

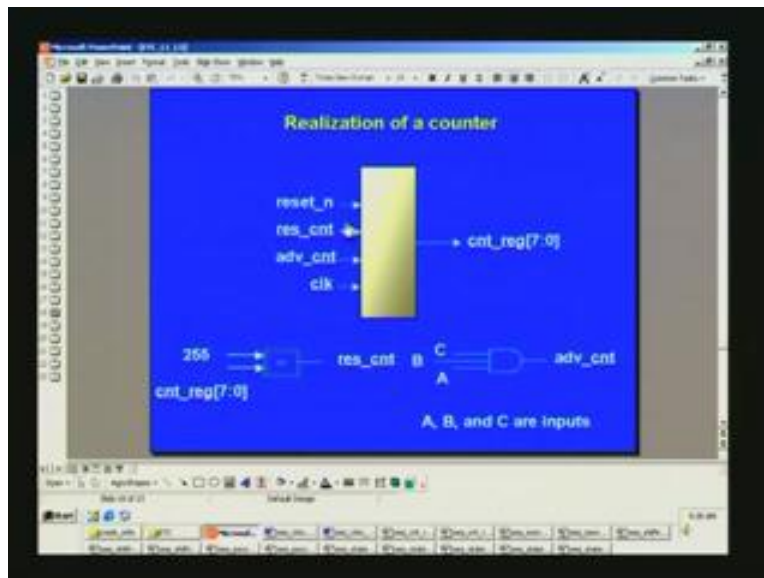
Digital VLSI System Design
Dr. S. Ramachandran
Department of Electrical Engineering
Indian Institute of Technology, Madras

Lecture - 26

Analysis of Waveforms Using Modelsim

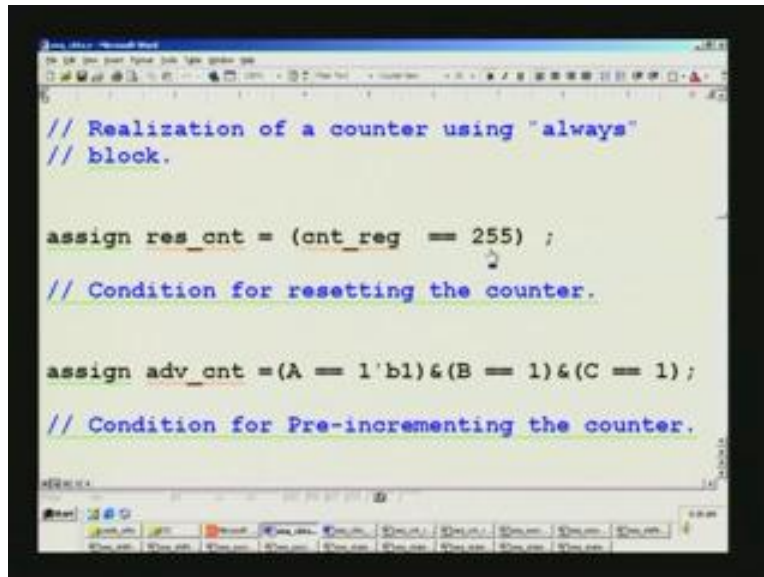
We were analyzing waveforms of a counter in the last class. We will continue to do the same in this.

(Refer Slide Time: 2:10)



As you see, a counter has a reset input and further there is a reset counter here which will take place only when counter reg, which is running counter, is equal to 255. So this setting point can be changed if you wish, another setting here. You can advance the counter when this condition is satisfied that is A B C being one in each. Of course, there is a clock that is reckoned at the positive edge. Finally, there is an 8 bit counter whose output is displayed here.

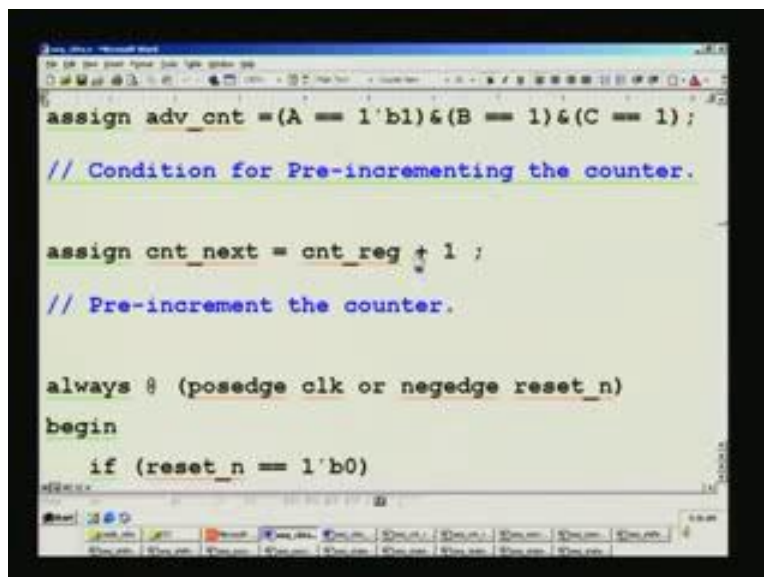
(Refer Slide Time: 02:50)



```
// Realization of a counter using "always"  
// block.  
  
assign res_cnt = (cnt_reg == 255) ;  
// Condition for resetting the counter.  
  
assign adv_cnt =(A == 1'b1)&(B == 1)&(C == 1) ;  
// Condition for Pre-incrementing the counter.
```

We have also seen in actual design where the reset count is when counter reg is equal to 255. We advance only if A equal to 1, B equal to 1 and so on, that is A B C being collectively equal to 1.

(Refer Slide Time: 03:10)



```
assign adv_cnt =(A == 1'b1)&(B == 1)&(C == 1) ;  
// Condition for Pre-incrementing the counter.  
  
assign cnt_next = cnt_reg + 1 ;  
// Pre-increment the counter.  
  
always @ (posedge clk or negedge reset_n)  
begin  
    if (reset_n == 1'b0)
```

We increment the counter register, this is pre-incrementation which will take effect only during positive edge of the clock and that is at this point.

(Refer Slide Time: 3:24)

```
// Pre-increment the counter.

always @ (posedge clk or negedge reset_n)
begin
    if (reset_n == 1'b0)
        cnt_reg <= 8'd0 ;

// Initialize when the system is reset.

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

These are all nothing but clearing the counter when reset is encountered.

(Refer Slide Time: 3:30)

```
    if (reset_n == 1'b0)
        cnt_reg <= 8'd0 ;

// Initialize when the system is reset.

    else if (res_cnt == 1'b1)
// Reset if terminal count is reached.

        cnt_reg <= 8'd0 ;
    else if (adv_cnt == 1'b1)
```

This is corresponding to the counter being 255. And then also, you need to make the counter 0, clear the counter.

(Refer Slide Time: 3:42)

```
cnt_reg <= 8'd0 ;
else if (adv_cnt == 1'b1)
    cnt_reg <= cnt_next ;

// Advance the count by one if the counter is
// still running.

else
    cnt_reg <= cnt_reg ;
```

You count only when advance count is active here. This was pre-incremented earlier and we merely assign here in order to speed up the operation. If nothing else is satisfied, in fact all aspects are covered here. It is a dummy statement wherein you do nothing, just preserve its contents. If you dispense with this, then also it will work

(Refer Slide Time: 04:13)

```
cnt_reg <= cnt_next ;

// Advance the count by one if the counter is
// still running.

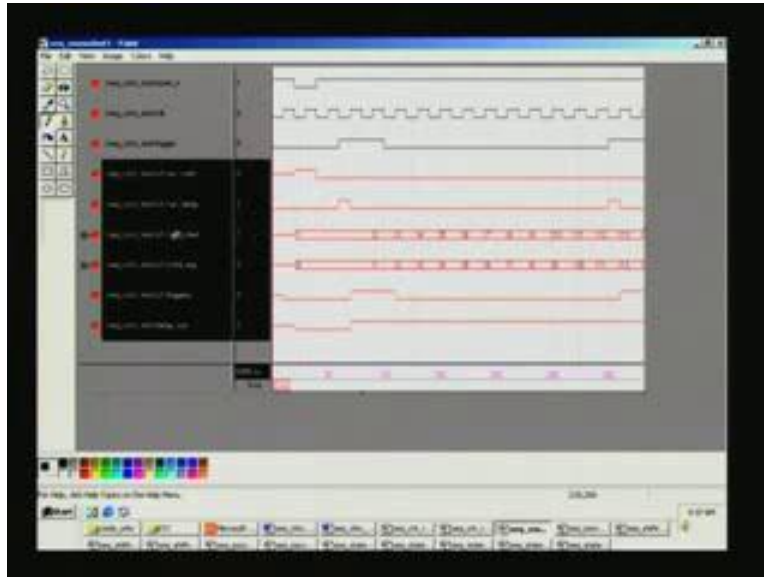
else
    cnt_reg <= cnt_reg ;

// Otherwise, clear.

end
```

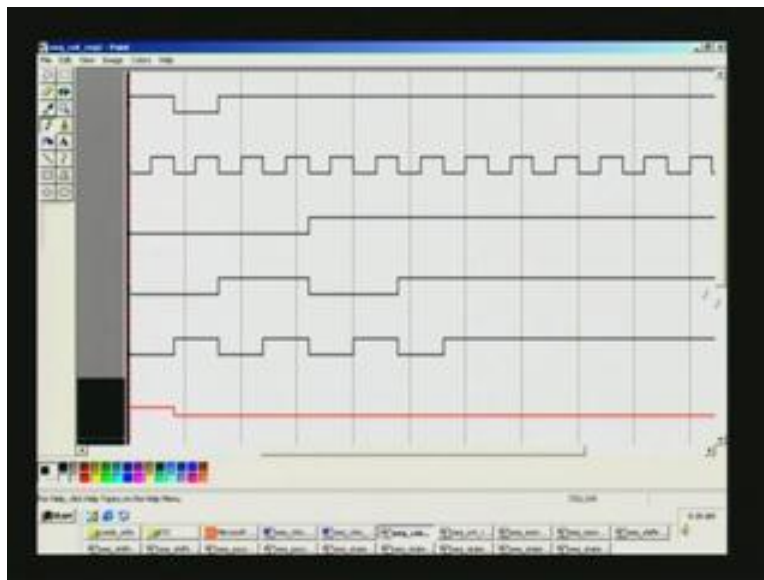
So, that is the end of this counter.

(Refer Slide Time: 05:57)



You can see reset clock A B C are the inputs; first one is reset count here.

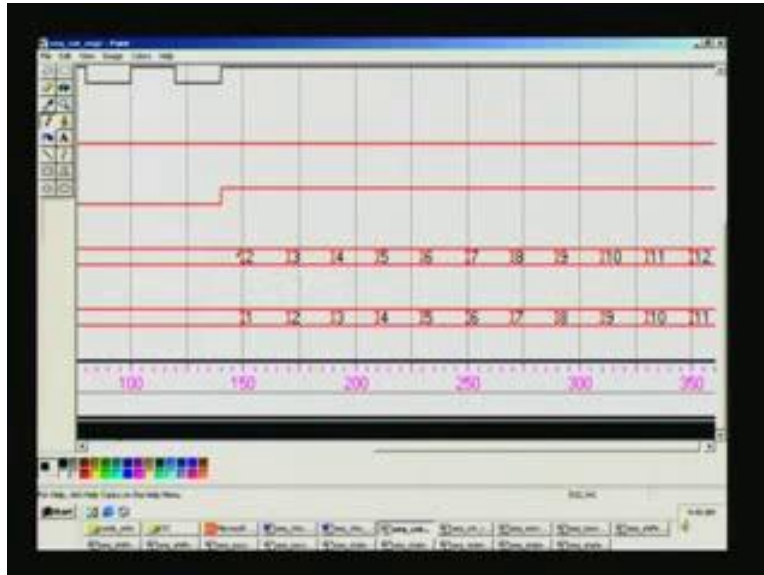
(Refer Slide Time: 6:05)



You can just have a look here and with reset his counter reg is cleared right at that point of time. This last one is counter reg and you can see the time axis clearly here, 50 nanoseconds and so on. Around, I think 20 nanoseconds it goes 0. That is what we had put in the test bench. Here it starts incrementing only at this edge, so you can see that this

is the edge that is at 150 nanoseconds. Corresponding to that the clock has become active there. That is a rising edge of the clock if you notice. Here A B C conditions are 1, 1 and 1. Reset is no longer active; it is active during low phase.

(Refer Slide Time: 7:05)

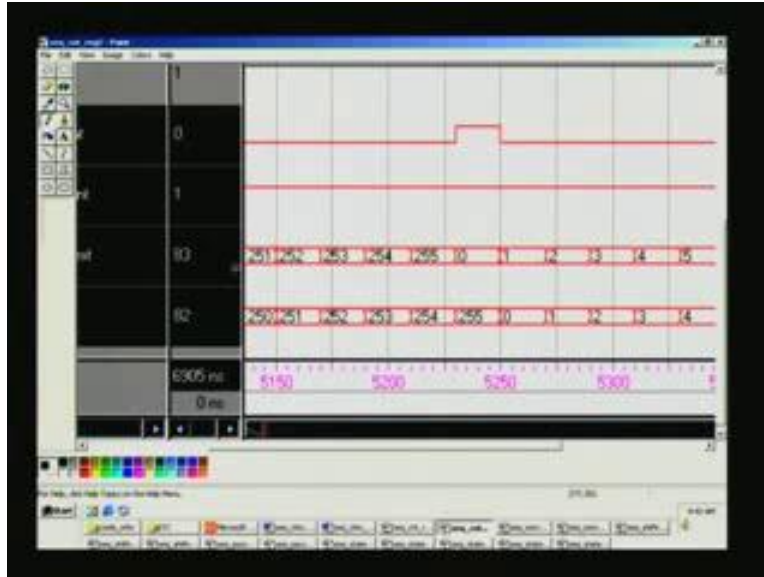


You can see the counting progressing smoothly by 1 at every clock pulse. If you look it will be a clock pulse here, rising edge of the clock pulse here and once again the rise here. Again you can see at all points of time this counter next is pre-incremented here.

Going to the next waveform which is the continuation of this once again you can just assure that all the other points are clearly counting. What shown here is a snapshot of the end points. When it is approaching 255, what happens? That is what matters. Being offline of the waveform view, I think it would be better to have start and end point. That should make it very clear to you. You can see the counter reg is 250, 251 and so on. Corresponding to 250 it is a pre-increment, so it is 251. Right at this point of time, you notice after 255, the counter next has become 0. This counter 255 is appearing only at the next clock edge. When it is 255 this reset counter, which is a combinational circuit of sensing counter reg equal to 255, which happens here immediately goes high being a combinational circuit; it remains so long as it is 255. Once this is done, this will make the counter 0 in the following positive edge of the clock. That is what you see here.

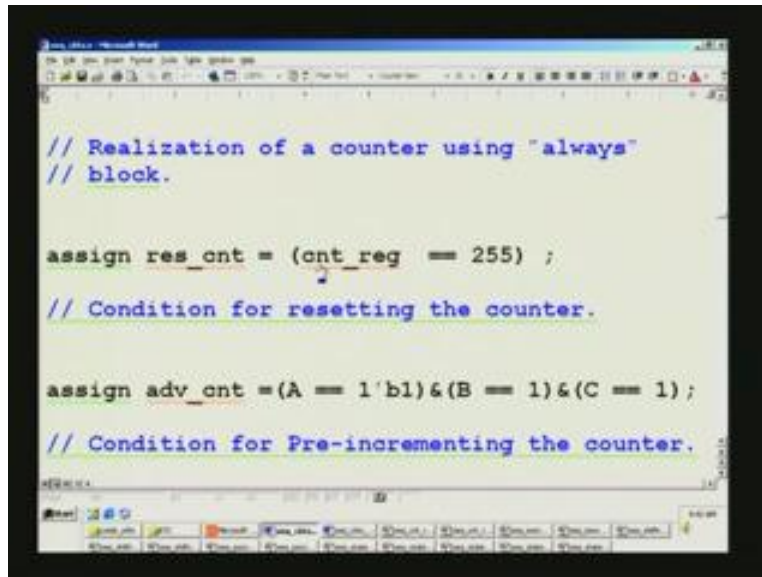
Thereafter it keeps on rolling around; it increases from 0 onwards because it is being reset here. Just zoom here.

(Refer Slide Time: 09:06)



So you can see this here. The last two are the counters. You can see 250, which is the counter edge and pre-increment is 251; this happens to be reset count here. Advance count continues to be 1 and therefore it is counting. Had it gone to 0, it would have reset. So that is how you see that the counter is working. You can see this signal here which resets the counter. This is the same as this 255, same duration because it is combinational circuit that is A here.

(Refer Slide Time: 09:55)



```
// Realization of a counter using "always"  
// block.  
  
assign res_cnt = (cnt_reg == 255) ;  
// Condition for resetting the counter.  
  
assign adv_cnt = (A == 1'b1) & (B == 1) & (C == 1) ;  
// Condition for Pre-incrementing the counter.
```

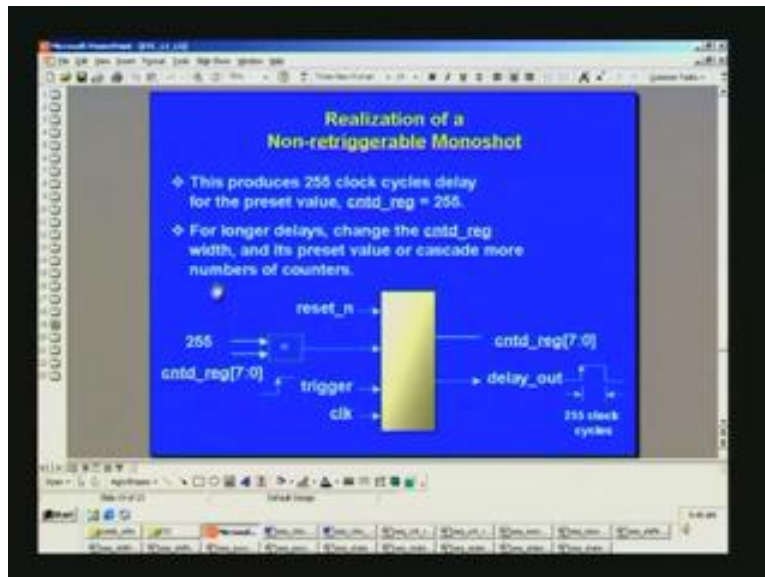
When reset count has become 1 and if counter reg is equal to 255, only then this action takes place. This is a combinational circuit as we have seen. The moment 255 is encountered, immediately it will go without delay. Of course, there will be a great delay only after a back annotation we can see. A word of caution is many people simulate at a very high frequency. You can simulate even at several giga hertz in this simulation or under mistaken impression they have achieved giga hertz rate. That is simply not true. Because the real life situation on the chip, even if it is on the latest technology also, it may not get more than 400 mega hertz or so. So it is a very misleading conception to just simulate and jump to conclusion that you have achieved giga hertz operation. For example, it depends upon the FPGA that you choose. There are wide varieties of FPGAs and different vendors and in each vendor, there are so many categories. Even in each type number, one specific type number of an FPGA, you have once again two, three or four varieties of speeds.

So depending upon the actual device, your speed will depend on that and typical thing you can take is 40 mega hertz or 50 mega hertz on Xilinx and even ultra chips. It is only an assumption based upon our prior experience. I have just put some 50 mega hertz, 20 nanoseconds is the time period. That works out to be 1000 by 20; always divide 1000 by

20, you will straight away get in mega hertz and that works out to be 50. So 50 mega hertz is a typical value on FPGA and that is the reason why I stuck on to that.

If it is ASIC, I have tried 0.13 micron technology done for MPEG video compression, say DCTQ, etc., which we will be showing later on in design applications. In that, I had achieved 0.13 micron technology, around 270 mega hertz. You can jack it up to 300 mega hertz. Only thing, synthesis will take a long time in ASIC. We have finished the counter as such. We will now move on to the monoshot vibrator.

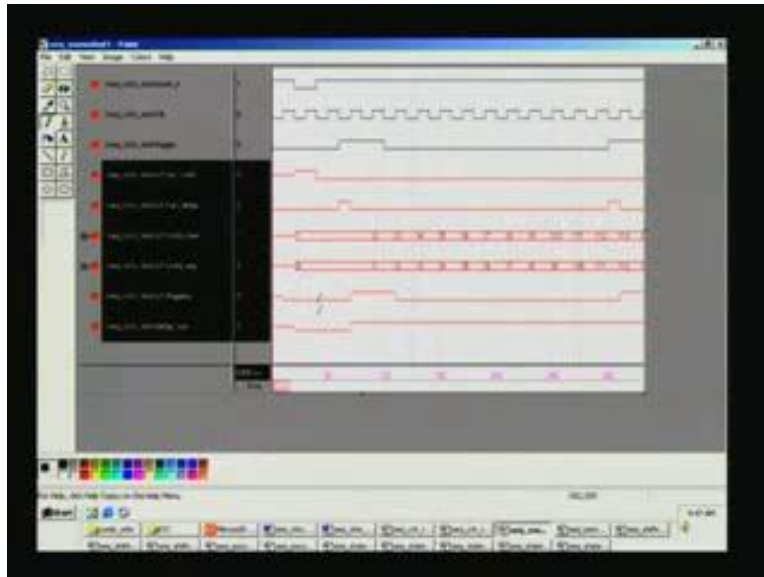
(Refer Slide Time: 12:34)



Earlier I have said we could achieve for a set value of 255, 256 clock cycles. I said that while looking at the simulation, we will have an explanation for that. Now we will do that. But prior to that, let me tell you that from user's point of view, it is desirable to have same delay that you have already set otherwise they will not be aware of these internal complications, why it should take 2 digits. There will be a misunderstanding and they may get the wrong timing. It is always better whenever you design a system to look at the user point of view. The user normally is not very knowledgeable. They can only use. So whatever system you design, you should go down to their level and then cater to that. That must be the attitude of a good designer.

Here there is a reset once again because this is based on counter and that counter we call as counter D; D stands for delay and the output is delay out. You get two clock cycles. If you require, you can also see the counter for the present value or see it internally. All this process triggering will be done by here at the rising edge of the trigger. Clock is also there. Once again all transactions take place at the positive edge of the clock. When counter register equals 255, this will be activated and this will be reset here. That is how you get delay of 255 (14:25). We will see the waveform so as to get a better grasp. Now we will have a look at the waveform.

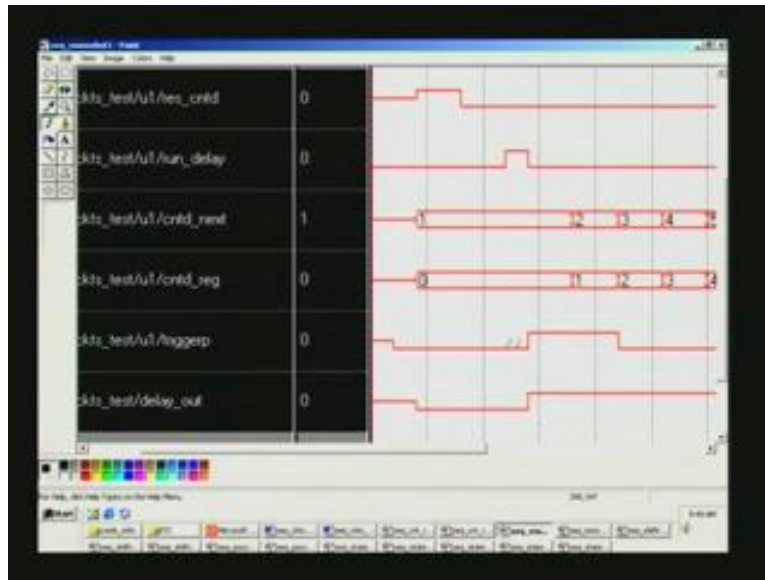
(Refer Slide Time: 14:32)



The first waveform, I will show in an unzoomed condition so that I get a clear picture then we will zoom it. If you see we have a reset here. You may be able to make it out fairly though you may not be in a position to read the numbers. I will read it for you and then finally zoom it for you to view it. To start with, we reset the system. There is a counter as I said that is counter register. This is next, and this is as usual a pre-incrementing of the register. Always see if this is 0, this is already 1, so that is how you pre-increment in order to speed up the system. Otherwise, there is no difference between the two things. You have a trigger P also which is the trigger value. Current trigger value is here and whatever is the current trigger value will be pushed as the previous trigger value for further processing namely, to find the rising edge of this trigger. That is the

reason why we preserve this. We preserve the current value of a trigger and then use this particular thing in the next clock cycle. That is how you sense the rising edge. If you see the run delay, if you see here, it is counting by 1 2 3 4 5 6 and so on. Not much information and we will come back here also.

(Refer Slide Time: 16:17)



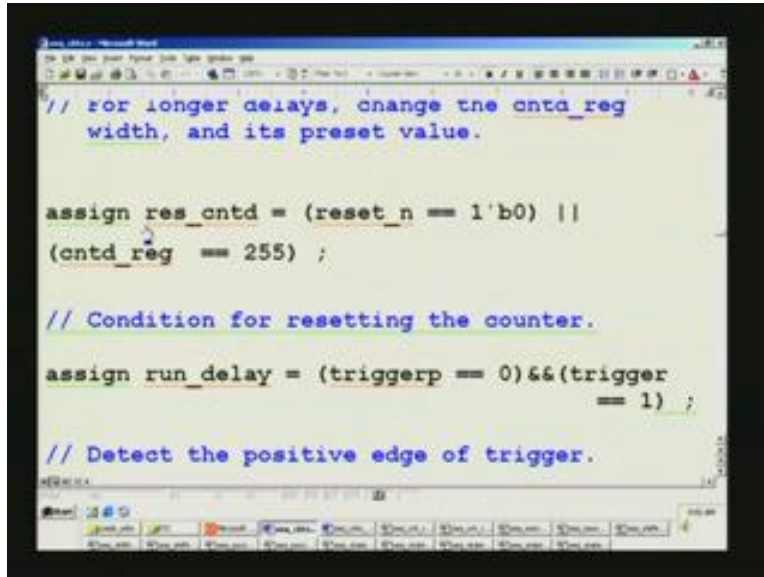
If you see here this is the trigger and this is resetting the counter and this run delay is the one which senses the rising edge of the trigger. Trigger is going high here right at this point. The earlier value of trigger is 0 but that must have been preserved in trigger P; trigger P is actually 0, whereas, it becomes 1 at the rising edge of the next clock pulse. That makes it clear. Trigger P is nothing more than a previous trigger. That is how we compare the previous trigger and the current trigger and decide that it is the rising edge. So the rising edge is limited to the clock period; it cannot act fast. Like this you may have if you are designing a complicated system such as programmable controller or programmable logic controller. You may encounter hundreds and perhaps thousands of I/Os. In such a case, so many I/Os might be on this rising edge direction. For that you can implement in this fashion.

(Refer Slide Time: 17:49)



Next we see run delay is rising. Somewhere here, when the counter is 12, I purposely triggered it once again when the counter is already running just to demonstrate this is non-retriggerable. It does not reset here; it merely ignores. You have applied the first trigger. It has started working. While it is working if you deliberately trigger it, it will not trigger. It simply ignores any number of such trigger inputs. Although that rising edge of run delay is still encountered, it will be ignored because the counter is already servicing the previous trigger. That is what we want and that is the meaning of non-retriggerable mono. The counter is clearly working 0 1 2 3 4 and so on, and it keeps going. We will come to the end and see. Another thing is the actual output is delay output that you have seen here. That is this 1. You see that when reset had gone low or the reset counter D. That is also a **oaring** of the actual system reset as well as the reset for the counter D being equal to 255.

(Refer Slide Time: 19:30)



```
// for longer delays, change the cntd_reg
width, and its preset value.

assign res_cntd = (reset_n == 1'b0) ||
(cntd_reg == 255) ;

// Condition for resetting the counter.
assign run_delay = (triggerp == 0) && (trigger
= 1) ;

// Detect the positive edge of trigger.
```

If you look at the code for this, we have combined two resets into one statement, that is, power on reset here and also counter D being equal to 255. When any of this happen, this an OR here, so this will go high. If you see here and run delay, we have already seen trigger P and trigger 1 being equal to 1; rising edge that is how you sense. All these are combinational. Therefore when this happens, immediately it will show at the output. These are all mere incrementing. You can put a flip flop and realize straight away as a clock input. But another rule we will be violating. All the flip flops should have a system clock alone fed to that. So naturally that is not the correct way of RTL coding.

(Refer Slide Time: 20:42)

```
always @ (posedge clk or posedge reset_cntd)

// Because of posedge res_cntd, delay_out
// will be 255 clock cycles instead of 256.

begin
    if (res_cntd == 1)

// Initialize when the system is reset
```

The best thing is to realize in this fashion. After all clock speeds are very high, you are dealing with 40 mega hertz, 50 mega hertz. How does it matter if you are delayed just by a few nanoseconds? Here we see that reset counter D equal to 1 is checked. We are in the always loop at positive edge for the monoshot.

(Refer Slide Time: 21:44)

```
// This produces 255 clock cycles delay.

begin
    cntd_reg <= 8'd0;
    delay_out <= 0 ;
    triggerp <= 0 ;
end

else if (delay_out == 1)
```

For that we are resetting here.

(Refer Slide Time: 21:50)

```
end

else if (delay_out == 1)
begin
    cntd_reg <= cntd_next ;

// Advance the count by one if the
// timer is still running.
    triggerp <= trigger ;
end
```

Otherwise, increment when delay out, which is already on, the timer is on, delay out equal to 1 is the output of the timer. When it is on then automatically starts incrementing. That is how the code has been written. Otherwise preserve the contents.

(Refer Slide Time: 22:23)

```
// Advance the count by one if the
// timer is still running.
    triggerp <= trigger ;

end

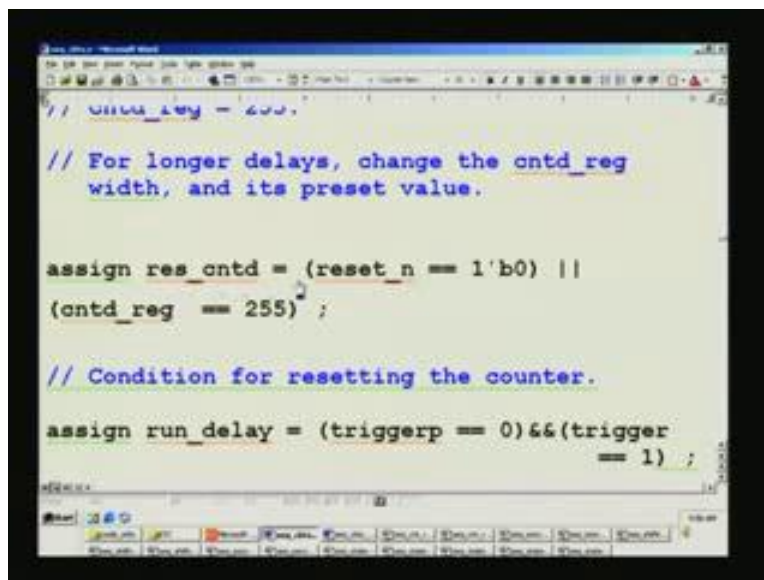
else if (run_delay== 1'b1)
begin
    delay_out <= 1 ;
end
```

If you sense rising of that run delay, that was your question, it is tested here. Only then you make that delay out 1. This will happen only one time when previous value is 0 and

present value is 1 for the trigger. Subsequently previous value will become automatically 1 because next clock will be updated. This will be encountered only at the very first rising edge. If that happens, this is already 1 from that assign statement. Under that condition only this will be made 1. Once the always positive edge clock, whatever signal that you put, they are all automatically registers. That is why once you have set the value it will remain in for an indefinite time till you redo it; otherwise, just preserve here.

What I want to point out here is, (Refer Slide Time: 23:16) let us see how 256 clock cycles and 255. That is all we have to consider in this.

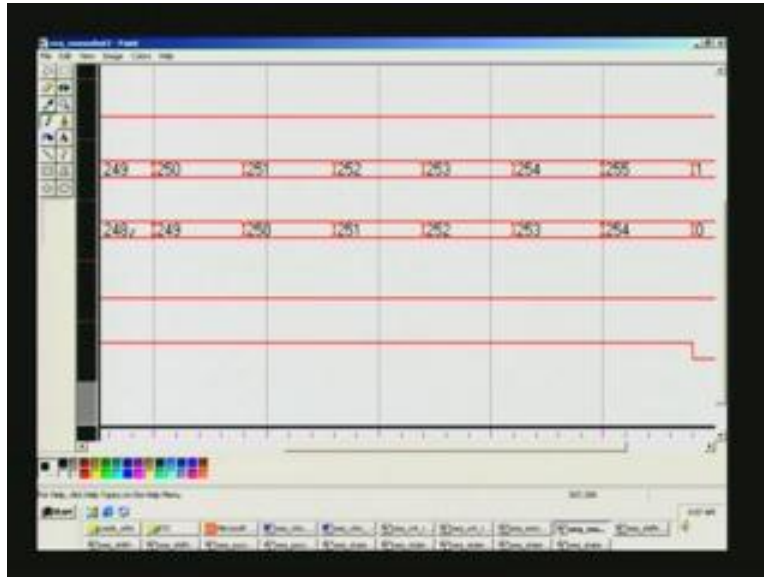
(Refer Slide Time: 23:36)



```
// cntd_reg = 255.  
  
// For longer delays, change the cntd_reg  
width, and its preset value.  
  
assign res_cntd = (reset_n == 1'b0) ||  
(cntd_reg == 255) ;  
  
// Condition for resetting the counter.  
  
assign run_delay = (triggerp == 0) && (trigger  
== 1) ;
```

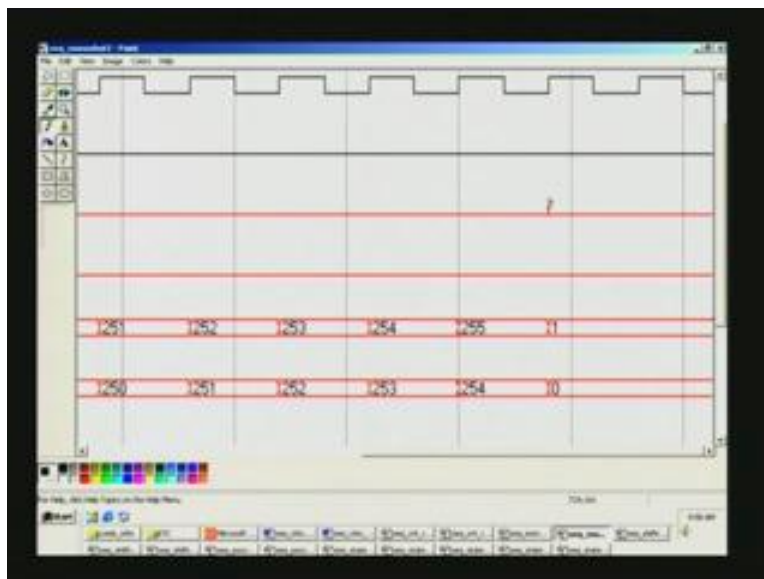
Now let us see what is happening here. We will see the waveform of this. Also this reset counter is a combination of the power on reset as well as counter being 255. You have to follow very carefully because this concerns very sharp pulse which is going to result and we are going to see in the diagram. We have already seen this one. There is nothing more to add to this existing one. We will go on to the second waveform of the monoshot where you see the end towards the end. We will see once again. You have a counter D here. Delay out is the last one that is already high.

(Refer Slide Time: 24:21)



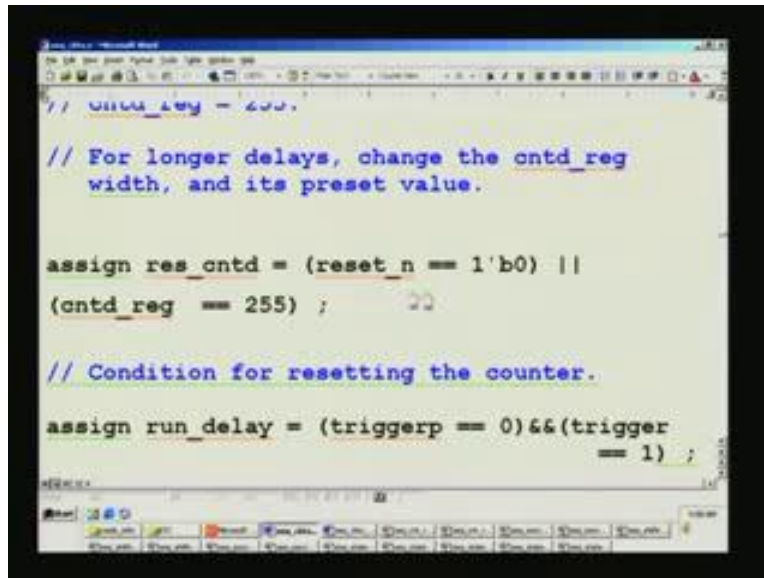
You can see the counting taking place once again, 248 corresponds to 249 in the next value. Now let us see what happens. Follow it very carefully. As a designer you should know the intrigues of design. This is the most crucial step here. What is shown here is, this is the counter D reg, that is, 254. What is happening after 254 is it is incremented to 255 actually but you see only 0 here. What is happening at that split second? You can see a very narrow pulse here. This corresponds to reset counter D.

(Refer Slide Time: 25:05)



Follow it carefully. So what happens when 254 is equal to 255. It goes for a very short while. Even if I zoom out to a very large value, you will still see a single line because we are dealing with the simulator and there is no gate delay coming. But fortunately the simulator is very well done and they do not have a bug on that score. At least show a very line there.

(Refer Slide Time: 25:30)



```
// cntd_reg = 255.

// For longer delays, change the cntd_reg
width, and its preset value.

assign res_cntd = (reset_n == 1'b0) ||
(cntd_reg == 255) ;

// Condition for resetting the counter.

assign run_delay = (triggerp == 0) && (trigger
== 1) ;
```

Look at the code. The code is, when counter equals to 255, which is precisely what has happened; the moment that happens this goes to 0 because this is a combinational circuit. This is nothing but an OR gate. The moment the counter touches 255, immediately after a delay of this OR gate, this will get reflected here. Are you getting it? If this happens what happens to this statement? (Refer Slide Time: 26:00) That is reset counter D. Follow this carefully, this is crucial.

(Refer Slide Time: 26:08)

```
if (res_cntd == 1)
    // Initialize when the system is reset
    // or if the terminal count is reached.
    // This produces 255 clock cycles delay.

begin
    cntd_reg <= 8'd0;
    delay_out <= 0 ;
    triggerp <= 0 ;
```

We have a positive edge of the clock just now arrived at which made the counter to 255. Immediately following this within less than a nanosecond or around a nanosecond, this assign statement of reset control D is making it 1. What happens if that is 1? Earlier it was 0. What is put here as a condition is OR positive edge of the reset control D. Immediately this condition is satisfied and right at the positive edge following that this is also satisfied. It immediately enters within a nanosecond

(Refer Slide Time: 26:50)

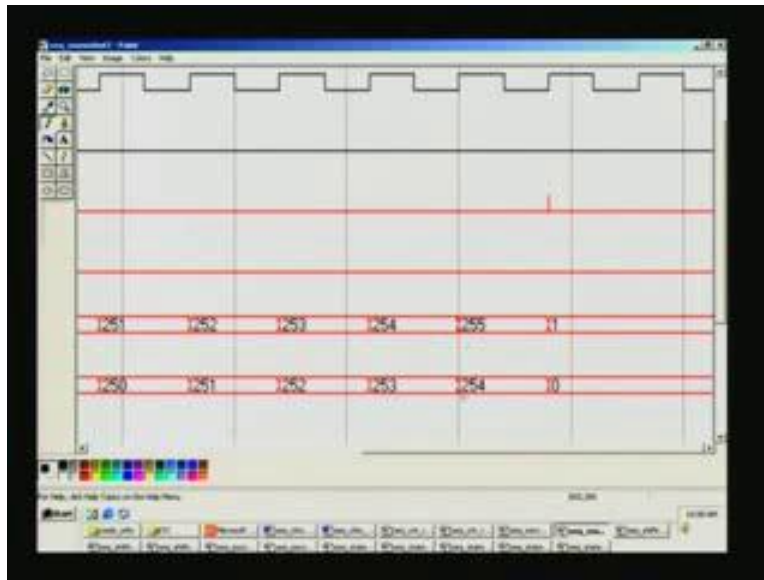
```
// or if the terminal count is reached.
// This produces 255 clock cycles delay.

begin
    cntd_reg <= 8'd0;
    delay_out <= 0 ;
    triggerp <= 0 ;

end
```

It satisfies this condition and then goes on to clearing the counter itself. That is precisely what you have. Simultaneously delay out is also made 0. That is how you get that.

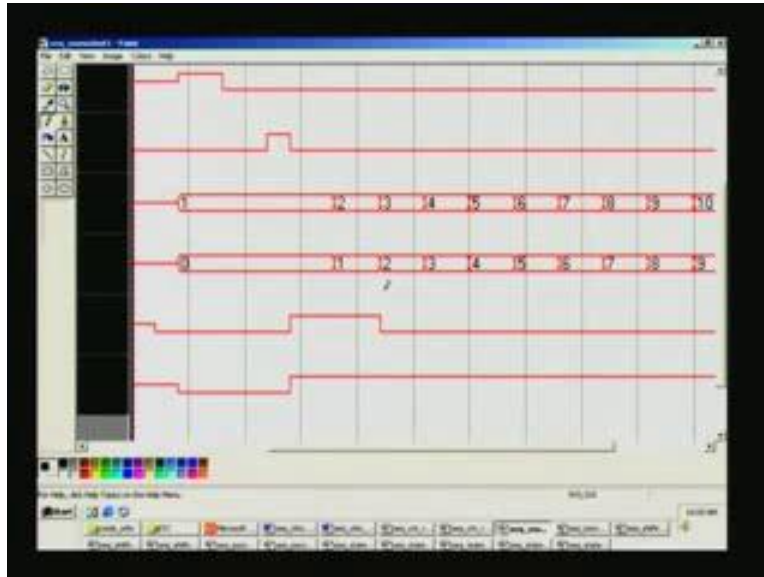
(Refer Slide Time: 27:00)



Looking at the waveform once again you see here 254, the last one here is the counter reg. This is the delay out; it is still 1 here. Now, when it is 254 at this point of time, instead of 0, 255 has come over there. Because of the combination action, reset control D is activated here. For a split second only it is there, and the moment 255, this reset counter is 1, 255 is cleared to 0 that is what we have seen. Also you should have the output as 0 at precisely that point of time this happens. As a designer you should be aware of all these things. One slight mistake in the code will make it **tough (28:04)**.

Next is 1 more point I would like to mention. This is 254. In the earlier case we saw that it is this delay out. Make a note here. This delay out is going high here and remains high till it was 254. But this is corresponding to this 0 here; 0 itself is spread over a one clock pulse. In other words, last one was 254 plus 0 if you add you have got 255 clock cycles delay. That is what you have got. This proves that whatever is the setting, you get precisely that.

(Refer Slide Time: 30:18)

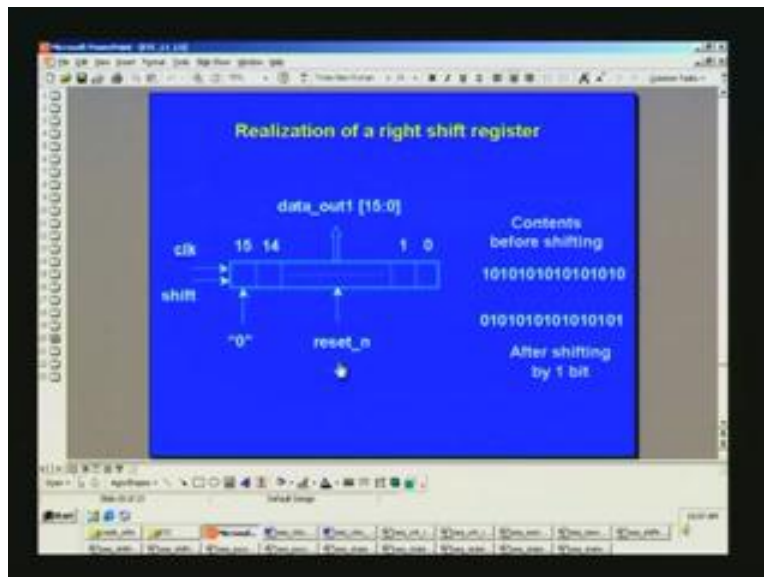


Now, another point is earlier I said 256 cycles. Why that happened is that the code was different. I did not take that reset counter D as asynchronous, that is an asynchronous way of setting in this. Earlier what I have done is that I had taken reset counter inside itself. So when reg is 255, I make a decision (Refer Slide Time: 30:43). Although we say here, this is for combining the power on reset as well as the counter reaching 255. Earlier what I had done is I did not combine this. Why I combined here is I have also said always block is a priority encoder. Therefore we had to limit the number of if else, else if to less than 4 or 5; by combining the two I have reduced by one. That is the advantage. Thereby speeding up the operation, and of course with a little understanding complexity which we have already cleared and you can see here how many ifs you can just see here; one if is there, then 2, else if 3, just with 3 we have made. Otherwise, it would have reached 4 with the earlier thing. Why 4 is here is that if reset will be negative edge reset, first if. We had put first if as here instead of this, we had used a negative edge reset underscore n. Because we have combined that it happens to be the same condition, whether this condition takes place or this. In the earlier way of representation, this has been separated out. This power on reset has been dealt here and the actual counter reaching 255 has been dealt somewhere here in between. It added one more else if, thereby slowing the system. Instead of 255 set points we got 256. The reason being that when it touches 255, in this case, you would have very easily seen 255 completely for one complete cycle. What we

had seen here is 254. In the previous case you implemented that one. After 254, it would not become 0. It would go to 255 and next only it will go to 0. That is because counter reg is activated in the present clock cycle which will be taken into consideration only on the arrival of the next clock.

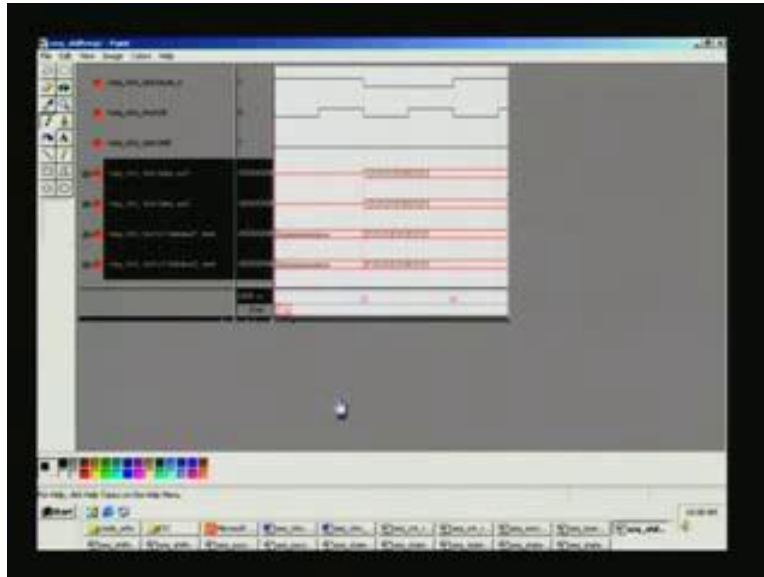
In other words, in the previous design, it has to wait for the next. Are you clear about this? Thereby 255 plus this 0 remains the same. Earlier you would have got 256 clock cycles. That is why I have said by looking at the waveform you can understand better. That is why I have deferred on those days and also I changed only this. Are you clear about this? We will move on to shift register.

(Refer Slide Time: 34:26)



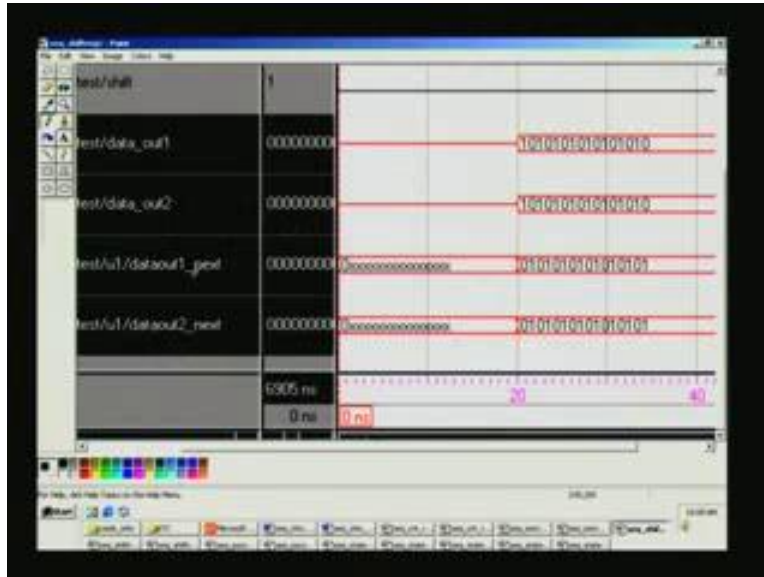
Now, we will move on to the shift register. In this shift register, we had a 16 bit shift register. Data out can be had here. We had two types of shift register: data out 1 and data out 2. Both primarily the same except that the way of writing in assign statement for shift was different; one was with less than symbol and the other one was to concatenate simply. There is a clock input and a shift here; vacated bits are filled by 0, which is also a power on reset. We start with the contents 1 0 1 0 and then right shift by 1 bit because this is a right shift by 1 bit. You should get this. So let us look at the waveform.

(Refer Slide Time: 35:04)



This is the data out 1. I will zoom out further. Earlier, I think you had asked this question, can we have 16 bits represented? That is precisely what we have here. Whenever there is a reset that is active here and clock is free running, it can start any time or after the reset, it does not really matter because reset is asynchronous active low. Asynchronous means whenever that happens, immediately it will take an effect whereas in synchronous it will take effect only when clock arrives. This is asynchronous for the reasons we have covered earlier. We see that this data 1 is preset by this value whenever reset is encountered. That is why you have the same 1 0 1 0 appearing here, and so also data out 2, which is alternative approach for the shift. Once again your corresponding next will be there pre-shifting.

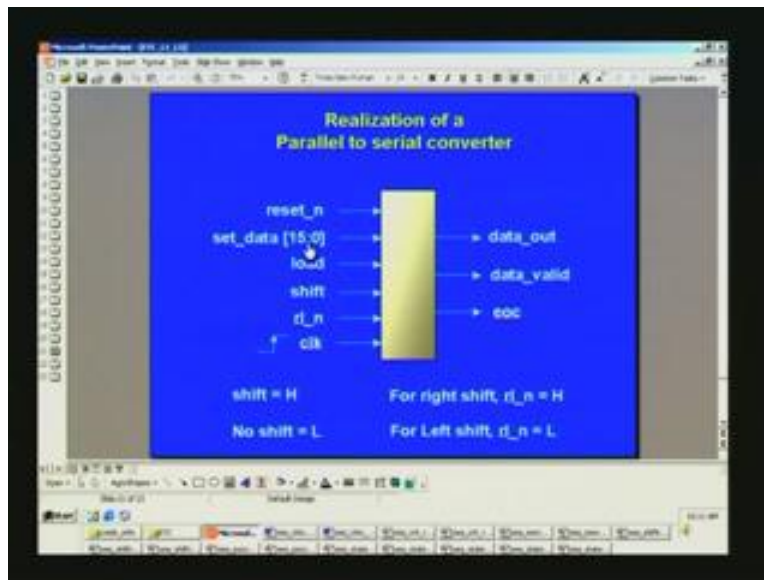
(Refer Slide Time: 36:08)



Data out is here. It is enough if you have a look at the data out. We will just have a 1. Data 1 next if you see it will always pre shift because right shift by 1 bit. So, 1 0 has become 0 1 0 1 0 1 here because it was right shifted. So this will be assigned to this only with the following clock edge. If you have a look, you can see that this is the data 1 and data 2, which are exactly same because the same problems are solved in two different ways. If you look at it, this pattern will keep shifting. It is there in the next two slides. So I will once again zoom this. Coming to this, these are all the data 1 and data 2. So the first value was 1 0 1 0. With the arrival of the next clock pulse, it has become 0 1 0 1 and both are same.

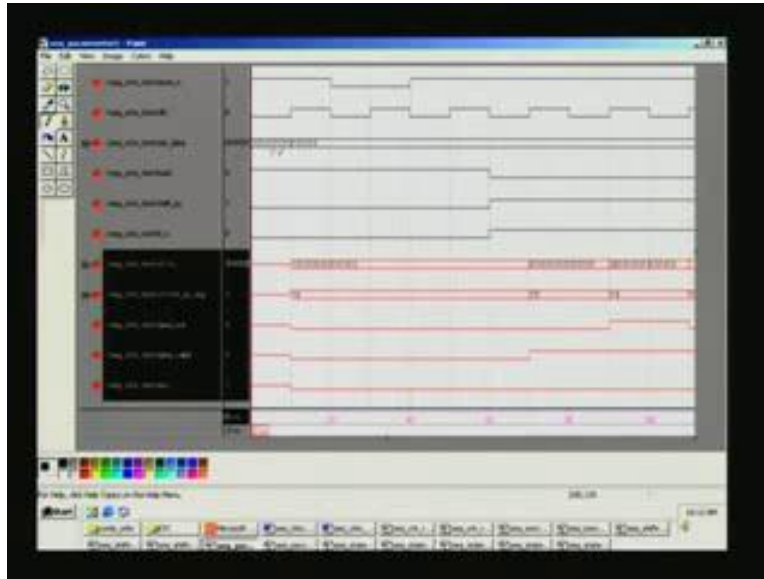
If you see here the same pattern will be continuing. You can see 1 0 1, all are 0s. This 1 will get dropped so you have 1 0 here, and then this 1 will move here as shown and finally that 1 will also disappear. That is how you see a shift register working. We had seen a parallel to serial converter. There is a reset.

(Refer Slide Time: 38:29)



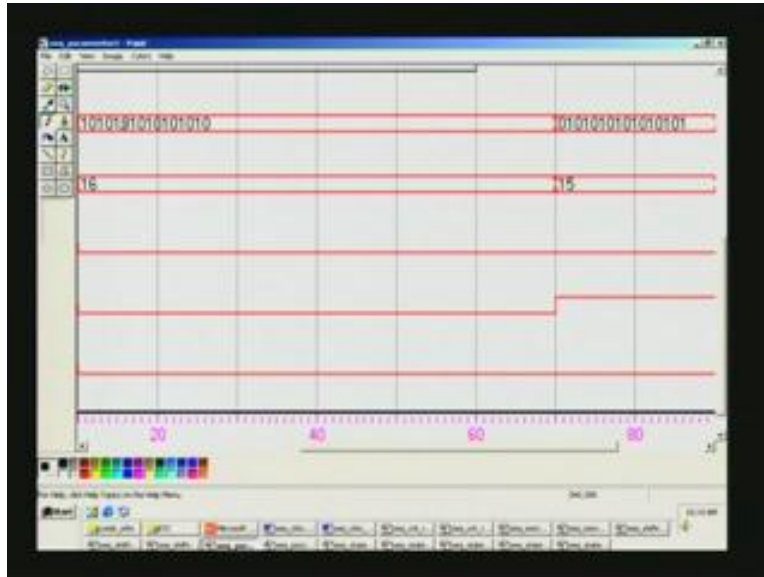
You can set the data and then load it and then shift once you have loaded. What it does is we have 16 bit parallel information which loads into the counter shift register here. Then shift either left or right depending upon this control. RLN 1 being a right shift here and at the positive edge of the clock as long as shift is valid. There will be a data out here which will be the single bit because it is parallel to serial. So whether MSB of the data arrives or LSB arrives it depends upon whether it is right shift or left shift; for left shift 15 will go out here. We will look at the waveform because we have already seen the code. There will be an end of conversion once all the bits are serviced. We also had internally one register allocated for keeping track of number of bits processed.

(Refer Slide Time: 39:44)



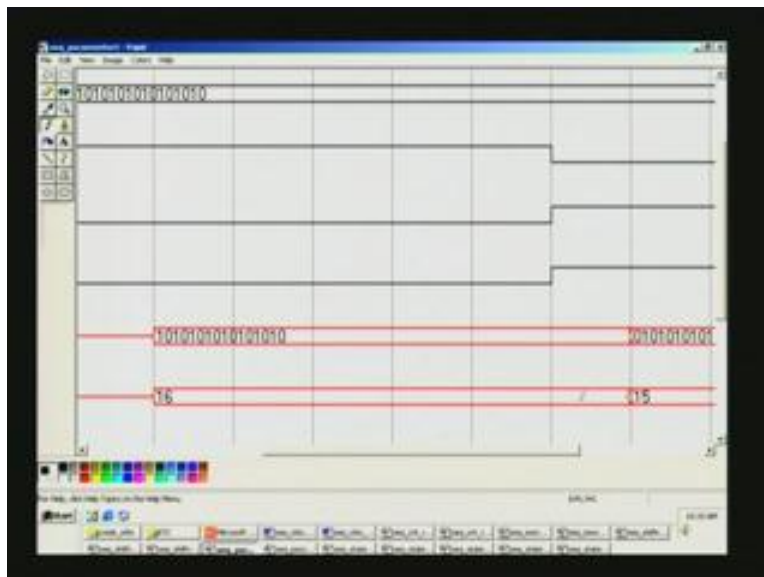
As usual, reset is applied. This is set data here. Also we are setting the same 1 0 1 0 pattern. Internally, there is a shift register of 16 bit. That is preset to the same 1 0 1 0 because that is what we want and at the positive edge of the clock because code has been written like that. This is the counter to keep track of the number of bits. Now you need to send 16 bits, so it is initialized to sixteen here. Every bit going out this will be decremented as you can see, going to 15 here, 14 here. There is a long gap because we have first loaded this value by activating this high and it goes low only here. Then simultaneously we will make the shift control high so that with the next arrival of the next positive edge of the clock the counter starts decrementing as well. As it has output the value, data out is here. To start with, it is all 0 here and what happens is this is the shift register and this is the counter. This is the data out and a data valid and end of conversion is also here. All these are 0s here, data out, data valid and end of conversion right at the point of loading the value.

(Refer Slide Time: 41:29)



Remember the last, this one is the data out and it is in 0 here. Initially this is left shift but it is now right shifted. So it is made 1 here. After this, it will be right shifted including the very first bit. That is why it is going right here and not here.

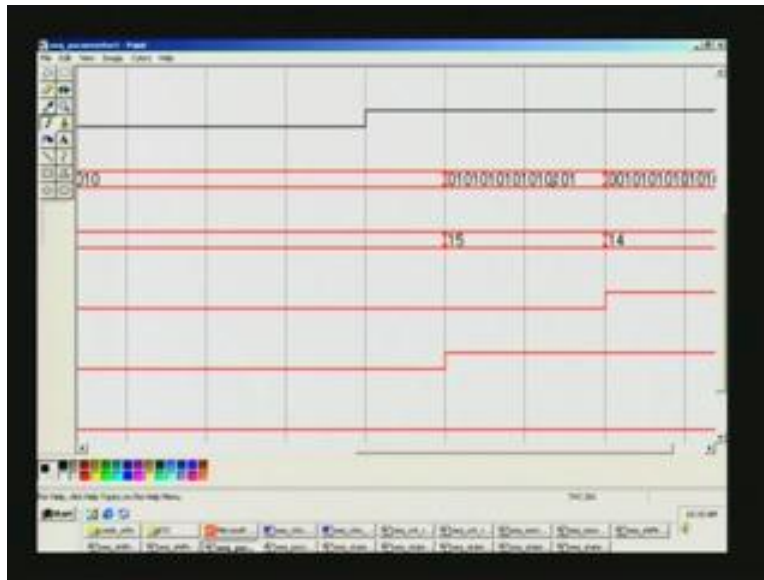
(Refer Slide Time: 41:59)



We started with 1 0 1 0 1. Now you can see that it is shifting 0 1 0 1. Also it is a right shift so actually it was 0 here, first bit. What you need to get is this is the output; data out

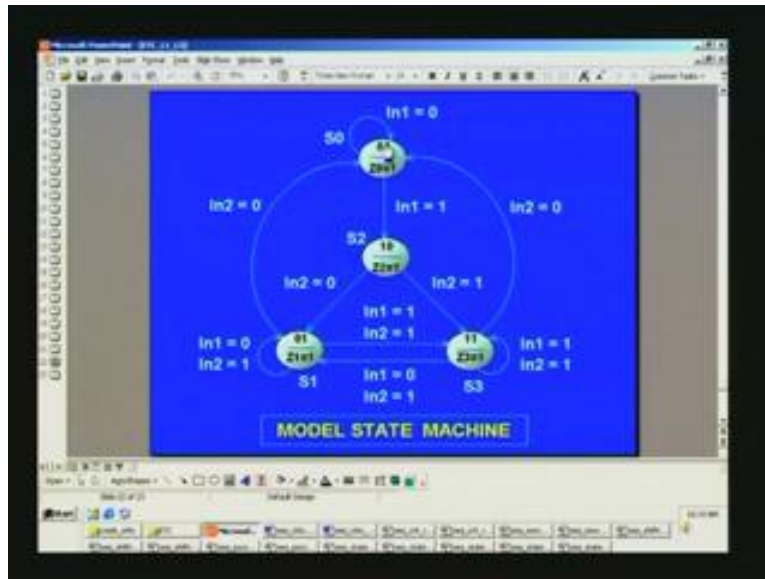
is this one. This is immediately next so it is 0. Because first bit was 0 and second bit happens to be 1 because this is 1 here and this is also right shifted.

(Refer Slide Time: 42:40)



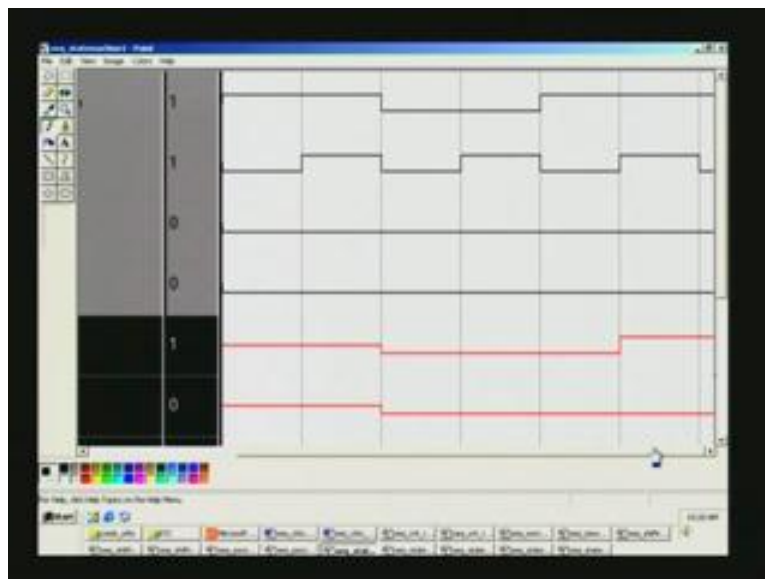
You can see it is right shifted as well as the last bit goes over to this. You can just see in the next because this is the fag end of this. You can see here and it goes on there. Here you see that count is right shift remaining high. You can see that 3 2 1 is the counter, the last value. Every time it is going, here what has happened? This is left shift. I have done both right shift as well as left shift to show. But I should have shown two more waveforms. Since you have the code you can do it yourselves. This is only to show how to analyse the waveform. You see that it is working for this 1 is here and 1 goes to this being left shift here and finally 0 is the thing. You can see corresponding outputs at every point of time. Have a look at this. It has to keep on toggling from 1 to another because alternate values are 1s and 0s. We will go on to the state machine.

(Refer Slide Time: 46:27)



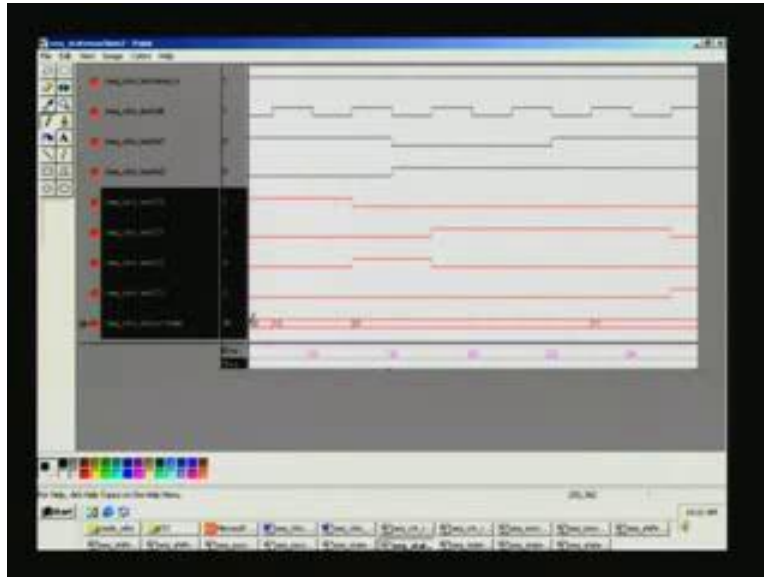
We have already covered S0 through S3 as well as the code for this and test bench for that. It is enough if you just have a look in the waveform, you can make it out. For example, if it is S0 and so long as input is 1, you remain there and also lighter, I mean an LED any output you can turn on the output as the 0. For every other state you have Z1 representing the same state. Based upon the input conditions In1 and In2 it is going from one state to another. You can see one of the waveforms here to start with.

(Refer Slide Time: 47:17)



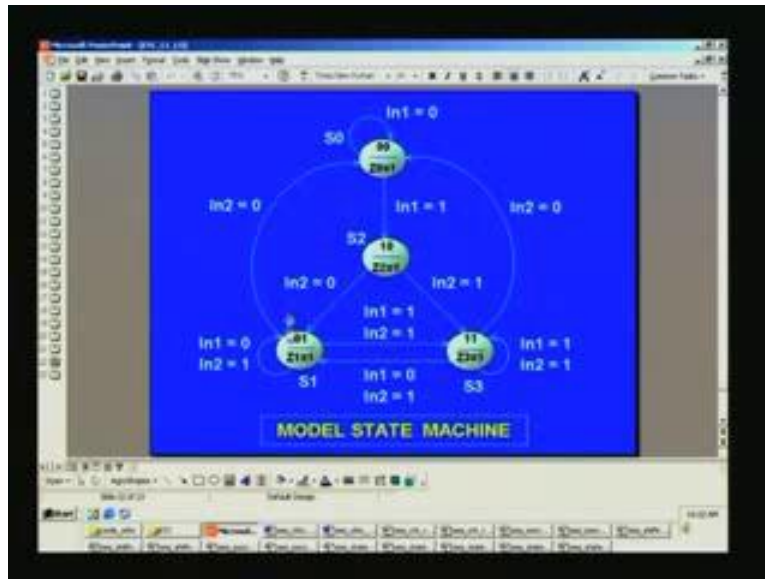
It is in reset here. In1 and In2 are here, both are 0s. What will happen here is all the outputs are cleared because of the reset. It only shows the reset condition, nothing much in this.

(Refer Slide Time: 48:04)



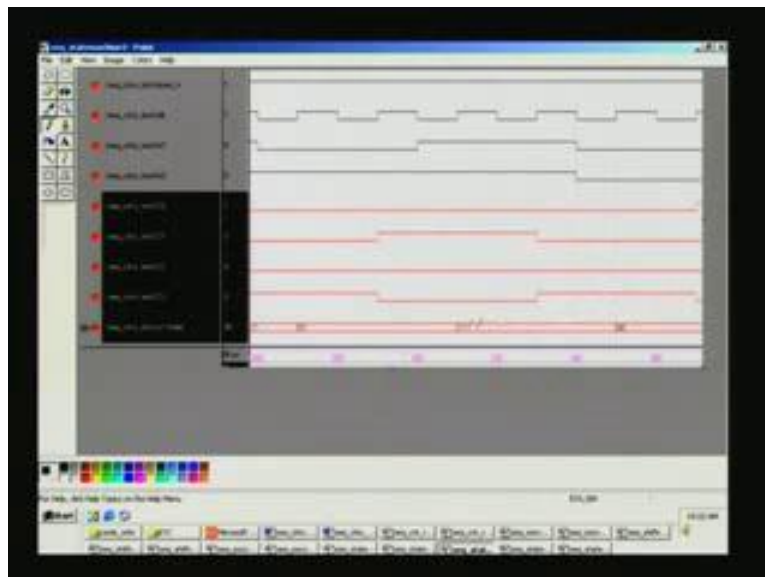
If you go to this second one you can see it is advancing. This is the state field. Last one is in 0 0 condition. The very first state, which is going to change from 0 0 condition it goes to 1 0, provided it depends upon what inputs are; input 1 is 1 and input 2 is 0. This means input 1 is 0. It has to be the same state. That is what we have already seen. If it is 1, it has to go to second state. It is going to second state 1 0, and it will go again to next state 0 1 depending upon input condition. If it is here, its input is 1 and 0 here. But it now in 1 0 state, that is, S1 state. In S1 state what will happen?

(Refer Slide Time: 49:13)



In1 is 1 and In2 is 0. S1 state is here. If In1 is 0 and In2 is 1 it remains in the same state. We will see at random because it is too involved. Before we wind up for today we will see at random a few of them.

(Refer Slide Time: 49:27)



Let us say S1 state. We will see one more state here, say it is S3 state. In S3 state what happens? Let us have a look. Depending upon the state, that particular light also will go

up. For example Z1 is high here. That is because we are in S1 state. That is what we want. In 1 1 state what happens? See the clock is arriving here. So only at the positive edge of the clock the transition will be made. That is why it is extending here. So this will get reflected only later on. In 1 1 state, let us see what is the input, 1 and 1, both are 1. We are in 1 1 state. Let us see what happens there. (Refer Slide Time: 50:41) We are in 1 1 state and both are 1 so it has to remain in the same 1 1 state. This is 1 1 state so it goes to Z3 only at the rising edge pulse here. What is the input here after this? Again input is changing, 0 0; here it was 1 1 but now it is 0 0. What should happen is in S1 state if its in2 is 0 it should go to S0. We will continue with this in our next class. Thank you

(Refer Slide Time: 52:11)

