter is at low, wait for the you know right time when it comes high go to the starting state and the whole thing is repeated.

(Refer Slide Time: 40:22)



So that is the control algorithm you can write it down upon reset come to an in each state wait for start in each slide output in active upon the start go to next page maker SOC 1, then do not wait for anything transit in next state, make SOC 0, so that SOC 0 is 1 here, 0 here. So you get one clock period pulse here and wait for end of conversion to become one and then upon end of conversion transit next state.

Start the counter make it 0, make FIFO right active and wait for this right time signal to go 1, when it becomes 1 go to the starting state. So very simple, very very clear and you might wonder that in a complex case writing this is it worth I would say writing is worth because many times writing allows you to think of you right as you draw waveform, you tend to ask questions,

Some question is come to my come to the mind and you will be able to address that but if you most people what they do is at either they do not draw the waveform and start writing the code ok from the head and that is gone to mistake but anything you write it down at the mind works if you are thinking and the logical something logical is missing it comes to the mind and then it can kind of you can sort it out okay.

Even sometime discussing telling the same thing whatever is known to another person sometime the process of telling brings this question to the mind or sometime you know question from the over is listening maybe they have not understood things properly, but doubt clarification can figure some answers your mind. So it is very important that you worked on the paper discuss some time speak aloud.

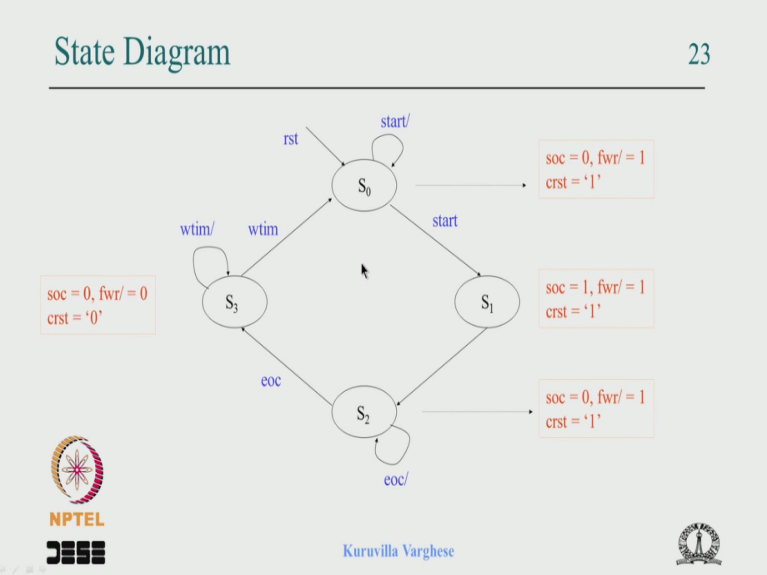
So that these things come to the mind ok of course this is not my kind of topic it is something do the cognition, but then that is related to education that is why I am I am pointing it out because people stay away from this kind of simple steps and waste lot of time and it is very costly many times whatever maybe you have a public limited company, kind of designing a chip but that is the money of the shareholders.

And if you make a mistake if you do not need the deadline lot of money is wasted, it is do not think that it is your company's money, your money, it is the people's money. So that you have to be responsible as an engineer many times you do not feel responsible sometime a doctors treating a patient and one thing that is their daughter is a medical doctor say kind of more has more responsibility than engineer.

But I think engineers should you know kind of understand that we are responsible for the problem solving, how efficiently we are solving, how cost effectively we are solving, very rarely that a n a system which is designed by engineer fails in the field and the engineer is you know kind of drone to litigation many times, it does not happen but it happens in the medical profession.

So one has to keep this is a work ethic to the kind of address issues. So let us move to the state diagram, now to implement the control algorithm we have implemented like we have written down. So let us look at the whatever we have discussed how to draw the state diagram.

(Refer Slide Time: 44:21)



So let us move on to the slide, so see at the other power on you come to a starting state okay S0, then at that stage we make the signals are inactive, SOC 0, FIFO right is one, counter is reset one ok that output of that state more output and then here we are waiting for the start signal. If start signal is low remain there, if start signal is one go to next state. We know that we have to make SOC1 rest all is remain same SOC made 1.

Now it transit in next state because you want pulse on SOC and SOC is made 0 rest all things are in same and here the ADC is converting start converting wait for the end of conversion. So if end of conversion is low we remain there upon the end of conversion we go to the next state and what we have to do is that we have to give the FIFO right signal we have start the counter.

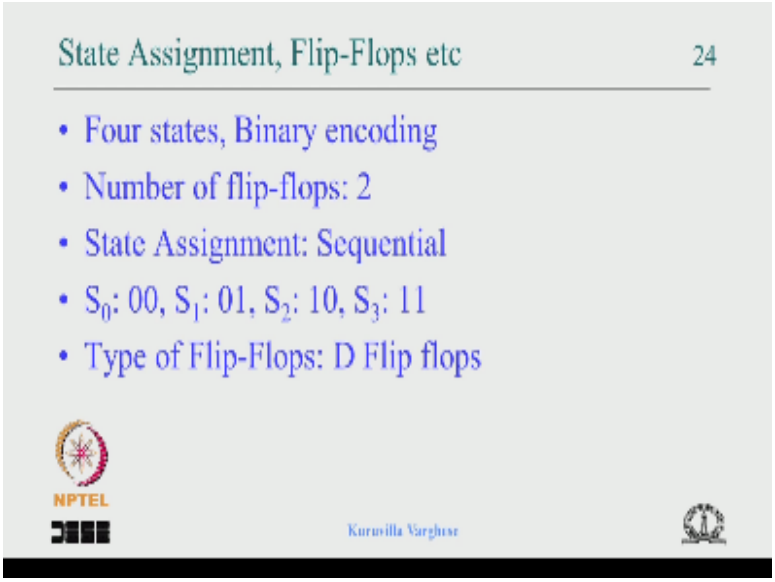
And that is what we do before it signal is active the counter is active the counter is counting, so we are waiting for the Decoder output to come. So the right time is low then remain there right time is high go back because then everything is remain same and wait for the start signal ok. So you should like symbol come to start you know starting state make SOC 0 wait for start signal to become active.

When it becomes active come to next state make a SOC 1, go to next state SOC 0, wait for end of conversation when it comes start the counter before it is active, wait for the Decoder output and when it comes come to the starting state and you know repeat if the start is low. So one thing is that you see we are kind of looking at the start signal only at this stage, so it is a very good thing you know it is with the logic become simple.

Because if you are looking at the start signal every state the next state logic will become complex but it is a beautiful advantage saying that we started to whole you know start of conversion process, even if start goes low in between we do not care, whatever was started is completed. So sometime making keeping things simple will result in very elegant solution like that is evident in the timing diagram like here since we are only looking at the start at the beginning.

Even if it goes in between you are not stopping anything, completing it and then stopping the whole thing ok. So keeping simplicity sometime can result in really elegant solution which sometime people saying that even if half way the start is become inactive our system make sure that it is completed unsolved but we know that it is a very simple decision which has resulted in some elegance.

(Refer Slide Time: 47:39)



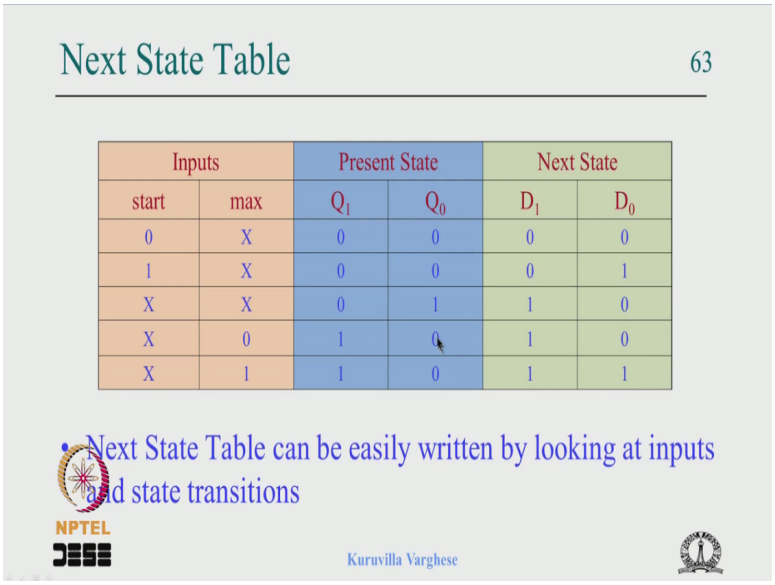
And here then we are 4 states so we have we can do binary encoding we number of flip flops are too then we can do the state assignment because too who developed the next a table and output table we need to know what kind of flip flops you are using for the transition and how many flip flops in the sun so what is the state of the flip flop. So we will assign a 000 S1 0 1, S2 1,0, S3 1,1 and we will use D flip flop.

You know that if user JK flip flop we can use that maybe we can see that later, but we assume that you flip flops are you so that the transition is same as whatever you want to transit same thing you give it to the D. Then you get the proper transition. So now we are able to write the

next state table and output table. From the state diagram I hope the state diagram you remember.

So we move on, so o not forget this is a finite state machine you have 2 flip flops here, giving the present state and input comes and depending on the important present state we are you know from the state diagram we are deciding this and in each state we are using output logic you know creating output by output logic. In our case is all are more output, so there is no kind of this line is not necessary.

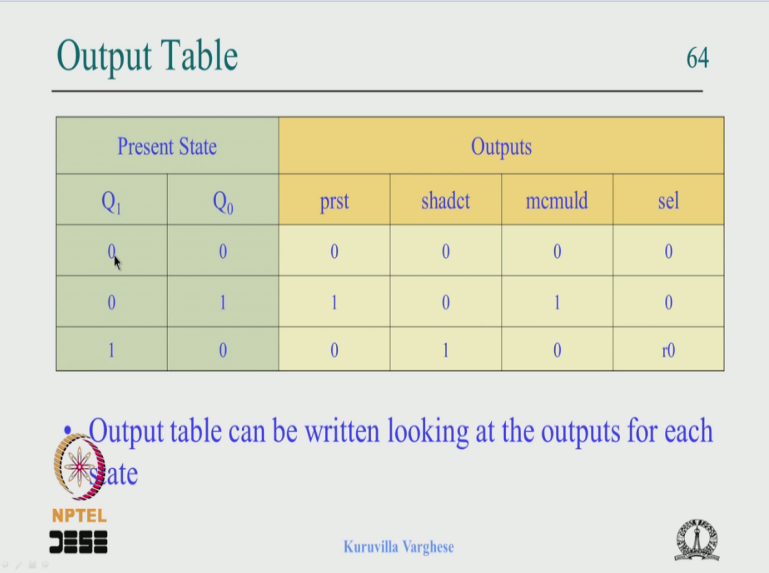
(Refer Slide Time: 49:06)



So let us developed the next state table, so next state table have a present state as input, the real input as inputs ok to the next state table. So we write the next state as a function of the present state and input, so at the beginning 00 if start a 0 is these are do not care remain there, if start is 1 go to the next state 0,1 in the 0, 1 irrespective of the condition transit to third state 1,0.

And in 1,0 we are waiting for end of conversion, as long as it is low remain there 1,0. If it is 1 you know 1,0 is translating to 1,1 and in the 1, 1 state you are waiting for the right time Decoder and if it is kind of 0 remain there it is one go to the starting state. So that the simple and we can form the min terms of the D1 and D0 minimise it get the equation. So D1 and D2 are the function of the present state and the input.

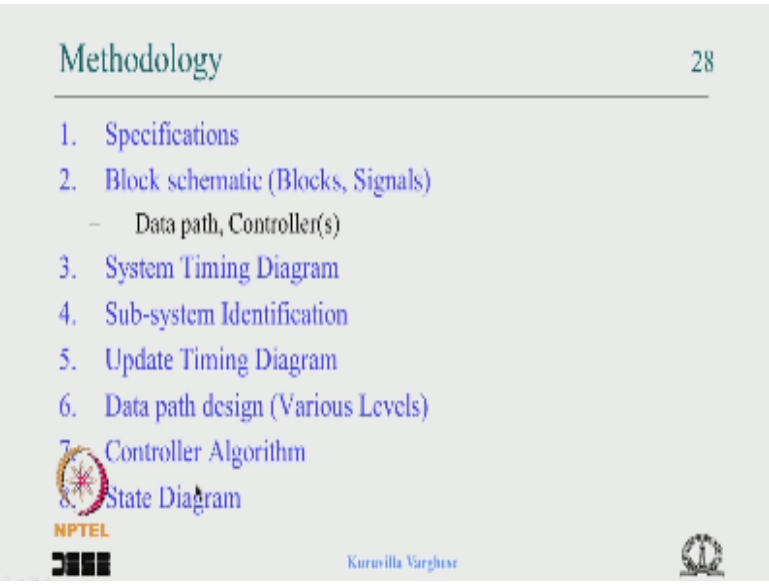
(Refer Slide Time: 50:05)



The design is done and similarly the output table we can develop you have the present state output 3 output in the first state the sources SOCs are rest of 1, next SOCs one, in the third state we make a SOC 0 and then the end of conversion is start of conversion has happened and in the last state we are making the reset and the FIFO right active that is all and output can be you know inadequate can be for minimise and so on.

So hear the outputs are function or the present state that is what is shown, so that the whole process of designing are shown, so let us recapture that the whole process in step, so that you know clearly what is happening.

(Refer Slide Time: 50:56)



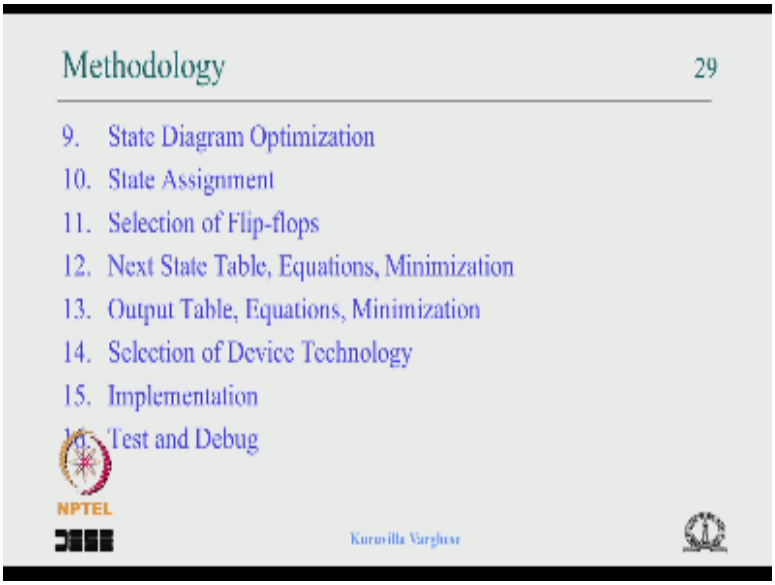
So the design methodology we have used, we have started with the specification which was written like a setup spec, then we do a block schematic of the block diagram and signal and

we kind of argue for some block like temporary storage what is a temporary storage and so on. So basically there is a data path which is FIFO and controller. So in a complex design the datapath is quite can B complex.

And then third suppose that we have drawn as system timing diagram. Then some question crop top whether how to meet some timing diagram so we have timing requirement and we need to put up counter. So we had some subsystem to meet the timing then we have updated the timing diagram ok, then we have described of course we have to do the data path design we have assume the FIFO design is done ok.

We are not attacked that, but ultimately of to design the FIFO 2. In this case or use a FIFO as a block from somewhere or you have to literally design the FIFO. So that datapath has to be designed and in a complex case you have to go through a hierarchy top down design. Then we have developed the control algorithm from the timing diagram, then we are you know kind of the do the state diagram.

(Refer Slide Time: 52:18)



And after that we have optimise okay, in our case we did not do much of an optimisation, but there was an optimisation like if you remember from the last state we said we could go to another state and make things like and then we realized that instead of going to a fixed state you can go back to the each state because those are equivalent okay. So in a complex case this may not be obvious.

You have a lot of states and their could be some states somewhere in the in the state diagram which are kind of equivalent and that can be brought together ok. So that is state diagram optimisation, you can remove the redundancies and we will see how these are done by the tools but what are the kind of algorithm to use to detect the redundancy and so on. So there is a step in state diagram optimisation.

Then we have to do the state assignment of optimising it was to assign the state assignment, we have to do the selection of flip flops, what type of flip flops are used, then we have to you develop the next a table depending on the flip flops from the equation minimise it, output table equation minimise, then ultimately we have to choose some device technology and implement, it implemented.

Then things would not go work at the beginning of to debug get, if something you feel that something is working at the beginning there is something wrong with you, wrong with the whole thing something has to go wrong ok, it is better things go wrong because at least in the learning process you can debug and learn things is not that you need not plan, but then it is better you know something goes wrong.

So that you can learn things fall and ultimately have to document it this is what you know I do not why I am talking about be the academic or industry, one it document everything properly, so that the people who are coming afterwards can follow it also. These are the steps which is now you realize that some of things these things are done by the tools ok. So let us look at it if you look at it all these things has to be done by the designer.

This is where the maximum creativities required you know you have to come up with an architecture, datapath, then identify subsystem, draw the timing diagram, come out with the datapath design, come out with the controller algorithm, draw the state diagram. Then you can describe that in hardware description language. All these can be done by tool, see up to here the tool can do it, and you need to give some directive to the tool.

(Refer Slide Time: 55:40)

And test and debug you can use some equipments to do the test you know debug and documentation maybe the tool can help you with some report for documentation. So steps 1-8 is done by the designer, 9-13 is you know done by the tool and directives and the next steps is done by some tools and equipments and the designer. So I have taken you through a whole you know case study starting with a specification we have seen how to automate of load the process by a design.

A data acquisition system basically we have described, the system came out with the block diagram the ADC the host controller and the temporary storage you have chosen FIFO and then we of interconnected that, then we have put the timing diagram, then we have identified the subsystem, updated the timing diagram and we have drive the control algorithm, drawn the state diagram.

And we have develop the next state table and output table. We are done all the process, all the steps in the design process, the initial part of to the state diagram data path design and the controller design is done by the designer where the maximum creativity that is where you have to spend time. Then you should know that tools you know it well and we will we will learn about the tools.

And then proceed just debug and documented. So we have been learning how to design the date of birth on the controller. There are lot of timing issues various issues in the controller to discuss and datapath design, but for the time being I will stop the digital design part and we will in the next lecture onwards we will have the basics of the VHDL. So that you are able to

describe the combination circuits and sequential circuit of some complexity using the hardware description language.

Then you will come back to this issue, so that that is fine and the and VHDL can go hand in hand. So next quite a few letters we will go through the VHDL part. So please revise whatever I have done till now for whatever reason I am sure that you will be you know you are seeing this video course but there is no point if you mechanically go through the lecture after the lecture like leisurely without walking yourself.

So please revise work yourself learn it well, so that at the end of once you go through the complete course you are a master in the subject, so I wish you all the best and thank you.