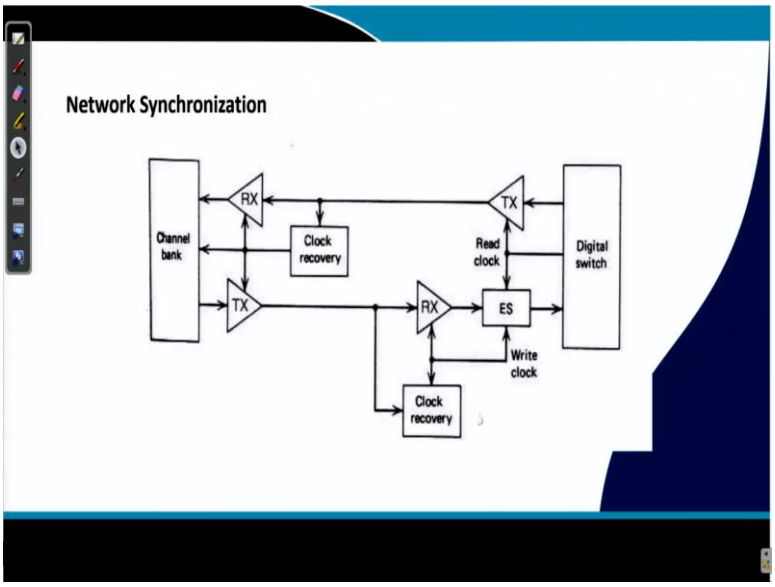
Communication Networks
Prof. Goutam Das
G. S. Sanyal School of Telecommunication
Indian Institute of Technology, Kharagpur

Module - 03
Circuit Switched Networks
Lecture - 14
Synchronization cont'd

Hi, we are in lecture 14 of Circuit Switch Networks. We will continue our discussion on Synchronization, which is one topic that is probably still left for this particular discussion of switching in circuit switch networks. So, we will quickly go to this one, the one thing that we have started discussing.

(Refer to Slide Time: 00:47)



So, basically, there are we have discussed already that there are two kinds of synchronization issues one is for multiplexing, and another one is for switching. So, switching Synchronization is a quick recapitulation; we have already discussed that switching the input or output is coming or going to multiple other geographical locations.

So, it is not necessarily that all those streams will be exactly clock-synchronized or frame synchronized; they might not be. So, it is our job to actually make them synchronize. So, this is what we will have to do. So, at the input of the switch, because it is getting heterogeneous input from different locations, different bit streams are coming.

So, their data rate might not be exact, or their clock rate might not be exactly matching; there must be slight desynchronization; it can always happen, even if we try to synchronize it to the best of our ability, but there will be some desynchronization and that something we have to take into account because otherwise there will be a huge problem.

So, we have also discussed that there might be this desynchronization issue, and we have to somehow, at the input of the switch we have to do something. So, that is where we started discussing this elastic store. We have already discussed that we have also given in the next slide probably also have the discussion about its circuit. So, we will go into a deep discussion of that, but before that, what is the other component that is used in the circuit switch network? That is multiplexing.

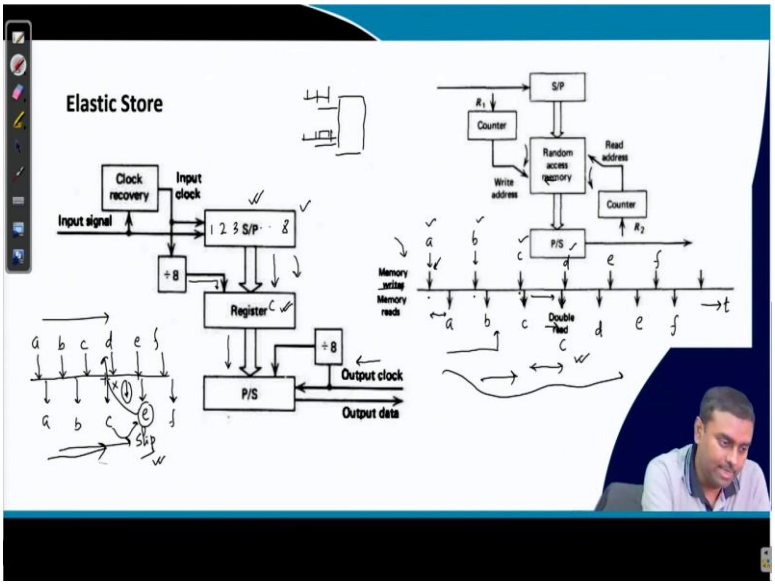
So, again in multiplexing, what might happen? Different streams are coming; they might also come from different locations, and again, their bit rate might not match, or they mean exactly; of course, they will be almost similar, but there will be a slight deviation in the clock frequency, and there, accordingly the associated rate.

So, what do we do about this multiplexing also, because multiplexing takes exactly what we talked about this bit interleaving or byte interleaving exactly takes from each stream within a particular 125 microseconds; they take an equal amount of bits.

But the problem is if they are desynchronized, then there might be, or there is a possibility that if you count for a longer duration, the bit's number of bits might not be equally counted for both the streams because they have a slight deviation in clock rate. So, that is another thing that we will have to account for.

So, we have to make provisions so that these things are properly accounted for. So, we will see how to do that. So, first, let us talk about the one discussion that we have already done, which is network synchronization ok.

(Refer to Slide Time: 03:22)



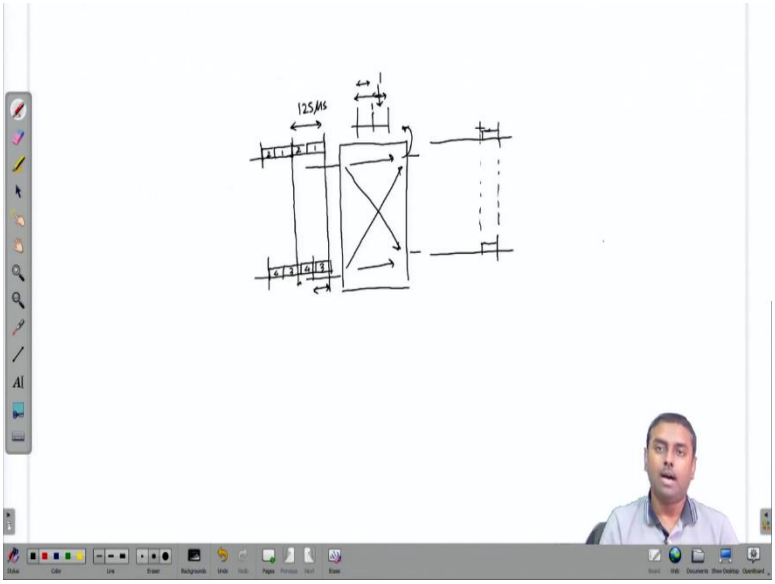
So, at the input of the switch, as we have shown in the previous slide. So, basically, what you do, you have. So, this is our switch, ok. So, this is the digital switch, and we are probably getting output from here and input over here, ok. So, what are we trying to do every time whenever we transmit something? So, that is the output that is going out. So, that should be the switch output. So, therefore, it must switch its own clock. So, we want to synchronize it with respect to the switch clock.

So, any data that are coming we cannot actually synchronize with each other right then it will be very difficult to synchronize with whom. So therefore, we get the reference from the switch; whatever clock it has, all others have to be somehow synchronized to that. So, we will see how to do that.

So, basically, from the switch transmitter will take that actual clock and that particular clock, or from the switch, we take the clock, and that clock will be fed to the elastic store; we have already demonstrated that and also in the receiver part whatever data is coming that might have a different clock. So, we recover that clock, and actually, that particular clock fits into the elastic store.

Now, the elastic store will do something with these two reference clocks and somehow make the data properly readable by the switch ok. So, this is something we have already started discussing. Now what we will try to do is what the problem is; probably the last time we have discussed this problem. So, briefly, we will again discuss it for convenience.

(Refer to Slide Time: 05:01)



So, basically, what happens? You have input to the switch; now, these data are coming. Let us say this is a 125-microsecond frame, ok, and let us say only two are being multiplexed, ok. So, they are coming one after another, ok, no problem with that, ok. It just keeps on repeating. So, this may be data from 1, data from 2, data from 1, data from 2; they keep on coming ok.

In the other one, this is another port to the switch. So, this is also now let us say I have slight desynchronization ok or slightly higher this one ok. So, what will that mean? That is probably the frame because the rates are slightly different; the frame might not be contained. So, this if this is the frame.

So, there might be some slight desynchronization among the frames ok. So, they get repeated similarly, and there might be some desynchronization because the rates are different. So, how much they are misaligned also might keep on changing over time because the rates are different ok.

So, this is probably if we take 8-bit, 8-bit 16, bit is coming over here. So, the other one might take a little bit longer duration to transmit those 16 bits. So, it might happen that they are not completely aligned with each other if this happens, but what do we do if we have a switch over here? Suppose it has a space switch or it has a time switch. What does it do for each one of them? So, within this, there will be switching decisions.

So, a clock means this has a clock with that clock frequency; it actually decides. So, this is the overall slot, and within this slot, I have subdivided it into multiple small slots. So, 125 microseconds are subdivided, and over this slot, I will apply some logic in the switch; maybe I will put them in the bar stage. In the next one, we have already demonstrated that we will probably be putting another logic maybe that time switches in; sorry, this time it is in the bar stage, earlier it was in the cross stage.

So, like this, this might happen. So, because of that, what will happen? So, let us see the target at the output; what is going on? So, at the output, what will be happening? So, let us say this is 3, this is 4, this is 3, this is 4. Now because at first this one, the switch is in the cross-state. So, therefore, one's data will come over here, ok.

So, the data will come over here. For the exact same duration, this data will actually get transferred over here, but the problem is that they are not synchronized. So, some partial data will be transferred, some partial portion of 3, and some portion of 4 will be transferred over here; this is a detrimental issue because now the switching is not being done properly.

So, this is what will happen if we do not make the input synchronized at the byte level; if we cannot do that, there will be a problem. So, this is the problem we wish to solve through our elastic store. So, let us now try to see elastic store does ok. So, in the elastic store, what happens? So, first, we take the input signal ok, we make the clock recovery that is all fine, then the clock is subdivided by 8; that means we are actually trying to identify the byte exactly that 8-bit or byte boundary ok.

So, if it is not byte inter lift, then it will be some other thing; of course, if it is 10-bit, then it should be 10 by 10-bit interval. So, it will be subdivided by 10, but right now, we are assuming that it's an all-8-bit. So, actually, we do this byte interleaving, and that is why we are subdividing it by 8; that means the byte boundary we are trying to find out ok. So, once we find out the byte boundary and what we do with that, we will see that later.

So, first, this data will be converted to serial to parallel ok. So, with this particular device. So, let us say this one, ok. So, it is first converted to serial to parallel; that means you keep a shift register; it will keep on storing those data one by one, and then all 8 bits will be stored over here.

So, bit numbers 1, 2, and 3 like this, all 8 bits will be stored over here, ok. Once they are stored, then this clock, if it is trailing edge or leading edge with that clock, actually whenever that divided by 8; that means, byte boundary is identified once a byte boundary comes entire byte has been read over this register.

So, then we bring it down over here. So, this gets enabled, and the entire thing is read over here ok. So, this is this register has been read, but as you can see, this data still has not been transferred to the output data, which goes to the switch ok. So, that is being done with the other clock. So, this is where we take the switch clock. So, this is the switch clock we take, ok.

So, switch the clock again; we divide by 8, which means the boundary we actually fix is the switching boundary; that means exactly at what boundary we want to get this data read ok. So, at that boundary, we actually enable this one again in this register. So, what happens? So, this data gets transferred over here.

So, there is a slight delay when the data is getting read according to the clock input clock of the data and the way data is being taken out to feed into the switch. So, there is a slight delay; this will always happen. So, that delay is because of this desynchronization ok. So, once this is read again, this will be converted parallel to serial, and that is fed to the switch.

So, now with this operationally, if we try to see what exactly is happening. So, let us try to see suppose this is my time frame ok. So, in time what we are trying to do over here is let us say this is the data input. So, these are the byte boundary. Actually, ok, these are demarked as byte boundaries.

So, at that byte boundary, only data 8-bit data or each byte is being read to the register. So, this is the point where the first data has been read into the register, this is the point where the second data is actually fetched into or put into the register, and so on, this is what is happening.

Now, while writing, that is with the switching speed or switch clock speed, right? So, there let us say, as I have told you, there is a slight delay. So, with that delay now, switch this is the byte boundary of the switch. So, at this point, the switch will be trying to read something, whatever is there in the register; remember, ok.

So, now, switch to this one. As you can see, it is slightly faster than the other one, ok. So, you can easily see the number of bytes being transferred in this overall time that is higher for the lower one; that means it is a faster one.

So, it is reading a little fast, whereas the writing is a little slow; that means the incoming data is not able to keep up pace with the input ok. So, that scenario is happening. So, what will happen in that particular scenario? As you can see at this instant, what I can read, whatever data has come over here, let us demark it as a. So, data a has come, data b has come, data c d and so, one, f.

Now, as you can see, he will read over here what he will read at this point b has already been written. So, b will be read over here, so, on c, but at this point, you can see there is a difference that is happening over here. So, over here, as you can see already, c has been written in the shift register.

So, now, the shift register is stored in c the first time it has been read. So, this is reading c. Next, nothing has been written over here. So, c is still stored in the shift register. So, therefore, if you try to now read, c will be again read over here. So this is called the double read or double read.

So, you are getting a double reading, and after that, again, d will be read over here, e will be read over here, and f will be read over here. So, as you can see, the data sequences are good over this time; the only thing is that there might be an occasional double read, and when this double read will be happening if your switch input clock is a little bit faster than the input clock ok. If the reverse happens, we can again demonstrate that if suppose this is faster, the input clock is faster, but the writing clock that is a little slower.

Then as you can see, a b c d e f, these are the data coming; I am reading an over here because I have the last data available an over here, I will be reading b over here, c over here I skip this data. So this is called the slip. I skipped the data, and I actually read e because e is already available d was available, but in between, no reading operation happened.

So, therefore, I read some data, e. So, there is a slip one data is missing, and then again, next time, I will be reading f something like that. This pattern will be continuing there might be if my input is slower or the switch has a slower rate clock rate, then there will

slip if the switch has a higher clock rate, then there will be a double read, and this will be at a repeated interval this will be happening, and we can easily calculate with what repetition it will be happening that depends on the difference or deviation in the clock frequency ok.

So, if the clock frequency deviation is very slight, then this repetition interval will be very high. So, at a very low, after a very long duration probably, this repetition will be happening, or this slip will be happening. So, this depends on how much the clock will be deviating. Generally, clocks will be deviating by a very slight margin, but the problem is not that there is any clock deviation; we know that there will be this byte boundary mix-up.

But now what we are doing is because all of them are now being synchronized with the elastic store. So, every data sequence might be a little bit disturbed, but they are read at the exact boundary. So, the problem we were discussing over here is that if I have a switch and then inputs are being fed.

So, there was a disruption in the data reading because of desynchronization. So, this data was partially read in between; that will never happen because the switch is now at the input. With the help of this nice device elastic store is getting synchronized exactly ok.

But now you might be asking; there is a very natural question that comes into our mind what will happen to the switching? Because of switching, we have some discipline, and basically, all this switching depends on the address, and the address is embedded in the time sequence right where the data is located within the frame.

If these data are supposed, let us say a is the first data, b is the second data, c is the third data, and this is the fourth data. So, in the switch you have seen, we have a space switch inside the space switch; we actually configure it so that the first data should go over here. So, accordingly, we have logic in the time switch. Also, we have a logic for which data has to be swapped to whom.

If these sequences are disturbed, then the wrong data will go in the wrong direction; now, that is another problem because of the elastic store because the data sequences are getting jumbled up because after c, now e is coming. So, switch if I do not do anything, it will

treat e as if it is the data coming from this d ok, and it will switch it to that direction. So, wrong direction data will be transferred, and this is a big problem.

So, we have to now rectify this, but this is a very easy thing to rectify. What do you have to do? All the switch logic that was soft logic ok. So, we had a software-based logic implemented in hardware, right? So, what we can do is that logic can be easily changed. So, we can detect this event wherever there is a slip, or there is a double read which is happening that you can easily detect by putting a counter that how much is being read in between one write.

So, if there is more than one read, then you know that it is a double-read. So, you can immediately issue that because of this probably, there will be disruption. So, your switch logic has to be now modified to some other logic. Once you do that again, the switch will be functioning properly. So, this is all that you will have to do to actually make it completely synchronized.

So, now you can see even if my data is asynchronous, even if my two inputs have different clocks, I have no issues. We will install this elastic store at every input; they will synchronize it to the clock and byte boundary of the switch, then the switch will do the switching because of this because there is a mismatch of the input data this one there will be a loss of sequence occasionally because of the slip or double read we have already seen.

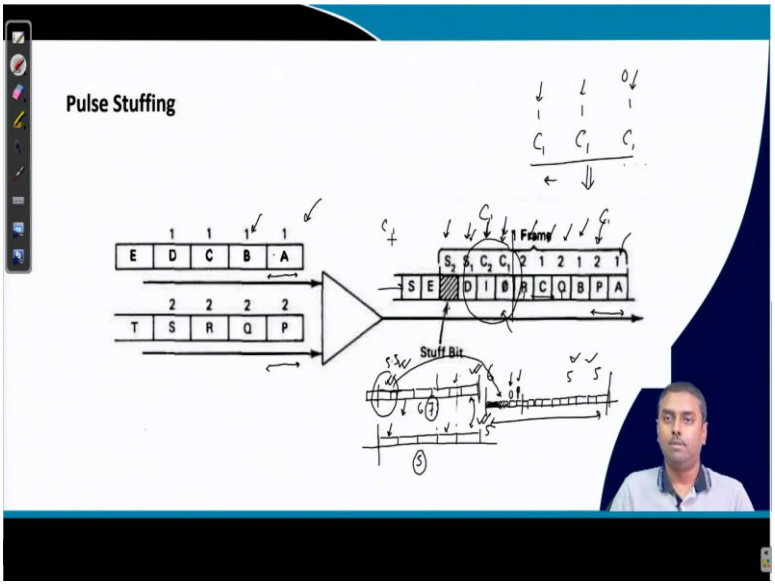
So, that has to be triggered from the elastic store, and this message has to be fed back to the switch logic center, and switch logic will accordingly take corrective action so that the switching is done properly. Now there is no desynchronization or misalignment so, so there will be partial red within a slot that will not be happening.

And if there is some wrong switching of data, that is also rectified because I already take this event double read, and then I know what to do and what kind of switch logic manipulation will have to do so that the earlier switching is restored ok. So, as you can see, with a very simple device, we can actually alleviate this particular problem which is a practical problem, and we have to really deal with this problem.

So, this is exactly what we could do now; you might be thinking that if this is a particular kind of synchronization loss ok that might be happening over the space in some of the locations, there will be desynchronization, and we can easily handle this, but what will be

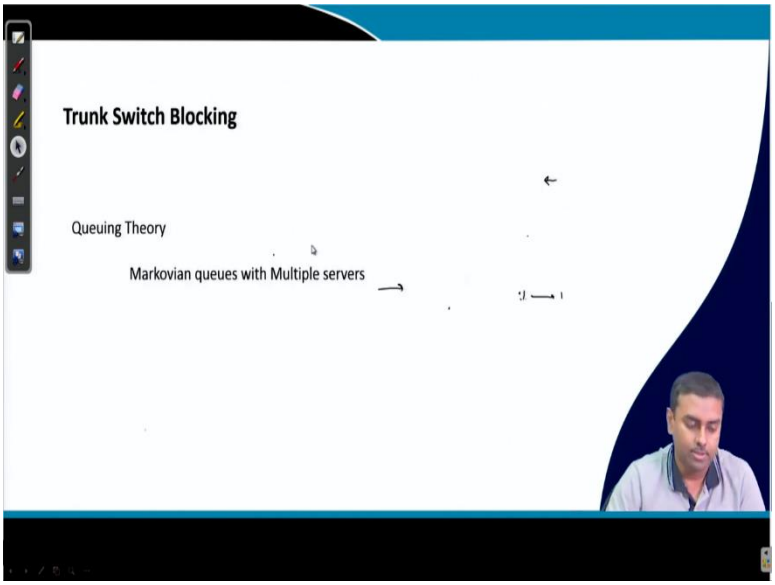
now happening if we have a multiplexer desynchronization. So, that is exactly what we will be discussing next.

(Refer to Slide Time: 20:46)



So, let us try to see how we manage the multiplexer ok. So, multiplexer again, the same thing might happen. So, you have these data which are coming ok. So, let us say these are the bytes ok in the first this one A is coming over here, then B followed by C. So, these are the words encoded in the byte ok. A second stream also has some data P, Q, R, S T; let us say ok.

(Refer to Slide Time: 21:15)



Now, if there is a desynchronization in the clock sorry if there is a desynchronization in the clock then what will happen? This byte duration and this byte duration might not be exactly the same. So, there might be a slight deviation, ok, once you are trying to interleave them. So, interleaving means within single 125 microseconds.

So, I am putting data streams from both of them. So, we squeeze them. Basically, we increase the data rate; we have already talked about that how we do multiplexing. So, we will be interleaving them one after another. So, A followed by P, and then again take B from here and then followed by Q. So, B, Q, C, R, and so on. Ok, we do it this way.

But as long as they are synchronized, no problem, but what will happen if they are not synchronized? You will see that over time, this will have, let us say, a higher rate; this will have a higher number of bytes delivered, whereas this will have a lower number of bytes delivered. So, let me give one example probably. It will be very clear.

So, let us say the first stream has 6 such things ok, and the second stream within this is only having 5 OKs because this is a little slower ok. Now I have to actually transfer them. So, what can we do at this instant? We can actually combine these two again at this instant; the slower one I am taking as a reference, I can combine these two ok, and I can keep on doing so, like this.

Up to probably the slower one if I see I will be over here if I come; this will be combined with this one, but this will be left because this has 6 and this has 5. So, if I am combining in time, if you see over here at the output, if I combine with respect to the slower one, I will be able to 1 2 3 4 5. I will be able to combine 5 of them, but this last one will be left alone. What do I do with this? That is when people have decided that maybe they will do something else. What will we do? We will have this kind of encoder.

So, basically, let us say I make a frame over here. I am giving an example. So, I make a frame of 4 4 bytes each, ok. So, we take 4 bytes from one of them and 4 bytes from the other one we take. That after that, inside the frame, we are now adding a redundant bit. So, we are giving one control bit C-1, another control bit C-2, followed by ok one control byte. Let us say if we do it as a byte or if we do it as a bit; then then it will be a bit. So, it depends on bit interleaving or byte interleaving whatever we have done.

So, then another control, sorry, another storage, ok. So, this is C, and this is S. So, because of the two streams, we are putting. So, there must be two control and two storage; we will see why that is required. Now what will we do? We have to see within this frame how many of this one has been supplied and how many of this one has been supplied. It might happen that one of them has one extra.

So, that extra has to be stuffed ok for the other one. So, what are we doing? In control, we are specifying the fifth one as optional 4 over here in this example, probably 5 5. So, 5 will be putting together one after another; then the sixth one is optional; in the sixth one, what do we do? Through the control, we say which one has data and which one does not have data.

So, 1 and 0 are the encoding for specifying whether we have data or we do not have data. So, if we specify 0, then it means that we do not have any data for stream one. If we specify one, then we specify that the next following byte, there is data ok. So, followed by two storage bytes, and this makes the whole frame actually, and these particular things get repeated. So, by these two redundant bits, what are we doing? We are actually stuffing.

So, over here, we have only 5 data; over here, we have 6 data we have to transfer the whole data. So therefore, for the first one, we specify over here 1, and for the second one, we specify 0; that means, over here, I will be putting the data from the first stream the excess data, and over here, this is being stuffed; that means, nothing is being put. So, this is where you are putting an extra space so that you can adjust their clock whenever it is required in a bigger frame ok.

So, if there is a clock mismatch immediately by adjusting this C 1 C 2 bit, you can always put somebody, or you might not put somebody you might be asking ok within, let us say, one frame, which can take 5 and 5, this is desynchronized in such a manner that it is having only 5 and half bytes then what do I do? Ok.

Then the solution is again simple. So, in the first frame, you do not stuff anything you say for both of them; it’s 0 0. So, you just do not put anything in the next frame definitely because it is 5 and a half in the next frame; it will become 11. So, one will be extra that extra you put in the next frame. So, like this, with an integer multiple of these things, you will be able to always manipulate the rate misalignment ok.

So, this is very simple; you can always do that with whichever degree of misalignment you need, but remember, it has a limit. The limit is within this one frame; if it has 5 and this has 7, it is so much misaligned that one is only getting 5 data the other one is getting equivalent 7 data; then you cannot do anything because you have only located one space for stuffing not more than that.

So, one data misalignment within a frame is possible, but more than that, it is not possible, so you have to keep in mind, so, there is a restriction it just says how much up to how much clock misalignment you can still handle and all the data can be faithfully transferred.

So, this is actually this is called the stuffing, ok. So, you can call it pulse stuffing sometimes; it is also called byte stuffing. So, you or if you just do it bit by bit, then it is bit stuffing.

So, you define a frame, you put some redundant bit, and within that redundant bit, you actually do this stuffing control, and then some extra space is kept where you put data; if it has extra data, you stuff it; that means, you just put garbage value let us say some particular encoding which is not included inside it. So, basically, you do not put anything over there; you might put just a clock so that the clock is not getting lost.

So, you just put a 1 0 1 0 sequence probably over there because this C control bit is already telling that that is a garbage thing; if it is 0, then you do not read that right. So, you can always do that. So, that is the extra thing you put for pulse stuffing or byte stuffing. This is exactly what you do, and this has a limitation, as I have told you that up to some clock misalignment, it can take and always in practical scenarios without us knowing that the clocks are not heavily misaligned.

If we say it is 64 kbps data, it will be almost close to 64 kbps; it might be 59 points, sorry, 63.9 kbps or 63 0.99 kbps or 64.11 kbps or something like that, but it will not be misaligned by a heavy margin. So, these things will always work.

Remember, there are certain issues over here. The issues are these control bits are now very important bits because if you do not know what is there in control or if you do not know if the control bits are somehow due to an error getting flipped, then you will completely miss reading the data stream whoever does not have any data which has been

stuffed you if 0 becomes 1 in the control bit you will be unnecessarily reading that data ok.

Whoever does have some data might miss them if the 1 becomes 0. So therefore, these two control bits are very important. So, generally, you have to make them error-prone you have to protect them from errors because it is a digital transmission. So, there might be this bit of flipping that might happen; there might be errors ok.

So, due to different reasons, there might be noise, and there might be interferences. So, all kinds of things might happen. So, due to that, there might be erroneous data reading; one data getting erroneous ok, that is not that severe, but if this control bit gets erroneous, then you have a whole byte because of this probably will be misread. So, because of that, generally, you give protection against the failure or error proneness of those data.

So, what do you do? The first thing is you generally keep redundancy over there. So, what do I mean by redundancy? That means, this C 1 bit, you actually put multiple C 1 bits, ok some even number of maybe 5 or 7 or 3, something like that, and you will be putting the same thing over there.

So, suppose I want to put some data in the first stream, ok. So, in D 1, I want to put something. So, this must be one. So, I will be putting one in every C 1; what does that signify? That even if there is a and then what I will be doing, I will be taking this and then voting among these bits.

So, if there is an error, one of them might be erroneous, but these two will still survive, and because of the voting, I will get 2 out of 3 are 1. So therefore, I will take one. So, even though one bit is erroneous, I will still be able to decode it properly. So, this is often being done, but remember, often, these errors come in bursts; that means, if one bit is erroneous, the neighboring bits will be around here. So, that is why within the frame, you generally distribute these C 1 bits.

So, that means you can put C 1 somewhere here, C 1 somewhere here, and C 1 somewhere within the frame you distribute them so that a burst error never actually sacrifices your means error protection, ok. So, this is often being done.

So, basically, what you will be doing will be reading all these things, then all C 1 you take voting among them whatever is majority you then say ok this is my decision. So, I should get a means to stop the fight, and accordingly, it will be deciding whether it is proper data or it is not a property ok.

So, that is the means whole discussion about Synchronization that we have so far done. So, basically, we have discussed two kinds of Synchronization; one is for the multiplexer because we had two devices over here, and we have introduced two devices, one is a multiplexer.

So, multiplexer synchronization, we have already seen how to deal with that. Similarly, the demultiplexer will be just doing the reverse thing, ok. It will be decoding them. So, basically, from that reading, the way we have done this pulse stuffing, it will do just the reverse thing. So you will be able to get the data properly.

And you have switches at the switch input port. If they are desynchronized, then the switch will be doing partial switching, which will completely disrupt the data. So, for that, we have started installing an elastic store, elastic store at least makes the data means exactly aligned to the byte boundary of every slot of the switch, which is good, but due to that elastic store, we have also seen that there might be either double read or slip.

So, you take that input from the elastic store and accordingly rectify the switching characteristics or means. I should not say characteristics or switching logic so that it still does the correct switching because he now knows what the shift is in the location. So, accordingly, he will just reinstall the switching logic ok. So, with this probably, we have finished our synchronization issue so; which means what we have so far done is we have seen what the components required in a circuit switch network ok.

A circuit switch network is something where you first establish the connection ok, and after establishing the connection, you actually send your data ok. So, over here, we have also appreciated that it is TDM multiplex. So, the multiplexing is TDM; that means the exact location is his address.

So, with respect to the location switch decides where to switch it. So, basically, the switching and everything happens through this, then we have seen what are the components that are required for circuit switching for facilitating circuit switch network.

So, one is, of course, the multiplexer or demultiplexer, then which actually generates from a lower data stream the time division multiplexing and generates a higher data stream.

Once we have to multiplex, after that, it is switching, then switching also, we have evaluated what kind of switching there might be. So, we have seen that there is a space switch, there is also a time switch, and there is a combination of these things; we have also discussed the advantages, relative advantages, and disadvantages of this space switch and time switch we have done that ok.

After seeing that we have means, we have started discussing the performance of switch or switch design characteristics or criteria. So, there we have discussed one very particular quality of a switch that is called the blocking probability.

We also try to evaluate what should be the efficient design of the switch that something we have tried to evaluate, how we reduce the switching complexity that something we have discussed we have also discussed the control of the control logic of a switch, and how much memory is required for control logic that is also we have evaluated.

And then we have also discussed what should be the means characteristics of a switch. Should it be completely strictly non-blocking or rearrangeably non-blocking or a blocking switch? If it is blocking, then what will be the associated blocking probability, how do we evaluate those blocking probabilities, and what are the guiding equation in Lee's graph we have discussed?

And then, we started actually evaluating that for a particular targeted blocking probability, how do we design a switch, and we have seen the multi-stage switches also by combining time switch space switch multiple space switch. So, all those things we have seen.

In the end, we discussed a very important aspect of the circuit switch network, which is Synchronization. We know that there will be a loss of Synchronization because it is a spatially dislocated network. So, different components are at different locations; they are getting input from different locations.

So, they will be desynchronized. So, desynchronization and how we handle them from an engineering design perspective is also something we have appreciated. So, that completes the whole gamut of the circuit switch network, probably.

So, over here, we will end this circuit switch network with only one topic, which is left; we have seen how to evaluate the space switch, and those space switches were not; we have also discussed space switch or any switch.

There are two kinds of switches; one is the local subscriber loop and they are the switch where it is some input port to some specific output port, but we have also discussed there is a switch called trunk switch. In the trunk switch, the output ports are not specified; you have multiple output ports; they are like servers and any of those ports you can take.

So, it is a kind of switching, whichever is available. So, this trunk switch we still have not characterized it; that means we have not analyzed it how to do how many trunk links we should put so that we get some amount of blocking probability. So, this blocking criterion cannot be resolved for trunk switches through Lee's graph, which is something we will now be exploring, and that is where a very fundamental theorem comes into the picture, which is called a queuing theorem.

So, what will we do? We will go a very brief this is not the course queuing theory, of course, but we will do a very brief overview of this queuing theory, and then we will see how to evaluate a switch, especially a trunk switch, which is something we will be discussing next.

Thank you.