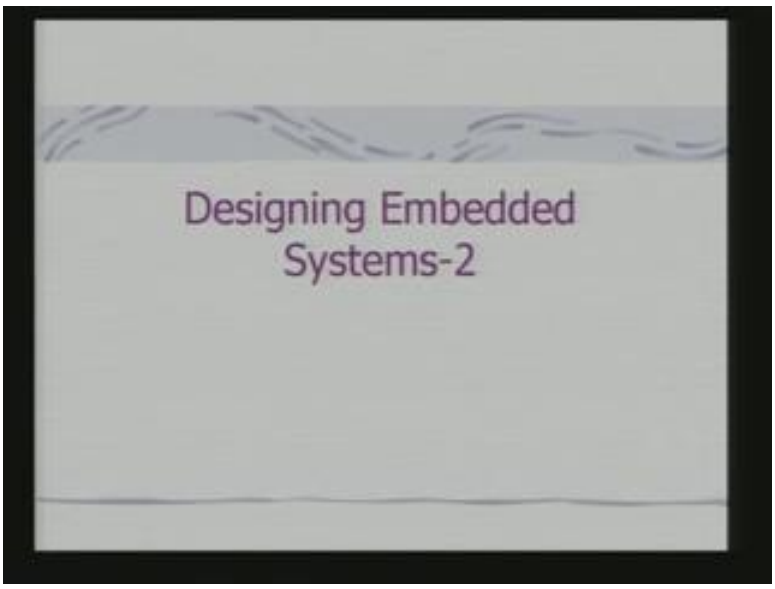
Embedded Systems
Dr. Santanu Chaudhury
Department of Electrical Engineering
Indian Institute of Technology, Delhi

Lecture – 29
Designing Embedded Systems – II

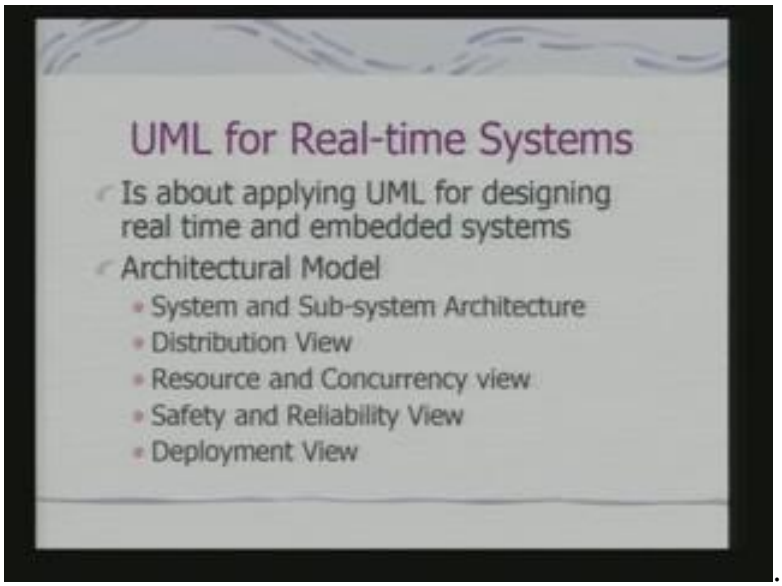
We were discussing the different strategy to be adopted for designing embedded systems. In the last class, we have looked at design requirements and then we had started looking at modeling tools. We have discussed basic features of UML unified modeling language, which provides us with an object oriented representation scheme for specifying the design.

(Refer Slide Time: 01:31)



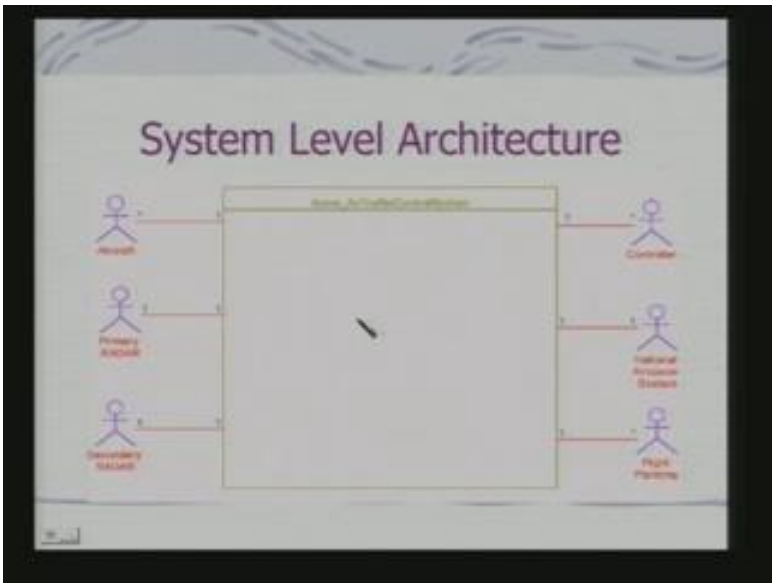
We shall continue on usage of UML for designing real time and embedded systems.

(Refer Slide Time: 01:38)



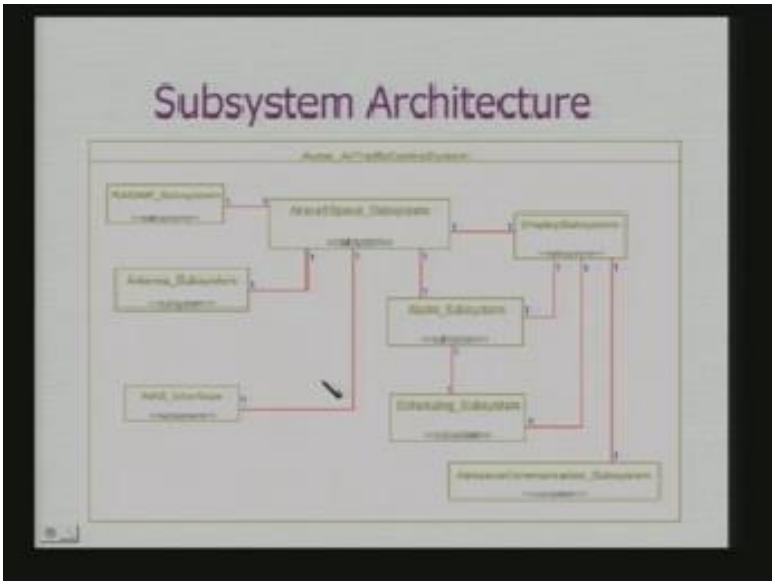
In fact, UML if you see is a kind of a on encompassing design language by which you can represent both hardware as well as software components. In fact, UML for real time systems, which is about applying UML for designing real time and embedded systems. So, one of the basic issue related to that is the different architectural models that are part of the UML language for specifying your design. You can specify your system and subsystem architecture the distribution view resource and concurrency view safety and reliability view as well as deployment view. These different views we shall discuss today and consider the relevant for designing real time as well as embedded systems.

(Refer Slide Time: 02:37)



At a top level we have the view of an embedded system as a system. So, this is a representation of work we call a system level architecture. So, in a sense if you look at this diagram this basically represents a kind of a encapsulating class which represents the system as a whole what we have shown here different kinds of actors, actors or entities, which would interact with the system. If you are talking about an air traffic control system then there would be aircraft. So, with respect to 1 air traffic control there may be many aircrafts, because many aircrafts may control by 1 air traffic control then there would be a primary radar secondary radar these are all actors. Because they are all stand alone systems providing data to your air traffic control system. Similarly there are controllers who will actually take the decision then air space system which may be the global manager who has got the global picture of the air space and the flight planning system which would plane the paths of different flight. So, this way we can represent the top level view of an embedded system. So, once we start with top level view of the embedded system we should go in to the details.

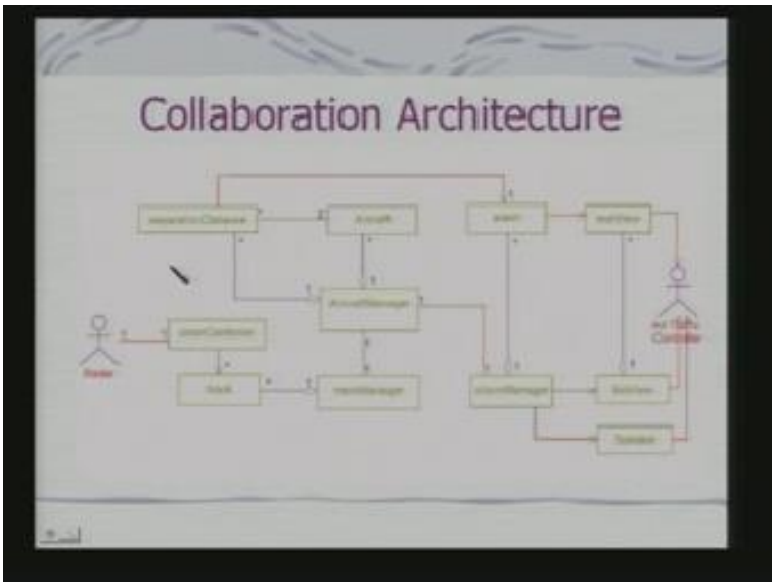
(Refer Slide Time: 04:07)



So, using the basic principle of top down refinement we can go in to the details. So, the details will give us what is called sub system architecture. So, sub system architecture indicates the different subsystems which actually form the overall system. So, here what we have seen is that aircraft space sub system is 1 subsystem. So, you have got radar subsystem antenna subsystem so on so forth subsystem. And what we have indicated here is that these are all stereo types; stereo type subsystem. And this 1 is to 1 this 1 at

the 2 ends of the link is indicating there is a class relationship of one class being linked with one of these classes. So, radar subsystem will provide the interface with the external radar for getting the data and managing the radar bending on the requirements. And this will provide the data to the name aircrafts space subsystem. So, you can have a number of such subsystems now what is the advantage of doing this? You can now, look at designing of these subsystems individually and look at their details.

(Refer Slide Time: 05:36)

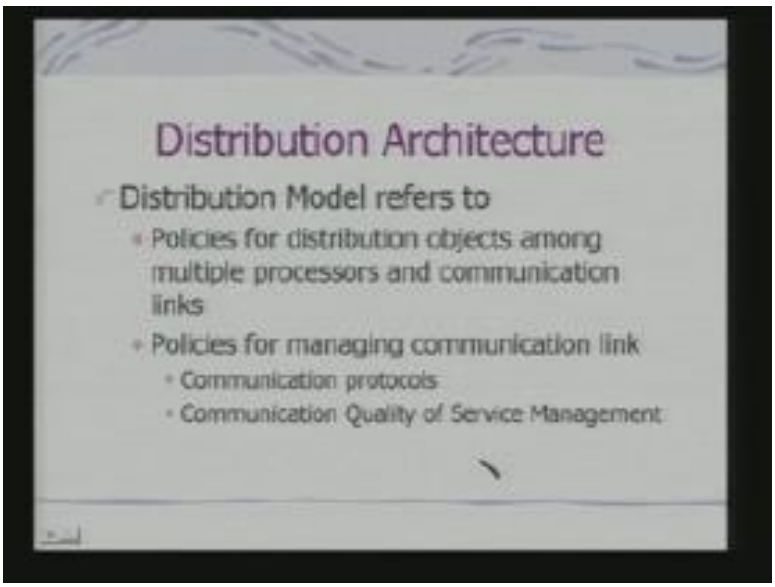


Related to this what is called collaboration architecture. In fact, collaboration architecture tells you how exactly the different components collaborate with each other. Say for example, if you look at the separation distance as a parameter. So, this separation distance is related to the 2 aircraft, because we shall be measuring the separation distance between the 2 aircrafts. And these objects if you consider these as an object or as a sensor or as a hardware element which is monitoring this distance. It will generate what and alarm if the distanced is less than a threshold. So, that is why the alarm object is related to this separation distance. So, this is representing the collaboration. And in fact, you will see that from alarm you actually have the view which is getting generated. And these are again actors that we have shown here. You have got a radar as an actor air traffic controller has an actor.

In fact, when you took a top level view of the system the requirements are the functionality decide by the actor actually specifies the functional requirement of the

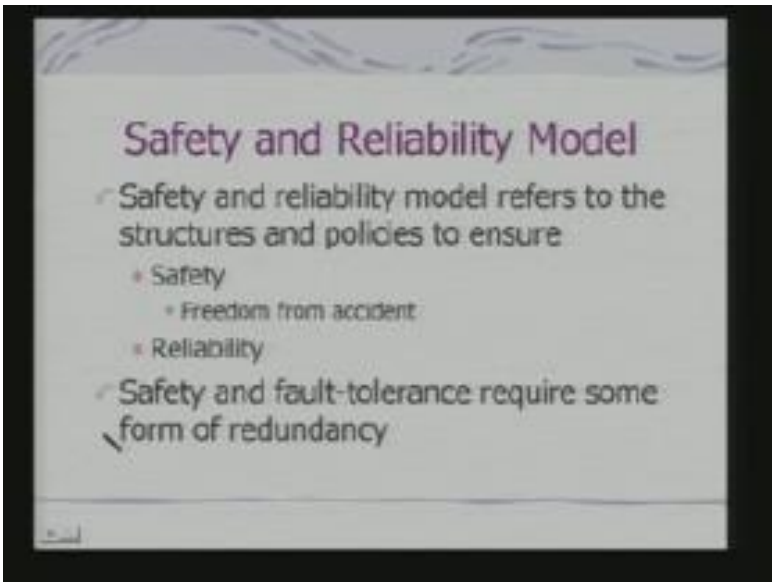
system. Depending on that functional requirement we had come to the subsystems. So, that subsystems can scatted to individual requirements and then we have linked up through this collaboration architecture these subsystems objects, because 1 subsystem cannot act independently. It has to make use of a other subsystems and that is why we say the subsystems collaborate with each other. So, this subsystem collaborates with each other and this collaboration architecture is represented through this kind of a diagram. So, here you find the radar. So, radar has got link with a radar controller that is linked with your tracker which tracks the aircrafts and then this is part of your track manager. Now, similarly there is a manager which is where alarm which is linked with. So, alarm will be a component used by the alarm manager.

(Refer Slide Time: 08:01)



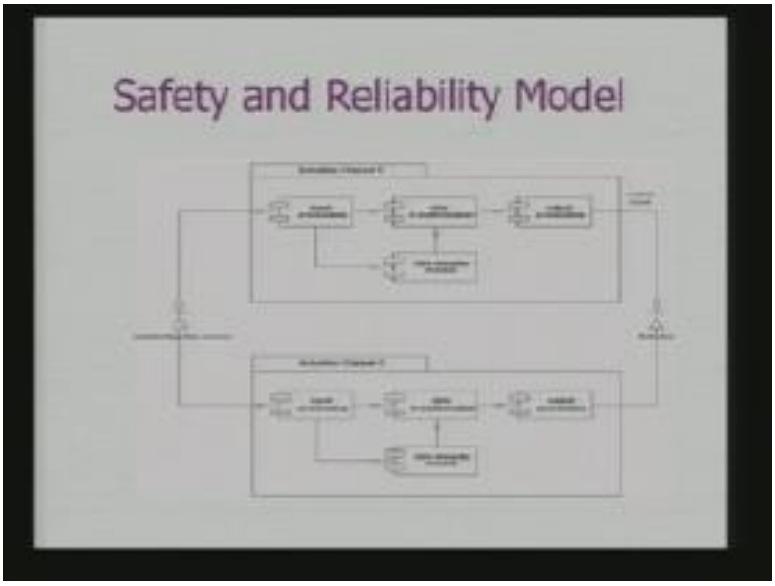
Next we come to distribution architecture. So, once we have got top level view with respect to the requirement and analysis. You have got the top level view from top level view we have got the subsystems we have got the collaboration view and next comes the distribution view. The distribution view tells us how the objects are to be distributed multiple processors and their communication links, because your embedded system need not consist of a single processor. Your air traffic system a complex system like a air traffic controller system will not be built around a single processor there may be multiple processing elements. And there would be policies for managing communication link the communication protocols communication quality of service management. So, all this things can be actually specified through your distribution architecture.

(Refer Slide Time: 08:55)



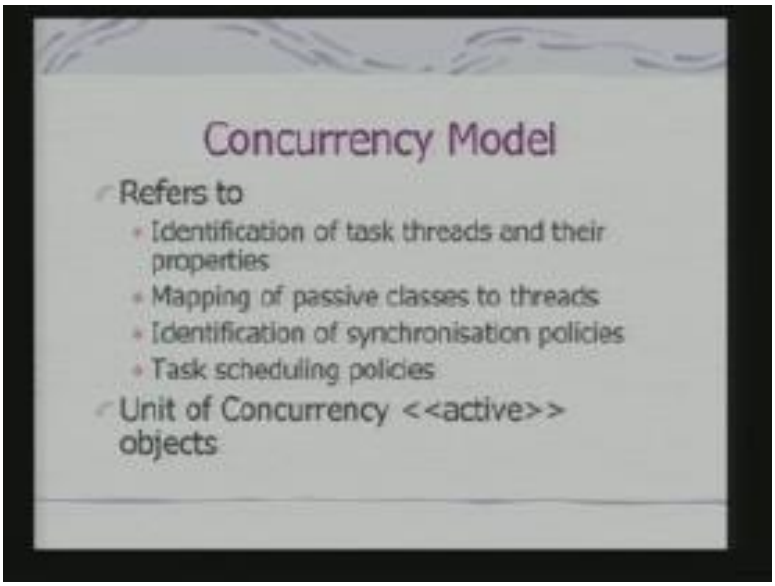
Then comes the safety and reliability model, because air traffic controller cannot permit air traffic control to fail. So, there has to be safety and reliability issue. So, that safety is freedom from accident and the reliability is the ability of the system to work without failure for a large period of time. And safety and fault tolerance require some form of redundancy. So, when we are designing for safety and fault tolerance our design specification should debit this redundancy.

(Refer Slide Time: 09:30)



So, here is an example. So, what we say this is an actuation channel 1 this is another actuation channel 2. In fact, you look in to this diagram this is representing an object and the object is consisting of is there actually components. In fact, this notation is for indicating what we called the components in UML notations. So, components can be piece of software components can be even a hardware realization can be event a kind of documentation if it is so, desired. So, here what we have showing is that different components input processing, data transformation, data integrating check, output processing all these put together form the actuation channel 1. And for the purpose of redundancy you have indicated that there would be another actuation channel. And they will both be connected to there I have got an actor. This is a monitoring data source the monitoring data source will feed both the actuation channels and this actuation channels will feed the actuator together. So, even if one of them fails other one can works. So, there is an inbuilt redundancy and fault tolerance in the system. So, the specification for redundance in fault tolerance can be provided through this kind of safety reliability model.

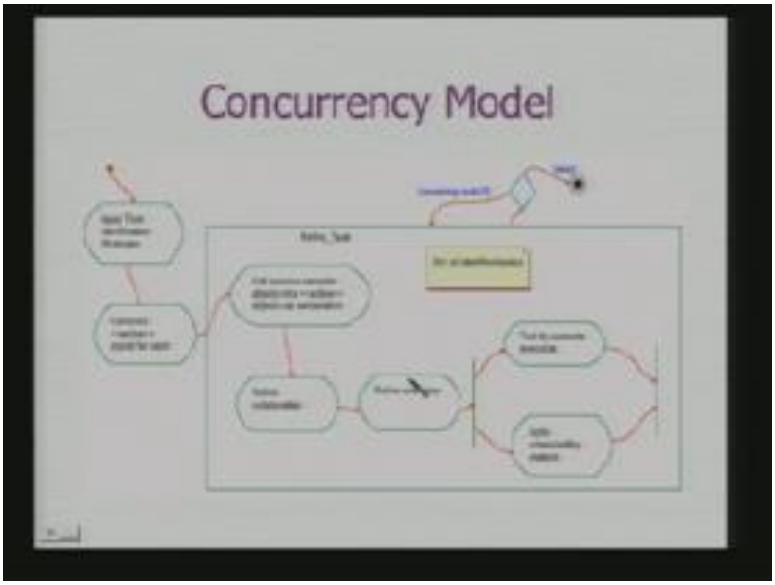
(Refer Slide Time: 10:59)



Next things come as the concurrency model. So, we have seen that we need to identify the concurrent tasks depending on external world and its requirements. So, concurrency model refers to identification of tasks threads and their properties. It tells you how to map passive classes to threads identification of synchronization policies task scheduling policies. And unit of concurrencies what we called active objects active objects are again

stereo type object. So, therefore, if you if you look in to it that when we are specifying the overall behavior of the systems what you said? You said we shall use a straight transition diagram to specify behavior of the system. But when there is a concurrency here would be what a concurrent set of state transition diagrams debiting the behavior of the system. Now, how therefore, you have looked at the FSM. FSM may represent one aspect of the system. And when we want to actually consider the concurrent processing aspects of the system we should use the concurrency model to debit that concurrency aspect. So, simple approach is that of using what we call an activity diagram.

(Refer Slide Time: 12:23)



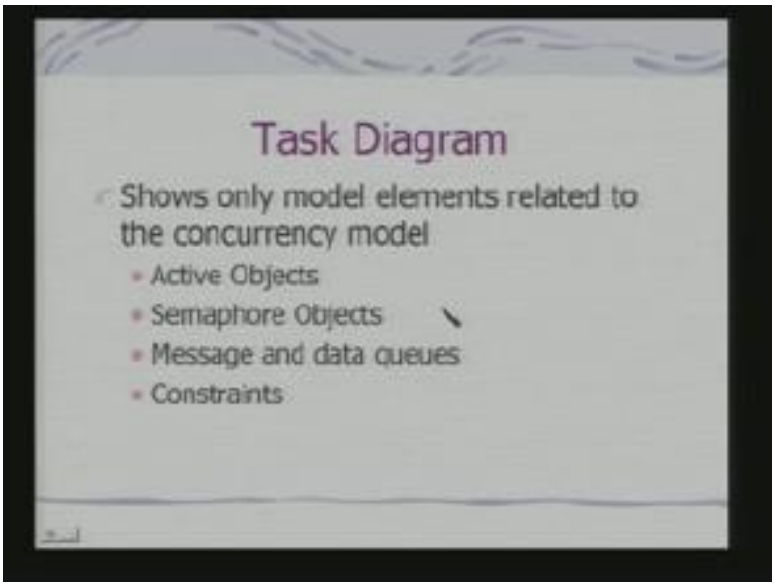
So, in fact, this activate diagram is similar to your in the loop. It similar to your state transition diagram, but it is not state transition diagram what it says? It actually gives you a set of steps frequencies steps in fact, if you see these diagrams tells you how to do the concurrency design. So, what it says? That applies task identification strategy so that means, once we have the requirements specification done. And we have got the top level view of the system with respect to the top level view of the system we should try to identify what are the different tasks? Then you construct active objects for each of the tasks. In fact, these active objects can also translate to the subsystems in the global system.

So, you construct active object for each. At passive semantics object in to active objects where composition, because why passive semantic objects? There may be components

which are to be controlled with in a task. So, there may be an LED indicator. So, an LED display or LCD display will be a passive object under the control of the task. So, I can put that tasks as part of you active object through composition that means it forms what? Basically a flit in that active object, because it is a component of the active object then you define your collaboration, because collaboration is collaboration between different objects. So, depending on how the objects have in mapped to active objects or threads. You will be modifying the collaboration then you refine scenario what does scenario? Scenario use as a situations which this concurrent tasks need to work and act. And what are really scenarios?

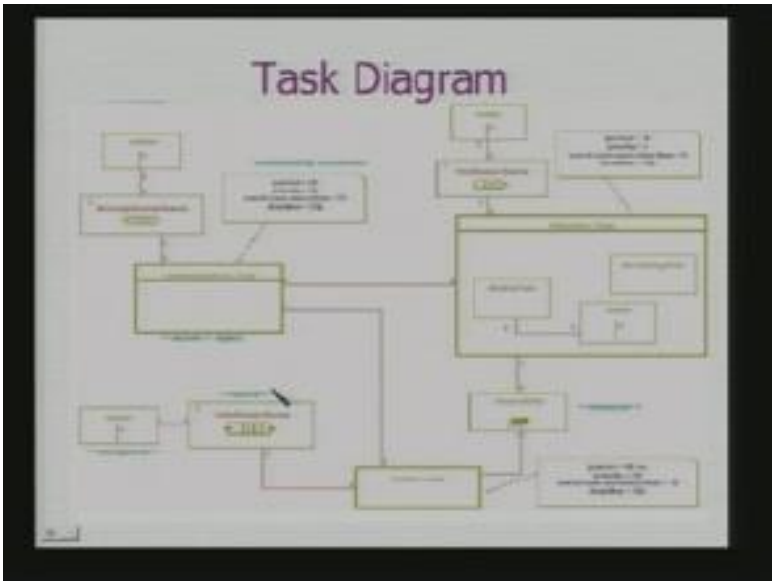
Scenario will again come from the requirement analysis, because requirement analysis tells you the different condition under which the system is expected to operate. And then you can have 2 parallel this is actually represents the concurrent activities. In fact, this activity diagram when there is a vertical line there are 2 branches from the vertical line this actually represents the concurrency concurrent task threads. So, test by scenario execution. So, once you have the scenarios you test each of the tasks a scenario execution and apply the schedulabilty analysis that is whether the task can be scheduled. If it is scheduled then what are the constraints the task should satisfy in fact, what kind of execution task time the task should meet. So, that will again dictate your choice of the processors. So, In fact, this is an activity diagram and this activity diagram through this kind of concurrent tasks actually debits the set of concurrent tasks the system should really handled.

(Refer Slide Time: 15:47)



In fact, the more detail depiction of concurrency comes via what is called task diagrams. Task diagrams shows only those model elements we shall related to the concurrency model. Active objects; so, active objects are objects associated with threads active threads for execution, semaphore objects message and data queues communication between these threads as well as the constraints. These constraints would be primarily schedule ability constraints. So, let us take an example.

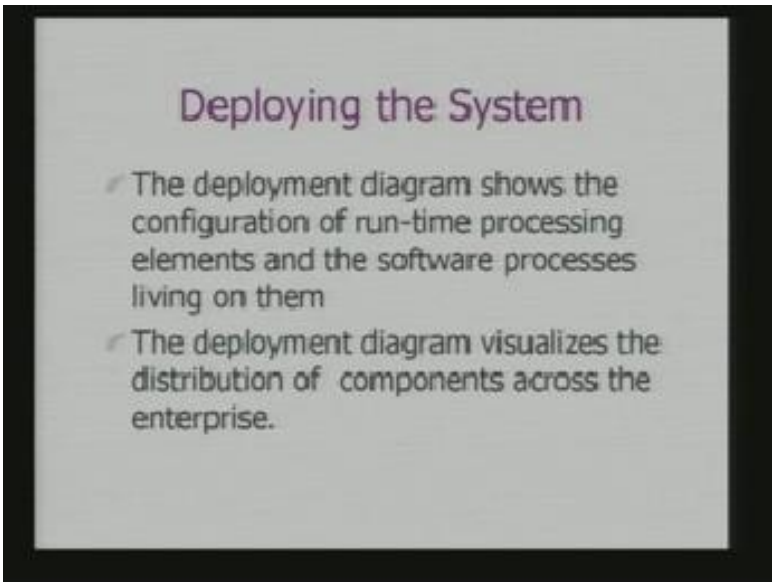
(Refer Slide Time: 16:22)



So, here if you see this is this task diagram is typically similar to that of your collaboration diagram. In fact, this is representing the different classes and it is also representing the links between the classes how they are linked? But along with it you will find there is interesting additional information which has been provided. Here you have actually provided the scheduleability constraints. The periodicity, because this is if this is active object have indicated by this stereo typing that this is an active object. So, since it is an active object the thread corresponding to the active object should satisfy the scheduleability constraints. So, you have specified what are the scheduleability constraints? The period has been specified worst case execution time has been specified and deadline has been specified. So, these details have to be represented so, that your specification becomes complete similarly, if you look at here what you have got?

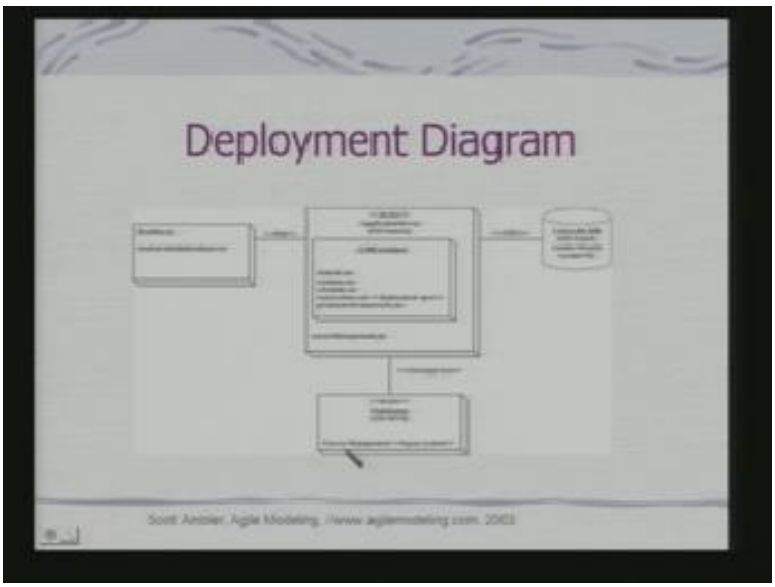
You have got a queue a data queue I have represented in data queue for the purpose of communication with other objects and a data queue. If it is a data queue which is be used for communication between multiple active objects that queue has to be synchronized. Because 2 processors cannot simultaneously access hat shared data object. So, you need to have synchronizing primitives like your semaphores. Semaphores would ensure that only 1 process can access the sheared memory. So, that semaphores has to be associated with this kind of sheared memories. So, here this is queue is a sheared memory and you have got a mewtex which is actually an actually belongs to this stereo type of semaphore objects mewtex is a semaphore objects and that is associated with this data queue. So, how this concurrent active object has to be managed in terms of their schedule ability constrains? How these active objects would communicate with each other via shared memory? And what kind of synchronization primitives to be used to manage access of the shared memory is debited to this task diagram.

(Refer Slide Time: 19:11)



So, next comes the deploying the system. So, the deployment diagram shows the configuration of run time processing element and the software processes living on them. And the deployment diagram visualizes the distribution of components across the enterprise if it is a distributed system it has to debit that depiction as well. So, if you look at that so, what you have got this is an example of a deployment diagram.

(Refer Slide Time: 19:39)



So, this is basically an object. So, this is indicated as a as your stereo type device so, on a device a particular set of objects have been mapped. So, this is basically yours server

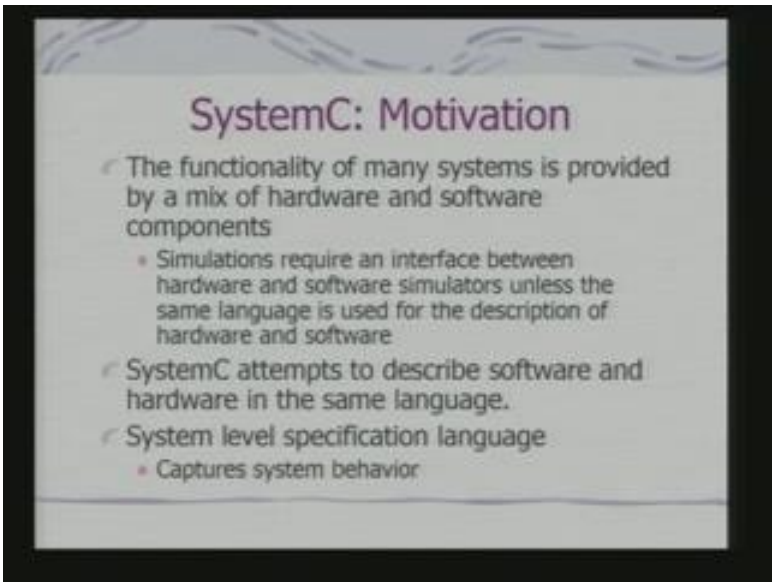
your database can be located at a remote point and connected via may be a JDBC call. And this is the main frame which may provide you the actual access to the system which can provide you the basic access to the systems fronted which is again a device. So, I am just showing a deployment diagram to illustrate how the different tasks get mapped? How different objects gets mapped? So, this is your basic applications server this can be your main frame for processing as well as access and this basically your database. So, this is the way which you can actually represent even a distributed embedded system. Although this application and example that I am showing here is not strictly an embedded system it is a kind of a general computational distributed computation application. So, what you can therefore, conclude by looking at UML?

UML provides with you a modeling tool by which you can provide the top level specification of the system. You can start the top level view on the basis of the requirements. The requirements come from the actors of the system. The system architecture of the system level view gives you the top level view of the system. You can identify the tasks corresponding to the system you can decompose the system in to subsystems. Subsystems can be further decomposed in to persuade object. Now, depending on the concurrency model these objects can be mapped on to active objects. You can specify the scheduleability constraints for the active objects you can specify how this object active object can shared memory. You can specify the dynamic behavior of the subsystems using straight transition diagrams. You can also use the sequence diagram to show how the messages can be passed between the objects for the purpose of realization of its behavior expected behavior.

So, in summary what we can say is that the UML provides a kind of a complete specification tool for specify the functional as well as component wise architecture of the system. Now, if we come from here and look at other kind of tools will find that there are other tolls which are targeted for various kinds of specific applications. In fact, when we say that we are using UML as a modeling tool formally we are not distinguishing between objects which may be physical hardware objects and software objects part say we can distinguish them subsequently. But when you are designing the system with may not distinguish them at the beginning itself. There can be objects of the systems; obviously, which will have part of it implemented in the hardware part of it in the software? Now, those objects if we think it in terms of their actual implementation of the

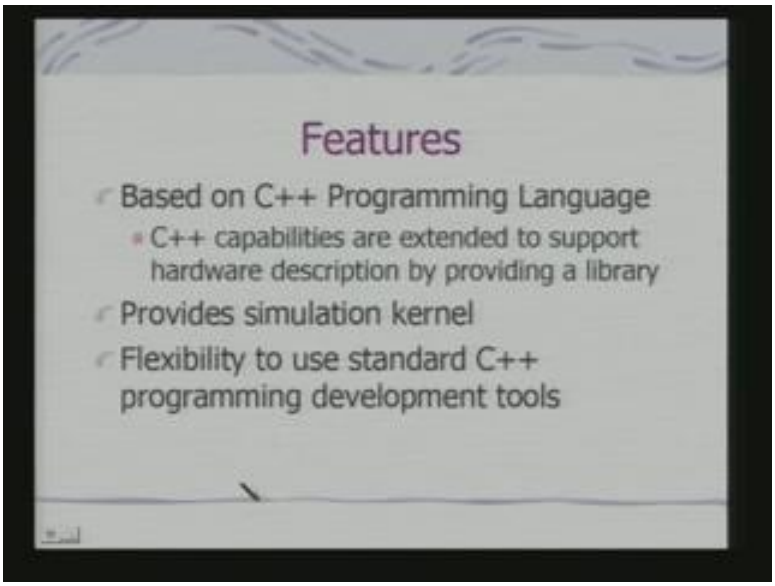
translations we would need different kinds of specification tools. In fact, if we look at even a standalone system for them as well we would like to have tools through which I can specify both hardware and the software aspect. In fact, system C is one such tool which had been developed around the language C plus.

(Refer Slide Time: 24:15)



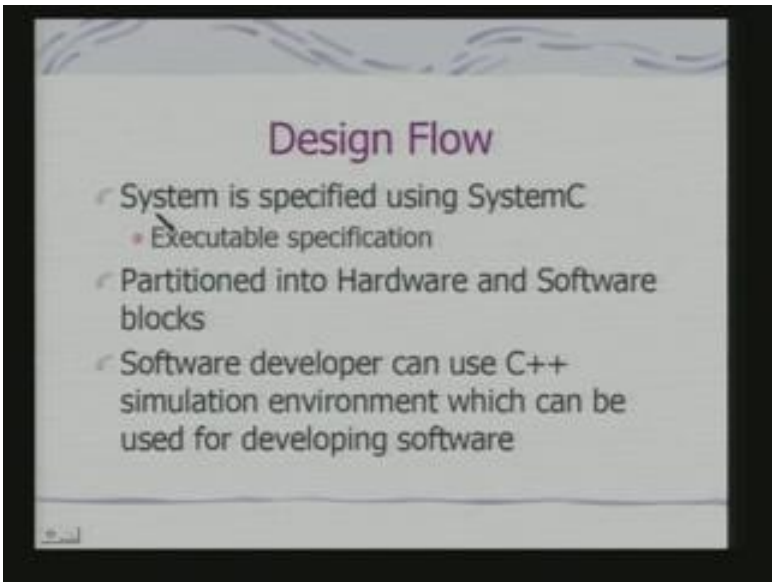
So, what we say the functionality of many systems of subsystems is a mix of hardware and software components. In fact, that is what we had seen if I specify a complete architecture using UML notation. And the simulation would require in interface between hardware and software simulators unless the same language is used for the description of hardware and software so, this becomes a constraints. And system c attempts to describe software and hardware in the same language. See if I have identified a subsystem or an object whose functionality are you like to describe. So, if I describe that functionality using the modeling tool like system C I can describe both software as well as hardware using the same language. In fact, that is the basic motivation for developing this tool called system C. So, system C in a way use a system level specification language and captures system behavior.

(Refer Slide Time: 25:30)



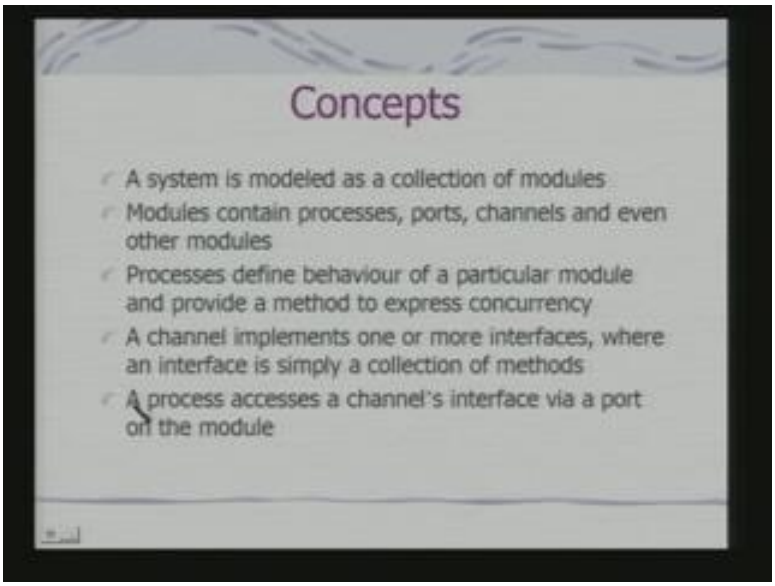
So, what are the features? It based on C++ programming language. So, it satisfies the advantages and the properties of object oriented design. So, C++ capabilities are extended to support hardware description by providing a library provides as part of the language a simulation kernel why this simulation kernel is required, because you have to simulate the hardware to execute the software. So, if you say that the first point we said that if we have hardware and software different then that simulation time interface becomes critical. So, if you have a simulation kernel then what we can do is we say we can do a co verification. That means, I can verify my software along with the hardware that I am designing. And it provides a flexibility to use standard C++ programming development tools. So, what is the design flowing that in that case?

(Refer Slide Time: 26:29)



The system is being specified using system C. So, system C provides executable specification. So, just thing in terms of your UML notation, in a UML notation you have identified a class you have identified the method you have identified its fields. Now, that class has to be actually implemented. And I can actually use may be system C to encode that class. And when I am encoding it I am encoding both hardware as well as software. And what I get I get an executable specification; because the system c code can actually be executed compile an executed. Compiling an execution of system code will mean what will actually mean execution of the software with the simulation kernel. But the simulation kernel is simulating the hardware. So, once we have the executable specification so, therefore, that gets partition into hardware and software blocks. And a software developer can use C++ simulation environment which can be used for developing as well as testing the software.

(Refer Slide Time: 28:00)

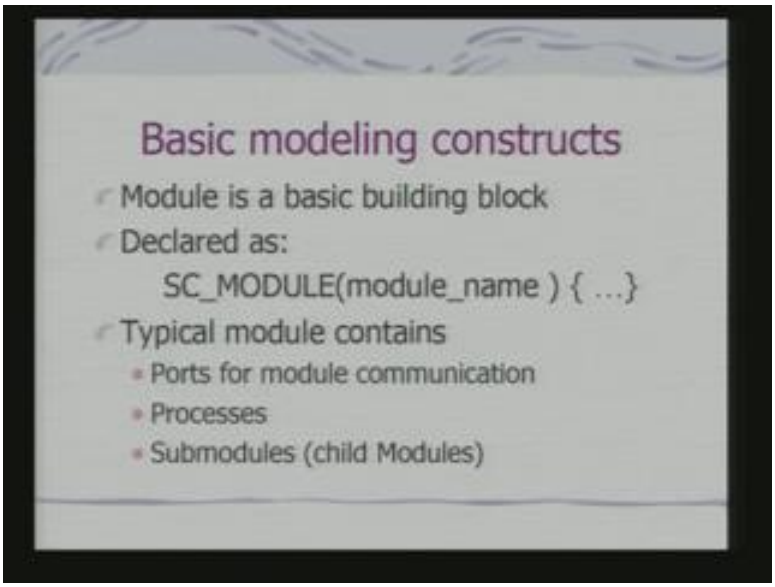


So, what are the basic concepts involved in system C? A system is modeled as a collection of modules. And modules contain processors, ports, channels and even other modules. Processors define behavior of a particular module and provide a method to express concurrency. A channel implements one or more interfaces where an interface is simply a collection of methods a process assesses a channels interface via port on the module. So, what is the implication of the description? So, I am modeling a system or a subsystem as a collection of modules. The modules contain process, ports, channels and even other modules, because it can be composed of sub modules and the processes are actually what ((refer time; 29:06)) processors provides me with the specification of what the module is going to do. It has got ports. Ports will be for communicating with other modules and a process defines behavior and provides a method to express concurrency.

In fact, the moment I express as concurrency now, this concurrency can be actually implemented in the hardware. So, this is the flexibility of the facility that this module provides you. A channel implements one or more interfaces where interface is simply a collection of methods. Just like any object when we have an object what is the interface of an object? It is basically its public view, because you actually use methods and this public method provides the interface to use the features of the object. So, a channel in this case of a module provides that interface. So, a module can be actually a hardware module and what we are talking about here if you can see that encapsulation of the entire hardware module through an object oriented parodied. A process access a channel

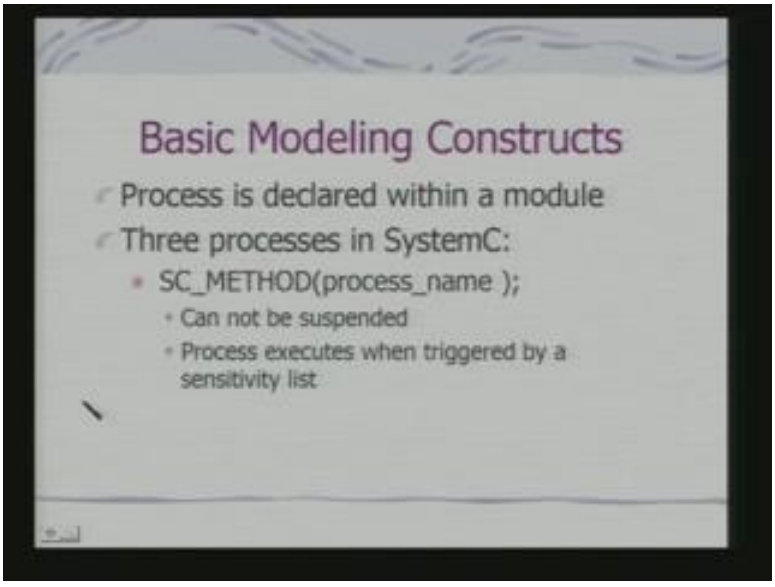
interface via a port on the module. Because process has to access the methods if it is collaborating with other modules it has use those methods. So, accesses channel interface via port on the module.

(Refer Slide Time: 30:49)



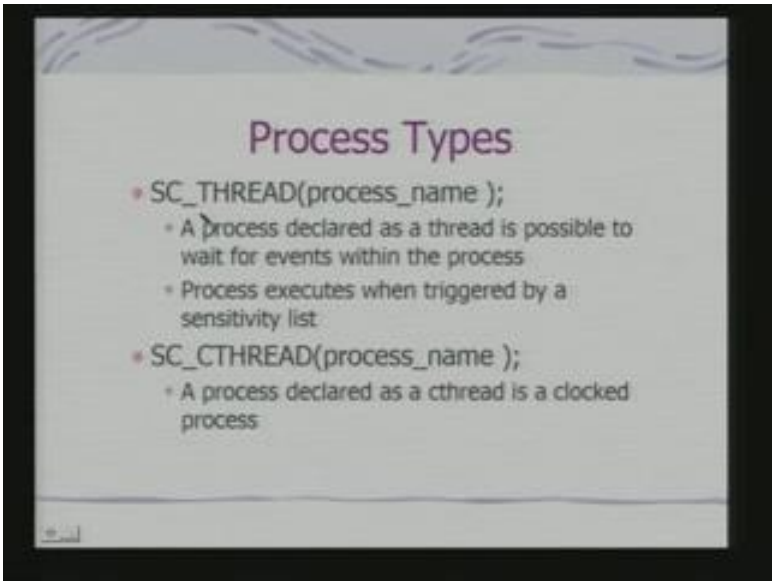
So, basic modeling constructs therefore, would be module. So, you need to have feature to define a module. So, a module is define this is a SC system C module and module has got a module name. A typical module as we have already shown talked about discuss will have ports for module communication processes as well as sub modules.

(Refer Slide Time: 31:17)



And a process is declared within a module and there can be 3 types of processes in system C. So, because processes if you see what is the interesting? I am talking about the method and method is a process name actually. So, process is declared is in the module and this process in a way if you want to mapping this processes mapped on to a pure software based object if you look at task a process is nothing but a method. But in case of a module I am calling them as a process a method as a process. So, if I declare a process as a SC METHOD what does it mean? This process cannot be suspended and the process executes when triggered by a sensitivity list. Sensitivity list means a there may be sensitivity list may translate to external signals to which the process is linked. That means, you can imagine that this can be an action which would take place when a particular signal is true or valid then you have got what is called thread.

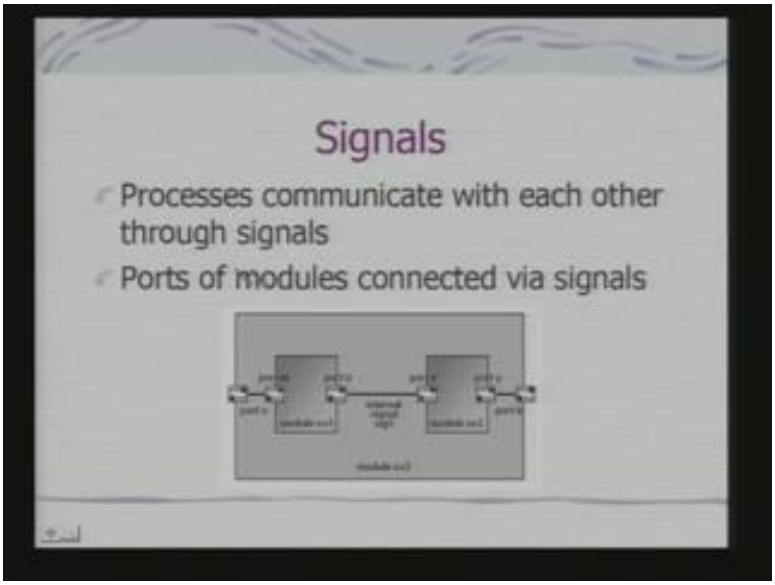
(Refer Slide Time: 32:34)



Thread process name provides you a process is declared as a thread when a process is declared as a thread is possible to wait for events within the process. If you remember the previous case what we said that the process can not be suspended. It is triggered by a sensitivity list once triggered its can not be suspended. But many other threads it is possible to wait for events within the process and process executes again when triggered by a sensitivity list. So, this aspect of its behavior is same as SC method that the other one that we have discussed then you have SC CTHREAD and a process name a process declared as a c thread is a clocked process. So; that means, you can understand therefore, what is happening here this is actually a process. Process means a set of actions. The set

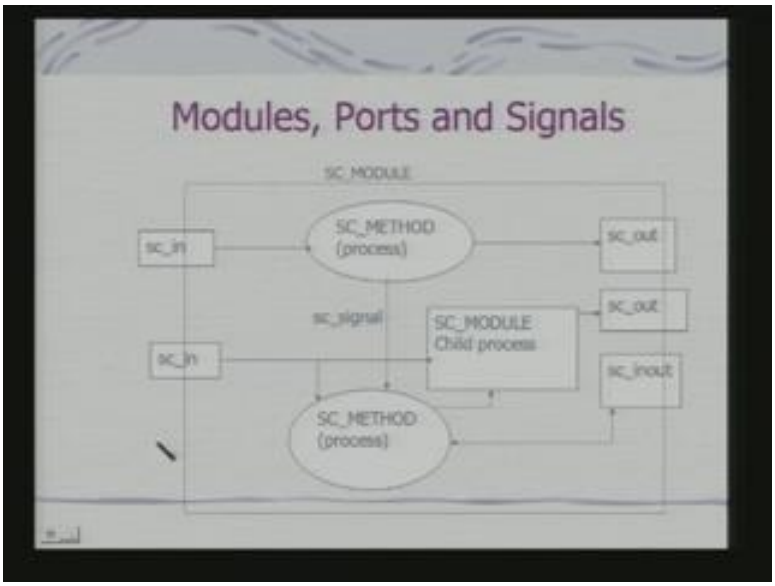
of actions is synchronized with respect to a clock. So, actually providing you a description of the hardware through this method and actually a module would be nothing but in a way a hardware with the ports providing the channels for communication. Please note the word channel that I have use not in the sense of system c that in a English generic sense.

(Refer Slide Time: 34:01)



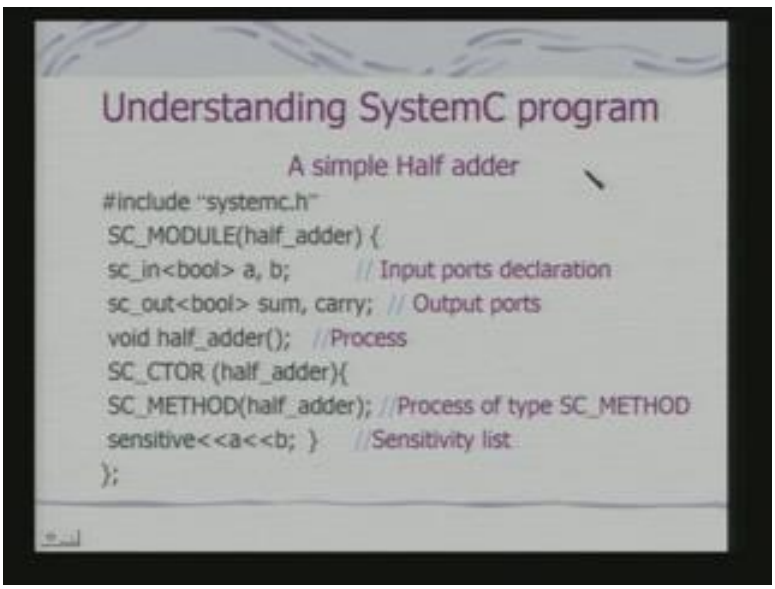
So, processes communicate with each other through signals. So, if I say ports of modules connected via signals. So, your module 1 this is your module 2 and these modules are to be connected they have to be connected via port. And there would be signals connecting the ports. And this are if you consider these part of a bigger module ex there which consist as part module ex 1 and ex 2 then these are the ports for communicating with the external world. So, you can see that how you are actually describing really hardware using an object oriented language along with that you can describe the software as well. So, this is the basic advantage of your system C.

(Refer Slide Time: 34:57)



Let us take an example this is a modules if I look at this module. So, this structurally what we have? We have this input signals, input ports, specified you have got this methods this are all process and this are the child process, because there is module which is part of this module. So, the method it can be a method or a thread process that is the process which is defined in this child module will be actually a child process. And these are the outputs which is going out these port is a inout port; that means, I can have an input as well as output. So, effectively what I have done I have got a specification of the hardware.

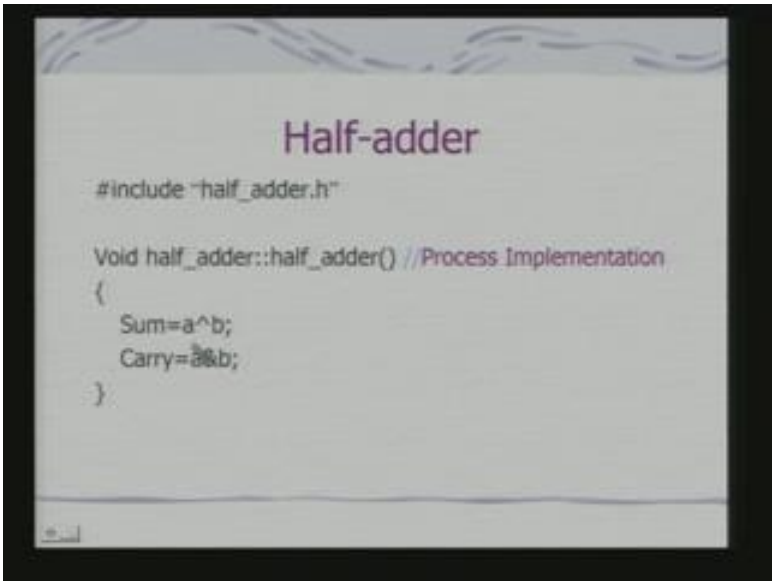
(Refer Slide Time: 35:54)



Now, let us take an example a simple half adder all of us familiar with the concept of a half adder. And what we are looking at here is basically very simple system c construct for describing the hardware. There complex contracts which we are not going in to we shall just look at the basic pictures of systems by which you can describe a hardware. So, here we are defining a module which is the half adder. Now, it has got this are input ports, Boolean ports let us say a and b then output ports there also Boolean sum and carry. And In fact, you have also support for a data types a programmer can use the data types.

And there are also hardware specific data types which can be used in a program. Now, you have got this process, because module has got a process so, that process is half adder. And what we are doing is you are declared in this SC methods half adder, because half adder is a combinational circuit this can not be suspended. And it is sensitive to a sensitive list would contain would contain 2 signals a and b which are you input signals. So, the moment I have got input signal a and b I shall be computing the addition that is your SC method half adder the process and that can be a suspended.

(Refer Slide Time: 37:35)

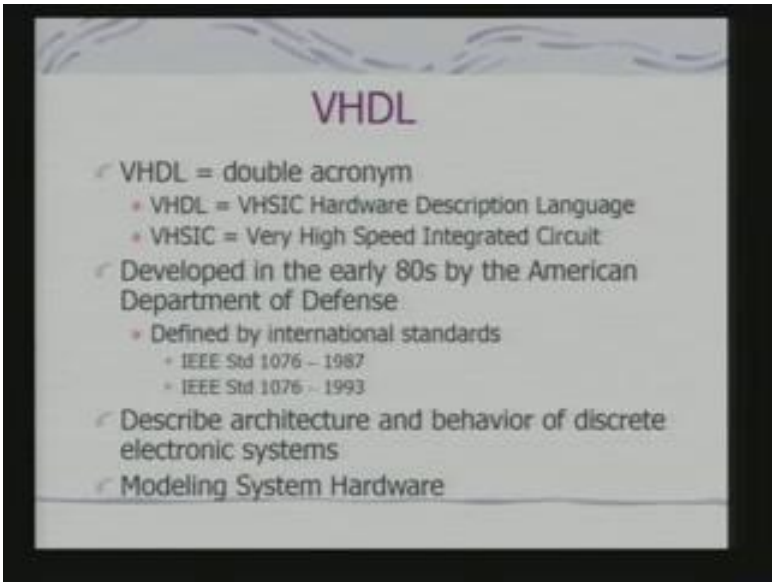


And if you look at here it is half adder the method is basically defining the logical function for some enquiry. So, I have actually specified the hardware and this is what we have referred to us process implementation. Now, In fact, if you look in to that the system C as I have started with the basic concept of system C is that for providing

specification language and executable specification language by which you can describe your software as well as that of your hardware. So, I am referring to this as a system specification language. Please keep in mind is that system C is not just a hardware description language.

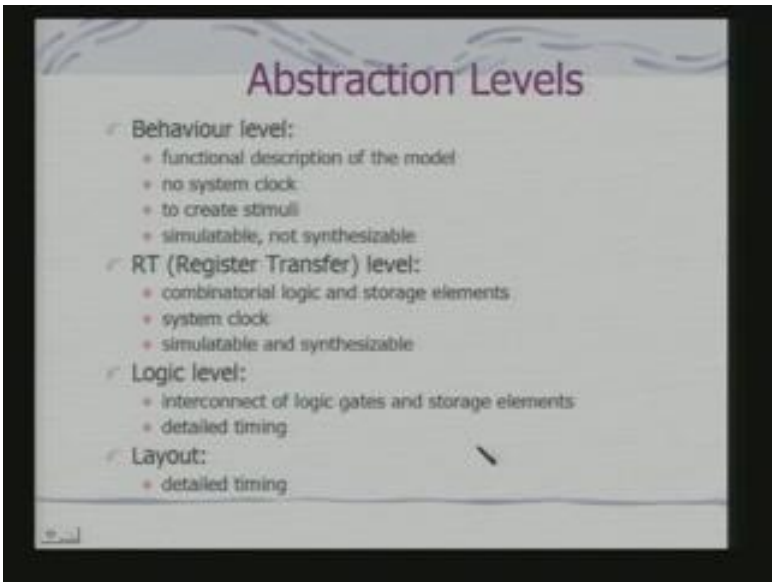
So, what would implies therefore, that if I am coming from a top level design if I identify and entity which can be translated to both hardware and software. I would like to have the hardware software partitioning describe the hardware as well as software using the same executable specification and then what do I do? I actually compile run. So, I have got a co verification and then I would go in to actual physical translation. That is actually building the hardware component and then trying to have the software may be running on it as the requirement may be. Now, we shall look at another language which is your VHDL which is again a modeling tool.

(Refer Slide Time: 39:23)



But how VHDL is different from system C? VHDL is purely a hardware description language. Now, what have try to provide you with a given the constraints of time the flavor of modeling tools which can work at different level of abstractions. UML is at the top level modeling tool then you have got the system C which is a both software hardware specification. Now, I am referring to VHDL which is the purely hardware description language. So, it describes architecture and behavior of discrete electronic systems and primarily we used for modeling system hardware.

(Refer Slide Time: 40:16)



And In fact, if we talk about the hardware there are multiple levels of abstractions with which you can look at hardware. At the top level what we call the behavioral level which is the functional description of the hardware module there may be no system clock. You want to create the basic stimuli and this is simulatable; that means, I can stimulate I have a behavioral simulation. But I cannot really synthesizable, because until and analyze the timing constraints are known. I cannot synthesize what does synthesize means translating this functional specification to actual hardware blocks in terms of gates and even at the lower level in terms of actual transistors and their implementation. So, next from the behavioral level we come to what is called register transfer level. At a register transfer level we specify the activating terms of combinational logic and storage elements we specify the system clock, because if I am having storage elements could be actually flip flops.

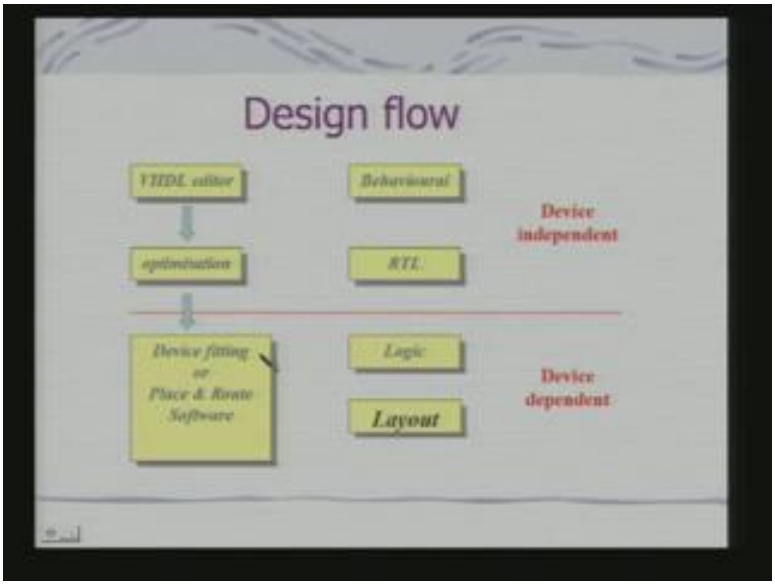
So, why it is register transfer, because I can specify behavior of the circuit in terms of data transfer between the different registers. And while the data transfer takes place the data gets modified a transform using your combinational circuit that is exactly how you have designed your digital circuits. If you think in terms of designing an FSM what does the designing of the FSM means? You identify the states, you identify the flip flop required for the realization of the states and then you require what the input signals. And you design the combinational circuit to provide the input signals that would cause the state transitions as well as you identify the output signals that is required for driving your

external circuit. So, you look at from that point of view your register level actually gives you a gate level or storage element level specification of the circuit and that is there; obviously, the clock is an important component.

So, the register level specification actually gives you a simulatable as well as synthesizable specification. Next it gives you a logic level where actually you say that how the gates are connected, because lets look at this level. You can understand very simply if I am talking about a register level I am talking about a register. Register will be actually implemented may be using a set of gates, because register is nothing, but a flip flop it has to be implemented using a set of gates. Gates; we have different timing depending on the technology that is being use for implementation of the gate. So, when we come to the logic level we actually provide the specification in terms of actual logic gates and the corresponding storage elements so, the detail timing can only come at a logic level. And the layout level it tells you physically how these transistors are put on to silicon what is there interconnect pattern and that will provide you to the absolute detail timing.

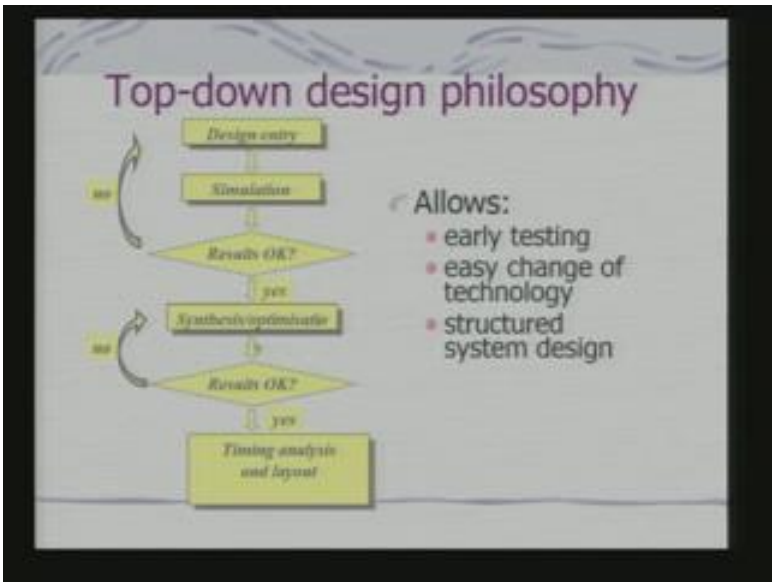
Now, why VHDL is important for a embedded system designer? Obviously, for the purpose of designing hardware at an appropriate level of an abstraction and also the same time you have the ability to use the core. Because today, if you remember when we discussed the processing elements we talked about arm, we talked about peak; we talked about other processors, as well. And we said that these processors are available in the form of may be soft core or hard core. If it is a soft core it can be a pure VHDL specification. So, when you want add functionality to that processor depending on your system design you add on a module to VHDL specification of the processor. So, you have got actually an enhanced hardware. So, if it is a absolute hardware code; that means, you may have actually the layout level design available with you. So, you had functionality to the system and may be design an ASIC or a ASIC as the case may be. So, design flow we say that VHDL editor goes to an optimization process.

(Refer Slide Time: 44:55)



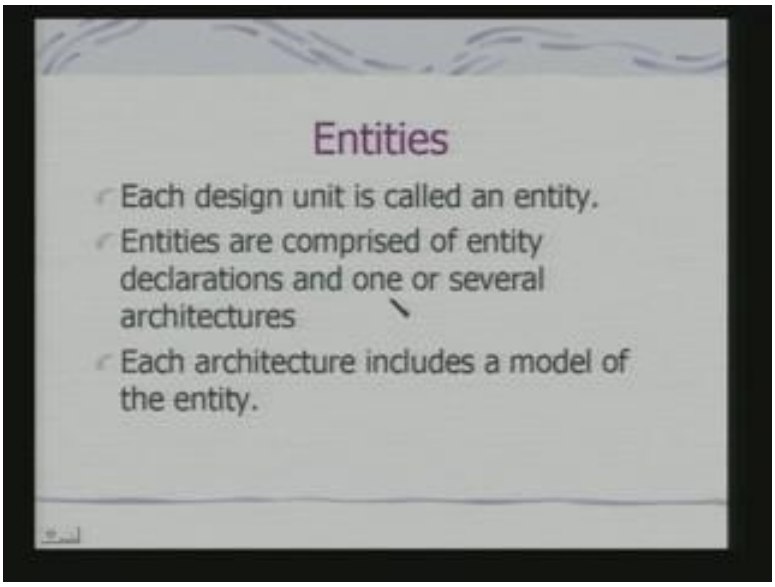
So, that gives your device or places an route software which would device fitting or place and route software which is the lowest level which is device dependent. So, when you talk about the VHDL editor and provide a description of the hardware in VHDL it is a behavioral description from where you can derive an RTL specification which is actually device independent. So, if you remember if say that I have got VHDL description of a processor core and it is a soft core, because it is actually independent of its implementation. And many of the designs of embedded systems built around this kind of soft cores being made use of.

(Refer side time 45:40)



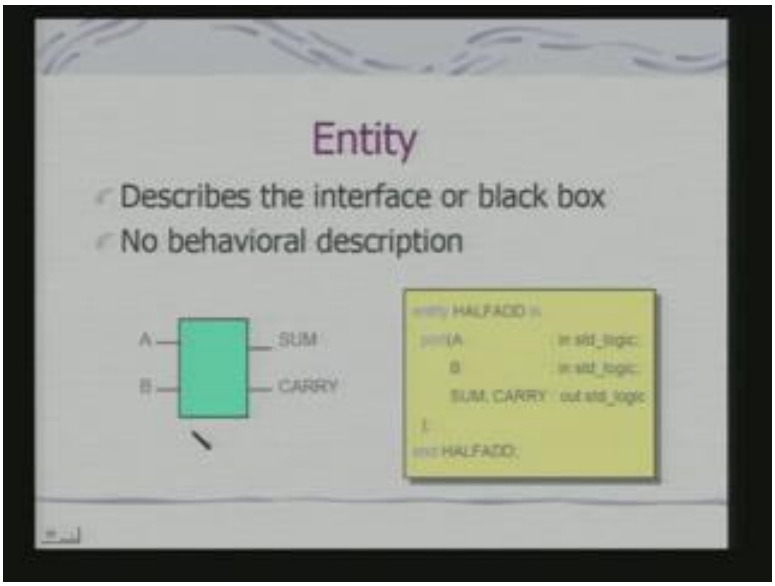
So, the design philosophy is you have a design entries simulate if it is else modify then you synthesize and optimize again do the simulation. And then do timing analysis and layout for coming to the final form so early testing easy change of technology and structured system design. So, what you see that before even coming to the device level which is coming here when we coming to layout we have we can do a simulation and testing of the system and do know its correctness.

(Refer Slide Time: 46:15)



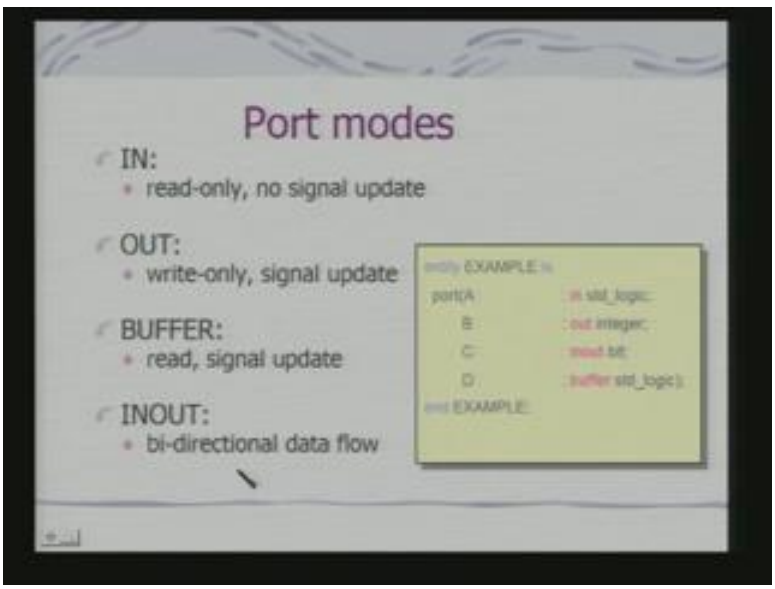
In VHDL each design unit is called an entity. So, the designs hardware designs specified as collection of entities. Entities are comprised of entity declaration and one or several architectures. And each architectures includes a model of the entity there can be various architectures.

(Refer Slide Time: 46:39)



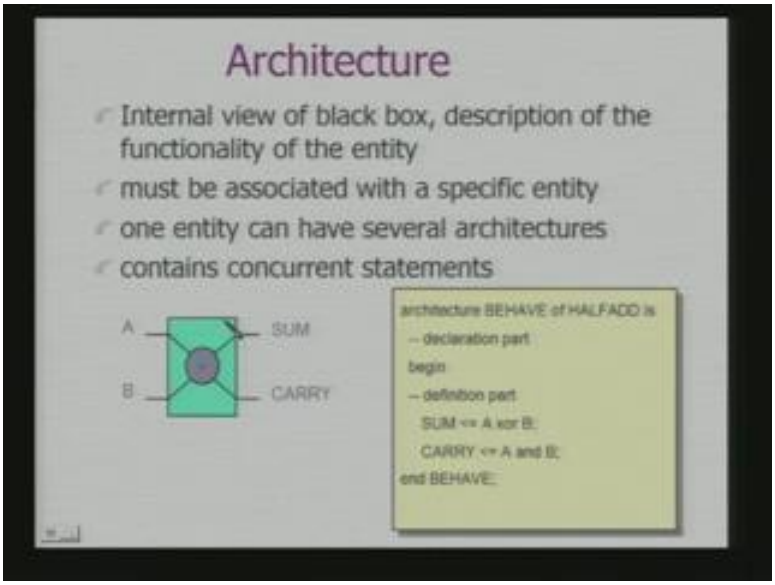
So, let us take an example. So, entity describes the interface or black box. So, again if you go back to the half adder example that we delt with in system C here also it is in a way similar. But when you look at this entity is telling you what? Simply the ports a b these are in and standard logic this is the standard logic, because VHDL can support multi valued logic as per the standard. So, you this is all standard logic some in carry there out there also standards logic signals and these is an specification of this entity. And this is the black box, because we have not included what will be done inside black box in this description.

(Refer Slide Time: 47:26)



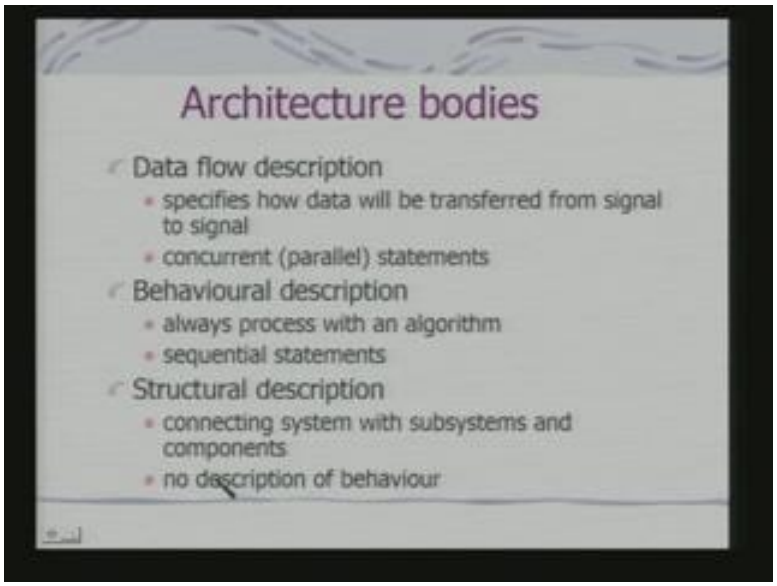
Then you have got in the black box what we have used we have use port. So, ports can be in out buffer inout. So, it is an input no signal update out is the right only. So, there has to be a signal update. Buffer is read, but signal update; that means, buffer why signal update, because the signal would change when its gets updated and inout is the bi directional data flow. So, these are the ports which are to be provided with an entity. So, it can be connected with other entities. And when a port gets connected with other entity the connectivity should be consistent in terms of the nature of the port then you come to what is called architecture.

(Refer Slide Time: 48:15)



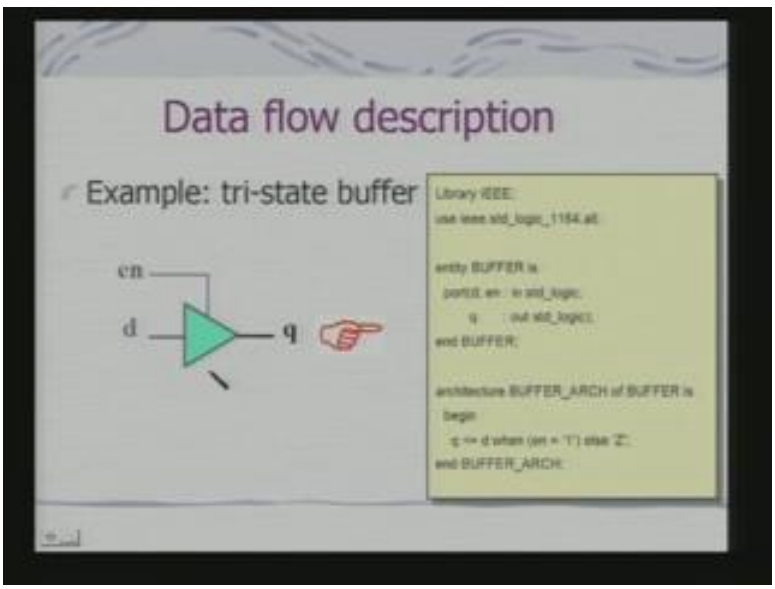
Architecture describes what is inside this black box. So, internal view of the black box description of the functionality of the entities what the architecture provides for. It must be associated with a specific entity one entity can have several architectures and contains concurrent statements. This concurrency can be because of the hardware as well. So, here if you look in to it this is an example of this adder and this is architecture behave. In fact, this is behavioral prescription of the architecture what it sells? It defines your sum defines your carry and that is of end of behavior part of the architecture. I said there can be various architectures this is the behavioral description of the architecture.

(Refer Slide Time: 49:04)



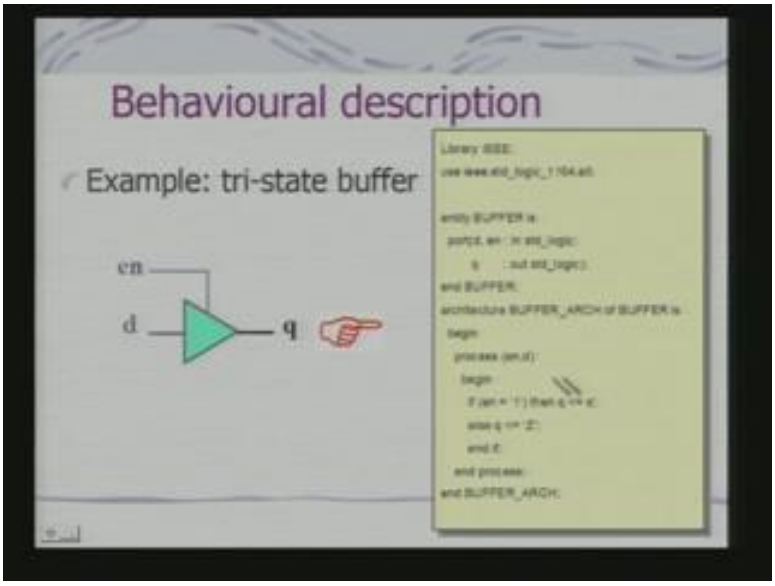
So, architecture bodies can have data flow description, behavioral description, structural description. The data flow description specifies how data will be transfer from signal to signal. And it can actually have concurrent statements, because they can be hardware operations executed concurrently. Behavioral description is typically an algorithm, because you can use simple sequential code to simulate the behavior of a entity. Structural description provides a connectivity between the components and the subsystems it does not really have a description of the behavior. Let us take an example it is a very simple try state buffer.

(Refer Slide Time: 49:53)



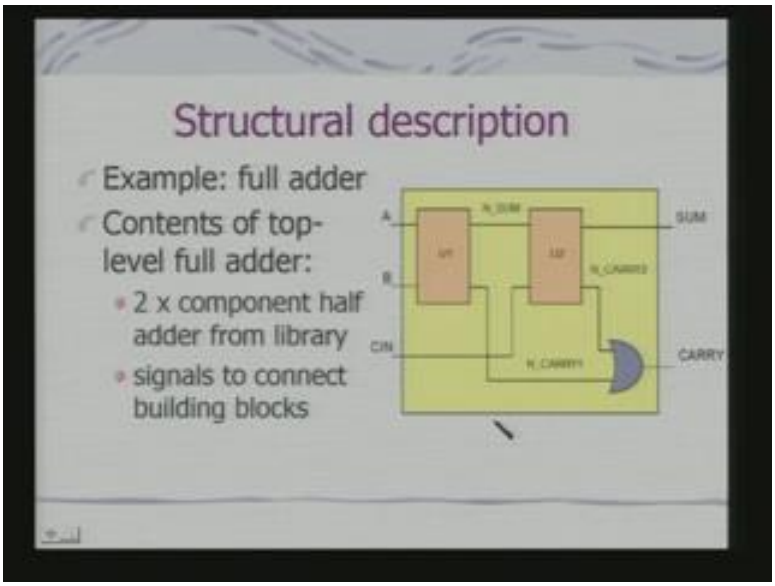
So, what we have describing here a data flow description. So, this is the entity which was already defined earlier and the data flow description is if you see what it says queue this is the signal assignment. That d signal is the sign to Q when n that is enable signal is 1 else z tri state. So, if you look at when means why it is a data flow? Model, because it tells you that when in enable if you consider when in a literal sense it is associated with the concept of a time when this signal becomes 1 then only that q should be what representing d otherwise it would be set. Since a data flow model it is not just a behavioral model. So, lets look at a behavioral model in a behavioral model what we are telling?

(Refer Slide Time: 50:51)



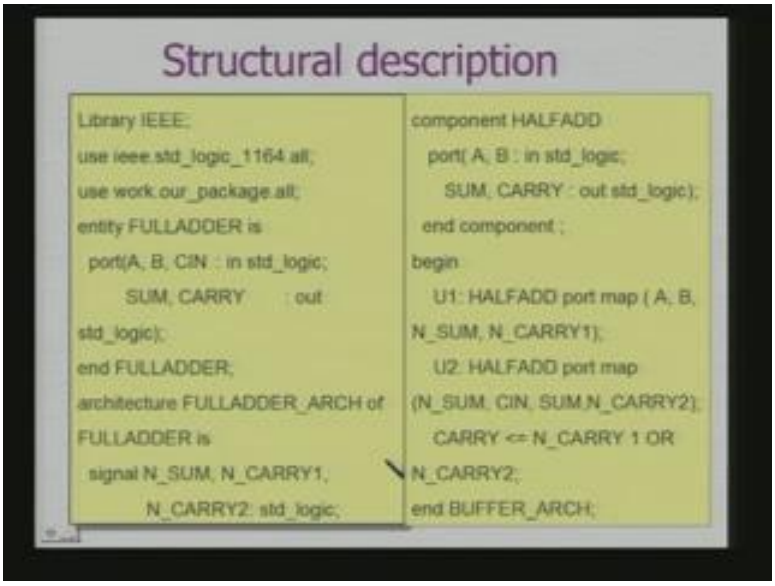
For the same tri state buffer we say that if this is true then this will be true. So, this is the adjust specifying the conditions that not telling then when this is true that this has to depict the value of t. Please note the difference between the data flow model and the behavioral model I said it is more like an algorithmic description. So it can be run as a sequential code to do a behavioral simulation.

(Refer Slide Time: 51:20)



Structural description basically tells you the structure. So, here what we have got is a 2 half adders can be put together to make a full adder. So, there are we have got component of top level full adder and 2 component half adder from the library and I can use the signals to connect them.

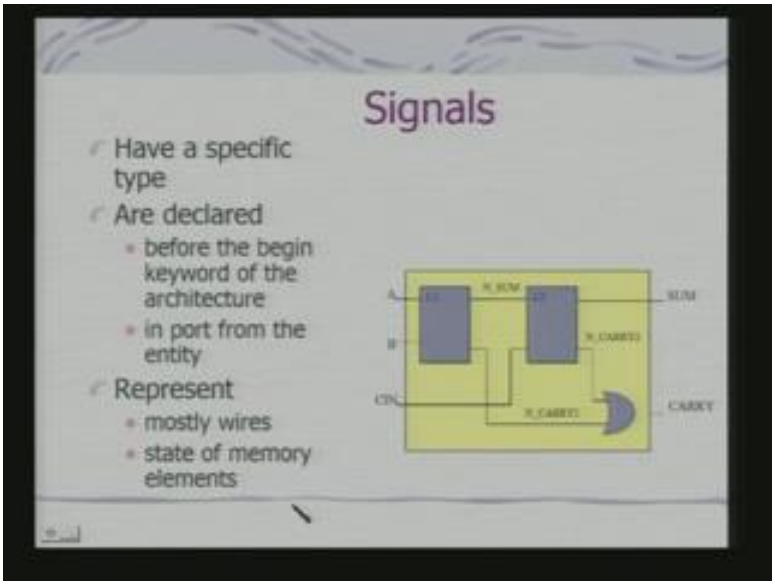
(Refer Slide Time: 51:39)



And the description would be like this. If you look at it here I am describing the entity full adder and architecture then I am describing this architecture which as got the signals and the component is half adder with its port which is a b input sum carry as output. So,

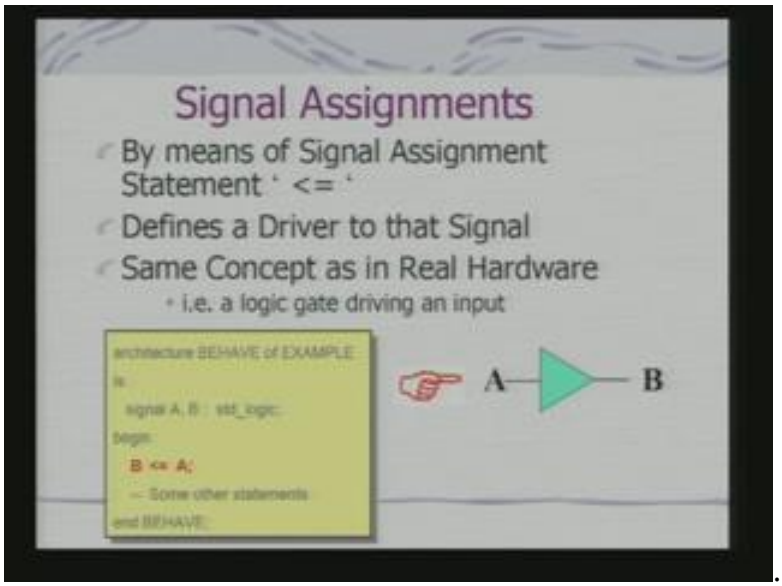
it gives you the description of this full adder in terms of components. And then the architecture actual architecture this is the definition part architecture is U 1 consisting of U 1 and U 2 which does a port map. And it is now, defining the carry so, this is completely defining my full adder as a connectivity between the 2 half adders. So, here if you see that half adder this half adder we have port mapping. So, what we it is doing a b and I have got n sum n carry and this C in and n carry 2. So, this are doing a port mapping, because why this port mapping is required this port mapping is required. Because if you look in to here this is the connectivity that I want to represent the C in goes to here and this output here should come at this point. So, this is what the structural description is all about.

(Refer Slide Time: 53:07)



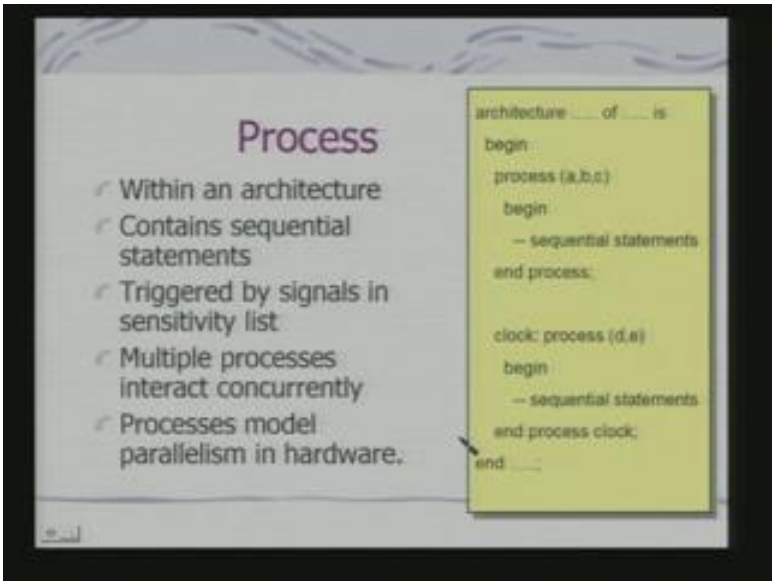
Signals are specific type and they provide the inter connect between the different elements. And this inter connect tells you that how the signals have to flow.

(Refer Slide Time: 53:23)



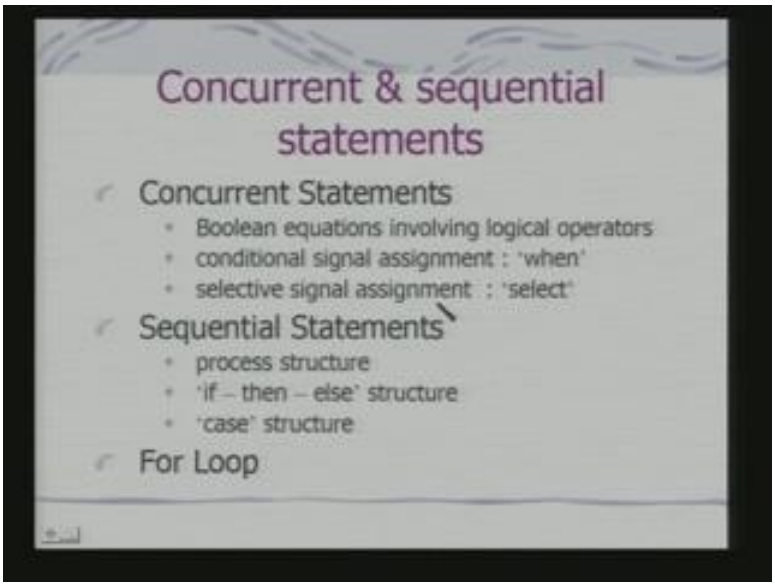
In fact, the signal assignment through this symbol which we have already discussed and this tells you the p is assign A is assign to B.

(Refer Slide Time: 53:34)



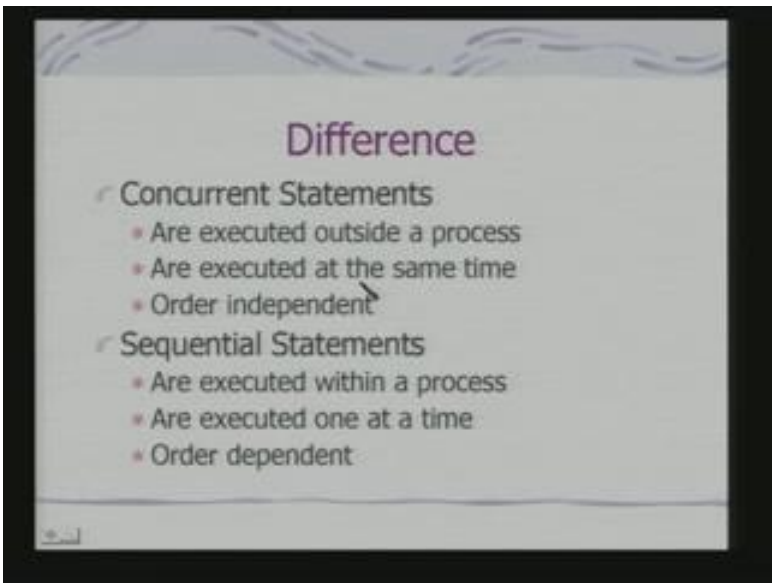
Now, process actually gives you what? Within an architecture contains sequential statements, triggered by signals in sensitivity list. This is very similar to system C and multiple processor can interact concurrently. So, this is a set of sequential statements and the sensitivity list comes from here. So, there would be concurrent sequential statements.

(Refer Slide Time: 53:59)



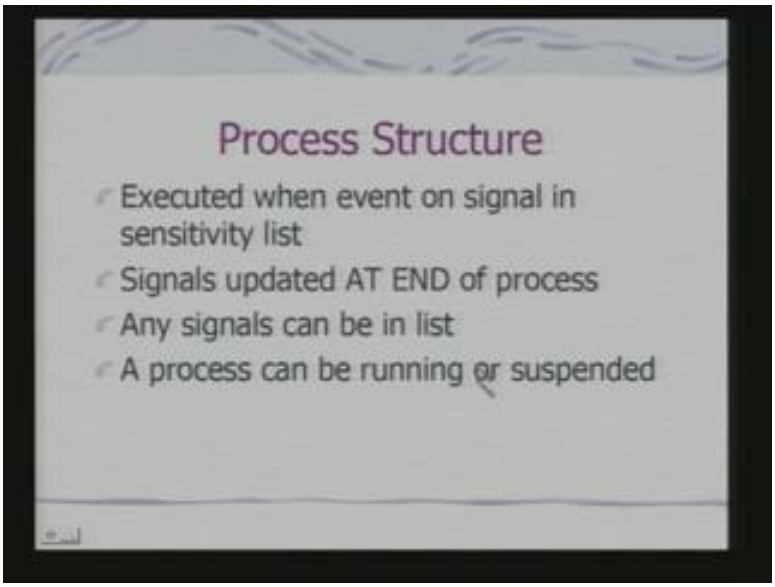
Boolean equations involving logical operators would be part of concurrent statements say can takes place in parallel. Conditional signal assignment that is when that is when that occurs is in parallel and selective signal assignment that select if there are multiple signals and select. So, these signals arrive in a concurrent fashion so, that is why it is a concurrent statements. Sequential statements provides a process structure we have already seen if them else structure for defining the behavior you can have case structure there also for loops to do a processing in an creative fashion.

(Refer Slide Time: 54:37)



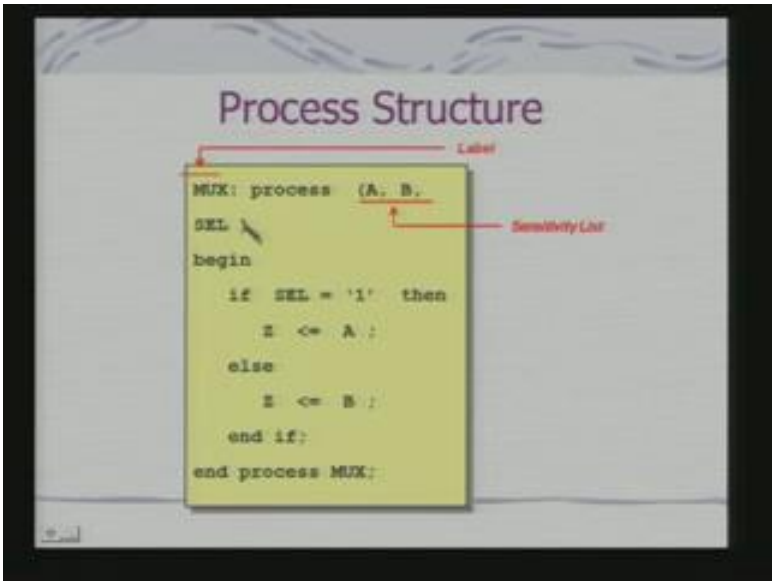
Concurrent statements; that executed all at the same time and outside the process. Sequential statements can be executed with in the process and they are; obviously, order independent.

(Refer Slide Time: 54:49)



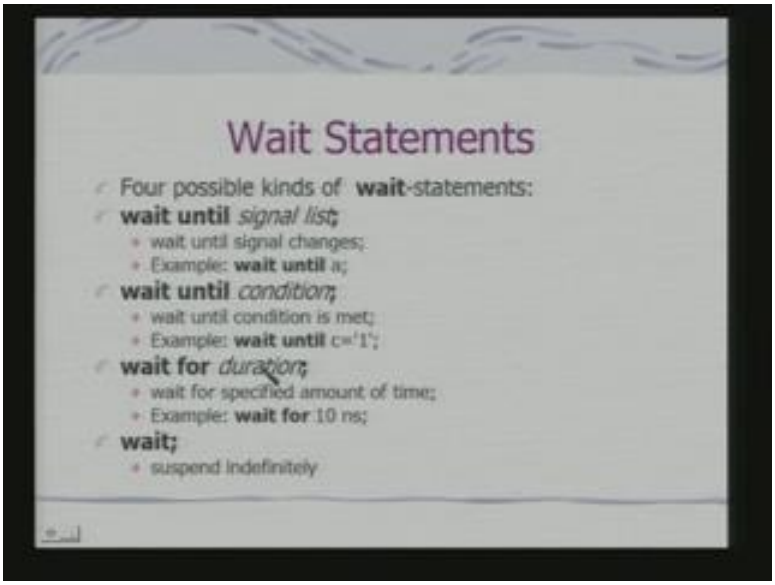
The process structure is process is executed when event on signal in sensitivity list is their and signals are updated at end of your process. Any signals can be in list and process can be running or suspended depending on the case may be. So, if you look in to here this is process structure.

(Refer Slide Time: 55:08)



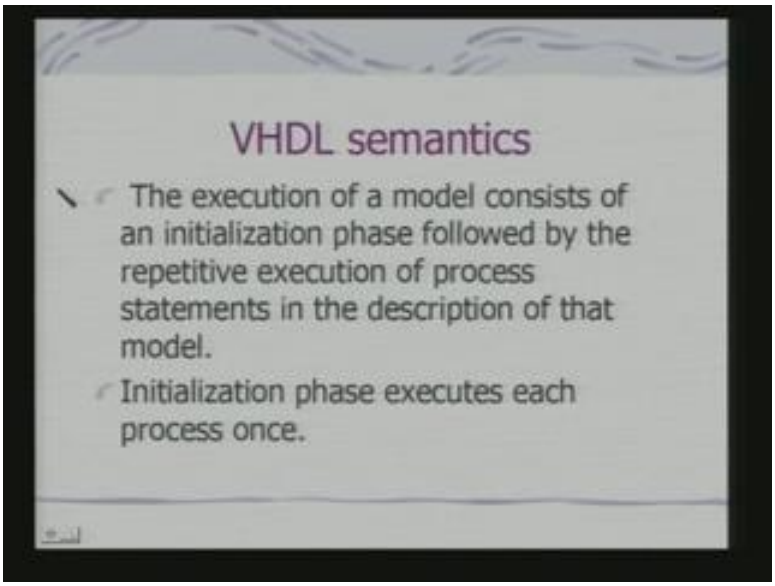
This is the label this max is the label. So, process is part of the architecture. And this a b are the sensitivity list this is the signals and this is remember if this is same what I am doing? I am basically designing a multiplexes. So, select is 1 then out z is a else z is equal to b. So, I am defining this behavior through this process structure. There are wait statements.

(Refer Slide Time: 55:39)



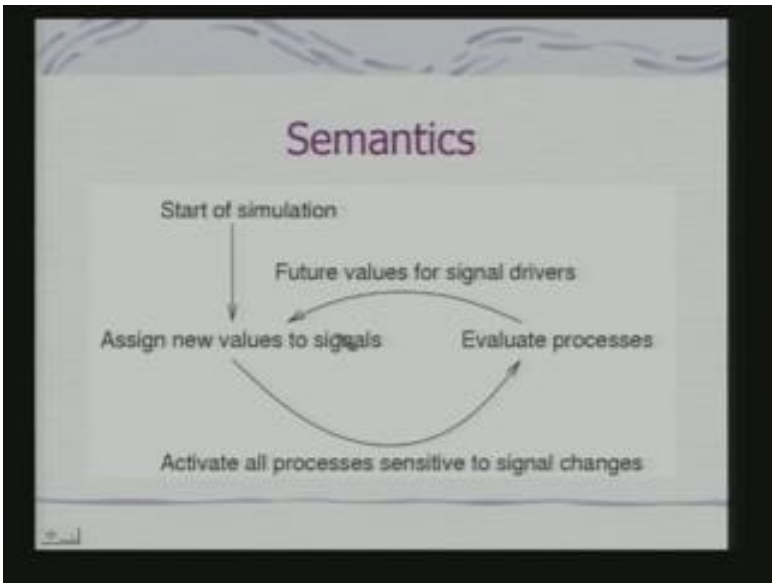
So, wait statements for quardination with the signals there can be waiting until a signal list. So, there can be waiting until the signal arrives, wait until a condition occurs there can be waiting for specific durations. So; that means, you can set a timer and then when that timer expires then actually the action takes place and wait can spend in definitely. So, this wait can be put in the process statement itself and specify the different timing quardination between the processors what is the semantics? Because once you have specified the model the semantics actually it is how it will actually simulate.

(Refer Slide Time: 56:20)



So, execution of a model consists of an initialization phase followed by the repetitive execution of process statements in the description of that model. And initialization phase executes each process once.

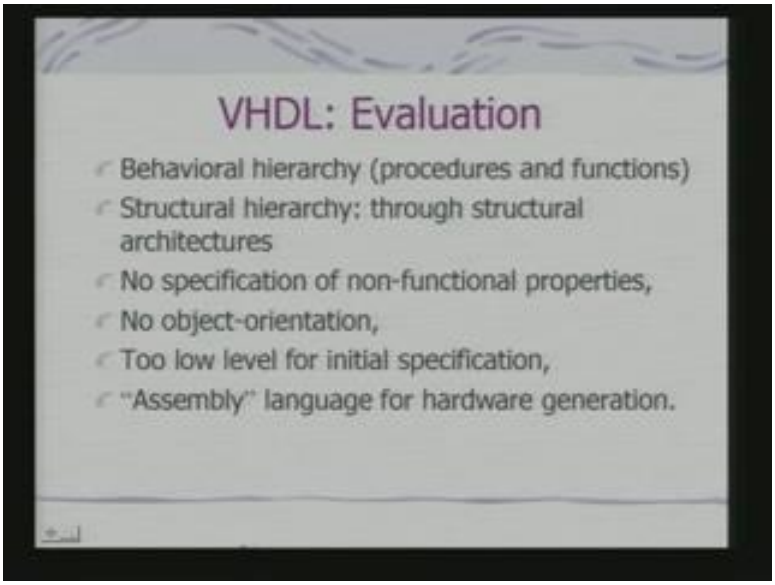
(Refer Slide Time: 56:35)



So, basically this is the basic semantics you start a simulation assign new values to the signals, evaluate the processes. The processes will now provide the future values for the signals, because there will be connected. So, that value comes to the signals and that we to simulate and activate all processes sensitive to signal changes. So, this way when you

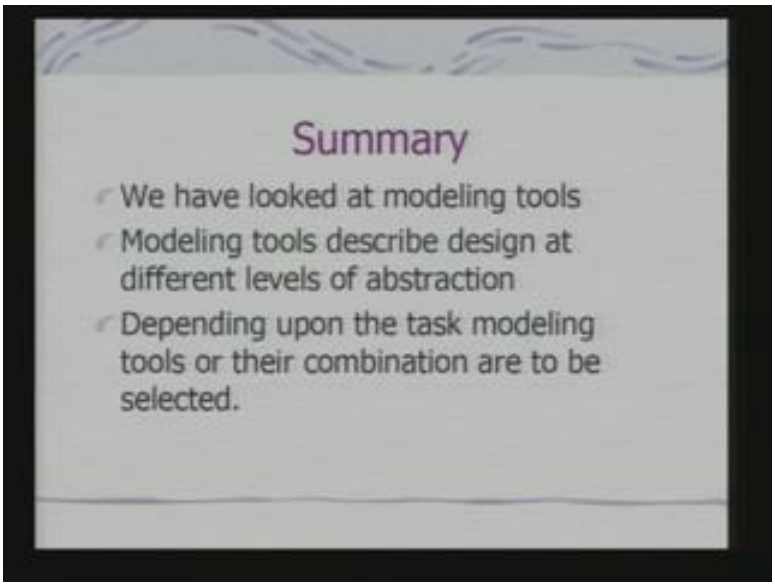
simulate you can actually verify when there your hardware is giving you the desirable behavior or not. Obviously, there had to be more details to be looked at when you talk in terms of cycle accurate simulation; that means, your timing conditions are taken care of accurately. So, but through this simulation you can actually verify your hardware. So, if you look at an overall evaluation of VHDL.

(Refer Slide Time: 57:32)



It provides a behavioral hierarchy in terms of procedures and functions provides structural hierarchy. There is no specification of non functional properties, no object orientation. It is too low level for initial specification in actuality it is nothing but an assembly language for hardware generation. So, if we go through the complete change what we find the top level specification comes from ((refer time: 57:59)) system gives you a combination of hardware and software wage due gives you an mechanism to specify completely your hardware. So, if you come to the summary we have looked at the different modeling tools which can be used for defining embedded system.

(Refer Slide Time: 58:14)



This modeling tools describe design at different levels of abstraction and depending up on the task modeling tools or their combination are to be selected. And now, in the next few classes, we shall look at this design task can how this modeling tools are to be used as part of this design tasks we shall look at that.