Computer Architecture

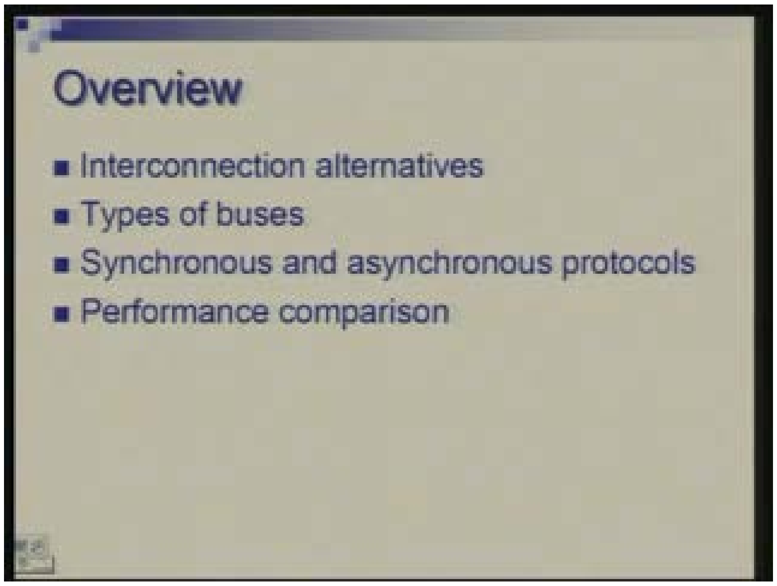
Prof. Anshul Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Lecture - 34

Input/Output Subsystem: Interfaces and buses

In the first lecture on input/output subsystem we talked mostly about peripheral devices which are in some sense transducers carrying information from processor of memory to the outside world and vice versa and transforming the information into suitable forms in the process. Today we will how these get interfaced to the processor in memory and become part of the whole system.

(Refer Slide Time: 1:36)

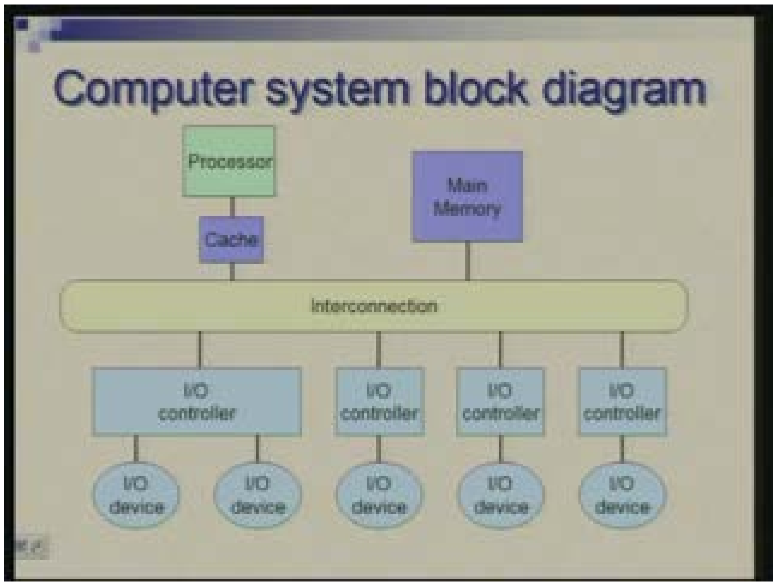


Overview

- Interconnection alternatives
- Types of buses
- Synchronous and asynchronous protocols
- Performance comparison

So we will begin by looking at different types of alternatives which are available to interconnect various subsystems and then we will focus on buses as the most common of these. We will look at different types of buses, how buses can be classified, what is their role and the information which flows in the buses can follow different types of protocol; it could be synchronous or asynchronous. Synchronous means it will be timed to the clock and asynchronous means it will be not synchronized with any clock, there will be signals which will carry the given sequencing information and we will then take some example trying to compare performance of two different protocols.

(Refer Slide Time: 2:40)

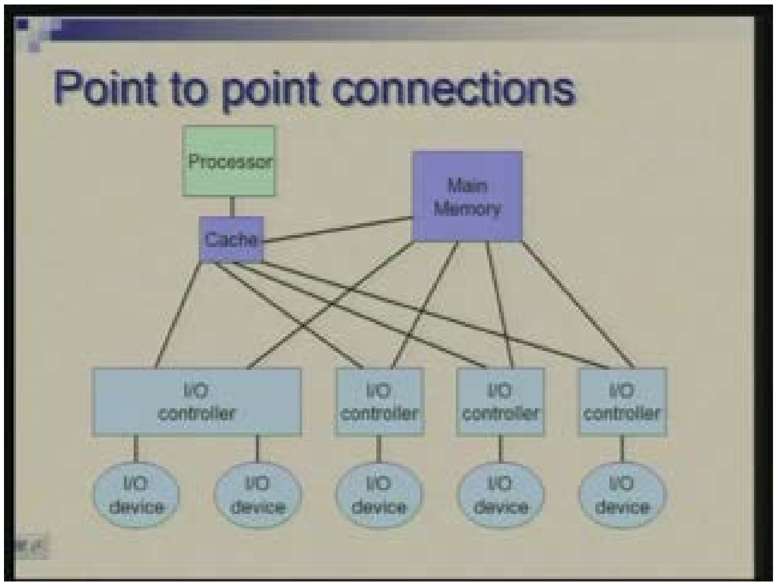


Computer system block diagram

So, beginning with this diagram which we had seen last time that you have processor, memory, cache is also shown with the processor is typically the case and on the other hand you have number of I/O controllers connecting to I/O devices. So a controller may connect to one or more I/O devices. Then we left this open (Refer Slide Time: 2:49) that there is something in between which connects these. So information needs to flow between processor cache combination and memory, between this combination and devices and between memory and the devices. Of course there is unlikely that information goes from one device to another device directly, it comes from this side to this side or vice versa.

So what are the different alternatives which exist or which can be conceived of for connecting all these subsystems.

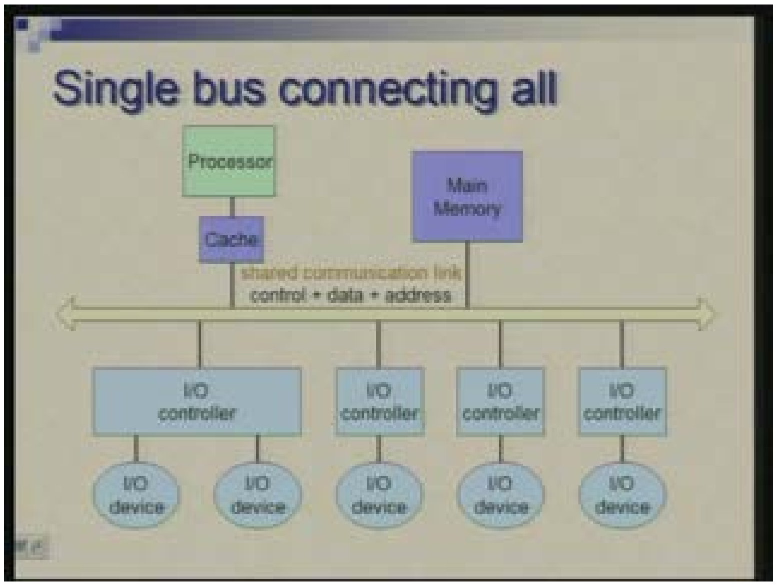
(Refer Slide Time: 3:25)



Point to point connections

This is one straightforward answer that you have point to point connection. That is, if you if you want memory to talk to this device so establish a connection. If you want cache or processor to talk to this device establish a connection, also connection between cache and main memory. So whichever pairs of devices or subsystem need to talk to each other there is a dedicated connection and alternative to this is a shared set of wires which is called a bus.

(Refer Slide Time: 04:00)



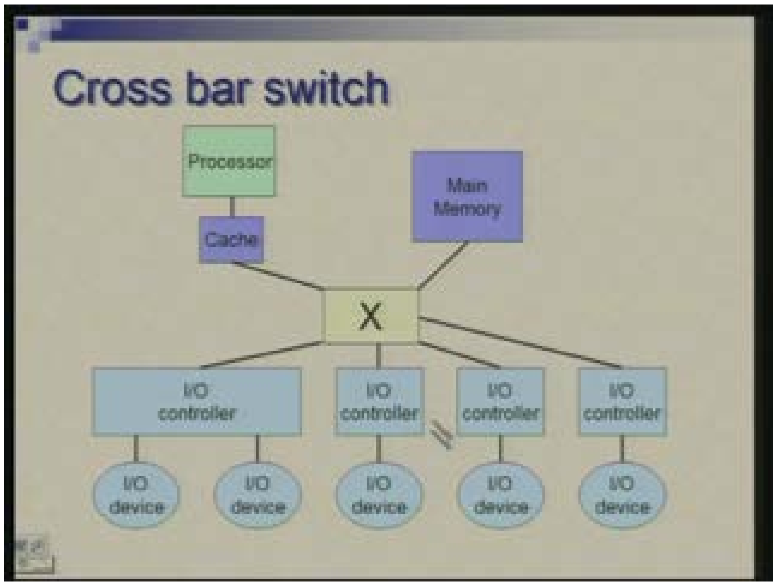
Single bus connecting all

Bus is nothing but a shared communication link where different types of information flow. So it is a collection of basically wires and signals which are carried on these wires which will connect two or more of subsystems so same wires run through everything. These wires would carry different types of information; it could be address which would be used to select to whom you want to talk to, the data which needs to flow in some direction and control which will tell what is going on on the bus, it does the timing and it also indicates the nature of information which is flowing. So the design of a bus is determined by how you organize these signals, what is the number of wires assigned to each and in terms of time sequence how this information is carried.

The basic idea here is (Refer Slide Time: 5:06) that you simply allow one set of wires to connect everything and obviously alimentation of this would be that only one pair can be communicating on this anytime. If there is more than one pair which need to talk to each other and there the conversation has to be sequenced but it is very convenient and cost effective.

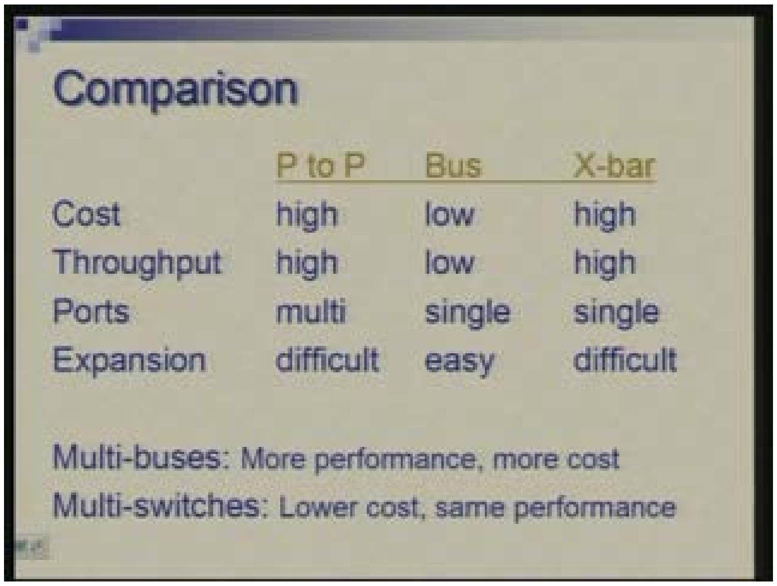
Yet another alternative could be to put some sort of exchange like you have a telephone exchange, a cross bar switch to which we connect everything and here responsibility of this box in between is to make establish connections. So, if this unit wants to talk to this unit then these lines will be connected.

(Refer Slide Time: 05:45)



In general, it should be possible to connect not just one pair but many many pairs. So this can allow multiple conversations to take place the same way as you will have in a point to point scenario. So how do these three alternatives is compared with each other.

(Refer Slide Time: 6:22)



Comparison

	P to P	Bus	X-bar
Cost	high	low	high
Throughput	high	low	high
Ports	multi	single	single
Expansion	difficult	easy	difficult

Multi-buses: More performance, more cost

Multi-switches: Lower cost, same performance

I am showing it here; point to point connection, bus and cross bar. In terms of cost, bus offers you a solution at a very low cost because it is just one set of wires whereas the other two are much more expensive. The throughput is low in case of bus whereas it is much higher in case of the other two solutions. In point to point you also require multiple ports. If you go back to the diagram (Refer Slide Time: 06:55), if I/O controller sitting here needs talk to processor as well as memory then it requires two separate ports because we are having distinct isolated connections and memory needs to talk to 1 2 3 4 5 five parties so there are five ports. So, having multiple port is an expensive affair. So the cost here is not just the cost of the links but this also is contributed to the cost. So the issue of multiple ports does not exist in other two cases.

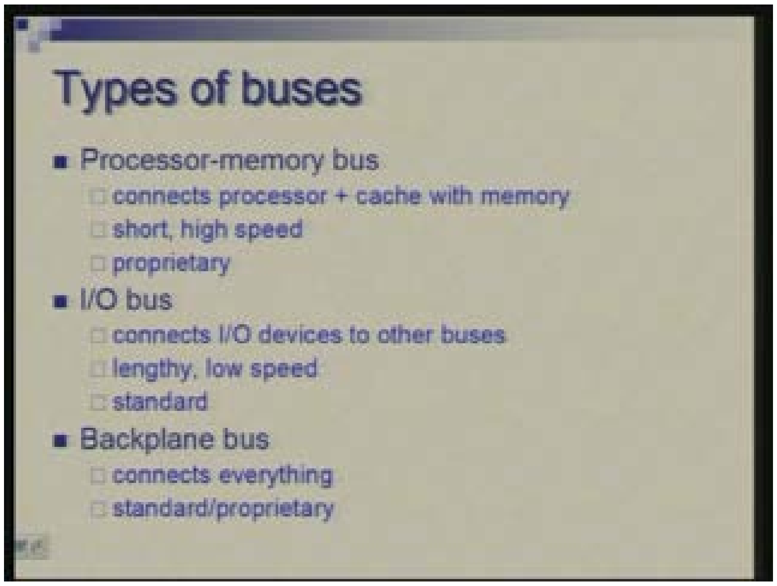
The next question is expansion. Once you have designed your system, you want to add more pieces of hardware to that. It is much easier in case of the bus because there is a uniform interface you can plug-in more and more things but in case of point to point or cross bar you would need to create more links or increase the capacity of the cross bar if you have more things to be connected.

Now these are only three specific possibilities but there are many more variations which are possible. For example, you could have multiple buses not just one bus. The limitation of one bus is having one conversation at a time. You can provide multiple buses and then more can go on. So there is a spectrum, you form a completely low cost low throughput solution you can slightly improve things by introducing more buses, you incur more cost but you also increase performance.

On the other hand, instead of having a monolithic single crossbar switch which connects everyone you could also divide into a fewer but smaller cross bar switches and that would be meaningful if everyone need not talk to everyone. For example, in our situation we do not necessarily want one peripheral device to talk to another peripheral device, so that complete crossbar capability is not required. So we can organize it in terms of multiple smaller switches and effectively reduce the cost.

What I mean here is that if, you let us say you have n units, a crossbar will make it possible for anyone to talk to anyone which means you have n into n minus 1 by 2 pairs. If the number of communicating pair is much then you can incur less lower cost, so that is the whole idea.

(Refer Slide Time: 10:10)



Types of buses

- Processor-memory bus
  - connects processor + cache with memory
  - short, high speed
  - proprietary
- I/O bus
  - connects I/O devices to other buses
  - lengthy, low speed
  - standard
- Backplane bus
  - connects everything
  - standard/proprietary

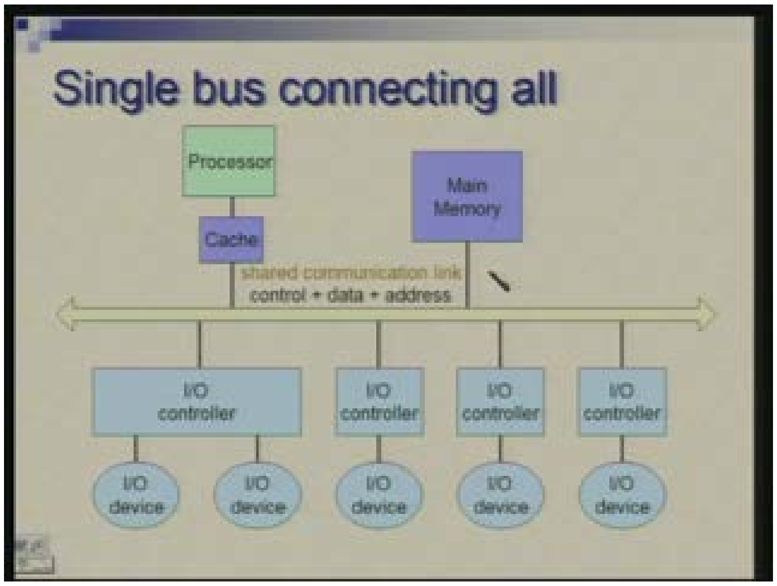
Now, from this point onwards let us focus on the buses which are most commonly used for interconnecting subsystems in the computer system. There are several types of buses or there are few....... it is a clear definitions which are possible. Once again the picture could get blurred if one mixes characteristics of these things. So you can have on one hand what is called processor-memory bus whose responsibility is to connect processor and memory. When I say processor it would typically mean processor and cache together. So such buses are required to be extremely high speed buses and they are short in length that means they do not spread out physically in the system, the design of these is proprietary, they are not standard; standardization is required when you need to connect a large variety of subsystems as you would see the case in the other two scenarios. So these are defined for a specific case.

Let us say you have a system based on Intel's Pentium 4. So, depending upon the signals which are there in Pentium you need to connect that to memory and a bus would be designed specifically to keep in mind the Pentium's behavior whereas if you connecting Spark processor to memory then the whole design will be oriented towards that and it is this is a proprietary design in the sense that it is not possible to just pick up any arbitrary subsystem and plug into it. Of course somewhere this would need to be connected to peripheral. So somewhere there has to be a path to the peripherals but primarily it connects processor and memory.

The second in our list is I/O bus where the main purpose is to connect I/O devices to rest of the system. So, number of I/O devices can be hung on the bus and this bus is carrying comparatively lower speed of traffic and this could be lengthy because these devices and their controller could be physically spread out, they could be larger in number and they could be physically spread out. This necessarily has to be standard because it is expected that there may be different devices which may be manufactured by different manufacturers and unless there is a common definition of the bus operation, the buses signals, the bus width data rate and so on it is not possible for multiple parties to design their system which are compatible. So we want interoperability and that same standards are required. We will talk of some examples of standard buses in the subsequent lectures. But this issue of standard here is very very important, that must be noticed.

Thirdly, we have what is called a backplane bus and this is more versatile in the sense that this may connect everything, it may connect processor, memory, I/O peripherals so everything could be actually hanging. So logically this is what we have shown in the previous diagram. So, if you look at this, if we really have a bus like this, this would be a backpanel backplane bus.

(Refer Slide Time: 13:44)

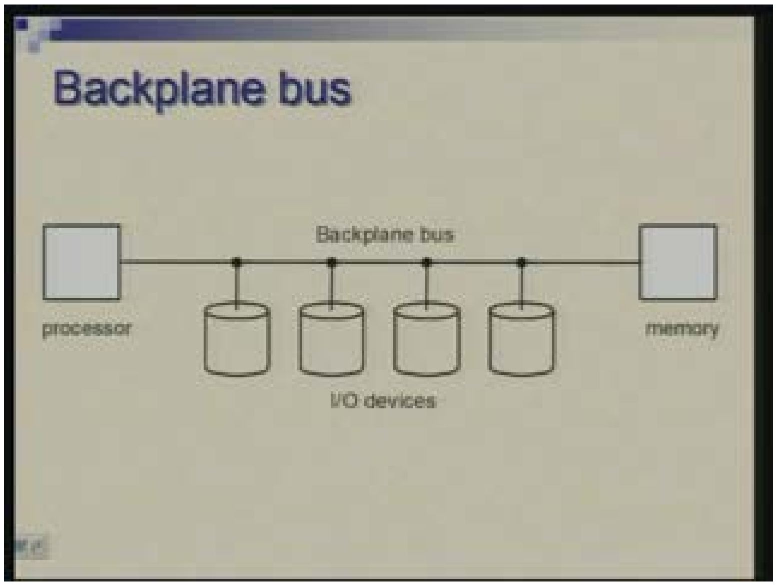


Question is why this term backplane?
This has historical reasons because earlier the way system was designed was there would be a cabinet or a box where each subsystem would be a separate card and they will all plug into a connector which are connecting a single bus and that would be typically in the backside; on the front side you would have LED displays and the switches, on the backside you have a bus running on which there are connectors, on which processor could be processor would be a single board or multiple boards not a single chip, memory could be one or more boards and each peripheral could be typically one board. So it was typically a bus connecting everything and physically located at the backside of the cabinet.

Now, if you look at a system like we have Pentium 4, there is a concept of what is called a motherboard and there is a bus running on that which would be something like backplane bus. But physically it is not a not a backplane bus. But in logical sense it could be backplane bus on which many boards could be plugged in with a difference that you do not have processor board and memory board plugging into that. So it is more for input/output.

Let me illustrate some of these ideas.

(Refer Slide Time: 15:40)

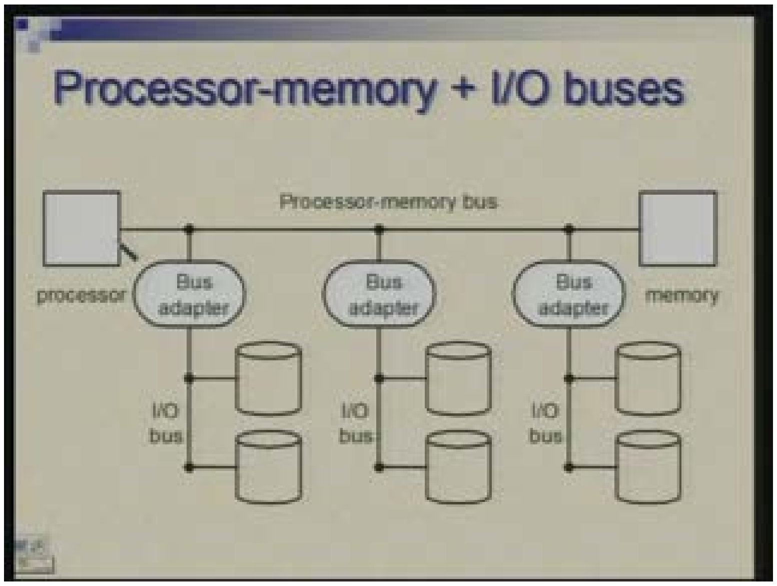


Backplane bus

Here is an example of backplane bus where you have processor memory and various devices, all connecting to the same bus. Here I am not showing the device and the controller separately. It is a complete subsystem which is shown as a single box here. But as I described last time, that there is a there is a controller which may be physically integral part of the device and there is a device which actually transforms information between 1s and 0s which CPU or memory will understand and an information which is more readable.

For example, in case of let us say a laser printer, what you need is characters of pictures in the printed form whereas what comes from memory process would be all binary encoded.

(Refer Slide Time: 16:28)



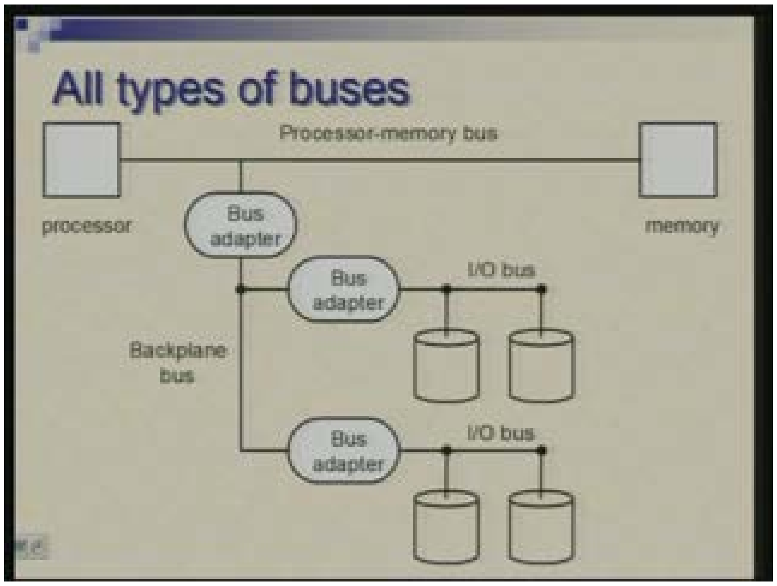
Processor-memory + I/O buses

This is an example where you can see a processor memory bus, so this bus is connecting a processor with the memory. But of course these two cannot be isolated we have to have input/output somewhere and input/output is organized in terms of I/O buses. There are multiple I/O buses here. Also, notice that although processor memory bus and backplane bus would be typically one in a system, I/O buses could be multiple. So this I/O bus takes care of these devices, (Refer Slide Time: 17:00) this takes care of these devices and so on and these I/O buses are further connected to processor memory bus through some adapters or interfaces.

Now why is it that we do not simply connect these; why would you separate?
The reason is that the requirement here, the behavior you see here and the behavior you see there would be quite different. The width of the data could be different; you may be transferring 32 bit or 64 bit data here, this could be for example just 8 bit, you are transferring character by character or byte by byte, the speeds here could be slower, the speed here is fast, here (Refer Slide Time: 17:47) it is primarily catering for processor memory and they occupy the bus mainly which is only the free slots which may be passed on to I/O for its usage. So this would be more expensive in terms of its design and cost and this could be comparatively cheaper.

So there are various reasons why one needs to separate and in fact multiple these different I/O buses which you are seeing here could also be different in characteristics. One I/O bus example could take care of the storage devices: disk and CD ROM and so on. There could be others which could take care of serial peripheral let us say this could be a USB serial bus where you may you may connect your scanner and maybe printer and so on. So, multiple buses again are for grouping peripheral devices according to their needs and characteristics.

(Refer Slide Time: 18:52)



Finally this is a picture which shows all the three types of buses. So, we have a processor memory bus in the same manner, they are multiple I/O buses but these I/O buses do not hook on to processor memory bus directly, there is a backplane bus which is connecting...... now logically this is the one which is connecting processor, memory and I/O, so everything is meeting here and looking at in more details there are bus adapters which connect backplane bus to the processor memory bus and also different kinds of adapters which connect this to I/O bus. So each of these adapters has to understand the definition of signals, the protocol, the speeds at the two ends and make them match. So each adapter would be a different design although I put the same name on each. So, in fact a typical sophisticated system will be quite like this that it has multiple buses and in fact our Pentium 4 is also following this approach. So what I said is a bus on the motherboard could be a backplane bus.

Yes, here we are grouping peripheral devices which are somewhat similar in nature, which would be, there would be a common interface here and the controller is within each device would make sure that the device is able to talk to that interface. So you are right that multiple devices will have their own variations and those variations will be taken care of here (Refer Slide Time: 20:50). So some adaptation is necessary but that is all built in here.

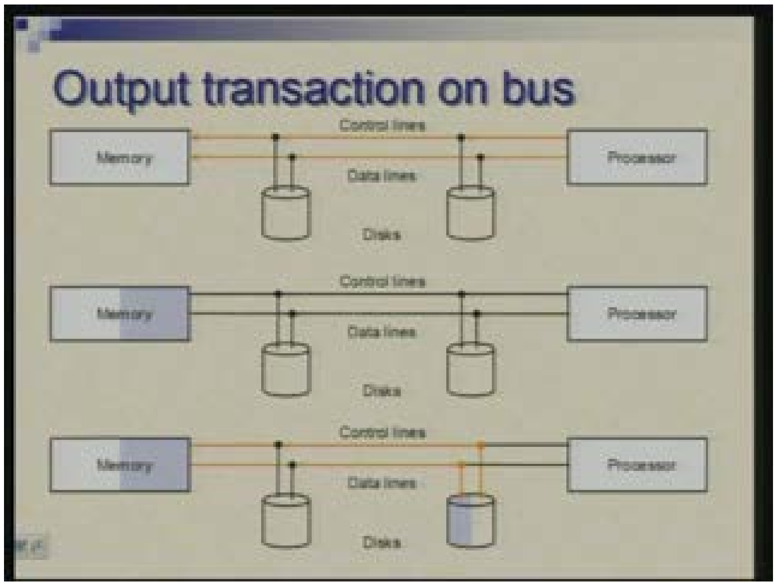
Now let us say it is a matter of how you allow sharing of information or sharing of that bus by multiple devices. So we will we will look at that. This is this is a very crucial issue that if you are connecting multiple devices how is it that you are you are getting....... I mean we have multiple buses, are we getting some parallelism here or not that is the question which we need to understand.

So, firstly as the concern was that would multiple devices hooked onto the same bus be able to communicate concurrently or not that is one question, then this whole system together are multiple transactions possible? So now ultimately if many devices simultaneously speak to memory that means there is a let us say data is being transferred from a disk to memory, some file is being read, at the same time something is being printed in a printer, information is being brought from the memory, so now at the memory end everything has to be sequentialized because we are assuming it to be a single port memory and at any given time it could be either reading a word or writing a word. So how is that concurrency is possible.

Concurrency would be possible in let us say localized area. So let us say the disk was here and printer was here (Refer Slide Time: 22:38); the disk data could be flowing on this bus at the same time as printer data is flowing on this bus but here they are coming on the on a common bus so here they will get sequentialized but what we need to understand is that the rate at which data flows would be different. Suppose the data is flowing at the rate of 5 megabytes per second here and say for printer it is flowing at 1 megabyte per second here and if suppose this capacity of this backplane bus is say 20 megabytes per second so here we have two different streams of data parallel but here they get sort of interleaved. And if you have designed this bus to take care of the traffic it is like you have smaller lanes on which some traffic is coming and here on a bigger road it merges and from here it is going on to even a faster highway. So, on the whole there is, yes, at the system level this kind of a structure brings in some concurrency. At some point things may have to be sequentialized but as long as we have been able to keep the disk and the printer busy we have utilized there capacity in the best manner. So this is how one could visualize it.

[Conversation between student and Professor: (24:09)] ....does it answer your question? Yeah, yeah, yeah it is managed in terms of time sequence. So as I was saying, let us say, coming back to same figure, suppose backplane bus is able to carry the data at the rate of 20 megabytes per second so just for the sake of argument let us say each of these is 1 byte wide which means that every 50 nanosecond each data each byte of data occupies 50 nanoseconds, sorry yeah 50 nanoseconds and each byte of data occupies this bus for.... I said how much? 5 megabytes or 200 nanoseconds. So this adapter will take this data coming 1 byte every 200 nanosecond and try to accommodate it somewhere here. This is where you have; let us say 1 megabyte per second. So every 1000 nanosecond or microsecond data is coming which needs to be accommodated here. So these things could be in some way sequenced. Whether these are interleaved at the byte level or they are interleaved at a packet level or group of byte level that depends upon how you organize this bus. It may be that this adapter may collect several bytes and put them as a bust on this. This one also is collecting its own bytes (Refer Slide Time: 25:50) and when it gets a chance it puts its own packet on the bus. So the key thing here is that if bus has a higher capacity it should be possible to put things in sequence interleave them in some manner.

(Refer Slide Time: 26:11)


Output transaction on bus

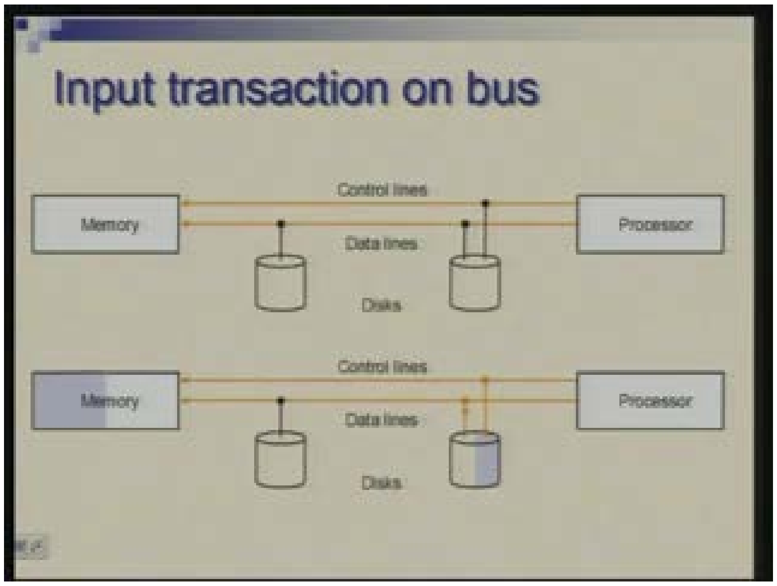
Now what actually goes on on a bus?
Here is an illustration considering that a processor memory and some devices like disk are sitting on the same common bus. These ideas could be extended to various bus organizations but let us look at the issue of input and output transactions on a bus and once again this would be one possible protocol and lot of variations exist.

So here we will assume that there are the bus has two types of lines; they are control lines which tells you what happens in the bus and when it happens the other data lines apart from data they also carry address. Since they are multiple devices there is a need to address them and also since memory is sitting on the bus there is a need to address the memory words or memory bytes. So we are assuming here that the address and data flow actually on the same physical lines.

Now we are trying to show an output transaction that means something is to be read out from memory and sent to a device. So we will try to shade memory or the device reflecting whether it is reading or writing. So if right half is shaded it indicates reading and if left half is shaded it indicates writing.

So, to begin with, address is sent to the memory and the control lines will indicate that now address is being sent, it is not the data. So, memory would use that address and try to internally pick up the word or byte which is being addressed and bring out the data then the data will flow on these data lines and this disk is supposed to take that data and absorb it, write it. So at this point of time (Refer Slide Time: 28:33) we also need to tell the disk drive that yes, here is the data which is flowing out and it is meant for you to write. so there is a What we are trying to show here is that there is a sequence, there is some protocol, there is some sequence of information flowing back and forth which is involved. So memory needs to be given an address and it responds with a data or memory would need to be given address it would need to be given data which needs to be written and then it will take time and record it.

(Refer Slide Time: 29:11)

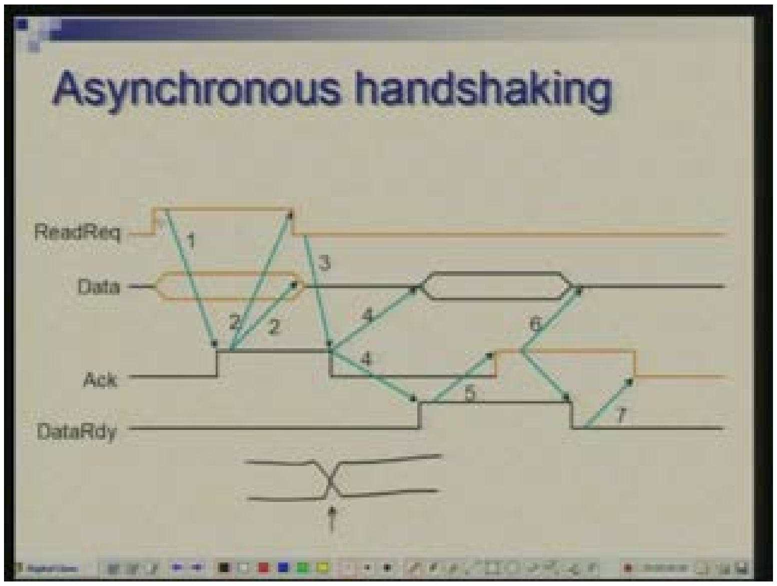


Input transaction on bus

This picture shows the opposite operation where you are reading from the disk and writing into the memory. So, again memory is given the address and told that yes there will be data following which you need to write and disk is told to read data, this data goes out and memory is supposed to write that.

Now how this happens; exactly what is the nature of signals is what we will we will take up shortly. There are two ways in which this information could be organized or these transactions could be organized: asynchronous and synchronous. In asynchronous case there is no clock involved to synchronize the events. Basically the party which are involved will tell each other what needs to be done next. So let us let us look at a case of reading from the memory. So the party which is trying to read from memory will send a signal so it is a control signal which I am calling as read request (Refer Slide Time: 30:27). So when the signal is activated the meaning of this which is understood by the memory is that something has to be read, memory has to do read operation and send out data. The data bus while this signal is active is carrying the address, so it has to be understood by both the parties that at this time what data bus is carrying is actually the address. So the processor which is sending the address and the memory which is receiving the address they should understand what it means.

(Refer Slide Time: 30:55)


Asynchronous handshaking

In response to this change of read request 0 to 1, memory would send an acknowledgement that yes it has seen the read request. Now, in asynchronous transaction all that we want is that this acknowledgement should come in response to read request, we are not saying that it should come after a specific time; it could it could come after 1 nanosecond, it could come after 10 nanoseconds, what is important is that this carries a meaning, this acknowledged signal is going from 0 to 1, indicates that memory has noticed that read request has come and it has picked up the address. This read request could have actually been originated from processor or it could be from I/O device. So whosoever had sent this request will keep looking at acknowledged signals and when acknowledged signal becomes 1 then this can be lowered.

So this is a confirmation, this change or this event is a confirmation that this event has been noticed. So there is a sequencing shown by arrow 1, another sequencing shown by arrow 2 so these have to be done these events have to occur in this particular order. And another response to this change is that, since data has been read it need not be kept on the bus, so it can be removed from the bus and bus becomes free.

Now, what is this hexagonal shape indicating (Refer Slide Time: 32:43)?
This basically is indicating that there is a change occurring here. let me......... see, generally when you are talking of an individual signal you can show it going from 0 to 1 or 1 to 0 but when you are talking of a bunch of signals some may be going from 0 to 1, some may be going from 1 to 0 so you will often show it like this (Refer Slide Time: 33:22). So this is an indication that here is a transition point where some change is occurring. We are not particularly worried about what change is occurring, this indicates that some bits may be going from 0 to 1 and some bits may be going from 1 to 0 so this is a change of the same nature but you notice here a third level. This third level is a level indicating that bus is free, there is nothing on it. These buses are organized as what is called a tri-state bus. it could be in There are three possible states: 0, 1 and a high

impedance state. High impedance state means that everything which is connected to the bus is in fact disconnected. It is neither trying to put a 0 nor trying to put a 1. So, if everyone isolates from the bus the bus is in a floating state; there is neither low impedance path to ground or zero level nor low impedance path to power supply or one level so it is a floating state. And symbolically this is shown as an intermediate level. So what we are indicating is that data bus here upto this point (Refer Slide Time: 34:36) was floating, it was a high impedance state, here some bits go to 1 some bits go to 0 and this state continues. At this point the data is removed not the data but the address is removed from the data bus by the device which was driving it and bus floats again.

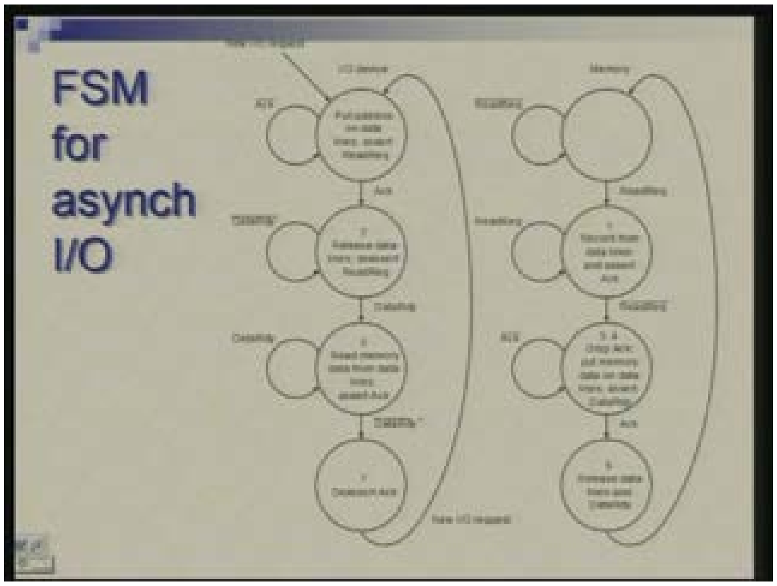
Now, this sequence actually needs to be completed by lowering the acknowledged signal. So memory will, after raising acknowledged signal, it will keep monitoring this and when this is gone down, when it has noticed that this has gone down this will also go down so these are the four events 1 2 3 4 which completes a cycle which is called handshaking.

Now, after memory has seen the address it does its homework and gets back with data at some point of time. when it is ready with the data it will put the data on the bus and activate the signal which says that now data is ready otherwise the device which is expecting data has no other means to know when the data is available from memory so it is this signal which indicates the data is available and now it is the turn of that device to acknowledge. So this change is a recognition of a fact that the device as seen that data is ready, it has read the data and it has confirmed that I have read the data. Having seen that memory will remove the data from the bus and also lower this signal which is an acknowledgement of the fact that the data has been read by the it has seen that data has been read by the device and then device will lower the acknowledged signal saying that the cycle is completed that it has recognized that data ready has gone down (Refer Slide Time: 36:45) so again there is a sequence of four events which occur in a particular order in a lock manner.

Is there any question about this? this is a from
From a bus to bus there could be variations in terms of nomenclature of the signal or whether you get a positive pulse or a negative pulse and the exact sequence in which it occurs but the basic idea is that of handshaking. So you see no clock here, it is just that one event is timing the other, so one event triggers the other that event that event triggers next one and so on and that is how the transaction completes.

I wonder if it is readable, can you read the text here. I will; sort of read it out aloud.

(Refer Slide Time: 37:44)

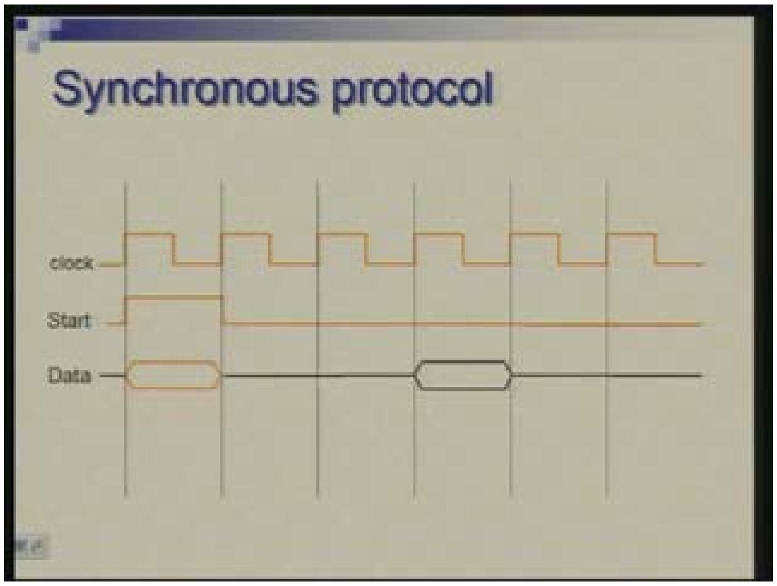

FSM for asynch I/O
New I/O request
I/O device
Memory
Ack
ReadReq
DataRdy
Put address on data lines; assert ReadReq
Release data lines; deassert ReadReq
Read memory data from data lines; assert Ack
Deassert Ack
Record from data lines and assert Ack
Drop Ack; put memory data on data lines; assert DataRdy
Release data lines and DataRdy
New I/O request

Now for this handshaking to occur, internally the I/O device and memory have to go through a sequence of states. So there is a controller within each of these which has to step through a sequence of states and in each state it responds to some changes which are observed and noticed. So the cycle......... this is the state diagram (Refer Slide Time: 38:28) for the device and this is for the memory. So we will start here. In this state the requesting device is putting address on data line and asserts read request signal whereas on the other hand memory is in this state and it is waiting for read request to become 1. So, while read request is 0 it keeps on cycling in this state, when read request becomes 1 it comes to this state where it observes the data lines which are carrying the address, it remembers the address and it asserts the acknowledgement line.

On the other hand, after having done this it will be watching the acknowledgement signal. As long as acknowledgement signal is 0, it remains here in this state (Refer Slide Time: 39:18). As soon as the acknowledgement signal is seen this proceeds to this next state and in this state it will release the data lines and deassert read request whereas on the other hand, after having sent acknowledgement signal, memory would be waiting for read request to go down and it will remain in read request state as long as it will remain in state 1 as long as read request is 1. So you would notice that the transitions in these two state diagrams are occurring one by one. So one transition occurs here, then one occurs here, one occurs here, one occurs here and so on. So you could follow these diagrams and see that the behavior which we had described in the previous waveform is captured here.

Basically this is indicating how this action will be carried out inside the memory and inside the I/O device. So they would have to have states like this which observe occurrences on the bus and accordingly carry out state changes inside.

(Refer Slide Time: 40:30)

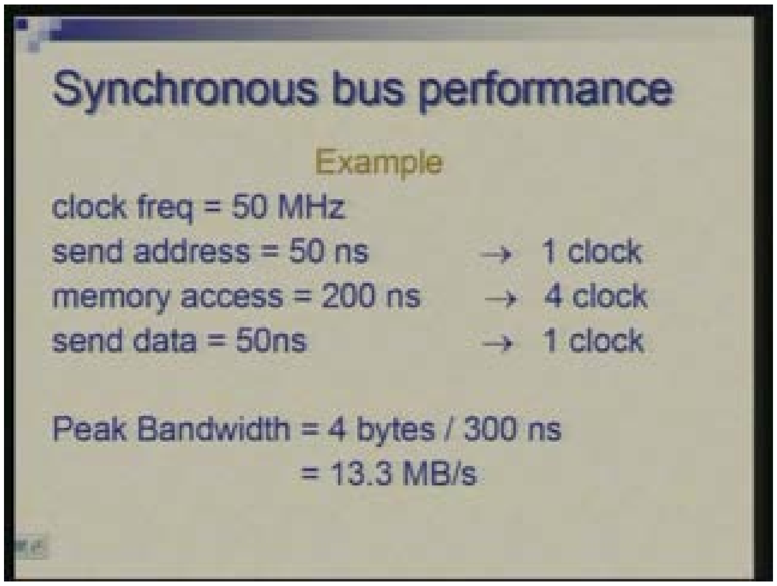

Synchronous protocol

Now let us look at the synchronous bus. In a synchronous bus there is a clock which is timing everything. In a complete system if there are multiple buses and multiple synchronous buses they may have their independent clocks. So one bus may have 20 megahertz clock, the other may have 5 megahertz clock, the other may have 100 megahertz clock, they could be different depending upon the data rate they need to carry. So in this case all events will get timed by the clock.

I am trying to show, let us say, raising edge of the clock is active edge, all changes will occur here, somewhere there has to be an indication that this is the first cycle where transaction begins. So I am showing it with a signal called start and there will be address first of all put on the bus, it could be available for one cycle or two cycles or multiple integral number of cycles, most common is for one cycle. Suppose again we have a similar situation that a device is trying to read memory so this address goes to memory (Refer Slide Time: 41:38), memory does its homework and it is expected to come up with data after fixed number of cycles.

So in this case we are assuming that with a gap of two clock cycles the data will be available so the device can make this assumption that bus address has been given without waiting to check whether memory has seen the address or not seen the address, no confirmation is sort after and you simply wait for two clock cycles and then pick up data from the bus. It is assumed that memory understands this and everyone else sitting on the device sitting on the bus understands that. So all these assumptions are there in terms of what happens in which cycle. You do not see too many waveforms here, everything happens at a fixed clock cycle.

(Refer Slide Time: 42:36)



Synchronous bus performance

Example

clock freq = 50 MHz		
send address = 50 ns	→	1 clock
memory access = 200 ns	→	4 clock
send data = 50ns	→	1 clock

Peak Bandwidth = 4 bytes / 300 ns

= 13.3 MB/s

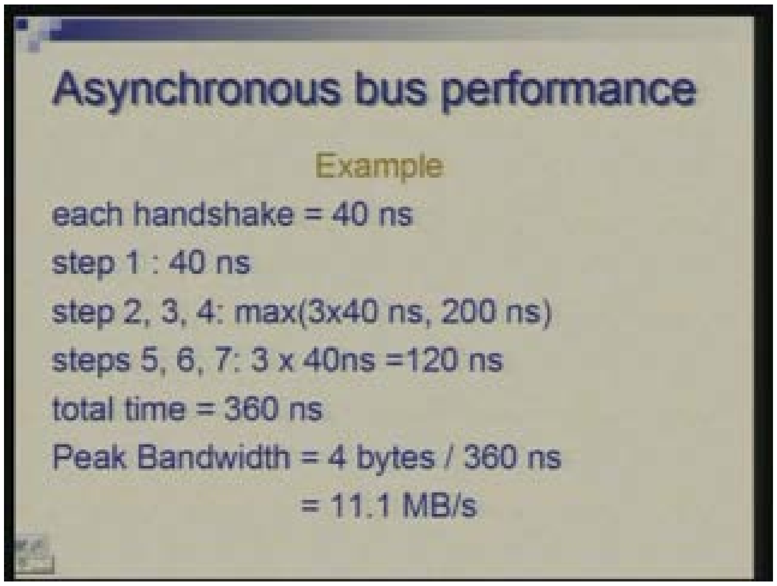
Now how do these alternatives compare and how throughput could be determined depending upon the protocol. So let us take an example.

First we will look at asynchronous bus which is running at 50 megahertz and suppose it takes the address needs to be put on the bus for at least 50 nanoseconds which means one cycle, memory takes let us say 200 nanoseconds to respond after bus, after address is given which means we take it as fixed four clock cycles and suppose 50 nanoseconds are required to send the data again we interpret this as one clock cycle.

Now the entire sequence is very clear. One cycle you send the address, then four cycles you do nothing, wait and then have another cycle to transfer the data. So, on the whole, you require six cycles or 300 nanoseconds for transferring one word. Let us assume that the bus width was 4 bytes. So, in one transaction like these 4 bytes are transferred and therefore the peak bandwidth, if bus was continuously doing this then it could be 4bytes divided by 300 nanoseconds which is 13.3 megabytes per second. So bus bandwidth or bus throughput would be characterized in terms of megabytes per second and this is how it could be related to the basic operations.

Let us take example of a similar speed asynchronous bus.

(Refer Slide Time: 44:20)



Asynchronous bus performance

Example

each handshake = 40 ns

step 1 : 40 ns

step 2, 3, 4: max(3x40 ns, 200 ns)

steps 5, 6, 7: 3 x 40ns =120 ns

total time = 360 ns

Peak Bandwidth = 4 bytes / 360 ns

= 11.1 MB/s

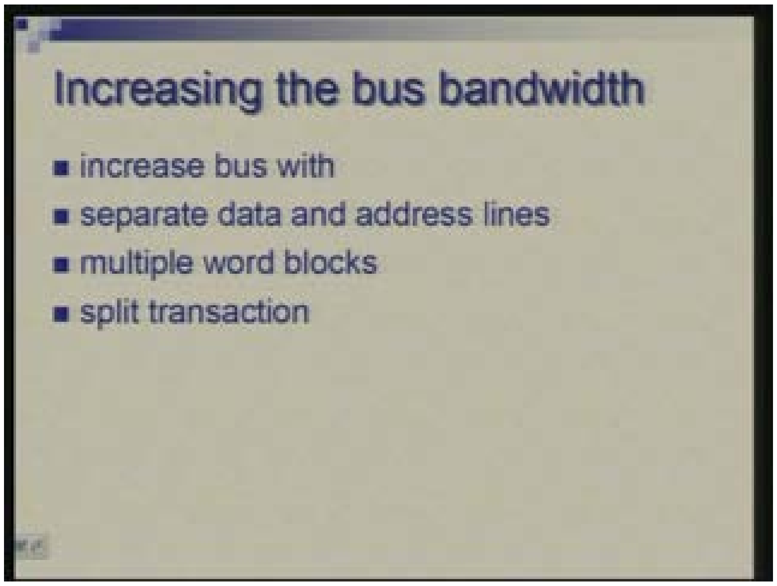
Here there is no constraint on timing in term in terms of when one event is coming in response to the other event what is the interval, it could be very small, it could be very large. But let us for the sake of calculation let us assume that each response is taking a minimum of 40 nanoseconds. So every time we are chaining two events we have to allow for 40 nanoseconds and if this is the minimum time we will get the maximum throughput by this assumption.

So now the step 1 2 3 4 5 etc these pertain to this diagram (Refer Slide Time: 45:06). This is a chain of events 1 2 then 3 4 5 and 6 followed by 7. So, first step takes 40 nanoseconds. It is basically delay between one event and the other event. If you look at 2, 3 and 4 which means this delay, this interval, this interval and this interval effectively the interval from this point acknowledgement to the availability of data (Refer Slide Time: 45:45) the minimum as far as bus protocol is concerned is 40 plus 40 plus 40 120 nanoseconds. But this is the time memory also has to be allowed to get its data. So we are seeing that it is max of these two quantities, 3 times 40 and 200 nanoseconds which in this case happens to be 200 nanoseconds and then there are three more steps 5, 6 and 7 each requiring 40 nanoseconds so another 120 nanosecond goes for that. Adding all this, the total time for all 1 to 7 adds upto 360 nanoseconds and once again assuming that bus is one word wide we get total bandwidth of 11.1 megabyte per second.

Well, what this example is indicating is synchronous bus is slightly faster which indeed is the case. Of course we have picked up........... if instead of 40 we take 30 things may be different, but the relative values are quite realistic in the sense that synchronous buses are generally faster because you are not waiting for sensing the signals and acknowledging them, you make assumption and simply go by that.

What are the factors which can be used to increase bus bandwidth?

(Refer Slide Time: 47:21)



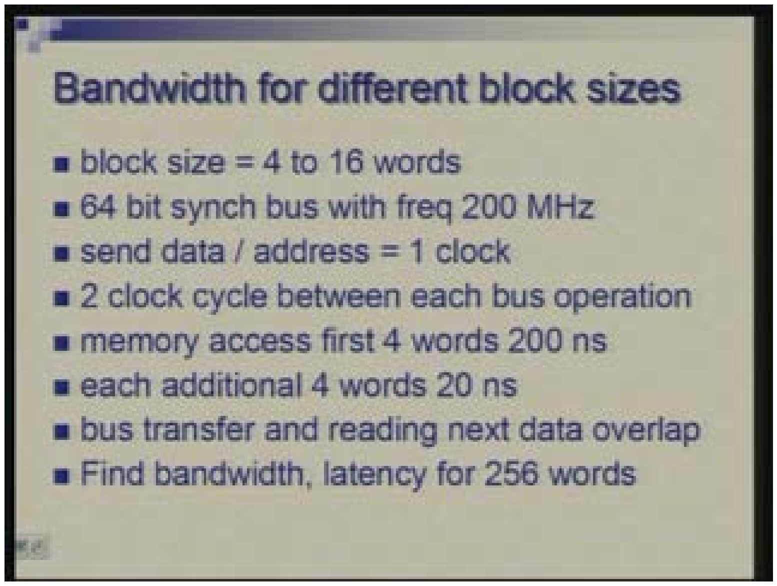
Increasing the bus bandwidth

- increase bus with
- separate data and address lines
- multiple word blocks
- split transaction

You could increase the bus width itself; instead of 32-bit wide bus you could have 64-bit wide or 128-bit wide as we have seen earlier when we were talking of block transfer between cache and main memory, you could separate data and address lines so that wherever possible the two could be carried two different thing at the same time. You could transfer multi-block words; rather than taking of one word at a time you could transfer multiple words as we have seen earlier and we will quantify it further in a moment. You could also have what is called split transaction. Split transaction means that between initiation of a transfer and completion of that you would have noticed that buses lie unutilized; the interval here or (Refer Slide Time: 48:12) the interval here. So, if this delay is large we can do something on the bus, we can initiate another transaction.

So a transaction gets split into two parts: initiation part and the ending part and there could be something else which could sit here. I will not going into details of at the moment and close by just looking at this example.

(Refer Slide Time: 48:38)



Bandwidth for different block sizes

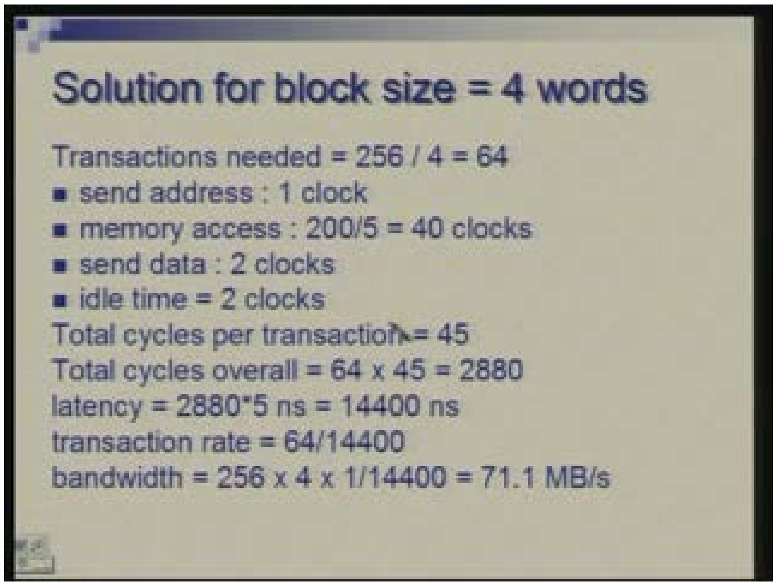
- block size = 4 to 16 words
- 64 bit synch bus with freq 200 MHz
- send data / address = 1 clock
- 2 clock cycle between each bus operation
- memory access first 4 words 200 ns
- each additional 4 words 20 ns
- bus transfer and reading next data overlap
- Find bandwidth, latency for 256 words

Let us look at two scenarios of a synchronous bus where what you are varying is the size of a block which you are trying to transfer. So now we are not looking at just one word but multiple words which need to be transferred. Suppose we like to talk of block size varying from 4 to 16 words it says 64 bits synchronous bus frequency is 200 megahertz which takes one clock to send the data or address, two clocks between each bus operation, so one transaction then you have to have a rest of two clock cycles. So this is a protocol which has been defined for this bus.

Memory access takes 200 nanoseconds to get first four words but subsequent words could be faster. Let us say, subsequent words come at 220 nanoseconds each so this could be by interleaving or page mode in a DRAM.

And bus transfer and reading next data overlap: that means let us say address was given, you are getting one data which is getting transferred in the bus, meanwhile memory is working on the next word so that 20 nanoseconds or the next word are overlapping with the transfer of the previous data. So with this scenario we can we find the bandwidth and latency for transferring 256 words. Total we need to transfer 256 words which we will see as to what happens if you make blocks of size 4 or blocks of size 16. So with the block size equal to four words you will require 64 transactions to get 256 words.

(Refer Slide Time: 50:30)



Solution for block size = 4 words

Transactions needed = 256 / 4 = 64

- send address : 1 clock
- memory access : 200/5 = 40 clocks
- send data : 2 clocks
- idle time = 2 clocks

Total cycles per transaction = 45

Total cycles overall = 64 x 45 = 2880

latency = 2880*5 ns = 14400 ns

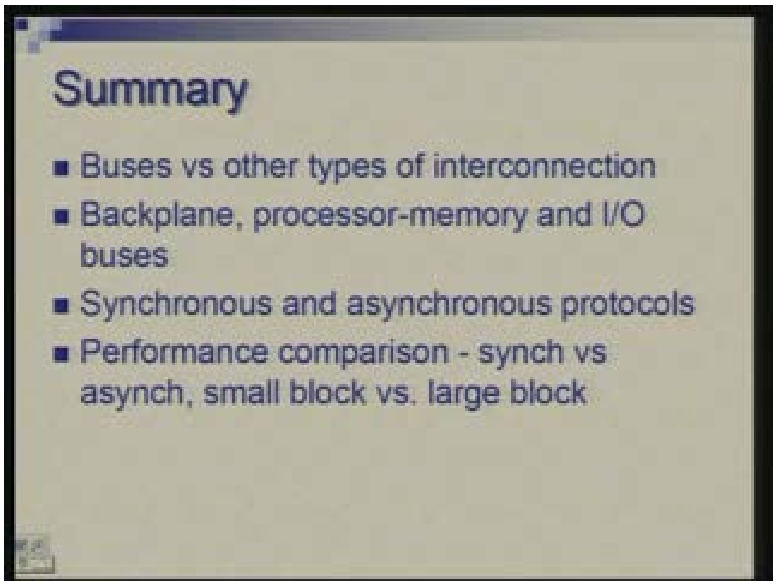
transaction rate = 64/14400

bandwidth = 256 x 4 x 1/14400 = 71.1 MB/s

We can count the cycle 1 for sending address, 40 for memory access because 200 nanoseconds is the access time, 5 nanoseconds is the clock period, we send the data over two clocks and then there are two cycles of idle time. So adding this altogether the total cycles are 45; this is 1 plus 40 plus these 4 so overall total number of cycles is 64 multiplied by 45 these many cycles you will require for entire 256 words and each cycle being 5 nanoseconds the total latency in terms of time is 14400 nanoseconds and the transaction rate, how many transactions per second we are carrying out is that we have 64 transactions those many nanoseconds or effectively the bus bandwidth is (Refer Slide Time: 51:41) so many words or so many bytes over that much of time, so 71.1 megabytes per second so this is a four word block.

The same exercise we can do with sixteen word blocks, the number of transactions reduces. I am sorry this is....... this is a mistake, so this should be 16, this should be 16, this should also be 16 (Refer Slide Time: 52:05) and this remains same, this remains same, here we have to do it four times because we are talking of sixteen words so this total adds up to 57, the total number of cycles is 16 which would have been the number of transactions into 57, latency, this number multiplied by 5 nanoseconds, transaction rate 16 transactions over this much time that means 3.51 million and bandwidth is these many bytes over these many nanoseconds so 224.56 megabyte per second. So you could see a tremendous increase in the bandwidth. Let me stop at this point and summarize.

(Refer Slide Time: 53:03)



Summary

- Buses vs other types of interconnection
- Backplane, processor-memory and I/O buses
- Synchronous and asynchronous protocols
- Performance comparison - synch vs asynch, small block vs. large block

We looked at different interconnection alternatives. Bus was the most popular among those. We talked about three different types of buses namely backplane bus, processor memory bus and I/O bus and we noticed that I/O buses necessarily have to be standard buses, standard which are applicable across multiple organizations whereas processor memory buses are proprietary and backplane buses could be either of the two.

The two types of protocols of the bus we have seen: synchronous and asynchronous. Synchronous is generally faster. We have also seen the effect of choosing a suitable block size in terms of how it improves the bus bandwidth. I will stop at that, thank you.