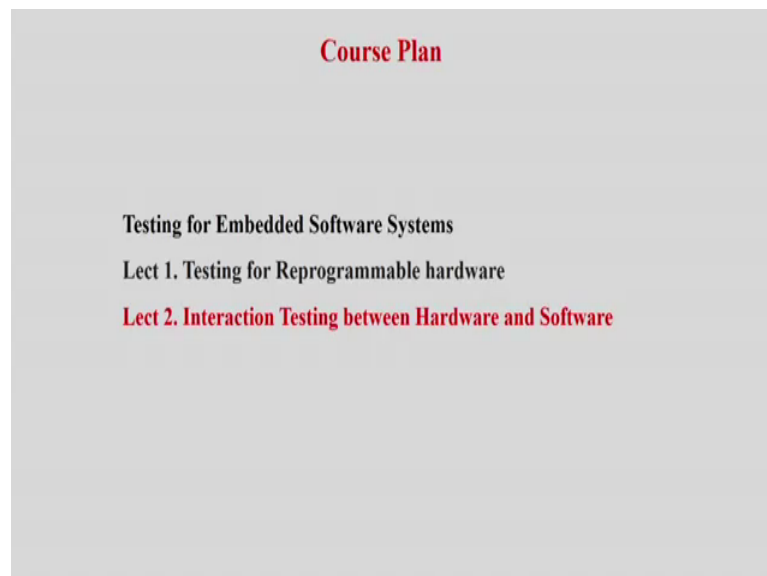


**Embedded Systems – Design Verification and Test**  
**Dr. Santosh Biswas**  
**Prof. Jatindra Kumar Deka**  
**Dr. Arnab Sarkar**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Guwahati**

**Lecture - 37**  
**Interaction Testing between Hardware and Software**

So, hello everybody; today, we are I have come to the last lecture on the course of Embedded System - Design Verification and Test. And we are in the last part of the design as well as today we are in the last lecture of this course.

(Refer Slide Time: 00:37)



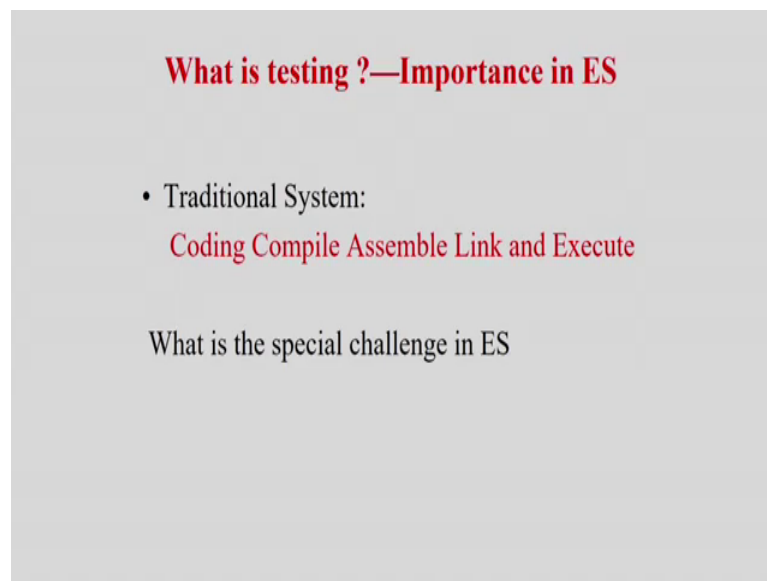
So, we throughout a long journey, we have covered all the basics about design, then we have studied about verification and finally, we have learnt of lots about testing of Embedded System. Today, basically in the last lecture, we are going to cover a very futuristic topic and in fact, in this part in or in this paradigm and a very little very little work has been done and in fact, there are lot of research challenges and open problems that exist in this part which we called as the interaction testing between hardware and software.

So, from the term itself it seems that there is a hardware paradigm to the software paradigm; how you can integrate and test. So, in this lecture rather going into any

particular solution, we will try to look at the problem; what are the issues and how tentatively people are trying to solve and even if people are trying to solve the problem, what is so challenging about it. So, what is testing and what is the importance in an embedded system? Basically why is it, what is the integration testing rather basically?

So, the traditional system on software basically is Coding, Compile, Assembling, Link and Execute. That is normally you have a PC or your computing environment, you write a code and basically you see whether the code is proper or not; you do a lot of software verification and test on that which already we have been covered by Professor Dekha in when we are discussing on the verification. But what is the special challenge in an embedded system?

(Refer Slide Time: 01:55)



**What is testing?—Importance in ES**

- Traditional System:  
Coding Compile Assemble Link and Execute

What is the special challenge in ES

So, the problem mainly is the hardware platform is not available in the embedded system directly right. If it is a normal computing system, I have a laptop like this. I can write my code and I know that the laptop hardware or the computing hardware is not having any faults. So, if there is any problem etcetera, it is mainly with my software. So, basically I can do a lot of verification of the software like Black Box Testing, White Box Testing and Fault coverage analysis and so, many stuff which already we have discussed at length in the module in the second part of the course.

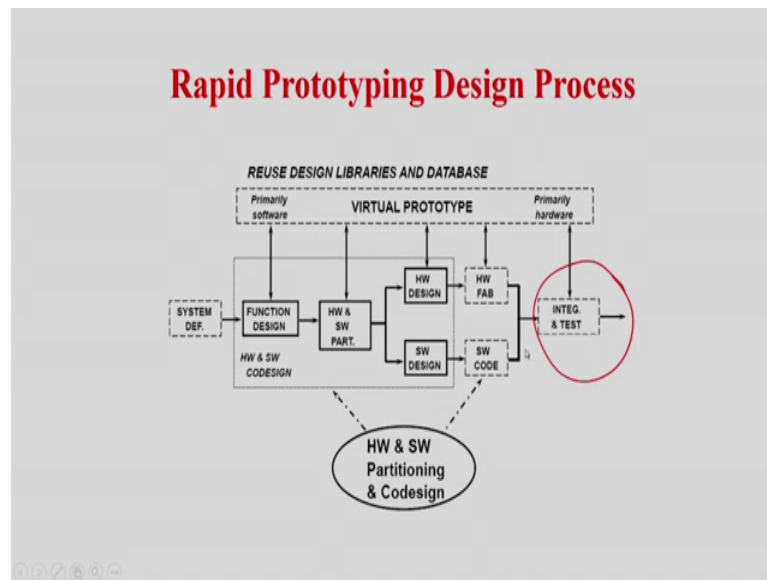
But in an Embedded System, the special challenge is the laptop given to us is not yet ready or they say that a similar version of hardware will be available to you which will be very much application specific, you know the embedded system is totally application specific.

And in fact, the hardware may not be available to you in the beginning. So, it is a kind of a stuff that 2 people have started the journey and or it is making a tunnel through 2 parts of the hill. So, one tunnel will come from this side, another tunnel will come from this side and they should meet that is what is the integration testing and in fact, it is a very very difficult problem that if the slight mistake in any of the tunnel drilling problem, the two tunnels will pass and they will never meet in the end.

And historically many many such incidences have happened in the engineering of tunnel drilling. So, simply it is very synonymous problem here also. A group of people will make the hardware; a group of people will write the software. Knowing that such a hardware will be available to me in future. So, you can understand how difficult the problem is and the hardware people will make a hardware, understanding that there will be some code which will be available to me in some point down the line which we have to work on my hardware.

Now, we can ask me the question, then let me first make the hardware and then, I can ask the software people to develop the software. If you do this, there will be longer time to market of the product and you will lose the market business. So, you cannot do that. So, both the teams will start drilling and finally, they have to meet it. So, that is actually the special problem of embedded system, compared to any traditional system where the hardware is built a priori.

(Refer Slide Time: 03:42)



So, this is what is the idea. So, basically what is having we have a system definition, then we go for a hardware software codesign that is partition.

There is a function design and then we partition what has to go in a hardware and what we have to go in software. So, there will be a hardware design, there will be a software design. Hardware is fabricated. There is a software code which is developed and interestingly, the hardware fabrication people have to think in mind that such a code will execute on my on my hardware. I have to design like that the software people have to think in mind that such a hardware is available and the code should run.

So, you can understand the difficulty basically. So, both of them have some picture in mind and they try to develop. Finally, hardware is tested on its own as in the last third model which I have taught you. Software is tested is testing and verified on its own methodology. Finally, they integrate and test. So, practically speaking this last phase although it is very very important, but practically before time to market we have a very very little time to put on that. And in fact, it is a very difficult problem. So, people have found that the most of the bugs which still remains in the hardware and software paradigm are because of this lack of integration testing and available paradigm.

Now, you can ask me what I can able to do it? So, basically can I have something called a preliminary software; can I have a preliminary hardware, where I can do some kind of a integrate test from the beginning? The answer is very very difficult. Preliminary

software still can be developed; you can say that I will write some basic blocks and maybe I will write some software which will work for only a very few of the inputs, but how can you have a temporary hardware.

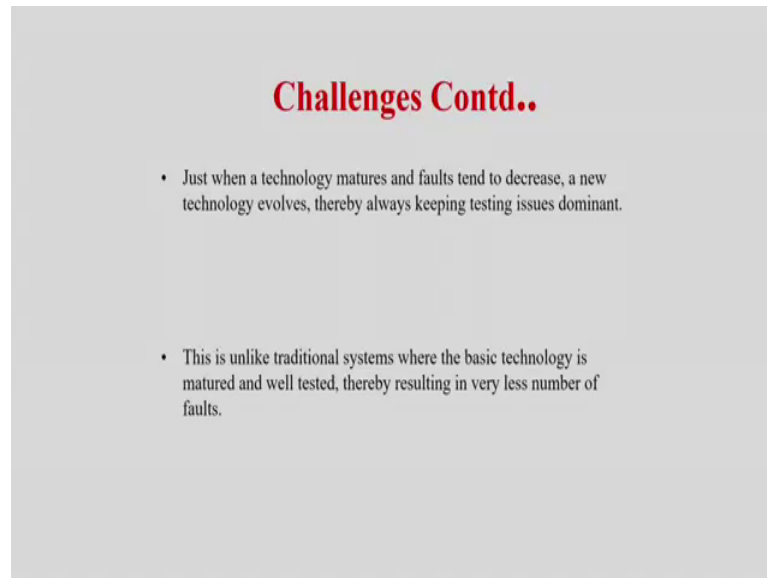
So, if you have a temporary hardware, at least with the advent of FPGs and CPLD devices which you have discussed in the last lecture; slight research has been going on to this direction, that if there is a prototype device where I can download my hardware and then I can still give you a hardware platform to download the software and go for partial amount of integration testing.

But still is a very nascent stage because as I told you in a general purpose or in general sense the hardware are quite involved and you require may require lot of FPGs to download a real hardware which was to be fabricated in a ASIC because nowadays it is true that we have very large size FPGs, but for many practical applications multiple FPGs has to be used to design which is equivalent to a single ASIC.

Because as we have seen that FPGs are basically some kind of general purpose hardware; so, the difference between the advantage of a FPGs is that you can very quickly have a prototype. But the area will be higher and at speed testing cannot be done because FPGs are generally slightly slower than your contemporary ASIC designs.

But still some work has been done or people are trying to enumerate or I mean they make a prototype hardware and download the software and try iterating it debug the bugs easier so that when you read to the FAB and software code, many of the design handshakes has been done. But again not it is a very very challenging problem.

(Refer Slide Time: 06:18)



**Challenges Contd..**

- Just when a technology matures and faults tend to decrease, a new technology evolves, thereby always keeping testing issues dominant.
- This is unlike traditional systems where the basic technology is matured and well tested, thereby resulting in very less number of faults.

So, basically why all these problems happen because it is first class only I was telling you, when a technology matures fault tend to decrease, a new technology evolves. Therefore, always the testing issue or verification issue comes into picture. Because if I was fixed hardware which is a very well tested like in a normal computing platform like a server, like the laptop which is already well tested.

So, if I write a software on this, then I can concentrate on debugging my software, but in embedded system the it is always going toward cut during the cottage technology. So, always we target that a new processor or new embedded processor will be coming into picture and newer embedded hardware will come into picture. So, my code should run on that. So, in fact, that high end hardware is not available to me as of yet and for the hardware designers, neither the fabrication technology known a priori neither the software which will run on in unknown a priori.

So, basically everything is very futuristic; everybody has something in mind that that the target they should meet and they all are rushing towards something like to meet some goal for which the technology is not yet matured. So, if you have such a you a futuristic goal and you are trying to match it, the problem is that you will have tend to have lot of bugs and you have lot of faults which may be in place. But what is the solution? You have to debug, you have to go for a integrated testing and you have to make maybe you can go for some kind of prototype develop and preliminary level of test.

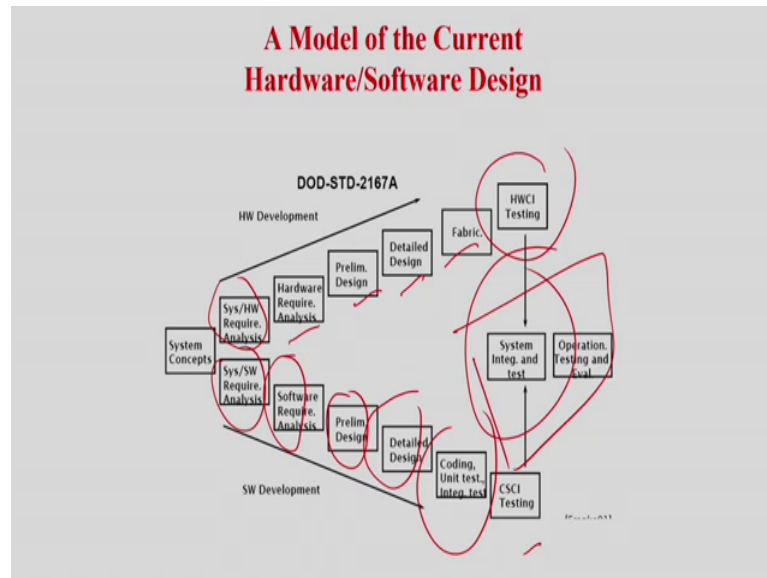
But you cannot see that I will have a predetermined hardware and I will write a code for that. If you do that, you will lack the market speed because the all your competitors are making a futuristic hardware assumption, writing the software. The hardware people are having a very futuristic software application in mind, writing the hardware for that and finally, they are going to meet. So, you cannot lag behind anybody.

So, therefore, we should have a futuristic fault model into picture. We should have a prototyping device and we should start aiming that something will be available down the line and all our design should be targeted towards that. So, this is the unlike any traditional system with the basic technology is mature and well tested resulting in very less number of faults or bugs. So, in many of the cases the hardware platforms are mostly well established.

And there is a traditional software development. The hardware platform is well known; only the bugs are discovered in the software. But in hardware nothing is available everything is in the air. So, their target hardware target software both are developing; finally, they should meet and they therefore, there are lot of bugs which can be found out when you are going for an integrated test, but still this is the need of the hour.

So, you should have very good technologies to go for this integrated test, we should start from the beginning of the die; not basically you should not target here. We should rather target something like where we can do our prototype design verification much before the final phase like something like this.

(Refer Slide Time: 08:46)

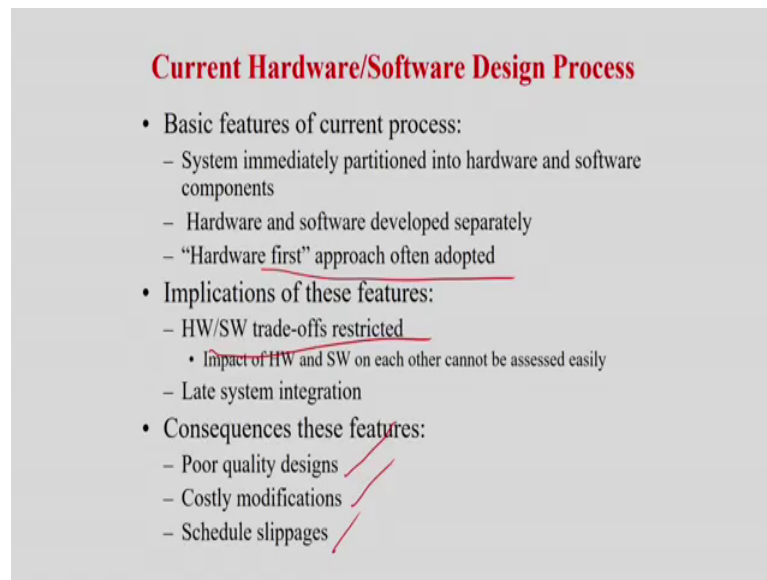


So, basically it is some kind of a hardware is the current model of hardware software co design. So, you have concepts Software Analysis, Hardware Requirement Analysis. So, basically is your hardware development. So, you see you partition into hardware, then you have Requirement Analysis, Preliminary Design, Detailed Design, Fabrication and finally, the Hardware Test. Software means what part you are going to put in software? Software Design Analysis, Preliminary Design, Detailed Design, Unit Coding Integrated Testing, Black box testing, White box testing; finally, we have the code and finally, we will integrate and we are going to do it.

So, you see that these final level of test is only towards the end which makes our life very very problematic.



(Refer Slide Time: 09:21)



**Current Hardware/Software Design Process**

- Basic features of current process:
  - System immediately partitioned into hardware and software components
  - Hardware and software developed separately
  - “Hardware first” approach often adopted
- Implications of these features:
  - HW/SW trade-offs restricted
    - Impact of HW and SW on each other cannot be assessed easily
  - Late system integration
- Consequences these features:
  - Poor quality designs
  - Costly modifications
  - Schedule slippages

So, what is the problem? System immediately partition into hardware and software. After sometime down the line you might have find it ok, this part I should have moved in software or this part of the software I could have moved in hardware, then I would have been a better design; very difficult to do go in this straight flow of design. There is no iteration back.

If you see as we would not have any iteration back, then from some part I can again change the hardware software partitioning. That people do not do because the partition, then the two teams start doing their job interchanging hands is a very very difficult problem there. So, that is one of the main issues. Then, hardware and software are developed separately hardware first approach is often adopted that is what I am saying, but still it is not very good. And if you first fabricate the hardware targeting that that software will come and execute with me and then, you employ the software and see everything is there.

But it will make this, this is actually a practicality even if many of the industries may tell you that we are having co development phase and all, but still it is practical, but still it hits the market because if you have make a hardware and the software team starts the job later. Then you can find out that we will lose time to market. So, it is always people try to do it parallel.

But still for the more most of many of the practical applications, hardware are developed first and tested; software is downloaded and seen whether with the assumption that the hardware paradigm is correct which is the most of the normal computing platforms, they follow these approach, hardware first. But for embedded system industry practically many of the industries follow hardware first approach, but this will lead to a situation when will be quite bear in others market. Like this story I always tell you about Nokia and some other phones like Samsung.

So, I mean this is just some example should not go about the exact vendor and for just an example I am telling you. This is does is nothing to do with favoring Nokia or Samsung in general. But the idea is that Nokia wants to give a very nice robust mobile phones; Samsung on the other hand started giving lot of features, but initially there they have much more bugs or robustness compared to a Nokia.

But, who won the market? Samsung, the smart phones. Why? Because they were have futuristic development ideas that this is my hardware available, this is the GUI development. So, all started in parallel and finally, they met. But if you do such an approach, then it is true that there will be some bugs and fold might have some problems, but Nokia was following the hardware, full verification kind of a method. So, that all their phones are very very robust and they were very less bugs, but still who won the market? Who could be more features with little amount of bugs because with the handheld device generally for home usage or a non critical usage. So, we find bugs I can tolerate the design, but I need lot of features.

So, that is just the idea of embedded systems. For if this is the slightly different for mission critical applications, where we have to be super sure that everything is operating fine, but 99 percent of the embedded systems are basically for homemade or home devices or for personnel and non-critical uses. So, the market is very very high there. Your microwave wave microwave and washing machine, mobile phones everything basically is an embedded system nowadays.

So, therefore, people try to go for a parallel approach and try to see basically how much features they can give. In that case the integrated testing and integrated development environment becomes very very important. Another problem is Hardware Software trade-offs are restricted. As I told you, this is hardware, this is software. If I down the

line I find this put should be put in a hardware; this should be put in software, such chances are not available to us. So, that is the problem.

The trade-offs are decided in the beginning and in middle you cannot have a handshake which problems are Poor quality design, Costly modifications, Schedule slippages; that means, Schedule slippage means I cannot meet the target. I was saying that this hardware would be available; then my software would run.

Hardware people have taken more amount of time. So, such type of stuff happens. So, we require co-development and co-integration and co-testing phases of hardware and software in embedded systems.

(Refer Slide Time: 12:52)

**Incorrect Assumptions in Current Hardware/Software Design Process**

- Hardware and software can be acquired separately and independently, with successful and easy integration of the two later
- Hardware problems can be fixed with simple software modifications
- Once operational, software rarely needs modification or maintenance
- Valid and complete software requirements are easy to state and implement in code

But again, as I told you it is a very very difficult problem. So, there are some incorrect assumptions in the current design process. Hardware and software can be acquired separately and independently with success and easy integration in the two later that is what people have a thought that I will make a hardware with the assumption, this is exactly the hardware with what when I am making the hardware, I should have a knowledge that exactly the software would work and finally, they should integrate (Refer Time: 13:10).

But that never happened basically. Hardware problems can be fixed with simple software modifications. Therefore, these are some practical things which people have faced. So,

people thought that if the hardware has some bugs, then I can write the software with tried modification those are these bugs become maskable right. They are not faults; they are bugs basically. So, and people also thought that because software can be modified; hardware cannot be. So, the hardware people are thinking that such a software will be available and I am going to do it. But in the end software hardware had some bugs not faults design errors.

But, because the software is not available with them; so, they cannot directly verify the hardware exactly that how it would perform when the software is executed. So, that is a problem. So, people generally thought that we have a hardware and we write the software, if there are bugs in the software, we will try to hide them. And in fact, many processors or many embedded hardware's have software bugs. The software was tightly tweaked or a operating systems were slightly tweaked. So, that the faults become marks that you do not feel the effect.

. So, even no industry will tell you these stories because that will put a very bad mark on them, but that is the truth. So, once operational software rarely needs modification or maintenance that is also a not true basically; means I mean that is not true basically too much relevant for a embedded system, what people thought is that the hardware is there you dump the software. Then, your job is done and in fact, it is not we require quite updation of the software's because on the same hardware paradigm you can also improve the versions of the software.

So, your hardware should not only target that this is the software that would be downloaded you should be certainly futuristic like I have a module mobile phone and I download a version 1 of the embedded wise. After sometimes, I will I will download some applications. Then, I will also update my always of the hardware frequently. Your hardware should be able to support that.

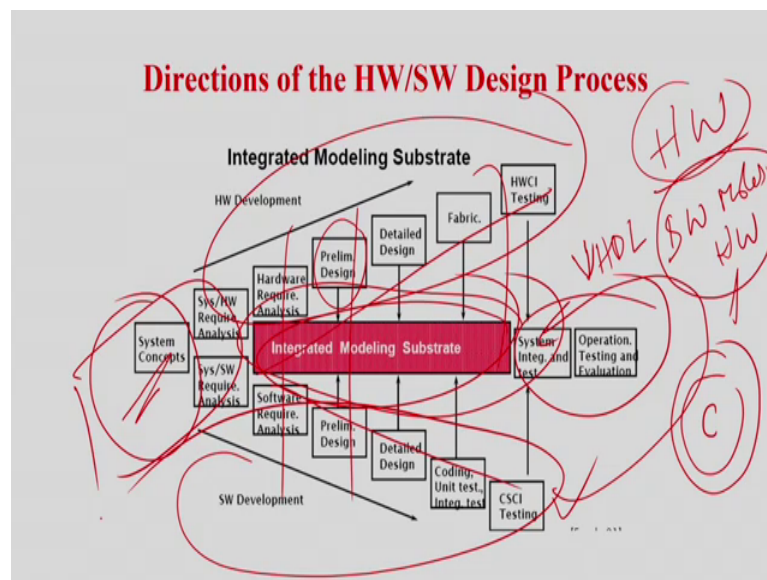
So, previously embedded people thought that this is the hardware developed for this target, software after that there will be no renovation on the hardware is simple because hardware cannot be renovated software will also not be enhanced, but that was not true.

So, you if your phone or your handheld device do not allow, enhancement of the software is a very bad marketing or design policy. So, your hardware should also be futuristic; not only it will be able to download this operating system and operate within,

but you should also see that it should be able to work on version 1, 2 and 3 and so forth. So, because there are a lot of hardware design challenges and requirements, this is basically some of the assumptions which were not true in the embedded domain.

So, basically hardware software cannot be designed so easily, separately. Integration is a very big challenge. Hardware problems can be solved by solving the software from that is true. But with the very modern paradigm, it is very difficult that all hardware problems can be masked with software. Secondly, like some part of the hardware has a problem. So, maybe we will not try to mask access those parts, I mean there may become from the redundant parts will only access them something like that ok.

(Refer Slide Time: 15:51)



So, again basically software rarely needs modification in an embedded system, not true basically because hardware is fixed, software is fixed. In an embedded system, generally it was understood, but in fact, hardware cannot be changed, but a lot of upgraded versions have to be downloaded and executed.

So, you can understand the problem, this is an extremely challenging problem to design a pure new embedded system with a lot of guarantee of success, a lot of tradeoffs to be given because two people are starting there if in between something, we find that this has to be changed to hardware from software, this can be mapped from software to hardware.

But I am not able to provide that because we are calling you in a directional way. In direction, you may find lot of bugs. In that case people cannot modify the hardware; they will try to tweak the hardware environment. So, you can understand if I can somehow make a co development environment, then all my problems will be solved. So, that is what is shown in this slide. So, Integrated Modeling Substrate; so, we require us part which can model your hardware as well as your software. Software modeling is very simple is then it just programming language that is the model of the software.

And if you want to go for a preliminary design, you can go for block level design; then you can slightly make it more you can enrich these blocks with code and so forth and simulating a software is very simple. Main problem is with this software part. The software part is very very difficult sorry hardware the software part is very simple actually hardware part is where the problem is because I can have a hardware in simulation which I call a verilog simulation or all the models which Arnab sir has told you.

But they are all at the hardware level; their one level is hardware simulators, one level is software simulators. Software simulators are as the C language, Java whatever programming language you are using, compilers you are using, you can tell them that is the software simulator. There is nothing that executes your software.

But when I am talking about hardware, the hardware is not available; only I have the software representation of the hardware and the software compilers like HDL verilog compiler, VHDL compilers; they will take the verilog or VHDL code which is representing the hardware and then, they will try to tell you that this is how the software will be having.

But we rarely have some kind of a simulators in which case you can put the software in the hardware which is also in the software version, like this is a hardware pure hardware and we do not have that verilog VHDL. All are software representation or software models of the hardware and we have some say C code or Java code which are which I am going to execute on this hardware.

So, these hardware simulators, this is the pure hardware we have some hardware models which are running in its software like verilog VHDL. So, if I say that VHDL. So, this VHDL basically models this hardware which is written in a soft language, but VHDL

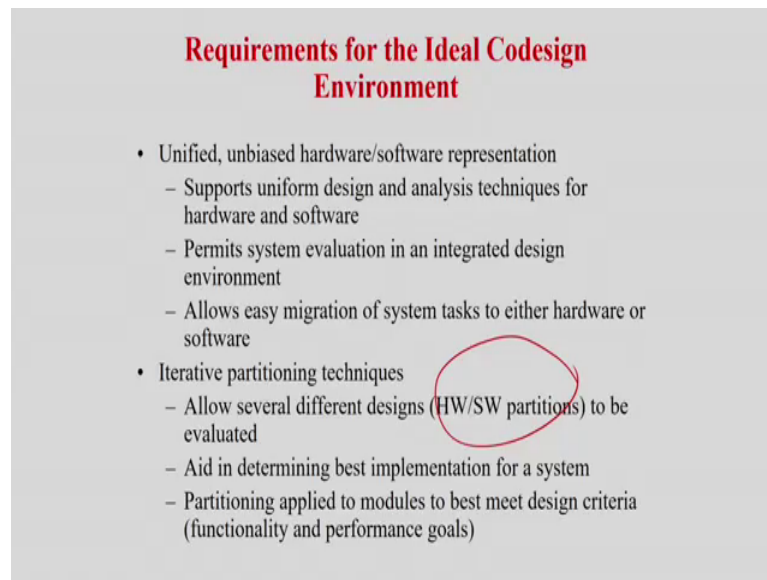
cannot simulate this hardware with this C language that those things are now not available or not even if it is there they have take a long time to simulate. The idea is that I have written a verilog code for an adder. So, it is there verilog, the verilog will the code will simulate and tell you the how the hardware will behave.

But if it is a C program which will access this adder, the verilog simulator cannot simulate that because verilog simulator will simulate the hardware which is written in verilog called VHDL , but if a C program runs on top of that; how that C program will be executed on that hardware there are very difficult very very, such simulators are very very difficult to develop even if its develop it is very very slow which actually takes the hardware represent in verilog or HDL there is a hardware is represent in a hardware definition language.

On top of it there is a C language program or some language program which is running on that and totally on entirety, it is simulated on your some kind of a platform. So, the if that is available, then our problem is solved. So, you some simulators are available, it is not that simulators are not available. But it will take a long time to simulate a large or a practical design because it has to simulate the software. It has to simulate the hardware and how they are integrate that has to be simulated and everything is done in the software paradigm.

So, it is something like one of 1 hardware, on top 1 processor is there on top of that you are putting a another sandbox, on top of that is you are putting another sandbox and you are running multiple operating system on top of one another. So, one OS is simulating another OS, another OS is simulating another. So, OS and it makes things very very slow. I am just giving an analogy, but here the hierarchy is C language on top of it is the hardware. Hardware that is not exactly hardware, this is a hardware written in verilog on HDL and an entirety there should be a simulator which can take the hardware in VHDL and C language which is running on that VHDL version of the hardware and total how it behaves very very difficult to get something like that.

(Refer Slide Time: 20:15)



**Requirements for the Ideal Codesign Environment**

- Unified, unbiased hardware/software representation
  - Supports uniform design and analysis techniques for hardware and software
  - Permits system evaluation in an integrated design environment
  - Allows easy migration of system tasks to either hardware or software
- Iterative partitioning techniques
  - Allow several different designs (HW/SW partitions) to be evaluated
  - Aid in determining best implementation for a system
  - Partitioning applied to modules to best meet design criteria (functionality and performance goals)

So, Requirement for Ideal Co-design Environment; Unified unbiased hardware software representation. So, you can represent in a single environment, both the hardware as well as the program. Supports uniform design and analysis techniques for hardware and software. Permit system evolution in an integrated design environment; allows easy migration of tasks either into hardware and software.

So, if I can give you an environment where I can write the hardware in verilog on VHDL I can write the c program or which has to be running on that hardware same environment. I can code and its simulator what it will do it will take the C language or whatever language, you have written in the soft code, we will take each instruction of that mapping on to the hardware which is still in a soft language like verilog and VHDL and finally, it will tell you how the hardware will behave when such instructions are taken into picture. So, that is actually called unified or co simulation environment that is much challenged to design. In fact, it is much challenging to design a very fast working simulator on that.

So, if it is there, then you can allow different tests like this part in hardware, this part in software; how it will behave? So, basically you can find out if there are bugs that when it points the bugs. I think it is very obvious that if I have a entire simulation, environment from the beginning a single team or people are working in very close hand in hand to find out whether means how the system is behaving.



If this part is a hardware, this part is software; if it is a bug; whether the bug is in part a, or part b or if it is a hardware or in the software? So, maybe in fact, is the entire software given to you a part is for the hardware, a part is for the software, together you are simulating it and it becomes a co development environment, co design environment and life becomes very easy.

But the problem is making such a simulator is difficult; as well as it will become terribly slow to simulate practical designs right. So, that is what is required. Integrated Modeling Substrate.

(Refer Slide Time: 21:57)

**Requirements for the Ideal Codesign Environment (cont.)**

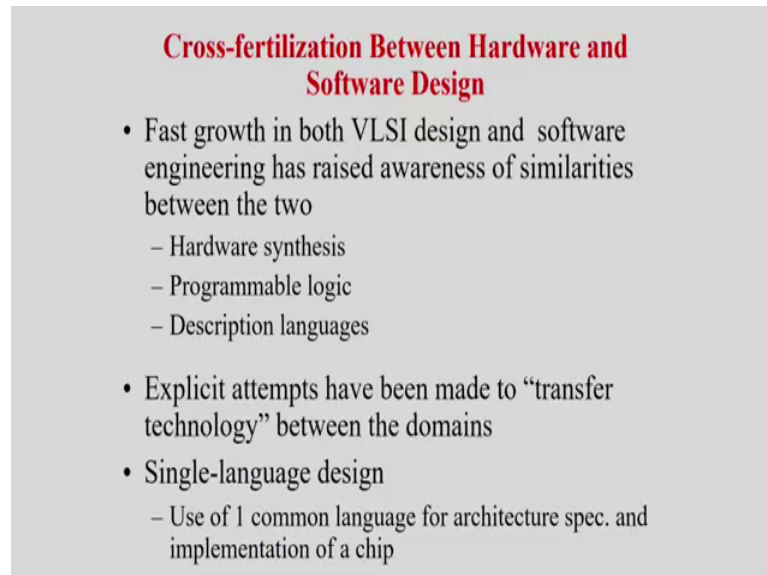
- Integrated modeling substrate
  - Supports evaluation at several stages of the design process
  - Supports step-wise development and integration of hardware and software
- Validation Methodology
  - Insures that system implemented meets initial system requirements

Supports evolution at several stage of the divine design; supports step-wise development and integration of hardware and software. As I was telling you validation ensures that the system implemented means the initial system and final system requirements. So, basically it ensures that the entire system going to be final design will meet the initial system requirement.

So, initial system requirement as you see is not partition into hardware and software, in the initial requirement, now you have a any now if you look at it what you say this is very interesting. So, if you look this is the system requirement right. This part is in hardware and this part in software. So, if you take the traditional way, the whole system concept can again be tested only in the end. When the practical hardware is there you can test it, but here the idea will be basically you have a integrated modeling software. So, at

each point of time basically you can validate the entire system specification which was not possible in the traditional design flow.

(Refer Slide Time: 22:47)



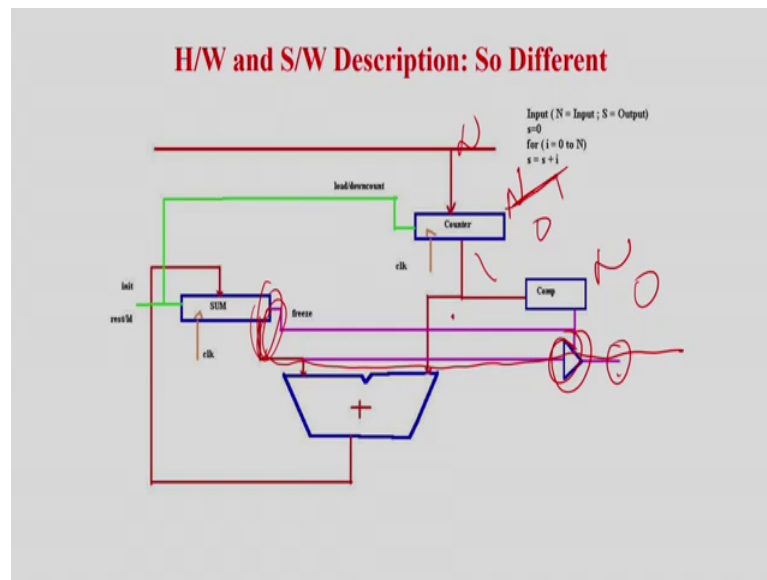
**Cross-fertilization Between Hardware and Software Design**

- Fast growth in both VLSI design and software engineering has raised awareness of similarities between the two
  - Hardware synthesis
  - Programmable logic
  - Description languages
- Explicit attempts have been made to “transfer technology” between the domains
- Single-language design
  - Use of 1 common language for architecture spec. and implementation of a chip

So, that is actually called cross fertilization between hardware and software. So, basically you know what I was telling you is basically is written in this in this slide that mainly we should have a co design environment that is very important, that is cross fertilization. We should have a hardware representation as simulation. We should have a software representation as the simulator.

Both should be able to integrate at the simulation environment and then, you should be able to see the results. This will allow to you to develop test scores much before. This will allow you to find bugs whether in the hardware and software in the beforehand, they can allow you easy tradeoff between the two. So, that you get a very very nice and when a system design environment. There are lot of designs available right.

(Refer Slide Time: 23:24)



Environments available to do it, but they are all at the very what I can say the theoretical level or even if it is the cross hybrid environments are there at the simulation level also, they are very less in numbers and the main drawback is this p and then that is why I am just going to show you a very very simple example. Why it is challenging to build such a simulator. Like for example, let me zoom this code for you. A simple code, what it says input N, output S; s equal to 0 for i equal to 1 to N; s equal to s plus i.

What it does? It iterates and add s equal to 1 2 3 4 5 6 7 8 up to N that is it. So, it is a simple software, but if I write in the hardware for this, if it is the c representation, you can understand it is a a class 12 exercise or just we take first year exercise. But if I write it in terms of verilog, what is the representation? Extremely-extremely difficult. So, this is basically the hardware implementation for the, or the register transfer implementation for this. So, what will have? You will have a adder fine you should have a counter.

So, initially the counter will be loaded with N; then there should be a comparator because every time is a down count. So, every time it will become the N minus 1 up to 0. So, whenever there is a comparator will be comparing with 0. So, whenever it is 0, it will freeze the value and this is going to freeze this one and initially this sum will be reset to 0. So, there should be a reset line which will reset it to 0 right then basically this counter will have the value N. So, in the time first time, the initially it is loaded with N, then basically this is reset to 0 and basically the, this compound has a value 0.

So, I represent 0 and N are uncomparable. So, now, this is a freeze line. So, this is not frozen basically. So, what happens? So, you will have 0 plus N is equal to now it will be N. So, it will keep on; then it will be coming down counter N minus 1. So, sum will be equal to N minus N, N minus 1, N minus 2 and so forth. So, this is a simple RTL design, I am not going into the VLSI design as present, but I am just trying to give you an idea that why it is very difficult to make a co simulation environment.

And it decrements finally, becomes 0. So, when it becomes 0, it will make this tri state as a high. So the, whatever value is in this sum register will be coming out over here; that means, now 0, 1 from N to 0, it has come the counter has come started from N coming to 0. So, at that time the tri state is enabled. So, whatever value is basically retaining over here is coming out and it is also freezing this sum register. It will no more increment or decrement or do any addition.

So, there is a simple typical circuit.

(Refer Slide Time: 25:47)

```
H/W Description: Verilog

module test(in , reset , out , clock);
input reset , clock ;
input [3:0] in ;
output [3:0] out;
reg [3:0] out;
reg [3:0] count;
reg ready;
always @ (posedge clock)
begin
    if (reset)
        begin
            out = 4'b0 ;
            count = in;
            ready = 1'b0;
        end
end
```

So, basically this is the verilog representation. So, you say always are the posedge of clock if reset out is equal to 0 count equal to in count equal to in means it will be loaded with the value N and ready bit is set as one some variables you have set.

(Refer Slide Time: 25:59)

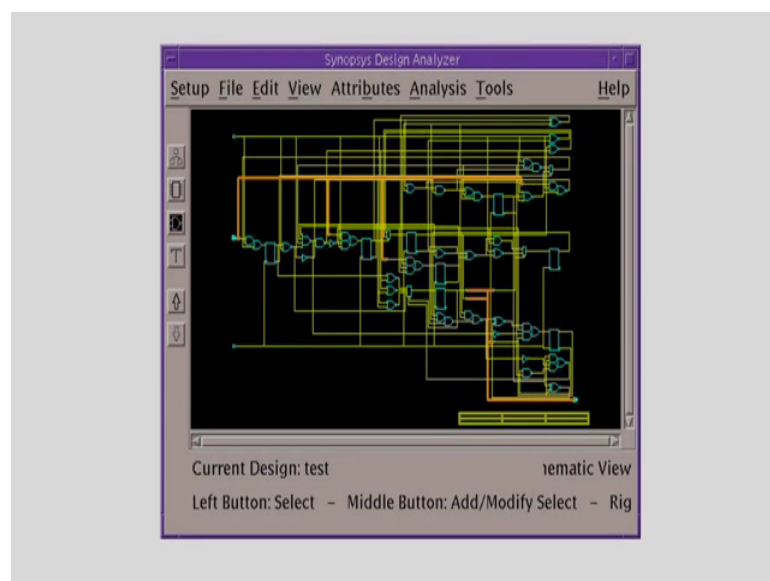
```
H/W Description: Verilog

else
  begin
    if(count == 0)
      ready = 1'b1;
    else
      begin
        out = out + count ;
        count = count -1;
      end
    end
  end
end
endmodule
```

Then, if the reset is not there if count is equal to 0, ready bit equal to 1. Count is equal to 0 means the counter has been set to you make the ready bit equal to 1; ready bit means it will be freezing and sending the value to the tri state else out equal to out plus 1, count equal to count minus 1.

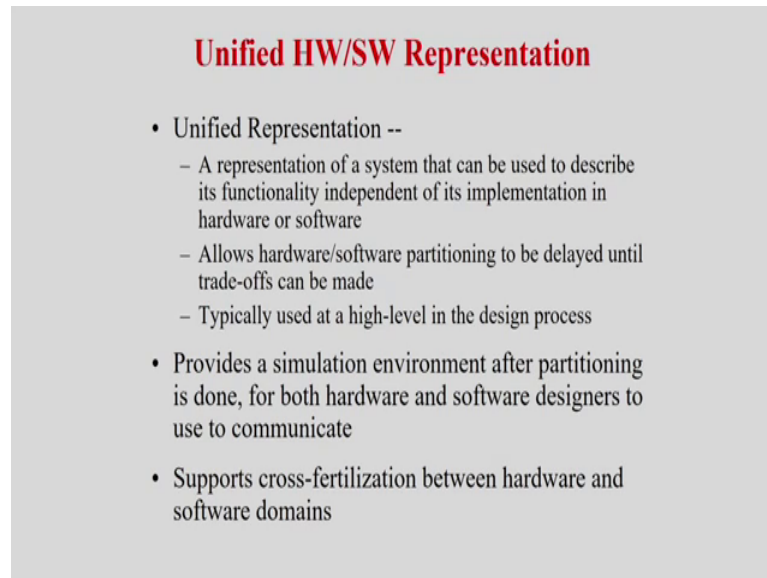
So, you can see there is a clock edge, there are registers, there is an initialization code, there is basically a incrementing code and so forth. So, this is in a verilog representation.

(Refer Slide Time: 26:26)



And finally, this is your it will be synthesized as a hardware like this and so forth. So, what I try to mean by this? I am not going to teach you I am not teaching verilog or so hardware design in this key in the slides and then, you see the difference. So, this is the software and this is basically hardware representation for a, same design intent. So, such difference in the hardware and software description.

(Refer Slide Time: 26:47)



**Unified HW/SW Representation**

- Unified Representation --
  - A representation of a system that can be used to describe its functionality independent of its implementation in hardware or software
  - Allows hardware/software partitioning to be delayed until trade-offs can be made
  - Typically used at a high-level in the design process
- Provides a simulation environment after partitioning is done, for both hardware and software designers to use to communicate
- Supports cross-fertilization between hardware and software domains

So, it is very very difficult to make a uniform coding environment or uniform simulation environment. True, but still people have overcome these lot of theories has been developed as I told you this is a very very futuristic research problem and people have really tried to develop environments where even if it is a hardware or software still how they can be simulated together. Just this was just to motivate the challenge that why it is difficult to have such a co different co simulation environment right.

So, that is what is very very important. So, if I somehow can do it, then what is the idea? So, if I can somehow have a unified software hardware representation. So, a representation of that system can be used to describe its functionality, independence of its implementation in a hardware and software in a we have a initially we have a very integrated environment. Then, you can software partitioning you can delay till a long time that I am trying to say integrated environment, I am trying to go right and even when I am going in the final stage. Then, you can have a more clearer picture that which should go in hardware and which can go in software.

Because I have a integrated environment. So, I can try all this and towards the end even more or less all the design components are clear, I can do a partitioning which will give me a better control for efficiency right. So, that supports cross fertilization between a hardware and software provides and simulation environment after partitioning is done. So, that means, is a the in a in a nutshell if such an environment is available to you, most of your embedded system solutions problems are solved. Problem number one what is solved? The problem number one is very simple that is I have a co environment.

So, whatever hardware software version I have I will download and I will I mean I shall I should they give the simulate and see whether there are anywhere bugs or not. So, bugs will be developed very early. I can also find out that at a every stage the design is correct or not because I have a entire working version in software.

So, I can see whether the design intent is made; bugs are caught earlier; is the design tradeoff that I find this one is good in software, I can put it in a hardware and vice versa and also it is said that I can delay the partitioning. Then I can have a initial big model and then, I can go and go quick iteration, still I when I have a more or less a complete system idea in my mind; then, I can find out this I put it on hardware this I can put in software.

So, this will actually give you a better debugging capability, the better test capability, better solution efficiency and so forth, but only issue is that making such a environment.

(Refer Slide Time: 28:57)

### **Current Abstraction Mechanisms for Software Systems**

#### Virtual machine

A software layer very close to the hardware that hides the hardware's details and provides an abstract and portable view to the application programmer

#### Attributes

- Developer can treat it as the real machine
- A convenient set of instructions can be used by developer to model system
- Certain design decisions are hidden from the programmer
- Operating systems are often viewed as virtual machines

So, that is actually people call it is called a Virtual machine. So, basically what we can have is a Virtual machine is the some kind of a machine which will emulate or simulate your hardware. This is a software you will run on a hardware. So, basically the software layer is very close to the hardware that hides the hardware details and provide provides an abstract view of the to the x programmer , but when I am writing a C code I need not think about the hardware.

But in case of a embedded system, the software person has to think of the hardware that is what is the problem. So, if I can have a virtual machine or if I can somehow emulate the hardware on which the software will run our job is solved. So, basically all the co simulation environments work on that. The hardware is written by a hardware person in verilog, but the co simulation environment it is basically abstract out the hardware part from the software part.

So, the software developer will find that the hardware is already there in place which can be a which can execute in software. Of course, the hardware will also be developed in phases initially there are block diagrams so that I mean you can do just functional simulation; maybe after that it will be and they will implemented register transfer level. Then you can also put delays and you can find more detailed analysis of your hardware. Finally, they will simulate the gate level design. So, in that case you can have exact delays, you can have a exact latch problems, you can find out how many power what power it is taking and so forth.

So, it is a hierarchical solution first we have that is the functional implementation, then you can have delay implementation, then you can have power implementation and all these parameters will slowly come into picture. But the idea is that at every point of time, you have a entire simulation of your entire system that is the key code. In traditional design, you have a hardware version hardware partition, software partition you cannot simulate the entire solution only when the hardware has been fabricated and come; then, you can put it in the software in that and you can get the working version of your entire system.

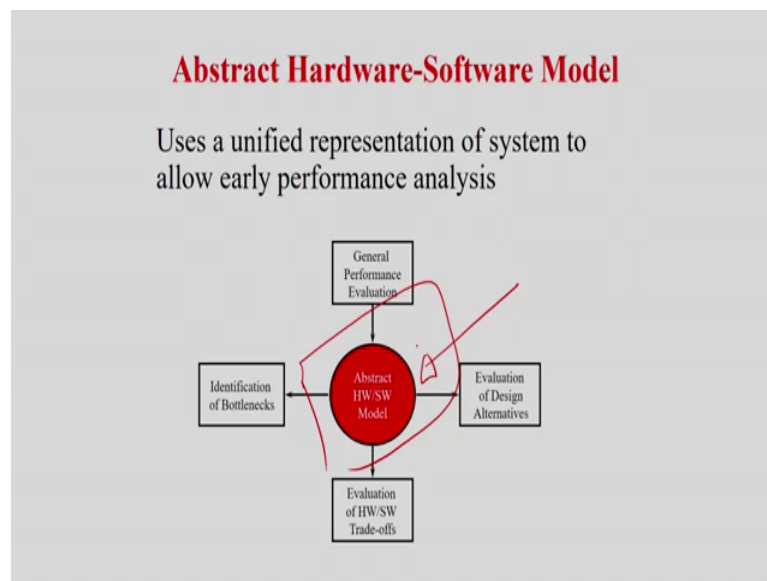
But here it is the other way around. From the starting only we will having a simulation environment of your design, but maybe I will have set level in the beginning, then after sometime you will put more details on to the design. So, you can get more features out of



it or more test characteristics or more what do I say more inside you can get about the design like I can have a black bugs software and I can have a very high level architecture design at block level, at the RTL level. Then I go for gate level, then in a when I am going to gate level I can find out the delays I can model delays in a better way. I can now very well, what will be the frequency of my circuit, I can also have a power model.

So, I can get more insights on the design, but at every point of time, I have a simulation level. So, I do not need to wait till the end to find out the bugs and land up in surprises that is what it is basically the idea right. So, we should always have some kind of this abstract. So, this is the current abstraction because this is for software system. Basically this is what is currently being done. So, they have a hardware emulation or a hardware software hard software virtual machine for running the hardware right.

(Refer Slide Time: 31:37)

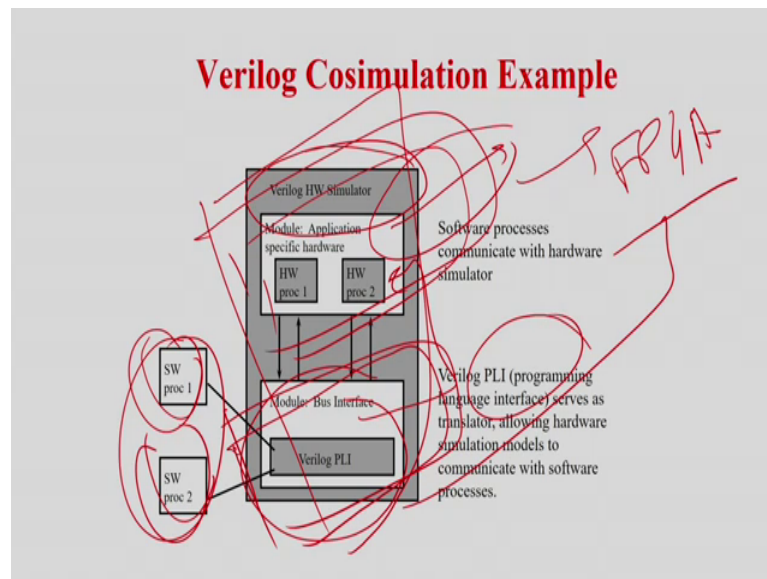


This is what basically is the Abstract Hardware-Software Model.

So, you have a Abstract Hardware-Software Model. So, there is general purpose evolution, it can support so many things, but this is what is basically your challenge that how can you have a similar table or angular table abstract hardware model; that means, what it is basically this challenge. So, you can do a lot of general purpose system evolution you can do, evolution of design alternatives you can do, evolution of software hardware tradeoffs identification of bottlenecks or bugs everything is possible.

But I require a simulation version of this which is the main challenge.

(Refer Slide Time: 32:09)



So, how people have tried to do it? They have something called a verilog hardware simulator. In this case you can download your hardware. First may be an RTL level, then you can go basically at the gate level and maybe at the very detail level. So, you can do a phase wise. So, this is your hardware; this is the hardware where which simulates the verilog and you have some software procedures, but as I told you verilog can only simulate your hardware. It cannot some of the software on top of it.

But people sometimes also run put some kind of a interfaces. They have written some kind of a interfaces are that in the software environment, a poor development environment we will have lot of interfaces in the verilog itself. So, in that case what happens? It will take the software. It will come convert this into some kind of a hardware representation so that they can be integrated with this environment.

Again, making the story from the beginning, verilog can simulate hardware which is written in software in languages like verilog or there are a lot of HDL simulators. So, HDL simulators will take the soft version of your hardware in it can execute, but it cannot take how a software will run on it that they cannot do for that people have written some kind of what is called a bus into what they call it Wrappers that is a verilog PLI programming language interface as they call it PLIs. In that case is the hardware which is written in verilog you write some kind of wrappers.

. So, in this what the wrapper will do the wrapper will take the software which is written in a C plus or whatever language you want your software to be. It will translate those softwares into a verilog representation. So, that they can integrate with the HDL representation of the hardware; maybe this is some kind of a maybe simple like can be a simple adder which is written in basically your verilog.

Now, this is a software which are using this adder. So, what this PLI will do? This is basically will translate this code of the C language which uses the adder so that it can fit into this into the an hardware environment simulator for this verilog code that is what is basically the idea. So, it is an integrated environment. So, then such an environment can be developed and there are lot of environments there, but the main the problem is this p. So, it is actually you can think this it is a very it has become a very complex. Hardware is generally some maybe a 100 lines of verilog code makes a hardware.

Now, if you try to integrate with the C. So, C language will be taken it will be translated into verilog or mapped into a corresponding verilog language so that it can interface with the hardware which is written by the verilog that is the pure hardware. So, this software cannot the c language cannot be run on that in simulation. So, they are converting into a hardware language which is compatible.

So, that code actually becomes very very high because C codes or software programs are generally very long in length compared to the hardware counterpart. If the hardware represents a hardware block or hardware definition language represents the hardware in software language. So, those codes are generally not very long because it just be optimized because that finally, it is to be fabricated. So, generally there are lot of optimization techniques that goes in.

But software codes are generally implement the general purpose functionality. So, if you are representing general purpose functionally using a code generally it becomes very very long. So, if I try to represent such C codes using its verilog representation, the code becomes very very large and complicated and the simulation takes lots lots of time. So, what is the solution? People try to use this and dump it into FPGA. So, it becomes a hardware representation that is a real hardware, I do not required to and simulate anything this is called emulation.

Then, now you can directly download your software and you can test it. So, nowadays people are actually taking this environment. So, that is why FPGAs have become a very very field role player in an integrated environment business; instead of simulating the hardware, they write the hardware, develop the hardware fully, but I cannot fabricate it because it takes long time. You download it, download into this the FPGA and basically you download your software onto the FPGA which is called an emulation not a simulation and then, basically you can see whether everything works.

Still it's good, but not as better as best as the co-simulation environment. Co-simulation environment is very flexible; you do not require any hardware. Everything in a software you just write a code and see whether everything is proper or not, but as I told you it which will be require very high speed computers to make a time optimized simulators for such a co-simulation environment. So, many people are trying to go in the other direction in which case programmable devices are available, you download your hardware there, it becomes a prototype device and you soft test your software and that.

So, with this we come basically come to the end of this course as well as this lecture. So, these are the very open problem. So, verification test. So, main problem comes because we do not have the full design at because when I talk about the embedded systems you partition it, something into hardware and something into software and you test it separately. It is very challenging to develop an integrated environment in to solve the problem and to solve the problem. So, that is one of the key challenges where all embedded system people are working for.

So, it is an open problem like. So, you have to look at machine critical systems, you have to see look at for handheld devices, you have to see for personal devices and I require a common modeling environment, common verification environment and finally, common hardware development and test environment. Can I do it in a single window? That is what is the key role which is the open problem.

So, with this we come to the end of the course. I think you have enjoyed you have enjoyed the course throughout the last 3 months basically and in fact, what you should try to emphasize that this course as was a part which included 3 different aspects like design verification and test. We have tried to give you the some basic broad lines of all the different paradigms. So, this becomes a and course in an entirety; otherwise means we

have not gone into too much details of each of these steps because they would (Refer Time: 37:34) a course in itself.

But we thought that basically this course should be a first course towards the embedded system in a entirety because when I am putting this 3 phases together because no embedded system can be complete with the design phase alone or test phase alone or verification phase alone. Entire embedded system design flow comprises of the design modeling verification and test. So, so that was our emphasis. So, with this, we would like to all thank you all for your patience listening over the 3-4 months and all the best for your exams.

Thank you.