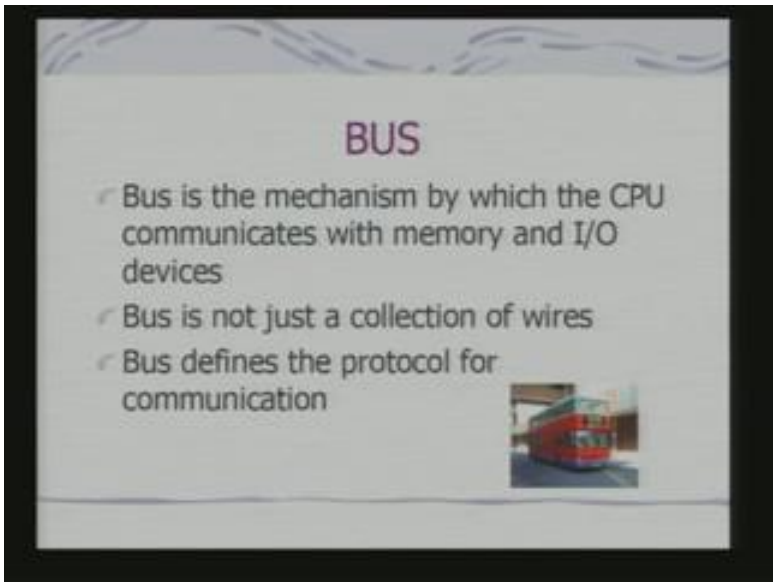
Embedded Systems
Dr. Santanu Chaudhury
Department of Electrical Engineering
Indian Institute of Technology, Delhi

Lecture - 14
Bus Structure

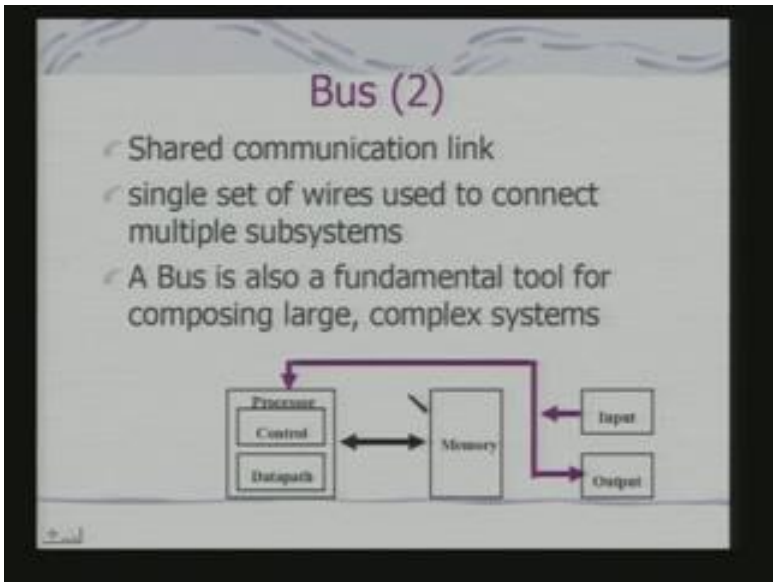
Today, we shall discuss about the Bus Structure that is how a CPU communicates with memory and devices.

(Refer Slide Time: 01:34)



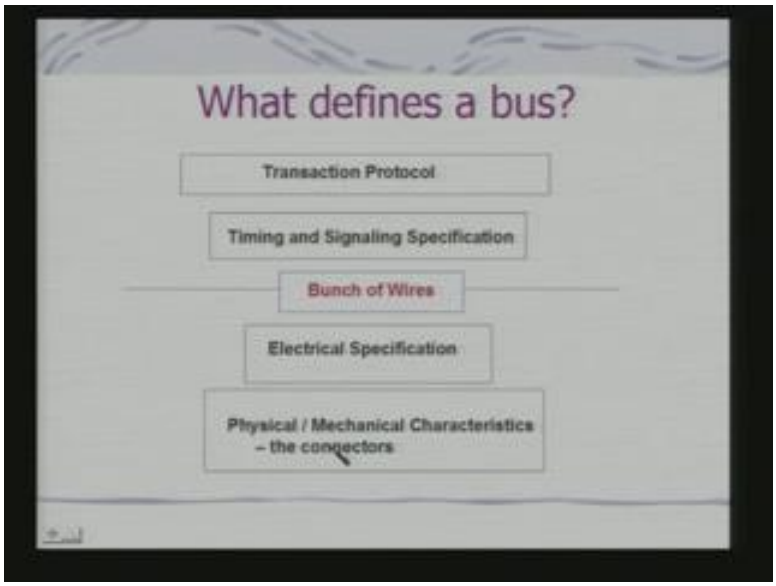
So, for this communication CPU vs bus; bus is the mechanism by which CPU communicates with memory and IO devices. And it is not just a collection of wires. In fact, bus defines the protocol for communication. In fact, you can see that it bus is a transport device then the bus in the context of Embedded System or computers is also defining the corresponding transport mechanism between the CPU and memory and IO devices.

(Refer Slide Time: 02:12)



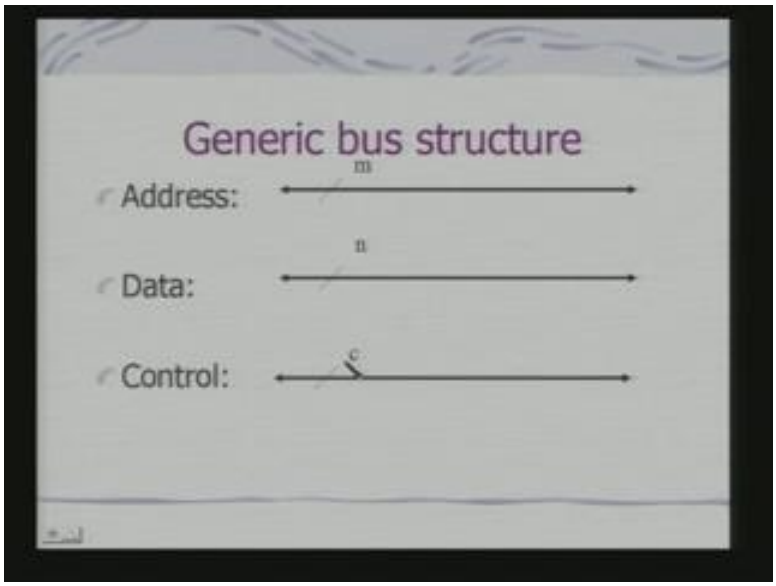
So, typically bus is a shared communication link, because there a mini devices which can actually sit on the bus. A single set of wires at therefore, used to connect multiple subsystems. Bus is also fundamental tool for composing large and complex systems why? Because a bus defines the mechanism for communication and once I have to put in a complex sub system along with CPU. That subsystem should have the ability to communicate with the communication scheme specified by the bus. So, a bus enables as to build a system on the basis of individual components which are compatible with the bus. Here, we have shown an example an in this example the interesting thing to notice that I have shown not 1 bus, but 2 buses. One bus connecting the memory and the processor another bus input at output devices to the processor we may have this kind of two distinct buses or else we can have a single bus and both memory and input at output devices sitting on the same bus itself.

(Refer Slide Time: 03:43)



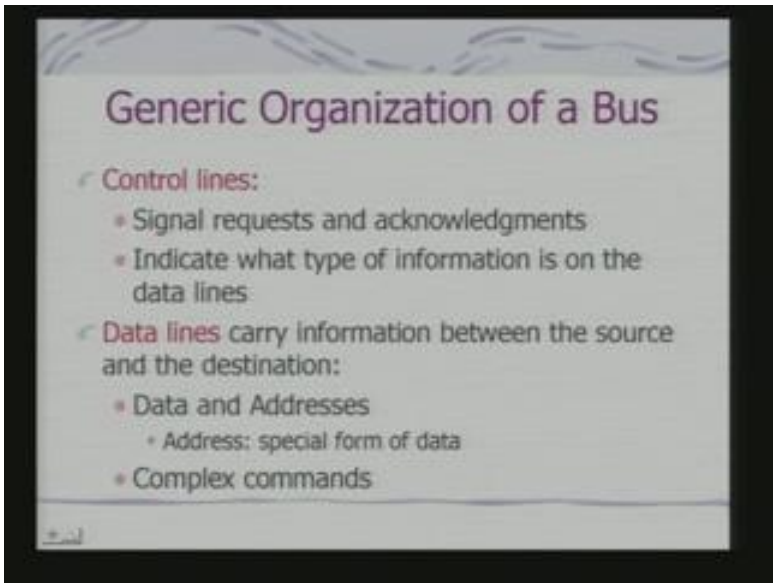
So, what defines a Bus? All these aspects are important for definition of the bus. First is Transaction Protocol, how really the exchange of data takes place along the bus? How the devices really talk to each other? What is a language by which they talk to each other? And for talking you need to have the Timing and Signaling Specification and this signals are obviously, carried by a bunch of wires. So, I have the Electrical Specification for the signals which are carried by the bunch of wires. But where will these wires actually lie that is defined by physical or mechanical characteristics as well as the nature of the connectors. So, starting from mechanical to the top level transaction protocol all these things together constitutes a bus.

(Refer Slide Time: 04:45)



A Generic bus structure will have; obviously, address data and control. So, I am showing the number of lines I can have for address for data for control m n and c. And accordingly I shall be setting up the connection or the communication link.

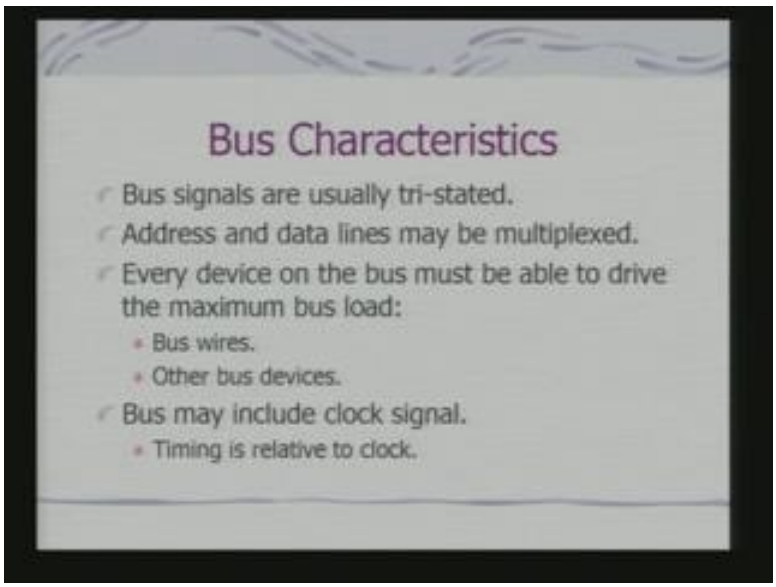
(Refer Slide Time: 05:10)



So, what for the control lines? Control lines actually implements the transaction protocol. The signals which flow along the control lines are really instrumental in implementing the transport protocol. The signals request an acknowledgement. So, these are request an acknowledgement signals are the 2 basic and generic types of signals you will find on the

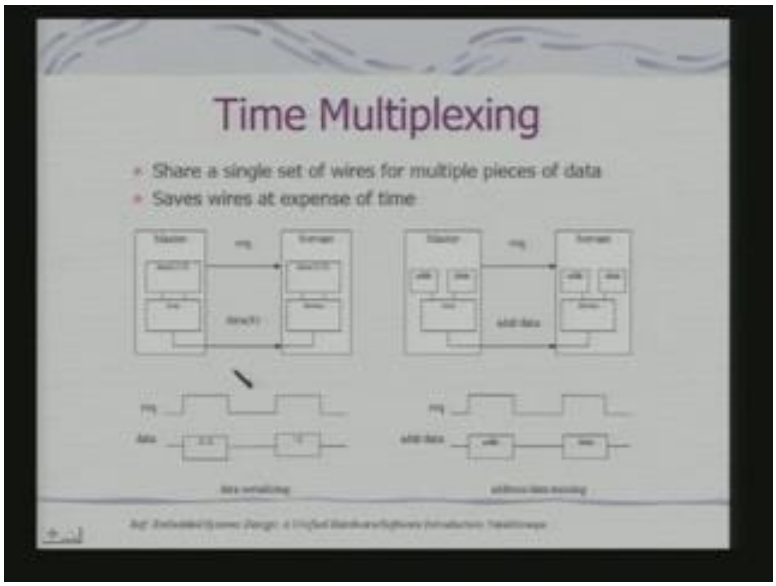
bus also the signals indicate what type of information is on the data lines? Data lines carry information between the source and the destination. In fact, we have clubbed data and address together, because address is nothing, but a special form of data. Also there may be complex commands which may be given to devices through data lines through the bus.

(Refer Slide Time: 06:10)



Bus signals are usually tri stated. Now, why tri stated, because when a particular device let us consider the interface of the device to the bus. So, if a device would like to disconnect itself from the bus, it would drive its interface lines to tri state high impedance state. So, effectively it gets disconnected from the bus. So, all these devices are really sharing the common set of signal lines. So, typically bus signals are tri stated. In many cases address and data lines may be multiplexed. This would say on the number of actual connections or wires. The other interesting thing to be noted, that every device on the bus must be able to drive the maximum bus load. Because maximum bus load would determine what are the maximum number of device that you can actually put on the bus. So, it has to drive the bus where is another bus devices to deliver the signal so, that is the compatibility with regard to the drive. Bus may include a clock signal. It is not necessary that all the buses or clocked, but some of the buses do include clock signal. And all the timing, timing particularly for bus signals which implement the complete communication protocol is defined relative to the bus clock. Now, the bus clock may not be same as that of the system clock of the processor.

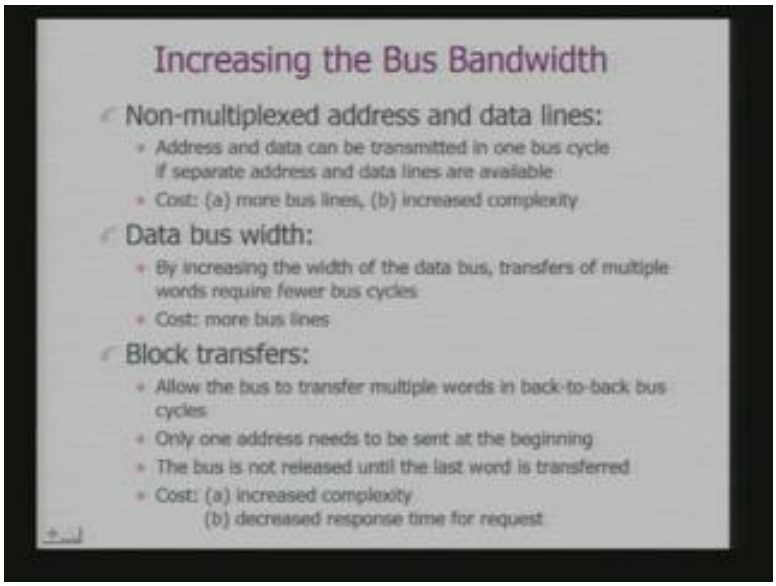
(Refer Slide Time: 07:56)



Here we are showing the time multiplexing. So, when there is a single signal line which is used for carrying two kinds of signals we do have time multiplexing. I am showing here a master another is a servant device. Now, master is requesting the servant to receive data or to get data from and the data is flowing here. Now, the interestingly there is no address pass that is where will these data be stored by the servant is not being specified by a separate address lines what is happening is the same data line which is of 8 bit width is being used for these purpose. So, see how it has been done in this case. So, you have the address as well as the data. So, when you first send the request you have the address which is being sent next time with the next request you are sending the data.

So, the address and data is getting synchronized with respect to the request signal. In these cases what is happening is the master is likely to transfer 16 bit data, but you have got 8 data lines available. So, you have to do multiplexing. So, if the first request, you are sending the more significant byte and then you are sending the less significant byte. It can be other way also. So, what you can see is here the 16 bit transfer I would have required 16 lines instead of that I am doing it using 8 lines. But obviously, I am saving wires at the expense of time, because I would to require 2 clock periods for transferring 16 bit data.

(Refer Slide Time: 10:05)

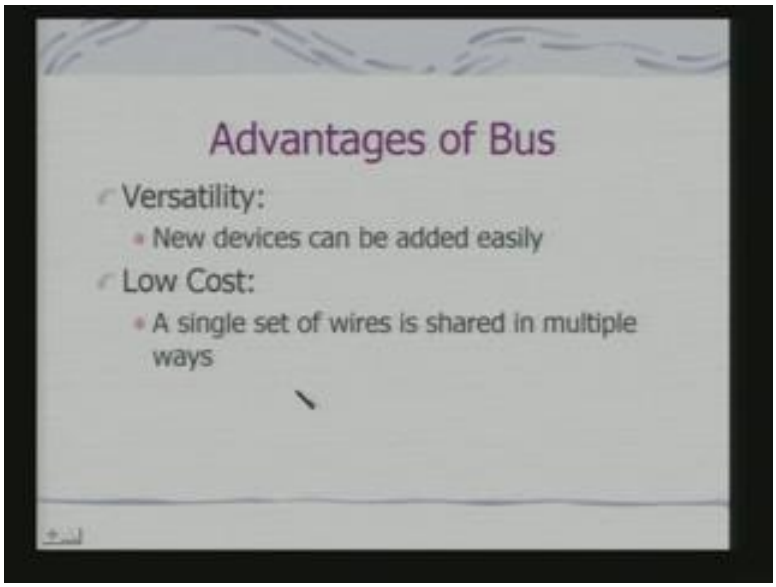


Now, obviously, I would like to increase the bus bandwidth the way I have shown if I am using time multiplexed mechanism for transferring data or address and data. The basic limitation that comes is with regard to the bus bandwidth. So, obviously, if I do a non multiplexed address and data lines I shall have increased bus bandwidth. So, you have got more bus lines and increase complexity in terms of the physical layout as well as the space requirement. The other option is increasing data bus width. In fact, this is done in many of the main frame systems. That means, you increase the width of data bus and transfers of multiple words therefore, will require few a bus cycles. So, consider as system is having its basic word length to be 16 bits that is sitting on the bus of say 64 bits. A bus which can support 64 bit data lines. And if you really doing a sequential access to the memory you can transfer 4 words in 1 bus cycle.

So, effectively increasing the bandwidth of the bus, but obviously, here the drawback is your using additional wire. The other option is block transfers. Allow the bus to transfer multiple words in back to back bus cycles. This is also using the same concept of using sequential address. That is I shall just provide a single address and then I shall do a number of this data transfer cycles. So, only a 1 address needs to be send at the beginning and the bus is not released until the last word is transferred. Obviously, the logically this has an increase complexity an decrease response time for request from whom? From the other devices which are willing to use the bus, but for a transfer you can have a large volume of data transfer and a fast data transfer. In fact, if you remember

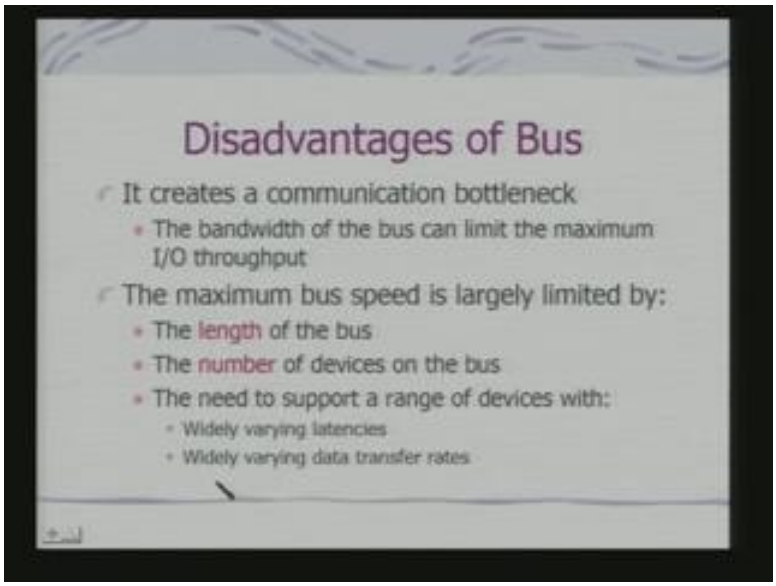
DRAMS an a said I had illustrated use of sequential signal from ARM for accessing the DRAM. That is you do a row select and then transfer the data in the row buffer for all column sequentially that can be very easily done through a block transfer mechanism.

(Refer Slide Time: 13:01)



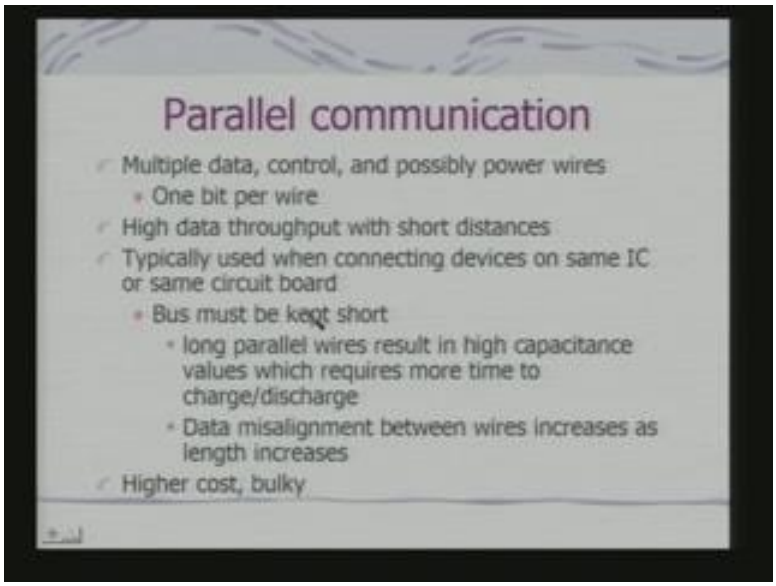
So, what are the advantages of bus? Since we are using a bus we have an agreement of the communication protocol. So, new devices can be added easily. It is not that each device having its peculiar communication protocol. If you have a 1 to 1 connection then each device can have it is own communication protocol and then it is implementation in terms of connecting into the processor will become more complex. It is a low cost, because a single set of wires in shared in multiple ways depending on the requirement of the device. What are the disadvantages? It obviously, creates the communication battled although we had started from the time multiplexed organization and talked about ways and means of increasing bus bandwidth.

(Refer Slide Time: 13:57)



But the bandwidth of the bus can actually limit the maximum IO throughput. And that can actually be the limitation on the processing capability not really processing capability, but output of the processor. So, if you may have a processor working at 2.4 Giga Hertz, but if it is not getting data at a rate more than 133 Mega Hertz then obviously, you is not using the processor to which full protection. So, bus can ((refer time 14:32)) that kind of a bottle neck. The maximum bus speed is largely limited by length of the bus and the number of devices on the bus. And there is a need to support a range of devices with widely varying latencies and widely varying data transfer rates. Because these are the inherent limitations of the devices, because I am looking at a bus if we look at the simplest picture of the model of the bus, where we having a single bus and the single bus I have memory and other devices hanging from it. So, in that case the latencies are different, because the devices would take different times to become ready to put the data on the bus and there would be widely varying data transfer rates as well. So, we shall see there are various ways by which you can try to minimize this kind of disadvantages.

(Refer Slide Time: 15:41)

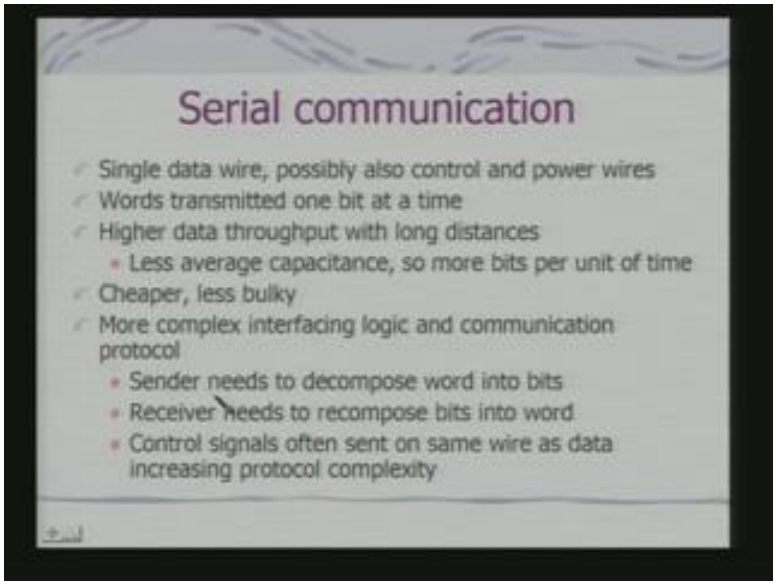


The other way look at the bus is so, far the key model or the basic model by which you have looked at is mode of a parallel communication. That is multiple data control and possibly power wires an effective data transfer unit is 1 bit per wire. So, this is useful for high data throughput with short distances and typically used when connecting devices on same IC or same circuit board. The motivation is this passes must be kept short, because long parallel wires will result in high capacitance values which requires more time to charge and discharge. That will limit the data transfer rate and data misalignment between wires increases as a length increases, because there is a capacity cost associated with each wire. So, higher cost means bulky. So, this is wired is a classical model where we have parallel lines defining the bus and you will find that particularly when we are talking about system or a general purpose computer system.

The bus on your mother board is typically the parallel bus, because these buses are exceptive to run for shorter length. But you will fine when we are looking at Embedded Systems is not always the devices are connected by this kind of parallel buses, because you can realize that if I have these parallel buses and so many signal lines then there are lots of problems which we need to face. A problem would be that of the space. Just consider that peak as a microcontroller which has got inbuilt peripherals and it has to supports some external peripheral as well. If it is supports a full fledge it parallel bus by which external peripherals can be connected then it has to provide a large number of

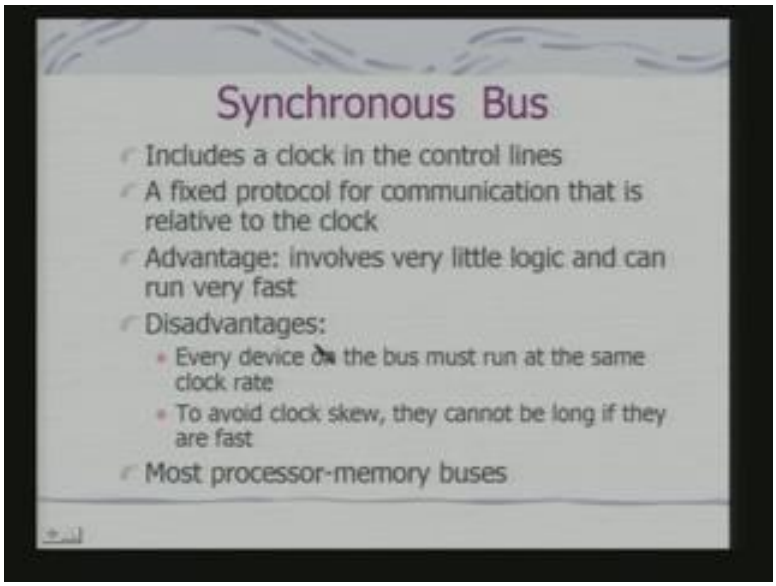
external beans. So, that is an area requirement even for the device itself putting so many connection lines can actually ask for additional physical area.

(Refer Slide Time: 18:04)



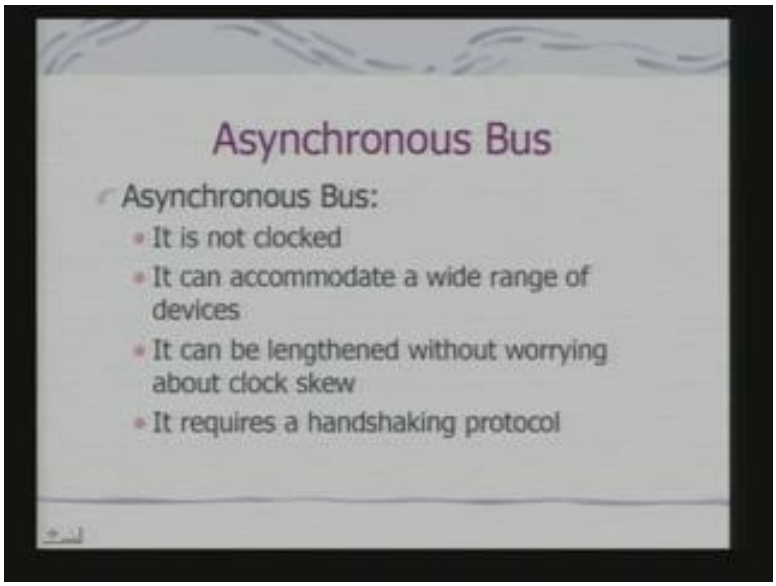
So, the other option, but we have is serial communication. Serial protocols and a very few lines for doing the actual communication you may have a single data wire possibly also a single control and power wires. Words are transmitted 1 bit at a time. So, effectively you have a higher data throughput with longer distances, because your capacity for load does not build up. Because since you have multiple parallel lines long parallel lines there will be mutual capacitance which actually reduces or puts constraint in the transfer rate. So, my bus is cheaper it is less bulky. You have got more complex interfacing logic and communication protocol. Sender needs to decompose word into bits for serial communication. Receiver needs to recompose bits into word. And control signals are often sent on the same wire as data. So, that increases protocol complexity, because you do not have separate data or control lines. So, you might like to use a same line to send the control information followed by data. And since send a needs to why send a needs to decompose what would so bits, because it has to send the words in a bit by bit fashion.

(Refer Slide Time: 19:51)



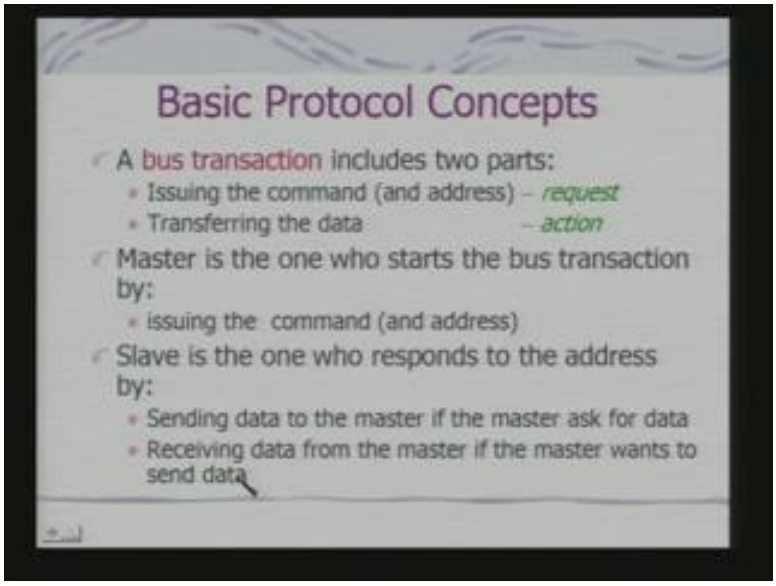
The other way of looking at bus is a whether a bus is a synchronous bus or an asynchronous bus. A synchronous bus includes a clock in control lines. So, you have a fixed protocol for communication that is relative to the clock that means, timing for your signals control signals are defined always relative to that of the clock. Since it is defined with respect to the clock you can understand that you logic can be simple and it can run very fast. A disadvantage; every device on the bus must run at the same clock rate means it should have the ability to deliver data the same clock rate. And to avoid clocks skew they cannot be long if they are fast, because a skew will buildup if the skew buildup then there would be timing errors for data transfer. So, you will find the most processor buses which are expected be fast buses in such cases you have synchronous bus. But otherwise when you are really using peripherals the buses are not strictly synchronous bus.

(Refer Slide Time: 21:11)



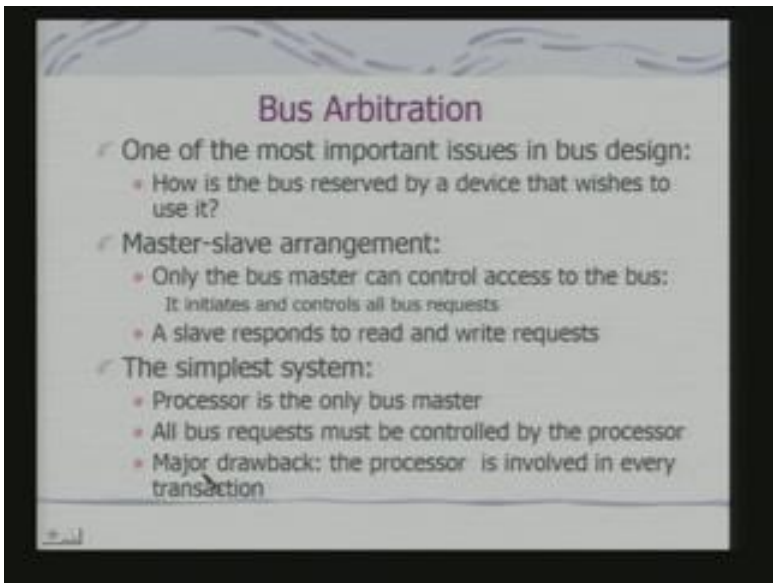
So, you got to have what you call asynchronous bus? Asynchronous bus is not strictly clocked. Since it is not clocked it can accommodate white range of devices they can work will different latencies. They need not respond with respect to the same clock. The bus can be lengthened that without worrying about clock skew, but it definitely requires what we call a handshaking protocol. So, that these devices can talked to the processor in a proper fashion ((refer time 21:46)).

(Refer Slide Time: 21:47)



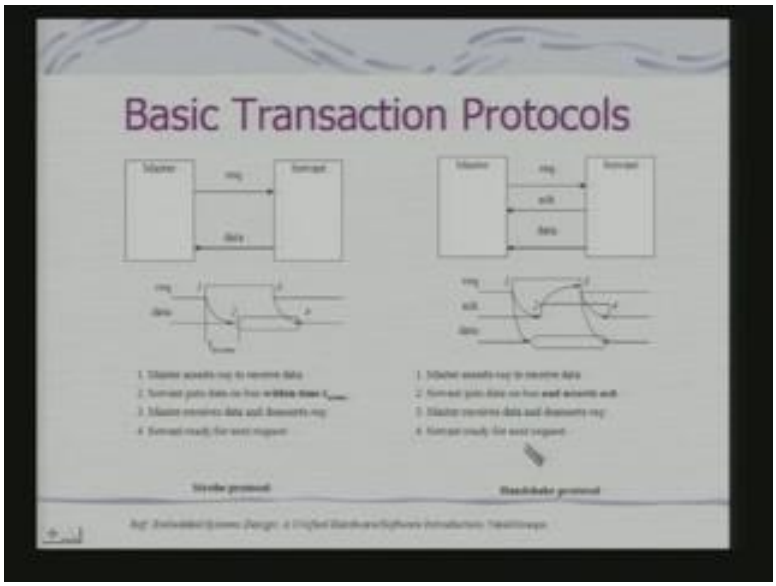
So, this takes as to the concept of protocols on the bus. So, what we say a protocol at the basic element of a protocol is bus transaction. A bus transaction includes two parts request an action. So, typically a request consists of a command and an address an action is basically transferring of data. Master in fact, we have already seen a master in a slave in a previous diagram in a time multiplexed bus organization. Master is a one who starts the bus transaction by issuing the command and address. Slave is the one responds to the address by sending data to the master if the master asks for data or receives data if the master wants to send data. But the device which a line on the bus they are not assigned they may not be assigned permanently to this master and slave role. A slave in one transaction can become master in some other transaction.

(Refer Slide Time: 23:15)



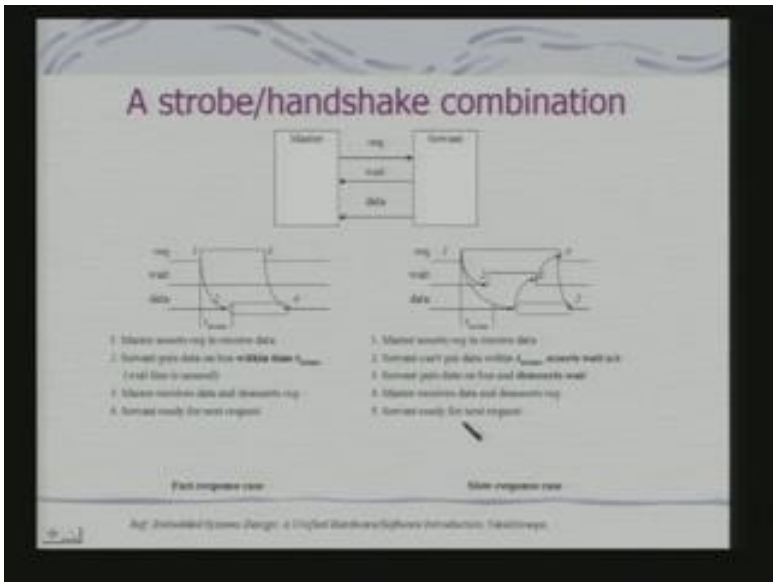
Next question come is that of Bus Arbitration. How the bus is reserved by a device that wishes to use it? If there is a single master there is a simplest system when we say processor is the only bus master then really arbitration problem is not there. But if the slave on the master can have interchange role then we really need an arbitration skin, but let us start looking at master slave arrangement. So, in case of a master slave arrangement only the bus master can control access to the bus. It initiates and controls all bus requires. As slave response to read and write request. And in the simplest system of this master slave arrangement I told you is the processor based system, but here the major drawback is the processor is involved in every transaction. You might not like to have processor involved in all that transactions.

(Refer Slide Time: 24:36)



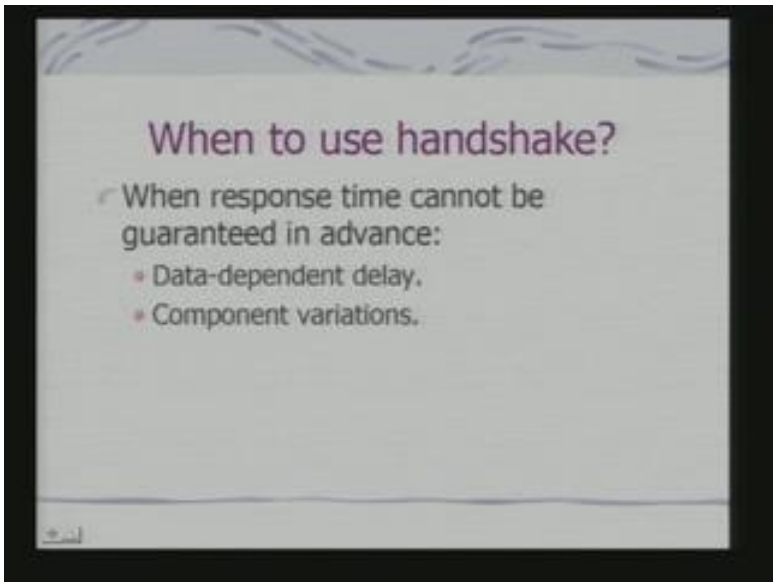
So, let us look at the basic transactions protocol. This master and this is a servant this is called astro protocol. In fact, what is interesting to note is that this is what we were looking at when we looking at the multiplexed bus. So, there is a request line and your data transfer is synchronizing with respect to your request. So, what we say the basic protocol ((refer time 25:02)) master asserts request to receive data. Servant puts data on bus within abstain T access master receives data and then D request servants becomes ready for next request. So, these when will work once these devices can guaranty the response within a bounded T access. The other one is what is calling a handshaking protocol where you have request acknowledge as a kind of a peered signal. So, master hazards request to receive data, servant's puts data on bus on sends on hazards acknowledge. So, master receives data and D hazards request servant becomes ready for the next request. So, you have what we a telling is the master hazards request to receive the data the servants puts the data and also tills the master that it has put the data. It says the servant puts data on bus a hazards act. So, when the master gets act it knows the data is available. So, master receives data and then it wills D hazards request and then only the servant becomes ready for the next request. So, obviously, you can understand this is a purely asynchronous protocol. So, they can be a strobe or handshake combination.

(Refer Slide Time: 26:59)



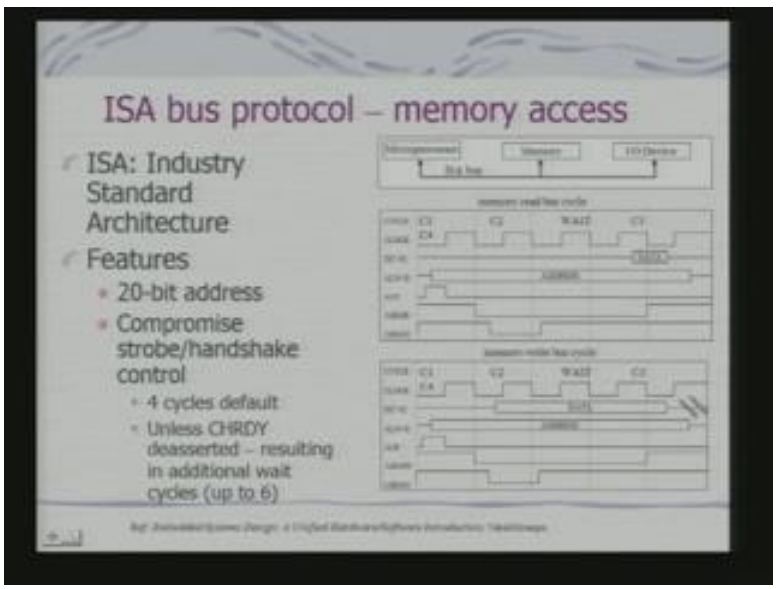
So, this is a combination what we say the first response case where we have introduced an additional signal line called wait. In fact, we have talked about wait when we discussed the ARM interfacing with the memory that ARM need to wait for RAM access, because it is slow. So, that is exactly what is being handle by this wait signal the same thing is being now, handle and we are talking this not in the context of just a particular chip or a particular processor we are talking about it in the context of a common signal line which is connecting processor and different devices as well. So, for a first response case there is no need for the wait line to become active, because this is a typical strobe access, because I am guaranty to provide the data within a T access time. If I am not guaranty to provide the data what is important is wait. Now, wait signal comes from the servant wait ask the processor to wait. So, what we say that the master resorts request to receive data? Servant cannot put data within T access. So, it hazards wait acknowledge. So, it is almost like an acknowledge signal servants puts data on bus and D hazards wait master receives data and DA hazards request so, the servant becomes ready for makes request. So, this is a kind of a strobe handshake combination.

(Refer Slide Time: 28:48)



So, the question is obviously, when to use handshake? When response time; obviously, cannot be guaranteed in advance. There would be Data dependent delay and there would be component variations. And in fact, this is what we had already seen that SRAM can put the data within the T access corresponding to a single cycle requirement for ARM, but this cannot is not true for ROM. So, there has to be wait cycles insert it. In fact, we shall look at these whole protocols in the context of a bus ISA bus protocol. In fact, ISA is actually a bus standard which came in the context stuff PC. It is an industry standard architecture.

(Refer Slide Time: 29:38)

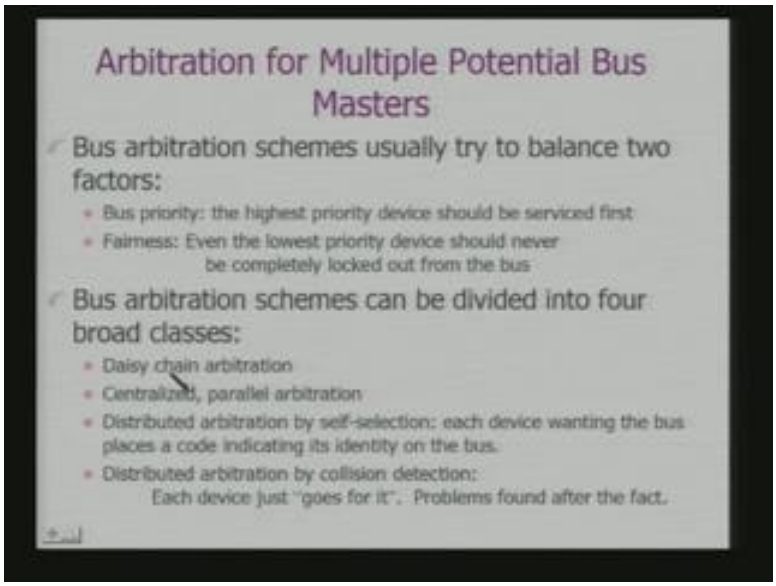


Now, one point you have to understand is all these bus protocols and all these things have to be standardized and standardize via an agreement in the industry. Because there will be individual developers of the device and memory as well as the processor. If they cannot agree on a same bus protocol or the same bus definition their devices will not be useable across multiple platforms. So, this is just an example, because a bus specification is much more complex this is just a simple example of a memory access. So, since it was developed originally keeping in mind 8086 kind of processor. So, it has got a 8 20 bit address. It uses what we called compromise strobe handshake control that is wait state base control. It has 4 cycles by default unless channel ready is deasserted, it is resulting in additional wait cycles. It can there would be upto 6 wait cycles can be inserted. So, effectively what we are seeing here is memory read bus cycle so, this is the basic clock.

Now, this read it is not a single clock read it is a 1 cycles by default is allocated for each operation. And since is a read bus cycle the address is put in it may be it actually has got a multiplexed address bus. So, that is why there is an ally signal address latch enable, because if you are using a multiplexed bus what happens? You have first put in the address, but address should remain valid when you are either reading or writing the data. So, this address has to be latched the input of the memory device. So, you have a control signal address latched enable. Then you have this memory read this is telling the over the bus that you are actually executing a memory read cycle. Now, since the device is not ready so, the channel ready is like a wait signal which is going low. So, it is deasserted resulting in an additional wet state.

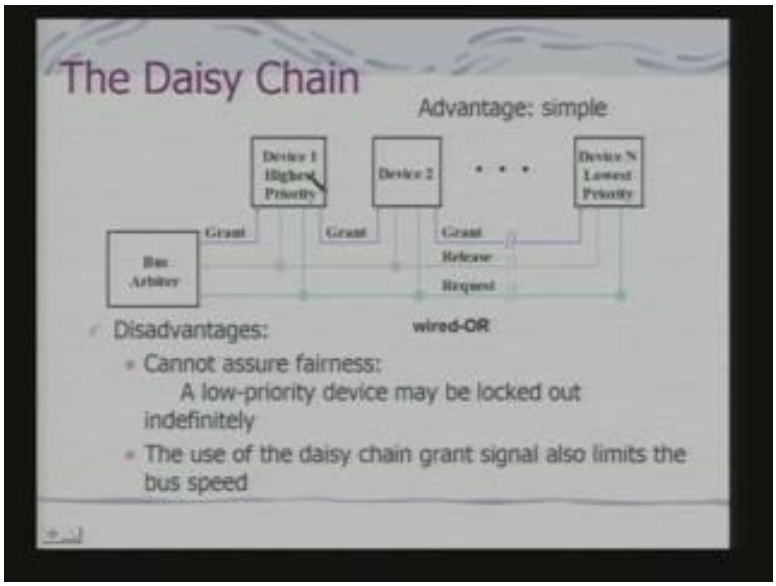
And then when it goes up the next cycle in the clock the data becomes available. In fact, I have not shown, but the duration the standard specifies exactly the timing parameters with respect to each of these transitions. And may it says there can be up to 6 wait cycles it implies the devices which are slower than these I am not expected to be connected by ISA bus. Similarly, I have got a memory write bus cycle. Here, the control signal is memory write bar otherwise, everything is identical. In fact, you have a channel ready this a assorted so, following this you have got a wait clock cycle and then you actually do the operation it is a subsequent cycles. ISA bus although we have it developed in the context of PC we shall see later on that a variant of it is used to in Embedded Systems as well. Next we shall look at arbitration for multiple potential bus masters.

(Refer Slide Time: 33:16)



So, far we were happy to have one master one or more slaves. Now, they can be multiple bus masters. So, burst arbitration schemes usually try to balance two factors what you call bus priority because the devices would be now, associated with priority. So, the high's priority device should be serviced first and they should be a fairness Even the lowest priority device should never be completely locked out from the bus. Bus arbitration schemes can be divided into four broad classes. Daisy chain arbitration, Centralized or parallel arbitration, Distributed arbitration by self selection each device wanting the bus places a code indicating its identity on the bus. And distributed arbitration by collision detections each device just goes for it and problems are found of that after they have started the process. So, this is exactly what your networking protocol of CS and SED. We shall primarily look at these two in today's discussion.

(Refer Slide Time: 34:38)

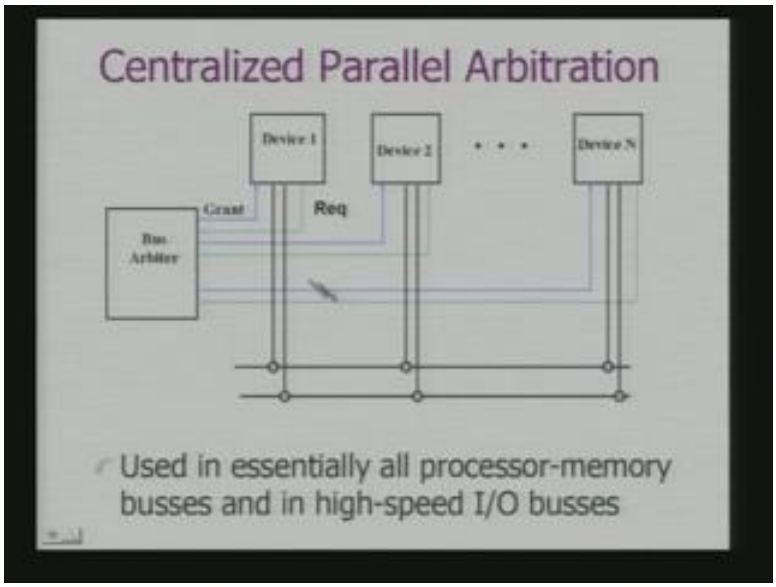


First is daisy chain. Daisy chain is there is a bus arbiter this is not necessarily the processor, but it is a separating processing element which is their along with the bus. These are the devices and all these devices are can be potentially bus master. Now, what happens is when this device owned the bus they will generate a request. Now, these request lines are connected via wired OR. You will find that this is a request line and these request lines are connected via wired OR. And bus arbiter depending on its availability of the bus cycles it will generate the grant signal. Now, these grant signal is not connected to all of them in parallel, but it ripples through just like a ripple counter it ripples through. So, the ground signal goes to first the device which is nearest to the Bus Arbiter. Now, if it has generated the request what it will do? It will not allow the ground signal to ripple through.

So, it will now, make use of the bus once it is done with it will generate the really signal to tell bus arbiter that my job is over. So, obviously, the priority is defined according to the order in which these devices are put on the bus. Now, this is also used for servicing multiple device interrupts when all of them are connected on the same interrupt line and connected to the processor under those conditions also daisy chaining is used. The advantage is these arbitration scheme is very simple straight forward. It is based on the physical location of the device with respect to the arbiter, but it cannot assure fairness. A low priority device may be locked out indefinitely. And the use of daisy chain grant signal also limits the bus speed, because this grant signal has to ripple through. So, it is

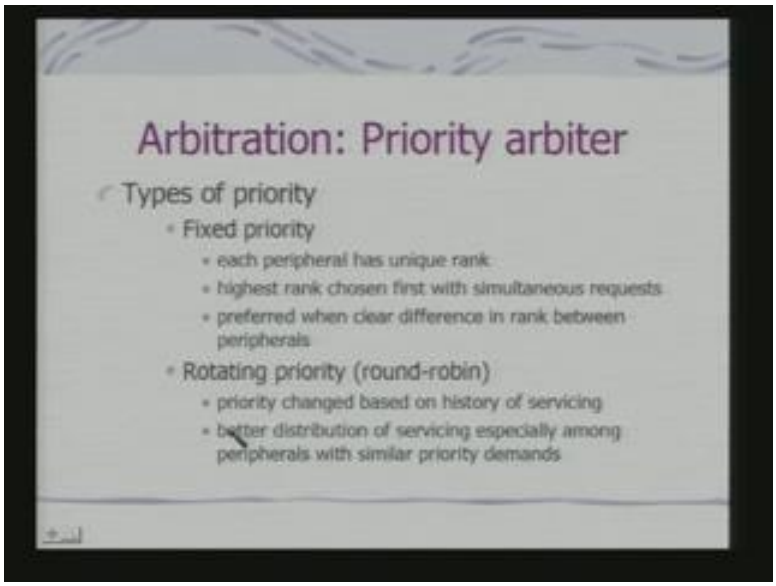
not available in parallel to all of them. So, each of this device will introduce a delay and that will be the limiting factor on the bus speed. And the other side I can have a centralized parallel arbitration.

(Refer Slide Time: 37:10)



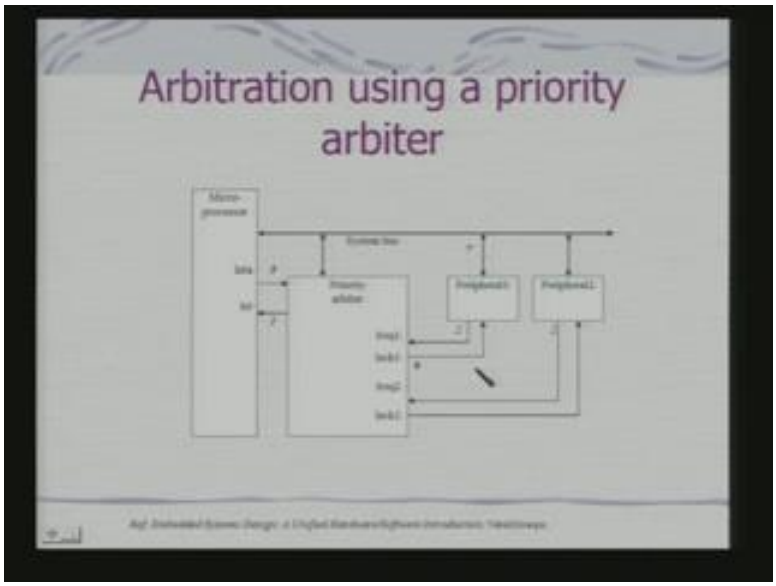
The basic difference here is each device has got its own control lines. The when it request and it gets grant so, obviously, when a grant is generated this is generated in a device specific fashion. And it is used obviously, you can see this can be a very high speed bus obviously, I need to have complex bus arbitration logic, but it can be very high speed and used essentially all processor memory buses and high speed IO buses. And the drawback of these should be a number of signal lines and I would require a complex logic.

(Refer Slide Time: 37:55)



Now, how to deal with priority? This is a basic issue, because you are talking about priority. So, there are two kinds of priority I can have a fixed priority. So, that each peripheral has a unique rank the highest rank chosen first when there are simultaneous request. This is particularly true when we are looking at this kind of parallel arbitration scheme. It is preferred when there is a clear difference in rank between peripherals. The other option is rotating priority. Priority changed based on history of servicing that currently which is being service joints at the end of the queue and that is the basic principle of round robin priority. This has got data distribution of servicing especially among peripherals with similar priority demands. So, ties and not getting resolved arbitrarily tier gets getting resolved in a rotating fashion.

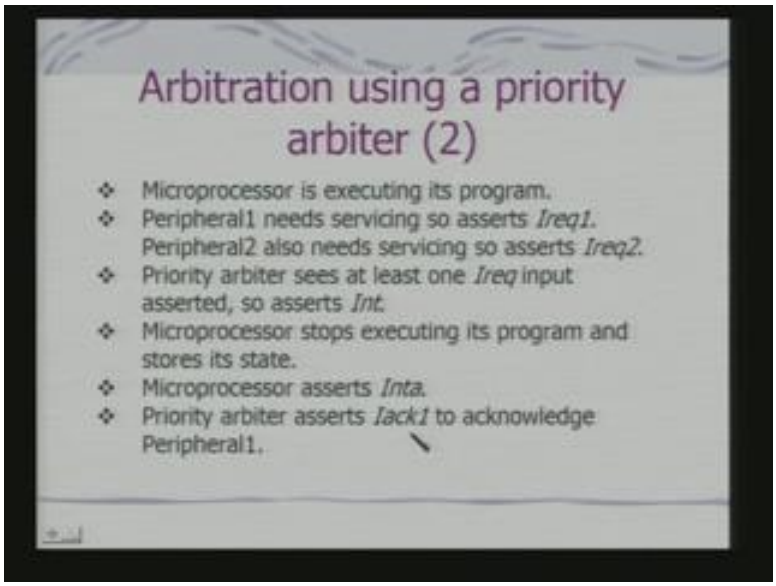
(Refer Slide Time: 38:59)



So, the basic structure wise will have something like this we are just showing an example of priority arbitration scheme for dealing with interrupts. So, this is a processor and these peripherals so, this is these peripherals should like the access of the bus and when they ready the generate this interrupt request. In fact, these kinds of priority arbiter are components of all these interrupt controllers that you have with variety of microprocessors and microcontrollers that we use. So, these peripherals would generate interrupt request. Now, these priority schemes are not that of the interrupt priority scheme of the processor. This is a priority scheme which is signed by the, to the peripherals by the priority arbiter and the way you program the priority arbiter.

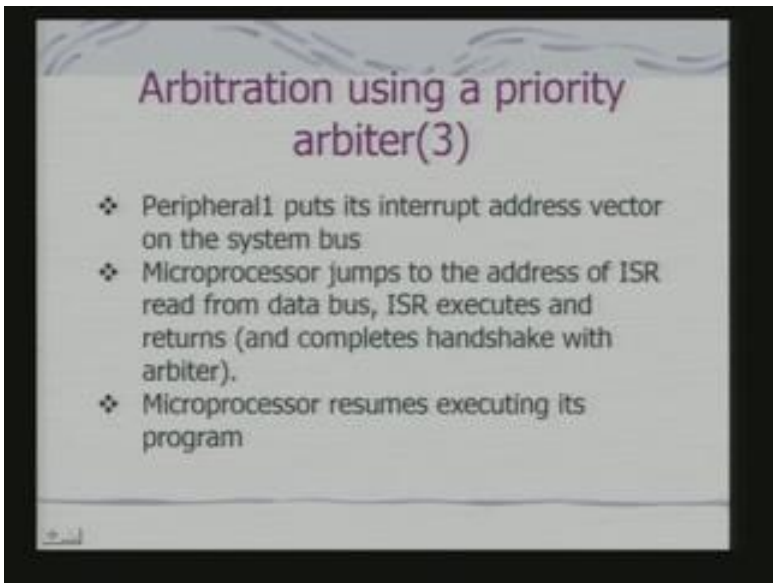
So, depending on the priority of the interrupts the priority arbiter will generate the interrupt. So, what we have shown here is if you look at there I have marked it as number 2, because this is a peripheral controller. So, there would be an actually a request coming from the actual device to the peripheral for during the servicing. So, the peripheral generates the interrupt request the priority arbiter not rises the interrupt. So, appropriately if there processor is ready it will generate the acknowledge signal when the acknowledges are available the priority arbiter would generate the acknowledge signal to the peripheral. It will generate only to that peripheral which is currently schedule to receive service according to the priority. And that will lead to the actual system bus transfer.

(Refer Slide Time: 40:50)



So, the steps involve is microprocessor is executing its program, peripheral needs servicing asserts this, request peripheral 2 also can assert it request. The priority arbiter sees at least 1 request inputs so, asserts interrupt. Microprocessor stops executing its program and stores its state. The microprocessor asserts interrupt technology and priority arbiter asserts Iack1 to acknowledge the peripheral1.

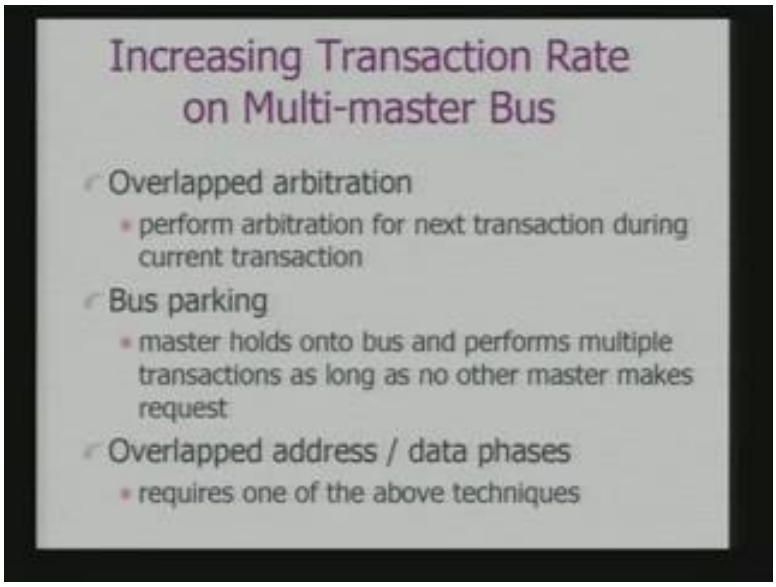
(Refer Slide Time: 41:23)



Now, Peripherals 1 puts its interrupt vector address on the system bus, because the processor needs to know which peripheral it is going to service. So, peripheral now puts

the address of the corresponding ISR. The microprocessor jumps to the address of ISR read from data bus and ISR executes and returns microprocessor resume executing its program. In fact, this is the scheme. In fact, if you remember when we talked about various levels of interrupts we discussed this in the context of peak as well. So, when there a various levels of interrupts this is how the interrupt arbitration is done with respect to the devices.

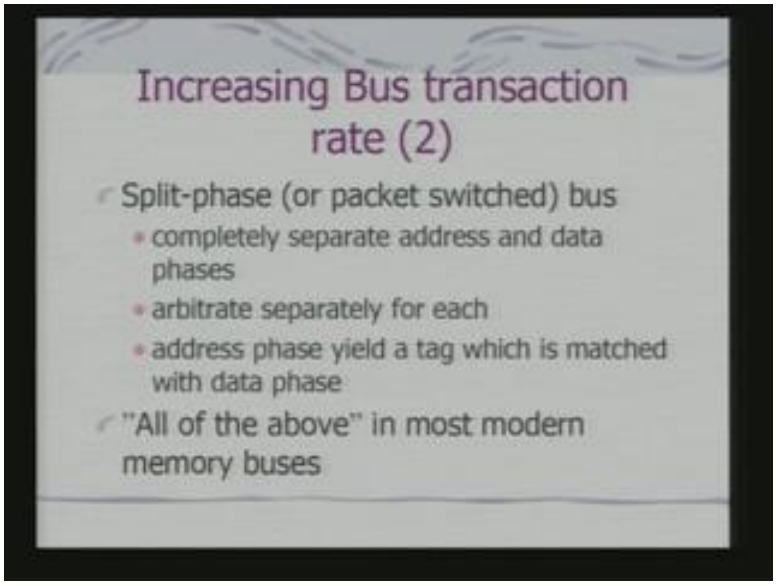
(Refer Slide Time: 42:15)



Now, how to can you increase the transaction rate on multi-master bus? Because you have seen you always when there is a multi-master Bus the basic problem is that you actually pay for your protocol to dissolve who is the current requesting device who will become the master. So, the one we have doing it is by exploiting the basic principle of pipeline, but in a different context what you say that there can be arbitration for next transactions during current transaction. Because in the current transaction actually we have transferring the data, but we can actually finish of arbitration beforehand, because when the arbitration is taking place the signal lines which are involved may not be strictly involved during the data transfer. Next thing what is called Bus parking? Master holds onto bus and performs multiple transactions as long as no other master makes request that means, there is no for each transaction there is no priority resolution. Priority resolution only takes place on remark.

Obviously, that would require more complex logic for priority arbitration It can understand this is something like this that if my priority arbitration block or the logic it is something like a processor. So, that processor will be interrupted when there is an request from another device then only it will execute the priority arbitrations script. If there is no such request then the current master will continue to remain master for next cycles. And next transaction will takes place without arbitration and hence there would be faster data transfer. There is overlap address and data phases, because you providing the address and data. So, these both these things since you are using what we are using non multiplex lines if you are using non multiplex lines address and data phases can actually be overlap. There is other interesting thing which is something like a split phased bus.

(Refer Slide Time: 44:46)

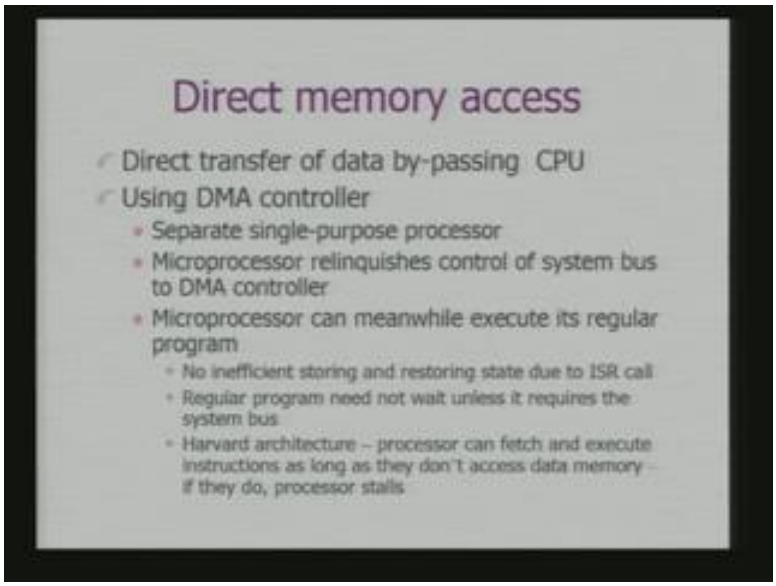


This is a most sophisticated bus protocols which have which have come in. So, in a packet switch it is more like a packet switch network and when we talk about and interconnection framework between numbers of devices. This kind of a packet switching can takes place. So, you have completely separate address and data phases we are not talking about only buses, but we are talking about completely separate address and data phase and they arbitrate separately for each. So, address phase yields a tag which is matched with data phase. So, what is at mean? It something like this I send first an address very simplistic picture would be I would first send an address on the basis of the

address at device get selected. So, device provides a tag when the data is being sent the data contains the tag. So, tag is matched and data is received.

So, you can have these data transfer with regard to each of these requirements. And see all of these above in most modern memory buses, because there actually the bottleneck of speed really comes to end. The in these context why you should understand these basic packet switching is why it is interesting and why you should note in these context is. So, far whatever we have discussed in terms of bus transaction protocol its very similar to a kind of a circuit switching scenario, because your lines are getting dedicated to a particular device for a set of transactions. That set make include a single transaction as well, but when we viewing a packet switching is not like that. So, your number of data packets is actually flowing on the bus. Depending on the tag which is available with the data the devices will accept the data. So, truly speaking it is not that is for communication between CPU to a device. A device 1 to device 2 the circuit is not getting completely allocated or dedicated for this transfer.

(Refer Slide Time: 47:27)

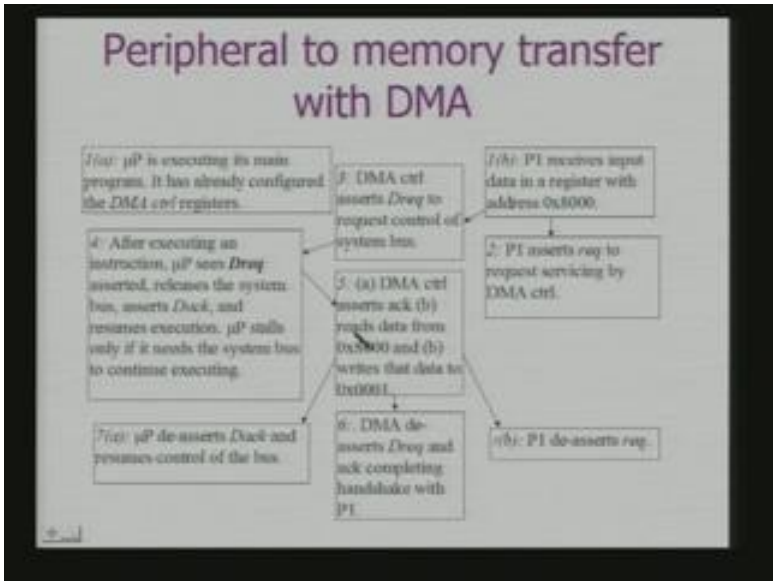


Let us look at direct memory transfer, because this is the most common form of having multiple bus master. That is when we use a DMA controller, because the DMA is what we are bypassing the CPU to transfer the data from peripheral to the memory. And you are using a DMA controller and DMA controller is strictly speaking as single purpose processor which can take over the control of the bus. So, in the very common and the

low in systems DMA controller would be the most important other device to become bus master. So, microprocessor relinquishes control of the system bus to DMA controller and while DMA controller is carrying out the data transfer microprocessor can execute its regular program. So, what we have is no inefficient and storing and restoring of state due to ISR call. You should keep that in mind because these always the ISR has got associated with this the latency and storage in the memory.

And time which goes in for this kind of state storage and retrieval that interrupt latency overhead that gets completely eliminated if I use DMA for this kind of data transfer. Further, regular program need not wait unless it requires a system bus, because if it is a pipeline processor in if it as already fetches the instructions. It can continue execution of its instructions while there is a data transfer on the external bus between memory and peripheral. And this is so, it is very interesting in the context of a hardware architecture, because processor can fetch and execute instructions. Because its instruction memory is different from data a memory and when you are transferring a data from peripheral you will be transferring the data to data memory and not to the instruction memory. In fact, this is a another reason why you have found that to this microcontrollers which are targeted from embedded systems pickup Hardware architecture. Because you can have parallel data transfer from slow peripherals while execution of the code can really continue by fetching instruction from instruction memory.

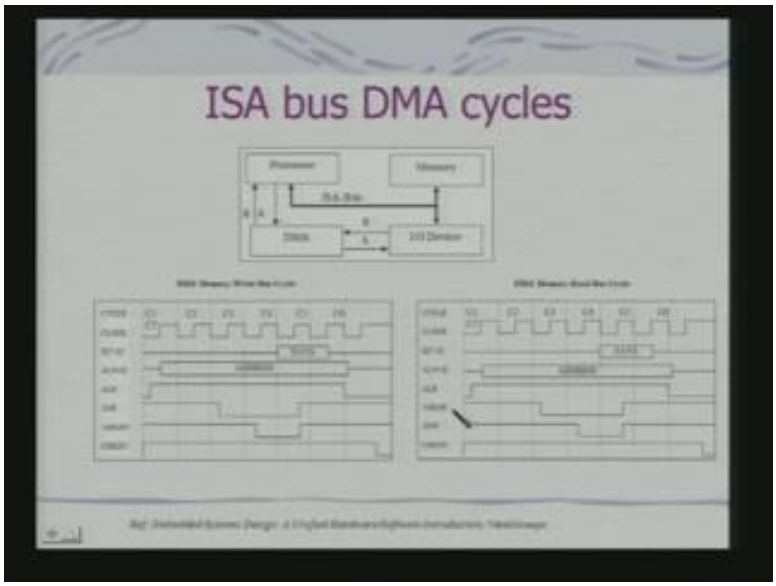
(Refer Slide Time: 49:46)



So, how does this the whole process what is the protocol for these kind of a DMA transfer? So, if P1 is a peripheral it receives input data in register with may be an address 8000. So, this is and these DMA control with data in register so, now, that data has to be transfer this a simple example. So, DMA controller what it does? It asserts DMA request for the control of the system bus. If I go through the timing sequence if I look at here the Microprocessor has executing its own program after configuring the DMA control registers what is the DMA control registers need to know? Need to know definitely the target address where the data has to be stored. They may controller also needs to know from which peripheral or which peripheral ports this data may be ray and put into memory. So, the first you actually configure the DMA controller and then DMA controller can receive a request from actual peripheral device. So, the step number 2 is the peripheral device asserts request to request servicing by the DMA controller this is a step number 2. The step number 3 would is that after receiving this requests from peripheral, because peripheral is not requesting now, not the processor, but DMA controller for the transfer.

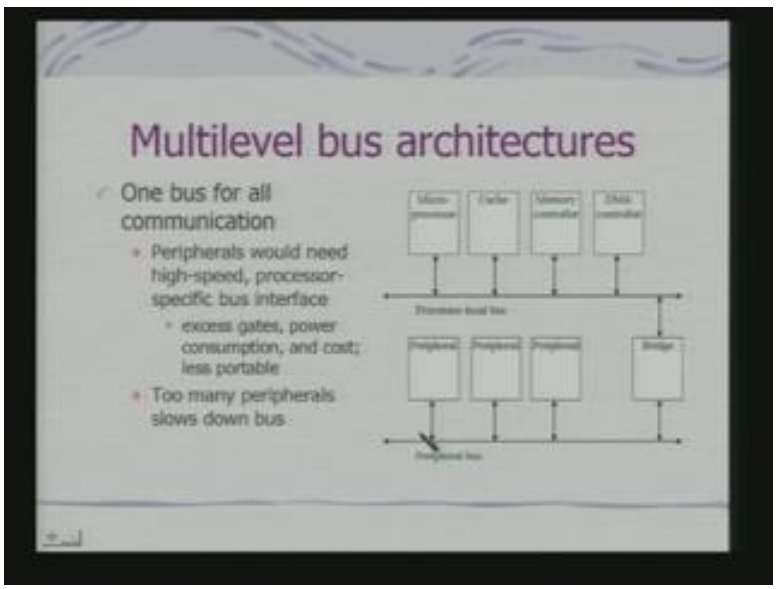
So, this comes to DMA controller DMA controller now, sends request to the processor. The processor releases the system bus whenever it can in fact, it says that here we are looking at an example after executing an instruction in many cases processors can relinquish bus even while executing an instruction. So, you need not wait till the end of the instruction. It asserts the DMA acknowledge in fact, tag when it asserts a DMA acknowledge the DMA controller asserts acknowledge reads data and writes a data to a memory location. So, these could be a memory location where it is to be written. Then the 6 step is DMA de-asserts D request and acknowledge completing handshaking with the PI. So, because it has already it has to tell already the peripheral that your job is done. I have done your job and these when it de-asserts tag. So, processor knows DMA controller no longer needs a bus. So, it can reclaim the bus. So, 7 A is told this to the DMA controller and 7 B is that peripheral de-asserts a request because the request now is its job is done. So, it should no long a request a DMA controller let to provide the service. So, you can see that the whole DMA transfer it is coordination from peripheral to CPU via DMA controller and while actual transfer takes place DMA controller becomes a master. So, it generates who generates memory address? DMA controller generated memory address when actually the DMA transfer takes place.

(Refer Slide Time: 53:10)



So, if I look at the DMA bus cycle this is for ISA DMA bus cycle you will find that both control signals. The other part of it is same that your address and the data part are same. The address is now getting generated by whom? It is getting generated by the DMA controller, but interestingly you will find that you are asserting both IO read and memory write signals why? Because you have to read data from the peripheral so, you are asserting IO read and that data has to be rate to the memory. So, you are asserting memory write. The other corresponding thing should happen when it is a memory read bus cycle.

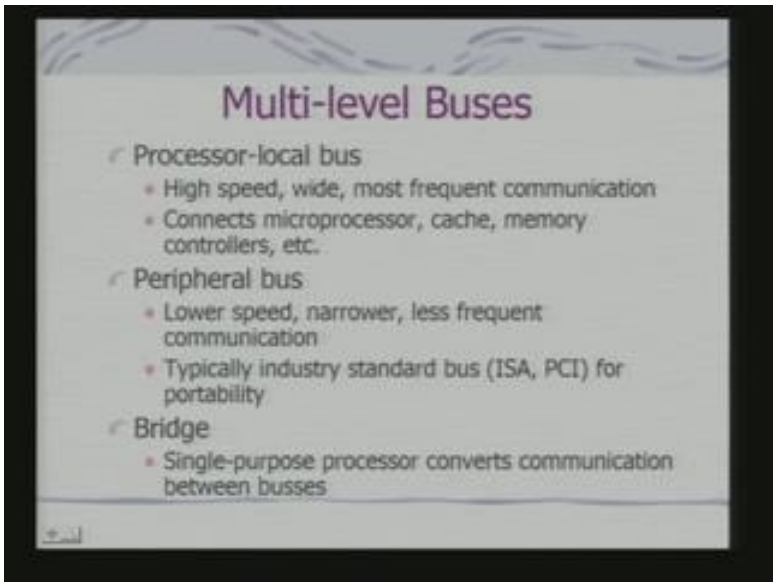
(Refer Slide Time: 53:54)



Now, I had already talked about the disadvantages of the bus if there a various devices with various kind of latencies. So, one solution to this problem is what is called multilevel bus architectures. Because if you have one bus for all communication in fact, this is the basic disadvantage I talked about earlier. Peripherals would need high speed processors specific bus interface which is not possible always to provide, because then the cost of the peripheral devices would increase that means, it would lead to access gates power consumption and cost unless portable.

So, these are these you would not like to have. In fact, this is more true in the context of Embedded Systems. And so many peripherals putting on the same bus slows down the bus. So, 1 solution is you have hierarchal bus structure what we call a processor local bus and a peripheral bus? So, processor local bus is where your microprocessor that is your CPU your cache memory, memory controller DMA controller all these things really set. And on the peripheral bus you have the peripherals and the 2 buses communicate via what you call rich.

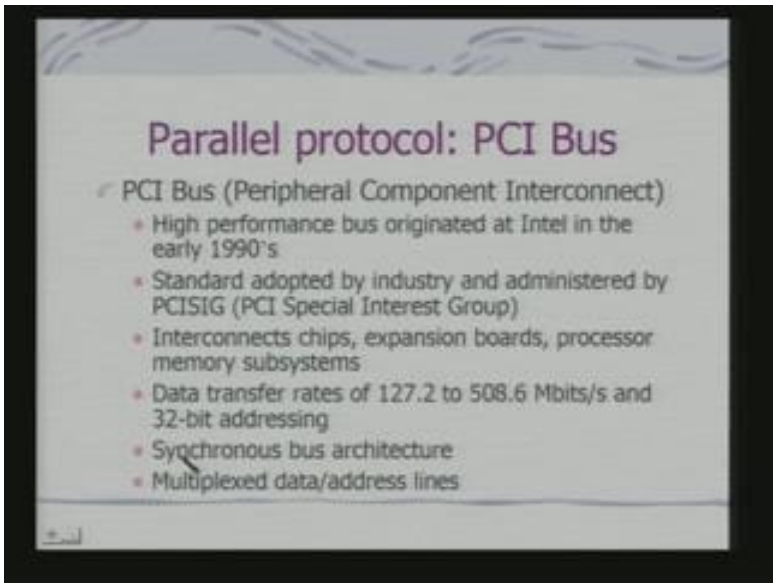
(Refer Slide Time: 55:12)



So, you have got processor local bus which is high speed most frequent communication on this bus which connects this microprocessor cache and memory. And you have a bridge which is the single purpose processor which converts communication between the buses In fact, in many a bus standards it will actually accumulate on number of transactions to get the from the slower bus and then post it to the high speed bus. The

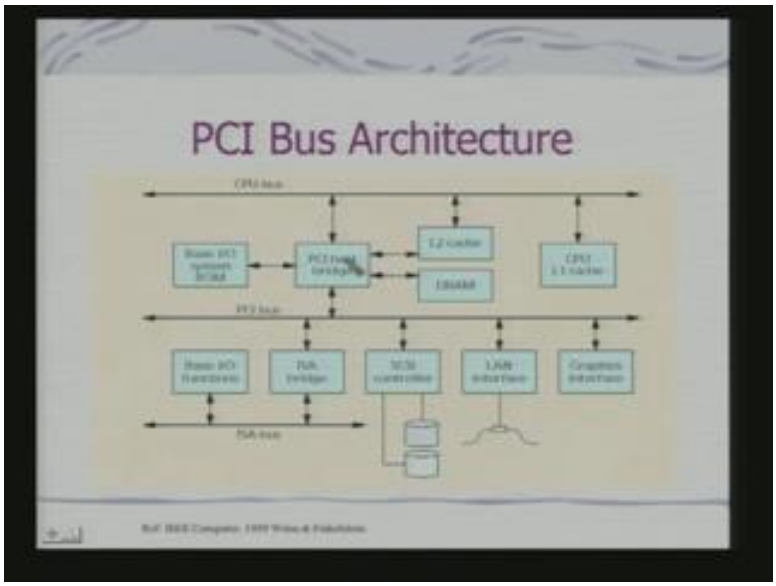
similar thing it will do it from high speed to the low speed bus. Peripheral bus is what is the lowest speed and typically you follow some industry standards for the purpose of port ability.

(Refer Slide Time: 55:54)



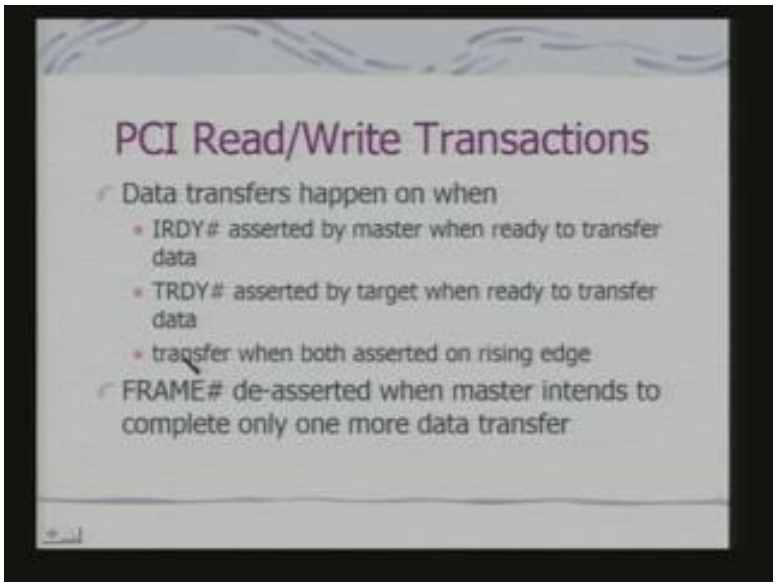
The most well known on the common hierarchal bus structure today, its PCI bus Peripheral Component Interconnect bus it is a pretty bus, because it can have a data transfer rates of these and as a 32 bit addressing. Because you are using Pentium as a processor it uses synchronous bus architecture, but still it has got multiplexed data or address lines.

(Refer Slide Time: 56:26)



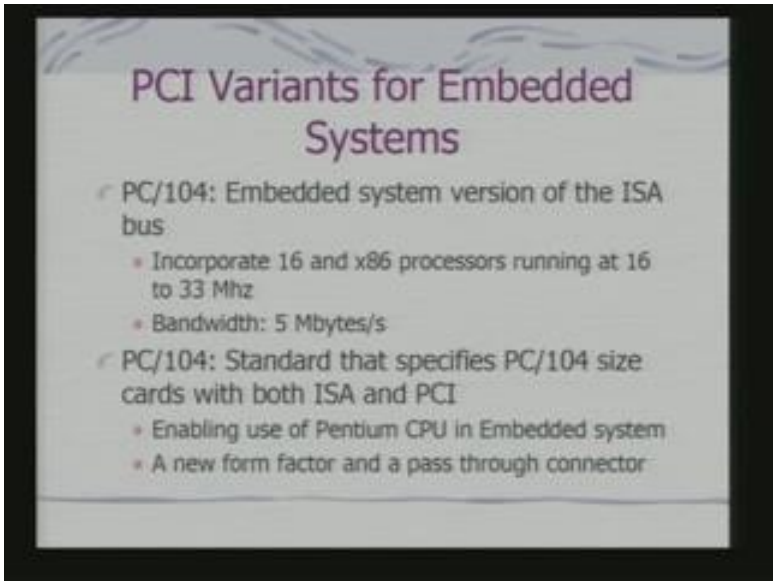
What is a basic protocol of that bus? Basic protocol is based on the architecture. In the architecture you have got the basic CPU bus which is high speed bus you have got PCI bus as well as you have got ISA bus which we are talking so, far for connecting the low speed device. And between each one of them you have got the corresponding bridge. This is the host bridge between CPU bus which is the high speed and that of the PCI bus the basic bus and that of the ISA bridge for bridging between PCI and ISA bus. Obviously, L2 cache is sit on the CPU bus, but for trying to write onto the DRAM it will go via the host bridge because DRAM is a slower device. So, you L2 and L1 cache are on the bus CPU bus, but your ROM your DRAM are connected via PCI host bridge, because they are slower device.

(Refer Slide Time: 57:23)



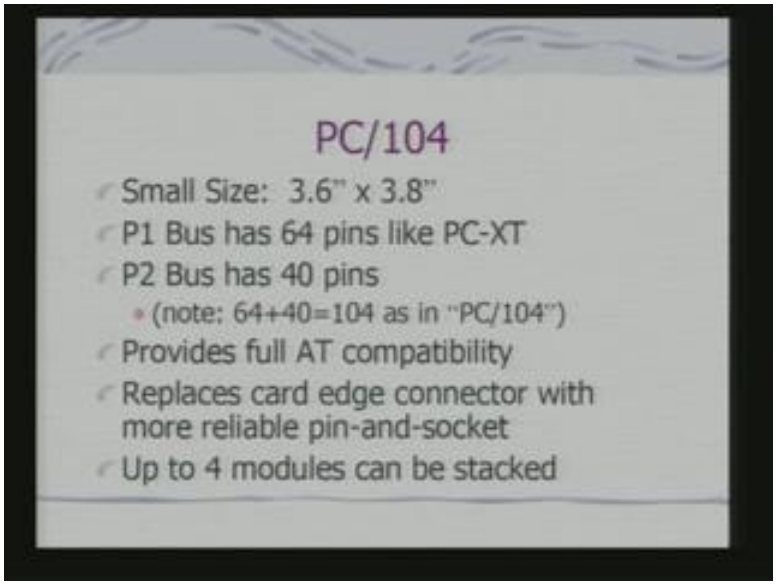
So, what we say the whole signals in the PCI bus as sampled on rising edge and follows a centralized parallel arbitration. And all transfers and nothing but bursts an address phase starts by asserting frame. Frame is a basic signal and next cycle initiate initiated asserts common an address. That means, it something like a complete cycle transfers. So, frame the address phase starts by the frame and then you have the common and as on the address for the next frame. And the transfer is through bus and you follow as centralized parallel arbitration scheme that we had already discussed. Data transfers are done with regard to 2 signals IRDY and TRDY when this is asserted when the master is ready to transfer. This is asserted when the target is ready to transfer and transfer when both asserted on rising edge. I hope you can understand that the master is ready to transfer and the device has to agree then only you can have the transfer. The frame has it de-asserted when master intends to complete only one more data transfer.

(Refer Slide Time: 58:41)



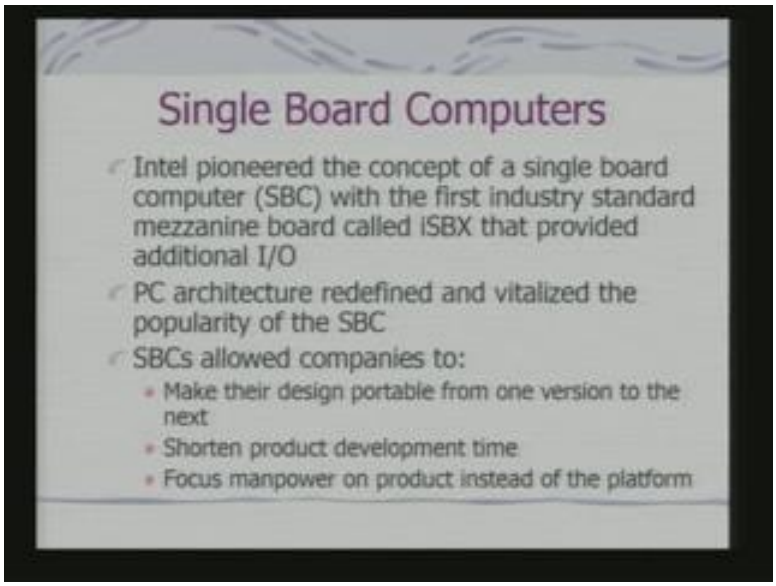
The PCI variants for Embedded Systems are PC 104 which is actually ISA bus and PC 104 plus which is actually both ISA and PCI.

(Refer Slide Time: 58:55)



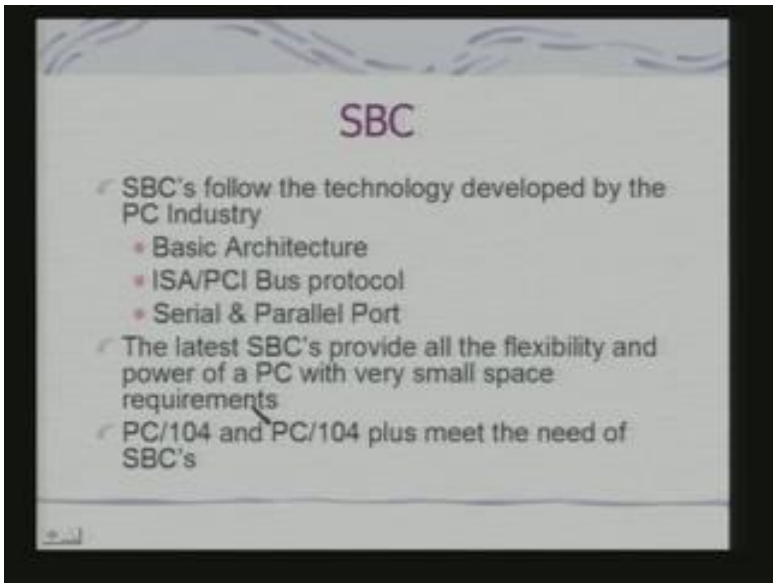
In fact, PC104 the basic issue is that for all these buses ISA and PCI bus they have same identical signal definitions and the protocols when we look at in the context of embedded systems. The only difference is the size becomes small the food print become small. I told you its critical to note that size the mechanical parameters becomes important for the bus. This is an example where the mechanical parameter's had been taken care of the purpose of embedded applications. In fact, this is used for what are called single board computers.

(Refer Slide Time: 59:33)



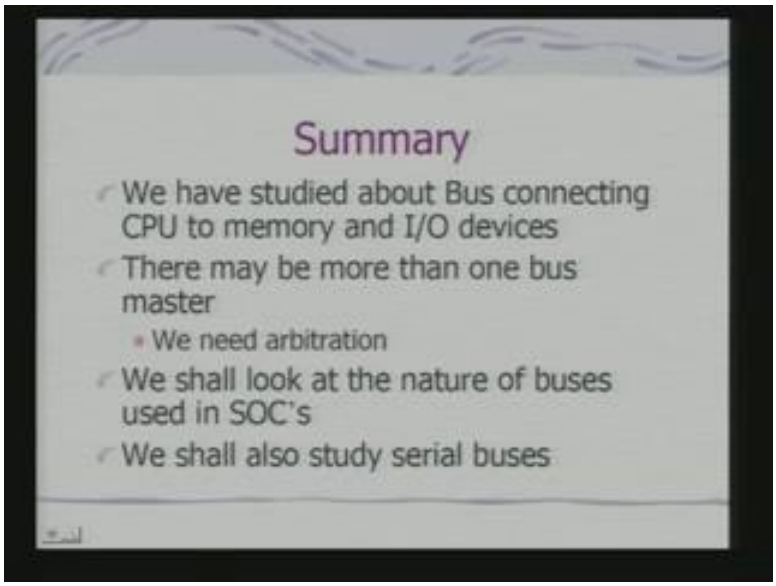
This was pioneered by Intel and to make use of your Pentium processor for embedded applications. And they reduce define the bus which can be fitted into a small mechanical food print. So, that it can be very easily used for various embedded applications.

(Refer Slide Time: 59:58)



So, the whole idea is that all flexibility and part of a PC with very small space requirements that was a motivation. So, you can have a real really a software development without being bothered about knowing new hardware.

(Refer Slide Time: 01:00:14)



So, this is what we have learn today, the basically the bus how it connects CPU and IO devices. There are may be more than 1 bus master and we need arbitration in that case. And we have looked that single board computers via we got a PC bus moved into an embedded system application. The next level is system on chip where you have got multiple processor another thing putting into one single silicon then how the bus would look like in the context. The other thing which we have not looked at so, far is the serial buses for connecting peripherals to this embedded microcontrollers this we shall look at next few classes.