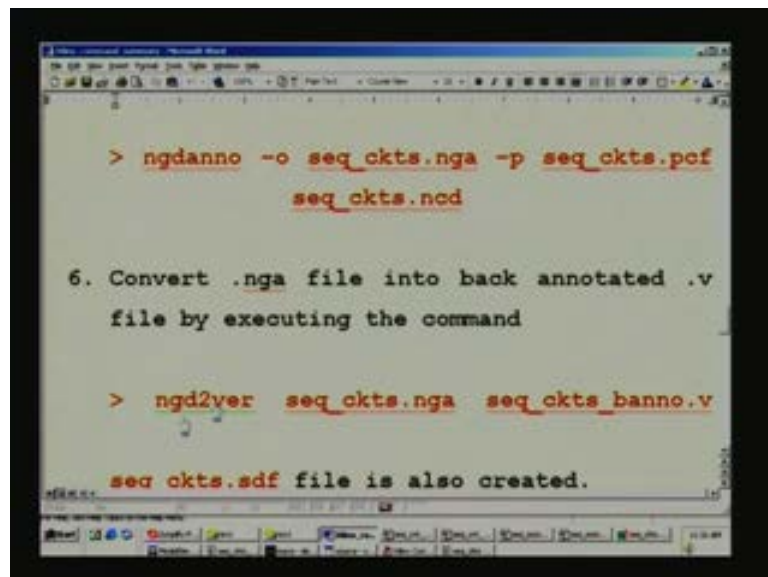


**Digital VLSI System Design**  
**Prof. Dr. S. Ramachandran**  
**Department of Electrical Engineering**  
**Indian Institute of Technology, Madras**

**Lecture No. # 35**  
**Xilinx Place and Route Tool**  
**(Continued...)**

(Refer Slide Time: 02:33)



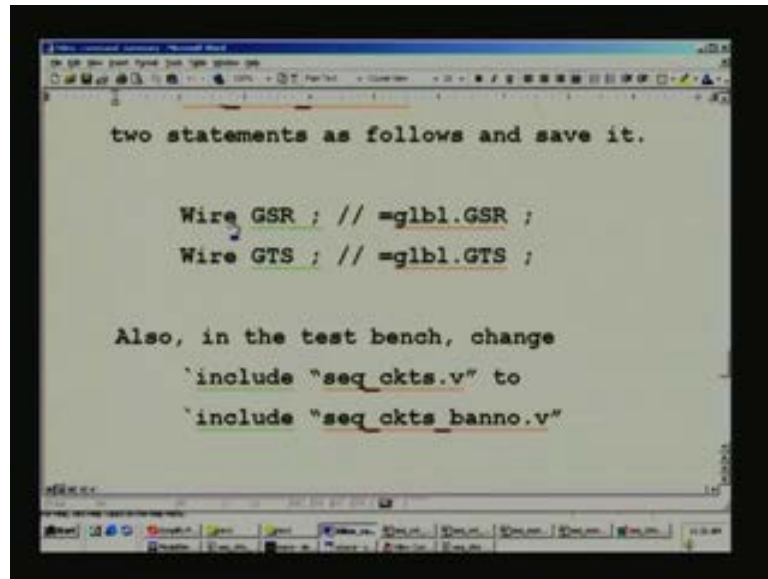
```
> ngdanno -o seq_ckts.nga -p seq_ckts.pcf
seq_ckts.ncd

6. Convert .nga file into back annotated .v
file by executing the command

> ngd2ver seq_ckts.nga seq_ckts_banno.v

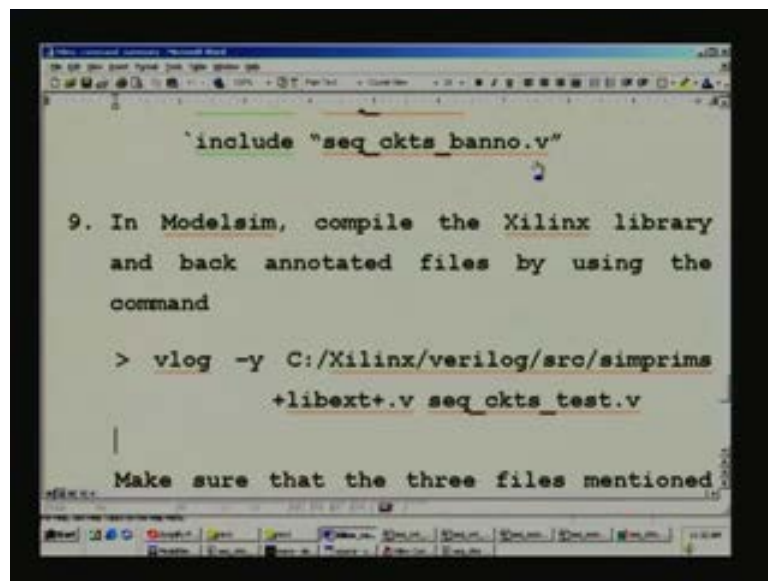
seq_ckts.sdf file is also created.
```

(Refer Slide Time: 03:20)



```
two statements as follows and save it.  
  
Wire GSR ; // =glbl.GSR ;  
Wire GTS ; // =glbl.GTS ;  
  
Also, in the test bench, change  
'include "seq_ckts.v" to  
'include "seq_ckts_banno.v"
```

(Refer Slide Time: 30:31)



```
'include "seq_ckts_banno.v"  
  
9. In Modelsim, compile the Xilinx library  
and back annotated files by using the  
command  
  
> vlog -y C:/Xilinx/verilog/src/simprims  
+libext+.v seq_ckts_test.v  
  
Make sure that the three files mentioned
```

So far we have seen how to back annotate design files by using primarily two commands: it is a ngdanno and this the output option, and these are all, this is the output file and input file being this and which was obtain by xilinx place and route. Then final back annotated file, which is sequential circuits underscore banno dot v which we gave and was derived from the input file nga which is the output of this first phase. You need ngd2ver log conversion to do effect this back annotation file and a standard delay format file was also created, and then, we also saw that we need to replace some two statements by commenting out this two here, as shown here.

Now, in the test bench, we should make sure that this back annotation file is added in place of dot bm file. We had used earlier for the optimization, and that is, I think this they want. So, if I will just increase the font size for this, so, we can see this. So, include sequential circuits underscore banno, I have already done this, and with this will be running.

So, now, next step, let us see, will just follow this is to compile. Before we compile, we will have to open the modelsim. The primary goal is to back annotate, and get a file back annotated file. This is very much shocking to the source file, your design file, and this file, will have to be taken into the modelsim and simulated. Make sure that after back annotation, the gate delays take effect that is the primary purpose of doing this. In order to do this one, let us open, I mean, if modelsim is, I think is already open here, and so, we had to key in the command here, and that has also been done earlier. I will just show.

Command that we have to give is, vlog is another command version. This is the prompt; it is like a Unix prompt and inside the modelsim window and vlog is one which recognizes as a compiler. What we have to compile is library file because after back annotation, we will get all the primitive cells coming into this file.

So, basically the mugs - LUTs - which I had told you earlier and these primitive cells will get reflected into this file. If you open out back annotation, you will precisely see this, and you have to compile that file, that is, this file is actually inside a test bench; that is what we have to already seen some time back, and wherein, we saw that this the one, this is the test bench actually. In that we have added included back annotation file instead of dot bm file which we did for optimization earlier.

Now, in addition to this test file which contains the back annotation file, you have also to mention some library files required for recognizing the modules of the primitive cells, etcetera. They are all basically modules and they will be recognized and or located in the xilinx path where it is the primary xilinx software is loaded and that entire path will had to be given here, and include this library there. So, this includes all the primitive cells library; along with that you have to compile, and of course, your back annotation file: when you run this, it does the compilation.

(Refer Slide Time: 06:47)

```
> vlog -y C:/Xilinx/verilog/src/simprims
+libext+.v seq_ckts_test.v
```

Make sure that the three files mentioned in 7 are present in the current working directory in Modelsim. Check it by using "dir" command.

10. Create a new work directory and load

(Refer Slide Time: 06:53)

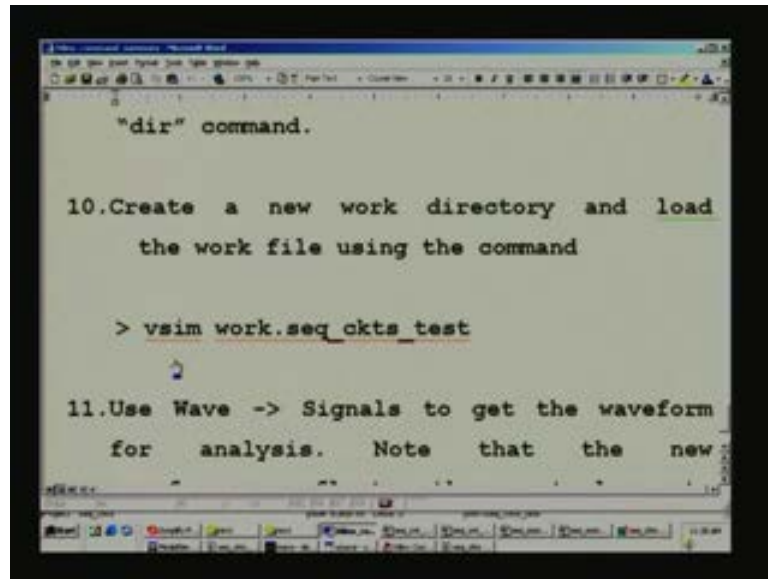
```
7. Make a new directory and copy back annotated files .v, .sdf and the test bench, seq_ckts_test.v into the same.
```

8. Edit seq\_ckts\_banno.v file and comment out two statements as follows and save it.

```
Wire GSR ; // =glbl.GSR ;
```

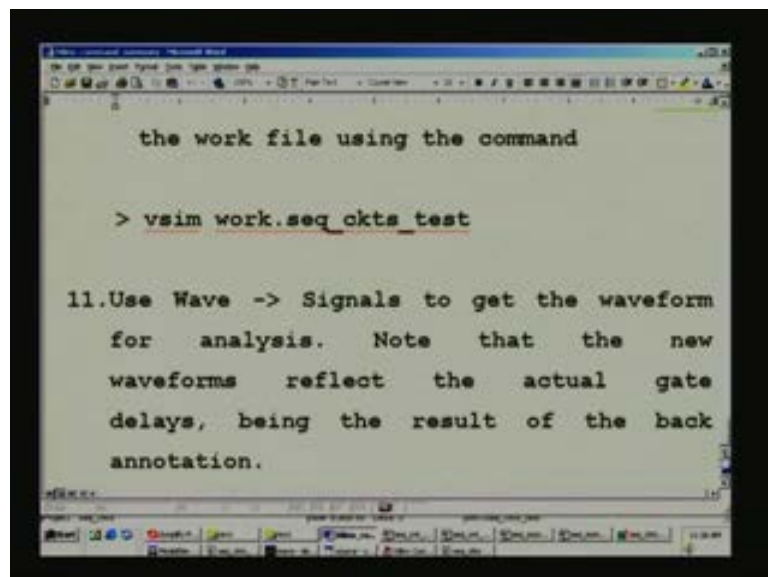
And after this, we had to make sure that the three files mentioned in seven, that is previous step we have done here, that is here, we had a dot v back annotated file, then sdf file and as well as your test bench which is edited for including back annotation file. So, this things, three files are what is referred here, that is what is referred here, and check it by using, you can use a directory command to, see, make sure that these files are present, otherwise, you will get an error if one of the files is missing.

(Refer Slide Time: 07:27)

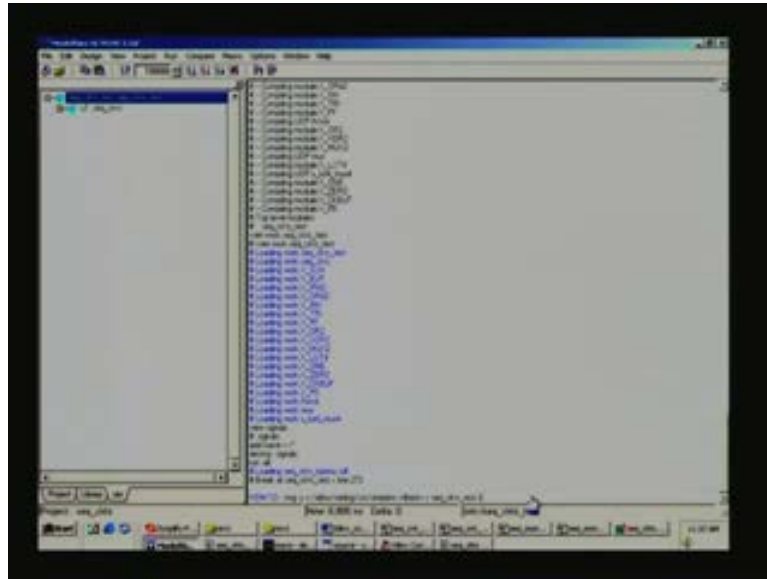


Next step is we will see this step and then go to the modelsim in order to do both. It is basically two comments that you have to execute in the modelsim prompt. Next step is, suppose a new directory is not created already, you may have to create a work directory there that is what it says here. Create a new work directory and load the work file using the command vsim.

(Refer Slide Time: 08:25)



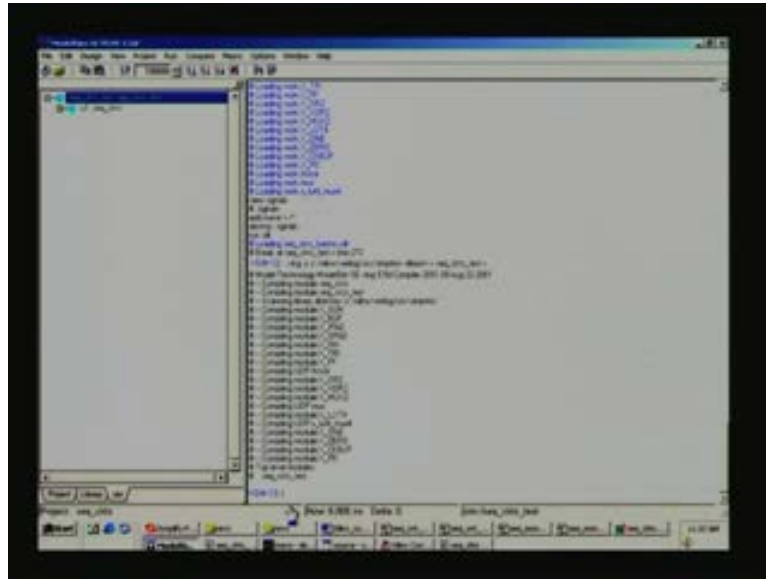
(Refer Slide Time: 08:54)



Vsim is the one command for the stimulation and its actually for loading the design file because we have compiled with the back annotated file and that corresponding file will have to be loaded. That is what we say, whenever you have a compilation, it should be followed by simulation, and that is what we do here and naturally that is in work directory and the name it has is precisely same as your top module which is the test bench, which includes the back annotated file.

So, we will do this together and we can even finish the last step. You can use as usual wave signals in order to see the waveform for analysis, and since there is a difficulty in you viewing this on the TV monitor, we will as usual show it on paint and note that the new waveforms reflect the actual gate delays being the result of the back annotation. So, this is what you have to bear in mind. So, now, let us see the modelsim window, that is here, so, those first is compilation, this has been already keyed in here, and this vlog minus 5, all that what we have already seen and sequential circuits underscore test dot v.

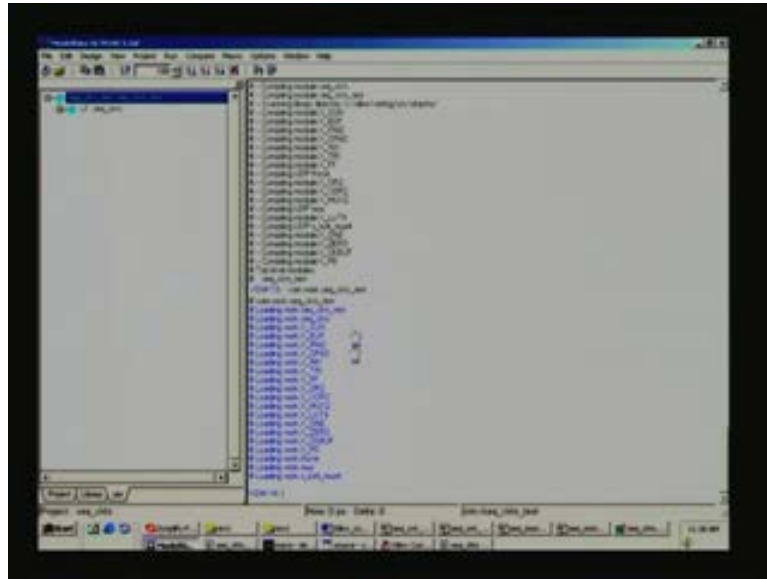
(Refer Slide Time: 09:13)



So, let us execute this just give a return. So, it has done the compilation. During the compilation it say, I mean it reports compiling module x underscore buff, then i pad o pad invert tri-state buffer flip flops or gate exclusive or mugs and so on.

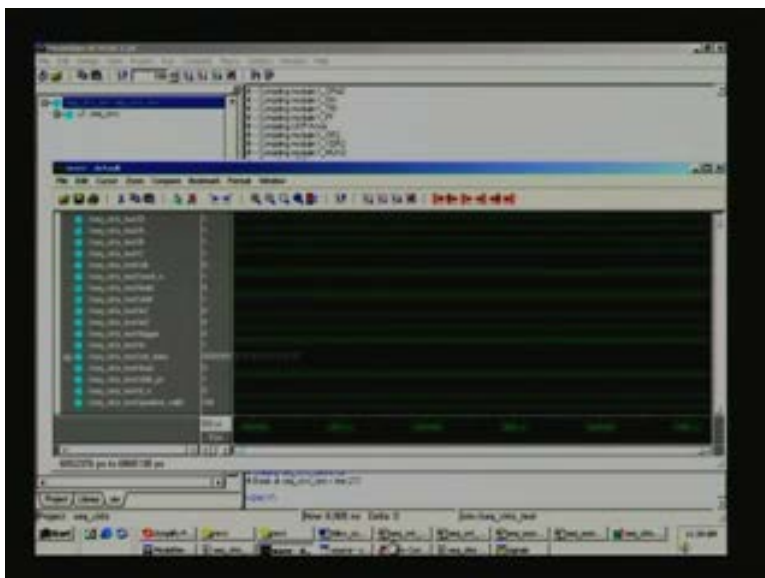
So, this is as a result of including the entire xilinx library as such here, and thereafter, you had to have the test bench also compiled and that is already done here. Prior to this it compile's your design actually; this is a back annotated design and this back annotation is not specifically mentioned because what we had declare as a module is actually sequential circuits in the design. So, we did not change the module name; so, what it reports is only the module name as you see here. So, it has successfully done the compiling work now. So, what the next step is to load this file.

(Refer Slide Time: 10:27)



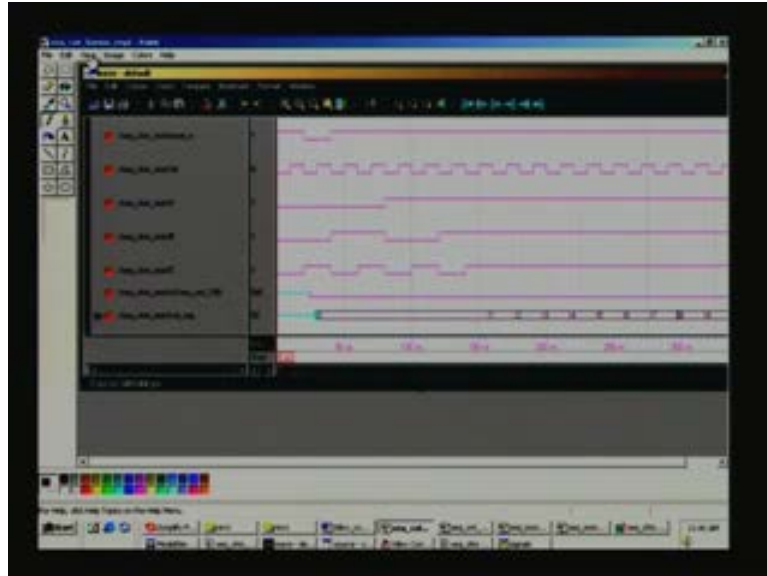
This is vsim work dot sequential circuits underscore test. This has loaded once again you can see the i pad o pad all are reported here; it is loading module by module; each one is called work directory, and along with the extension, whether it is a test bench, or the design or the library here. The next step is to see the wave form, so, which we can use view, then signals, and then, when the signals window opens, then use view wave; then the last option is signals in design.

(Refer Slide Time: 11:09)





(Refer Slide Time: 11:45)



So, this is what you are already familiar and it is already open here. So, now, what we need to do is run this. So, it has run and once again this is crowded here, we can't make out anything here. So, what will do is I will directly open some of the examples; I will consider a couple of examples in order to explain the result of back annotation.

So, as I mentioned earlier, the gate delays must manifest here in the wave form. Let us first open what is called as sequential counter which we used for the after optimization; we saw small delay and this is what it is. Before zooming in, I will just explain what it is. The whole scene is available in just a single page.

So, you can seek here a counter reg being reset 0 corresponding to a reset been here and you note that reset happens right here, whereas, only after delay, this 0 has been reset. So, earlier it was not the case. So, probably in the optimization, even in that phase, I think you did not get any delay as such. There was a delay in somewhere else; so, may be when the counter started. Now, if you see here, this is the positive edge pulse here of the clock, and the actual counter advances not at that one, but only after delay. For example, this may be around two nano seconds or so, and every count is always a delayed by the same amount. Once again you can see here as and it is counting from 0 1 2 3 and so on, right up to 9.

So, we will see one more view of this towards the end of this count. So, the counter is, I mean it counts from 0 to 255 in steps of 1 and then revolves back and it keeps on going like that in a cyclic fashion. There are other signals which we need not really worry about because we have already seen couple of times earlier. So, these are all to advance the counter, then reside the counter using inputs here. I will first zoom, so that you can clearly have a look.

So, we can see reset clock a b c as input here and finally a reset count. Here, you can see that after back annotation, I could not find reset count as well as advance count. Advance count is totally missing, as I mentioned it was used in other circuits. So, naturally that got drop because similar such signal was there and some other name. So, at the time of optimization after synthesis itself, it was missing advanced count, and here also it is not to be found after back annotation, and here reset count also got changed little, so, it has renamed as some 199.

So, it takes some more detailed manifestation here as far as the signals are concerned, I mean, in the sense, if it is a multibit, where it reported as one group earlier. In this back annotation, will, it will separate out all the signal's and report separately and it will be very tedious for you keep track of the count, etcetera. So, we may not be in a position to see the actual count value in some cases, perhaps we are lucky to see here in this case, back annotated file only, as the name implies here, and opening this one, the count goes in this fashion 0 1 2 3 and so on. You can see a b c becoming 1 that is the condition we set for the counting to start and that is precisely where it has happened. Now, we can have a look at the difference. See, here, the positive edge of the clock occurs here; where as the actual count takes place only here.

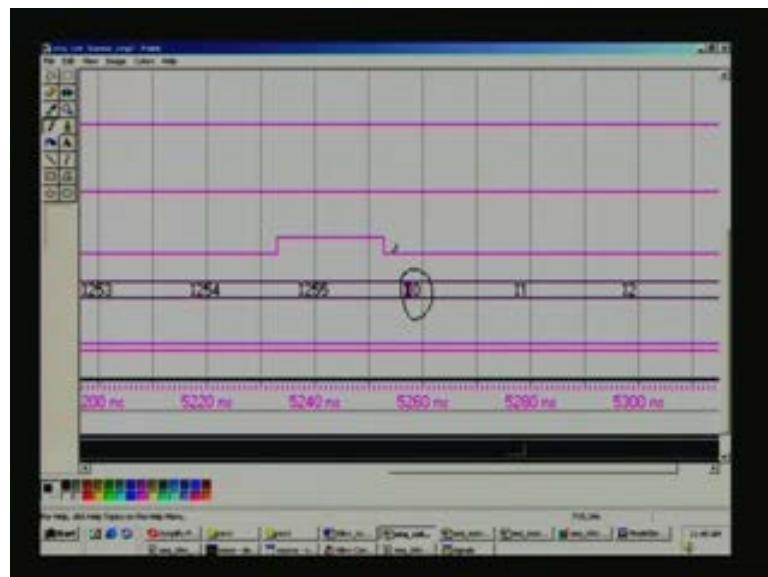
So, this is clearly due to the gate delays associated with the circuitry for that particular path, and so is the case, we can see equal amounts it is shifted from every clock here, and clock frequency is I think same 20 megahertz, I mean 50 megahertz, 20 nano seconds I think.

So, we will open another, this is going right up to 10. So, we will continue towards the end we will see. So, that one is here and you can see 250 251 and a b c inputs are continuing to be one and reset count is active only here at the time when it is 255. You remember that reset count is basically an assign statement earlier and that will have to

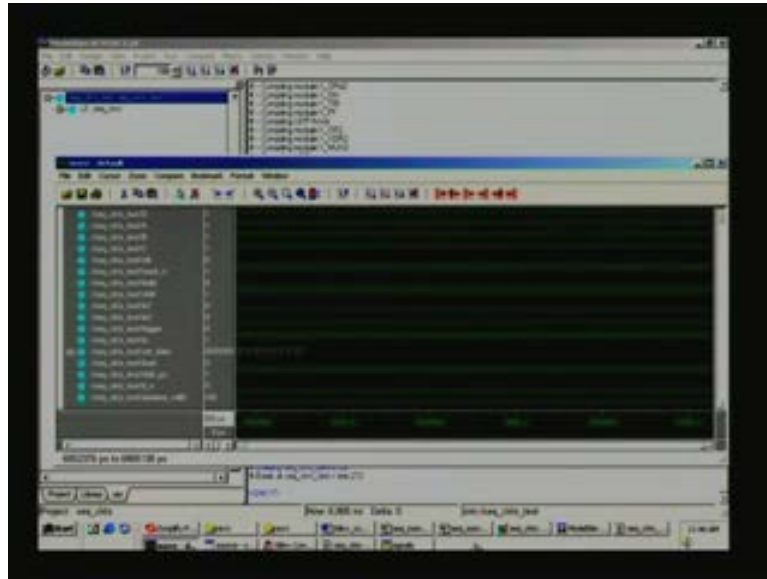
get reflected the moment 255 is encountered in the counter register. What surprising think here is, it is happening in advance.

I cannot offer an explanation, may be there is a bug in the software. So, probably we may have to write to the modelsim and find out why it is so. That is not a serious thing, its happening in advance here, both the edges you can see here, but any way the counting go, I mean after when this pulses encountered, it has to clear and that is what is happening here. You can even see here, some changes here; before going to 0, it goes to some different values, but finally it settles down. So, if i zoom, we may be in a position to see little better.

(Refer Slide Time: 17:33)

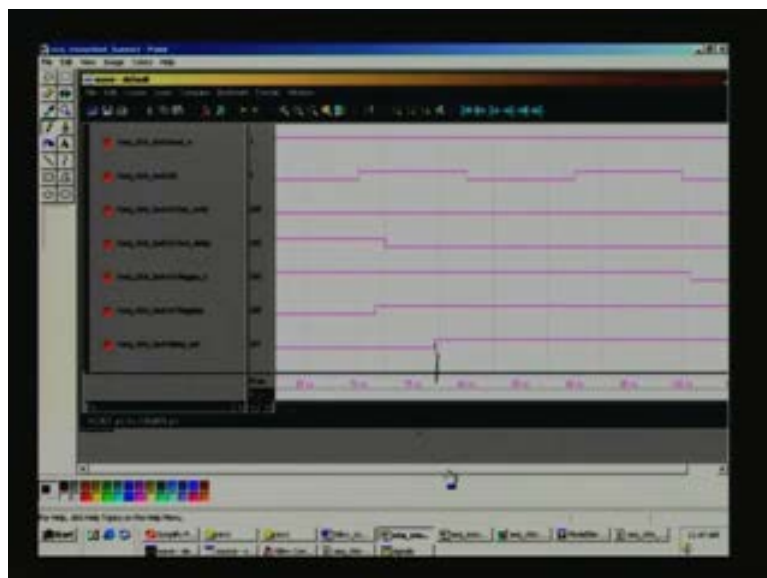


(Refer Slide Time: 18:00)



So, we can see 250 251, etcetera. This is the counter reg here; and you can see that pulse here; and you can see this activity here, some ringing like thing. In fact, it is changing to 2 3 different values and finally settling down at 0. That is not a handicap because this 0 will be sensed only at the positive edge, when it is stable. That is what the synchronous circuit is all about. In addition to this, we will see one more example, that is we have seen a mono shot vibrator, and here, it is too crowded, we may had to going to the searching for all the signals, and any way, you would not be in a position to see that is why we are seeing that paint zoomed version.

(Refer Slide Time: 18:15)



So, will now see, there are two more couple of wave forms for the mono shot. This is, here, in mono shot, we have once again a reset, then clock, then reset for the counter. The counter is c n t d here, and as I mentioned earlier, some of the counters are totally separated out. If it is, I think this is 16 bit counter, is it? If it is so, you will get all the 16 signals listed; you may have to manually group them. That will be very tedious; that is the reason why I could not show you a counter here.

So, that is not a handicap because what we are really interested is the final output itself; this delay out is the final output. We have trigger p trigger c, based upon 0 1 condition, we set another signal called run delay, which is, which simulates this short pulse of the mono short. So, it is high here; and goes low here. So, when it becomes high somewhere here, probably just before I should have captured one more on the left, but that is not a major issue. So, what we are concerned is the only delay out which is the required mono shot operation that we are looking for. So, the idea is we give a pulse for, a short pulse for the mono shot and the timer runs and it starts running right at this point. If you see here, it is 75 nano second. This is precisely at 77 here, and you just remember it is 77 here and it keeps on, it is high here and I will zoom this.

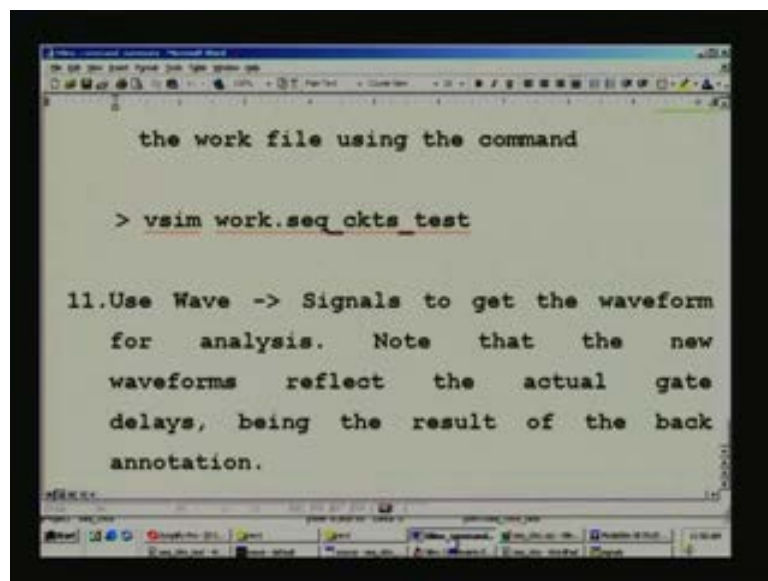
It is 77 here. This a fourth point here. So, that means 2 nano second 2 plus 75 77 here, and towards the end let us see. This is the delay out that we have; this is the decide output, and after giving the trigger, it goes high here. Naturally only after the second clock pulse, it will start. All that we have already seen, so, we do not have to see that. In addition to that gate delays also became operational because it is a back annotated file. Therefore, it is much longer it is taking, and only here it goes. In spite of the fact that clock is somewhere way off, somewhere here, this is a clock, and we will go towards the end of this delay out and that is there in the second.

Towards the end, I will zoom here. It is around 5 1 8 0. We started with 77 nano second remember, so, the difference is 5 1 8 0 minus 77. If you take 77 as 80 5100, and 3 extra we had taken.

So, it has given a time delay of 5 1 0 3 nano second. Now, as per our program, it was setting of 255 and we also saw that clock is 20 nano seconds. So, 255 into 20 is 5100. So, if it have the timer had run precisely, it would have made 5100 exactly, but unfortunately we have some 3 nano seconds extra here. So, that is the accuracy that you can have in

this design. This inaccuracy is created not by us or by our design, there is no design flow such, and the basic thing is the path delay has made all the difference. So after all delay out is depend upon to start with, when it is switched, it may have one particular delay; it may have to go through set of gates, and when it is switched off, it may have to go through another set of condition. So, that means another set of gates is coming into picture. So, but in this two sets, whatever is the propagation delay difference, will manifest as the error. So, and that is what you have here three nano seconds explained, is it clear?

(Refer Slide Time: 22:43)

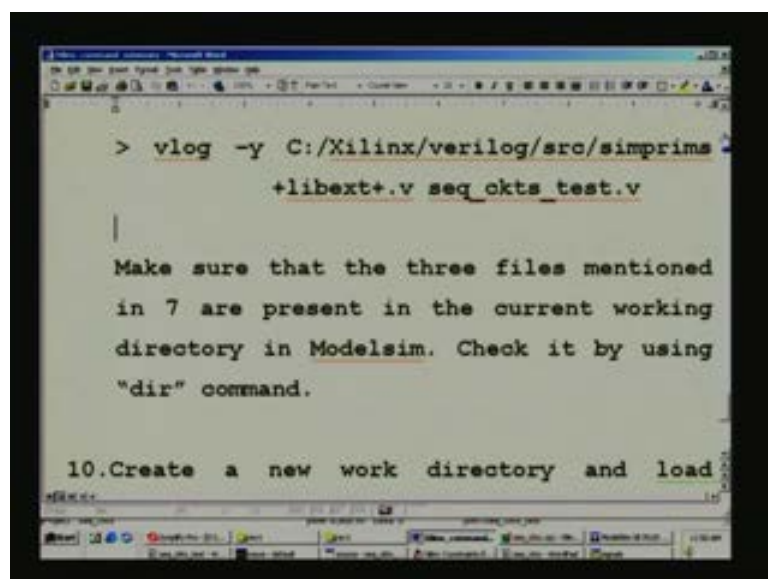


```
the work file using the command

> vsim work.seq_ckts_test

11. Use Wave -> Signals to get the waveform
for analysis. Note that the new
waveforms reflect the actual gate
delays, being the result of the back
annotation.
```

(Refer Slide Time: 22:54)

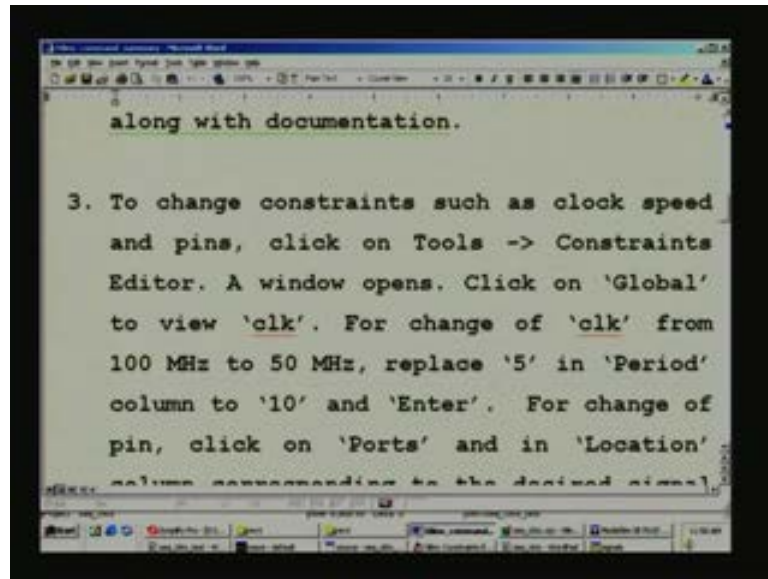


```
> vlog -y C:/Xilinx/verilog/src/simprims
+libext+.v seq_ckts_test.v

Make sure that the three files mentioned
in 7 are present in the current working
directory in Modelsim. Check it by using
"dir" command.

10. Create a new work directory and load
```

(Refer Slide Time: 22:58)

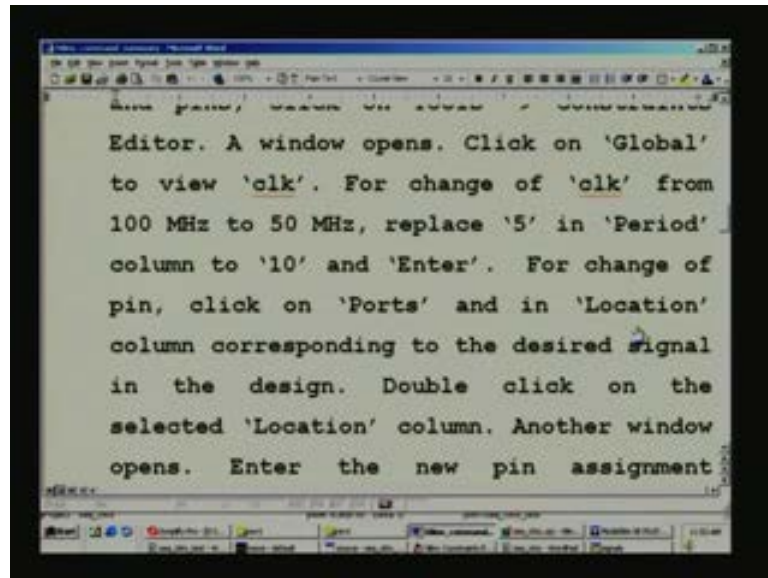


So, this completes the back annotation. So, what we will do is, next is, we will go back to that xilinx command summary. That is what we have seen using wave signals; we have seen the back annotated files. So, what we have done? I have forgotten two other aspects earlier, that is we have been handling constraints. I mentioned that we can take the constraints straight away from a file and open one dot ucf file, and so, will have little more insight into this constraints because this is an important thing that we may to have to handle quite frequently, if you want to make better timing closures, that is, if you want speed of operation, you may have to have all this, you should know, otherwise, you cannot try to get a better speed of operation.

So, I will just go through this. So, first is we will change the constraints for, there are basically two constraints: one is clock, other is IO pins actually. We are dealing with FPGA design, so, we are going to have a peep into how your design has been mapped into the FPGA that will be seeing. Before that, will see what a clock constraint is; how to give this; I will just read out this. To change constraints such as clock speed and pins, click on tools constraints editor. This is naturally in xilinx place and route we have to do this. A window opens, click on global to view clock, and for change of clock from 100 megahertz to 50 megahertz, so, we have programed it earlier for 100 megahertz. So, naturally, half time is 5 nano second; 10 nano second will be the time period, so, half time is 5 nano second. In clock, if you remember in ucf file, that is what we have given - 5 nano second, because it is, as a result of which, we had 100 megahertz, and actually,

this has been taken through the simplify tool - synthesis tool - there in we specify 100 megahertz. So, automatically that has been taken in to account.

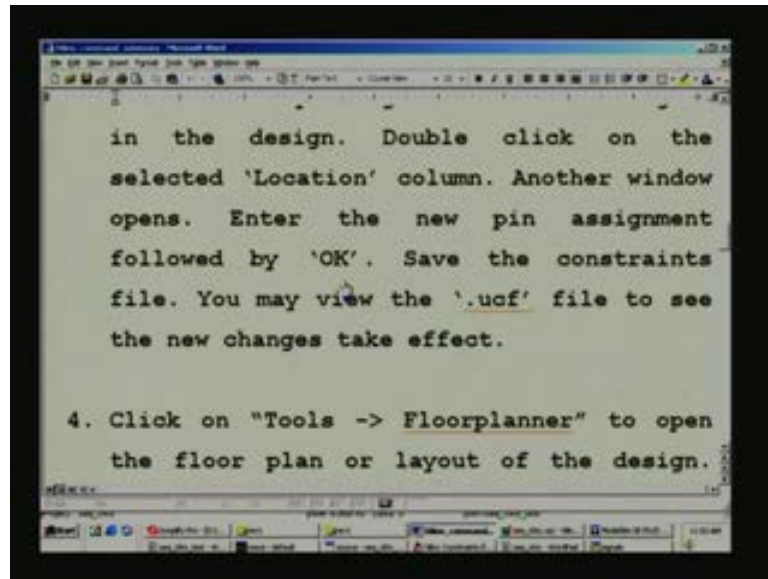
(Refer Slide Time: 25:06)



We change this parameter here to 50 megahertz, that means 5 will have to be changed to 10 or any other thing you can change, and see, whether it is get reflected. Then replace 5 in period column to 10 and enter. For change of pin that is for change of clock, and in the same window you have one more option called ports. For change of pin clock on ports click on ports and in location column corresponding to the desired signal in the design. This, I mean all the signals in your design will also be listed, and identify the design I mean particular signal that you want and then click on that, the location. Double click on the selected location column. Another window opens. Enter the new pin assignment followed by ok. Save the constraints file. We may view the dot ucf file to see the new changes take effect. After this dot ucf file would have been rewritten with the new constraints that you have taken.



(Refer Slide Time: 25:36)

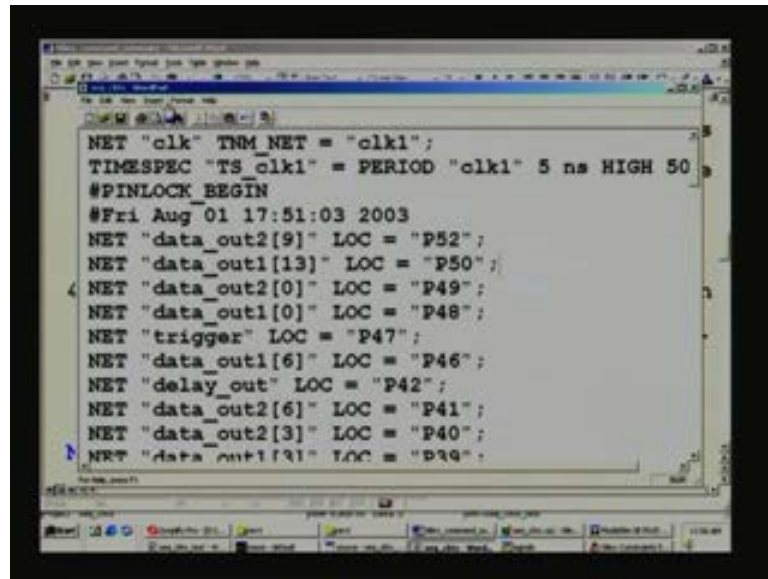


And after that we have one more what is called floor plane. You can actually see where your circuit is residing. So, you can have in terms of the primitive cells we have seen earlier that also we have look, and before that, let us have look at the constraints first.

So, we have, this is the constraints window, and before opening this, I will, this is the xilinx window already open. You remember we had seen this earlier, and it is this version that we are version 3, revision 1. It can state away from the synthesis, just simplify synthesis. Now, what we need to do is we go for tools, and then, in this you have what is called consent editor.

So, when you click on that, it opens here, and if you click on this, you have ports here and you have another thing called global here, and it is, here you have to mention what it is listing is actual clock here and this is 50 nano second high it says and I had change in between, that is why it is like that. Now, I will change back to and actually this 50 nano second is reported here. If i change the i, I will remove that one and it is 5 here. Just enter, that is what we have there on that commend summary, and immediately this 5 nano seconds has taken effect here, and so, next thing is, that is how you change the frequency; you do not have to worry about any other thing nano second high 50 percent; it is 50 percent due to cycle it goes from 0 to high and that is what the clock is for and clock is listed here. All other files also, I mean signals will also be listed, provided you, get from that ucf file, once again which is not done here.

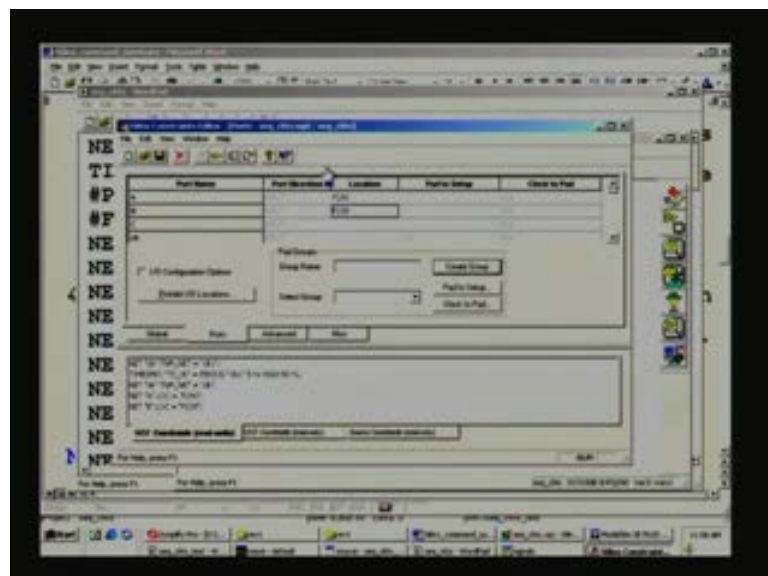
(Refer Slide Time: 28:12)



```
NET "clk" TNM NET = "clk1";
TIMESPEC "TS_clk1" = PERIOD "clk1" 5 ns HIGH 50
#PINLOCK BEGIN
#Fri Aug 01 17:51:03 2003
NET "data_out2[9]" LOC = "P52";
NET "data_out1[13]" LOC = "P50";
NET "data_out2[0]" LOC = "P49";
NET "data_out1[0]" LOC = "P48";
NET "trigger" LOC = "P47";
NET "data_out1[6]" LOC = "P46";
NET "delay_out" LOC = "P42";
NET "data_out2[6]" LOC = "P41";
NET "data_out2[3]" LOC = "P40";
NET "data_out1[3]" LOC = "D19";
```

Now, I will show another ucf file after with all those pins, we have mapped earlier that is somewhere here. You can see final ucf file will look like this and you have the place and route itself has mapped it on to a particular pin all your signals, design signals and clock is here and then clock period, we have seen is 5 nano seconds here. You can even edit a file, ucf file. You can create your own file and on similar lines, and you can straight away run it right at the beginning. You could have given that dot ucf file at that point of time, then it would have taken all the pins, etcetera with your mapping; your choice will get affected.

(Refer Slide Time: 29:24)

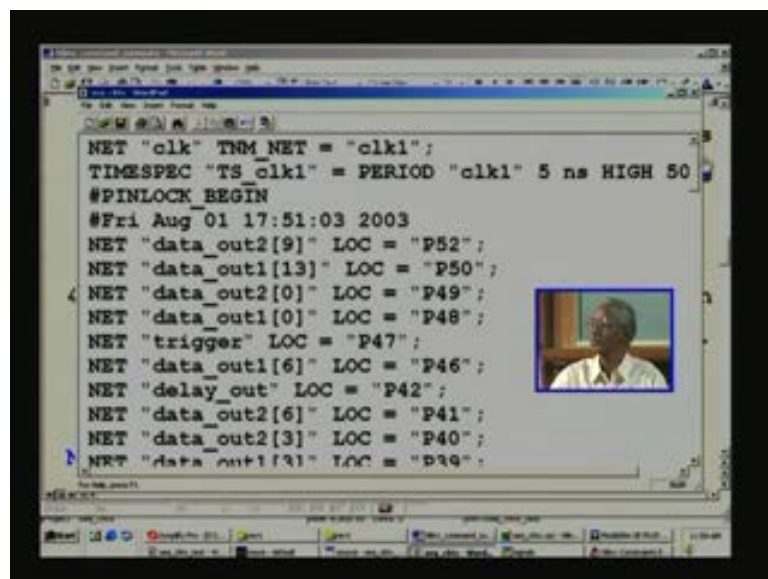


So, otherwise, you can come from these constraints what we have just now seen. From there also we can change the pins as we have already seen, and I think we did not see the pins here, change, and that was constraints global ports, sorry. So, in ports, here, there is a pin here, say for example, let us say clock are b here. So, I can key in straight away there or I can just double click here. Another window opens. We can just key in there. I say this 239 here and say ok that gets reflected here.

So, whatever is entered here also appears here in the list. So, when, now, next step is to save this. Here, if you save, it will go into ucf file. The warning that it is giving is, after you have mind, you have made a major change here which is totally different from what it was earlier, all pin configuration, your, then clock, everything must have changed. So, all this will have to take effect totally.

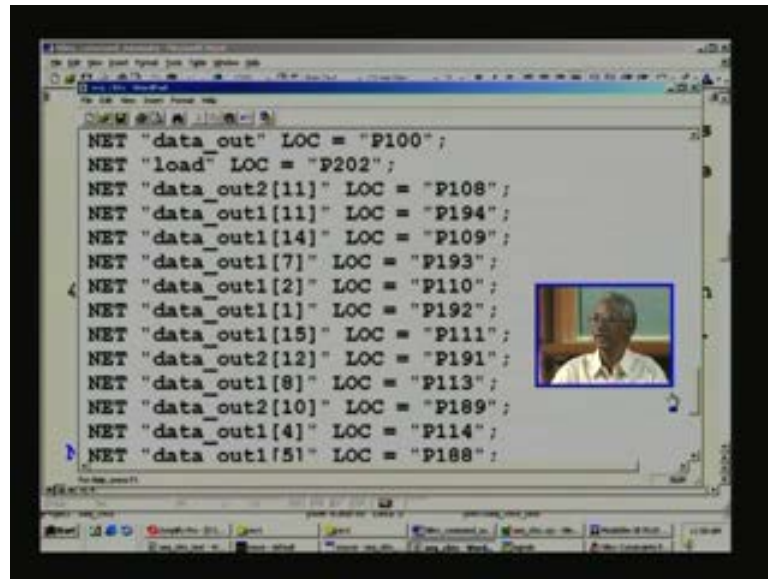
So, in order to do this, you have to go back to prior to back annotation straight away to the implementation - place and route, xilinx place and route implementation - start from the implementation. Then, do the back annotation once again and then make sure the functionalities with the pins and that is how you have to go. So, that way that is time consuming thing and it is precisely the same that we have already followed. Therefore, that particular aspect will not be shown and it is enough if just see a ucf file. With that, I think you will be in a position to remap your I Os and as well as the clock.

(Refer Slide Time: 31:07)



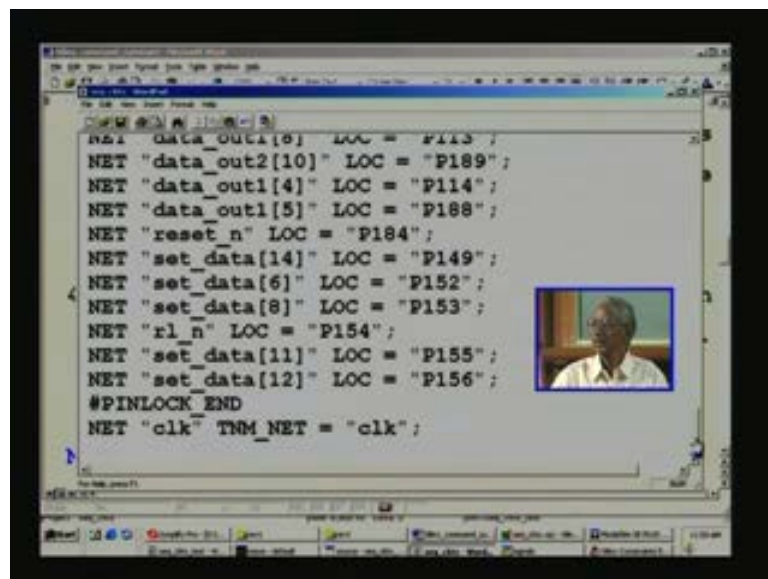
```
NET "clk" TNM NET = "clk1";
TIMESPEC "TS_clk1" = PERIOD "clk1" 5 ns HIGH 50
#PINLOCK_BEGIN
#Fri Aug 01 17:51:03 2003
NET "data_out2[9]" LOC = "P52";
NET "data_out1[13]" LOC = "P50";
NET "data_out2[0]" LOC = "P49";
NET "data_out1[0]" LOC = "P48";
NET "trigger" LOC = "P47";
NET "data_out1[6]" LOC = "P46";
NET "delay_out" LOC = "P42";
NET "data_out2[6]" LOC = "P41";
NET "data_out2[3]" LOC = "P40";
NET "data_out1[3]" LOC = "D39";
```

(Refer Slide Time: 31:22)



```
NET "data_out" LOC = "P100";  
NET "load" LOC = "P202";  
NET "data_out2[11]" LOC = "P108";  
NET "data_out1[11]" LOC = "P194";  
NET "data_out1[14]" LOC = "P109";  
NET "data_out1[7]" LOC = "P193";  
NET "data_out1[2]" LOC = "P110";  
NET "data_out1[1]" LOC = "P192";  
NET "data_out1[15]" LOC = "P111";  
NET "data_out2[12]" LOC = "P191";  
NET "data_out1[8]" LOC = "P113";  
NET "data_out2[10]" LOC = "P189";  
NET "data_out1[4]" LOC = "P114";  
NET "data_out1[5]" LOC = "P188";
```

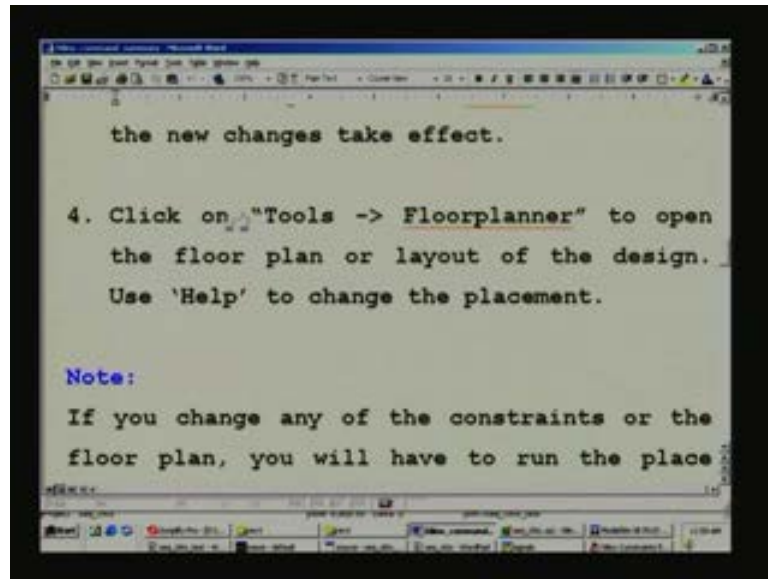
(Refer Slide Time: 31:34)



```
NET "data_out1[6]" LOC = "P113";  
NET "data_out2[10]" LOC = "P189";  
NET "data_out1[4]" LOC = "P114";  
NET "data_out1[5]" LOC = "P188";  
NET "reset_n" LOC = "P184";  
NET "set_data[14]" LOC = "P149";  
NET "set_data[6]" LOC = "P152";  
NET "set_data[8]" LOC = "P153";  
NET "rl_n" LOC = "P154";  
NET "set_data[11]" LOC = "P155";  
NET "set_data[12]" LOC = "P156";  
#PINLOCK_END  
NET "clk" TNM_NET = "clk";
```

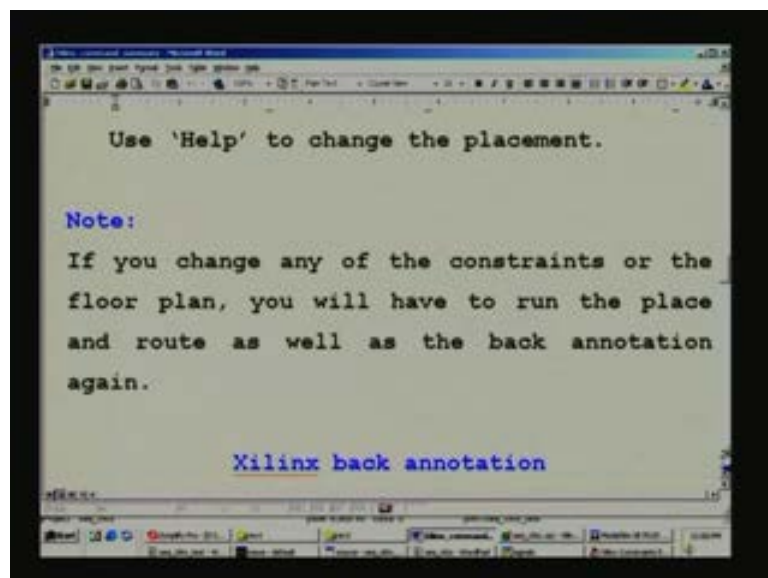
We will first see the, this ucf file, we have already seen here. So, pin 52 for data out 2 signal and so on it goes. This has been created automatically during place and route, and as I mention earlier, during place and route, you can see the report browser and get various reports such as pad report or log report. All pad report will contains precisely this and here also you can have a look.

(Refer Slide Time: 31:42)



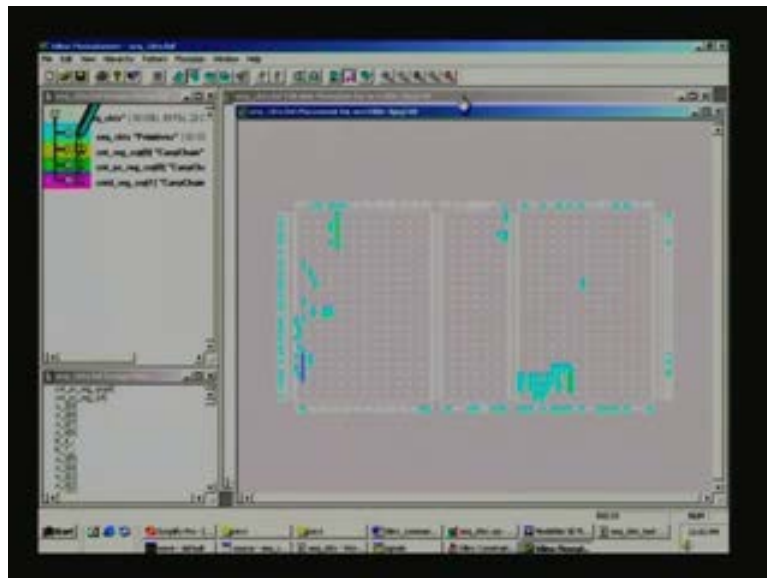
So, next what we will do is, I will go back to this comment here, and what we have here is floorplanner. Click on tools floorplanner to open the floor plan or layout of the design. So, this is the design which you can see as a layout, and use help to change the placement. This is involved, we will not go into depths of this and because it is quit involved, and right at the beginning stage, you do not really require, only towards the end of the project you actually require and help will take you step by step further.

(Refer Slide Time: 32:21)

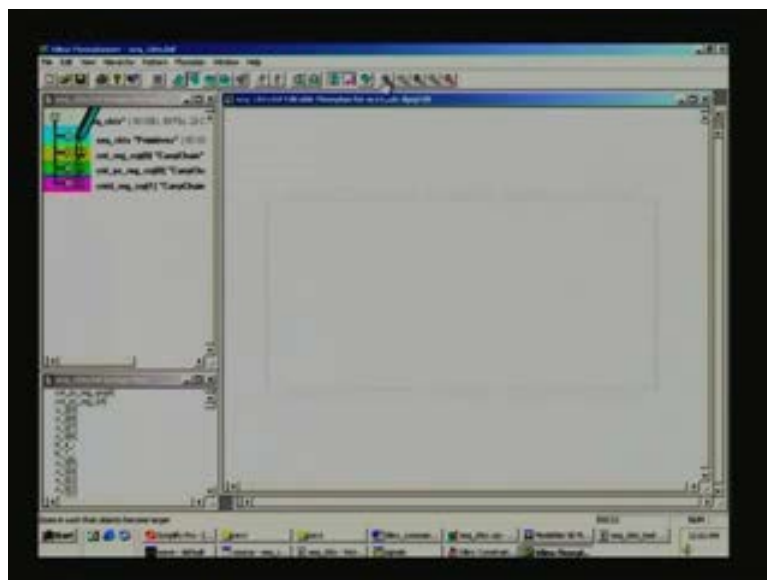


So, we will see only very essentially things of the floorplanner. How it looks without going into too much details. One important notice, if you change any of the constraints or the floor plan, you will have to run the place and route as well as the back annotation again, that is what I mentioned here. So, you have to place and route once again implementation onwards, then go to back annotation and only then your new constraints such as pin or clock will take effect. **We will now see the,** so what it says here is, you have in under tools; you have a floorplanner here; just click on this.

(Refer Slide Time: 33:07)



(Refer Slide Time: 33:50)



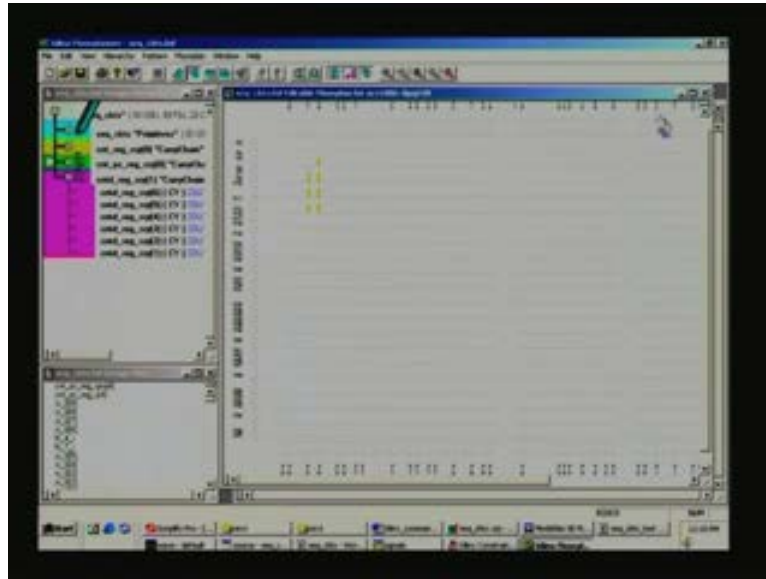


Another window opens called xilinx floorplanner. Here, you can see different windows opened and once again you may find it difficult to read. So, you have what is called placement. So, actually these are all the pins of the FBGA and your design has been map, this is only sequential circuit. You see wherever patch of light here, only those things are your actual circuits, here, and so on.

So, in this particular thing, I think it is a 100000 gate count device we have selected of which your design is almost a negligible area it occupies. Another point is there is one more here, and this is what is called editable floor plan, and in this, you can zoom if you want here by using this plus. You can zoom further. Let us zoom further and see whether we can read something. You can see actual pins here p 220 and so on to 2 1 7 and so on

So these are all the pins here. One more thing is you suppose, you click on this symbol - a gate symbol - here, so, what we have here is, in the editable floor plan that we have seen that this is the layout of your chosen FPGA, say 100000 gates equivalent, and you can see all the pins marked here on all these sides, and for example, pin 3 here 4 5 and so on. It is numbered like this, increasing order in anticlock wise direction, and each of you can see, I wonder whether you can see some paint lines here, there all basically squares. In each square, there will be group of primitive cells, for example, mugs, LUTs and all the primitive cells which we have already seen, that will be there. Now, what we have here - on the left is all your design signals are shown here, and group of signals, you can just click here, for example, i clicked here, and once I click here, I can place that particular part of your design anywhere, you want here.

(Refer Slide Time: 35:40)



Say for example, I can place it here. You can see here that this particular thing is placed here, and like this, you may have to go and pick different elements. Here we do not have much choice here. So, I have just shown one here. If you click back on the same thing again, this highlight, in fact, at flashes, here. If you want you can remove; suppose, you made a mistake, you can remove that and then place it here once again like this.

There appears to some bug, I think you will have to experiment with it, and this is how you, in fact, should come freely here. We will have to investigate why it does not work properly. Any way, you experiment; I leave it as an exercise to you. The main emphasis here is, suppose you are not happy with the frequency of operation that you have already achieved, and inspite of synthesis and you can have extra mileage. Here, by placing this, your different parts of the design at any other strategic point probably very close to the pins and you can also renumber the pins and all these will be added advantage by redoing the floor plan.

Once you do the floor plan, so, next use is to you have to using that pin constraints we have already seen. We had to renumber the pins because, now with the change, this might have been somewhere else here and that would have come here, but the pin connections are still the old connections. So, unless you renumber these pins, it will not have an effect over this.



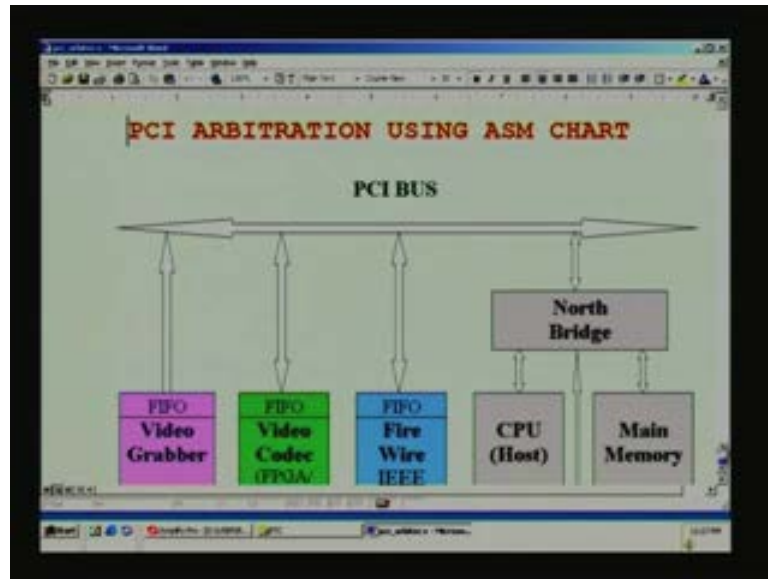
So, this step is to locate all this physically and that may be in a big design; that will be a very large number; that will be a tedious thing. There are, if you look into the help, so, it will list on different topics, and you can find out how they can be handle for a particular it is an index here. **You can go for various** or that is all online documentation as well and in you can use one of the index here find out. So, how to place placement? Something like that.

Then you can say display; you can find out. So, you can, it has help facility for placing placement as well. So you go through that, you will get it, and that might be bugs which you may to have get in touch with xilinx to set it right. In this fashion, you can reorder this placement and then redo the pins by changing the constraints files, and then, once again run starting from the xilinx implementation place and route onwards. Then go on to the back annotation and that will complete the entire cycle.

After you do all this, you are likely to get extra speed, speeding up of system. That is the primary goal here in changing the pin constrains and also changing the floor plan. Another advantage, as I mentioned earlier, there are two teams working on hardware as well as software. One team will be facing on the hardware design right at the inspection along with verlog coding in parallel concurrently. So, that would be possible in this FPGA platform because you can always come to the flag end of this that what we are now in and change the all the pin to suit their hardware engineers design. To suit that particular pin assignment, you can redo this pin.

Constraints you can change. This would complete all the tools. We have seen so far modelsim for simulation and then followed by synthesis for using simplify tool and then followed by place and route using xilinx and also to back annotate, and finally the back annotated file, we took into the modelsim, simulated once again and make sure that the whole thing is intact and we have also seen how to change the pin constrains, clock constrains. This completes in essence all the tools and we will consider other applications shortly.

(Refer Slide Time: 40:41)



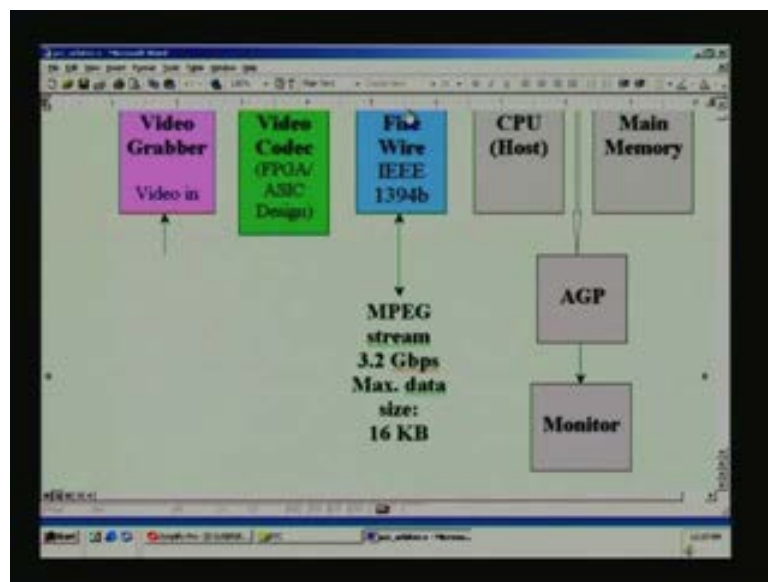
The application we are going to deal with PCI arbitration. So, what PCI is, this called expansion for this is peripheral component interconnect. So, this is a high speed bus data bus, and if you open your Pentium, you will see some three slots available on PCI bus. This is a typical scenario that we can see of in the application that is given here. The purpose of doing this one is, not to go into the details of the actual applications as such but only to deal with explanation of an ASM chart and how to use an ASM chart in your design and then to code in verlog to implement only the arbitration logic.

It is basically arbitrator controller, and let us consider an interesting example. So, this is for the video application. So, right now what we have this camera here, and that is basically video in coming in pal or it can be NTSC sequence. This is hardware to grab this video. It may even be a JPEG input and another input can be a VCR or VCP and there can also be another format such as XGA or SVGA format, straight away entering as a video. No matter what the video is; this video grabber collects this information. This is going to come in a continues stream and this may be even an analog signal which will be converted as a digital signal, finally input here. What we are dealing with this, or the, I mean basically in the digital realms, and this is the video in for this. You can think these modules as simple single card receding on the PCI bus in your Pentium.

So, in fact, there are other cards here, and this, in fact, this you are already familiar with the Pentium. So, this is the host what we are calling CPU and it will have an associated

main memory. In order to connect this host, there will be a bridge between this internal bus and the PCI bus and that is what is called North Bridge. Similarly, there will be a South Bridge, etcetera, and connecting to CDs and other devices, and that is not shown in this because primary intension is not going to the design of all this but only to show you how to design this arbitration logic which may be residing either in the mother board of the pentium or it can be in one of the PCI cards. In fact, you can view each of this as a card. This is the video grabber card which is available from third party, and so, also a serial, parallel to serial interface have called fire wire and this gone by IEEE 1394 standards. There are 1394 then 1394a then 1394b standards and this is called fire wire. The primary role is to convert parallel information in to serial information and send it out here.

(Refer Slide Time: 44:22)



And out comes, it can be a JPEG stream or MPEG stream. Once again in MPEG, you have MPEG 1 MPEG 2 and MPEG 4 and MPEG 7. So, no matter what the application is. So, the whatever logic you have in order to accomplish this such as a compression, so, that will have to ultimately reside in one of the boards and that such board is what we call video codec, and this video codec, codec stands for coder that is encoder as well as a decoder.

So, it has actually basically two sets of circuitry, and this will be finally residing in a FPGA or ASIC. This is the one which you are concerned about as a designer. Suppose,

you want to do MPEG 1, MPEG 2, let us say we do hardware core, write verlog source code and go through all the exercises that we have already covered right up to back annotation of xilinx and use Xilinx, and finally, program a device which will ultimately program the FPGA which is residing this card. If it is ASIC design instead of FPGA, naturally you need to house that ASIC which you have designed. Ultimately, the actual IC will have to power platted on a board here, and this board also has like other boards some first in first out memory here, and if it is video grabber, perhaps you need one or two frame storage and it should be available in this first in first out.

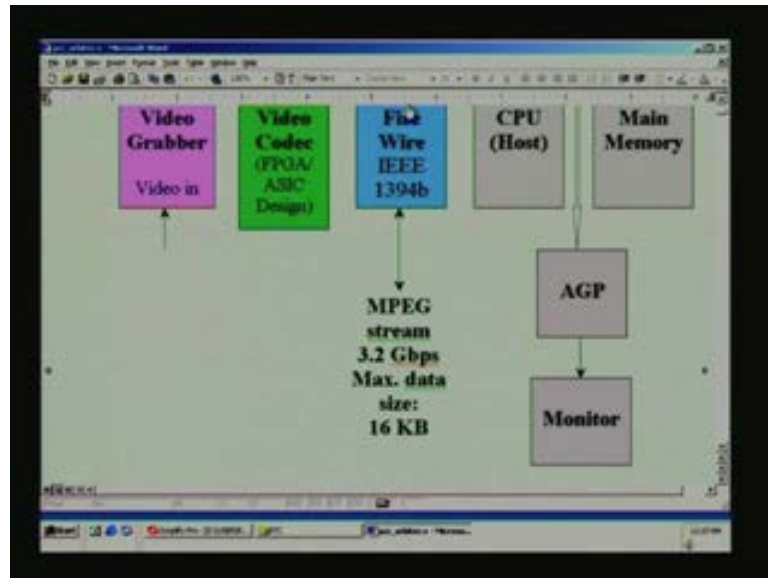
So, whatever image that comes in is basically a rod data, and rod data is already, I mean it is a very huge data, and hence, the need for compression. That compression can be effected only by the hardware, and for which, you will have to write the source code and after writing all that and going through all the exercises that we have already seen before and you can finally have a bit stream, that bit stream can be downloaded on to the FPGA on the card, we call it video codec. The primary role is to process in accordance with MPEG, be it 1 or 2 or whatever or even a JPEG, it can be for still pictures you can use that.

So, here, what we are going to do is we see that these are all independent processors by itself just like your CPU. In fact, all these things will be as complex as CPU thing and they are there for referred to as masters. In fact, you have a PCI bus. It is something like your national highway or simply call data highway. So, on this high way, unfortunately, you can only have one transaction taking place at a time. So, for example, if someone all these things cards basically masters and you have 4 masters here and of all these masters and all of them would like to grab the bus.

So, you will have to have some arbitration done among this. In a control manner only allow each of these devices to access the bus because there are common resources such as AGP and finally a monitor. The monitor that you see right here is basically on, if it is on AGP card installed in the computer, then it is through this. That is what is being communicated; I mean host communicates with the monitor in this fashion. It is all basically through the PCI bus, and of course, there is the bridge here, which is nothing but extension of the PCI bus here. As far as the video application is concerned, how it is? What it is? First you will get your input image here. It will come frame after frame and it will stored in FIFO and then there will be a bus arbitration which will need to design.

That is what we are going to consider actually. As per the bus arbitration, it will decide when to allow this PCI bus access. For example, this wants to access this video codec. In order to download the one frame, it has collected which is already stored here.

(Refer Slide Time: 48:50)



So, when it gets the chance and first it will make a request to the PCI arbitrator which we will see in the next diagram and then it communicates via the PCI bus. It sends the one frame information into this and that is corresponding FIFO here. That will be received and encoder starts working on this particular frame and brings about applies transforms such as DCT DCT stands for Discrete Cosine Transform and then quantization and then followed by VLC and finally that is also motion estimation in MPEG. We can dispense with motion estimation if you want just process as i frames, in which case the circuitry will much less. No matter what you have here. So, we have basically the compression software here, and if it is encoder, it does the compression; and if it is decoder, that other half, it will do the decompression, and for decompression, once again you have to get from outside world. For example, let us say, we have a computer here and we want to connect it to another computer which has precisely all this cards and there also there may be parallel to serial interface card which is called the fire wire.

This can operate at a quite very high speed up to 3.2 Gbps is a very huge thing, and in fact, you can network here. You can basically connect 64 63 nodes here, serial connection, whereas, in order to communicate with just one another computer, what all

you need is just two channels here - one channel to go out for the other computer to receive and whatever similar information the other computer sends it will receive through the other channel. Just with two channels, we can use it for communicating between two, and if you have multiple such computers, you can network them, and in which case, this same fire wire the very same hardware will connect it to 63 basic serial interfaces, and if you want, you can expand right up to 64 kilobytes, it is not a bytes, 65536, such nodes you can connect here.

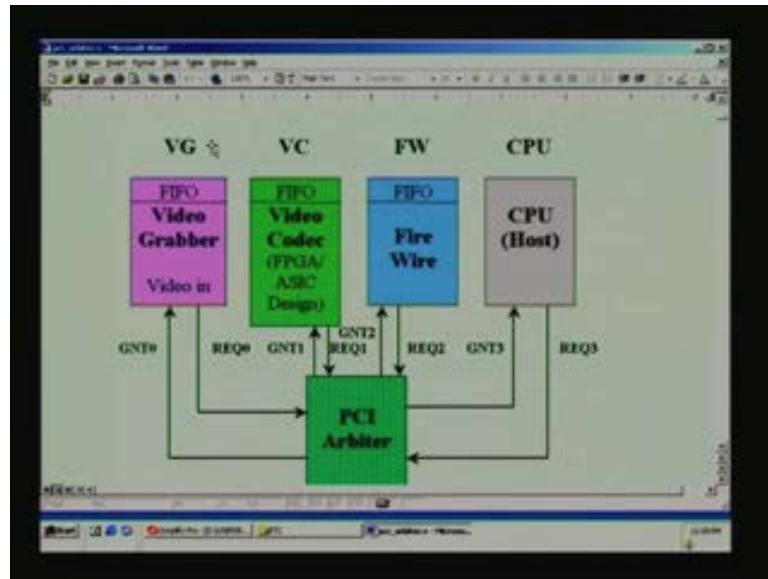
That means you say network number of computers and you can map the whole thing for video conferencing and video phone and apart from receiving motion picture. That is precisely the application is for, and what CPU does is, in order to configure it from one standard to another, for example, JPEG to MPEG, MPEG 1 to MPEG 2, and so on, because all this you can have right on the same computer that we are working on, say the Pentium base computer.

So, for that, you need the CPU and associated main memory and what all you have here, if you want to display it, you need a monitor, because any picture that you receive from the serial phase from the other computer, so, you will have to what go through disk here and do the inverse of compression. So, this is the compressed still that you will be getting here, that is pumped into this, and decoder will do the inverse operation and get the image which is called reconstructed image, which is precisely the same as the origin image the other computer as sent, same is the case with this here.

So, you can have a bilateral traffic here. Suppose, you can send one motion picture using this here and on to this and the other computer can receive and in concurrently the other computer can send some other picture. In parallel, you can process and what you have process is compressed still that you get here. This can be decompressed by this hardware here, which is our, which is in our realms for the design.

Decompressed image you can through the PCI bus, you can download to, I mean it is not downloading, straight away display it on the monitor through an advance graphics port here. I hope this clarifies the total application and out of which what we are going to consider is only bus arbitration here.

(Refer Slide Time: 53:11)



So, this is abbreviated video grabber, video codec, and then fire wire CPU. There are four masters basically and they have four request signals given to the PCI arbiter which is your current design which we are going to see. This, it will depend upon the priority. Naturally, it calls for the highest priority because this is the video image coming here and video sequence cannot be missed, not even a single sequence frame can be missed.

So, you have to give top priority to this. Then next high priority will have to be given to be VC that is a codec here because it brings about compression as well as decompression and it is all time consuming affair. So, and after that, compress image will be sent over this and this will naturally have lower priority. The lowest priority is for the host here. The bus grant is once again issued by the PCI arbiter has grant here. There are four request and corresponding four bus grant here.