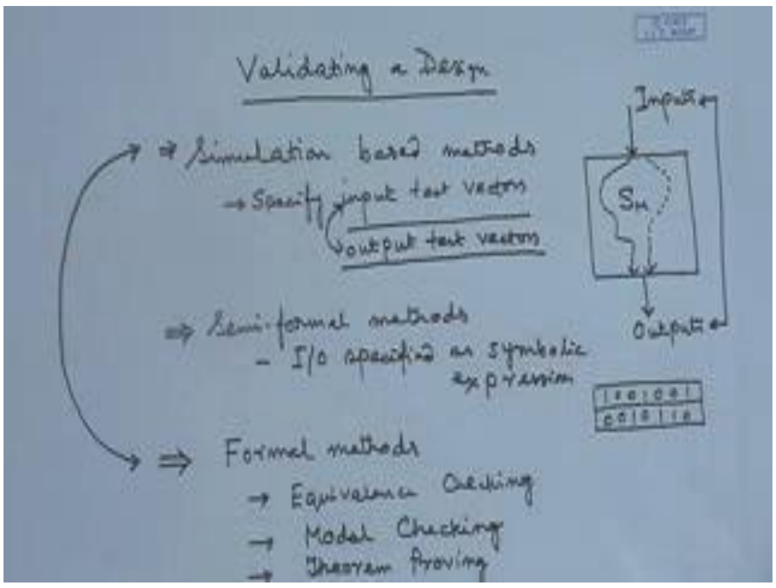
Embedded Systems Design

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

Lecture - 48
Simulation

Good morning, today we will be discussing about different methods by which we can evaluate a design. Say, we have seen also the series of the lectures that we had we started from a specification. We had a specification and then we also studied the different ways in which a specification can be made. Since, the specifications are executable, they can also be at that time they can be checked whether the specification is actually capturing the design intent. And after that we have gone through the hardware-software partitioning, we have done the hardware synthesis, we have done the software synthesis, we have done the optimizations. Ultimately what we get we need to test that whether that is actually giving us the desired output for the desired set of inputs.

(Refer Slide Time: 01:35)



Validating a Design

→ Simulation based methods
  → Specify input test vectors
    output test vectors

⇒ Semi-formal methods
  – I/O specified as symbolic expression

⇒ Formal methods
  → Equivalence Checking
  → Model Checking
  → Theorem Proving

Inputs
S_H
Outputs

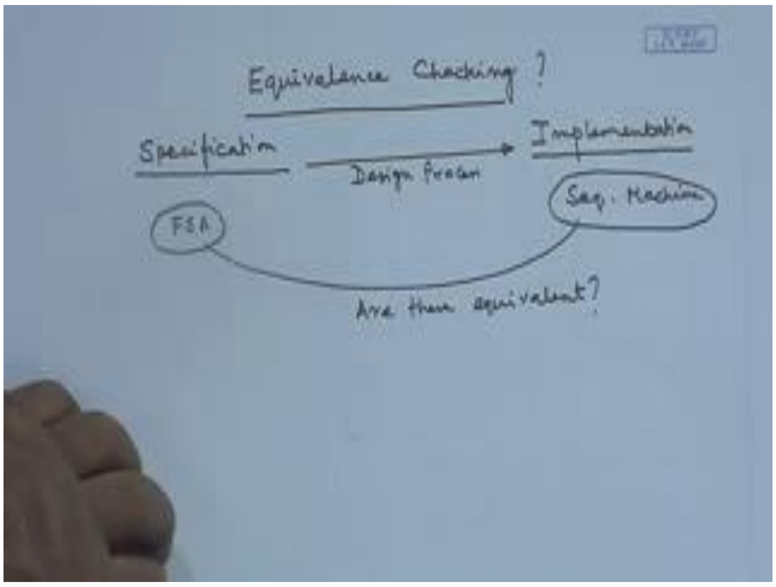
Now, there are different ways in which it can be done the say we can call all these under the umbrella of validating a design. So, we have got a design, and we want to validate that. The different ways it can be done is, one is simulation-based methods. In the simulation-based method, what we do is we run a model - a program a model of the entire system, let us call that that system is the model of the system is S m. We provide and that is that model is executed in a computer. This program that is being run is being fed with specific inputs and we get the outputs. We keep a trace of the outputs, and we also go to check whether these outputs correspond to the correct outputs that were desired with respect to these inputs.

So, for every input there is the necessity there will be some output, and whether the actual output that I have got is the same as the desired output or the ideal output that should have come. So, in that way, if I want to do that, so then I need to specify input test vectors the series of inputs that I give over here are the input test vectors, I am giving one after another like that. So, there will be a series of such 1 0 patterns you can think of, and may be the next one is I will show an example a little later like that a series of vectors, now what is the happening each of these ones and zeros are stimulating different parts of this software. This software is simulating the system.

So, the different paths of execution needs to be excited and each of these different inputs will some I mean will excite or not excite, some paths and ultimately for that particular path that is being excited I get some output. Again for some other pattern of excitement I want to get another output. So, in that way I want to have all the possible paths in this S m covered. Accordingly, we will have to design the input vectors. So, corresponding to the input vectors, there will be output vectors for each of these input vectors, there will be some output vectors. And we check the correspondence between these two, whether this output test vector correctly corresponds to the input test vector so that is the simulation based method which is very popular and in design automation circle or embedded system design.

The other approach is let me first come to the other extreme that is the formal methods. Now, formal methods why do we call it formal, because it takes some formal logic representation; and using that we want to perform certain things through formal means not by running the program and checking whether the output corresponds to the input. We try to prove certain things mathematically; and for that, there are different ways of doing that say one is equivalence checking.

(Refer Slide Time: 06:53)



Let me clearly say what we mean by equivalence checking, what is that. We have got a specification, for example, the specification can be a finite state machine. And ultimately through the design process, we arrive at a particular implementation. So, I had an FSM finite state speci automater may be some FSA, was here and a sequential machine I have got say I have synthesized them. Now, my question is does this implementation actually achieve this finite state this specification, whether the implementation achieves the specification. In order to do that, I want to see whether the behavior of this sequential machine is the same as that of the behavior of the finite state automata.

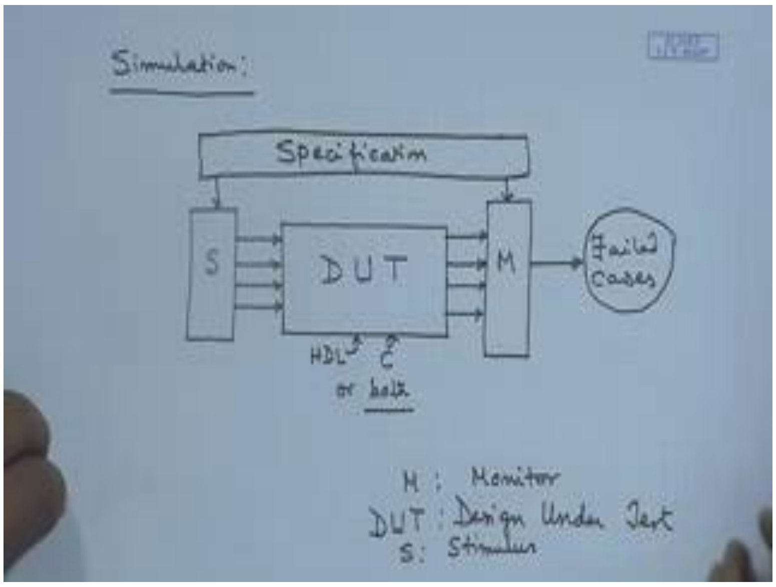
So, this and this I want to see are these two specifications equivalent? Formally I want to see, contrasted with the simulation method where I was giving all possible inputs and was getting the different outputs. And from there, I wanted to see whether the outputs are corresponding to the expected outputs for the different inputs that is one way I am doing it one after another with all possible excitations being given in the circuit. Here we want to formally see that the description for that obviously, the sequential machine has to be described in some form of language, which may be another finite state machine. And the specification is also given in some language.

Similarly, we can see the specification and a program. So, for software we can see that whether our program actually program is an implementation of a specification. So, whether a program actually achieves the specification by checking the equivalence, so equivalence checking is one of the ways of doing it. There are other ways, we will see that like model checking, and there is a more rigorous way, but more expensive is theorem proving. So, these are typically the three different ways in which formal methods of validating a design or verifying a design, this can be also called with the verification methods. All these, these are the three different ways of doing it

In between these two, in between these two methods, we have got semi formal methods, where we take recourse to specifying the input and output input and output specified as symbolic expressions and we check the simulated output with respect to these expressions. So, the desired inputs and outputs are given as symbols, symbolic expression may be a predicate, a behavior an assertion all right some sort of proposition or predicate may be in different logic format like temporal logic or normal propositional logic. And we try to see we simulate not the exact model of the machine, but a logical version of the machine and then we see whether, so that is a symbolic simulation it is called.

Then we try to see whether the specified symbolic expression that symbolic expression that depicts the input and the one that is being transferred, so there is an expected output given in the form of a symbol. And when we give the input and process through the symbolic model, we come to an output, and we check whether these output corroborates to the expected output. However, we will be concentrating on these two in this course; this can be done later on also.

(Refer Slide Time: 13:17)



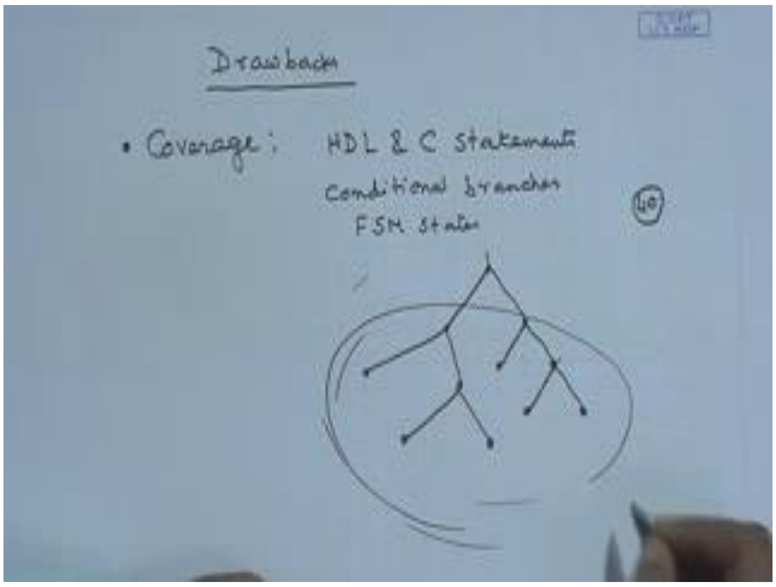
Now, today we will be first talking about the simulation methods. So, the task of simulation, since it is very popular is to create first of all we should consider this D U T. D U T stands for design under test. So, I am trying to test this. So, I have got some stimulus generator S, and this monitor M; S is stimulus generator and M is the output monitor. And we are feeding the specification to all these to this and this, what I this specification will generate the stimulus. And this stimuli will be fed to the design under test. And correspondingly this design under test, the simulated model of the design under test will generate the monitor.

The thing is like this that I take the entire design I have got the design and before I float the design I make it available in the market I run it in my lab with all possible inputs, and see whether all for all possible inputs now whether it gives the correct output or not. Now, in a simulated version I am not actually making the thing in hardware and encapsulating that making a box, before that I am making a software model of the whole thing and running it in a simulated environment. Now, the key word that I said here was all possible inputs. What do I mean by all possible inputs, for example, there may be a we will see that, there may be a number of paths some of the paths may never be traversed also. So, how can I ensure that I have generated large enough inputs, so that all possible paths have been traversed that is the key question here.

However, we will come to that, but this specification is can be done in the form of natural language most ideally, but unfortunately you know that natural language remains incomplete and an ambiguous at times. Therefore, we have seen different specification languages, which will be fitting in here. And that will be creating the interesting stimuli. I am saying interesting stimuli, because the stimuli should be such that it will look at very interesting cases for interesting combinations, whether may be at times I may be judicious enough to decide that I will not create all possible inputs I do not want to traverse all possible paths, I will just see the critical paths may be, because most of the paths own talker very frequently.

So, I can take that risk at the gain of the simulation time. So, many such judgments can be done. And we get the outputs, and these outputs will be monitored and ultimately these outputs should tell me the failed cases that you see here for this particular input the output was wrong. So, you can now look at the field cases, and reexamine this design that is the overall idea of simulation fine.

(Refer Slide Time: 18:21)



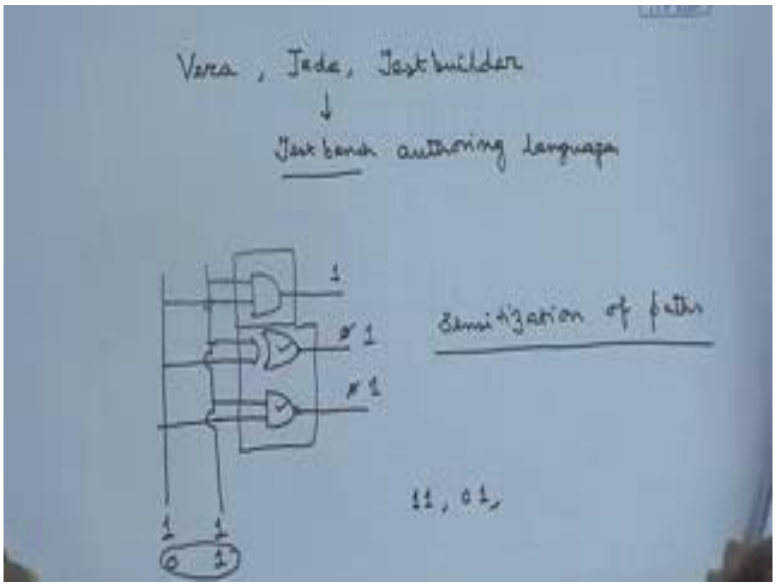
Drawbacks

• Coverage: HDL & C Statements
Conditional branches
FSM states

Now, what are the main drawbacks of this simulation approach? The main drawback of this simulation approach is first of all coverage. First drawback is the coverage. Coverage means now one thing that I should have told here, that this design under test can be specified here in the form of say this can be an HDL specified under HDL C or both of them. Why both, we will see that because we are talking of hardware software code design where we may prefer to write the hardware part in HDL and the software part in C. So, therefore, it can be both all right. Now, given this we have got HDL statements, may be where there will be some conditional branches, signals may be FSM states if I write in the form of an FSM. So, if I have got 40 flip flops in circuit say then how many possible states can be there to raise 40 that is a huge segment. Now, I will have to traverse all possible paths say at this point with some input, I come here; with some other output, I input I come there.

So, in that way, all the possibilities that will be there all these branches that I am drawing are the possibilities for different inputs different states I am arriving at from different inputs, and this will be huge. Therefore, in order to cover them that becomes a real coverage problem. How do I ensure that I have generated all possible inputs that is covering all the outputs, if I want to do that then the size of the input set will be very huge. And more than the size of the set, it becomes often very time consuming to run the simulation. You say that if we run a simulation of a circuit at a gate level with all details it can take million, millennium. So, I mean when the real time takes one day, it takes millenniums like that.

(Refer Slide Time: 21:54)



So, in order to support this, there are some test bench authoring languages by which we can write how to generate the input set vector, I am just mentioning some of them Vera is one such language, Jeda is another such language, test builder is another such language. Now, all these are state test bench authoring languages. What are test bench, test bench is a set of inputs that I have to generate to get the desired coverage. The coverage thing can be, now how can we improve on that. We can have different we can think of different means of improving on this stimulus generation. We can also see whether I mean we can take make interesting decisions regarding how do I minimize the stimulus set, so that I can traverse and cover the most critical path where I cannot allow any failure. In a complex design there may be some had hazard, I mean critical path, and there will be some even if they fail the performance may fall, but will not be catastrophic. In that case I may afford I may decide to afford to leave those out.

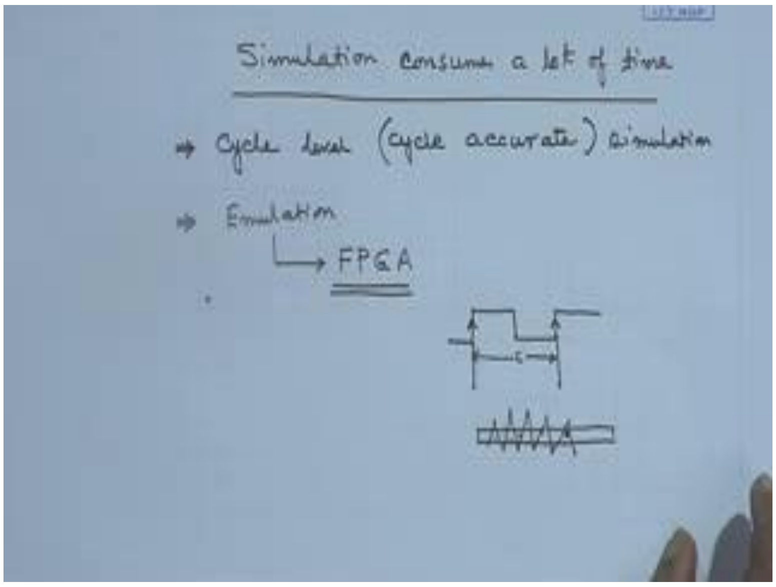
Now, how can I generate such stimuli that is a very interesting and challenging problem. Just to give an idea of what I mean by coverage let us look at this that suppose I have got an AND gate here, I have got an OR gate here exclusive OR gate here. I am just showing you in the form of hardware, but it is also true for software because software is nothing but implementing the logic. Now, all of these inputs are coming to different buses. So, this is one and this is another.

Now, suppose at this any of these can go back right any of these case, now if I put 1 1 as the input vector what will it cover 1 1 will excite this, and I will get a 1 here. So, if it is testing whether the AND gate is correct or not. But if I look at this 1 1 is giving a 0 here, but that does not tell me whether this is all right or not because a 0 0 would have also given a 0 here, it may be stuck at 0 also, I do not know the about that 1 1 will also give a 0 here. So, I could not excite this as well.

On the other hand, if after feeding 1 1, I feed 0 1, 1 1 will cover this thing 0 1 input if I give then this will become 1, and I can say that this is all right; and this will also become 1, I can say this is all right. So, this is the input 0 1 is the input 0 1 is the input that is sensitizing this path to see whether it is all right or not. So, this is sensitization of path. I have to find the input vector such that all the paths are sensitized now.

So, in order to sensitized all the paths, what should be my input vector set, what should be my set of stimuli it should be 1 1 and also 0 1, these two, only for these path three paths you see I needed two right. In that way there is a two simpler case, if I do both of them then both all these things are covered right. Now, only for this small circuit; so for a more complicated case, it will be even more complicated.

(Refer Slide Time: 27:03)

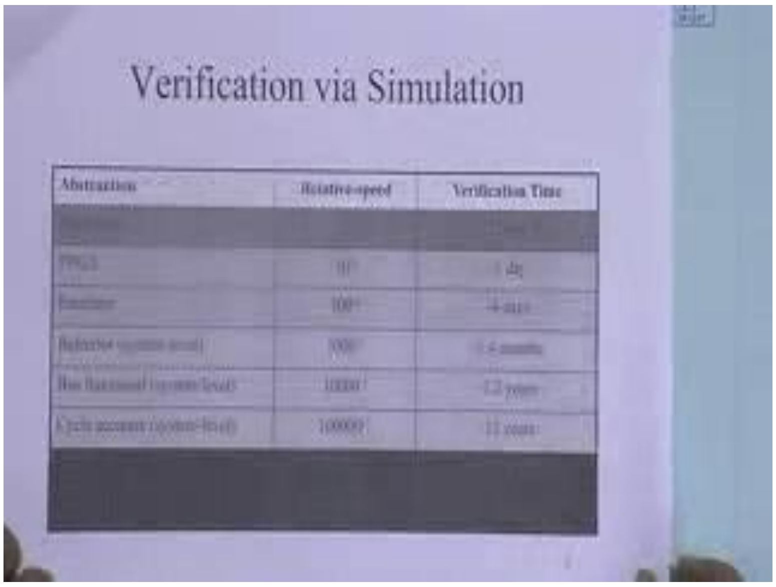


Simulation consume a lot of time

- Cycle level (cycle accurate) simulation
- Emulation
  - → FPGA

Next, how can we one major problem is of simulation is the speed. Simulation is not fast simulation takes time, let me say not that verification and other approaches do not take time, but simulation of a complex circuit consumes a lot of time.

(Refer Slide Time: 27:38)



Verification via Simulation

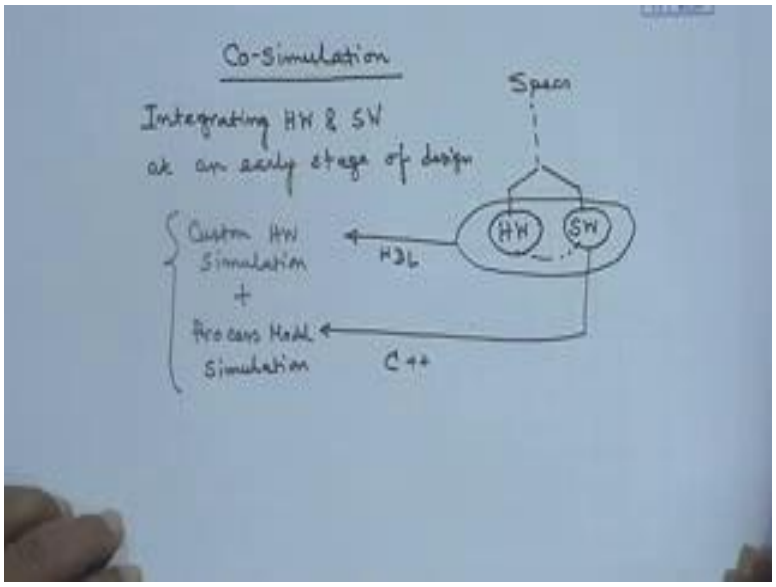
Abstraction	Relative-speed	Verification Time
[illegible]		[illegible]
FPGA	10	1 day
Emulator	100	4 days
Behavior (system level)	1000	1.4 months
Bus functional (system level)	10000	1.2 years
Cycle accurate (system-level)	100000	11 years

Typically, I can just show you one example I do not know whether it will be visible or not. See, if I take, is it clear? Say, if a circuit if simulated by an FPGA takes time of about 1 day. If I go on want to do a system level behavior it will take 1000 times more time, so it will take around 1.4 months. System level, if I just do the bus functional level, I look at the buses also what are the communications that are going through the buses all those things in my model it will take 1.2 years. If I go further down to the cycle accurate system level and every cycle it is accurate or not, then relatively you see it will take around 12 years and the thing that hazardous think that I was talking about is not visible here at the same scale and RTL will take 1 lifetime. And a gate level will take a million year, so that is the way the simulation speeds vary.

So, how can we speed it up? Practically speaking we do not go for event based simulation we try to restrict ourselves often in order to make the simulation faster, cycle level simulation or we call it cycle accurate simulation. Because say within a cycle if I have a, so I will just test at the clocks here I will check, here I will check. In between these, see if I suppose I consider a bus here, even in between this time say in this time, let me call it c - a cycle, lot of data is going through this bus. I am not actually looking at all these information that is flowing here, I will test it here I will next run I will check do it here. The simulation is restricted to the cycle level in between whatever events take place I am not going to do that. So, while event based simulation becomes even slower, cycle level cycle accurate simulation may lead you to a better situation.

The other way we try to do is emulation. The entire thing we try to emulate on some hardware. And the best possible tool that we have we already know that we can simulate the whole thing or emulate the whole thing on an FPGA wherever possible we do that on FPGA, and the FPGA will run faster than this software. So, these are two popular ways by which we can try to make the simulation faster. Next, we will briefly talk about what we mean by since we are working on embedded systems and embedded system is not hardware alone.

(Refer Slide Time: 31:47)

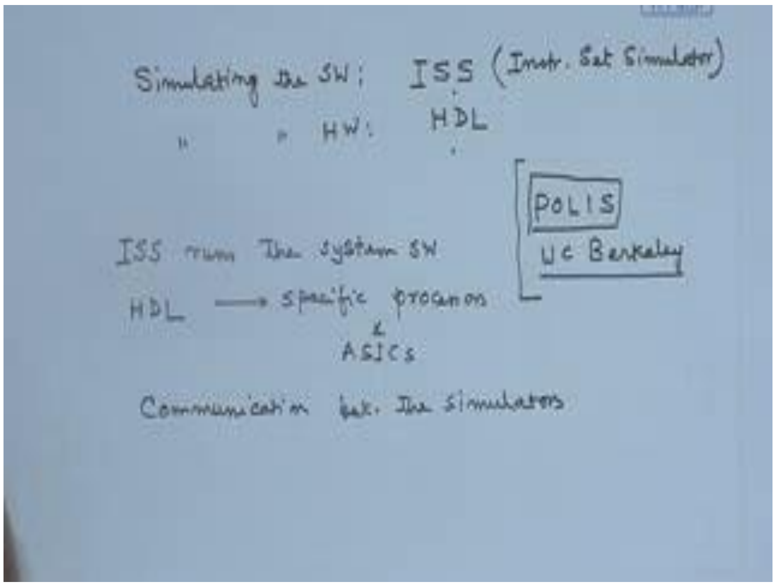


So, we are talking about hardware-software co-simulation. What is co-simulation is it clear? Let me write it here. I have designed, I had the specs and from the specs two different processes I have designed the hardware, I have designed the software. And during our discussions about hardware-software partitioning, we also have seen that the hardware and the software depend on each other. So, if I test the hardware separately, and the software separately, I can meet some problem points like for example, the simplest possible example that I can give is in the hardware you have got an 8235 programmable timer.

How can you test whether your operation is being done? Unless you know that it has been initialized properly at the correct time, or for example, some interrupts all right whether the interrupt service routine it is working properly all right or the interrupt controller has been programmed properly. So, all those things come into picture. So, we need to simulate both of them together, so that is known as hardware-software co-simulation.

The way we do it is traditionally what was done is software was simulated first, I am sorry software was traditionally tested, after the hardware has been fabricated. I fabricate the hardware and then run the software on this and see, so that takes a long time to market. But we are trying to do this simulation, so that we can test it even before the manufacturing therefore, integrating the hardware and software at an early stage is required; at an early stage of design, we want to do. So, that means, now this means that I will have to do for the hardware I will have to do a custom hardware may be hardware simulation plus for the software I will have to do a process model simulation. Both of these I have to do together. The hardware is being done through some HDL may be described and the software I have to may be in C++ maybe I can do it through that.

(Refer Slide Time: 36:03)



Simulating the SW: ISS (Instr. Set Simulator)

" " HW: HDL

POLIS

UC Berkeley

ISS runs the system SW

HDL $\longrightarrow$ specific processors & ASICs

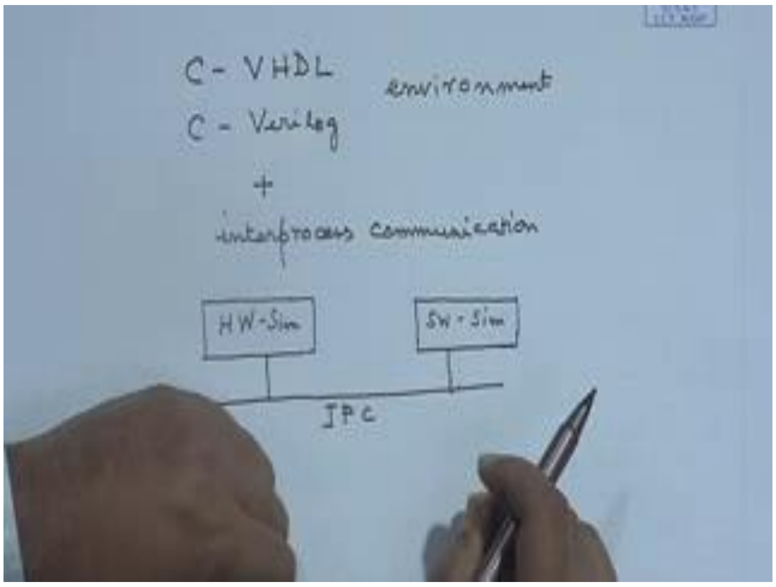
Communication bet. the simulators

Now, so for the software we actually need the instruction set. So, for the software simulating, the software typically we simulate we take ISS, which is instruction set simulator. And the simulating the hardware will be I have got HDL, say for example, if I describe it in very long I can immediately simulate it. So, we have to merge, if I want to do both of them in the same platform by co-simulation tool should have the provision of capturing the ISS as well as HDL. So, that is the main challenge in co-simulation, there are a number of systems which have been developed for this. One of them is POLIS system you can see that that is by University of California, Berkeley, UC Berkeley has developed this for those of you are interested you can look at POLIS at the UC Berkeley site, it is a very nice tool which has integrated all these and integrating all the models.

Now, the ISS model of the microprocessor runs the system software. So, ISS system software means the software that I have designed for the embedded system all right even the microprocessor can be simulated because the microprocessor can be represented using HDL. Whenever we are selecting a microprocessor say 8051, we have got the

VHDL code for that, so that code is coming as an HDL, HDL can this one can lead to specific processors and ASICS. What is left the thing that is left is the communication. So, I have got a two simulators. So, I need a communication between the simulators. So, I can try to arrive at a scenario, where I can have a heterogeneous co-simulation environment.

(Refer Slide Time: 39:23)



Where I will have a C- VHDL or a C-veri log type of environment simulation environment; just to give an example that the software has been written C and along with that, so that is an environment plus we can say remote process communication we can use some inter process plus some inter process communication that we have already seen that we can do using message passing and other means. So, basically we can think of that here is a hardware simulator here is a software simulator, and they have got some I P C - inter process communication. And they are communicating among themselves and this entire thing gives me the co-simulation environment, so that is for simulations which are one of the ways by which we can validate our embedded system.

In the next class, we will look at how we can adopt the formal methods of achieving such validation.