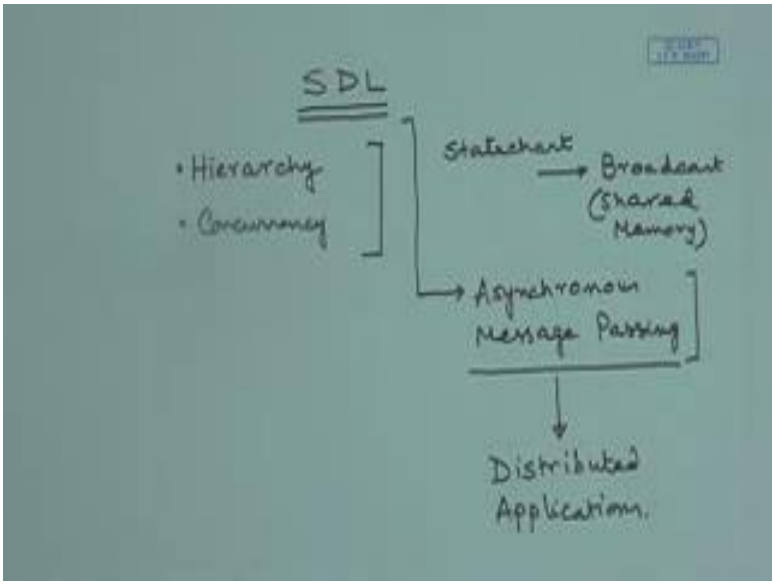
Embedded Systems Design
Prof. Anupam Basu
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 37
SDL

So in the earlier class we have looked at, state chart and program state machines right; a version of program state machine is pictured. Now we will still be in the paradigm of finite state machine as the presentation. All these state chart and spec charts were hierarchical communicating finite state machines.

(Refer Slide Time: 00:45)

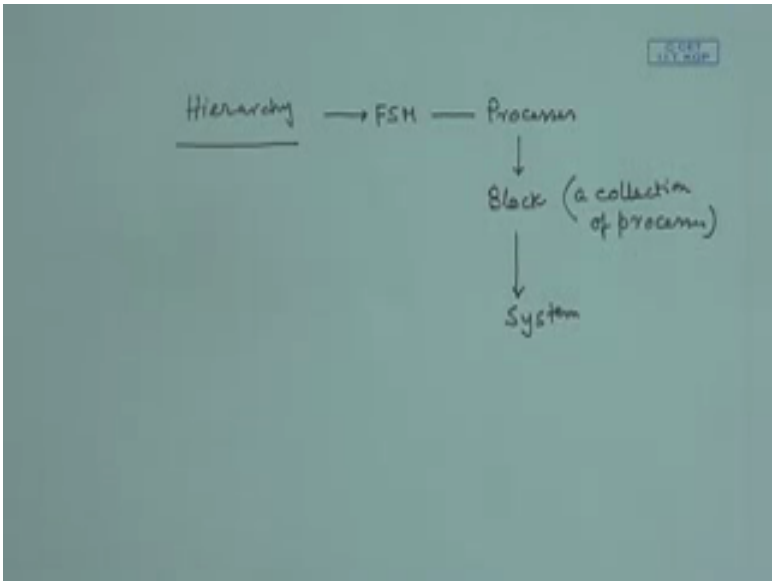


Today we will discuss about SDL system and description language. It is another example of state based representation. It has got our requirement satisfied; that is hierarchy is satisfied in this, concurrency is also satisfied in this. The major difference from state chart is that, in state chart, our mood of communication was broadcast or shared memory right, broadcast or shared memory.

In SDL, we will have asynchronous message passing. You know that there can be two types of communication; one is message passing, another is shared memory. So, here we take the course to, asynchronous message passing. SDL was initially developed for telecommunication applications historically, but then it has migrated to other embedded

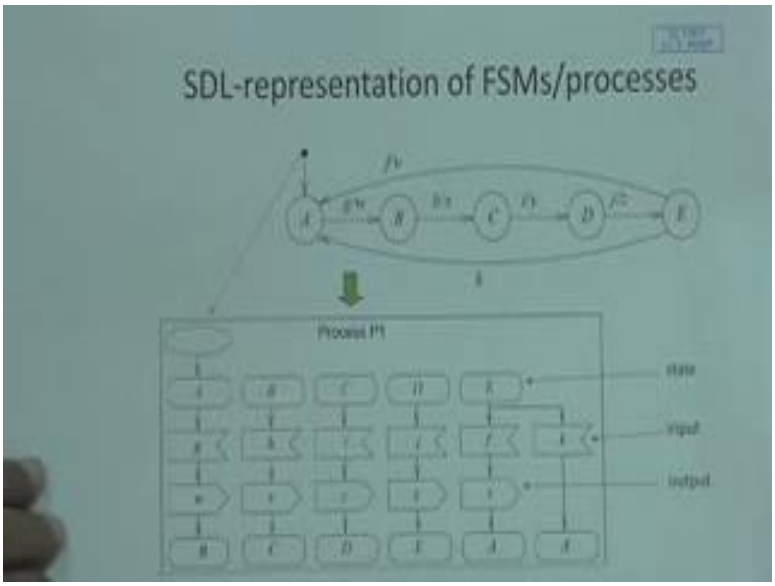
system synthesis as well. This is also by the way, unlike state chart, SDL since it is using asynchronous message passing. It is also suitable for modeling distributed applications. It is suitable for modeling distributed applications, because here we have doing message passing and asynchronous message passing. Now I can have blocking and unblocking, whatever we want to do right. So, here is an example of a SDL.

(Refer Slide Time: 03:27)



But before that let me just mention, about the hierarchy in SDL. The hierarchy is at the lowest level, at thus lowest level will have FSMs, those FSMs are known as processes here. And these processes are abstracted in the form of. So, this is the lowest level, then it comes as block. A block may consist of a number of processes; block can be a collection of processes. And then a number of blocks communicating blocks, give rise to a system. So, I have drawn in the other direction. So, this is the most specific, I mean more granular then we come down.

(Refer Slide Time: 04:34)

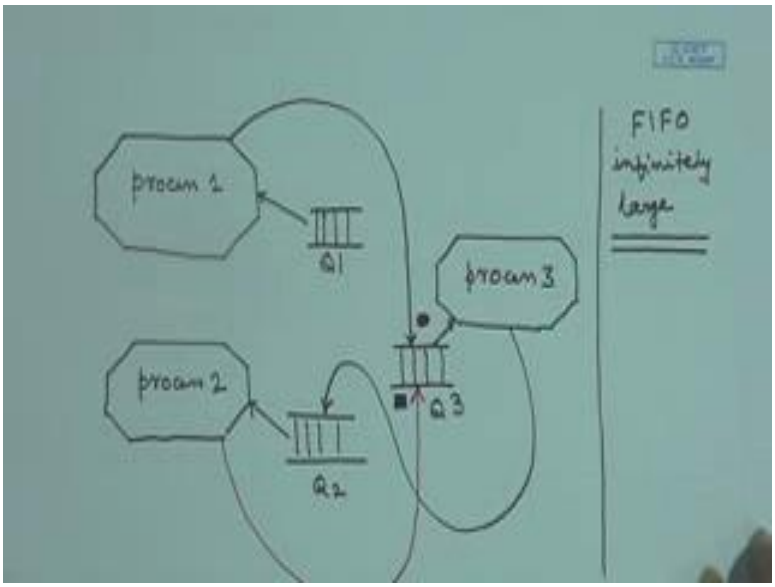


Now let us have an example of a SDL representation of an FSM. This is the typical FSM that I have. We have already seen this example FSM a number of times, in earlier example. So, A to B the transaction takes place on event g, and the output is w. From B to C on event x, the output is x; like that. And here from E we on A on f, event f, we come back to A with output v, and from E we come to k. So this when translated into an SDL, looks like this. So, my homing state or the initial state is here. So, I am entering A. Here I am not showing any concurrency. So, it is only one FSM therefore. Now you see here what happens at each of these oblongs these are the states, and this symbol a rectangle with an input, I mean this arrow inwards, is an input and the arrow outward is an output. So, from A, on input event g, here input even g I produce output w and come to the state B and then from the state B, or input event h, produce the output x and go to C, right.

From E you can see that I have got two edges. So, from E on event f, I produce output v and come back to A, and from event I mean state E on event k, I do not produce any output and come back to event A. So, this is the translation. So the syntax in SDL, rest on these oblongs, representing states, this rectangles with inward arrow as inputs, rectangles with output arrows as outputs, and that is all, so this is one process. Now let us look at, how the communication takes place in this case. So, this part is clear. Now I can have many such processes. Now the process as such, is designated as, an octagon typically, but actually you see whenever you will be working on SDL, you will be

provided with the graphical editor, and you will be doing with all those things. So, the time that I will be needing here, to draw those that you can simply select over there right. So, I can have a communicating scenario, because it is a, ours is a communicating FSM.

(Refer Slide Time: 08:25)



So, I will have one process. So, let us call it process 1. There is another process, process 2, and let us have another process, process 3. Now the communication mechanism between these processes is through first in first out queues. So, for each process, there is an input FIFO right. For this process there is an input, let us call it Q 1, we will have another q, let us call it Q 2, and for this one, we will have another Q, Q 3. Now these FIFO queues are potentially infinite in length; that is assumption. These are assumed to be infinite capacity, infinitely large. Now each process will fetch the next signal; like the signal that we are having here, this g h I whatever. It will be fetched from the queue; that is marked for this; it will be taken from this.

Now if this signal, something is here FIFO I will take this, if this is the trump the event that I am looking for. For example, this one is looking for this event. So, g if it be g then the transaction takes place. If not that signal is ignored by that, it reads, it is ignored except there is a special case, where we can save the signal for future use. Now obviously, when I say it is an infinitely large queue any engineer will certainly ask how is that possible. So, you must understand that these are modeling concept, that we are not taking into consideration any limit on this, but when you implement, you have to, you

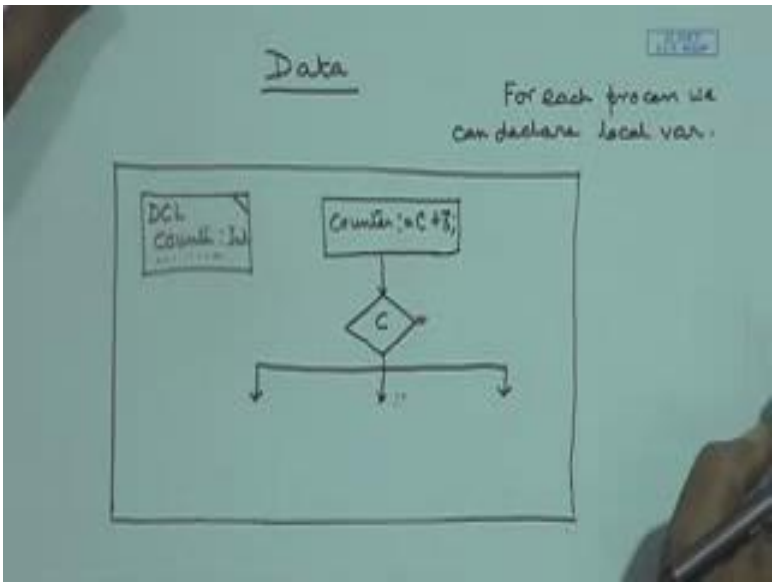
cannot really have an infinite queue. So, there you really require some analysis to be done on the whole problem, to decide on how much memory you need to allocate on this; that is a job, that you have to do, but as this assumption makes my modeling work, but simpler, because I do not have to block and all those things.

So, what happen says, that suppose process one takes some signal from this queue, and finds that, it enables a transition, and the transition produces an output. Now that output can be for a particular process. So, it can put that process, I mean then in this Q, suppose it is for Q 3. Similarly this one will read from this Q 3, and may produce an output into this so on and so forth. So, that is the way, the communication among SDL FSMs are done, communication about FSMs. Now I once again would like to make it clear this is the description of one FSM, one process. Now there will be multiple such processes here. Now all these processes will have to communicate among themselves, for that the communication assumes this, that there is existence of an infinite queue.

Now there may be a question, whether it is determinate or not right, that suppose now a situation occurs, that at a particular point of time; say here process one has put in some data here. Is it visible that is a right color, I do not think so, and Q 2 this process also writes on this, and that is a different token, I am just showing a different token with a different color. Let me put a different shape that will be much clearer. So, there are two tokens coming over here, in the queue.

Now please note that these two processes are asynchronous, and the message passing is asynchronous. Therefore, I have got no idea; I cannot say what will be the order in this queue, circle followed by square or square followed by circle. Accordingly this process three will consume, from these antique actions. Now this consumption will be either circle consumption or square consumption is not known; that is a point of non-determinate behavior in SDL. So, if I run it on a, actually run it on a simulator, the simulator can show different behavior for the same input, and all of them are correct right. So, that is a problem on SDL, but that is there for many such FSM things that is common phenomenon.

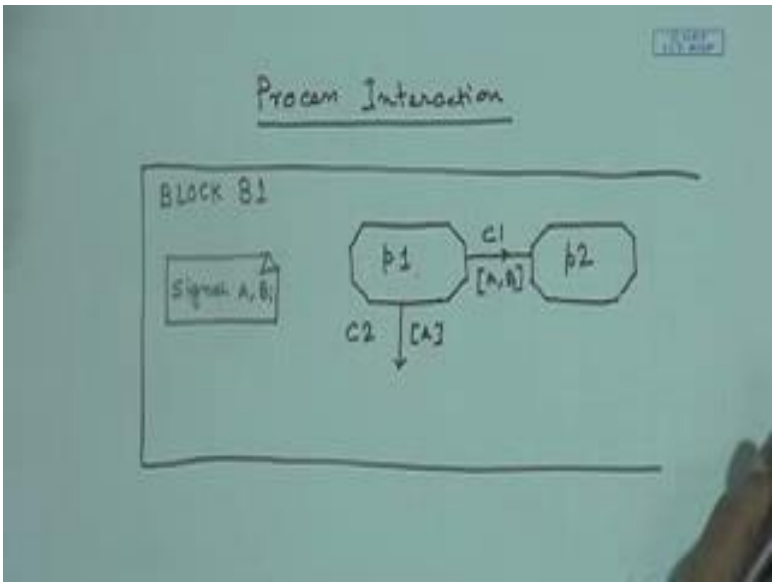
(Refer Slide Time: 15:25)



The next thing that will discuss is, how do we handle data. Now here for each process, we can declare local variables, and with type specified right. So, let us have an example; say I can have a block. Again graphically, I mean something like this, where we have got the declaration, where space is little less.

Counter is an integer, and may be other things like data is a string or other declarations here. Now, in my, I can also, is it clear, counter is an integer. We can use abstract data types also, but see here, there is insider particular process, inside a process; I can say counter is assigned very bad estimation; counter plus 3. So, this counter, and then we come and check, the value of counter and just like a switch statement, based on that I can take different parts. This sort of action is also possible. Therefore, I can also put in data over here. We will look at a concrete example of how we can do the data, but the main thing is, that we can specify data inside a process. So, we have seen data, we have seen the state representation, we have seen the communication primitive for that; that is a queue of unbounded queue.

(Refer Slide Time: 18:22)

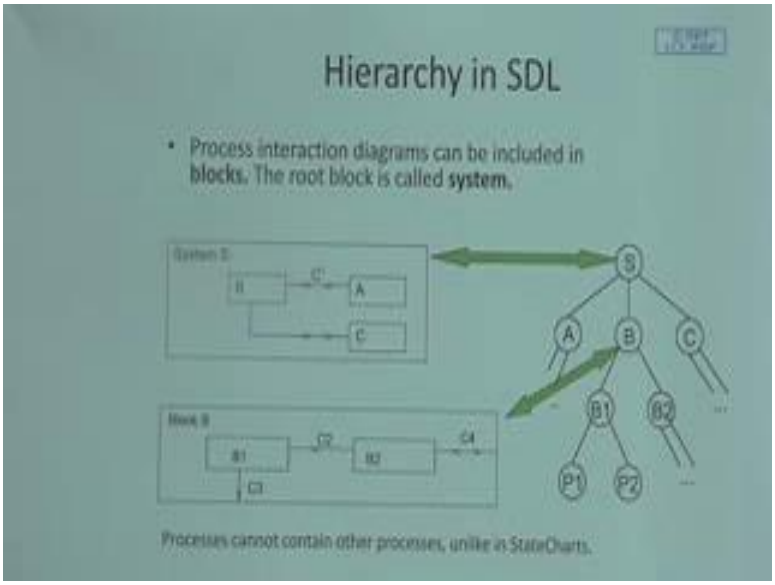


Now, next we look at the process interaction diagram, how would the process, processes interaction among themselves, how would they communicate. So, for that; suppose I have got a block. I said that a block is a higher level than a process. So, say I am describing a block, just as I had in state charts or higher states. So, let me call it say I have blocked B 1, and there I am declaring a signal. Actually the notation is something like this, where I see that A and B are two signals. Now I have got two channels, I have got two processes, let us see. Here is one process, let us call it p 1, and there is another process, let us call it p 2. Now I have got channels, there are two channels, let us call it channel two, and let us call this one to be channel one.

Now I can associate that few channel one, both the signals A and B can be propagated through channel one. So, p 1 can send through channel one A or B, and through this if I just denote that only A can be passed, then I declare it in this way. Therefore, this diagram depicts the connectivity for communication or interaction among the processes, because the in SDL the only mood of communication is message passing, and that will be done through channels. So, I have to specify the channels, and also by using local variables I can specify, which variables are connected to which channel, that does not mean that all the time the variables are only flowing through this channel. Whenever p 1 sends A it can send a through this or through this. In order to send b it will have to send b only over there. So that is above the process interaction.

So, we have seen, let us recollect up to this what are the things that we have seen. We have seen that, we can implement hierarchy as blocks and then processes for the time being. The processes can have local variables; the processes can communicate among themselves using channels. Now how do you really show hierarchy, let us have an example, where we show hierarchy explicitly.

(Refer Slide Time: 22:00)



Let us look at this diagram. Clear, it is coming clear. So, I have got a system s, I say it that at the top I have got a system s, and that I have got three blocks B, A, B, C; three blocks, and there are communications among themselves. Now process is, now this block can contain processes, and that is the leaf node, processes cannot contain other processes, because in SDL what is the process, a process is essentially of FSM. So, I cannot de composite any further which I could do in state chart. In state chart that state machine was itself broken down.

So, here is hierarchy; say s is this thing, then s is divide into A B C, A B C right, and then b can be further broken down, you see this a is a decomposition of the block B, B can be broken down to B 1 and B 2. So, that is B 1 and B 2, and here is an example of their communication B 1 and B 2 communicates with C 2 and C 3. Now b 1 can be further broken down into processes p 1 p 2 etcetera.

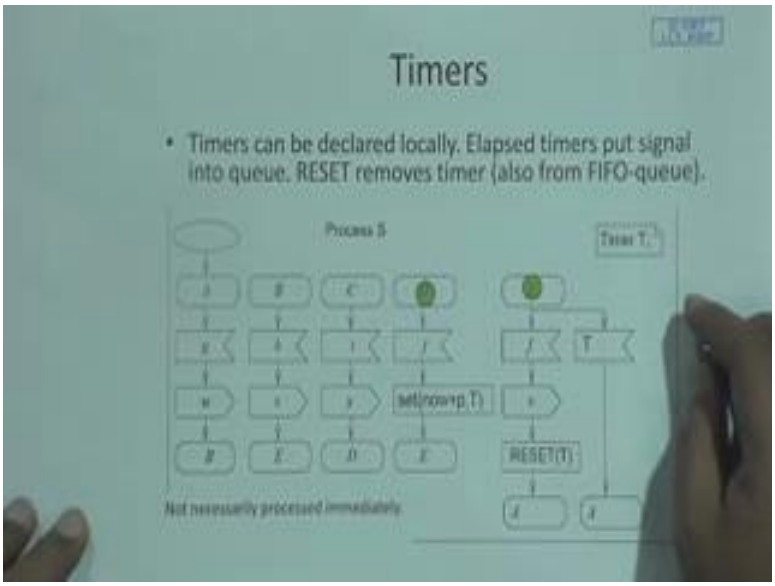
Student: (Refer Time: 23:46).

This is a hierarchy. I am exactly representing, I am representing the hierarchy which is so, essential for specifying complex system. So, whenever you have to do a real industrial level embedded system design, the specification will be so complex. We will not be able to handle that in a flattened manner. Therefore, you have to look at this hierarchy. You can have the details of the system in terms of blocks at one documentation page, and this is also very much applicable for software engineering as well, and then on another page you go, and you put a reference and you come to the lower levels of processes and all those. So, hierarchy I can represent. So, what are my hierarchies; a top leveling system, then blocks each block can be decomposed into processes, and then we have got channels and all those things.

(Refer Slide Time: 24:54)

Another very important thing that is there in SDL, is timers, we have seen timers. In the last class also we specified the timing specification, is a very important aspect of any specification of embedded system. So, how do we represent timers in SDL?

(Refer Slide Time: 25:15)



So, here is an example; timers can be declared locally. So, here I mean a process right, where I have declared a timer T. Again this notation is for declaration of the local variables, just has I have done the counter similarly I have done here. Now what is done here? Now all these timers, timing events are also kept in the FIFO queue. So, we start here. So, this is the old one, this is also the old one. Here for example, this is D and I have got, I am in this state. Suppose right now I have made a transaction from C, through the event I producing the output y, I went to the state D, and this is by states D, this is my state D, and this is by state E. So, I come to state D, and now event j occurs. Let me go back and find out that diagram. Here we had this diagram on D; I was waiting for the event j. Now on event j I produce z and go to e right. So, that one I am now representing as, I am not producing j here. Here I am setting a timer. Look at what is written set; that means, set the timer, now plus p. Now means it will take the current time, whatever time. I mean it can be a relative time, that when it comes to this state j event has occurred, then it is says the timer t, with a value now plus p; that means, what now this timer will start counting down, and then I come to E. So, instead of that action I have set a timer here. I could have done that whatever action was there that plus setting the timer, and then I come to E that E.

Student: (Refer Time: 27:48).

Yes let us see what happens here, I have set the timer. Now I come to this; f happens, I take v, if f happens earlier I will produce the output v and reset the timer. Otherwise whichever comes earlier just like a watchdog suppose I have said that from now give 20 millisecond and this event f does not take place within 20 millisecond. Then this t timer will be giving you the time out signal, and so that is coming as an input and it will go to a. This is clear so.

Student: (Refer Time: 28:30).

No whenever the timer triggers, then automatically this time timer is reset automatically. Here I am doing it forcibly right, here I am doing it forcibly, but when the timer triggers it will automatically reset, and (Refer Time: 28:48) means again it will be now, it is not active right, because again when I come through this process the time will be set here. It is not automatically set to the now plus p ok.

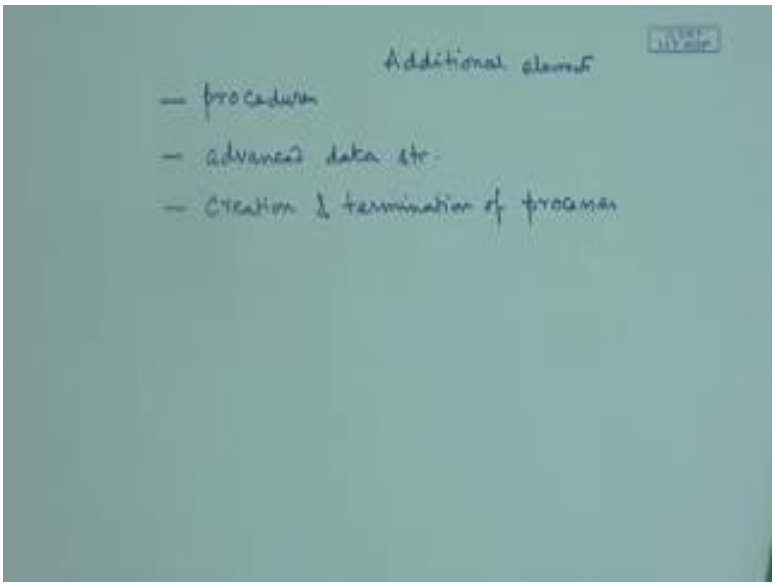
Student: What is now?

Now, means the exact, whatever time the real time where, whatever is the time now, you come to this point. So, from here it is an offset p is an offset, from here this. So, now, is a very useful construct, that has been used in SDL where I can fetch the time, current time wizard, I could say if now is greater than this or something.

Student: (Refer Time: 29:33).

That is from the global clock, global clock that the system will have. The system has got a clock and from there it will take a value, and then it can make it a relative value or absolute value that really does not matter in that case. There are additional language elements like procedures.

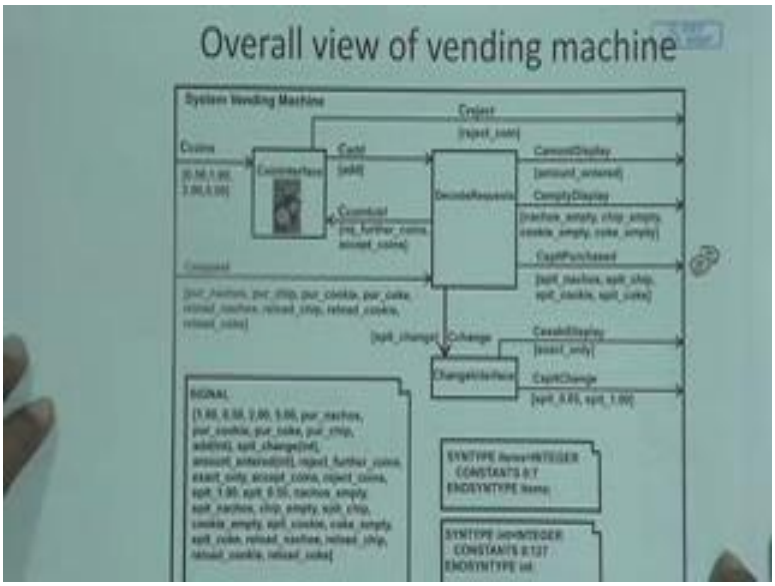
(Refer Slide Time: 30:02)



We can add procedures; we can have additional elements, which I am not discussing in detail. We can have advanced data declarations, data structures you can put in, and you can create and terminate processes, and there has been a different variety of this. Now the reason of discussing SDL is mainly to show that here in a state machine we can also do it using message passing, a synchronous message passing, and the timers and all those things are a little different.

Now I conclude, there is also another detail example that will do, but let me just show you a simple example, we will do the details later, say I want to build up a vending machine all of you know about the vending machine, and in finite state automata or switching theory of must have designed some FSM for vending machine.

(Refer Slide Time: 31:50)



So, here is a little complex vending machine that I am showing here. It is real complex, so we will deal with that maybe tomorrow, or in the next class. So here is the complete vending machine, there is a system. And what are the blocks in that? The blocks are one is the coin interface, one is the decoding the request, whatever requests are coming, what I want to have a coke or a cookie, a block to provide you the change. So, these are the three blocks, and there are number of declarations. These declarations will come to later, and along with these blocks, there are channels defined. So, I have got a channel through which I can have the coin, and what are the coins? 50 paisa, 1 rupee, 2 rupees, 5 rupees. There is a channel between coin interface and decode request, whatever I am adding that will go through this channel lab, so it will also have a coin control on this side, and this one, is accepting the coins.

So, it may, it will, and these, this one is also accepting the request for what you want to purchase, purchase nachos, purchase chips, purchase cookies, purchase coke. This shows you the behavior. Through this I can also say I want to reload nachos it is over, reload chips, reload cookies, reload coke. So, all these are coming through request channel. And based on the request if the amount has been given, is also displaying the amount, is showing whether the anything is empty, something that is purchased is spit out; that means, given. And if there is any change it comes this change interface, which shows the exact display, and provides a change. So, at the top level this is what I have right. So, here what I am showing, I am showing the blocks. See although I am just showing the

blocks. So, you can look at this and you can understand what it is though. I am coming to the signals; later signals are all these data that can come up. These are integers; like where I add is an integer everything. So, this is the block and there are channels there. So, that is why top level.

Now in the next lecture I will break it down, and show how this can be processed further.