Digital VLSI System Design

Dr. S. Ramachandran

Department of Electrical Engineering

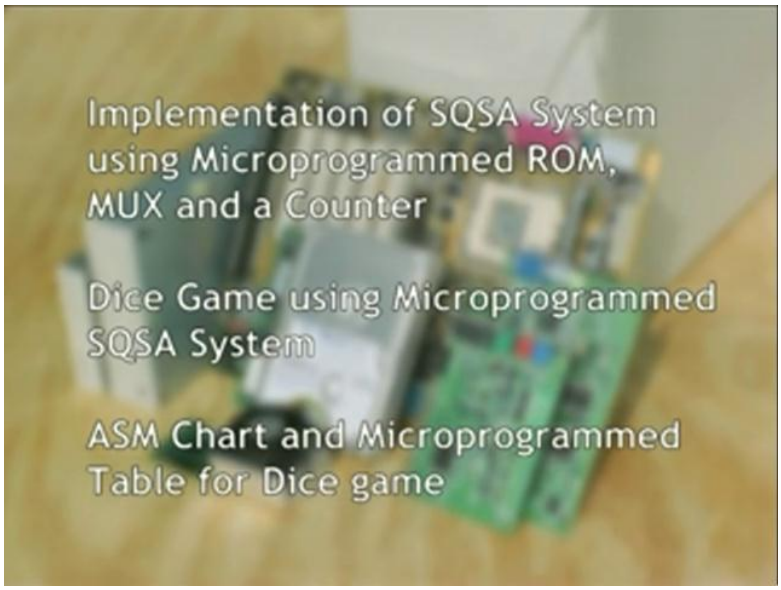
Indian Institute of Technology, Madras

Lecture - 23

Design Flow of VLSI Circuits

Slides - Contents of previous lecture - Microprogrammed Design (Continued)

(Refer Slide Time: 01:02 min)

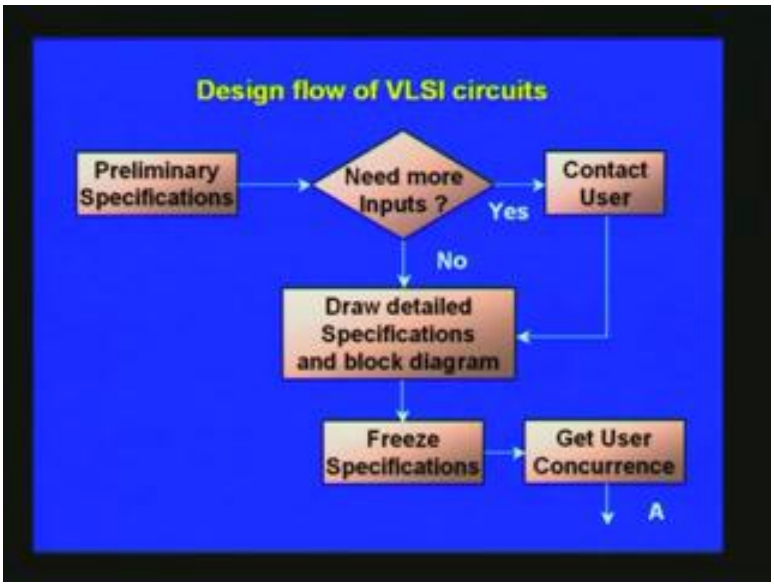


Slide - Contents of this lecture - Design Flow of VLSI Circuits

(Refer Slide Time: 01:59 min)

Earlier, we had seen how to write a test bench. Now, prior to starting simulation, we will just see how the design flows for VLSI circuits in general.

(Refer Slide Time: 2:29)



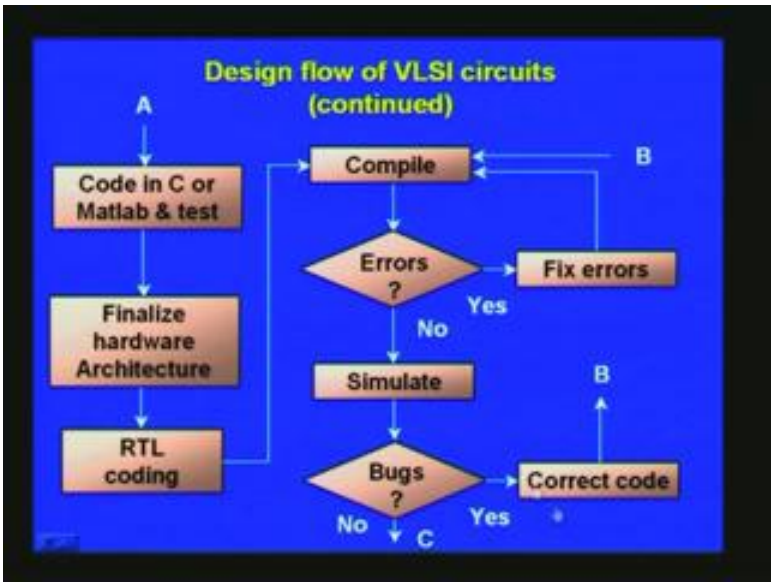
To start with, you need what is called preliminary specifications. Prior to this, you would have done market survey or what you call market research. Then talk to various customers or users and then identify a product. Having identified a product, you are now

in a position to formulate very preliminary specifications. Once you have done this, you may require more inputs as you progress along this detail specification - working out of that. During this course, you need to interact with the user and ascertain whether you are proceeding in the right direction or not and accordingly amend your specifications. Once you are through with this iteration, you come either through this path or through this path implying that you have finished with collection of all inputs.

Once you are through with the inputs, it is now time for drawing out very detailed specifications. You should also put a block diagram of the whole scheme both - including hardware and software.

Once you have done this exercise, this may also be an iterative process with the user and ultimately a time will come for you to freeze the specifications. This is exceedingly important because from prior experience I have seen that people keep changing the specs even towards the end and you are in for trouble then. Therefore, it is very important that you freeze the specs right at the incipient stage and only then go along with further steps; because, you may also have to do the hardware. So it may be easy for you to amend your verilog code etc. All software aspects can be done quite easily and rapidly. But if you once start on hardware, then half way through making a PCB etc., it may be very difficult for you to re-track and hence the emphasis on freezing the specs. Then you get the user concurrence and then go on to next step.

(Refer Slide Time: 4:57)



So this is the entrant for this particular slide and once you are through with detailed specification for both hardware as well as software, you may have to prove your concept. You might have developed a new algorithm or a new architecture you are attempting, but it is still unproven. In order to prove that, you may need to go for a higher-level language such as C or Matlab, and then code it in one of this or both, if you wish.

Finally, test it with all possible combinations, the I/O combinations that you may encounter or encounter later on in the actual design when you implement the hardware. So, once this is through and you are sure that your concept is working, you can even get user feedback even at this stage, although it is not shown. Then it is the right time for you to start the hardware design.

You start on this architecture here, while you are designing this architecture, you should keep in mind the hardware aspects as to how you are going to implement: are you going to put a register or a pipeline register? All those considerations you have to do here. And mind you, as I have said earlier that you tend to design this hardware architecture also in the same way that you have coded in C or Matlab. So this approach is entirely different from this. That you should bear in mind very clearly, you should not start coding in the way that you code in C. This is what we have already seen such as RTL coding is what is

implied here and that is naturally the next step here. Once the architecture is through, you start identifying different blocks and start coding. It should confirm to RTL techniques.

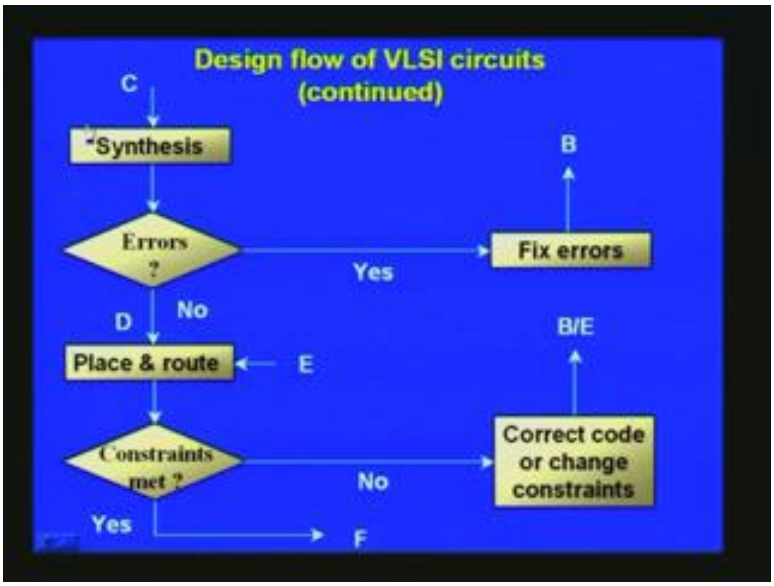
The next step is to use the simulator, let us say, or for that matter, even synthesis tool, you have a compiler and so also in simulation tool. So, this step is: you compile your verilog code - both the test bench as well as the design. Once there are no compiling errors, you can go on to next stage which is simulation and if you encounter errors, naturally you have to fix those and go back to compile. This is an iterative process and beginners especially will have frustrating experience in this to start with as they may encounter hundreds of compiler errors. You have to wade through these errors and once you are through with this, then you will be at home, later on. We will take some exercises practically to see what type of errors can come and try to resolve that as far as possible. This is the only bottleneck in the entire design, I should say, especially for a beginner.

Once you have set on that, this will be very rare, may be one or two, which you will be able to fix with very little effort and time. Once you are free of errors, it is time for you to go to simulation. Simulation, as we have already mentioned before, it is nothing but a timing diagram that you get here. Any signal will be displayed in terms of different times. You can choose microsecond or nanosecond as the case may be.

Today, we are going to see the simulation aspect, after we are through with this design flow. Here also you will again be encountering lots of bugs in your program, in the sense that it does not meet the functionality that you had really designed it for. If there are such bugs, once again you have to identify which part of the code is erroneous and then correct the same and then go back to square one again. We will keep repeating, coming back to this compiling, time and again. This whole design process is an iterative process. You have to do it a number of times till you get a totally error free or bug free code that will work on your IC finally. So if there are no bugs, naturally you go to next step C.

I forgot to tell one more thing. There are so many simulators available in the market. We will be using the model [ ] 5.5D and a version I do not remember. We can look into that.

(Refer Slide Time: 9:23)



So the next step is synthesis. Regarding the simulation, there are other simulator tools also. See, for example, if you are designing an ASIC, the simulation tool will be a DC compiler of synopsis, if it is an ASIC design. Then there are many other simulator tools also.

The next step is a synthesis. In the synthesis step, what you do is you map on to a particular target device. For example, if your end target is an FPGA - field programmable get arrays - then you need a tool. The tool that we have here, we will be seeing and using will be a simplifier. It is from Simplicity Corporation whereas the simulator tool is from Mentor Graphics. So, there is a Leonardo Spectrum also available, another synthesis tool. In an ASIC platform, it will be once again the DC compiler which we have already seen in Synopsis.

Once you fix, if you encounter errors here also, here the type of errors you may get: you might have given frequency of operation to be a very high value. What could be achieved on a particular target FPGA is not able to match that speed that you have specified. It may also give compiler error and set up hold time violation in a way it will give. Finally, it will do the optimization of your logic, which is what the tool is. In a sense, it is an optimization of logic as well as to provide you an .edf file, what is called as electronic

data information format. It finally gives edf with an extension of .edf file. So, that will be the file that it will create finally and you can take into the next stage. This is called place and route and here once again if there are errors, you have to again go back to compiling and do the whole iteration once again till you are totally free and it has allowed you to go to the next stage.

In this step, what you do is here it has created edf file and you will invoke what is called, [vendors] specific platform for place and route. What is place and route? Here in synthesis you will get gate level net list. For example, FPGA will be using what is called lookup tables, a [MUX]. Like all basic primitives pertaining to a particular FPGA or a group of FPGAs will have similar such primitive gates or MUX.

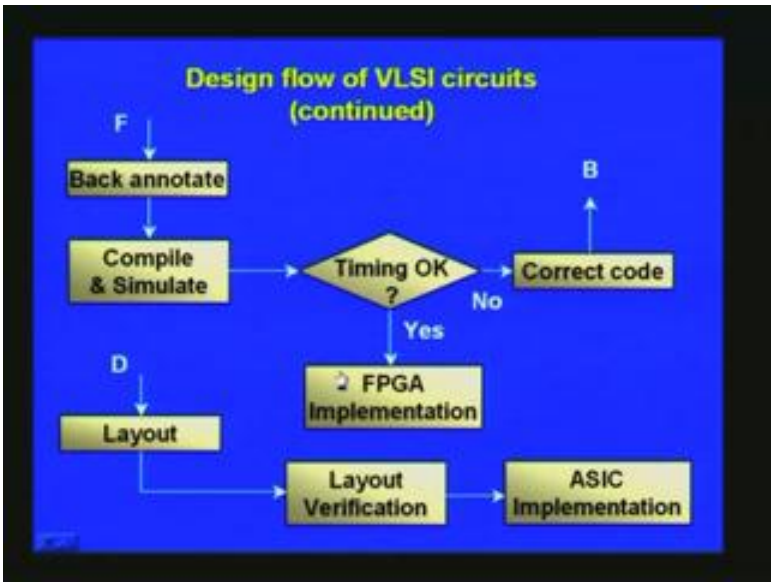
So in terms of that, the synthesis tool will give in the edf file a net list, and that will be the input for place and route on a particular platform. For example, if you go for Xilinx FPGA, you need to have place and route of Xilinx, and also for say Ultra, another alternate FPGA vendor and so on. There are so many vendors: Actel and Lucent and so on. Once you are here, what it does is it places different primitive gates or MUXes etc., what we have already seen. At strategic points, it locates the optimum possible point. You can also specify your I/O pins, etc. Finally, it starts routing it. Routing is just having placed different components inside, it starts interconnection. This is a time consuming process; it may take an hour or even more. I have seen even 72 hour running on place and route, and so also the synthesis tool.

Large files will invariably take very long time, but for the small files, which we have already dealt with, it is a question of 5 minutes or 10 minutes, not more. Here, as I mentioned before, I/O pins can also be assigned by the user, that is you as a designer. These are all what you call constraints.

You need a speed of operation and that also is specified, not only in synthesis, and that information is also conveyed into this. It will take that as a constraint. Or you can even overwrite that constraint and you can run it with the constraints. In FPGA, this constraint will be basically speed of operation and along with the pins that you want to configure. In ASIC, here the constraints will be in addition to this, you will have a power as well as the

chip area, which also will be available as a constraint that you can programme. Once again, you will encounter different errors or limitations that will be reported if the constraints are not met such as, let us say, a frequency of operation we have specified for a particular FPGA as 100 megahertz, but it has been reported that it can give only 60 megahertz. If you take 60 megahertz for your design, that is no problem. Even then, I think you will have to get rid of warnings etc., when you can go to the next step. Otherwise, if you are particular about 100 megahertz operation, then you will have to identify where you can introduce additional pipelines etc., so as to improve the speed of operation; for all that you need to correct your code. So, once again, you have to go all the way right up to that B compiling. If it is merely constraints that is of the speed, you do not have to go there and you can straight away tone it down to lower frequency that you want and then proceed here. The next step is… here in the previous thing, this is the route adapted for FPGA design. If it is ASIC design, you by pass this route and take from this straight away to the next.

(Refer Slide Time: 17:00)



The previous FPGA final stage was F; whereas, if it is ASIC you go to this. So, otherwise there are so many common factors, approaches and blocks. You have to go through exactly the same thing except for this small deviation.

In FPGA, this is the entry point after the place and route. After the place and route, it will give you what is called bit stream that is .bit extension of your design file. That is the one that will be selling, if you are the IP core designer, basically; you do not have to do anything other than just deliver the bit file and most probably, you do not deliver the source files. It all depends upon the contract that you have with your customer or user.

So far, we have concerned ourselves with only the simulation and mapping it on to the device on FPGA. That is not adequate, because what is the guarantee after doing all this [exercise] that it will work? So you have to do what is called back annotation. This will update all the gate level delays etc., incorporate all that. There are a set of commands for this, running which you get a back annotated file. This will be again a .b file. You started with a .b file and the outcome is a .b file, but the file size will be enormous.

If you go through the .b files, it may be very difficult for you to recognize your own design. The reason being that this reflects the gate primitives that we have already seen before, if such as MUX, etc. LUTs. All those will be appearing in this .b extension file and this will have timing also.

There is another file called sdf that is the standard delay format. Another group of files are required, which we will see when we go into the details of that. So only then, it can do the back annotation. Back annotation is the one that reflects the actual gate delays. Having got this, once again, you have to go to step B, and then do the compilation and simulation. So here, there will be most probably no compiling error, but unless you compile you cannot simulate. You have to load etc. So you enter the model sim for simulation once again; so you can straight away go to model sim and do the compilation right there and simulate here. So here, will there be any problem associated? Most probably, functionality is okay. But you do not know whether timing has been really met. Let us say, you are simulating still at 100 megahertz, whereas the previous tools like Xilinx also gives a revised estimate of the frequency of operation. That is the final. Synthesis tool also gives this one, but that is rather a rough estimate only. It does not actually reflect the true thing. Only the vendor specific software will reflect and that is place and route of Xilinx, which we have seen earlier.

If this timing is too close and if it is slow, then you will have timing problems. If that is okay, then what you do is you go on to the FPGA implementation. Otherwise, you have to go back to compiling once again and wade through the entire design cycle that you have done earlier, till you are totally free of all timing problems. As I mentioned before, the whole thing is just iteration till you settle down on to the final target.

When all the errors are fixed [in toto], then what you can do is go on to the hardware implementation of the FPGA. The development tool that we have can program let us say E square PROM or EPROM in which your bit stream will reside. Finally, you will have to make a hardware, a PCB, and then with this particular target FPGA that you have already designed using the whole flow design.

Then you have to evolve a strategy for testing the PCB along with this. After that you can integrate with this development system that we have - say that of Xilinx - and you can download your bit stream right on to the FPGA, that is housed on the PCB that you have made. So you have this software path, we can say and parallel to this, right from the instruction when you froze the specifications, you should also start the hardware work. For example, you should know FPGA. You can, with some few exceptions, change the I/O pins as we have seen before because, user constraints you can specify the I/O pins.

In the beginning stages, you can go about freezing even the hardware saying that FPGA, these are all the I/O pins, these are all the clock signals. Based on that information, PCB work could have started from day one, when you started the verilog coding. That means that hardware and software design go hand in glove and we have to interact with each other. Normally, a team of engineers will be allocated for the hardware design,; another team for software coding, say for this verilog coding; you have to interact with each other and get the final product, all that is implied here.

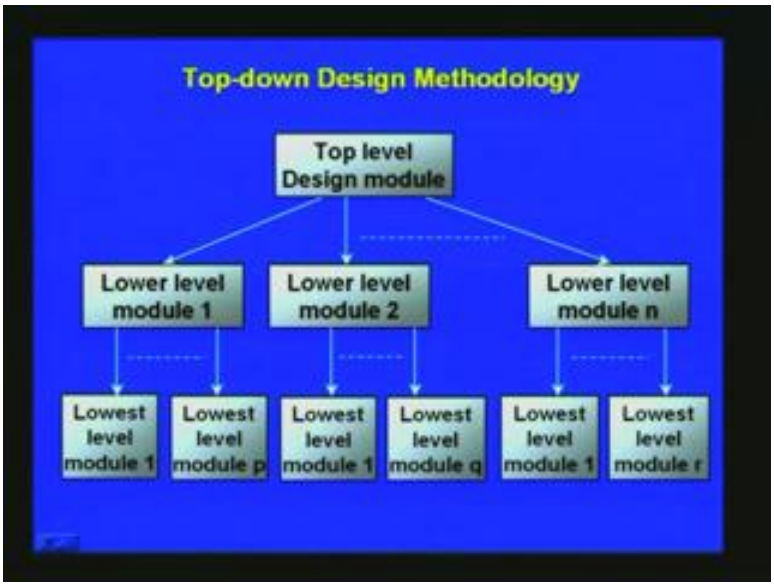
In addition to that, you have even mechanical design. Another group will be evidentially doing it. That is the final get-up; that is what you see apparently. You have to take the ergonomics also into account, while making a product and all that are implied in this. So is the case with ASIC implementation; whether, you have your code residing ultimately on an FPGA or an ASIC, what we had just now seen regarding the hardware is the same

fundamentally. In this case, we saw that (Refer Slide Time: 9:23) after a synthesis tool you may be using Synopsis DC compiler for synthesis tool. Once you have these iterations are all primarily the same. Once we have fixed all errors, it will come through this path and into this.

The next thing is the actual chip layout. A chip is a very complicated affair and you have to have a look at the layout. For this, we have a tool called Cadence in the ASIC domain. Another alternative is the Synopsis DC compiler itself, you can have your own constraints, some TCL files etc., and it will give database file. You can feed that database file straight away to the ASIC vendor, and then he will take care of the layout as well as layout verification. He may finally, give you back annotated thing because, we do not have that here and many people may not have this.

So, there is no problem because the ASIC vendor himself will support this. So no matter whether you do it using your tools such as Cadence here and finally do layout verification, finally you land up making the ASIC and you have to hook up the hardware as we had seen before, in this case. This explains the design flow for both FPGA as well as ASIC. Before you start on a particular design, you have to have a strategy for or a methodology for designing.

(Refer Slide Time: 25:29)



For example, you may be doing a very complex design. If you take FPGA platform, there may be some…. If it is Ultra, I think they have some LPM modules. If you take Xilinx, they have Coregent. They are all vendor specific (Refer Slide Time: 25:48 min). If you had coded on vendor specific tools, then you cannot naturally use it in ASIC flow. My approach that you have already seen here it is friendly for FPGA platform as well as ASIC platform, so long as you stick on to that style of coding. It is so to say a universal way. We will try to stick on to that particular thing. So as far as possible and avoid vendor specific standard modules - I mean their own modules.

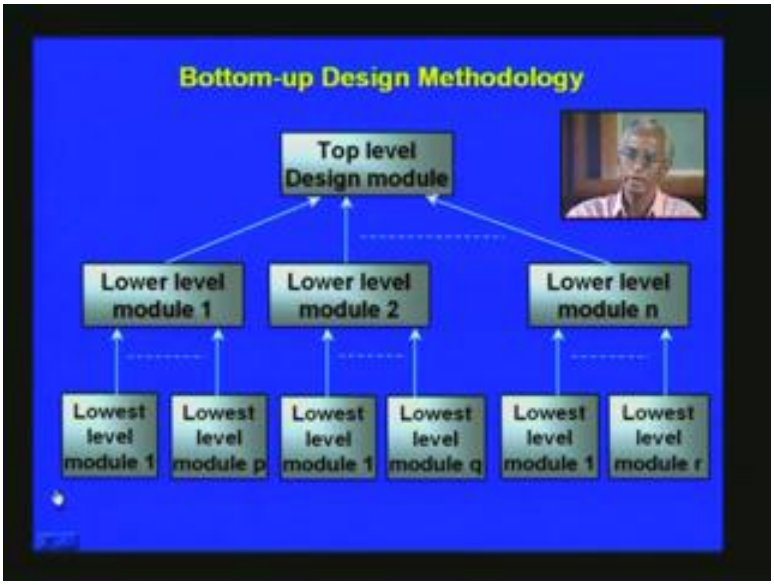
Coming back to this design methodology, you have what is called top-level design module. In the example that we have already seen, say combinational sequential circuit design or even in a small design that we have considered, we had a design file. That design file is the only file; there are no down the line sub modules; so that you can identify that module as the top-level module. If you have a bigger design, it is a good practice to downsize it to smaller sub modules and further sub modules in this fashion, and it can go any number down below.

The idea is that we manage only small things at a time. So, once you have tested small modules, then you can put them together, and then test the higher module. Another method is this top-level top-down design methodology; that is, you start with the top-level module and here it will have something like an ordinary IC with a different I/O specified. Take that one and merely call whatever sub module you want. You merely make a call of that; assuming that those sub modules are already available to you. You can, in that case, start from the top and go down. Where you have not done, you do not have a module there, a missing one or only partially available which you had to use for some other project, you can take that, make an amendment or write it afresh and then go down here.

Like this, you can go step by step into different modules and keep on developing those modules. At every stage naturally, you need to have a good practice. A good practice will be to have a test bench for every module, whether it is this module or any other module or even an intermediate one. If you have a test bench, you can rest assured that it will

work right on the top when you integrate all the modules. This methodology, wherein you start from the top and then go downstream is called the top-down design methodology.

(Refer Slide Time: 28:44)

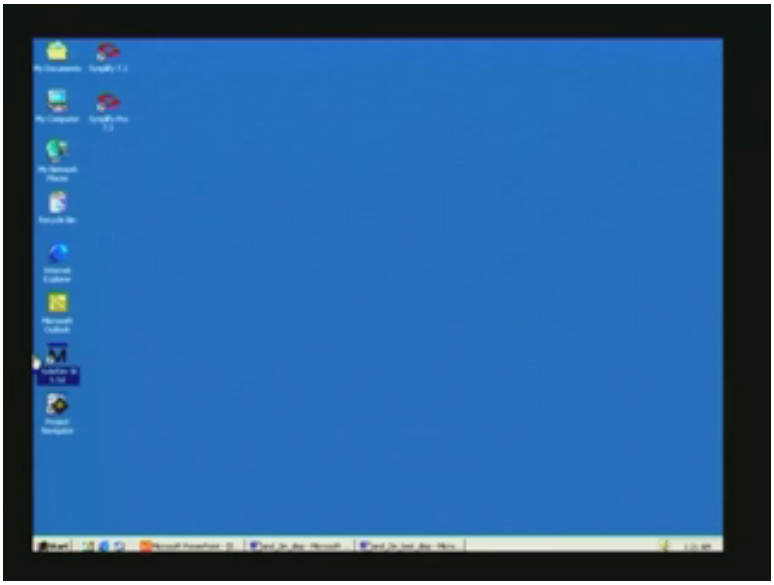


In contrast to this, you can even make an approach in the reverse manner. For example, you have already done a couple of projects, let us say, and you have used different lower modules such as adder or multiplier or subtractor or floating point and so on. They may be readily usable, so they may be residing at this level and you can start going in the reverse direction and putting these modules together, you can make a little more higher level thing, which will give a particular functionality for an intermediate application as such.

Like this, you can create all this lower modules and move up in this case and then finally do the top-level thing. We cannot say which is the best approach and many people prefer this approach. What is actually popular and what I have been following is a mix of both these methodologies. Sometimes, I may have to start here and also start from top to bottom so there may be a meeting point in between. We cannot just pin it down that this is superior to the other.

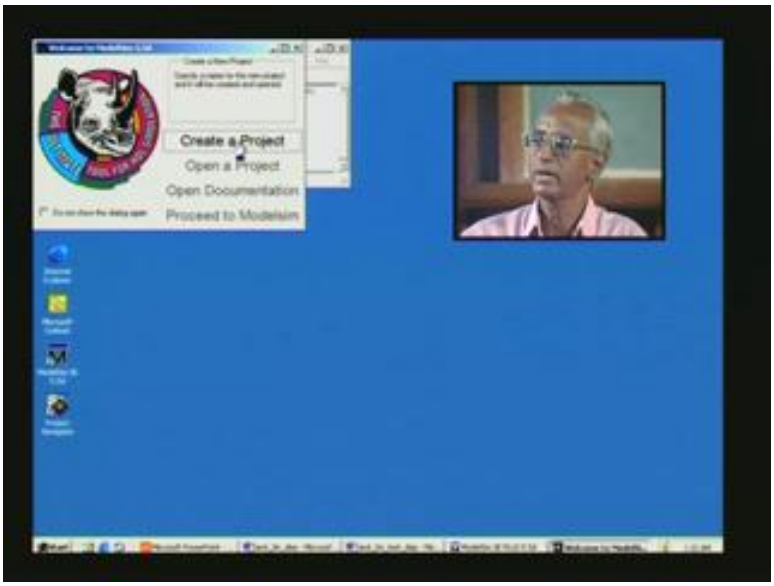
Let us now go to simulation. We will use model sim. This version is SE5.5d. I wonder whether you can read on the TV screen; anyway, that is what is here. I hope you can see this M.

(Refer Slide Time: 30:09)



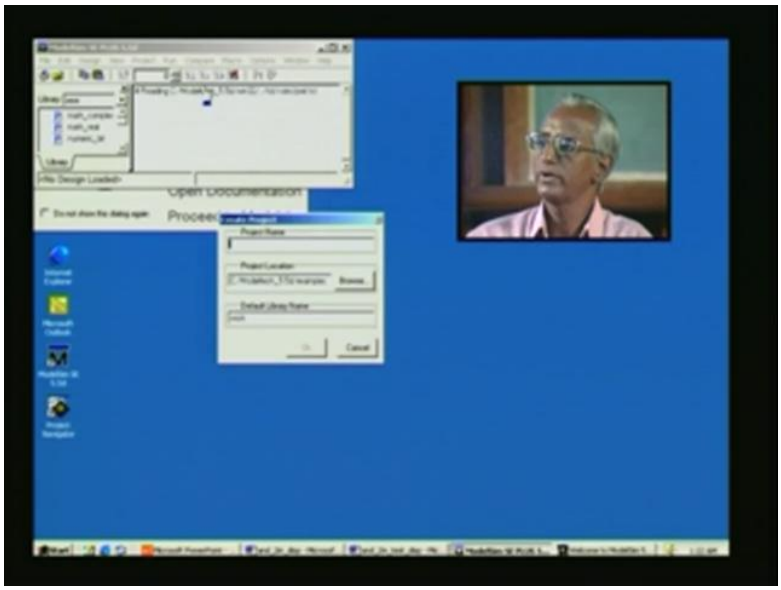
We will just double-click this, in order to open the model sim.

(Refer Slide Time: 31:08)



You see two windows opened. Now, the first thing we have to do is create a project and then open a project. If you want to create a new project, you use this one. That is what we are going to do now. If you enter the same project afresh another time, you will use open project and enter. If you want to see online documentation, you can use this here. I think you need internet connection for this. The next time you want to go straight away to proceed to model sim, you can use proceed to model sim. You can have either that way or this way also. If it is going to be the very same project that you have done before in the last session, you can straight away proceed to model sim.

(Refer Slide Time: 31:13 min)

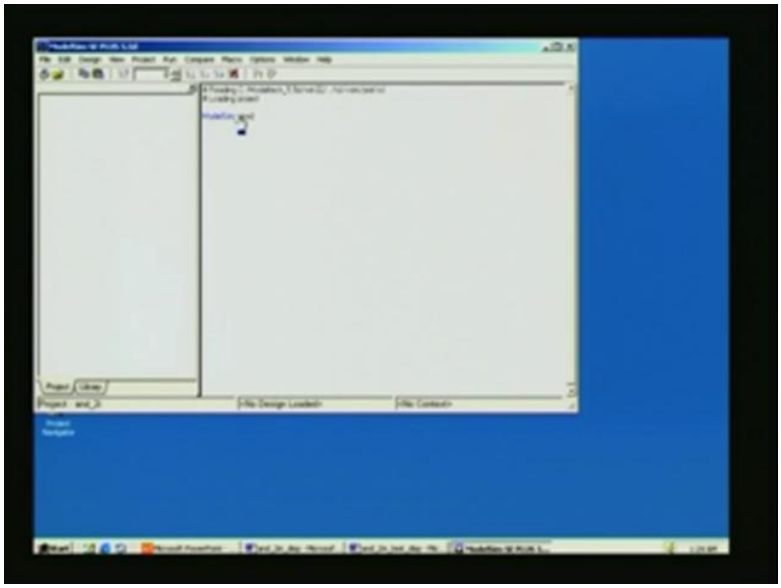


We will now just start with a creation of a new project. You just click and you see here there are two windows. What you have here is a project name you have to give here. This is corresponding to the create project that we saw in this menu, and you see here a project location. Where your design files are residing, it is a good practice to go there. Then, in order to do that, you have a browse. Another window pops up. Here it is there in this directory. So, User to ROM to DVLSi designs, then ETC. That is it. There is nothing more there. So we will just go here. We will say open and the whole thing has come over here.

Now you have to give a project name, a meaningful name. What we will do is take that simple design that we had seen earlier, that is a two input AND gate design. That name is AND underscored 2i. So we will give that as project name.

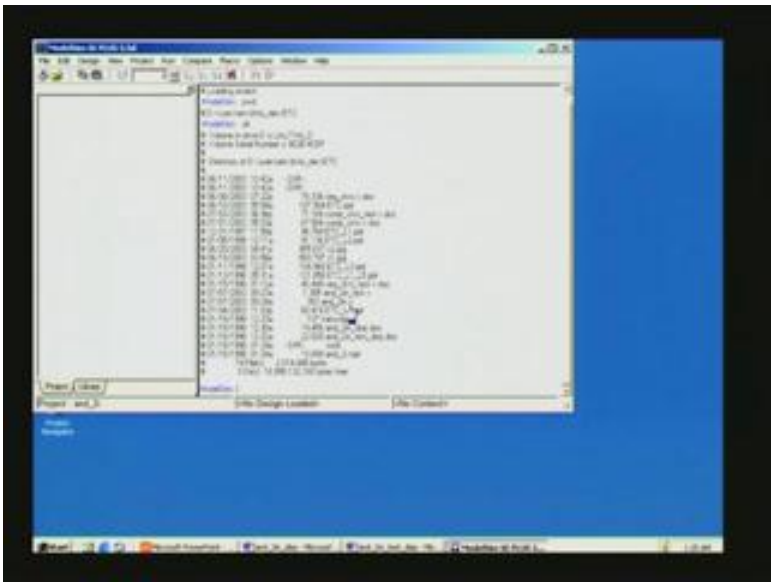
There is one more thing called… we have a library here. You have to create a work directory and you have to work in that one, not in IEEE; otherwise, some platform dependent files will be overwritten. That is not a desirable thing. So you use always a work directory. Just click OK. Let us see what is happening? That window is closed. Now, this may be difficult for you to read; I will enlarge this; even then, it may be difficult for you to read. If you want to know what directory you are in, you can just say PWD in this command mode.

(Refer Slide Time: 33:11 min)



This is a model sim with greater than symbol; this is just a prompt for the command. If you say, that one, note that we changed the directory up to this; up to ETC. That is what is in now. If you want to see what files are there in this directory, you can say just ‘dir’. It will list all the files.

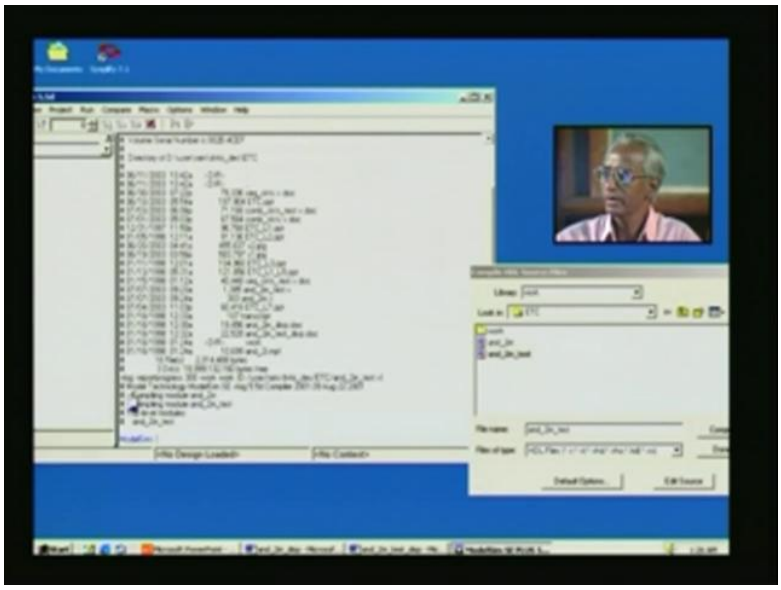
(Refer Slide Time: 33:39)



Now let us see whether we have that and to in dot v and its corresponding test. This is a test bench for that. So, we have this here. Our next step is, what we have to do is, we have what is called project and library. You click on the library so that work is used here and then do the compilation. So design and compile we had given; then a menu popped up listing .b files. Here you can just see here all b files are used. Now what we want is… we had a test bench as well as a design, and if … remember we have already included a design file in the test bench.

It is enough if you just compile this test. It will automatically go to this design and do the compilation. You do not have to do it afresh as such. If you just double-click this here or single-click, it will appear here, and then use compile. Had you double-clicked, it would have started on the compiler straight away. That will be a short cut for you that you can adopt later on. The compiler has and the result appears on this window. So, you see that there are no errors.

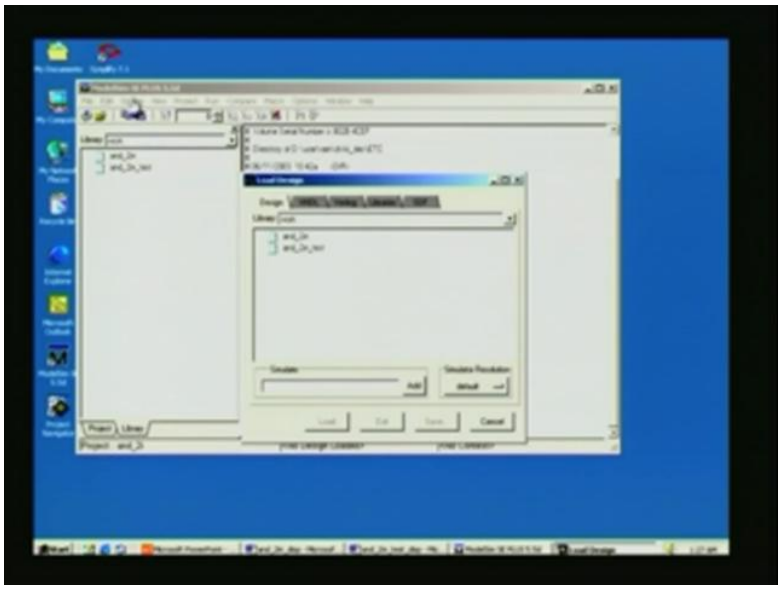
(Refer Slide Time: 35:03 min)



I will read it out for you. It says compiled module and underscore 2n. This is the design file and then same file with underscore test, extension for the compilation of the test module. A test bench is also compiled here. It always starts with a design, lower module and then moves higher up here and the top-level module also just shows it; that is all. Once you have done the compilation you are done with it. So just, say it is done.

The next step is you have now compiled your test bench as well as the design. Now it is high time that you loaded it into the simulator. Our aim is to see the timing diagram, which we will do shortly. Before that, what we have to do is load the design that you have compiled. Again use design and after compile, you see a load design. Just click on this. The whole thing is menu driven.

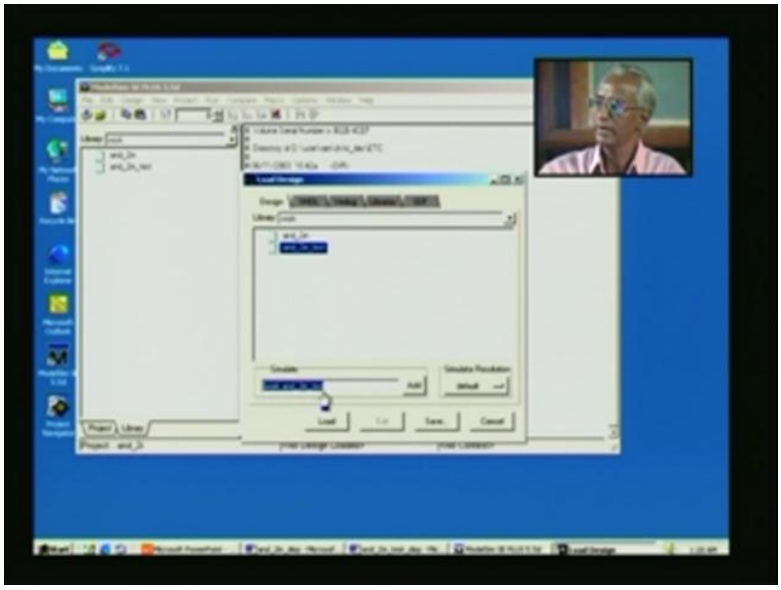
(Refer Slide Time: 36:15 min)



You have only to note what I have been saying for different things and we may be summarizing all these commands later on.

Now, what we want is the test bench here and you do not have to invoke this here because, this is the top-most module. If you double click, this will appear here and this will be energized. This is a load button here. Double-click this.

(Refer Slide Time: 36:48 min)

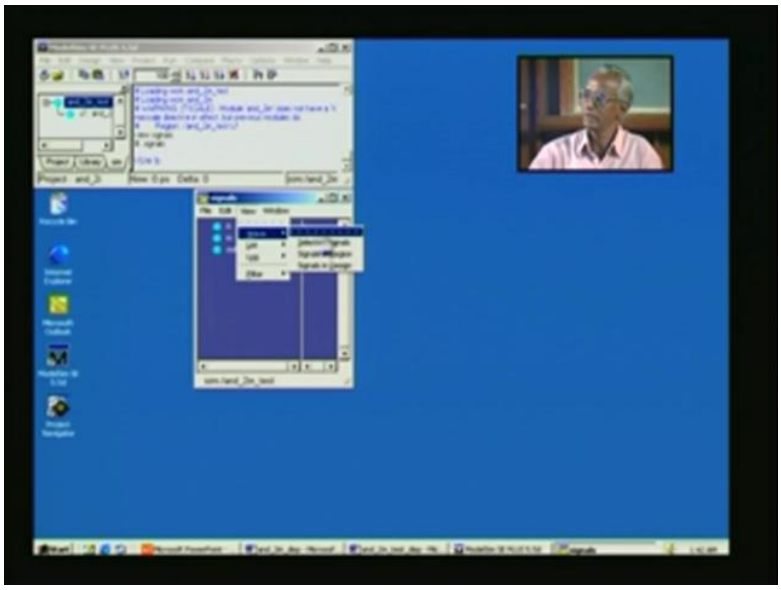


It says work done and in underscore test. So this is the top-level module that we have for the test bench and it is in that work directory.

Now we can load this file. Now let us see what happens when we load this here. So you just watch this here. While loading the design we just now encountered an error. I traced it to the system time being different. Now it has been set to correct time and now there is no error while loading the design. Anyway, I will repeat this for your sake: you design; then you will say load design; then you have this top test bench, so double-click on this and load. There is no error now, but some warning is given. This is a trivial warning you can ignore. What it says is module and to in does not have a timed skill directive in effect, but previous modules do. The previous module is the test bench that has a timed scale we have already given, but not mentioned in the design. That is what is mentioned here and notice that as mentioned before, we should be free from technology dependence. So there must be no time element in the design. That is the reason why I did not put a time-scale there, but only on test bench because, that is a vital thing for testing it on the simulator.

The next step is, you will be eagerly waiting to see the waveforms, how your design works as such. So that is what we are going to see here. After design there is one view here, just click on view.

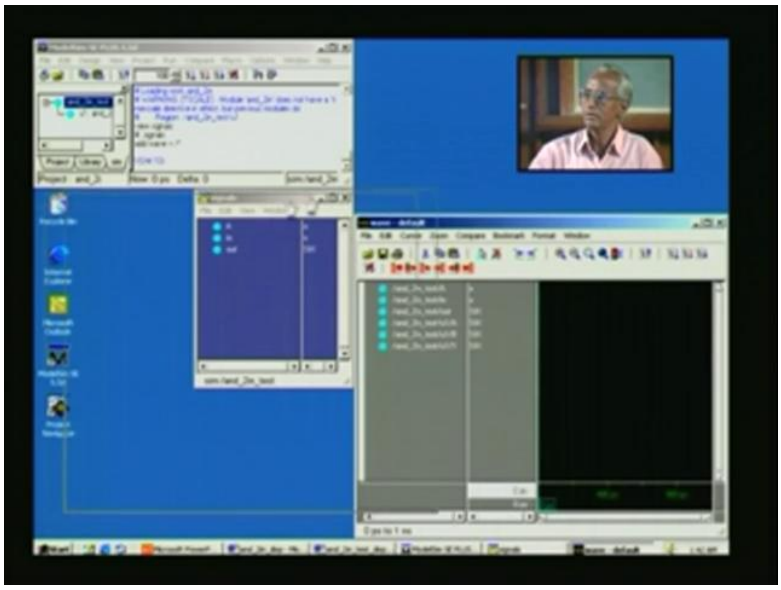
(Refer Slide Time: 38:45 min)



If you see different signals, you will have same signals by name. Just click on this. It opens a window which lists all the signals that you have in your design. A, in, out are all in the design.

In this, you have again file, edit, view and window. This may be difficult for you to read out and yet just follow my words. I am going to click on view here and what it says is again it has two to three options here – wave, list, log filter and so on. Go to wave, and then take your cursor in this fashion. Then go down. We want all the signals in the design, including the test bench. Not only your design, but also the test bench, all the signals. If you use this one option, you will get all those.

(Refer Slide Time: 39:45 min)

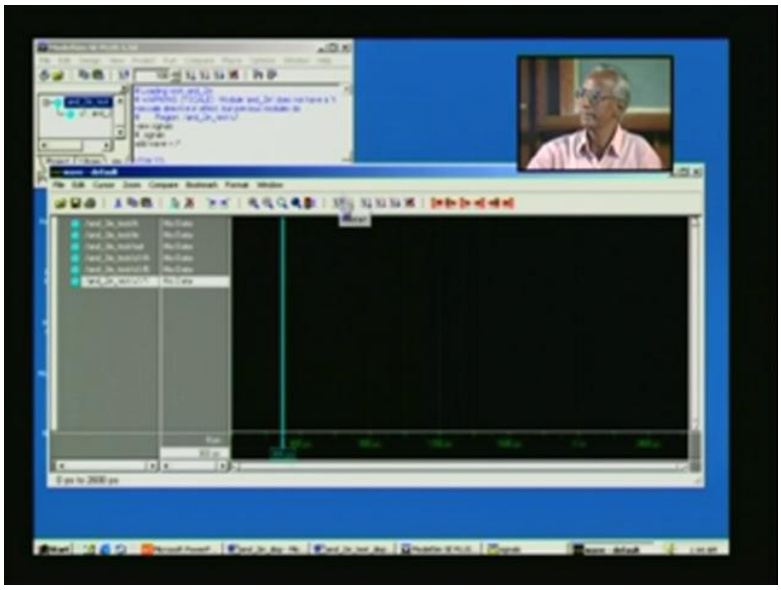


I am just clicking on signals in design. So, immediately you see a waveform window open. Just make it wide enough for looking at the waveform. Now what you have is this and underscore 2in underscore test. All of them are same except that these three signals pertain to the test bench. So, in and then out; you remember we had used in the test bench in and out instead of b and y. Those are design. U1 is instantiation we had used earlier, while instantiating in the test bench. We call the design module as U1. Now the signals here are listing for the design here, say a, b and y. This corresponds a to very same a, and after we run, we saw that a, b and y are exactly the same as a, in and out. Although this pertains to the design module and this pertains to the test bench.

Now another thing is unfortunately, on your TV monitor, you may not be in a position to read out this, that is why I am reading out every time here. If the waveforms are displayed in this fashion, it may be too crowded for you. Fortunately, you have one way out of this, and that is what is called Format. Just click on the Format. Now, nothing happens because all are de-energized.

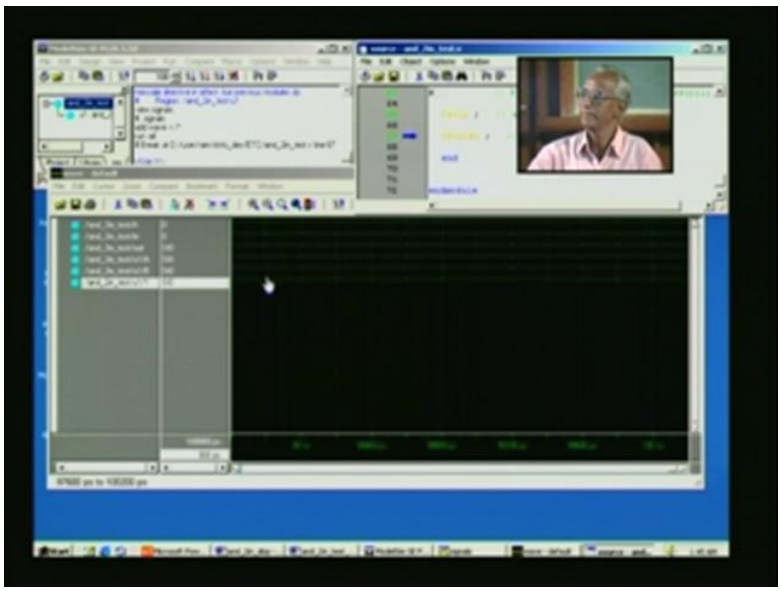
So we should first make a run then we will see Format. So what we have to do now is run the test bench.

(Refer Slide Time: 41:35 min)



You can see a different one called Restart here one button and run here. Continue to run, then run all, run minus all. If you click on this one, then it will run through your test bench for all the choice. The truth table of just two input corresponding to that four states were there of the truth table, that is what you are going to see here.

(Refer Slide Time: 42:00 min)



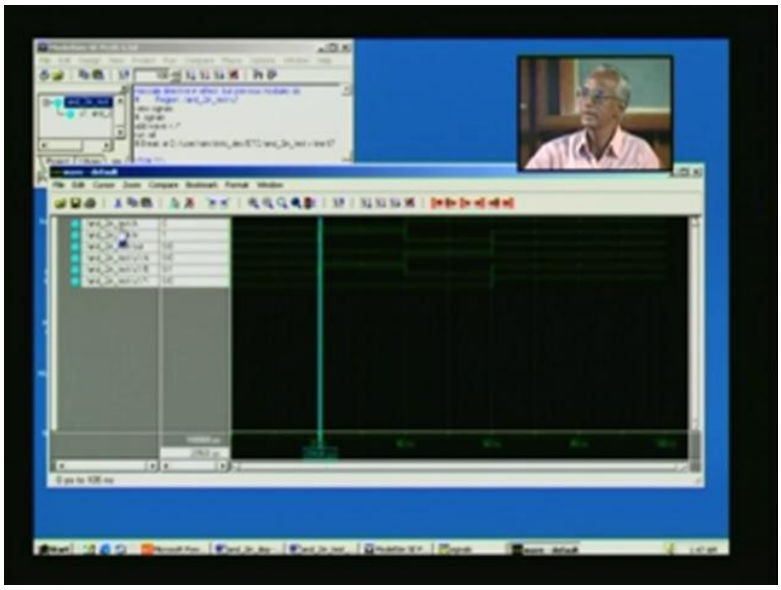
Upon clicking, it gave a waveform, and all of them seem to be flagged. In a minute, we will get everything working together. Also, you see it had run through your test bench that is source is and underscore 2in underscore test. Having run through the test bench once again it will be difficult for you to read this here and I will open another file with bigger fonts.

Finally, it has stopped at $finish because it has finished the thing. Since we do not require this source file, we might as well close it here. Similarly, if you do not want that signals also can be closed. But let us not worry about it. So now, what we will do is… we see a time axis here; right at this point it is 100 nanoseconds. This is not the correct range.

Let us see whether we can shorten it further. Now you see waveform appearing here. You can use plus to zoom in and to zoom out you can use minus here. It happens two times every time, I think. You can shift here, you can go to any part of the waveform. We had I think 80 nanoseconds or so. For that, we have one more here (Refer Slide Time: 43:28 min). There is one lens here. It says zoom area. If you want to zoom to a particular area, let us say from here, you can just press zoom area. There is a cursor in plus mode. If you just say this, it will zoom to that. Now, suppose you want the entire 80 nanoseconds, you just press this here. Say it was 100 nanoseconds perhaps. So, it has given the entire waveform and there is also a cursor here.

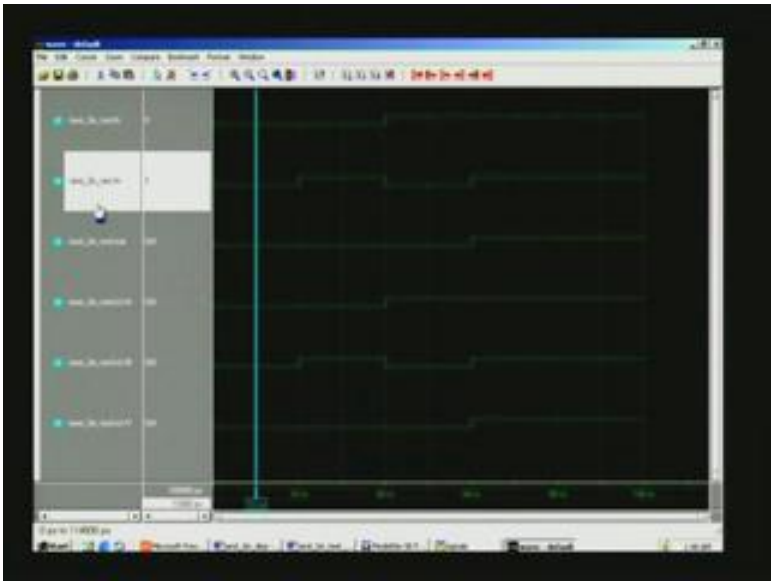
Now what I have… before that, these waveforms are very crowded. So we will change this here.

(Refer Slide Time: 44:14 min)



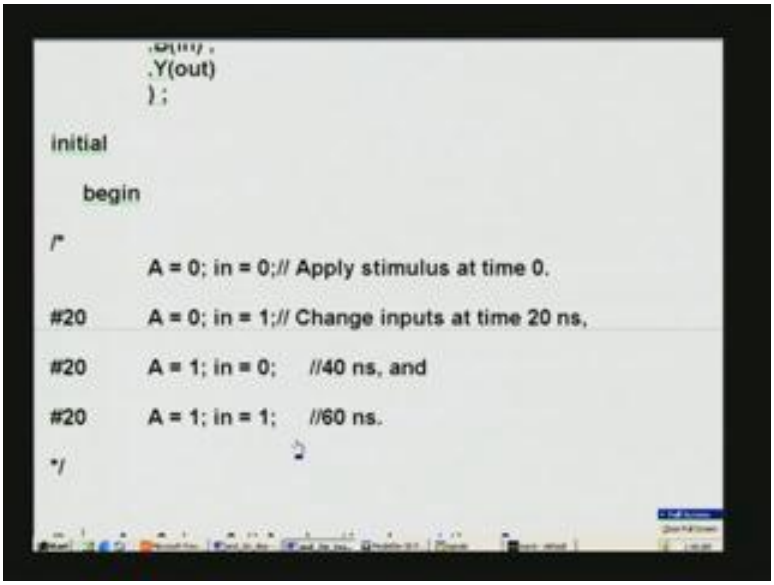
I just used shift and clicked on the first signal and last signal. It has highlighted the whole thing. We need to increase the spacing. In order to increase the spacing, we will say format, then height; so it is 17 pixels, it is marked here. I will make it as 90 pixels. Let us see what happens and then say apply. Then say okay. Now you can see the waveform here. I might as well blow it out. Now you are able to see the waveform, the complete waveform for the entire say 100 nanoseconds. Now let us see the functionality.

(Refer Slide Time: 44:59)



If you just click somewhere here, all the highlighted signals will go. You can go to any signal here, and now let us see the functionality. I can put the cursor anywhere; here, here, here and so on.

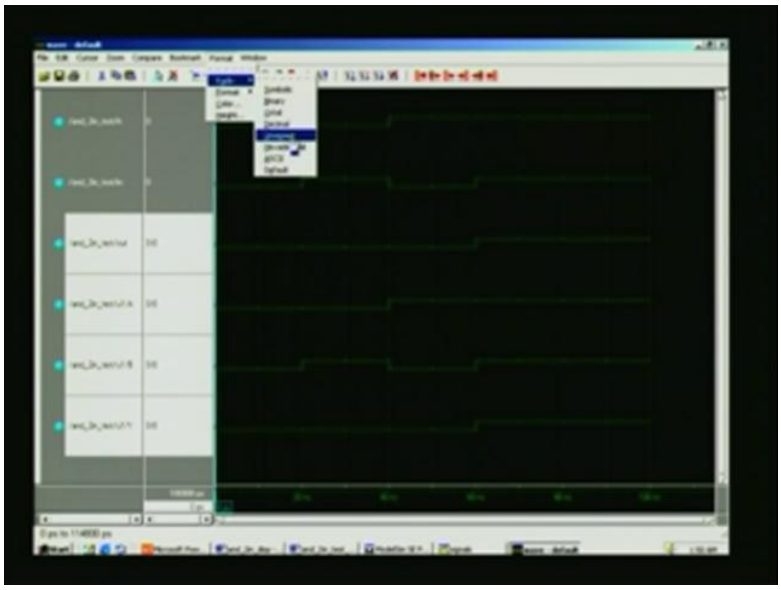
(Refer Slide Time: 45:30)



We will open the test bench here. What it says here is, we are applying at 0 point of time 0 0, then 0 1, 1 0, 1 1 for which we need the AND output. Let us see once again here the waveform that you have. Can you see the wave form here? No? That is why I am reading it out. This is on the design, I mean test bench, a, in and out - this is a waveform, and here it is 0 nanosecond and 20 nanosecond is marked here, 40 here, 60, 80 and so on.

As per the test bench, every 20 nanoseconds we are changing the inputs. For example, right at 0 point of time, we have 0 0. You can just see here, this is 0 for a, and for in, it is 0. Similarly, for a and b also, you can see that as 0 here. If you want this, this is called symbolic representation. If you want, you can have it in binary also. You can just say format, then radix, then say binary and okay. I did not mark. Right from here, just mark all this.

(Refer Slide Time: 46:57 min)



Click here and then with a shift, you click here. Then all will be highlighted. Then you say format, radix and then you say binary. So it has changed here. Instead of s t 0, it is now 0, implying that the whole thing is in binary. If you want in decimal, you can select right from here. See all decimal, octal and signed hexadecimal are all there. Symbolic is the one which was there as a default. We will see more of this later on.

Now coming to this, we see that 0 0 is the applied input. We can see the waveform. I am sure you can see whether it is low or high. Can you see this low and high there? Thereby you can recognize. This point is 20 nanoseconds; so it is 0 here and 0 here. You can also read it out here. At every step of the truth table, you can read it out straight here. Finally, its output is also listed right here. For 0 0, and in operation is 0, so you get a 0 here. Here also you can see pictorially, w have already seen earlier timing diagram. What you have here is precisely the same timing diagram.

Suppose it goes for 20 nanoseconds here, just click the cursor here, it is 0 1 as the input here. Once in again, a and b are also 0 1. So this proves that test bench and design are basically the same signals. As far as same signals are concerned, you can see the waveform right here. Suppose you want to locate this b closer to in, you can just hold the mouse button and then drag it. It has located b; so you can compare the two waveforms

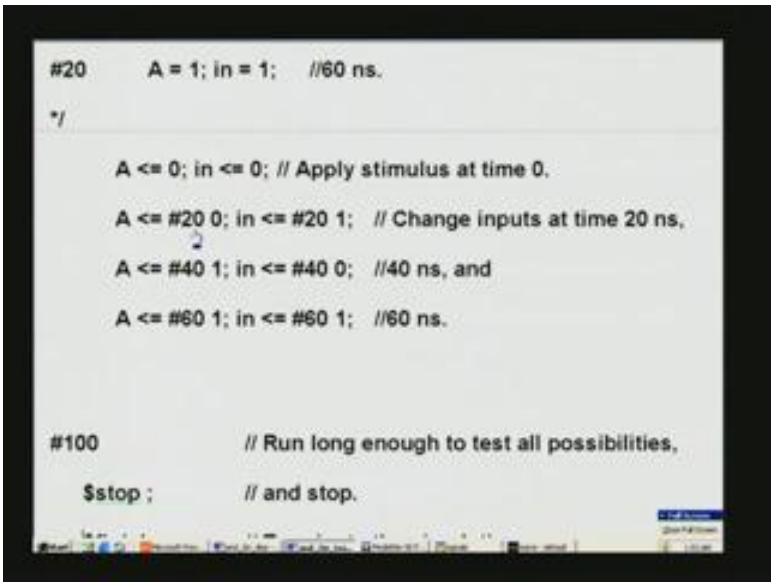
and see whether they are the same. I will revert it once again. So, now we have tested for 0 0 conditions. For 0 1 condition, this is 0 1 here. It is high. Now this is 0; so it is 0 1 condition. What should it be if you AND this 2, you should still get 0. That is what is here. It shows 0 here.

Similarly, for the next, at 40 nanoseconds, we are again changing here. That is now 1 and 0 are applied. The stimulus applied is 1 and 0 for a and in, and that is what you have high here and low here. For this, naturally, you had to have again 0. The last one is 1 1. This corresponds to 1 1 here and corresponding output must be 1 here that is what you see here. This confirms that your design is working; in other words, the functionality is fully tested. There is no gate delays etc., involved right here; that can come only with the place and route which we will be doing in future.

Another point I would like to mention here is - you just notice here I have commented out at the earlier thing that I had mentioned (Refer Slide Time: 49:53 min); one of you asked a question whether this 20 20 20 can be put in a different way to reflect straight way the actual timing? Yes, we have experimented with it. We arrived at another thing that is listed here.

That is what is actually working. What you have seen just now is that which is working. So I did not run the earlier thing with same 20 nanoseconds appearing. Here you can see 20, 40, and 60 and so on.

(Refer Slide Time: 50:22)



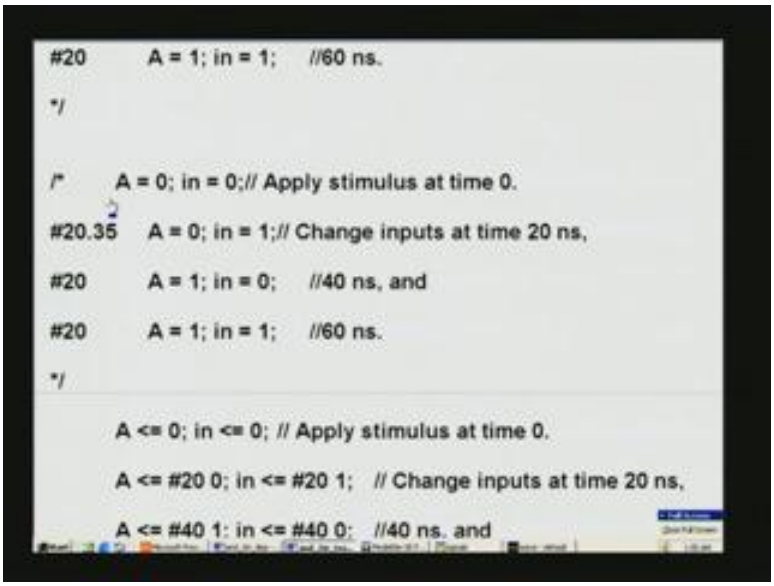
Here the advantage is that whatever you put, you get right the same time. But disadvantage is you had to put for every signal once again here. You had to pay a price for that; so you have advantage as well as disadvantage. Another disadvantage in this approach is in sequential circuit that you will be seeing, we want to initiate a data at a specific time prior to the clock's striking. It was shown at 25 nanoseconds.

In the earlier case adopted it was this and this approach. Here at 25 nanoseconds I had put a data there. That means once you put that one automatically all are relative; so all are offset by the same amount. But that is not the case here in this particular example. It is not an offset, but the actual time itself. So that is another disadvantage here.

The designer is free to choose one of these two depending upon the needs, circumstances, etc. Another point I would like to mention is there is one more here. We have also seen 100 picoseconds appearing in 1 nanosecond time scale. That is the resolution we have said. If you uncomment this one and comment the other things, this will come into force. In which case, we can see that this 0.3 has also taken into effect. This we will see later on.

Thank you.

(Refer Slide Time: 51:42)



(Refer Slide Time: 52:34)