Digital VLSI System Design

Prof Dr. S. Ramachandran

Department of Electrical Engineering

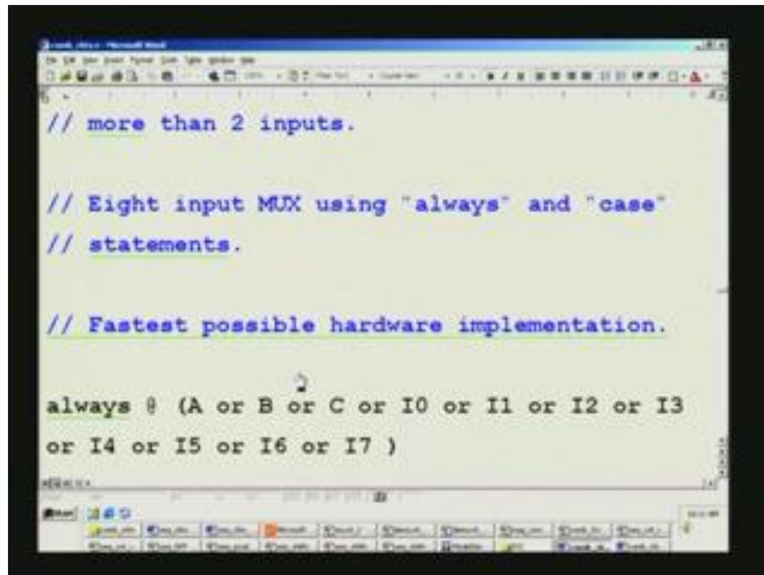
Indian Institute of Technology, Madras

Lecture - 25

Simulation of Combinational and Sequential Circuits

In the last demonstration of simulator, we saw eight input MUX and we will continue with the same.

(Refer Slide Time: 02:08)



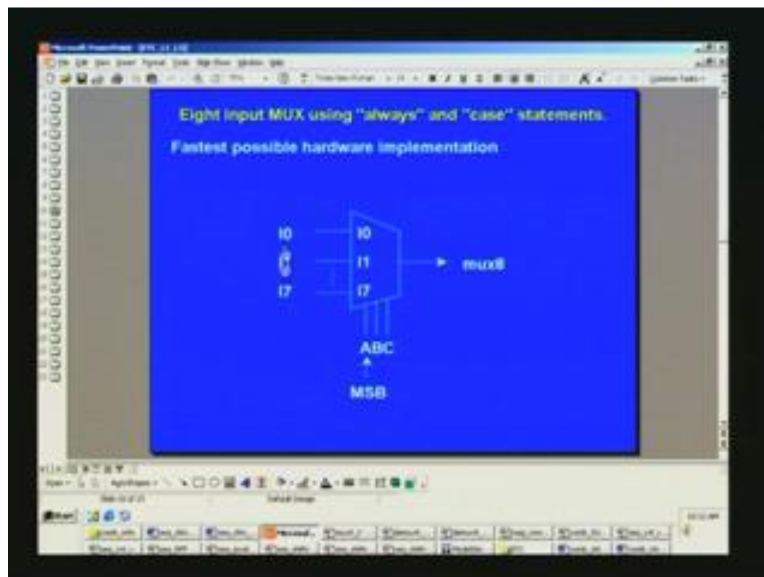
We will quickly have a glimpse of what we had done in MUX and continue with DEMUX and other circuits. First, complete all the combinational circuits and then go over to sequential circuits. To start with, this is the code that we have written already for eight input MUX (Refer Slide Time: 02:29) using an always statement. I am not going into the details; it is enough to point out that I0 through I7 are all assigned to MUX depending upon the A, B, C conditions of the input.

(Refer Slide Time: 02:44)

```
case ({A, B, C})

   3'b000:  mux8 = I0 ;

// Read the input addressed by ABC.

   3'b001:  mux8 = I1 ;
   3'b010:  mux8 = I2 ;
   3'b011:  mux8 = I3 ;
```

This has been adequately explained couple of times earlier so I will not go into the details of that.

(Refer Slide Time: 02:51)



Pictorially, you can see this in the PowerPoint and this is what it is and we saw the waveform for this MUX.

(Refer Slide Time: 03:02)

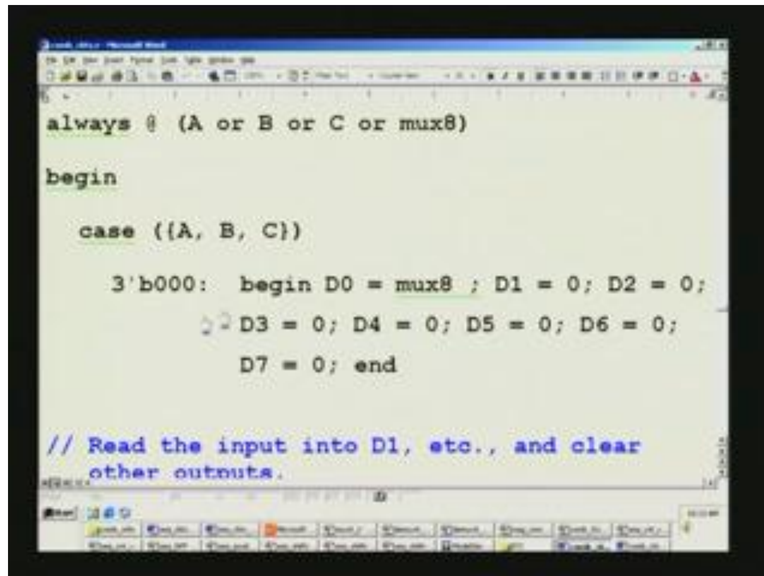
There are two waveforms as such. The last one gives the actual outputs. We have seen I0 through I7 as a pulse applied at varying points of time at every twenty nanoseconds if I remember right.

(Refer Slide Time: 03:18)

Finally, we had this waveform, which is the MUX8 and corresponding inputs, this is I0, I1 and since A, B, C keeps on varying in the binary fashion that you have already seen.

You have actually 3, 6 here and 7 here and 8th one is here because I have forgotten to deactivate of course, that is immaterial rather. You have totally eight inputs corresponding to the I0 through I7. Now, we will move on to the DEMUX and we will first have a look at the code.

(Refer Slide Time: 04:01)



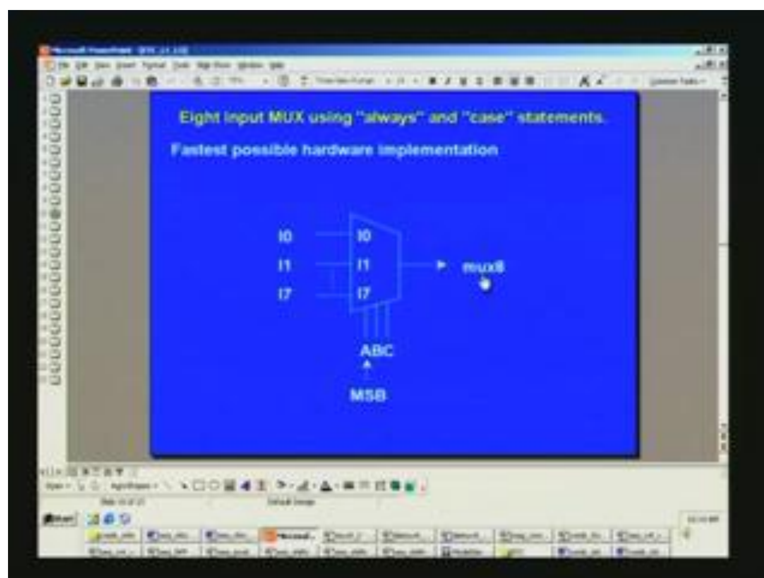
This is the DEMUX. This is an exact inverse of the MUX and we feed the MUX8 which has the output for eight input MUX into this DEMUX. Once again you see it is based upon the select pins A, B, C. As per the A, B, C value you have D0 through D7. We have also mentioned that we should not forget to reset all the other outputs and only that output under question needs to be input from the MUX8. What we are doing is the inverse of the MUX8. What we put in as input for MUX8 namely I0 through I7, we will get the corresponding D0 through D7 as the output. This will be the same as the I0 through I7 that we have. We have also seen default condition towards the end.

(Refer Slide Time: 05:00)

```
3'b111:  begin D7 = mux8 ; D0 = 0; D1 = 0;
         D2 = 0; D3 = 0; D4 = 0; D5 = 0;
         D6 = 0; end

default: begin D0 = 0;   D1 = 0; D2 = 0;
         D3 = 0; D4 = 0; D5 = 0; D6 = 0;
         D7 = 0; end

endcase
```

It resets all the outputs because in the course of the circuit working you may encounter x in any of the bit position or even z, which is indicative of a tri-state condition. So you take it to a safe state of resetting all the outputs under such extreme conditions when such invalid signals are applied.

(Refer Slide Time: 05:29)

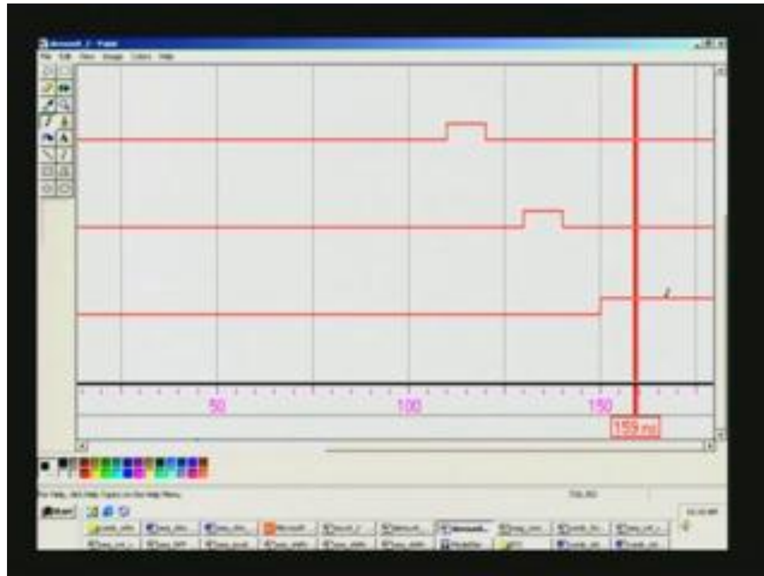


The same DEMUX in the power point that you have seen is pictorially depicted. This is MUX8 output, connected. Note that A, B, C is the same select pin for the DEMUX here. A is MSB and MUX8 is input here and outcomes D0 through D7 and D0 is the very same as I0 and so on right up to I7. The waveform for this one has been obtained from modelsim because it was not visible when we did real time. I have made it into paint shop and that magnification is far better, resolution is better on TV monitor. That is why it has been done. There are actually two files, one is this (Refer Slide Time: 06:18).

This is the DEMUX; A, B, C are the corresponding inputs that go in the binary fashion 0 0 0 0 0 1 and so on and till the end. This MUX8 is what you have seen already corresponding to I0 here; (Refer Slide Time: 06:36) this is MUX8 I0, I1 and so on right up to I7, which was flat on top here. What we expect here is D0 through D7 as the output for the DEMUX. As we said before, I0 was just this waveform and thereafter it is flat 0. Naturally, if it is working correctly we should get the same waveform; it should recover your inputs.

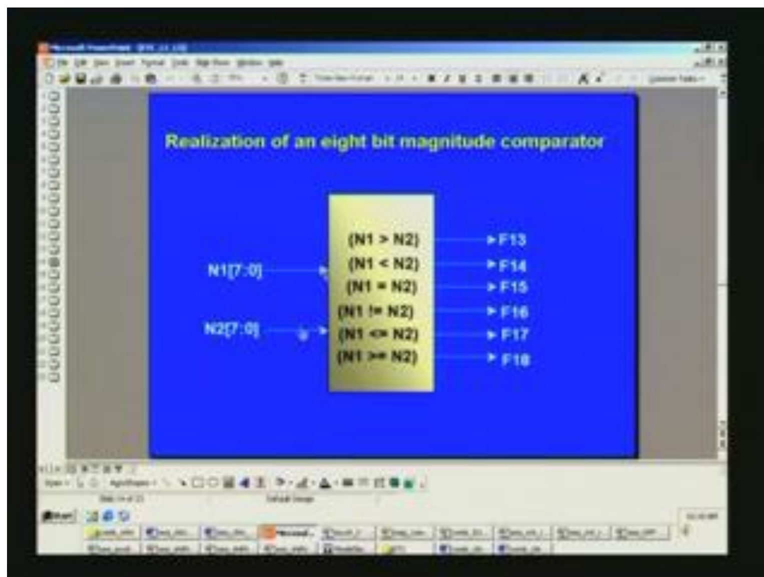
Let us see D0 first. That is precisely what you have here so you can see this D0 then D1 here and you can see it is a flat there. After 0 just single pulse is got. This way it will be easy for you to crosscheck so you can plough one output into another module and then check it out. It will be very handy that way and probably you can adopt such strategies while designing any circuit. Once again we have time based here; it is marked here at fifty. Before copying from the modelsim into the paint it was expanded. So that is why you see a different scale from 20 nanoseconds earlier which we have seen. Then we move on to other outputs in this file. We had seen D0 and D1 there and D first at this position and this position. In the third position, you have D2 here and the time based is also same fifty. D2, D3, D4, D5 you can see all of them go by one clock period. In fact, being combinational, there is no clock period; it happens in every 20 nanoseconds.

(Refer Slide Time: 08:39)



The final output remains at 1 as such. So this completes DEMUX as such and we will move on to the next one called the magnitude comparator.

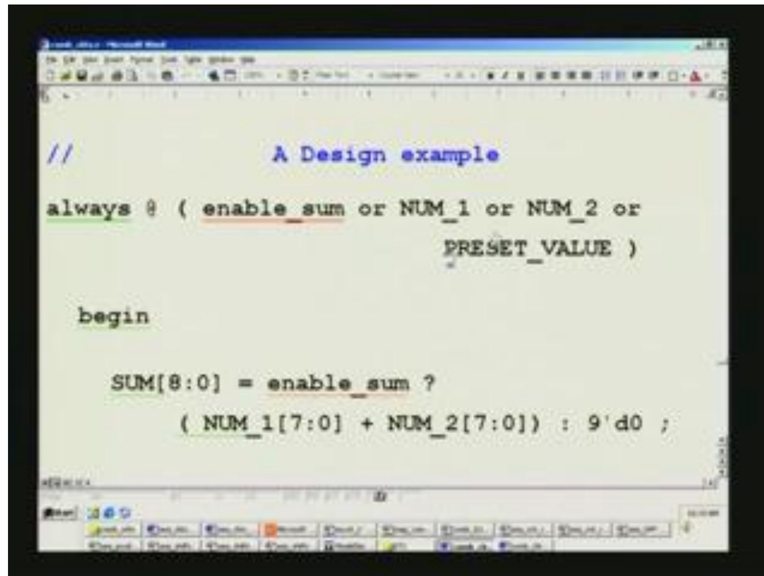
(Refer Slide Time: 08:50)



We have seen already full adders. We will move on to the magnitude comparator. I hope you have not seen this earlier in simulation. You have two numbers, N1 and N2 each eight bits in precision. If N1 is greater than N2, you activate F13 and so on for less, you

activate 14 equal to this and so on right up to this F18. This symbol is greater or equal to the nomenclature that you adapt less or equal to and not equal to. See the waveform for this.

(Refer Slide Time: 09:28)



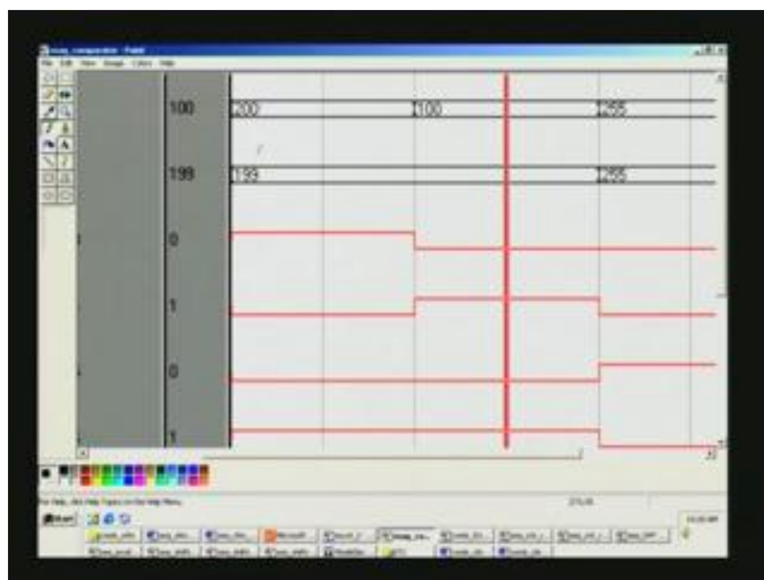
Before that, if you want you can have a look at the code also that we have already written. So that is this here. It is in the always block. Is this the one which we are doing? It is not this.

(Refer Slide Time: 09:49)

```
or ( carryo, o1, a3) ; //Realize carry out.

// Realization of a magnitude comparator

always @ ( N1 or N2 )
  begin
    F13 = (N1 > N2);
// Set output if N1 is greater than N2.
```

This one is the magnitude comparator. Depending upon N1 or N2, it is the always block we had used. So it is a simple assignment as you see depicted in the picture on the slide. That is what you have for up to F18, which is greater, just as you have seen on the slide. We now see the waveform for this.

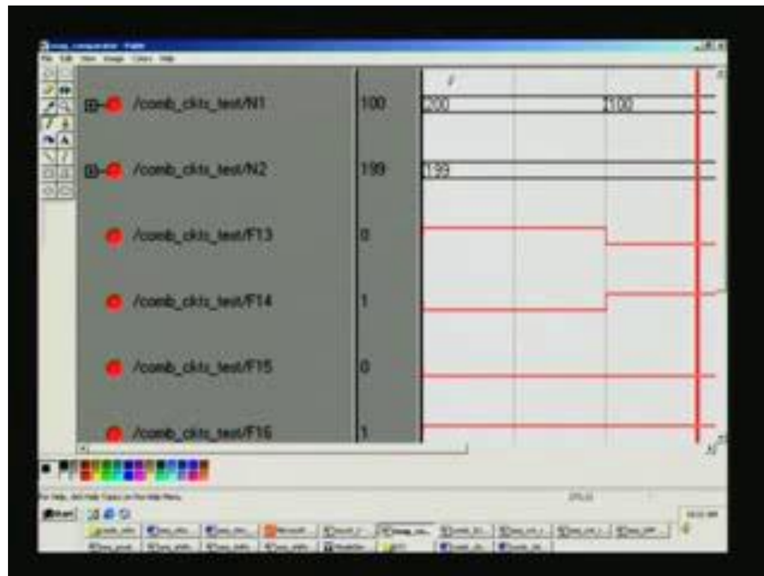
(Refer Slide Time: 10:11)



This is the waveform. What is listed here is N1. This is combinational circuit test bench and under that these are all the signals N1 and N2. Just now you have seen F13 through F18. These are all the outputs here; first only two numbers are there. You do not have to remember. You notice a difference. All along we have been using only single bit here, indicating a binary signal. We are dealing with multi-bit signal because this N1 and N2 are actually seven through 0. That is 8 bits here, and therefore you see a bigger number. We will also see the modelsim, how we set these menus. We are doing a magnitude comparator. How it should function? Let us see. 200 is N1 and 199 is N2 and if you compare the two, naturally you have 200 greater than 199. Therefore this signal must be on F13. That is what we have depicted here in the pictorial.

N1 greater than N2 F13 (Refer Slide Time: 11:33) must be on; then less and then equal to we will just swap between them in these two, then we will get a clear idea.

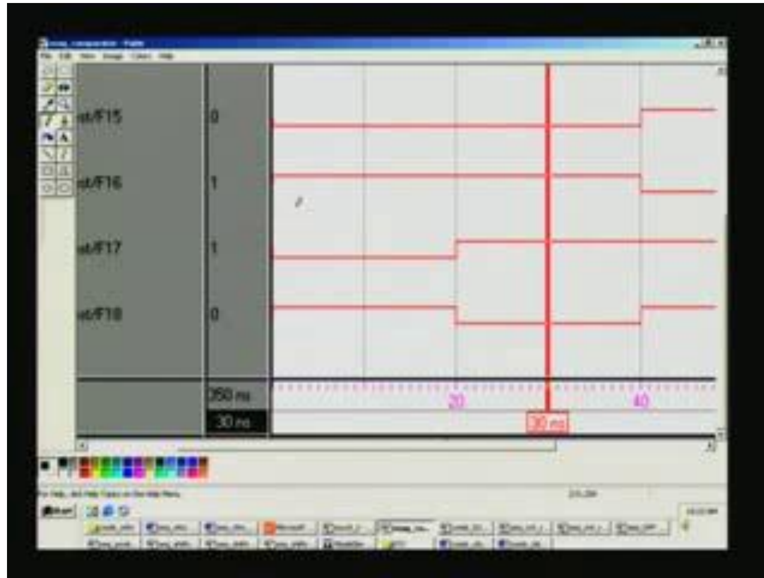
(Refer Slide Time: 11:46)



So you can see here, if it is 200 greater than 199, this bit alone goes high. This is the output here and thereafter it remains low because nowhere else that condition is satisfied. It is less here and it is equal to here. We can see that if it is less this must be activated. You see that this remains at 0; F14 is less N1 less than N2. Here it is greater; therefore, it has got to be 0; it goes high here only during this transaction condition. 100 is less than

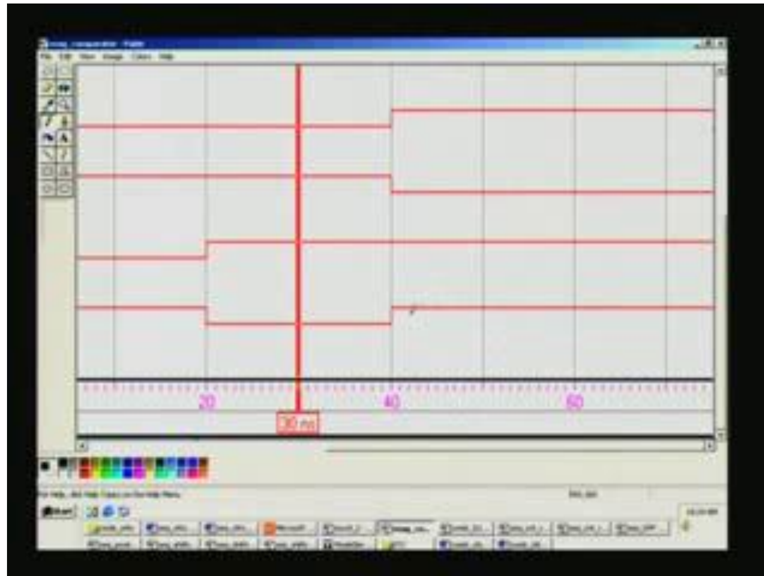
199, N1 is less than N2. Therefore it goes high and remains low all through. Similarly F15 is equal to N1. It is the condition for N1 equal to N2, this does not happen here. Therefore, F15 remains 0. It goes high only when it is equal to. You can see here 255 equal to 255. It goes high here till it is 0 here.

(Refer Slide Time: 13:53)



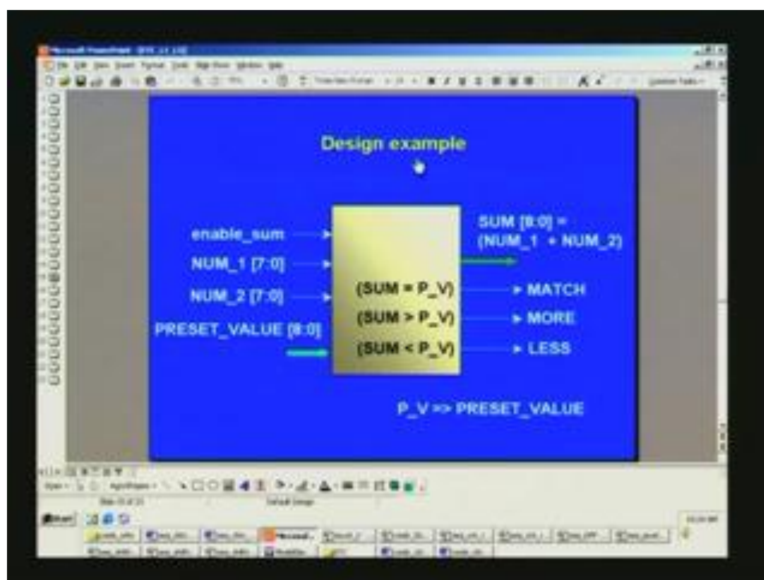
Let us see what the other one is, F16. Three of the outputs we will consider. We will go back to the slide and see what they are (Refer Slide Time: 13:25). Say F16 is what you are asking and N1 not equal to N2; the other one is less or equal to, greater or equal to just remember this so that we do not have to come, not equal to less or equal to or greater or equal to. In this case, this is not equal to, so not equal. This was equal to condition. So not equal to will be the exact reverse. We do not have to look at even the inputs. It is high here for F15 being 0 and it goes high thereafter; here it goes low here. That proves that, not equal to, is also working. Then here it is less or equal to. If it is less than or equal to the first one we started with a greater there so it goes low, and this goes high because this is greater or equal to. We can club together and deal with it here. It was N1 less than N2.

(Refer Slide Time: 14:44)



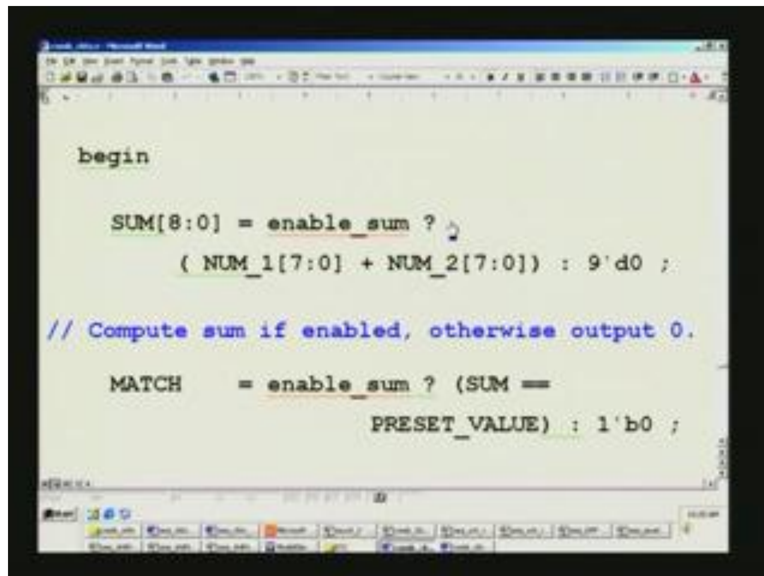
This goes high and this goes low, and for equal to, this is the case here. What are we dealing with? We are dealing with less or equal to or greater or equal to, so equal to is not satisfied here. Naturally it is 1 1 in either cases. We will close this window and see what more we have.

(Refer Slide Time: 15:11)



I think this is the last design example in the combinational circuits. You have here, just to recapitulates what we have done earlier. Briefly, we had two numbers, number 1 and number 2, each being eight bits here. These are all the inputs from the user. If enable sum is high, only then this computation will be done. This sum will be 0 if enable sum is 0 as a low or inactive. The actual sum will appear here only if it is 1. So is the case for other match outputs more or less which simply means the sum equal to preset value, P_V is abbreviated here. Note that when you add two numbers, it will result in one more bit extra. That is why it is eight through 0 here. We will see this is a sum example. So let us see once again the actual code for this.

(Refer Slide Time: 16:15)



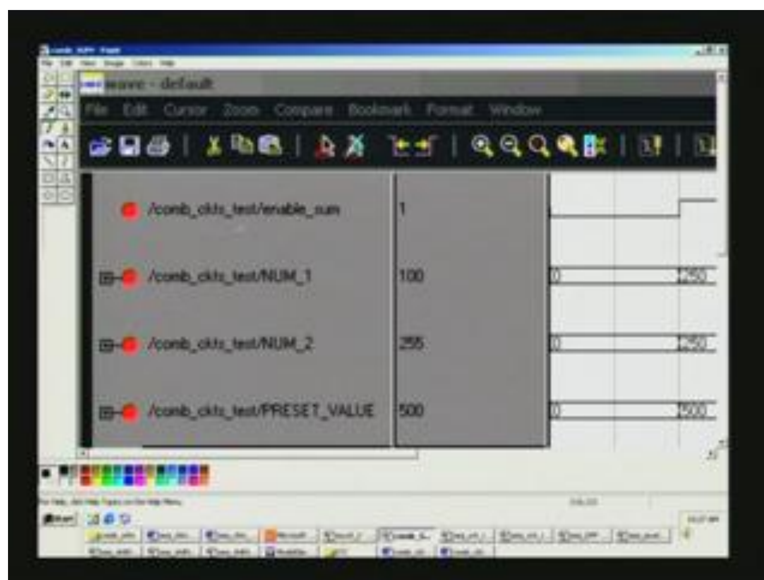
We have used a MUX here based on enable sum being high. This question mark is for the MUX here. You compute add signs here and I think I showed this on the board earlier. I might have left out some bracket; it has to be in place, normally. You can remove the brackets, run through the compiler and find out whether they are absolutely required. It is always preferable to have brackets where possible as we have mentioned before. If this is satisfied, the sum is allocated to this. Otherwise 0 is output into the sum. That is what it means. If enable sum is 0, 0 will be output. So is the case for more, less and so on.

(Refer Slide Time: 17:10)

```
    MORE      = enable_sum ? (SUM >
                             PRESET_VALUE) : 1'b0;

// Set output if SUM is greater than
// PRESET_VALUE, only if enabled.

    LESS      = enable_sum ? (SUM <
                            PRESET_VALUE) : 1'b0 ;

// Set output if SUM is less than
```

Only thing is those conditions will have to be satisfied for match more matches meaning the same preset value equal to sum then more and also less. We will see the waveform for this corresponding thing. We have already seen the slide on this pictorially and we will see the waveform for this is the sum.

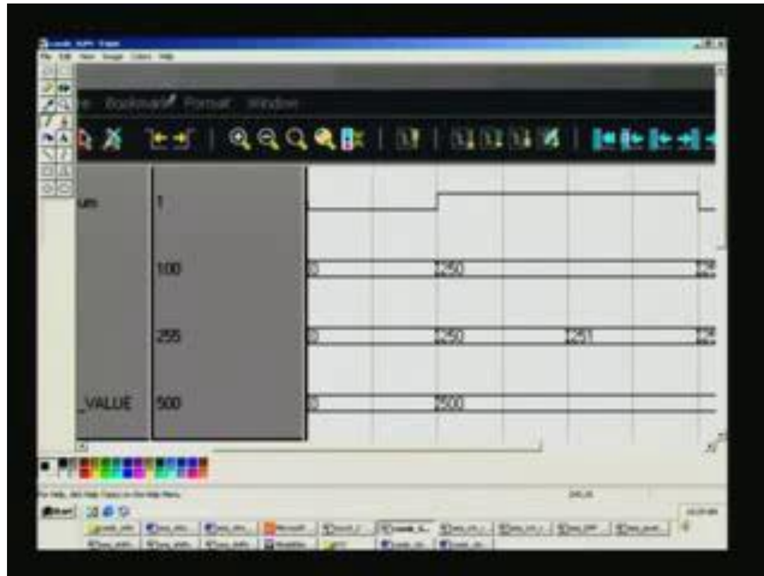
(Refer Slide Time: 17:35)



We have an enabled sum here and number 1 and number 2 are there. We have also a preset value here and you can see the entire waveform window. You can see in more clarity here. In the real time simulation, we could not see all this clearly. This is file edit cursor zone and there are different symbols here. This is the usual cut; it is the same as any windows command. This floppy is for Save and Open and Printer and so on and so forth. This is the cursor I have mentioned earlier. If you click on this cursor, dotted line cursor will appear and you can remove that by clicking on this.

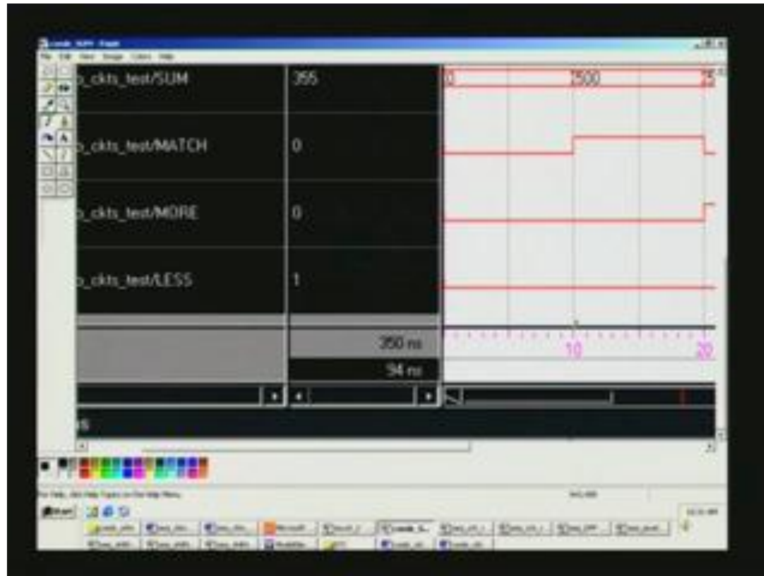
We have also seen how to use transitions of waveforms. Whatever you had marked or highlighted by merely pressing here and along with shift if you press here, all this will be highlighted. Whichever signal you want to highlight, you can use that one. In addition to this additional signal you want to highlight, you can use a control and press at the desired point. So once again if you press at the already highlighted thing control and on the signal that signal will be deselected. It is very easy. It is the same as any window platform. It uses very similar things and you have a cut we have seen. These are all zooms here. You can zoom in here. I think factor of 2 will be there in zoom in and zoom in, that is, to expand what you see on the screen. You can zoom out here. Whatever simulation you have done, the entire range is automatically fitted in the one whole screen as such which we will find out. You also have some help in the main menu. This not the main menu, it is only a wave menu.

(Refer Slide Time: 20:06)



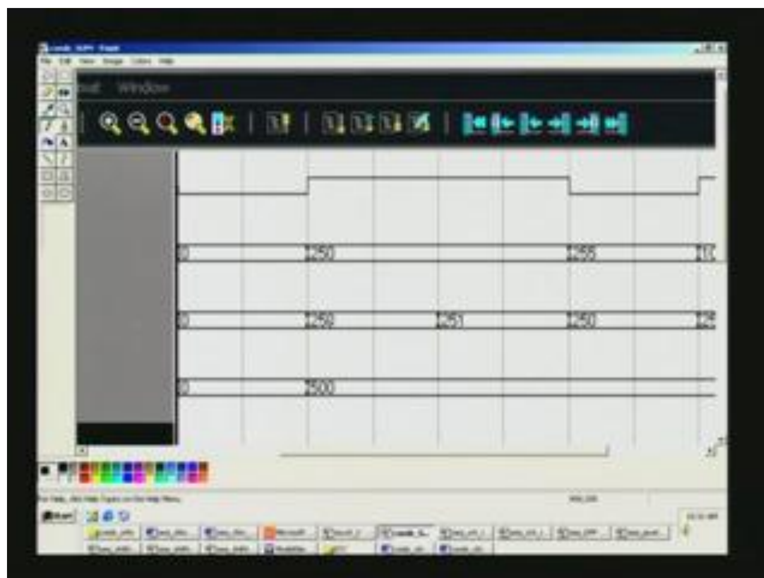
In the main menu you have more and you have also a help facility available. Documentation is also available which you can download using the Internet. There are so many other symbols; you can just get familiar by clicking one menu after another. Then it will be clear to you. (Refer Slide Time: 20:31) I hope you have these platforms at your premises so that you can experiment on this. Coming back to this application, when enable is low no activity takes place. That is why you see here for number 1 equal to number 2 preset all of them. This is through the test bench. I hope I do not have to show the test bench, if you require I will show it. For this condition, number 1, number 2 and preset value are all 0. What do you expect the output would be? It should be sum or not? Unfortunately the correct value is shown here. But actually, it does not really compute here. The reason is this enable sum is not active during this.

(Refer Slide Time: 21:21)



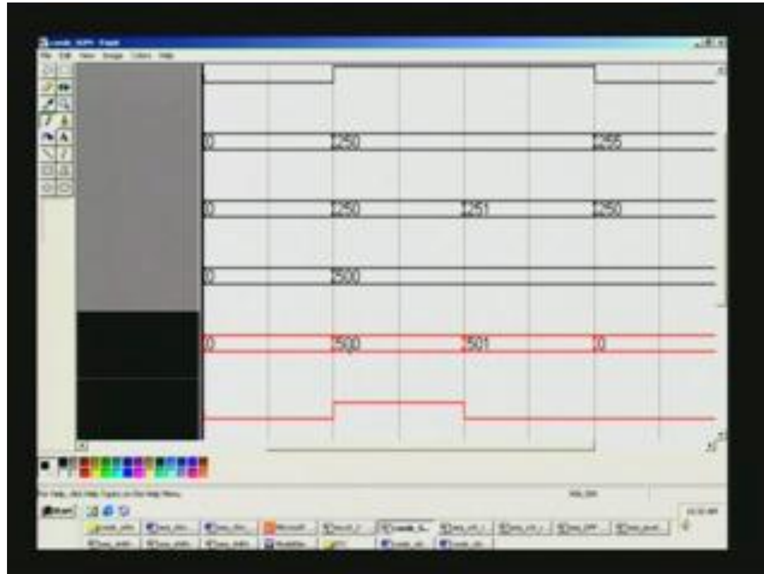
You can verify this by examining other signals. For example, if you add you will get 0 here and when you compare with preset value, it is equal to. So you should have a match output but here the match output is not there. This proves that this enable sum is working. That means enable sum is not active here for this duration. This timing here is 10 nanoseconds. If you look into the test bench, I would have made appropriately using hash 10 and so on. Going to the next one, let us take these values.

(Refer Slide Time: 21:54)



Now you see that this is enable sum. The first row becomes enabled at this point. So 250 and 250 are the two numbers. When you add these two you get 500. That is the sum. It is happening correctly here. Now let us see what else is required.

(Refer Slide Time: 22:17)

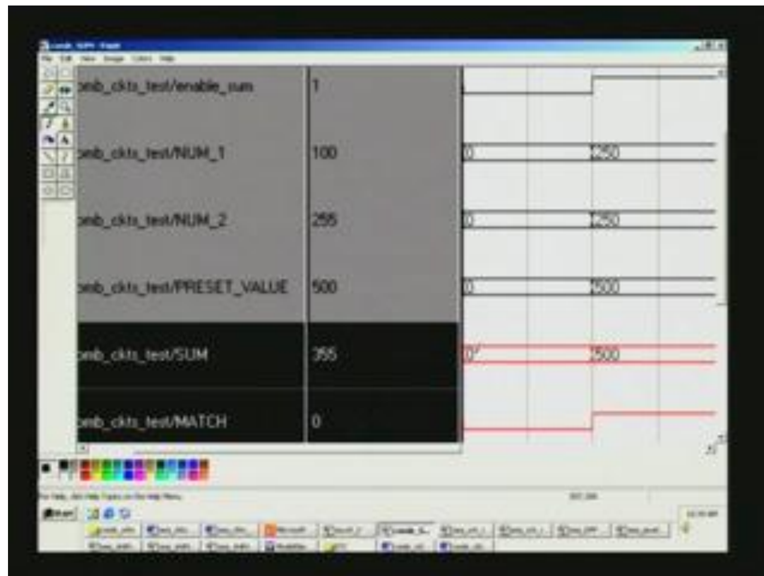


What is this and what are these, let us have a look. That happens to be the preset value, not the sum. I pointed to the wrong thing. That is a preset value 500. We had two numbers here. When you add, it amounts to 500, that is, this one. This is the computed value. Now you can compare 500 with the preset value, this is the preset value. Equality is seen here, so match signal must be on. Is this the match signal? Yes, it is. So you see here it was 0 all along on either directions and it goes high only during this equality being fulfilled. The next condition is 250 and 251 and this amount is to be 501. We are comparing it with 500 so it is less now. What should go high is this less corresponding thing.

Something is wrong. Sum is greater. So once again I made same confusion. Sum is not first it is a preset value. Sum is later; it is greater so more will be activated and last one is the less. We will examine that once again just to make sure that it really functions. So 250, we were here, and then here this is 250 and 251. It is 501. This is sum compared with this; sum is greater; it activates more signals, more is this one, and then the less

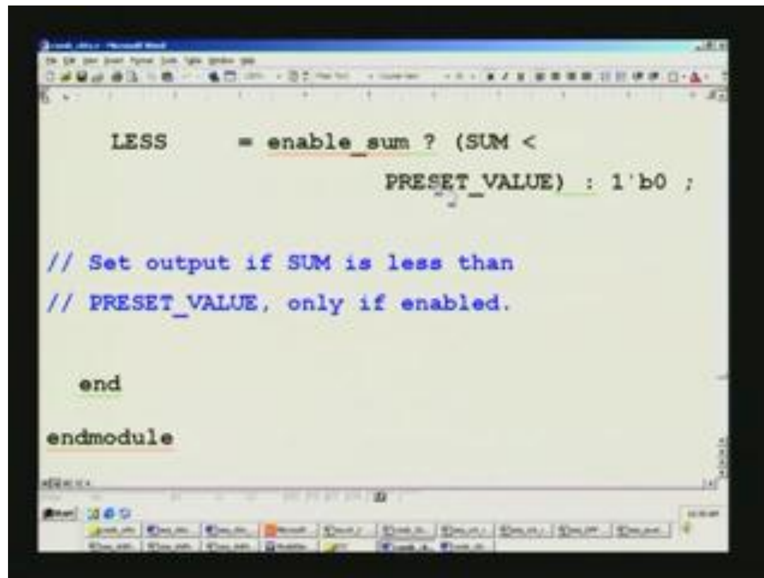
signal is activated at this point of time. Once again, before that one, I deliberately made enable sum once again low just to show there it is still working. At any point of time you can make this. When it is made low, which once again it ignores and you see there is no sum here and preset value is input condition; it continues wherever it was left. So here it is 505. That is absent here; the reason being enabled sum is inactive here. Once again it is made active here. All this, mind you, are stimulus applied through the test bench.

(Refer Slide Time: 25:06)



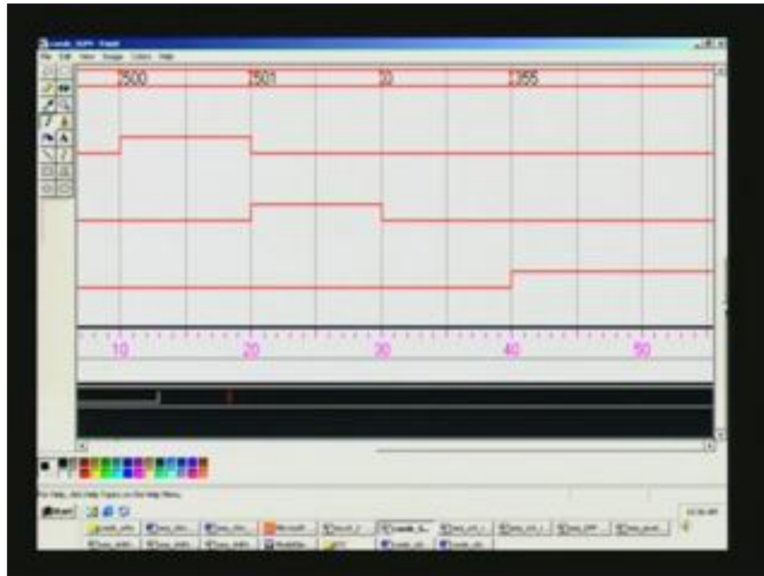
You had used a MUX there. So instead of putting 0 there you can say z. A tri-state will appear there, or you want x there, you can get that, or any other data that you want to output at that point of time and that thing is satisfied. You can always put there in the second input that we have seen here.

(Refer Slide Time: 26:14)



Because it was made 0 here in this so 0 is appearing if enable sum is low, promptly comes this 0 over to less and so is the case for sum. If you want a tri-state, remove this and just put z there, so 1 stroke b, z there. So tri-state will appear there, in case that is what you need. If x, you can put x and so on. Any other data there is no (26:37) of course, it should match. Being one bit number you had to give one bit not higher bit; otherwise, compiler will compile. I think we have completed this combinational circuit, probably less, we did not see the last one.

(Refer Slide Time: 27:14)



So less goes high and no other signals are activated; only among more match only one is activated at a time because only one can be satisfied. So you can see the last value as 100 and 255. When you add you get 355. This is less than the preset value here. Therefore, less goes high at that point of time, see at 40 nanoseconds. You can have a look at the test bench there.

(Refer Slide Time: 27:47)

```
        B = 1'b1 ;
        C = 1'b0 ;
// At time 40 ns, let the inputs be 010.
        I2= 0; I1 = 0 ;

        N1= 255 ;
        N2= 255 ;

        NUM_1       = 100 ;
```

It is like labyrinth, we will have to trace completely. I think I will leave that as an exercise for you. If you go through that you will see precisely the same. I have gone through and I am convinced so you can also convince on looking yourself. This completes the combinational circuits. We will go on to sequential circuits. Before that we will have a look at the modelsim once again. When we touch the shift register, you will be seeing that precisely. Let us open the project.

Let us say the one we want is in combinational circuit. If you want you can operate right from this project or you can open a new project called sequential circuits. Do you have a sequential circuit? Let us not worry about that. We will just see where we are. We will operate right in this. There is no harm in operating. So we should click on this library and probably next time I will show a bloated up figure of this so that you can view each and everything. Right now it will be very difficult for you to know. You may see very little activity going on. It is because the font size is much smaller, this being set in 1024 by 764 resolutions. You have a parallel resolution much less and also because you cannot distinguish fine lines on the TV. That is the reason why you are unable to see this live.

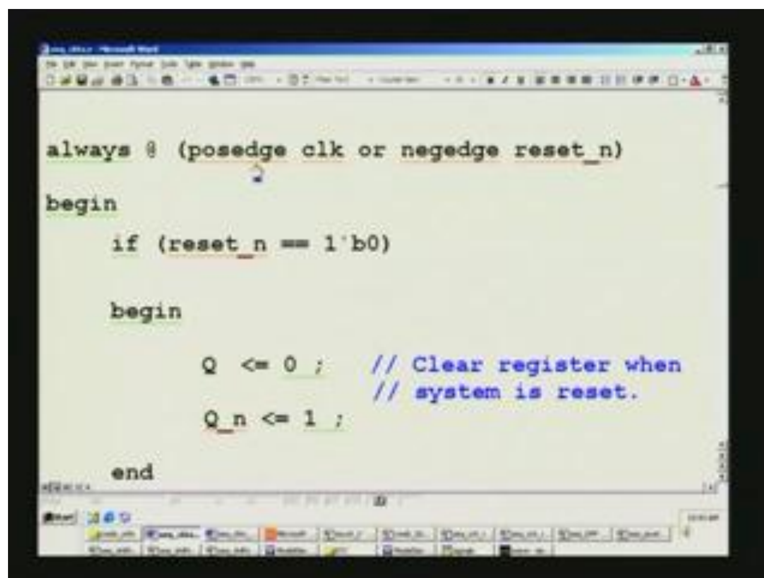
We will have a look at the sequential circuits here. What we will need to do is we had to compile the sequential. It is listed here in this. So here I can, as I mentioned, the circuit design is sequential circuits, and it is there included in this file. When we double click on this, it will do the compilation of the both right here. So once compilation is done and no errors are encountered, you can safely go to the load design. In this you invoke this double click on this, and it has come as work sequential circuits underscore test, you load this here. While loading also you should not get any errors. There may be some warnings; it has done the job.

Now, the next step is to use view, then signals and you have the signal window opened and you can see some of the inputs here, partially you can see as you take it; all the I/Os listed here. You can get it as a waveform; go to view then on these signals in design, and it opens the waveform window. Let us run this. For run, this is the command; run all is there. In fact, I showed in that waveform also similar such thing was there. Here we saw that it is only zoom area. The third one I was saying earlier, I pointed to zoom in then

zoom out. Then, third one is the area you can zoom if you want; you can zoom out the entire simulation to that extent. We will see that all once again. Let us run these codes now. You see different waveforms. I wonder you may not be able to see on the TV monitor. It also had stopped at a particular point of your test bench and this is the source file automatically opened here. This also will be very difficult for you to see. I will just close this and I will just indicate.

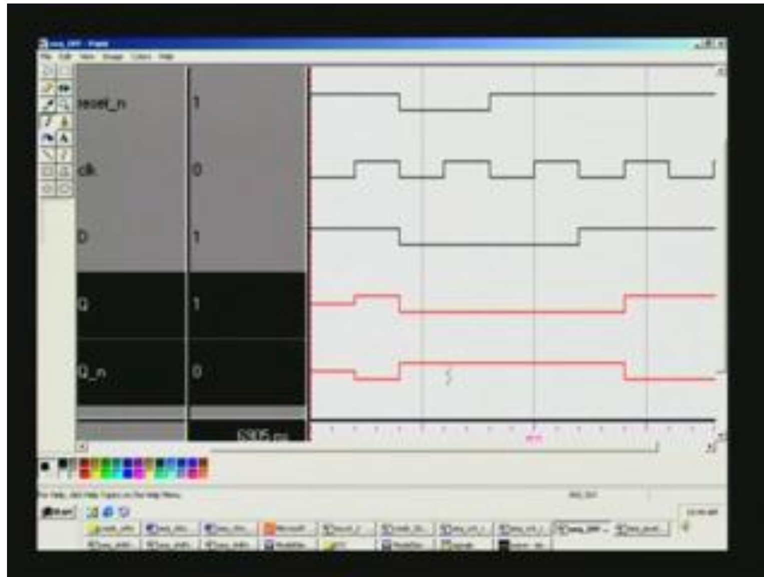
You take note of only the words that I mention and we will go on to the waveform as usual using the paint later on. Perhaps you may be able to see in a faint manner here. It is extremely faint. Only the clock is seen here and other things are almost practically unreadable. You have here all I/Os listed. There is one vertical bar here. You can just hold on to that and drag here so that you can see the entire thing without any obstruction. Similarly, you can take this also and then move it here so that you can see this. This is what you were asking 0 1 0 1 etc., it shows. This will be export to paint and we will be seeing that. We will see function-by-function; otherwise you will get drowned in this maze of signals here. These signals I can drag to any amount. So these are all sequential circuits. This will be extremely painful for you to read unless you have this simulator on your computer and working upend.

(Refer Slide Time: 34:06)



Now, let us go on to a particular thing. We started with sequential circuits. This is what we have already seen. We just had a simple D flip flop Q and Q underscore n, being set to 0 to start with. The course of normal working when reset is not applied and we take D input or its inverse and assign to Q and Q underscore n, respectively. This is simple D flip flop. We will see the waveform for that, it is here.

(Refer Slide Time: 34:52)

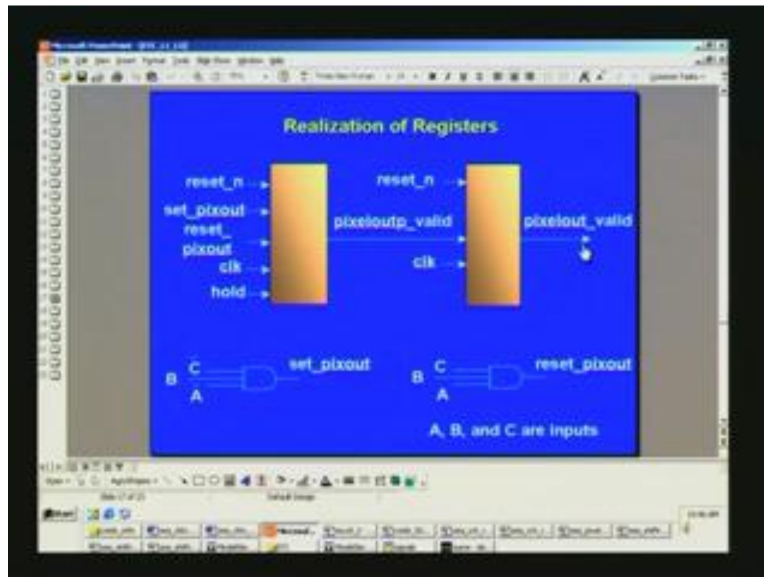


Here, we have reset in clock, d input and q. So it is quite simple for you to find out. Whenever you reset the Q and Q bar, the moment you reset Q is made clear. It was 1 here so it is cleared now and Q underscore n is made high so Q Q bar, basically. Subsequent to the reset being on after a small duration, this is the reset power, power on reset we can consider here. After that it has got to be high in the normal working state. Only beyond this the circuit will start working.

Now what we had to look for is the clock transition. We have written a positive clock transition. Whatever is the D input Q input will get the value. Here it was already cleared; reset is applied right up to this point and beyond that the very first positive edge clock is this one. So here at this point of time, D is 0. This 0 goes through and gets registered in Q. Unfortunately, this happens to be the very same state, initialize condition. You do not see the difference here but you will see when D is 1. For example, here being a

synchronous circuit, it will always happen only at the clock transition time. The next transition happens here and only then the Q goes high and its inverse goes low. This is quite simple. Once again a time base is depicted here. We have to avoid all the possibilities. D D flip flop is the most simplest of flip flops. We will close this and go on to the next here.

(Refer Slide Time: 36:42)



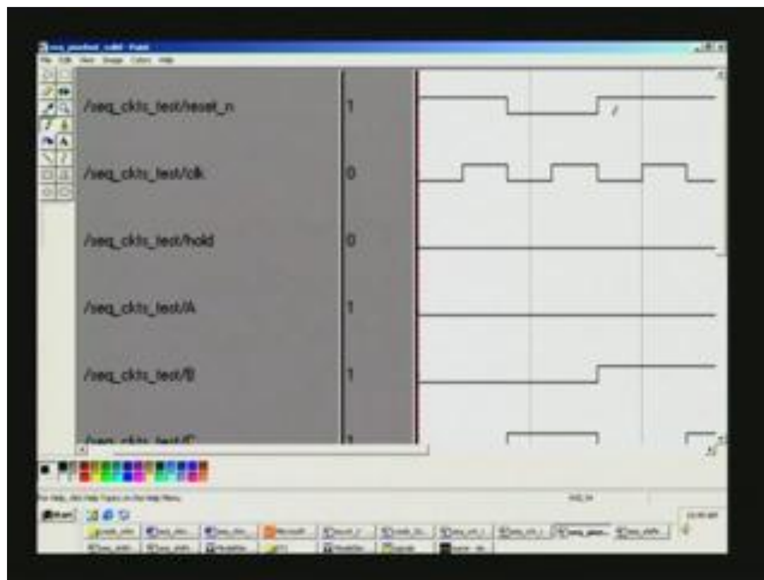
The second one is we had a delayed register and as I mentioned, this is for one of my projects I have handled. It is called video scaling. In this, video scaling is to take a motion picture or even a still picture of low resolution, let us say 800 by 600 pixels, and transforms it on real time at 30 frames per second into a high resolution picture, say 1024 by 768. For that, when a pixel will be out at every clock pulse, the clock rate may be 50 megahertz or so. That means you can process 1024 by 768 pixels color pictures, motion pictures at 30 frames per second. For that you have also to have another signal indicating when that pixel is valid. For that I had to generate this one. How is it generated?

In fact this generation is already done here at this specific edge of the clock, that is, it completes one clock prior before hand. So I want to delay that computed value by one clock pulse. Therefore, I put one more register. That is what I had explained earlier. These are all reset conditions and if you want you can set this register. Once again reset

based upon some other condition. This is power on reset, and this is reset for satisfying some other condition when the pixel is not valid you may have to reset. We had to do this setting and resetting accordingly and judiciously. There is a clock input and also a hold just like any microprocessor-based signal system.

The set point will happen based upon the A bar B C bar condition being satisfied and A B C condition satisfied. It will set the reset, this is the reset. So this register will be reset here. Whatever is the register value here, we will be delayed by one clock pulse by registering in the next register and outcome is the desired value that you need. Let us not go into modelsim because it is too crowded and we cannot view. For your sake, it has been zoomed out and then stored in paint. We will see that once again. That is here.

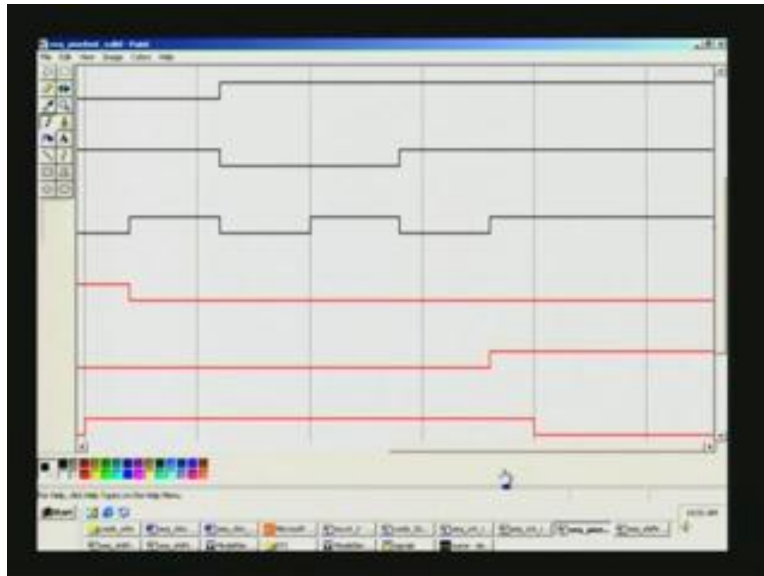
(Refer Slide Time: 39:35)



So you have a reset clock then hold probably. Hold is not used, so let us not worry about that. Clock is a train of pulses here and reset is active here. Only at that point of time, the two registers are pixel out P, that is the first register we saw and pixel out valid and this is the actual decided one. Something is happening here. We will see what is happening and we have also seen this will be set at a particular point of time for some condition of A B C and reset at another point of time. Before that let us see at reset point what happens when it goes low what you need to do is reset all the outputs. That is what happening here

in this condition, and mind you these two are assign statements basically, so they are all combinatorial outputs that will not be reset and naturally coming to the normal operation that is right from here and subsequent positive edge it becomes active only at that point of time. The only thing you have to remember is that the last three here are the A B C inputs. Once again it is in binary progression. Now, let us see what happens to this set and reset. Set is here. We have seen I think it is A bar B C bar or something. Let us see this here. This is A bar, then set pixel out. So it happens here for A is 0. This is B; it is 1, so it is for A bar B C bar. Set output goes high here only for this duration.

(Refer Slide Time: 41:30)

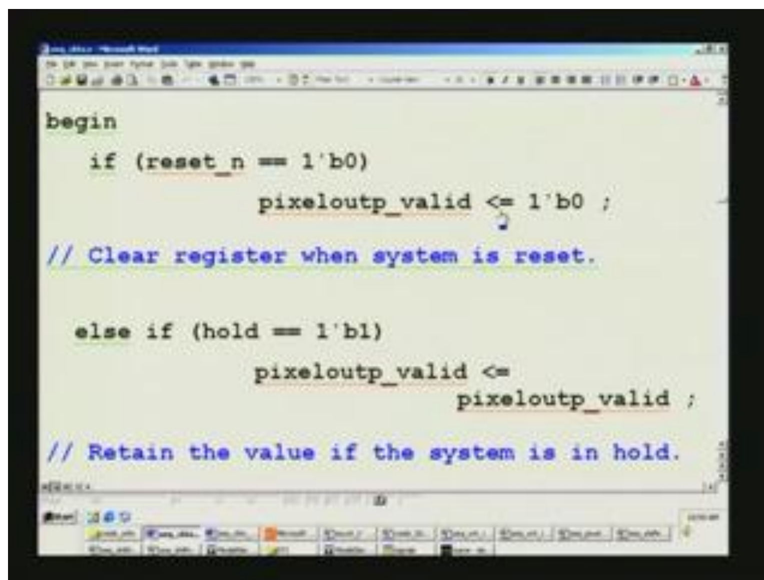


Similarly, reset will go for, this next one is the reset thing; it goes high only for this condition. This condition is what is A B C? Last three on the top will be A B C. Here it is 1 1 1, so that is what we had here in this. Here, A bar, B C bar for set and A B C for reset that is what we had here. What remains to be done in this pixel is P, pixel out valid. Notice that I said pixel out valid is precisely the same output as this. The only difference is this will happen one clock pulse later because that is the time I want to pick up and output saying that pixel is valid right at this point of time. The user can process further when this is asserted and he can take the actual pixel value. If he has storage elsewhere, you can store at the positive edge of the clock subsequent to its rise. Therefore, it indicates the validity of the pixel. Coming to this here, the last two are pixel out. You see

pixel out p is here, and this is staggered by one clock pulse. When it goes low, this also follows through**.** So it is nothing more than this is the basic logic in pipelining registers also which we will see in more depth when we go to adders and other multipliers, arithmetic expressions, how to compute them, how to evaluate new algorithms for that also will be seen in that.

So you have seen pixel out p valid, this being staggered just by one clock pulse. If you explain this pixel out valid, why it is happening here and why it is going low here that would complete these two registers used in a pipeline or just to create a delay as such. Now let us see why it is going high; it is going high here, right? Let us find out why and how.

(Refer Slide Time: 44:08)



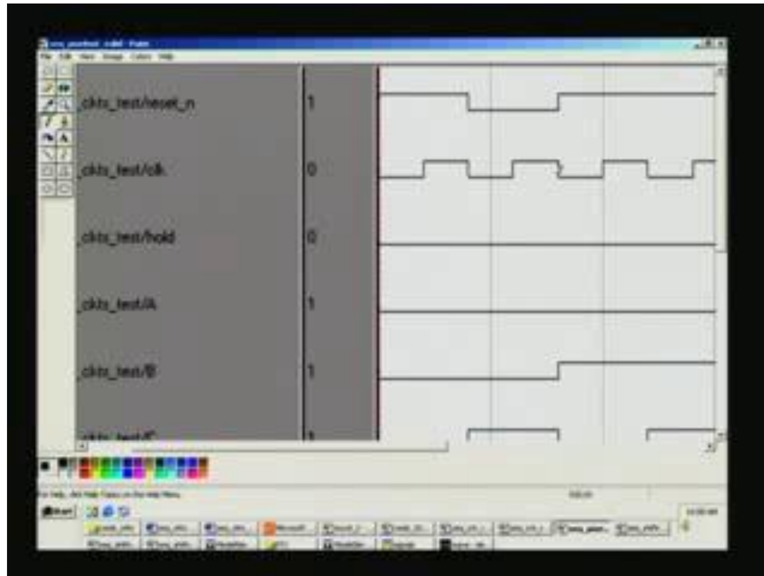
We have to look into the code as such this is A bar B C bar; we have set pixel out; this is an assign statement. Therefore it is combinational and A B C is for reset. At positive edge of clock, we are taking the action here and pixel out valid is reset here. We have not simulated hold; perhaps, you can take it up and find out whether it really holds this signal at that point of time by activating it.

(Refer Slide Time: 44:38)

```
pixeloutp_valid <=
        pixeloutp_valid ;

// Retain the value if the system is in hold.

else if (set_pixout == 1'b1)
        pixeloutp_valid <= 1'b1 ;

// Set or reset when the conditions
// are satisfied.

else if (reset_pixout == 1'b1)
```

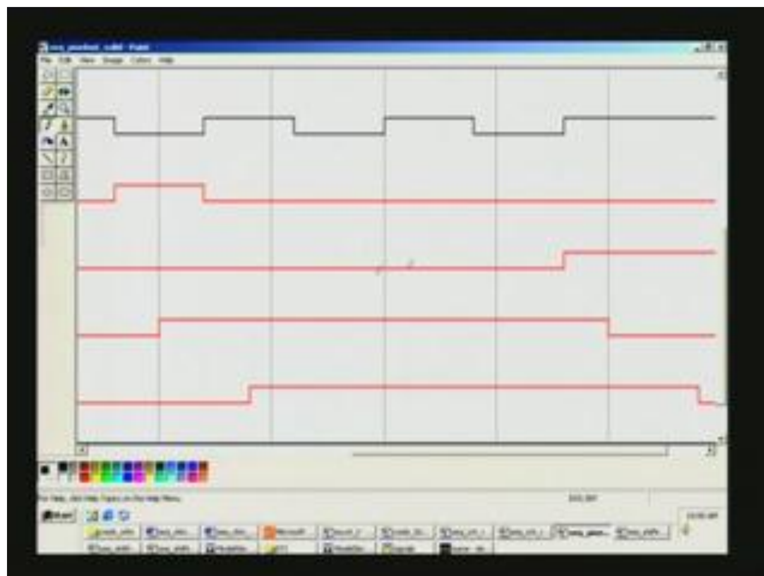
Otherwise, if set is 1 then pixel out p valid is made 1 now. This is the condition that has gone high. So A bar B C bar goes high with the subsequent positive edge clock. You remember that we are right in the always block for positive edge? This also must be satisfied, only then you will go into the reset set mode. So let us verify that particular thing here. Here what happens is, you see that set pixel point we have just now described and it goes high. Immediately here, it has to go right here. It does not go; it is delayed because it has to wait for the positive edge of the clock to come thereafter. So that is what is happening here. You just remember it is somewhere here and the positive edge of clock must appear here. Let us go up and find out whether the positive edge happens on that line. This is the line, although that is combinational logic as such.

(Refer Slide Time: 45:52)



This is in synchronous mode. Any register will always be a synchronous type and it will happen only during the positive clock edge transition. That is how you have set the pixel out p here.

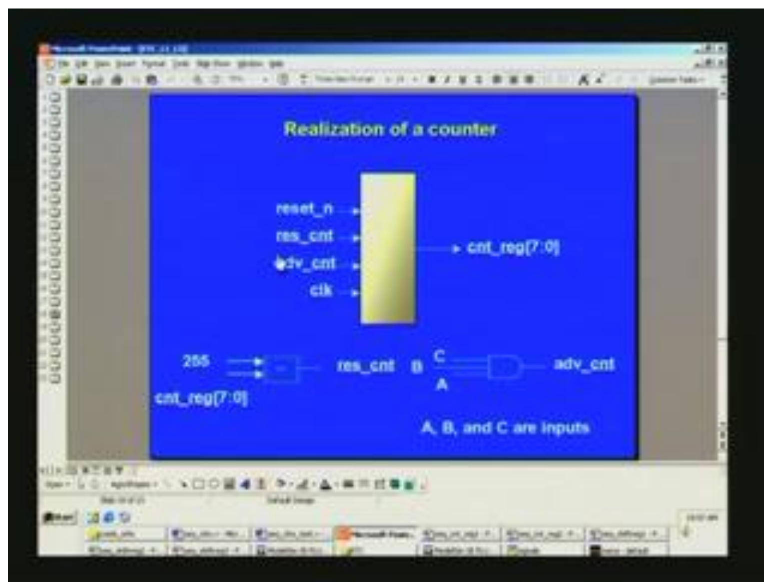
(Refer Slide Time: 46:18)



We have also reset here and that condition is satisfied; it remains high here. After delay you are getting here for the final output and it is going here because this has gone high.

This is the reset pixel out because it has gone high. But it is going low because it has to be synchronous with the positive edge of the clock. This has happened right here but it will not take action here; only at the positive edge of the clock you can take. On the other hand, had you decided negative edge transition, you could have written instead of positive edges in that always block a negative edge. For the same signal, you should not put positive edge clock or negative edge clock. It is a violation. Tool will reject that. So we will close this window and what we have here is a counter here.

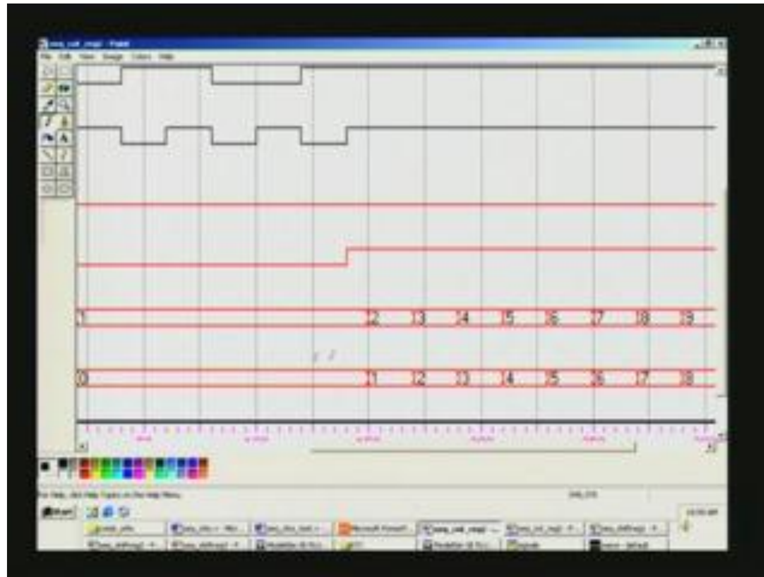
(Refer Slide Time: 47:32)



You have a reset count, then advance count, then overall reset. When counter is equal to 255, then this reset count is applied and you advance this power on reset. That is, increment the counter, only if this condition is satisfied, which is A B C. It is quite simple and outcomes counter register and the counter next was also used inside that you will be seeing in the waveform here. So the waveform for the counter, there are two waveforms here: the first one is based upon A B C condition. If A bar B C bar is encountered, what happens here? You have counter next counter reg here and once again you see it is a multi bit precision; it is in decimal here so it has been put as decimal. We will see that in the modelsim later on. You see that we had to increment the counter only if condition is satisfied. The condition is A bar B C bar, is it not? You can just go through this. These three are the waveforms. Here you can see when it happens, so it is happening here. This

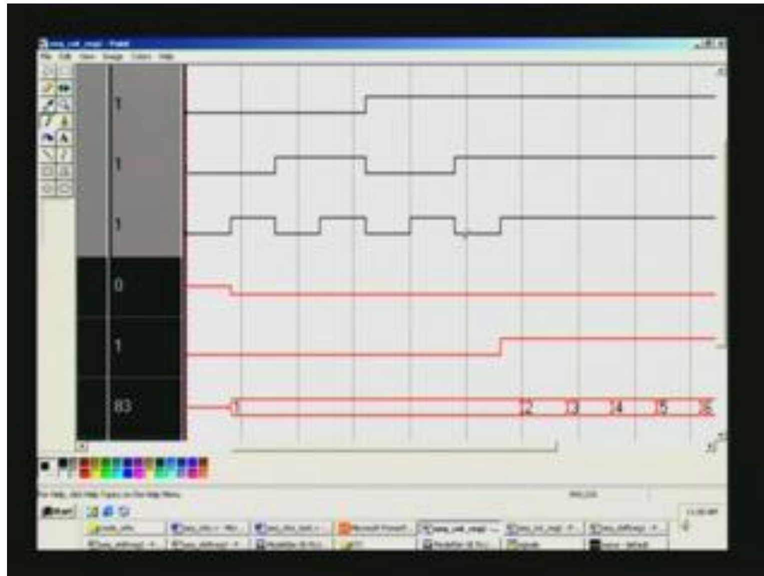
is 0 and that is 1. This is 1 1 0. Is that the condition that we had set? It is the same condition. No. we will come to that later. We will verify that. We will just see the counter output. It is clearly counting here, and it is counting because advance count is going high. What we need to verify is whether it is going at the right time. This is happening at some point of time. Let us see why it is happening here. We will come back to that.

(Refer Slide Time: 50:06)



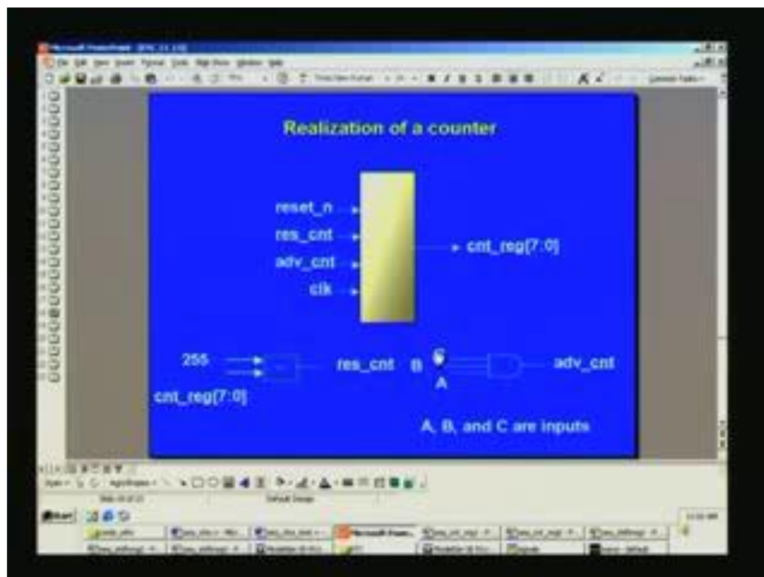
As mentioned before, we had an assign statement for counter register increment outside the thing. This will go in advance when compared to this. That is what you are seeing here. When it is 0 counter, the next is already 1 so it computes in advance. You save time; thereby you speed up the operation; frequency of operation is enhanced because of that. It starts counting 0 1 2, all in decimal here. So is the case. This is quite a long one so I had taken the very first few of this and then at the fag end, another file. You see here because advance count is1, counting is taking place and this got reset here because of the reset pulse here, right at the point here. This is the clock waveform; this is A B C here. Now, let us see at what point of time reset count is going high.

(Refer Slide Time: 50:54)



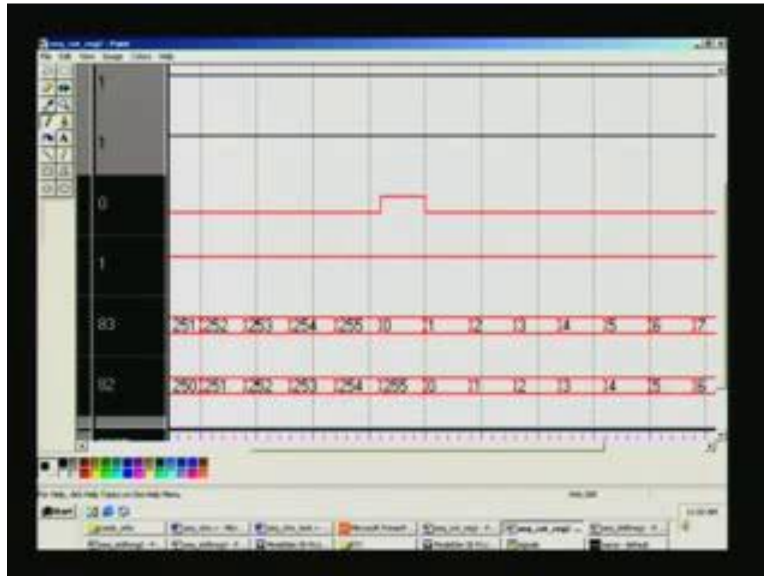
This is A B C, so what you have is 1 1 1. Let us see that. This is wreck ended positive edge of the clock. It sees 1 1 1 but that is basically an assigned statement there.

(Refer Slide Time: 51:30)



Let us see the power point thing. Advance count for 1 1 1, A B C, each will have to be 1. This is a combinational circuit. That is why it is going high. It is advance count. Reset happens only when the count touches 255. Are you convinced about this?

(Refer Slide Time: 52:05)



We have reset count happening; you see this advance count is 1. It has been 1 all through. Had you gone through the modelsim waveforms completely you will be convinced it was high. So, I am putting only the end point, how it leads to the final goal. You can see here, count next is always 1, count ahead of count register. This is the actual output that we desire. When you see it happens that 255 is here. This is for the count reg. When it touches 255, this being combinational circuit it goes high you see. So long it is 255, it goes high. This is nothing but the reset counter.

When it touches 255, the counter is reset. That is what we want. If you desire other than 255, you can put any other value straight away in decimal. That is the advantage. After that again it folds back and resumes the counting because it is an eternal counter. So that is how the functionality is. You can see in fact it has taken plenty of time because it has gone through 0 1 2 3 etc., up to 255. It has counted 256 rather totally. It has gone through so many clock pulses. You can find out how much it is here. But we started not on 0 but after a time lag. So you can find out, I think 20 nanoseconds we did, so 20 into 256 plus some extra time because reset was applied and so on. You can always verify this. This completes the counter. There are more circuits on sequential. We will see that in little more detail on this modelsim in the next class. Thank you.

(Refer Slide Time: 54:20)