

CMOS Digital VLSI Design
Professor Sudeb Dasgupta
Department of Electronics & Communication Engineering
Indian Institute of Technology Roorkee

Module No # 08
Lecture No # 38
Clocking Strategies For Sequential Design - V

Hello everybody and welcome again to NPTEL online application course from CMOS Digital VLSI Design and we start of with today on clocking strategies for sequential designing. In our previous modules we have seen what do you mean by skew gutter and we have also seen methodologies for reducing skew and gutter in a architectural fashion by which you will be able to manage the performance without compromising the performance of the circuitry.

We have seen that how therefore we need to route your clocks in such a manner that the paths clock paths are actually matched in terms of R and C and therefore even if this is skew for individual path between two clocks if the skew is same then I can assume that they are exactly synchronized with each other. If you do not have you have problems on the constrains and timing that we have already seen in our previous discussion that you do have a constraint in timing issues.

Which primarily timing issues have been said the maximum frequency of the operations of the design is basically a big question. Now with this basic fundamental principles let us go to the last part of the sequential design module. Let us see what we will be studying in this basic module. in this, we have till now, if you look very carefully we have been concentrating on H triggering base designs so you have a rising edge of the clock and your data's are samples in the rising edge of the other clock.

And therefore the next data would be always should be available one set of time before the next rising into the clock. And that is it, and that gives you the setup time sorry that gives you the capital T value which should be fixed up. What we were looking now is basically latched based clocking.

(Refer Slide Time: 02:27)

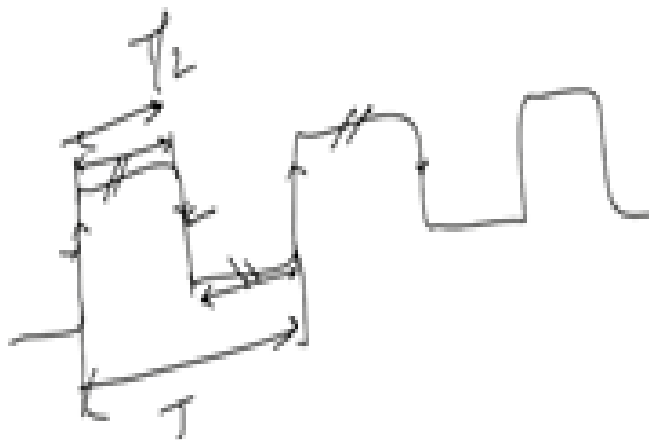
Outline

- Introduction ✓
- Latch-based clocking ✓
- Slack borrowing ✓
- Asynchronous design technique ✓
- Self-Timed Pipelined Datapath ←
- Synchronizer ✓
- Arbiter ✓
- Recapitulation))

So can we have a latch based clocking, we will look of course introduce to the subject which is basically the idea behind this clocking sequence then we will look into the latch based clocking which is here right. And then we will be doing slack borrowing, will explain to you what slack borrowing is all about. And then some, so this takes care of synchronous design techniques right. Then we go into the self.

Sorry Asynchronous design where in your clocks are not there later driven fully. And then you have self-timed data path pipeline data path approach which we will be looking into. Synchronizer, Arbiter and finally conclude the whole lecture right. So this is what we will be doing for this module at this stage right. Okay, let me come to what is latch based clocking now.


(Refer Slide Time: 03:24)



If you, as I discussed in the previous modules that whenever you are discussing an edge based this is registered whenever you have, got, this or this right. Latches when you have this part this part is basically called area where you are able to make your system go from high to low state right. And this will be the point where we will be accepting data so on and so forth. So it is basically depending upon the half the clock period. So if you are built in this configuration. T by T is basically the latch where you can hold the data or you can reject the data.

(Refer Slide Time: 04:03)

Latch-Based Clocking



- In a latch based design combinational logic is separated by transparent latches.
- Enables more flexible timing, allowing one stage to pass slack to or steal time from following stages.
- Add logic circuit between latches of master-slave flip-flop,
- Overall performance is increased. ✓
- Two phase clocking scheme is adopted. ✓

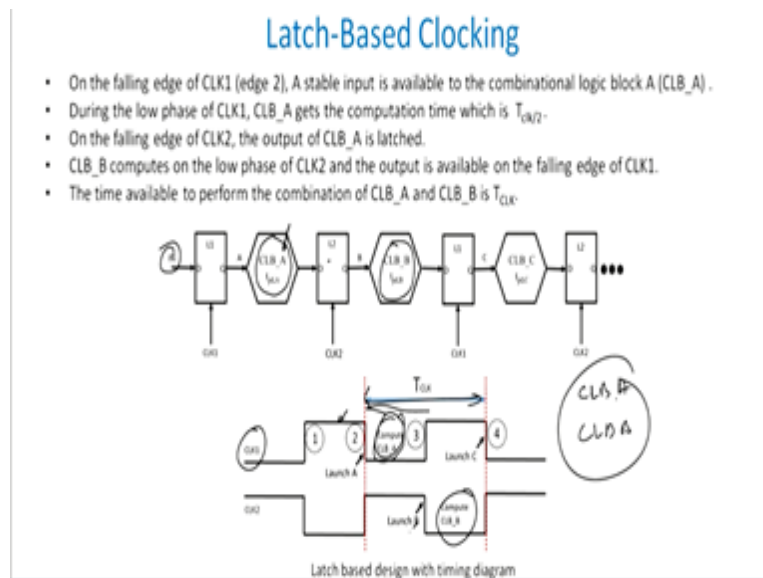
Now, if you look back therefore in a latch based design right, in a latch based design the first bullet point the combinational logical block is separated by transparent latches so unlike the previous cases where you had registers being inserted between combinational logical blocks now we will have latches being introduced. And these latches will insure that they will be transparent they will be a particular period of time, they will be opac during the period of time. Now one important point in a latch based design is that, we will learn it later on is that you will be able to borrow some of the delays of the previous blocks.

For example, if a latch gives you signal and the signal is passing very fast then I do have some time left before the next edge of the latch starts then again use that time to use other computation. So I can therefore borrow this slack and use it for all other purposes right. So, this is what we are trying to show and we will try to show that enables more flexible timing, allowing one stage to pass slac to orsteal time from following stages.

So if you have multiple stages you steal time from the previous stage it is done, already it was very fast so then I take time from him as a sort of loan sort of and do the processing in the particular zone. And what we do, we add logic circuit between latches of master-slave flip-flops so we do that. Overall performance is surely becoming better and we will be using two phase clocking systems.

Two phase means we will be having one phase like this, let us suppose, I can have another phase which is something like this. So I can ensure that this latch and this latch between these two points your system is not holding a state right. So that is what is our primary target tat we talk of double phase clocking sequence.

(Refer Slide Time: 06:07)



Let me discuss with you the latch based clocking now, if you look at the latch based clocking here you have a clock right. This is the clock you see and then you have the combinational logical clock A here followed by the second latch L2 and then combinational logical block 2 followed by combinational logical block TPDA is basically the propagation delay of combinational logical block A and TPDB is the combinational block of C.

And this for C so I have got L1 and L2, L1, L2 L1 is all driven by clock one and L2s are all driven by clock. So this is our combinational logical block. So let us see how it works out. on the falling edge of clock 1 so this is the clock1. At the falling edge of my clock 1 red line which you see, a stable input is available to the combinational logical block A. So at this point it was suppose it was latched here at this point of stable will is available at clocking right.

Now, during the low phase of clock one which is here, low phase of clock one clock A, means what let us assume Clock A is basically the clock 1 gets the compensation time which is T clock by 2. That you have already understood, why is it like that? Now, the whole time period this is capital T right. This is T clock so half the time period is the competition time which I can allow for the combinational logical block to work with.

So you are getting my point this is the amount of time that nothing will happen and I can still allow I can, it is acting from the low phase of the clock L1 is opac and therefore the combinational logical block can use that timing to generate its own output as per the requirement of the user. So that is the reason I said it gets the computational time which T clock by 2.

On the falling edge of clock 2 the output of combinational block A is latched. This is latched here, right. Where L2 block because it is driven by clock 2 and they are perfectly out of phase with respect to each other. So you assume that during the, this is basically the rising stage of clock 2 right. At this stage you are able to accept the data from computational logic block A. once it does that, you are able to compute to the data when the clock B has got a low phase.

So when clock B has low phase this is in the off stage sort of and therefore this will be able to compute data, similarly this will be on stage, so therefore any data which is available at the edge of clock B will be transported to L1 and L1 will be able to do it shift forward. But then you are ensuring that it is not available to combinational logical block C. So what I am getting the fourth point is that CLB B which is basically the combinational logic block.

This combinational logical block right.so computes at the low phase of so CLB B computes on the low phase of clock 2 which is this 1, this is the low phase of clock 2. And the output is available on the falling edge of clock1. So it is available, so when this falls down right, and this is rising then your output is available. This is quite critical to understand right therefore the time available to perform the combinational block of A and B is nothing but T clock.

Because, within the T clock period I have been able to compute both computational logical block A as well as that of B. and at the edge both are available now I am able to learn C and C will take of it in the previous date. You got the point here. Should I explain once again, I will explain whole thing once again, because this is quite critical in understanding the whole issue here.

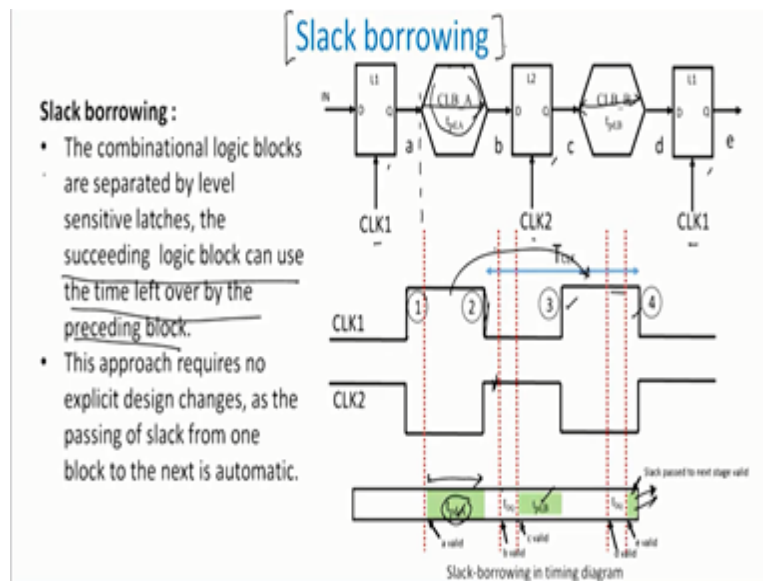
See I have 2 clocks, clock 1 and clock 2 and there are basically as I told you in the previous cases that these 2 are 2 faced clocking scheme as I discussed with you this is one phase and this is another phase right. And I perfectly they have under degree phased up sort of inverted in nature now, at the first. So let us look at the edge number 2 right. On the falling edge, let us suppose I have this I have this so what has happened is at this point.

Clock 1 was high I was able to insert the data from output world and go to the here to clock. Combinational logic block A. when the clock falls down for clock 1 L1 becomes off and therefore at this stage this is computing, combinational logical block starts to copulate that is what is written here right. Now, nothing happens when clock 2 goes high, because it is not transparent to it. It will be transparent, it will be doing when it is low.

So it will compute till this much time T by 2. This is T by 2 right. At this point since when the clock goes low it launches B which means because by this time you have already computed it goes to L2 launches it and after that the combinational logical block accepts that from L2 and starts to compute so after T clock period we have both combinational logical block CLB A and CLB B, both have been actually computed.

And both have been computed and stored at one point. So in one clock cycle 2 combinational logical blocks have been actually evaluated and they have been taken care of so let us look at therefore, so we have understood therefore what is the basic concept of a latch based clocking unlike triggered based clocking. Unlike H to get based clocking. We have latch based clocking here and we have seen how the latch based clocking works. Let us look into the slack borrowing mechanism right.

(Refer Slide Time: 12:08)



So what I was discussing was that you do have a so you have a L1, L2 and this is again L1 and I am driven by clock 1 and clock1 for L1 and clock 2 for L2. Now the combinational logical blocks are separated by level sensitive snatch as I discussed with you. The what does it do, the succeeding logic block can use the time left over by the preceding block. So whatever the preceding block has left over you can actually express that slot.

And I will explain you how it works out right. Let us say at the edge at 1, this 1 where it is edge number 1, edge number 2, edge number 3 and edge number 4 here so at edge number 1 let us assume A is valid right A is valid means A is perfectly valid and A is able why because you have already accepted the data and it is valid now. This TPD of A which is basically mean that this combinational block will use some time because of which green color.

I have written here is that which is available to you. Now. When it goes to this point at this point again now the B will be valid. You see the B will be valid now. Why because the clock 2 is high state and B will be valid and then at this stage you will have C, availability will be there, you have already, the reason being you have already high. Clock 1 is already high, clock 2 is already high and therefore this TPD A is basically the delay which is happening between this point.

And this point and TPD DB is basically the delay between these points right. Now what has happened here fore is that when you go from this cycle to this cycle, right. This is one cycle let us suppose and when you go to the other cycle the difference between TPD A and TPD B will automatically be passed one to the slack to the next stage. This is the next stage which is available here.

So whatever takes, so you utilize some amount of time to generate through logic A through combinational logical A and B and the difference between the gap will be automatically transferred to the next slide. So the next phase or the next edge of the clock or the next latch or the next combinational logical block will have that much extra timing in order to evaluate the data right.

So that is what is known as slack borrowing so you borrow the slack from your previous cycle and do and convert and send it to the succeeding cycle. So now you play with the larger amount of timing and therefore your timing is much more easily done right. So that takes care of slack borrowing and concept of slack, we have also understood what is basically therefore the basic concept of slack borrowing in the latch based design not true in case of edge triggered based design right.

So let us look at insigina design techniques so we have already discussed therefore finished off with our synchronized design techniques sequential let us come to asynchronous design techniques and let me therefore point you out what are the problem areas as of asynchronous design in a synchronized design of course.

(Refer Slide Time: 15:15)

Asynchronous Design Technique

Problems in synchronous approach

- In reality, because of the clock skew and jitter, all clock events don't happen simultaneously in a complete system.
- All the events occur at the clock edge, the overall current flow is very for these significant of time. This causes noise problem due to presence of the power supply resistance.
- { The throughput rate of the pipelined system is directly linked to the worst-case delay of the slowest element in the pipeline. }

To overcome these problems, **asynchronous design technique** is adopted. ✓

- Complete asynchronous system is very difficult to achieve because ensuring a correct circuit operation that avoids all potential race conditions under any operation condition and input sequence requires a careful timing analysis of the network and extensive simulations in CAD tools.

Because of clocks skew the first point is as I discussed with you because of clock skew and jitter, all clock events do not happen simultaneously in a complete system and that is a major problem area for a Asynchronous design. That not all the events happen simultaneously you wanted to happen but it is not happening because of the clock being having a larger amount of skew and jitter.

So all the events occur at the clocks edge. We then being a edge triggering and overall current flow is very small for all these significance times. This causes noise problem due to power supply resistance, we will come to this later one when we discuss. But primarily, when there are fast switching you associate noise with that, this noise is very harmful for any electrical circuitry and the signal to noise ratio actually starts to drop down.

And then the throughput rate of the pipeline system is directly linked to the worst-case delay which means that throughput of the system right is directly linked to worst case delay of the slowest element in the pipeline. So when you have a pipeline architecture which we discussed just now then the slowest element, for example we did with adder, a modulus maker and logarithmic this thing design.

Then out of the 3 whichever is the slowest we will determine the total speed of the system. That is the problem area of Asynchronous designing. To overcome this problem, asynchronous design is adopted and a complete asynchronous system is very difficult to achieve because ensuring correct circuit operation that avoids all potential race conditions under all operating condition is very difficult.

So that is the problem which I wanted to speak out. See if you have asynchronous conditions you have to ensure that there are no rest conditions for any operations of asynchronous design right. This conditions were not validated when you were doing synchronized design the reason being that you were quite you were able to block certain paths by registers or latches as we discussed.

But now in there are no register of latches because these things we can have a fixed data available between input and output and that will result in what is known as the problem area. Mainly problem area in design right. Okay, let us look at the self-time pipelined data path. So what will happen is for example it is quite interesting you have input here registered flag right. So this is first, second and third block right each having a propagation delay of TPF1, TPF2 and TPF3 what TP overall block is something like this.

So whenever you want a computation you send a request signal here right, it allows this to start right so there is no clock here as it starts it accepts input data through register right and this register will be giving it to this computational logical block when the computational logical block has given its output it sends a flag here that it is done, which therefore comes to

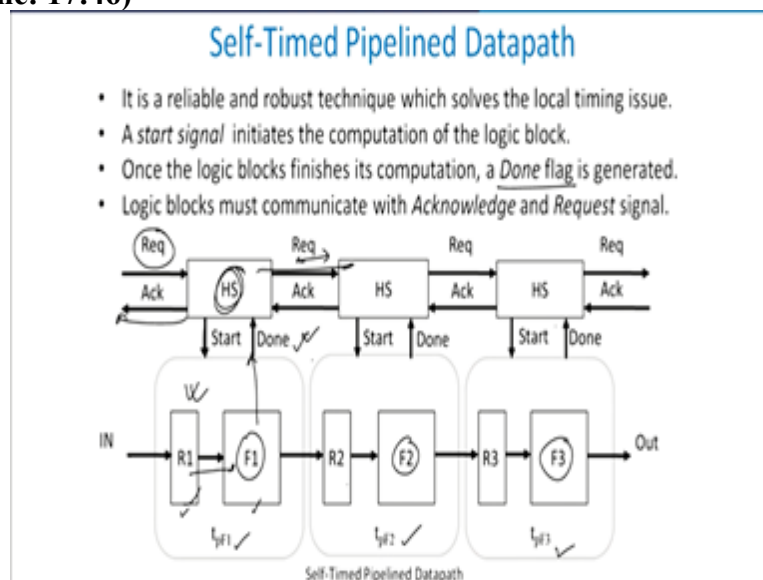
an control unit here and then it allows it to request the next block to start the functionality here.

And sends an acknowledgment in the backside path that the computation is done by block number1 right. And such type of computation therefore is available. So each block communicates with the other block so when the second block sends an acknowledgment to the first block that it is done then the second block sends an acknowledgment back to the control to accept the second request and then it can transform the data to F1 to R2 right.

So this is basically a block to block communication and nothing to do in this stage with clock right. When all the logic block finishes its computation flag is generated and the done flag is generally kept up. So your start clock start flag for switching on this logic block right it does and you have a done flag available which goes to the control unit right. And therefore this allows you to expertise the computation without the clock cycles right.

So this is what is known as self. So let us look at the advantage of self-time circuit design that it does is basically that the physical timing constrains are met with the done flag. I have discussed with you that the set up time whole time evaluations will always be there that can be removed with the done flag, done uh flag available to you.

(Refer slide Time: 17:46)



And this done flag will be always there now this is primarily if you look back is basically is hand shaking protocol which people would follow in a asynchronous design right. So this basically hand shaking between this block and this block. Through these two elements of a design.

Right in a data path so the primary data path is basically is right so this is the primary data path and you do have a control unit which is coming up like here and the control unit talks to the data path and tries to send a signal from one block to another once the second block is ready to accept it and the first block is actually evaluated the data, the combinational data path right so this is what you get as an advantage.

Such as discussed with you the through proper handshaking protocol the logical ordering is held properly there is no logical there is no irregular ordering. There is always a proper ordering of logical handshakes. So what we do is that rather than using a global clock we use a local clock and therefore it is much faster to operate.

(Refer Slide Time: 20:49)

Advantages of Self-Timed Circuit Design

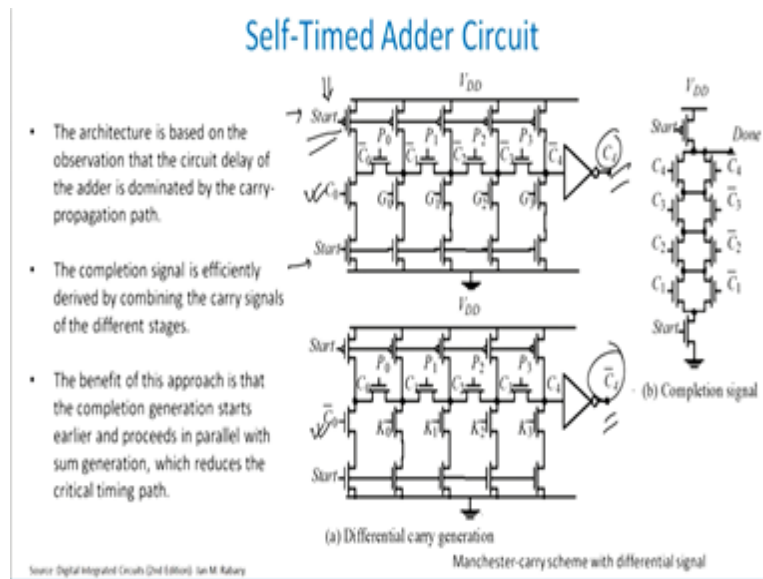
- Separates the physical and logical ordering functions.
 - Physical timing constraints are met with Done flag signal
 - Though handshaking protocol the logical ordering is hold properly.
- Use of local timing signal instead of global clock distribution makes system to operate faster. ✓
- A self-timed circuit proceeds at the average speed of the hardware in contrast to the worst-case model of synchronous logic.
- Power consumption overhead for distributing high speed clock is avoided in this design technique. ✓
- Robust by nature to the variation of manufacturing mismatch and temperature. >

So that is what is written here, is much faster to operate. In times if self-timed circuit proceeds at the average speed of the hardware in contrast to the worst-case model of synchronous logic right. So it works with the, so what we look into is say for example F1 is having a delay which is much higher because certain elements within it, then you are done flag will be available to you after quite a delay and that is basically what I am trying to say.

Power consumption overhead for distributing high speed clock is avoided in this design technique, need not to say that because you are not having any clock. So your activity factor is nearly very low in this case and therefore you can afford to have a much lower power consumption it is very much robust and there is no concept of mismatch in terms of temperature or process variation mismatches right. So these are the advantages of self-timed

circuitry. I can give you an example of a self-timed add circuitry which will help you to add say let us suppose 3 or 4 bits of data right.

(Refer Slide Time: 21:49)



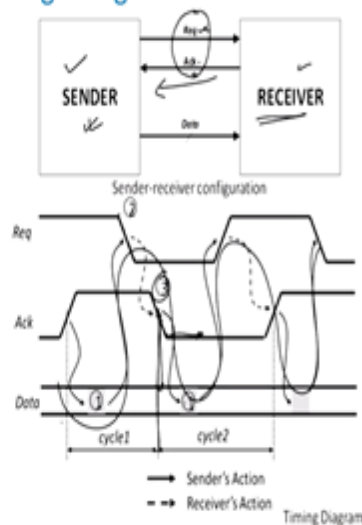
And this is basically a sequential adder which you see so if you look at the, so this is the start signal and this is the start signal for pull on this is the start signal for pull up right. So when this is active low, so when starts goes low it becomes active all this pull-up path are getting activated together. And depending upon the value of CO is the carry beat here and you will have obviously if you put CO here you will get CO bar here, if you put GO you will get CO bar 1 here so on and so forth and finally you will get the carry bit C4 here right.

And you will also carry bit C 4prime here. So if I start with CO and CO complement I end up having C4 and C4 physically 4 bit added here. And I can have this available to me right. Self-timed added circuitry can be taken care of.

(Refer Slide Time: 22:38)

Self-Timed Signaling

- A self-timed approach requires a handshaking protocol to logically order the circuit.
- Sender and the receiver modules communicate each other by Req and Ack signals.
- **Process steps**
 1. The sender places the data on the data bus.
 2. The control signal Req changes its polarity by the sender.
 3. The receiver accepts the data when possible and produces an event on the Ack signal.
 4. After receiving the Ack signal, the sender starts sending data for next stage.
- This process is called two phase protocol.



So this is basically your self-timed added circuitry so self-time approach, this is self-time signaling here, so I have a sender and I have a receiver here right. And so they communicating with each other with request signal and acknowledgment signal and our data path will be given to them on receipt or acceptors of acknowledgment signal. So a self-timed approach requires a handshaking protocol to logically order the circuit.

The sender and the receiver module which is this one and this one communicate using request signal and acknowledgment signal. What is the process step? The sender places the data on the data bar I have a data bar here. So in the first place when the acknowledgment goes high the sender places the data in the database right. The control signal Req which is the request signal changes its polarity by its sender.

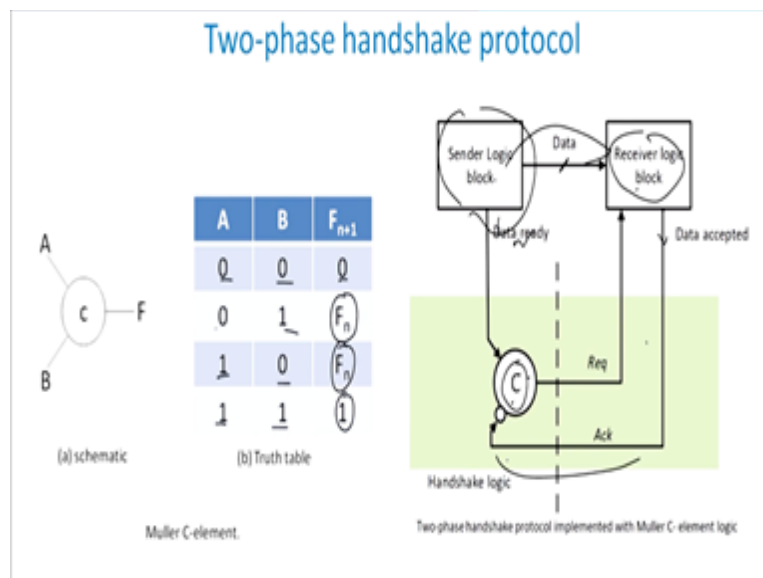
So the request signal goes low right and therefore this is number 2 point the control signal changes into polarity which means that it goes low the receiver accepts that data at this point, receiver accepts the data right wherein and produces an acknowledgment signal. So what it does the receiver accepts the data at this particular point right in the end of cycle 1.

And it generates a data that it acknowledges that it has received it and therefore the technology signal actually goes to lower value right. So I have acknowledgment goes high, accepts the data, data is launched by the sender then when request goes low then the control the request goes low it means that the receiver is ready to accept the signal if the receiver is getting somewhere here.

It receives the signal and it somewhere goes perfectly low and then again it comes to the this comes to the same cycle. 1,2,3 cycle now the receiver accepts the data and produces an acknowledgment signal back to the sender right, after receiving the acknowledgment signal the sender starts sending data for the next stage so the same process like this goes on continuing for all stages right.

This process is also called as a 2 faced protocol. Why 2 faced protocol? Because it actually requires 2 cycles to actually have full transfer of data between one point and another. Please understand request low is active here acknowledgment low is also active here. And when acknowledgement goes high it accepts the data it goes low it is ready to accept the second set of data from the previous cycle in a self-timed signaling procedure.

(Refer Slide Time: 25:09)

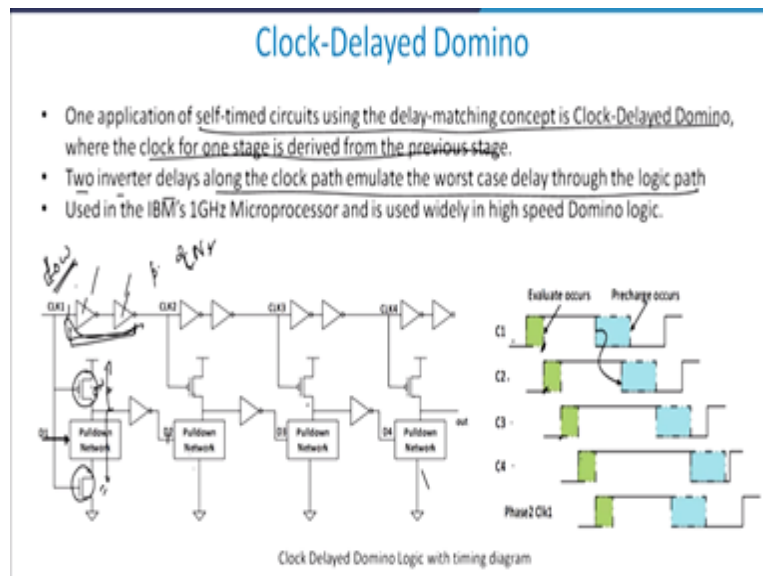


In a 2 phased handshake protocol if I was sender logic right this is basically so if I have 0, 0, and the output is 001 then then the previous value is stored 10 previous value is stored and 1 valid is suppose F_n is 1right. So this is the general logic which I am using. Then what happens is that if the sender I have the sender logic lock here right and I have a data. So data is ready it sends it to the control unit, this is the handshake logic.

This is the basic handshake logic block diagram. It send the request signal back to the receiver logic right and in the meantime the receiver logic accepts the data and then send the acknowledgement back to the control logic which again send the requester logic under data. So what it does is that sender logic sends it to control sends it to receiver lock it sends the acknowledgement. When it sends the acknowledgement the data is transferred to the receiver

block. When the receiver starts acknowledging the acknowledgment and the same logic is followed time and again for the 2 phased handshake protocol right.

(Refer Slide Time: 26:12)



We will discuss this clock delayed domino logic. And as you can as you know I remember the dynamic logic we discussed when we were discussing combinational logical block, so this is applied here also we look very carefully one application of self-timed circuitry is matching in a clock- delayed domino logic where the clock for one stage is derived from the previous stage. So, let us see how it works out if you have clock 1 here right passing through a double inverted double static inversion here.

So this will be this will be clock 2 will be 2 inverted delay which is basically the sum of this inverter plus some of this inverter right. So therefore 2 inverter delay along the clock path emulate the worst case delay through the logic path so this clock so whenever clock say one goes low right.

It goes low let us suppose then after 2 clock inverter delays clock this will go low by the time this has gone low this is on state and therefore let us suppose clock is low which means that if this is low basically sorry I am sorry, if this is clock wise low then this P nob switches on right, and depending upon the data path availability but at the same time n nob is switched off right then nothing will happen.

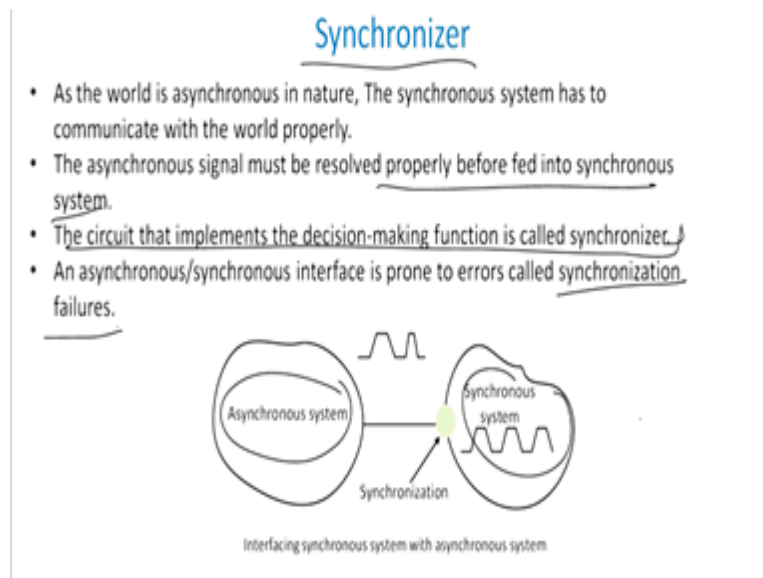
The next phase of clock cycle when the clock is high then this is off and this is on right depending upon your pulled on path this will go to zero right or this will go to 1 or when this

is off as a result D2 will also have a new value of network but it will be only switched on or off after the arrival you have 2 delay clock delay 2 inverted delay here.

Clock 2 will be basically 2 inverted delay here two inverted delay available similarly clock 3 will be 4 inverted delay as compared to clock 1. And therefore as you move from left to right the P nob will be switched on so that is what I am trying to say, so if you look at this diagram here C1, C2, C3 and C4 and if you look at C1 C2 is basically clock 1 clock 2 so evaluate occurs here whereas in the second phase evaluate occurs at much later stages.

Because they are actually 2 inverted cycles back side right. Similarly pre charge also takes place in the same manner right and that is quite interesting which we see. So without using an external clock by using a local clock we are able to generate this clock delay demagogic.

(Refer Slide Time: 28:47)



We define what is known as synchronizer. Synchronizer is a signal that is resolved properly before you fed into synchronous system. The circuit that implements the decision-making function is called synchronizer. So in asynchronous system I want to convert it into a synchronous system, I use a synchronizer to do that. We will not going to details at this stage but I will just let you know what a synchronizer is.

It helps asynchronous design to convert into a synchronized design right. That is the reason we use the work synchronizer here. Obviously it is prone to many synchronization failures which is one prone to synchronization failures, when you convert from asynchronous design to synchronous design. And that is a quite critical as far as probably other synchronizer is

concern right. If you look very closely here if you look at very simple synchronizer I will explain to you how it works out.

(Refer Slide Time: 29:35)

Synchronizer

- During clock rising edge, the data is sampled.
- But there is violation of setup and hold time violation of latch as the clock is not synchronized with the clock.
- For this reason, waiting period is required.
- A signal is called undefined if its value is situated between V_{IH} and V_{IL} .

$$V_{MS} - (V_{MS} - V_{IL})e^{-T/t} \leq v(0) \leq V_{MS} + (V_{IH} - V_{MS})e^{-T/t}$$

Where, T = waiting period,
v(0) = range of value causes for the errors.

- The probability of v(0) for residing v(0) in the undefined region is

$$P_{err} = \frac{(V_{IH} - V_{IL})}{V_{swing}} \cdot \frac{t}{T_{setup}}$$

A simple synchronizer

Linear approximation

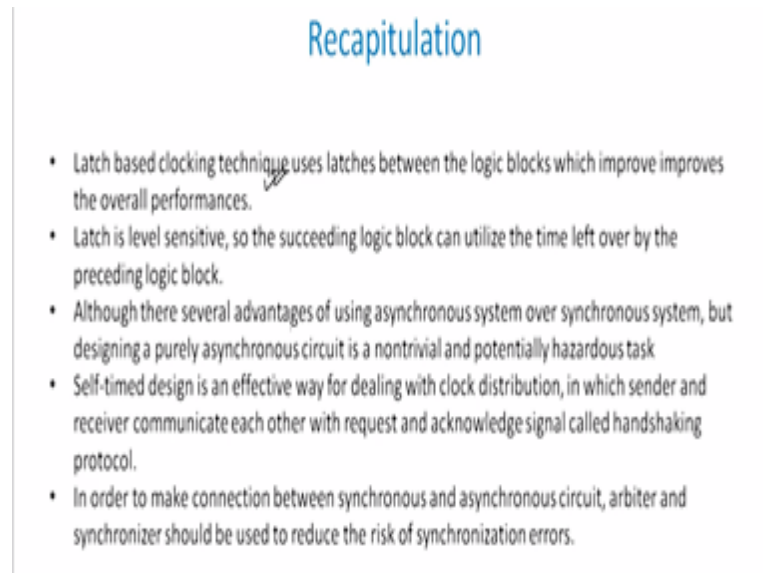
During the clock rising edge let us suppose so this is fed by clock this is clock bar right. At the rising edge of the clock the data is sampled. As I discussed with you rising edge means your clock is equal to zero and clock bar is equal to 1. Which primarily means that this will be on and D will be sampled to the sponge part. But there is a violation of setup and hold time of the latch as the clock is not synchronized with the clock the latch.

The clock is not synchronized as there clock of this clock is not synchronized with the data sorry this is data which we need to find out, with data right. Therefore if the data is available earlier as compared to the clock you have to wait and if the data has crossed the clock then you have to actually wait till the next phase of the clock comes. For this waiting period is required right.

So what I say is that a signal if the value is restricted between V_{IH} and V_{IL} which is input low and input high and input low and then the probability that V_o is undefined in the probability is this thing. This is the probability which gives me $V_{IH} - V_{IL}$ upon T is equal to T_r . Where T_r is the life time available to you. So you see if your $V_{RH} - V_{IL}$ is typically very large then the probability that output voltage is latched to a particular value is also very high in this case 0 right.

And therefore I can simply say that I have been able to synchronize a design into a proper synchronized design. This is what a synchronizer on how it works out we will therefore we will just recapitulate what we did for this case.

(Refer Slide Time: 31:17)

A slide titled "Recapitulation" with a blue header. It contains a bulleted list of five points regarding clocking techniques and synchronization. The text is slightly blurry but legible.

Recapitulation

- Latch based clocking technique uses latches between the logic blocks which improve improves the overall performances.
- Latch is level sensitive, so the succeeding logic block can utilize the time left over by the preceding logic block.
- Although there several advantages of using asynchronous system over synchronous system, but designing a purely asynchronous circuit is a nontrivial and potentially hazardous task
- Self-timed design is an effective way for dealing with clock distribution, in which sender and receiver communicate each other with request and acknowledge signal called handshaking protocol.
- In order to make connection between synchronous and asynchronous circuit, arbiter and synchronizer should be used to reduce the risk of synchronization errors.

The recapitulation is that we have done that we have done a latch based clocking design we have also understood the concept of latch is being level sensitive and therefore I can do large amount of slack borrowing from previous stages that we have also understood self- designed timing circuits have been taken care of whether the clock has been fully removed and therefore we get a fully design available to us.

We understood what is the handshaking protocol in a synchronizer and we also looked into a concept of the difference between the asynchronous circuit and synchronous circuit if you want to convert you want to use the synchronizer. But then synchronizers are always prone to large amount of failures right, with this we will be able to finish this clocking sequence designed for the synchronized circuits. Thank you very much.