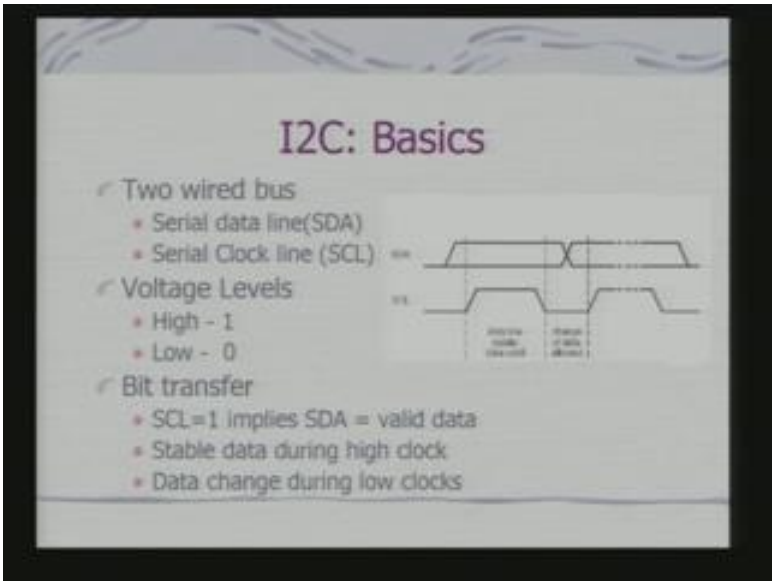
Embedded Systems
Dr. Santanu Chaudhury
Department of Electrical Engineering
Indian Institute of Technology, Delhi

Lecture - 16
Bus Structure – 3
Serial Interfaces

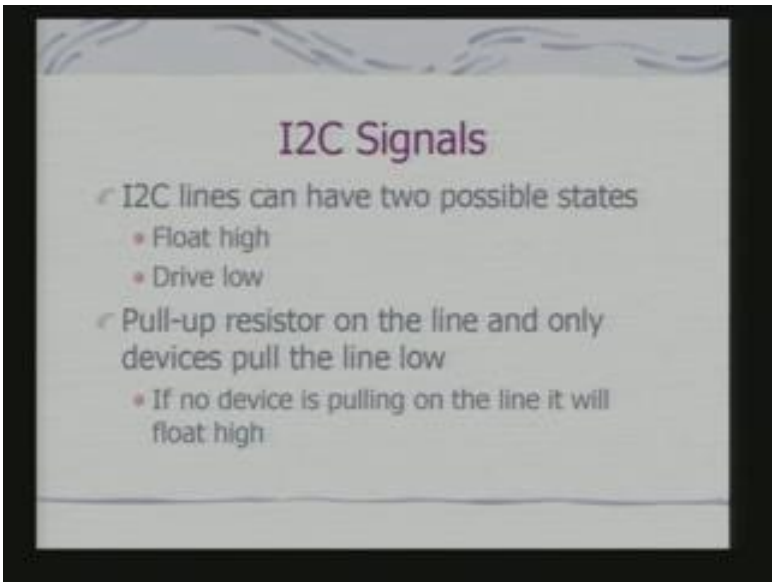
In the last class, we had discussed the buses which we expect to find we expect to implement in an SOC. We have also looked at serial buses and we had started discussions on one of these buses that is I2C. Today, we shall continue the discussions on I 2C and look at other serial buses as well.

(Refer Slide Time: 01:52)



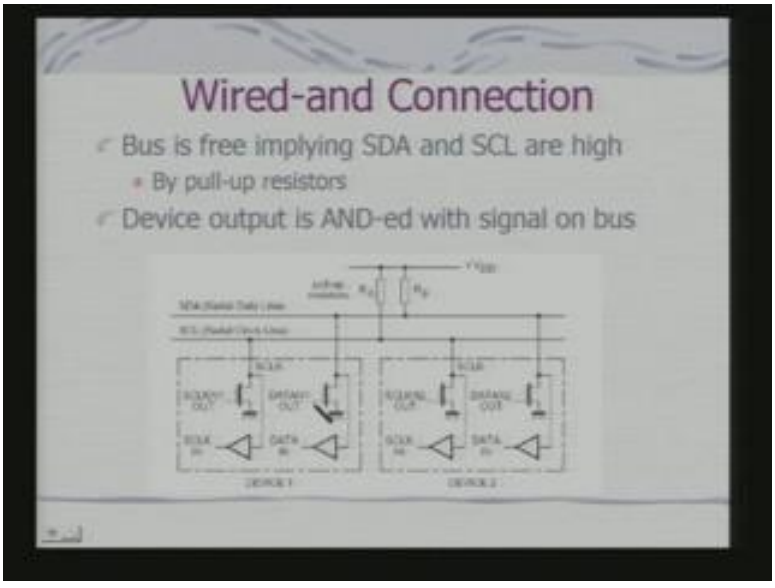
First let us recapitulate the I2C basics that I had done in the last class I2C is basically a 2 wired bus. It has got 2 lines serial data line and the serial clock line the voltage levels are logically high and 0 that is low. The bit transfer takes place when your SCL is 1. So, SCL 1 implies that SDA line now has a valid data. So, you got to have valid data when the clock is high and data change can take place. The transitions on the data line can take place when your clock is low. So, basically it is a 2 line bus and using 2 lines there can be number of device hanging one after another and they can communicate data with each other using just 2 lines. So, here the data transfer is in a bit serial fashion and that is the basic difference with any of the parallel buses that we have seen so far.

(Refer Slide Time: 03:07)



So, I2C lines can have 2 possible states what we say a float high and drive low and pull up resistor on the line are connected and only devices pull the line low. So, if no device is pulling on the line it will float high; that means, by default the I2C signals will be always high.

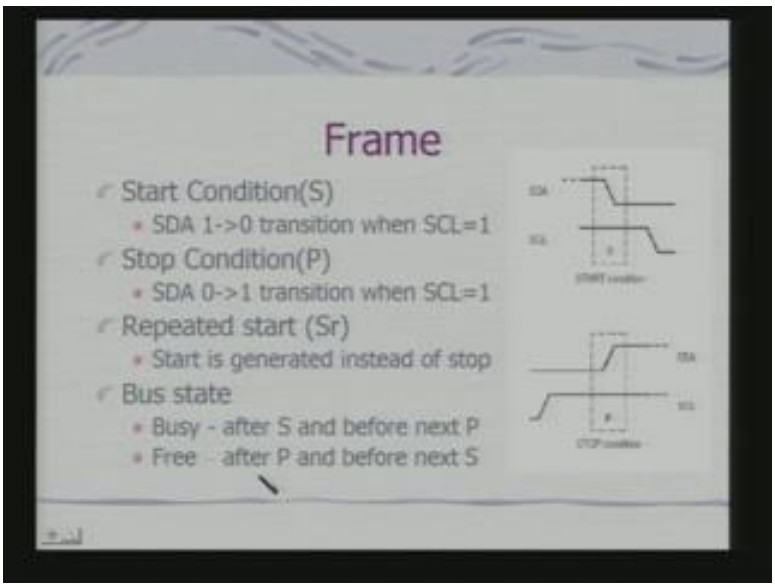
(Refer Slide Time: 03:39)



So, what is really the interfacing circuitry here you will find that I have shown 2 such blocks if you look at it the, these are the 2 lines. This is corresponding to the SCL and this corresponds to SDA the serial data line and there are 2 devices hanging from these 2

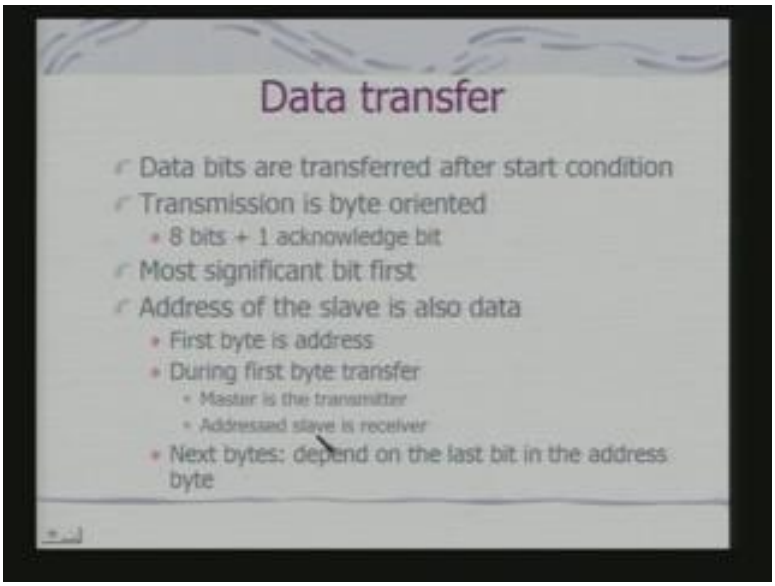
lines. So, here the serial clock is connected and this will be the internal input to the device and this is the out and this is basically in. So, what is that mean? That means, that when it is driving when this device is intending to drive the bus. The clock should be generated by this device else it will get the clock in that is the clock which is currently on this line being generated by some other device. Similar thing is true for the data line that is SDA line as well. So, these are all Open Collector Interfaces or Open Rain Interfaces. So, you have got the resisters connecting these lines to the VDD that is rain voltage. So, that the normal condition for all of them is high an effectively these connection is nothing, but wired ending. So, a device can pull the line low this is the basic structure and interface of the devices for connecting onto the I2C bus.

(Refer Slide Time: 05:33)



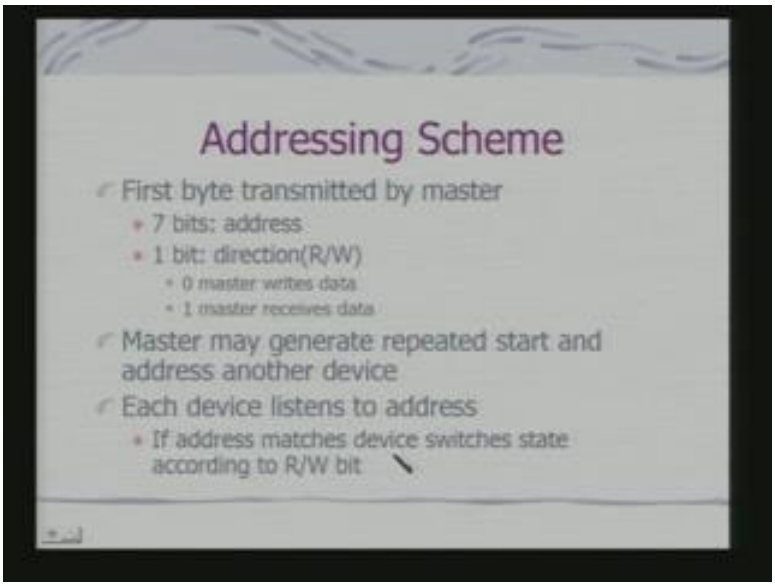
Now, when the transmission is to takes place there has to be start condition and stop condition. Start condition is when the SDA line makes a transition from high to low when SCL is high. So, when the clock is high and SDA is making a transition it means there is a start of a frame. There is a frame of data that is being transferred the stop condition occurs when there is a transition from 0 to 1 and when SCL is high. So, these two states this is S and this is P the stop condition there can be also another state what is called a repeated start. So, start is generated instead of a stop after a frame. So, bus is normally busy after S and before next P and free after P and before next S. So, this actually indicates beginning of a data frame which is to be transferred. So, what happens in data transfer?

(Refer Slide Time: 06:46)



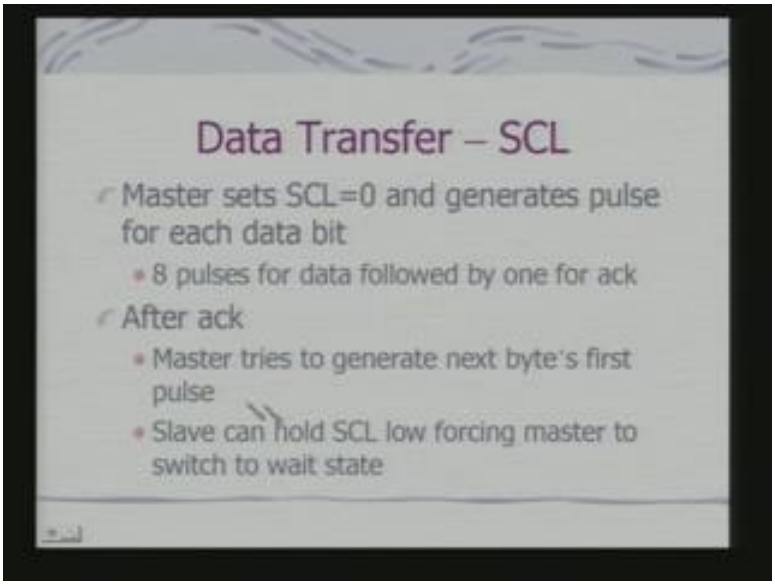
Data bits are therefore, transferred after start condition and whole transmission is byte oriented. So, one unit that your transfer is 8 bits, but that is consider as 8 bits data plus one acknowledge bit. This acknowledge bit is to be transmitted by the recipient of the data and the order in which data is send is that the most significant bit is send first. In fact, since there are multiple slaves which can be connected onto the bus we need a mechanism to identify the recipient of the data. So, address of the slave is also data. In fact, the first byte after this transition when you start transmission is actually an address. During the first byte the master is a transmitter and address slave is a receiver and next bytes depend on last bit in the address byte. So, depending on whether it is a read operation or write operation either it will be master transmitting the data or it would be the slave transmitting the data. The recipient will always generate the acknowledge bit after transmission of 8 bits. So, this is the basic addressing scheme.

(Refer Slide Time: 08:24)



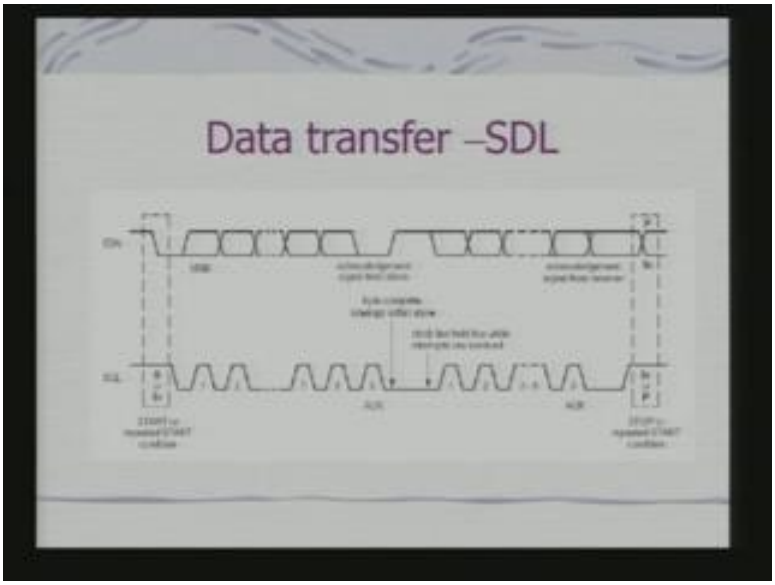
There is seven bits which is used for addressing the slave devices. So, each slave device can have a seven bit address. The 8 bit is basically the direction which tells you whether it is a write or read. Master may generate repeated start an address another device; that means, the bus still continuous to remain busy. Each device listens to address if the address matches that of the device the device switches state according to its read or write bit. Read or write bit as part of the data which is getting transmitted from the master.

(Refer Slide Time: 09:24)



So, what is exactly the synchronization takes place the master sets SCL 0 and generates pulse for each data bit and 8 pulses for data is to be followed by one for acknowledge. So, here master responsibilities to generate the clock although the acknowledge data will come from the slave, but master will be generating the clock because it is driving the bus. After acknowledge the master tries to generate next byte's first pulse. The slave can hold SCL low. So, if the slave holds SCL low then the master will switch to wait state because it cannot send the next data, because devices not yet ready to receive the data. So, that ready condition is now being provided by pulling the clock low.

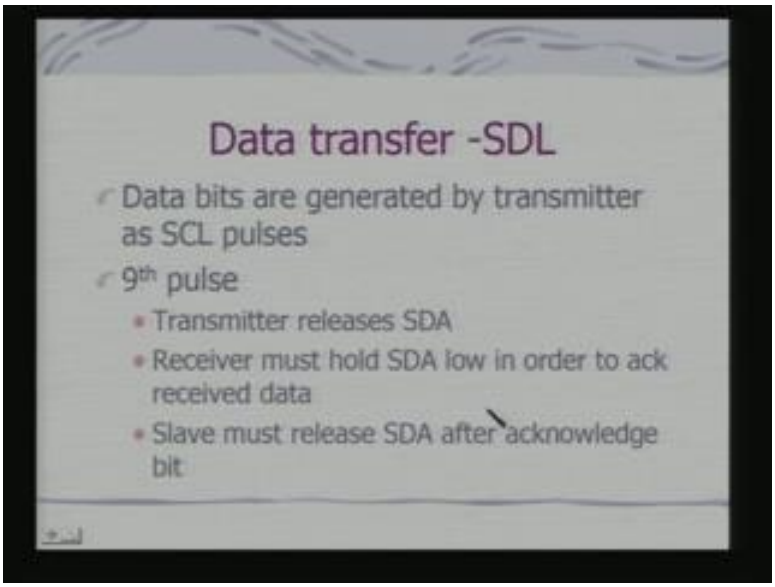
(Refer Slide Time: 10:22)



So, pictorially if you look into it. So, this is basically the transition, which is taking place from one to 0 when my SCL is high which is indicating that a data frame is starting followed by these. There are this clock pulses which have been generated by the master and these data is being sent with the most significant bit first. Now, at this point corresponding to these clock the master does not send any data. And here actually the whole reason is why master is not slaving because it is really waiting for the acknowledge to come from the slave. After these what happens the slave will pull down the clock because slave would like to now have a wait state. Because it may be using an interrupt slaves, may be using an interrupt call to transfer the data or do anything any other housekeeping task. So, it introduces a wait state here. So, when there is an wait state effectively, no data is sent again the clock starts pulsing at these point. And from these point onwards the data transmission can start. And here what is being shown is that

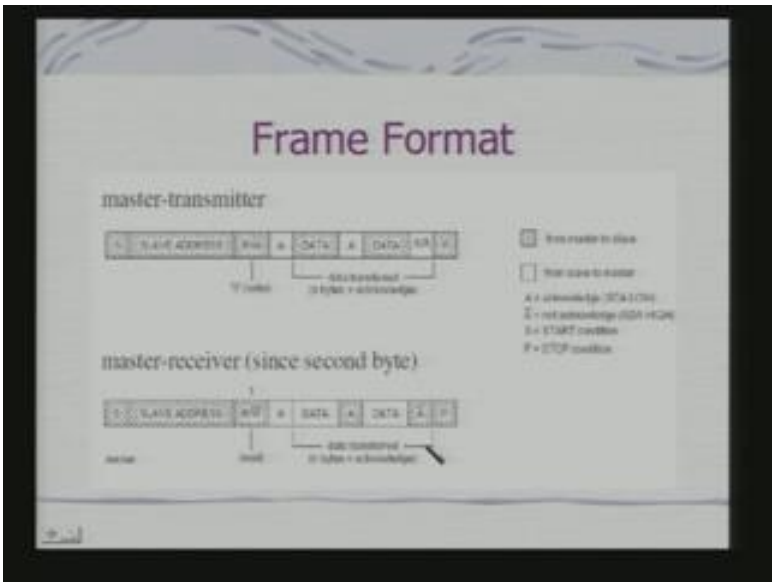
after that there can be a restart or a stop condition that is depending on how your SDA line is going through a transition when your SCL is high. So, this is exactly how the data transfer is coordinated by the clock the master clock.

(Refer Slide Time: 12:17)



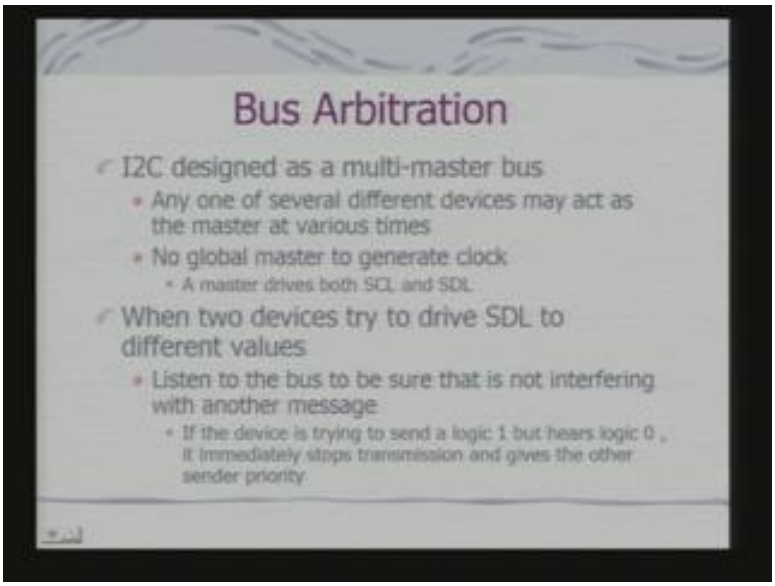
Now, the data bits which flows along your SDL Serial Data Line is generated by the transmitter. Now, transmitter can be master or may be the slave depending on whether it is a read or write operation in the ninth pulse the transmitter releases SDA. The receiver must hold SDA low in order to acknowledge the received data and slave must release SDA after the acknowledge bit. So, that now the master can against start transmitting provided the clock is not pulled low by the slave. So, now, if you see the waveform, if it is the same thing so, what you a finding here, this acknowledgement which is coming. So, this acknowledgement here the SDA line is now pull low by the slave and this is corresponds to this clock pulse which is generated by the master.

(Refer Slide Time: 13:34)



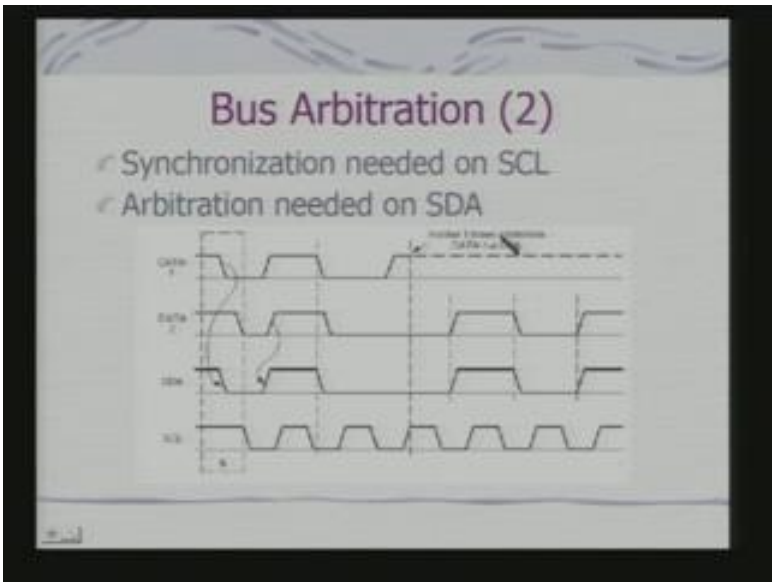
So, the frame format is you have got the complete frame format is you have got the start transition. You have got the slave address then you have got the read write bit then actually you should start the address. Now, here A is the acknowledgement which comes from the slave. So, after your first byte which indicates address you got to have an acknowledgement from the selected slave if no such slaves exists are currently connected to the bus. Then, obviously, the acknowledgement will not come and the frame has to be important. So, this is exactly how the whole frame format looks like and the master receive this is a second byte onwards this effectively similar. So, you have got the data which is getting transferred and with the corresponding acknowledge. Now, I2C is basically a multi master bus. So, in that I2C bus there are multiple devices which are hanging from and they can take up the role of master or slave in an interchangeable fashion. So, there could be multiple master which can be there on the bus. So, you need a mechanism for bus arbitration.

(Refer Slide Time: 15:25)



Now, any one of several different devices which are they are on the bus may act the master as a master at various times. And what you have observed I think by now that there is no global master to generate a clock. In fact, the device which is currently driving the bus becomes the bus master and the master drives both SCL and SDL as a case may be now when to devices try to drive SDL to different values there would be actually a conflict. So, what is what it is done is the device can listen to the bus to be sure that it is not interfering with another message; that means, if a device is trying to send a logic 1, but hears logic 0. It will immediately stop transmission and gives the other sender priority. So, the moment it detects a conflict it would be draw. The whole logic the hardware should be build in such a way that when the devices with trying it is not really disturbing the data which is pursing on the bus an after sometime again it should be tried out.

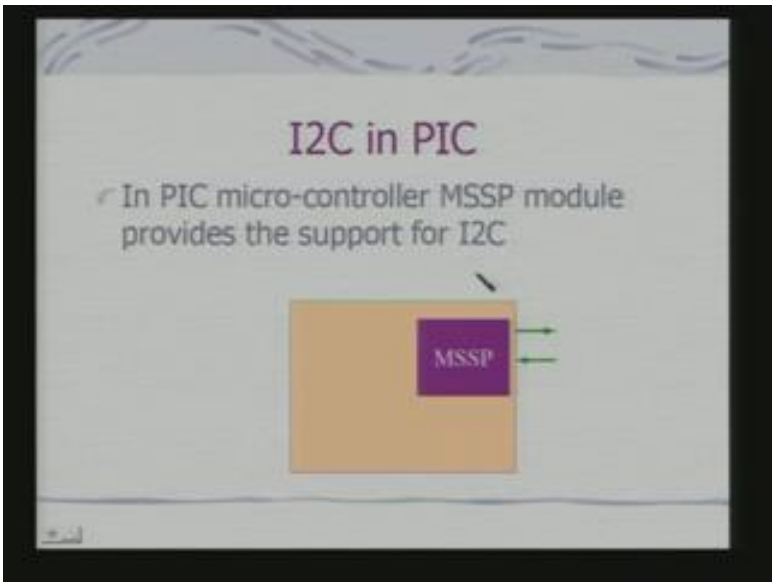
(Refer Slide Time: 16:53)



This is the basic scheme of the bus arbitration. So, if you looking to it that this whole arbitration implies the there is a synchronization with respect to SCL, because all these decisions are being made with regard to your SCL and SDA lines. So, what is being shown here is the SCL and SDA line of the bus and these are the data may be coming from 2 distinct devices. This is your data 1 and this is from data 2 now what you are finding here is that, the data when it is going through this point both devices are generating identical data corresponding to this clock. So, now since the data is identical there is no way to detect a conflict there is no way to detect a conflict. So, the other device which is sending it it will not detect a conflict and it may assume then the bus is with it only. Now, what happens is that when it comes to the next one there also you will find no conflict. The conflict comes in at this point only with respect to this clock.

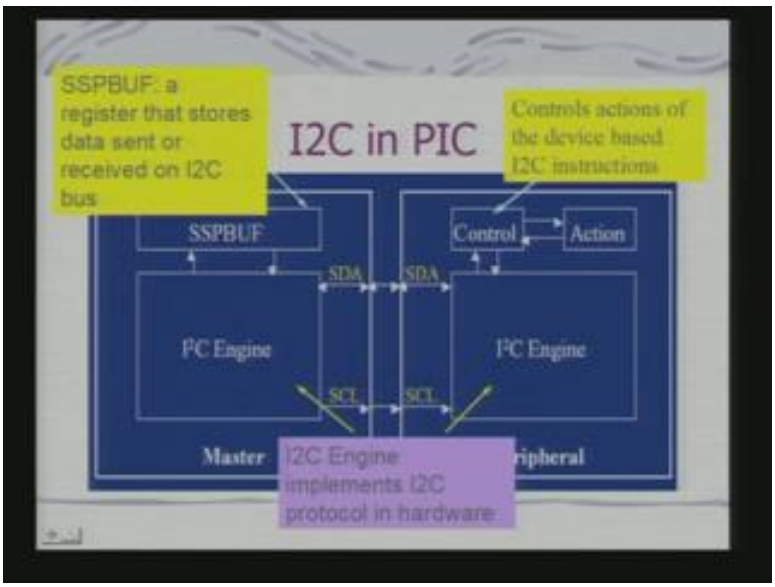
So, here there is a mismatch between the data to which is currently driving the SDA line and it is finding that this is the data is high. So, it cannot drive the line and hence it would realize that there is a conflict and it should ((refer time: 18:56)) the bus. Now; obviously, this whole logic works out with respect to a synchronized clock, because the all these transitions I told you that the data valid condition is when your clock is high. If the SCL is not synchronized for these 2 and if 2 devices start issuing non synchronized SCL then there would be a problem. So obviously, the bus arbitration requires as step for synchronization on SCL and then the arbitration on the serial data line. Now, I2C is available the hardware for I2C is available in a variety PIC microcontrollers.

(Refer Slide Time: 19:47)



In variety of microcontrollers and in PIC family as well typically this MSSP module which is their in PIC provides the support for I2C.

(Refer Slide Time: 20:00)

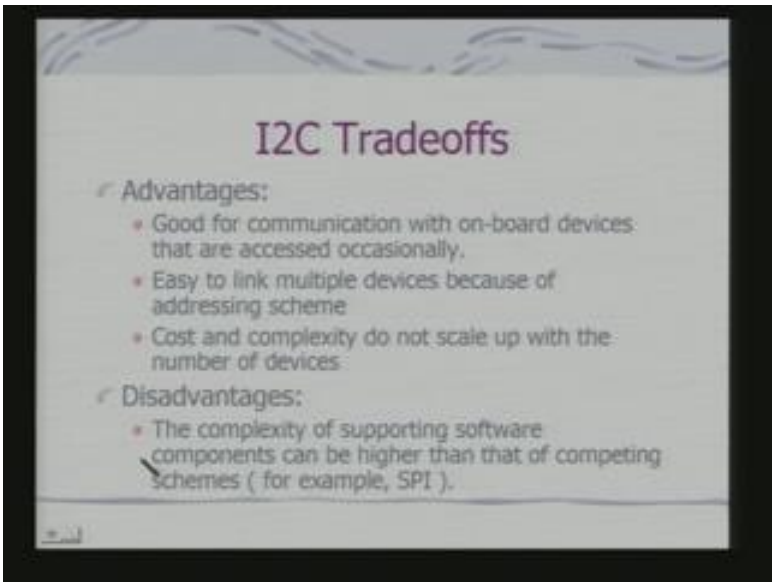


Now, the basic structure is that in the I2MSSP module you have got these I2C engine you have got these 2 registers SSPBUF. This is a register. So, this is the master and this should be the corresponding peripheral. Peripheral should also have the I2C engine otherwise it cannot communicate following I2C protocol. So, it has got the control on the control logic as well as action part. So, effectively the I2C engine implements I2C

protocol in hardware. This SSPBUF is a register that stores data sent or received on I2C bus because there will be serial data. So, it has to be stored and buffered and this control controls action of the device based on I2C instructions. Now, try to understand the difference between the functional role of I2C engine and that of these controls. Because if you remember I said that if I have let us say a data frame I have started and I want to read data the master wants to read data. So, I2C engine will detect this, read operation and depending on that read operation the control logic has to be activated to provide data on the serial data line.

So, this is this control logic controls actions of the device based on I2C instructions. In fact, in a mode general case if you remember I say the I2C bus has been used as a control bus for connecting on to various peripheral devices. So, if the peripheral devices has got as a set of commands then those commands will be transmitted where the I2C bus. There will be dedicated by the I2C engine and passed on to control logic. So, that there can be device specific action. Now, the most interesting thing that we have seen therefore, with respect to the I2C is that it almost provides all the capabilities that you typically find in a parallel bus. Although the bus is essentially serial and based on just 2 lines the basic point I was making is not in terms of the foot print that is being satisfied in this case. So, on a board if I on to connect my PIC microcontroller with say another device and use a small foot print PCB at desirable feature would be to use I2C bus. Because I have to just put in 2 wires to connect a variety of devices a peripheral variety of peripherals to my PIC microcontroller. And it has got the complete logic for bus arbitration which you typically find in a parallel bus as well.

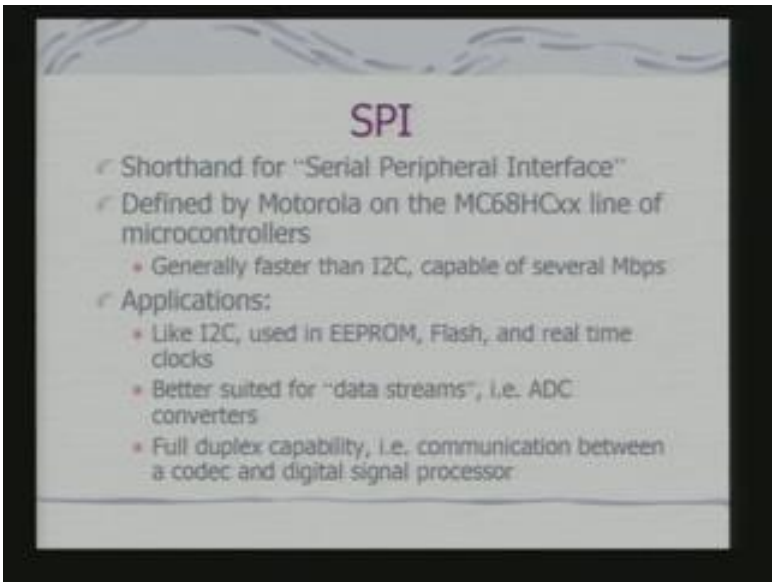
(Refer Slide Time: 23:30)



So, what are the advantages? Advantage is good communication with on board devices that are may be accessed occasionally because if it is being regularly accessed say a cache memory. Obviously, I would not like to use a serial bus because that would be delay. But if you remember I said that in PIC E square PROM has got an I2C interface why, because E square PROM will be typically a program which is stored which may be downloaded onto a RAM for execution or some book keeping data would be stored on the E square PROM. So, these are occasional reads and writes it is not an a regular basis. Now, this bus is easy to link multiple devices because of its addressing scheme and cost and complexity do not scale up with the number of devices, because you still continue to use simple 2 lines and you connect an additional device onto the bus.

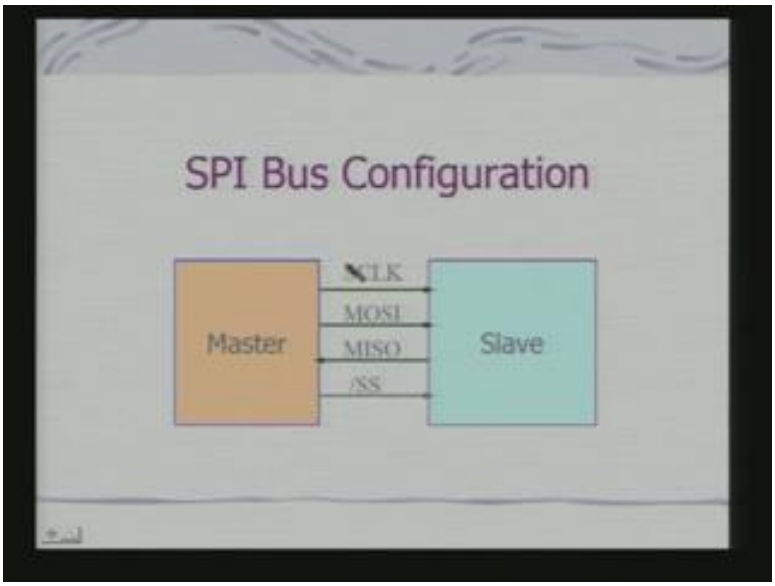
The complexity of supporting software components can be higher than that of competing schemes. For example, SPI why because you have a complete bus protocol starting from how the address has to be put on the bus, how the address has to be interpreted and made use of . So, the complexity of the supporting software as well as hardware may be not desirable feature with regard to a simple application , but the more important things is these which makes I2C more popular in. In fact, these software components they implementation of software components are also facilitated by making appropriate hardware elements available in the hardware blocks providing the I2C interfaces.

(Refer Slide Time: 25:31)



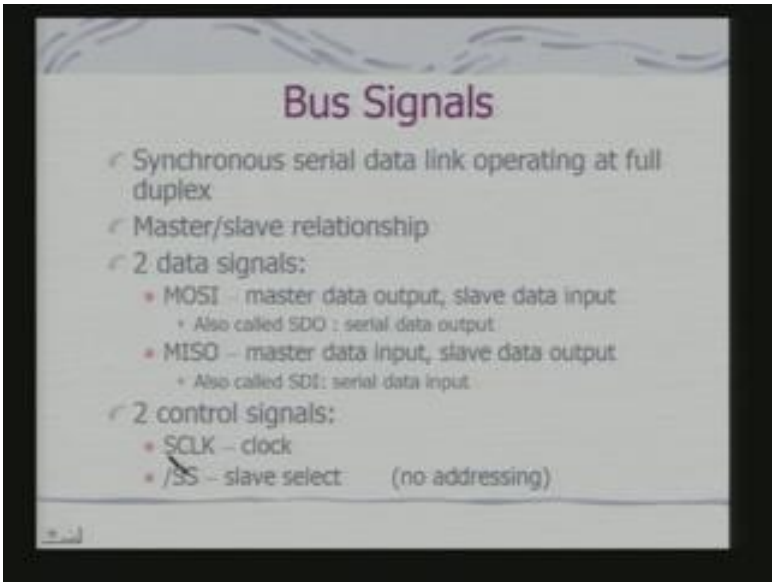
So, next we shall look at SPI which we say is a competing serial interface and this is called Serial Peripheral Interface an. In fact, it was defined by Motorola and it is now again available and can be implemented on a variety of this microcontrollers including your PIC microcontroller. It is generally faster than I2C that is capable of several MBPS. An applications is like to I2C used in E square PROM Flash real time clocks and it is better suited for data streams like your ADC. The basic difference is it has got a full duplex capability in case of I2C it is a how duplex, because I have got a single data line. It has got a full duplex capability that is and therefore, it can be used for may be a communication between a codec and digital signal processor.

(Refer Slide Time: 26:43)



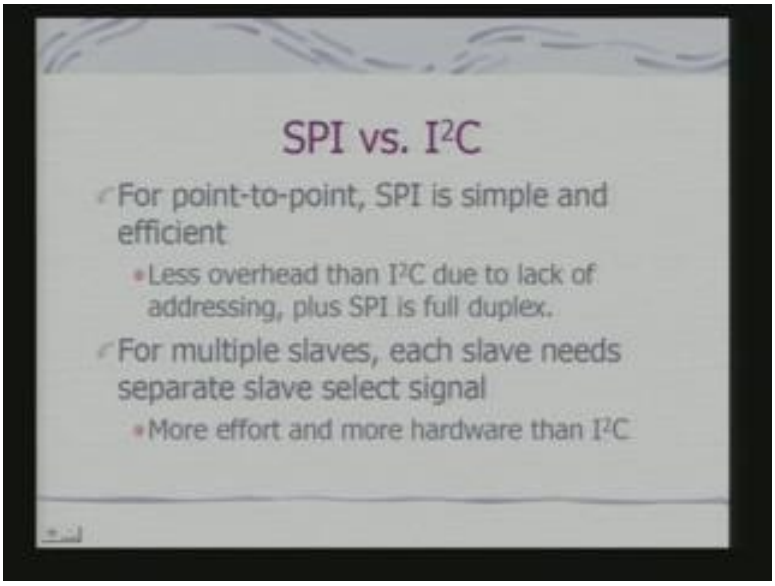
So, let us look at these now this is not like a it is not using just 2 lines, but using 4 lines. It is there is a master there is a slave and what are the interfacing signals, this is the basic clock. This is a input output this is from master output and this is an input to the master. So, these are the two data lines and that is precisely reason why it is a full duplex mode communication and there is another line which is called slave select. This is basically bar. So, active low this is a slave select line. So, using these lines you select the slave. In fact, this implies there is no distinct capability in the SPI bus configuration of associating unique addresses with the device themselves.

(Refer Slide Time: 27:54)



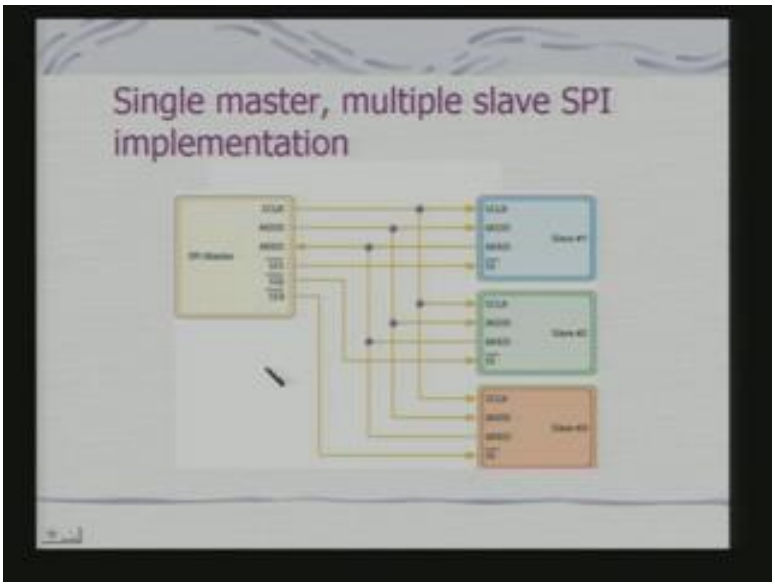
So, synchronous serial data link is operating at full duplex. So, it is a synchronous bus because you have got a clock. So, it is a pure synchronous bus and synchronous serial data link and it operates in a duplex mode. The basic model is master slave relationship. In fact, there is there cannot be really multi master on this configuration. There are 2 data signals these master data output slave data input also called is also called SDO Serial Data Output and MISO Master Data Input Salve Data Output also called serial data input. It has got 2 control signals one is; obviously, the clock other is that of selection of a salve. So, it is a it is an active it is a logical line active low logic line and there is no address in the point I had already told earlier.

(Refer Slide Time: 29:00)



So, now if you compare the 2 for point to point if you look at SPI is simple and efficient less overhead than I2C due to lack of addressing and plus SPI is full duplex. So, it can be much faster than that of I2C. So, for multiple slaves, each slave needs separate slave select signal. These implies that there will be more effort and hardware cost on the master itself. So, you can realize that here the whole approach is that I am trying to get fast data transfer to slave devices are from slave devices and there is no provision of a slave taking up the role of a master at any point in time. So, the basic configuration in these case would be therefore, something like this.

(Refer Slide Time: 30:07)

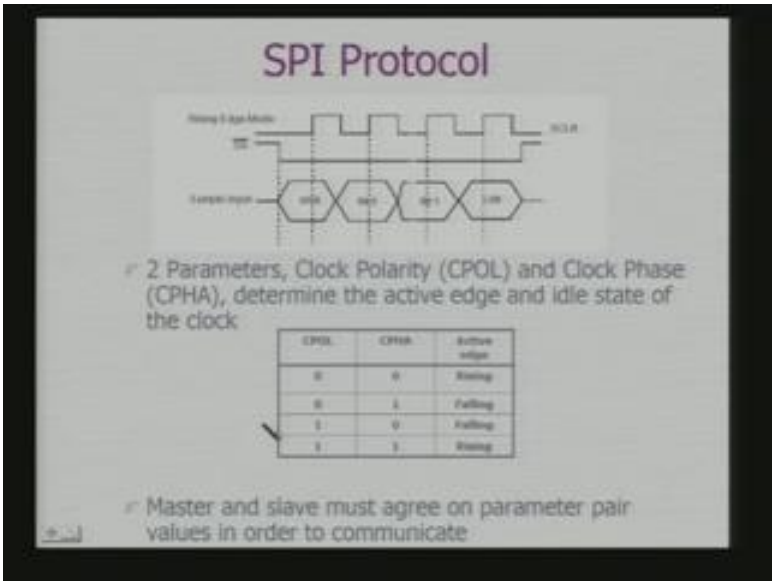


So, you have got an SPI master and then the interesting issue is that in this case. I have got multiple slave select lines just like if you remember that when we talk about interfacing memory what happens when we interface a memory I put in a decoder. In fact, we have seen that in the context of parallel buses we have a decoder where we fit the address. And decoder generates the chip select signal by which I select appropriate memory block and then I refer to a particular location in that memory block. Here, actually master needs to incorporate a similar kind of a decoder. So, master would know the address of the slaves and it would provide in the software the address to the decoder and decoder would generate appropriate slave select signal which would flowing to the slave and. So, that they are selected and they able to respond to whatever is coming from the master. So, in this case the entire onus is on the master and the hardware complexity of the master can also go up if it needs to interface multiple slaves. Now, so, here it is not truly speaking if you looking to these this not truly speaking a bus. Although we are comparing with I2C I2C is a bus why because I said there is a typically buses one feature is that of a set of signal lines and number of devices being connected to that set of signal lines.

Now, here all the slaves are connected to the same three signal lines, but each of them have their own private slave select line in that sense is not strictly a bus. The, these three signals are same, but the select option is different and here therefore, you have got distinct slaves related to the master. So, if I have a problem of interfacing a device and

when I know thus device will be always a slave device. And it is not likely to become master to take up basic job of initiating a transfer of data on the bus. Then I would select SPI as a protocol mechanism because it enables be faster transfer, but when I need to connect multiple devices with a controller or a microcontroller where I know that these slave devices have the capability to become master and I would like this capability to flow to the slaves. So, there can be even data transfer between 2 peripherals directly without involving even the processor then I would like to select I2C as a preferred option. This is a vary key point to remember when you designing a system based around microcontrollers like PIC which provide support for both I2C as well as SPI.

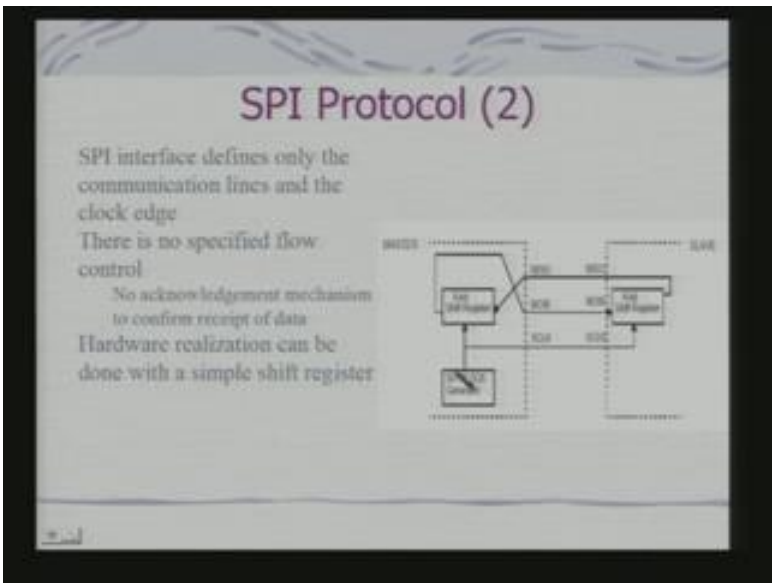
(Refer Slide Time: 34:18)



So, let us now look at the details of SPI protocol. So, this is the clock and what we are telling is I have got a rising edge mode of the clock. In fact, interestingly I can defined what kind of clock would be used for synchronization on the SPI bus by using basically 2 parameters what is called Cock Polarity and Clock Phase which determines the active edge as well as idle state of the clock. Now, this is a rising edge mode. So, these clock is. So, all these data valid conditions would be with respect to the rising edge of the clock and I got to have my slave select signal active for any kind of data transfer to take place and this is a sample input. So, you have got again the same condition that you start with MSP first just like in I2C. I had the most significant bits send first here also the most significant bit is send first and the entire sampling or the data valid time is defined with respect to the rising edge of the clock.

In fact, this is a kind of a convention or combination which SPI is specifies and talks about it is a 0 0 is a rising edge 0 1. All these a falling edge and corresponding to these combination there will be definition of the idle state of the clock. And that is why these combinations defined four distinct specification of clock and the master and slave must agree on these parameter pair values, in order to communicate otherwise the slave will nott understand whether you have got a valid data or not. So, in a way what it implies is that when multiple slaves are getting connected to the master. Master would need to know with reference to the slave, what is the convention to be followed. So, if you have a master hardware. The master hardware should have provision of specifying these 2 parameters.

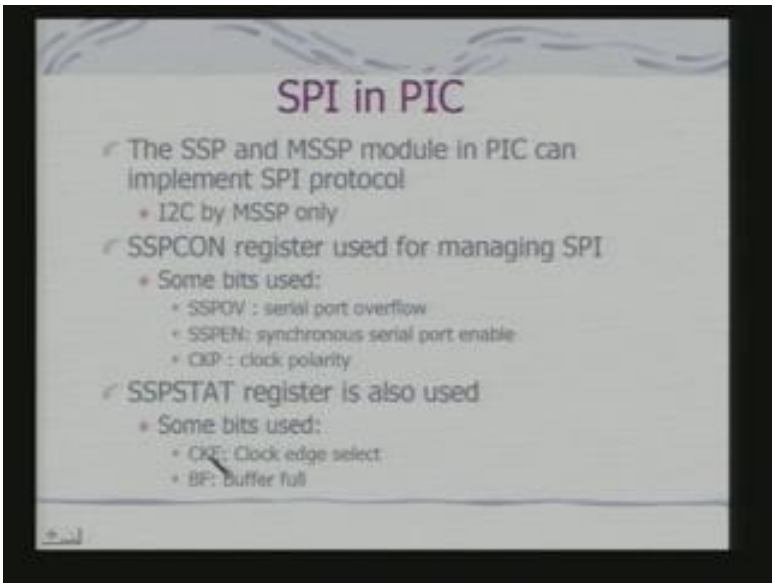
(Refer Slide Time: 36:59)



So, SPI interface defines only the communication lines and the clock edge and there is no specified flow control. No acknowledgement mechanism to confirm receipt of the data. So, if you want to build up that mechanism are the protocol that is up to you. So, if you have designed a peripheral with a certain kind of an acknowledgement scheme. Then you need to write the software on your microcontroller to make sure the whole transfer is confirming to that protocol that you have designed. Please note this, because in an I2C I have a basic protocol already specified as part of the standard where I have got definition of a frame that is start stop we have got definition of acknowledgement. So, that flow control is part of the I2C protocol. In SPI it is just definition of the communication lines and the clock edges your free to develop your flow control protocol and implement.

So, in that context your hardware realization the basic hardware realization can be done with the simple shift register. If you see, because shift register why because it is serial input output. So, if you look into here you have got the SPI clock generator and this SPI clock generator could come from the always master because master is driving the bus. Master would be to know that for a particular slave what kind of clock configuration is defined by those 2 parameters that I had earlier discussion. So, it will generate the clock and depending on the data which is getting generated. So, either it will be, if the slave is generating the data is shifted out by the clock and the data is coming here. This shift register is also effectively clocked by the same clock and. So, the data would be ready. In fact, this is a very straight forward way if I continue to use it in definitely. Then the same data will go from master to slave and from slave back to master, but in a processor its implementation request certain support.

(Refer Slide Time: 39:43)

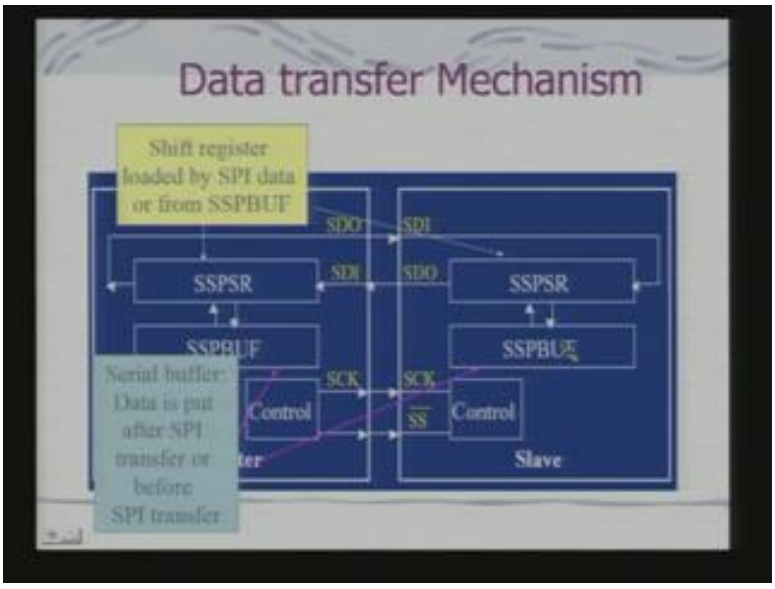


So, PIC provides that support for this processor for these protocol. In fact, there is SSP protocol and SSP module as well as MSSP module in PIC through which you can really implement SPI. MSSP is I2 can be used for implementation of I2C also. So, in fact, using a PIC, if you are using that version of PIC, which has got both SSP and MSSP protocol. You can setup on I2C bus a connect a set of peripherals via I2C bus as well you can connect specific devices via SPI protocol using may be SSP module when there are number IO devices is to be connected then you can adopt this kind of architectural module. The two basic registers other than registers other registers data registers which

are involved. The two basic registers which are involved in PIC for managing your SPI is SSPCON this is SSP control register an SSPSTAT SSP status register. The some bits which are used are one is SSP overflow if there is a buffer overflow if you have seen the shift register. If you do that kind of a connection there one be a any and any time buffer overflow, because data is flowing from master to slave and back to the master.

But really I have to store the data in a shift register and if that shift register is not getting cleared. The problem is there would be a overflow. So, the system should have an ability to detect whether such an overflow is occurring or not. So, a bit that SSPCON bit indicates whether there is a serial port overflow then you have got this enable bit. In fact, if you want to implement your SPI you have to first enable the synchronous serial port only when if you only when if it is enabled the whole implementation can really work. Then you have got the 2 bits one is your CKP which is clock polarity and other is CKE. Now, this is in SSP stack register clock edge select and this is also the buffer full indicator is also there in the SSP standard. So, obviously, depending on the slave and depending on the modality that the slave is excepting the modality that a slave is excepting you have to set CKP and CKE bit for during any kind of data transfer.

(Refer Slide Time: 42:41)



So, again this registers are involve this is SSPBUF register we have already seen earlier in the context of I2C. The same registers are involved and this is the control logic which would generate the clock and it has to also generate the slave select signal. So, what we

have got here is SSPSR these are now shift registers. SSPBUF was a buffer register and it is not a shift register . So, the shift register is loaded by SPI data when the data is being received and when the data is being sent the data has to be loaded in parallel onto this shift register and your serial buffer this is your SSPBUF. Now, the data is put here after an SPI transfer or before SPI transfer that is from in these case, but you see which is interesting to note is that once the data is loaded onto the shift register. The loading of the shift register will be guided by the clock once the data on is loaded onto the shift register it is transferred to SSPBUF.

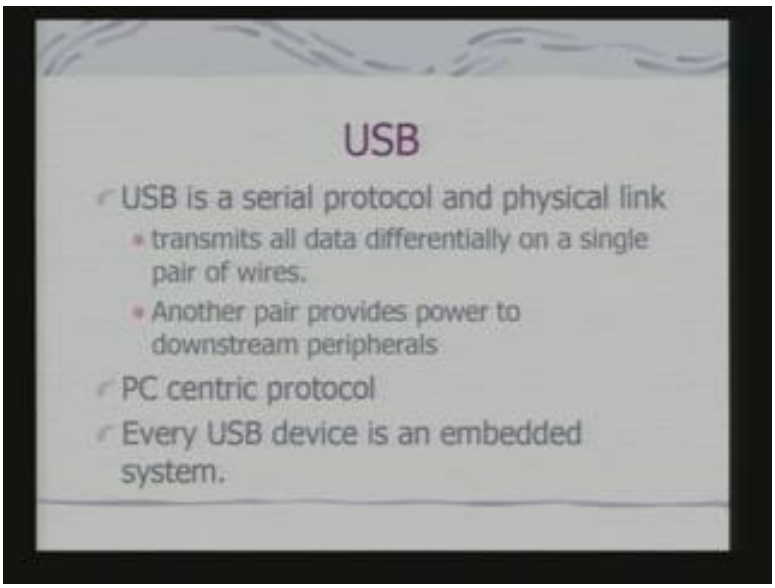
So, it is a kind of a serial in parallel out operation when you are actually transmitting the processor will load data onto SSPBUF and then SSPBUF will load that data onto SSPSR. So, it is parallel out from here and parallel in to SSP here and then it is shifted out when the data is being sent. So, this is the basic mechanism involved here with regard to the device of the slave I have also shown existence of similar registers. Whether their name SSPSR or SSPBUF or not they will have the similar capability I got have a serial shift register to get the data. And from there it has to be loaded in parallel to a buffer register and vice versa for the purpose of transmission. So, therefore, what we have seen so far is mechanisms for connecting peripheral devices to the processor using serial bus.

I2C is a true bus which can be a multi master bus this is SPI is a bus where I connect one device to the processor. There may be a multiple devices to be connected, but each of the device which have been connected will be slave. They cannot take up the role of the master and each device is being selected by a dedicated slave signal. Now, we shall take our attention to a slightly different problem. So, far what we have look that in terms of parallel bus as well as the serial buses, the problem of designing the Embedded System, and how to use bus inside an Embedded System? Fine. Now, the problem is once have made the Embedded System I need to communicate with that Embedded System as well. So, how that protocol is to be defined fine. So, we are going to another domain. So, we talk about what we call communicating with Embedded Systems.

Now; obviously, this communication means that there would be a kind of a host and we are talking about communicating with Embedded System with the host. What is the other possibility? Other possibilities there could be 2 Embedded Systems and I might like the 2 Embedded Systems to communicate with one another a kind of a peer to peer communication. Now, there are various modalities by which this communication take

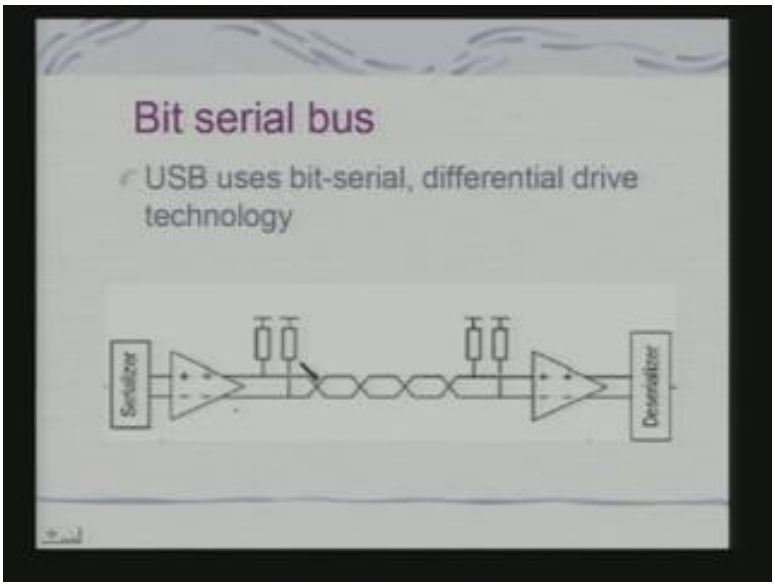
can takes place. If we are going into a complete network base protocols we can use that network base protocols for communications I can have Embedded Network devices that is embedded devices for the complete networking protocol is enabled. Now, before going into the network issues we shall address network issues later on in this course here we shall look at specific protocols which have some characteristics of the network protocols. But not really full fledge it network protocols which have been used for this kind of communications.

(Refer Slide Time: 48:02)



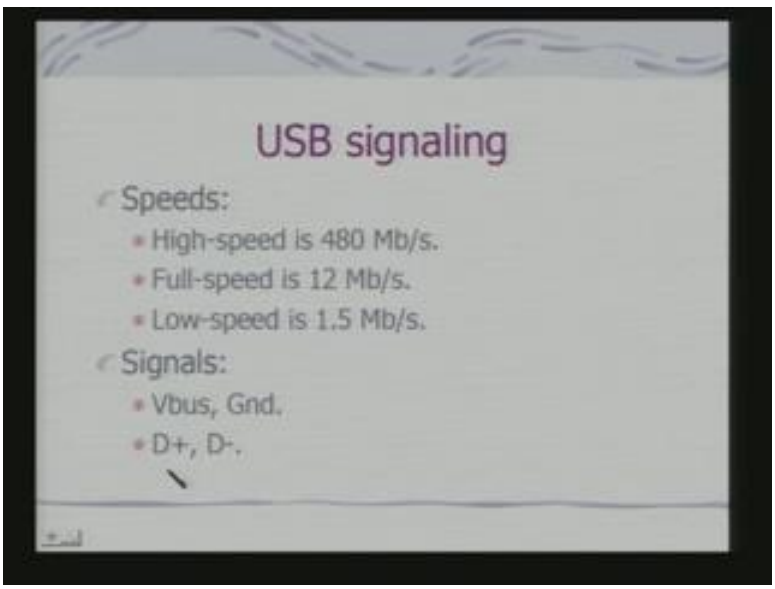
USB is one such protocol. In fact, USB is another serial protocol and also defines the physical link. Now, it transmits all data differentially on a single peer of wires and there is another peer which provides power to downstream peripherals. So, these bus also delivers power so far in the buses, we have not really talked about power delivery. And in fact, the whole development ((refer time: 48:37)) with keeping PC as a core host. So, it is basically PC centric protocol and in a way every USB device is an Embedded System. Even the flash disk, that I am putting onto my PC it itself on Embedded System with the small processor setting in it. So, why we need to k? Now, therefore, the USB protocol as an embedded designer must be very clear, because I am trying to design an Embedded System. And I want to provide that Embedded System the facility to communicate with host it should be enabled with the USB protocol.

(Refer Slide Time: 49:36)



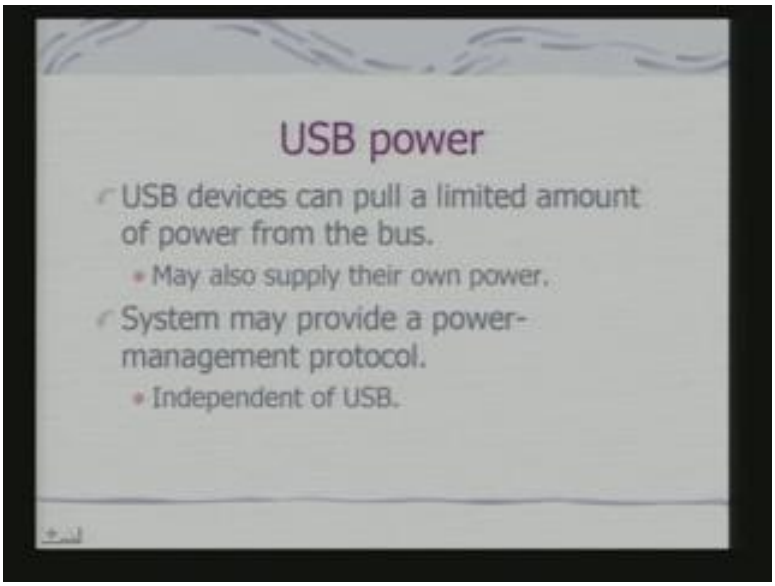
So, the basic concept is that of a bit serial bus now this bit serial bus which uses differential drive technology is distinct for USB and different from that of your I2C or SPI. Because I2C on SPI was primarily on board specifications that is typically on a single PCB when I am trying to put different devices I would connect them using I2C or SPI. Now, these are kind of external bus definitions. So, external devices are to be connected. So, it is a. So, I have got the differential voltages which are between these 2 lines which would define the big levels high and low. Obviously, it would be much more robust and signal to noise ratio would be better compare to that of a single wire bus.

(Refer Slide Time: 50:33)



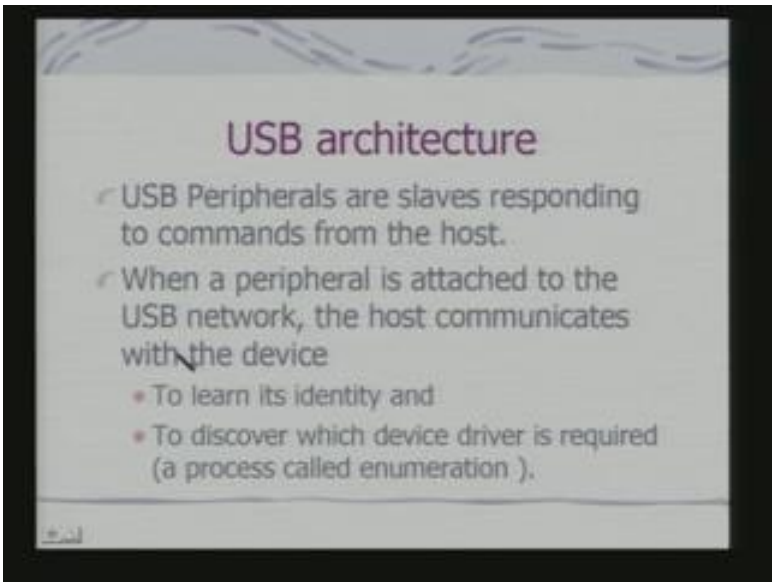
Typical speed is a high speed bus between. In fact, if it is a high speed its 480. In fact, 12 Mb it can walk to the lower speed is 1.5 Mb. Signals are basically Vbus that is a this is basically the power signal in the ground and the other signal is basically D plus and D minus is the differential levels for communicating the data.

(Refer Slide Time: 51:02)



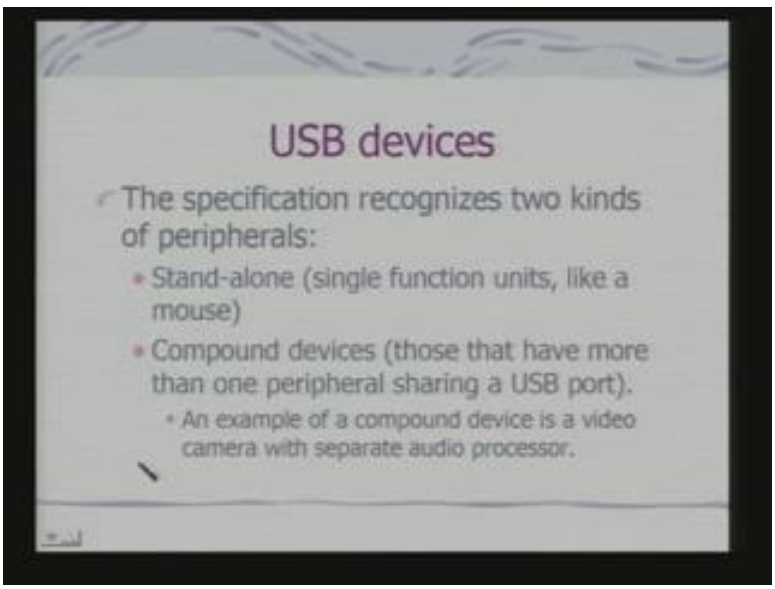
What we say that USB devices can pull a limited amount of power from the bus. If I device requires additional power it may be connected to a power line, but if it is not show it can be throwing power from the bus itself. So, my flash disk which I connect which does not need additional power is actually drawing power from the bus itself and system may provide a power management protocol. In fact, which is independent of the USB what would be the basic motivation of this power management protocol. If the devices not being used, shut down the power to it. So, actually save power in the process of communication

(Refer Slide Time: 51:51)



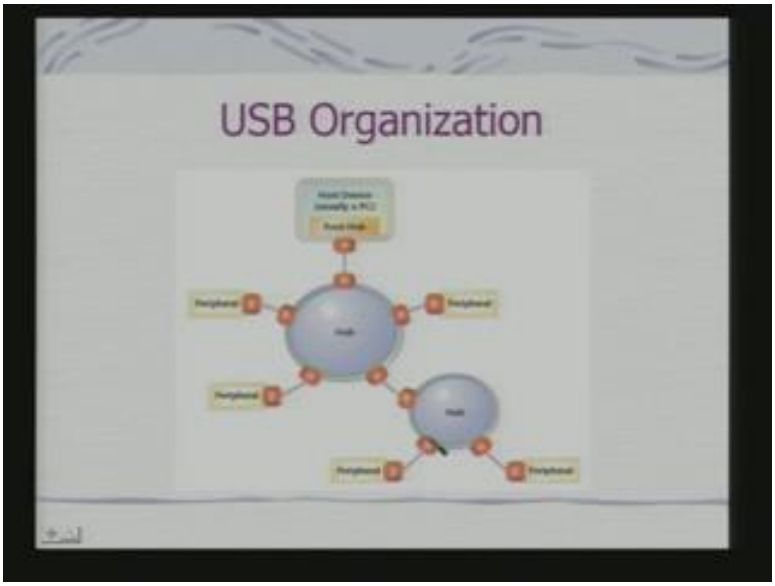
What is the basic USB architecture? USB peripherals a slaves which typically response to comments from the host. So, its I told you this a host based structure. So, there is a host and these are all slaves when a peripheral is attach to USB network the host communicates with the device to learn its identity and to discover which device drive a restrict wire. So, these process is called enumeration and these enumeration process is supported as a device driver for the USB port on the host on the master. So; obviously, the architecture wise a really a do not have what is called the multi master I have only one host and the multiple slaves which can be connected to the bus.

(Refer Slide Time: 52:49)



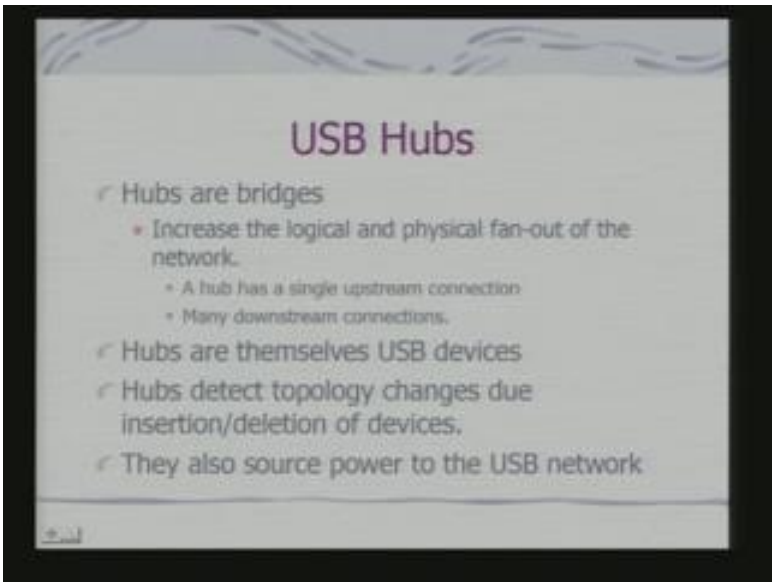
The USB devices are typically of two types one is called standard alone which has a single function like that of a mouse. There may be compound devices those that have more than one peripheral shading a port. Typical example is a video camera because it has got image captured device it has got a audio process device. So, it has to send both image as well as audio via the USB link. So, they are compound devices. In fact, USB distinguish between this to distinct types of device the basic organization is what is called a hub based organization.

(Refer Slide Time: 53:35)



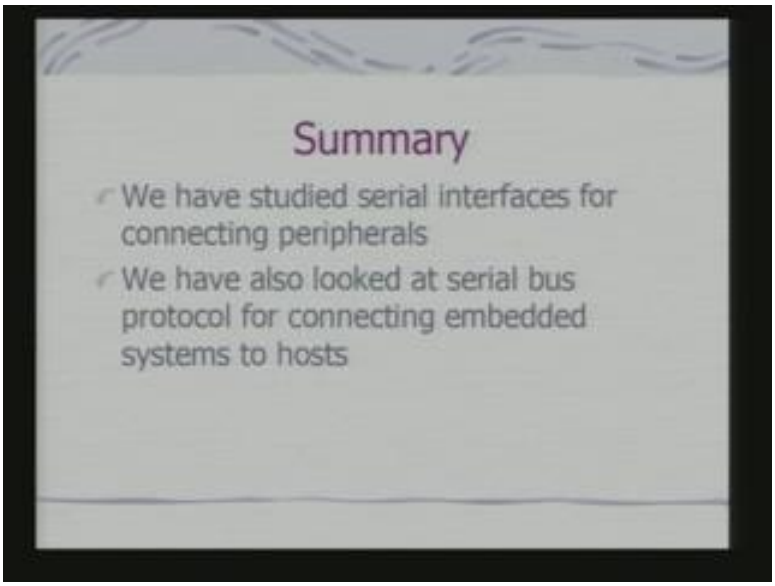
So, you have a host device. So, do a host device you can connect the hub and your peripherals gets connected to the hub otherwise you connect a single peripheral with the host device so, I can build now a tree. So, if I connect at one port another hub, I can connect more peripheral to this and that is why this is called a bus; that means, there can be multiple devices hanging from one link.

(Refer Slide Time: 54:06)



And these USB hubs are nothing but bridges; we have already studied in the context of parallel buses they are very very similar to bridges. So, the increase the logical and physical fan out of the network a hub typically has a single upstream connection and many downstream connections. So, these upstream connections connect the hub to the host. It may not be directly communicating in the host. But it may be communicating with the host through another hub. Hub themselves are USB devices and hubs of the ability which is not there in all USB devices to detect topology changes due to insertion or deletion of devices. And they are also source power to USB network because the power has to flow for the down. So, these gives us more or less architecture of the USB it has got a definite protocol for doing the data transfer which is much more sophisticated than that of typical I2C kind of a protocol. Because your now communicating with the with Embedded Systems and each of this system have got actually the capability to run this the whole protocol machine on its processor. So, it is just not a simple FSM implemented in the hardware the way it is in for I2C. Here, you have got processors involved may be at the various devices you can have more complex protocol scheme.

(Refer Slide Time: 55:54)



So, this brings us to end of today's discussions. So, we have studied serial interfaces like SPI and I2C for connecting peripherals to the basic microcontroller blocks making an Embedded System. Then you have started looking at the protocols which can be used for connecting Embedded Systems to hosts and other Embedded Systems. You have studied architecture of USB you have not really looked at protocol of USB. We shall look into protocol of USB and other similar protocols and interface mechanisms in the next class. Any question? See these are two parameters the question you have asked is what is the significant of clock polarity and the clock edge parameters. See if I have the clock now with regard to the clock whether the transitions that is the data I have the data line. So, whether the data on the data line should be valid with respect to the rising edge or falling edge can be configured. So, that can be done with regard to the clock edge. Now, with regard to the polarity what we can do is now polarity means whether high is an idle state or low is a idle state. So, therefore, using these two bits these combinations can be identified these combinations uniquely configures the clock signals for a slave device. So, that is a facility that SPI provides for.