Embedded Systems Design

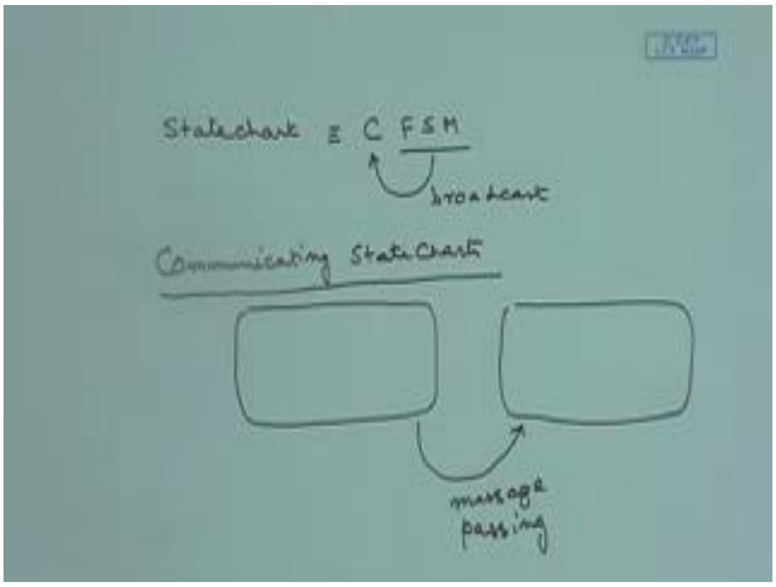
Prof. Anupam Basu
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 36
Program State Machines

So, we were discussing about specification languages. And, in the earlier lectures we have talked about state charts and, also looked at the features of synchronization, broadcast mechanisms and all these.

Now, I would like to stand corrected in one point, that is, I said that state charts can be applied for distributed systems also. But that is true for some extensions of state charts, not the state charts up to the extent that we have discussed; because in state chart the communication mechanism is broadcast. That means, there is a shared variable everything everybody is writing on the shared variable. In a distributed system because of the network communication and the delays, the updates of all the variables may not be effective at the same time. So, state chart as such is not suited for distributed systems, but there are some extensions of state charts, which have been worked out, where there are multiple communicating state charts.

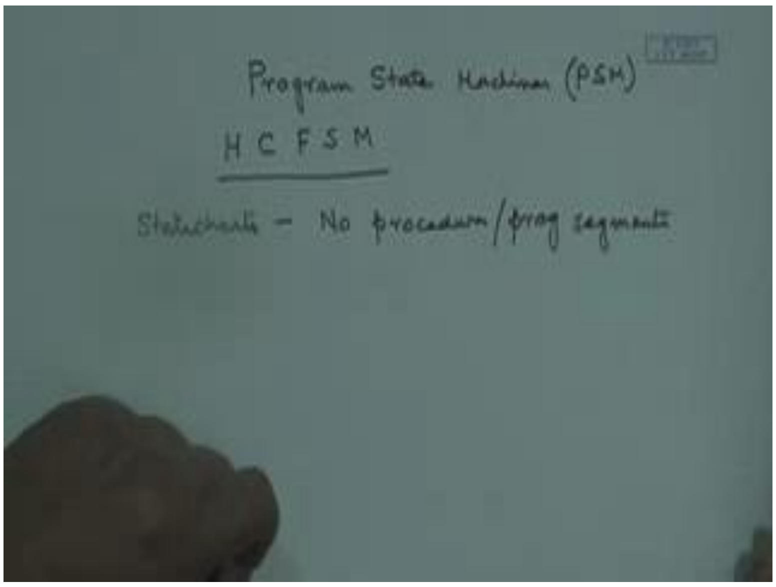
(Refer Slide Time: 01:38)



Now, state chart is basically a communicating finite state machine. State chart is communicating finite state machine, where there are multiple finite state machines communicating among themselves using broadcast mechanism. Similarly, there can be communicating state charts where their reference state charts and they communicating among themselves using message passing; type of synchronous as well as asynchronous message passing among themselves. So, that variety of this communicating state charts are suitable for distributed systems.

Given that, we come back to our point of behavior completion. We have seen that normally in finite state machines we show that the behavior completion is in the final state from program structures by the end statement and all those. And, we have shown that in structure type of representation some such square blocks are used for designating the end of behavior.

(Refer Slide Time: 03:20)



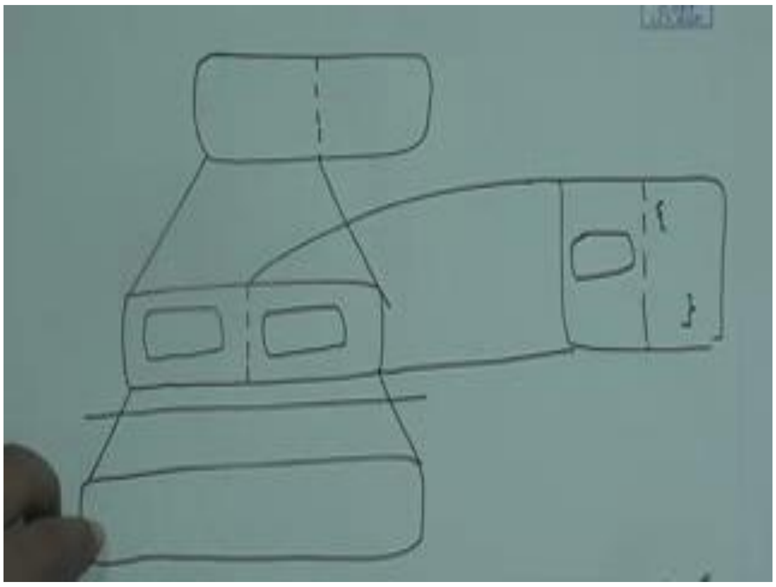
Program State Machines (PSM)

HCFSM

Statecharts – No procedure/prog segments

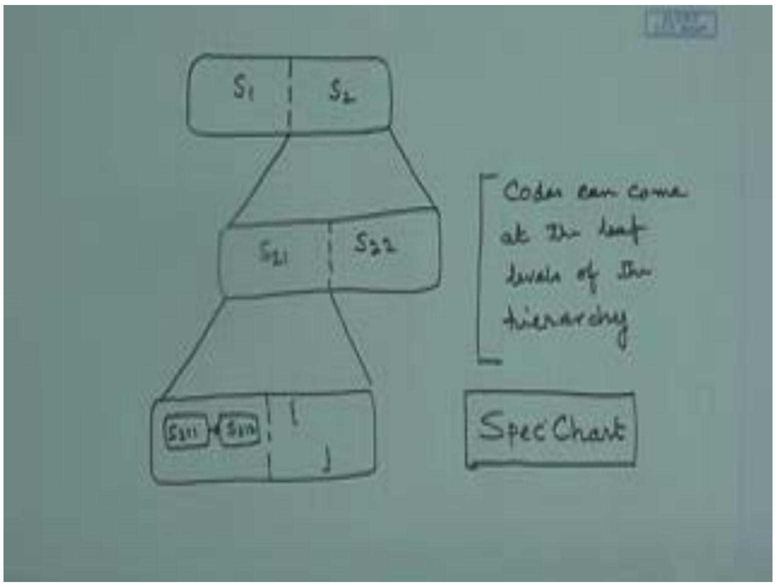
Now, based on that another extension; you can think of it is an extension also of state charts. That is known as program state machines or PSM. These are also heterogeneous models. And, Hierarchical Communicating Finite State Machines; these are also a variety of HCFSM, just like state charts. But, one thing that was. So, I can show hierarchy, I can show communication among different HCFSM. But, in state charts we did not have the scope of writing procedures. No procedures could be embedded. Procedures or program segments could be included; however, in this we can have both the state machine as well as program. For example, there are some behaviors. I told you in an earlier lecture that there are some behaviors, which can be better expressed as codes.

(Refer Slide Time: 04:56)



So, let us think of a situation of representing hierarchy; when I start with a level, alright, a particular level which shows two concurrent things, two concurrence state machines. Now, I blow this up. And, I show another; this part is blown up. And, I show two an, I mean, two other concurrent machines. In that will go on, but ultimately when I come further down it maybe so that here I am sorry, let us forget about this part. Let me. Suppose I take this one and blow this up, so this one can be further broken down into two machines. Maybe one is a state machine; another could be a piece of code. I think this has become clumsy enough.

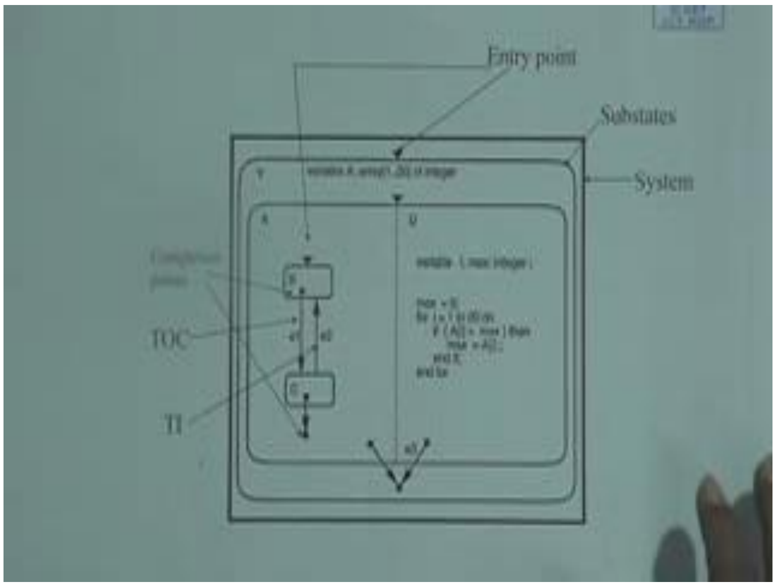
(Refer Slide Time: 06:15)



So, let me draw another diagram for this. That is, the top level I start with state machines. There are two state machines S1 and S 2. I blow up S 2. And S 2, still the situation is complex enough. So, I want to maintain the hierarchy. So, I blow it up as S 21 and S 22. Now out of these, maybe further when I break down S 21, then I can say that S 21 consists of say another two state machines, might be; S 211, S 212. And, the other, there is another concurrent part which is simple enough, which I need not to decompose further. So, that part I can show as a code.

So, the codes can come at the leaf levels of the hierarchy. Codes can come at the leaf level of the hierarchy. So, ultimately this can be either a state, simple state or a code. So, we will have a couple of examples of those. So, that is why this could have known as program state machines. And, there has been systems based on this idea. And, the system is known as spec chart. Spec chart. So, let us have a look at one such spec chart diagram here.

(Refer Slide Time: 08:53)



So, here you can see that since we have got codes, I have got the provision of declaring the variables also. I think this is visible, visible? So, here we are showing some entry points. Just like state chart, I entered this. And, these are the substates Y. Now, under y we have got two substates A and D. A, again consist of two states B and C, where there are communications. And, concurrently D is not decomposed any further, but has been represented as a piece of code. Like, variable I max is an integer, max is zero. It is a pure loop. So, I am finding the max. Finding the max; for that I do not need to go for a state machine So, I can, it is much more convenient if I write it as a code.

Now, we can see the combination. Now, there are two other very important things. In the last class, we talked about behavior completion. Here, we are showing two types of behavior completion notations. One is transition on completion; another is transition immediate. What is transition immediate at these? Like, look at this edge e 2. Whenever e 2, event e 2 occurs, if I am state C, then I immediately move to state B, wherever I am in my computation in state C.

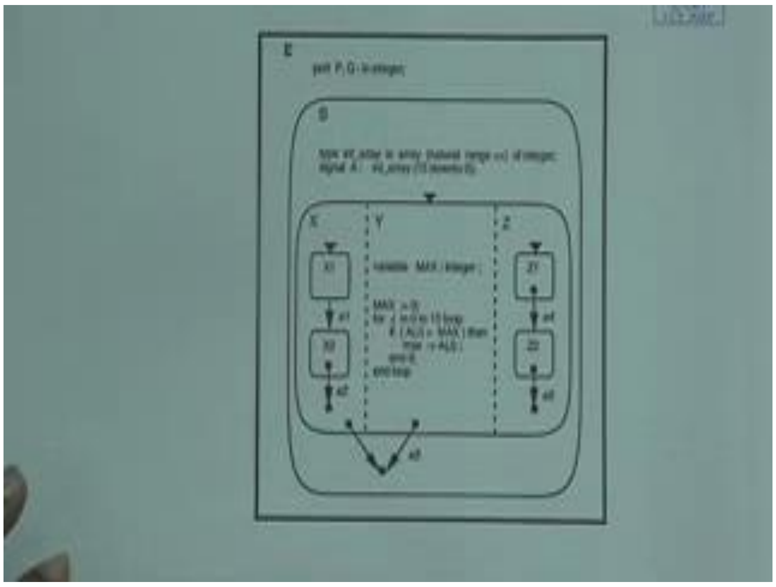
However, look at this edge e 1. Edge e 1 is connected to this black square. That means what? That this will be the completion, so if e 1 takes place this transition will be taken only when the behavior of B is completed. Very similar to the interrupt handling sort of thing, you remember that if an interrupt comes, I first complete the execution of the current instruction and then move out to another state.

So, there are two types of transitions; transition on completion and transition immediate. What is this? C to the end, this is obviously is the completion and here e 3; that I will come to this completion. When will this entire behavior be completed? If you look at it the behavior will be completed when the behavior A is completed. When is behavior A completed? When C is completed, when C is completed, then behavior A is completed, but this entire behavior Y will be completed, when also D is completed. So, if D is completed and event e 3 occurs, then I come out.

So, here is another transition and completion. So, suppose this part has been completed, computed, but still e 3 has not occurred. So, this has completed. But, I am not coming to this point from here. So, when both these complete, then only come to this point. So, the variation of PSM with respect to the state chart is two port, at two port. One is that I can encode the same sort of hierarchy. But, I can encode some behaviors as procedures and I explicitly put in the types of transitions. There can be two types of transitions; transition and completion and transition immediate.

So, we can see that PSM is representing the states, system states data. I can also represent data and the activities. Hierarchy is maintained, unlike typical procedural model. In a typical C program, for example, what is the problem in a typical C program? I cannot represent states. Here, I can also combine state representations. And, as I said one of implementation is pictured, which is designed by Dan Gajske, a big name in the area of embedded systems.

(Refer Slide Time: 14:05)



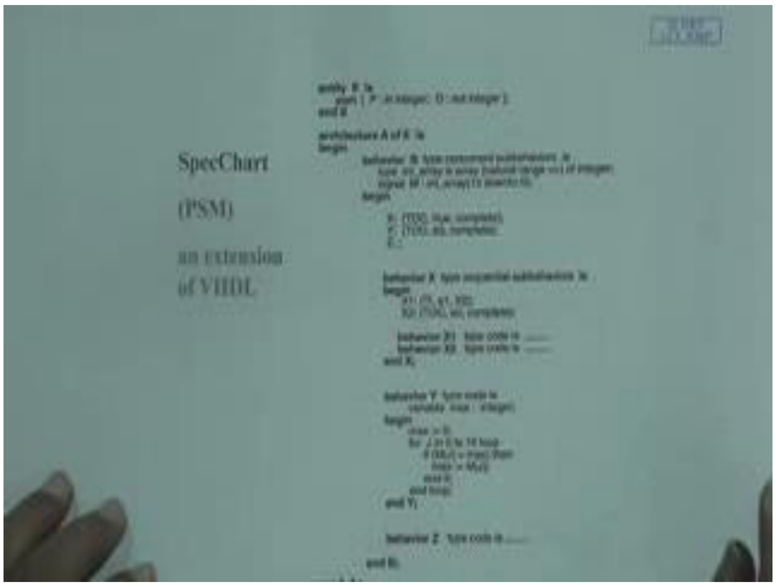
So, here is an example of a state chart. You can see this. Sorry, of a spec chart behavior. So, we have got a behavior E which has got two ports P, Q, which are integer. So, I can put in my, some languages. For example, VHDL I can put it here, embed here. Int array is array natural range of integer. Signal A, which is can be 0 to 15 an array. And, so this is, these variables will be used with in this block B; global to this block B.

Now, this behavior B is now being shown as consisting of X, Y and Z. X is again decomposed as the next level of hierarchy. Although it is being shown in one level, this is flattened, but say for example, when you are making a specification document, you are not flattening it. So, at each of the levels might be you are showing only this. E is a port, is a system consisting of two ports. And, in that is a behavior B.

Next page, you come and say what the behavior B is. Behavior B has got these variables and consists of three process - X, Y, Z period. Next step, if you come to x and show what x is. That is hierarchy. So, here you can see that there is a transition immediate e 1 and transition on completion e 2. Similarly, for Z we have got transition and completion on e 4 and transition and completion on e 5. And, Y has been put in as the code of finding the max. So, there are three.

Now, given this I can, I need to synthesize based on that. So, I may like to convert it into some high level language. I can transfer it to some intermediate. In the last class, I said I can transfer it to some intermediate language for which I will design a synthesizer, I mean synthesis tool, which will synthesize it or I can convert it to some standard language like VHDL. For example, let us have a look at the correspondence between these two.

(Refer Slide Time: 16:48)



SpecChart

(PSM)

an extension of VHDL

```
entity E is
    port ( P : in integer; Q : out integer );
end E;

architecture A of E is
begin
    behavior B type concurrent subbehaviors is
        type int_array is array (natural range <>) of integer;
        signal M : int_array(15 downto 0);
    begin
        X : (TOC, true, complete);
        Y : (TOC, true, complete);
        Z : ;

        behavior X type sequential subbehaviors is
        begin
            X1 : (TI, e1, X2);
            X2 : (TOC, e2, complete);

            behavior X1 type code is ......
            behavior X2 type code is ......
        end X;

        behavior Y type code is
            variable max : integer;
        begin
            max := 0;
            for J in 0 to 15 loop
                if (M(J) > max) then
                    max := M(J);
                end if;
            end loop;
        end Y;

        behavior Z type code is ......

    end B;
```

So, here I will just move back and forth. Entity E is when I convert into VHDL, entity E is port P, Q. So, in my diagram entity E is just port P, Q. End E. So, E is finished. So, that is the top level of description.

Next, I also say architecture of E is. What is there in E? Behavior B begin and end A. These are architecture. Behavior B, concurrence of behaviors or, I mean is they are concurrence of behaviors. What are those? X, Y, Z. Behavior B consist of concurrence of sub behaviors X, Y, Z. And, there are data declarations. So, I put in the data declaration and then put this X, Y, Z. Now, X has got a transition on completion to complete. These are the transitions.

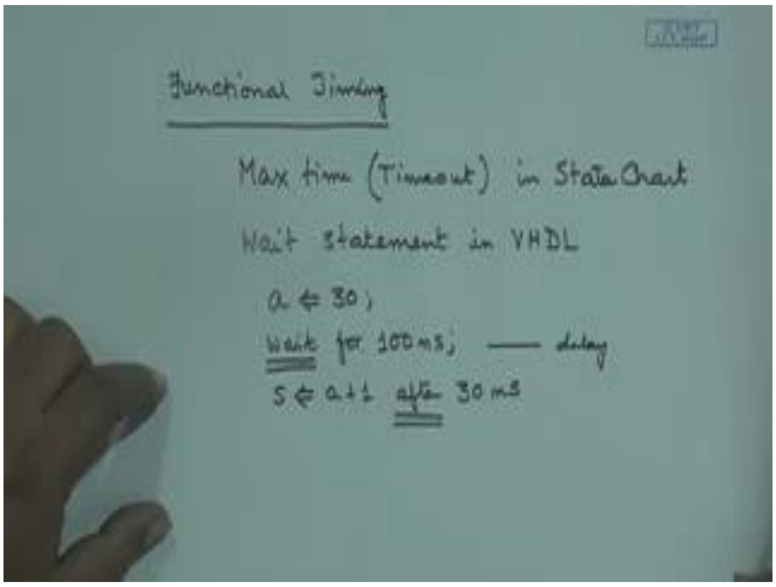
Next, I come. To the next level of indentation, you can see as I convert it to flattened code. Behavior X what is behavior X? Behavior X is sequential behavior of X 1 and X 2. So, behavior X sequential sub behaviors X 1 and X 2. X 1 is what? T 1, there is a transition e 1 and that brings you to X 2. That is X 1. Transition immediate e 1 brings you to X 2. And, X 2 what is that? There is a transition and completion named e 2, which will lead you to the complete state.

Let us check X 2. There is a transition on completion e 2 that brings you to the complete state. Then, behavior X 1 type code is now; now, this behavior X 1 can be further decomposed. So, I may next go to a code level for X 1; whereas for Y, at this level itself I have come to the code level. So, here what is shown is behavior X 1 type code is whatever, X 2 code is whatever. That is end X.

Next, behavior Y. Behavior Y is purely a code. So, type code. I put the entire code here. Similarly, behavior z and then, ends my behavior of B. That is the description of B. So, here I have shown it hierarchically. Each of them convert it and then club together in the form of a code. So, that is how we can handle hierarchy and concurrency in the form of PSM.

Now, let us look at some examples, specification languages. We have talked about some. I mean, they require some requirements of specification languages. Now, there are so many specification languages you have shown; state chart, spec chart, all those. But, there are other languages like VHDL. We will soon discuss another language SDL. And, so we can see that some languages, not that all languages provide all the features that are captured in the model.

(Refer Slide Time: 21:00)



Functional Timing

Max time (Timeout) in State Chart

Wait statement in VHDL

$a \Leftarrow 30;$

wait for 100 ms; —— delay

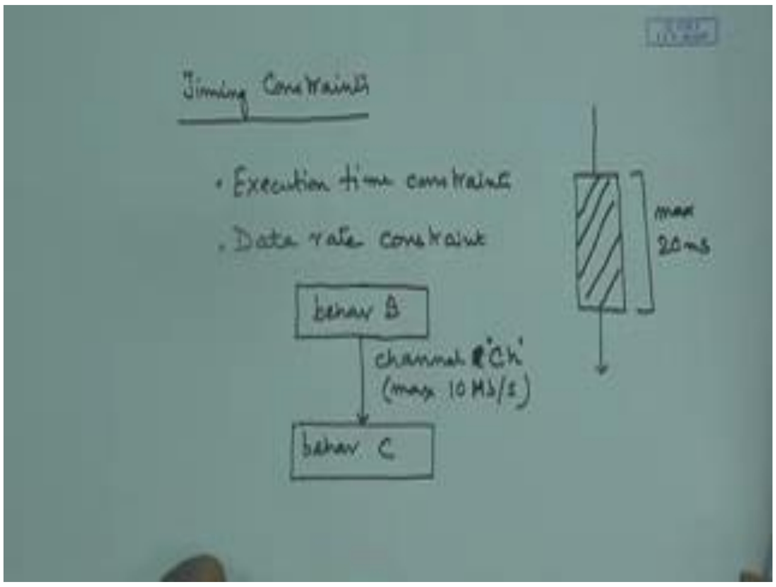
$s \Leftarrow a+1$ after 30 ms

For example, functional timing if we talk about, this is a very important thing. Now, functional timing is what? The time; there are two types of timing I said. One is non functional versus functional. Functional timing will be what will be reflected. The timing delays which will be reflected when I simulate the system. As I simulate the system, the system will show some behavior, and that behavior has got that timing behavior embedded in it. For example, say we have that Max time or Timeout in state chart.

Similarly, we have got the wait statement in VHDL. Now, whenever we actually start working on some specification, you will adopt some language. And then, those nuances of that language you will learn at that time. Here, I am just showing you some examples. For example, wait statement. So, for example, A is a signal, a variable given value 30. And then, I say wait for 100 nanoseconds, that is, waiting. It is not a timeout. It is just a delay. This is just like a delay. I provide a delay. And then, S is assigned a plus 1. S is a signal after 30 nanoseconds. So, here is, VHDL provides you this statement wait as well as after. So, when I, I wait for 100 nanoseconds, then I come here and again wait for 30 nanoseconds and then assign. So, these are two statements, which are inherent in VHDL; so that, you can. So, similarly there are other statements also which can be used for time.

Now, there are. So, one is what I talked about right now is functional timing. That this thing will be reflected as I simulate the system. If there is a wait, then the system will wait there. I can see that.

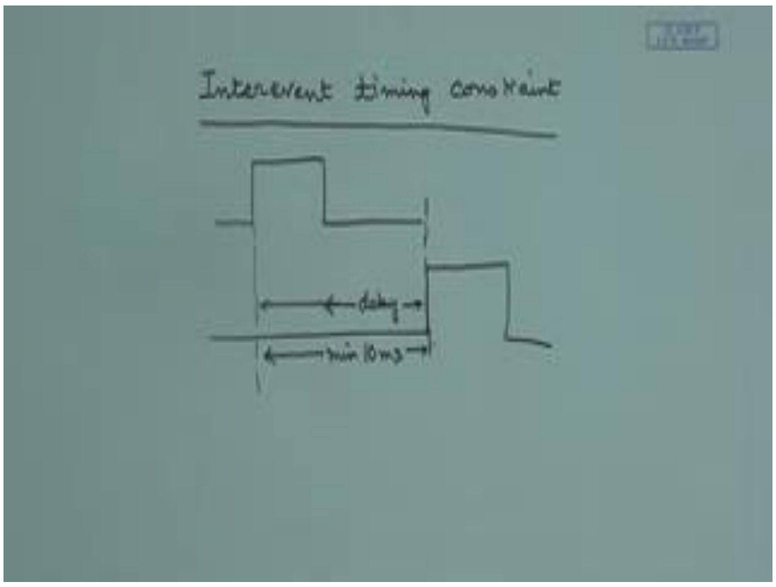
(Refer Slide Time: 24:01)



Besides, there are some non-functional constraints which are known as the timing constraints. Now, that can be sometimes, for example, there can be execution time constraints; that means, say I think, I mentioned it earlier that this is a piece of code or piece of behavior that has to be executed and that has to be completed within some max time, max say 20 milliseconds. I may like to specify that. So, this is execution time constraint that I must complete it within so much time. So, it is a behavior. Another thing can be data rate constraints. That is also a constraint.

For example, I am having two behaviors; behavior B and another behavior C. Now, this is passing on data from here to here using some channel C. Some channel Ch. Using some channel Ch, it is passing the data. And, I also say that max of data transferred it will be 10 MB per second. Now, this is another constraint that have to be specified. Now, the language is not providing me any feature to specify this constraint 10 MB; 10 mega bits per second. Now, that is not being possible. So, what we do in this case is we put them in a separate file and the verification of that we take care of during synthesis. And, at the time of verification we see whether these things have been adhered to.

(Refer Slide Time: 26:55)



Similarly, there can be interevent timing constraint also. For example, an event takes place here. You know that. And that other event that should take place will be at least with the particular delay of this. So, I can specify must delay here; something here. Maybe minimum of 10 milliseconds; that means, after this event occurs, usually I can say from occurrence of this event. So after this event occurs, this event should not occur before 10 milliseconds.

Now, one very important thing is that you have to consider; how can you implement these things in an embedded system. Obviously, let us look at the first one, say maximum of 20 millisecond delay. How do you do it? How do you achieve this? This, some of these things we will see later during the synthesis phase.

This is dependent on the entire behavior. The entire behavior can be represented in the form of a graph. We will show that with different operators and their dependences. How do we schedule them? Where? When do we start them? Give them the processors or the hardware that is needed. So, that is scheduling allocation. All those things are important. Also, when we write codes, how efficient the codes are? How can we generate an efficient code through compilation, so that this timing constraint is satisfied? That has to be done.

So, during synthesis all these things have to be taken into account. For example, this is a, this is whenever you are providing a channel, then you have to select the channel based on this bandwidth. Now, similarly how would you propose to do this? Say, there is some delay; minimum 10 milliseconds. So, you may have to use some timer, so that after this occurs, if this event comes, you have to put in something that with at least 10 millisecond delay, you will forward it. Otherwise, you would not forward it. You will have to keep it in a queue. So, these are some implementations used that we will come across. So, these are the some of the timing constraints that we are looking at.

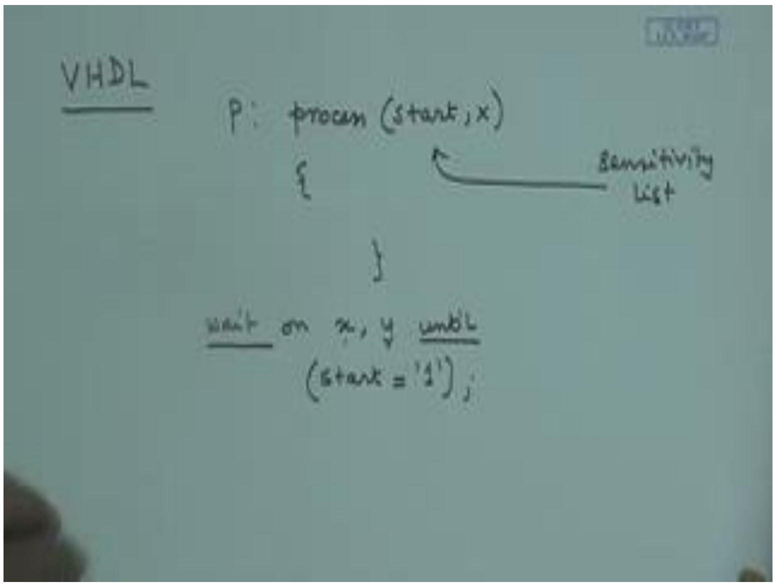
Student: (Refer Time: 30:07).

Timers and (Refer Time: 30:10).

Student: (Refer Time: 30:11).

Yes, I mean assert. But, assert is a statement. That you are specifying in the program, but when that assert is being implemented that is going through some hardware. So, that has to be done.

(Refer Slide Time: 30:41)



VHDL

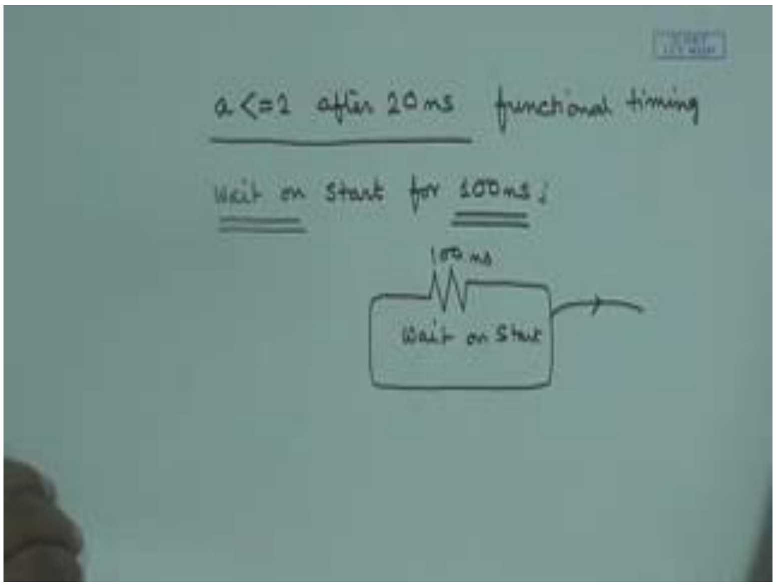
```
P: process (start, x)
   {

        }
wait on x, y until
   (start = '1');
```

Sensitivity list

See for example, in VHDL we can see what really happens. VHDL also supports concurrency. But, say VHDL has got some means like this. We can say there is the process P. It is an example I am taking from VHDL. Different languages provide you different features. So, we can say that P is the process with some sensitivity list; start and X. So, these are the sensitivity list; that means, these are affected in this computation.

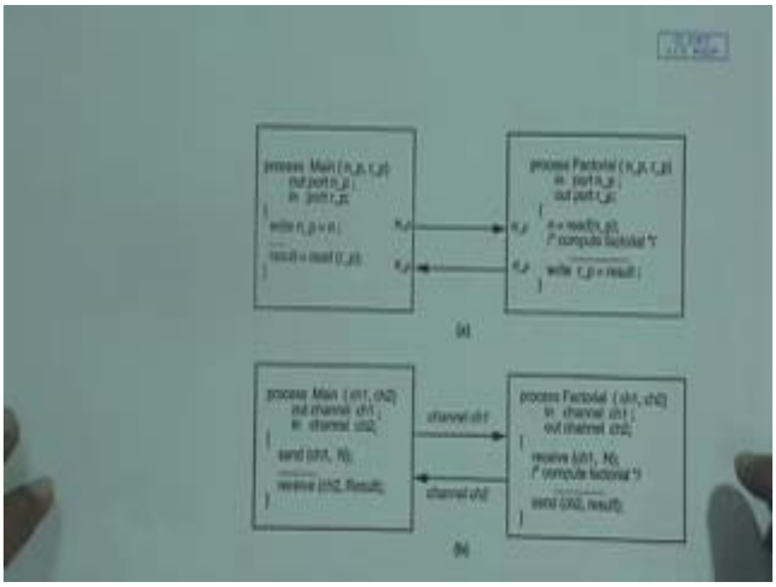
We can also have the wait statement here. Like, say wait on x, y, until start becomes 1. This is the VHDL constraint. So, wait on some variables; you are waiting on them. But if start comes, then you are not waiting any longer.

(Refer Slide Time: 32:03)


a <= 2 after 20 ns functional timing
wait on start for 100 ns;
100 ns
wait on start

Similarly, another way of representing timing in VHDL is say functional timing a assigned 2. I have already shown that. After 20 nanoseconds, so this is what type of timing functional or non-functional? This is a functional timing because this has to be reflected in the behavior. This is a part of the behavior actually. That a should be assigned to after 20 nanoseconds. Similarly, when I said, if I again, that said wait on clause, say wait on start for 100 nanoseconds. So, what is this wait on and this timing? It is essentially a time out; that means, I will be waiting on start. Similar to this that if I had gone for state chart type of thing that I am waiting here, this is a wait on start, but the max delay is 100 nanoseconds. If time out, I go out of this. So, that is the equivalence sometimes. So, these things you have to look at whenever you look at specifications, when you want to do some specifications.

(Refer Slide Time: 33:42)



```
process Main (n_p, r_p)
    out port n_p;
    in port r_p;
{
 write n_p = n;
 result = read (r_p);
}
```

```
process Factorial (n_p, r_p)
    in port n_p;
    out port r_p;
{
 n = read(n_p);
 /* compute factorial */
 write r_p = result;
}
```

```
process Main (ch1, ch2)
    out channel ch1;
    in channel ch2;
{
 send (ch1, N);
 receive (ch2, Result);
}
```

```
process Factorial (ch1, ch2)
    in channel ch1;
    out channel ch2;
{
 receive (ch1, N);
 /* compute factorial */
 send (ch2, result);
}
```

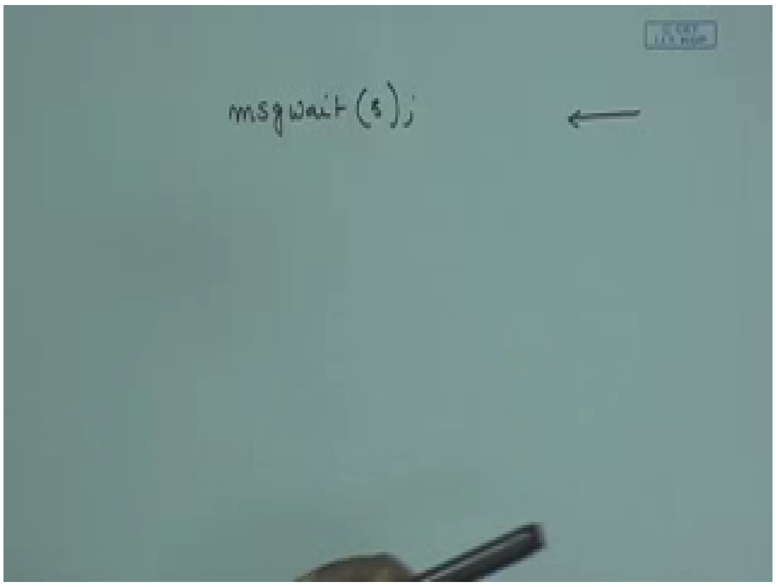
I conclude this with an example of say two pieces of code. This is also a representation. Visible? So, here I can see there is our old programming type of example. Process main having n p and r p, which are two ports. And, there is a process factorial or old C program, how i compute this. So, we do something. And, here between these two processes there is a channel say n p. I send n p here and I get the r p here. Now here, what is written over here? That this is writing on n p and this is reading on n p. Now, if it goes on writing two values very fast, then the second value will be read here. So this is a; I mean, this requires some synchronization.

On the other hand, these are just ports. So, ports are connected by some wires, for example. Now, how do you implement? How do you implement them? I can connect the wire. Now, in the wire as the data is coming, the earlier data is getting lost. There is no buffering.

But, here in the second case I have got two channels, out channel, channel 1 is out and channel 2 is in. Channel 2 is in and channel one is out. Now, here I am sending some data n through channel 1 and this is receiving through this channel. Now, whenever I come to channel, you know that we have got two different modes. What are the different modes? One is synchronous and asynchronous. So, in the channel I can have asynchronous or rather we call it blocking and non-blocking. This is I am passing this message. If I do it in a blocking mode, then I will wait till the acknowledgment and then

I will proceed. And, if I do it in a non-blocking mode I will go on pushing it. So, this is another type of description.

(Refer Slide Time: 36:12)

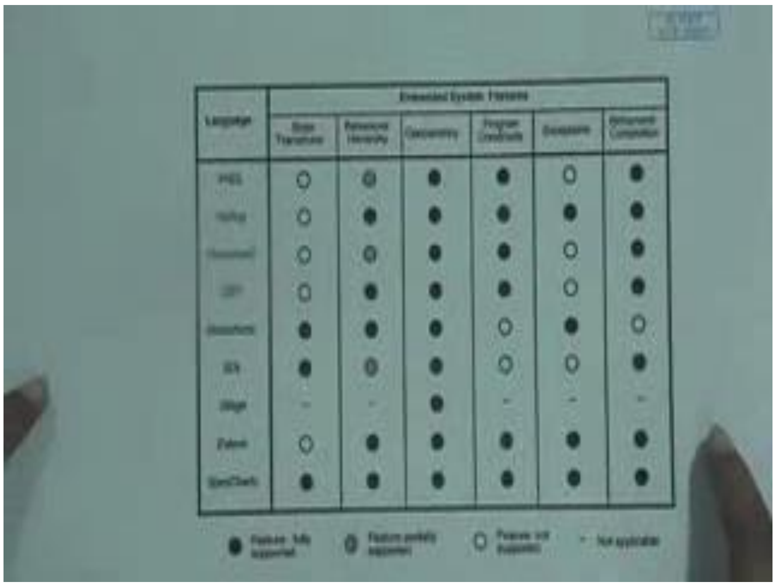


msgwait (s);

So, since blocking message passing for that again, since blocking message passing for that, again there are constructs like messagewait s; that means, whenever the sender has sent the message, it is doing, waiting for some messages which will come from the receiver. And then, I know that my message has been received. So, I can go ahead.

So, these are the different ways in which there are different languages. We will come to another language discussion on another language called SDL in the next class.

(Refer Slide Time: 36:48)



Language	Embedded System Features					
	State Transitions	Behavioral Hierarchy	Concurrency	Program Constructs	Exceptions	Behavioral Completion
VHDL	○	◐	●	●	○	●
Verilog	○	●	●	●	●	●
HardwareC	○	◐	●	●	○	●
CSP	○	●	●	●	○	●
Statecharts	●	●	●	○	●	○
SDL	●	◐	●	○	○	●
Silage	-	-	●	-	-	-
Esterel	○	●	●	●	●	●
SpecCharts	●	●	●	●	●	●

● Feature fully supported ◐ Feature partially supported ○ Feature not supported - Not applicable

But, before that I will just show you a comparison. Say for example, we have got different languages VHDL, Verilog, HardwareC. I mentioned about HardwareC. CSP; communicating in state machine, state chart; SDL we will discuss. We will discuss SDL. Spec charts, we have discussed to some extent. Esterel is by the way. We are not covering Esterel. This is Esterel; E s t e r e l, which is not very, we are not discussing that. But, it is a very industrial popular language.

Now, we are looking at it from different perspectives. State transitions; VHDL, VHDL does not support. Verilog does not support. HardwareC is a code. C program type, they do not do. State charts does; SDL we will see does. Spec chart does. What about behavior hierarchy? VHDL to some extent; partially supported. Verilog; Verilog you have seen that they show behavioral hierarchy. HardwareC, again partial; State chart, Esterel and spec charts do it very well.

Concurrency - all of them show. Normal C language does not do, but all these do concurrency. Program constructs, we have seen. Obviously VHDL, Verilog, HardwareC, CSP, state charts, no. SDL also we will see no. Esterel, yes because it is a code type; spec chart, yes. Exception handling; Verilog does, state chart does. We have seen the exception handling. Esterel and spec chart also does. Behavior completion, for all the programming structures they will obviously have. End of the program. State chart does not have. SDL has. We have seen spec chart also has.

So, this is how we can look at the different features that the different languages are giving us. So, summarizing what we have done till now coupling with the earlier classes. We have seen that in order to represent or specify a complex behavior, we need to take recourse to hierarchical abstraction. And that is provided by the languages. Whichever languages provide us that feature; we can think of that language to the candidate for the representation. Concurrency is very much inherent in most of the processes. So, that is a very much desirable feature. Timing constraints we have to see whichever we can replace it. Synchronization; what is the synchronization primitive. Behavior completion is another thing. So, all these things we have to look at. Now, we have seen state chart. And, in state chart and spec chart we have seen that the communication is mainly through broadcast or shared variable.

Next class, we will start discussing SDL, where we will see that that is also state based representation. And, there we will be doing the communication through message passing.