

Modeling of Verilog Sequential
Dr. S. Ramachandran
Department of Electrical Engineering
Indian Institute of Technology, Madras

Lecture - 11

Modeling of Verilog Sequential Circuits - Core Statements

Slide – Summary of contents covered in previous lecture.

(Refer Slide Time: 01:10)



(Refer Slide Time: 01:32)



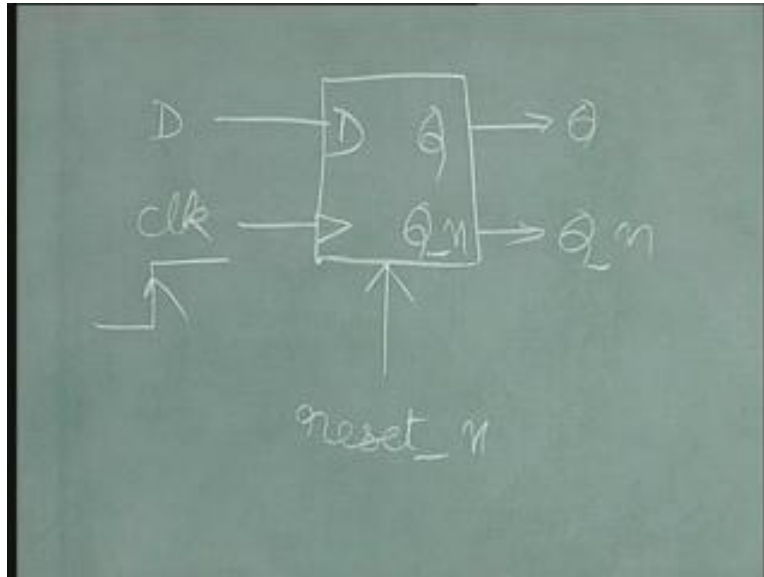
Slide – Summary of contents covered in this lecture.

(Refer Slide Time: 02:00)



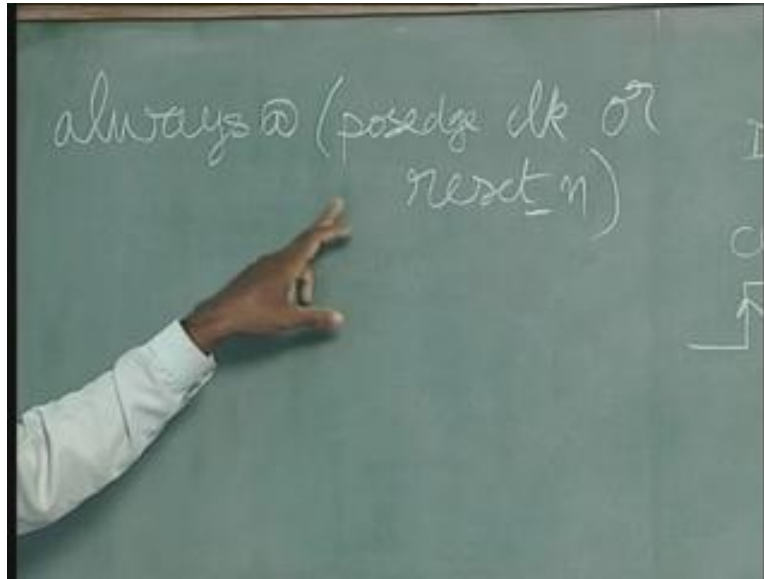
So far we have seen how to model combinational circuits in the previous lecture. In this lecture we will see how to model a sequential circuit. The simplest of flip flop is a D flip flop; we will start with that and grasp how the coding is.

(Refer Slide Time: 03:00)



Further, we will have 1 active low input called the Reset. When reset is applied, the flip flop is cleared. That is, Q becomes 0 negative, n stands for negative but, it is actually regarded as a low. Designers, use this as well as low; this is I think is more popular in that way. The clock input signifies a positive edge clock. Once again, this positive edge of clock, asynchronous reset, active low, these are very popular things among the designers, especially in United States; we adopt this throughout our courses. Now it very easy to quote this; first we will reuse the same 'always' block but connotation is different in this case because this is a sequential circuit, use 'always'. Once again 'at'; this portion is common to combinational as well as sequential but here makes the difference (Refer Slide Time: 04:41).

(Refer Slide Time: 04:32)

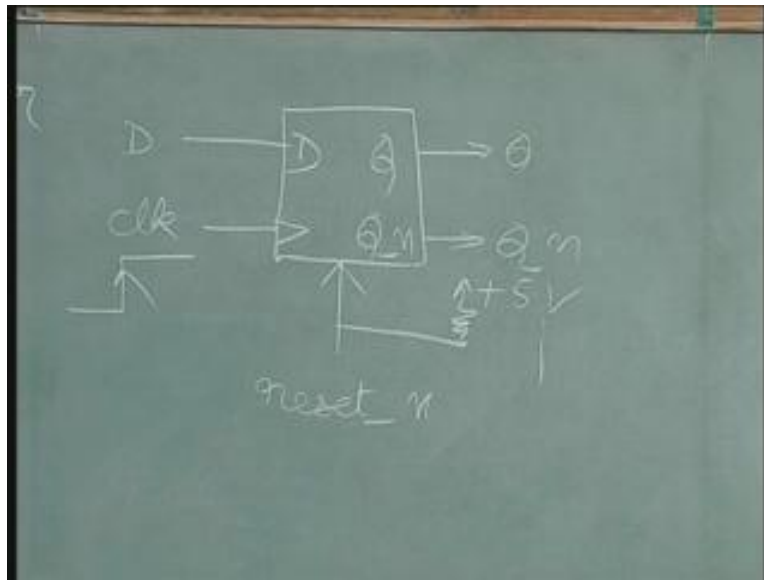


Now within brackets same inputs come into picture but this time it is going to be a clock rather than the ordinary signal logic signals that makes the difference. For example, you want this at positive edge; we use reserved term in verilog called 'posedge'. Note this if you make a spelling mistake, naturally tool will complain. Always give a blank, one or more blanks. One is mandatory otherwise the tool cannot distinguish between this and this. Do not put an underscore or any other symbols, just plain blank. These things are vital because you will have lots of difficulties with the tools later on.

Either at positive edge clock or once again this is not logical, just plain English statement, I can fold it to any number of lines there is no restriction it is Similar thing as we have already mentioned, it is C writing where in you have lots of freedom you can put 1 here 1 there and so on. List the next input that is reset underscore n. We have covered all the inputs. Basically, the inputs pertain only to the clock notice that d is not appearing here. This indicates a sequential circuit. What makes the difference is, here you would have actually put d in the earlier case but it is combinational. Here you have to list the edge transitions namely the clocks; reset is, strictly speaking not a clock but it has a very special meaning; why this reset underscore n, why should we go for low? It normally comes from, if you make a big system this reset will be residing as a master reset on a control panel or a panel you need to reset from that will be connected via long cable hub

in general. Suppose the cable snaps, it should be returned to a safe value; what we do in the circuit is, where the reset push button ends in the PCB you just put a pull-up resistor.

(Refer Slide Time: 07:15)



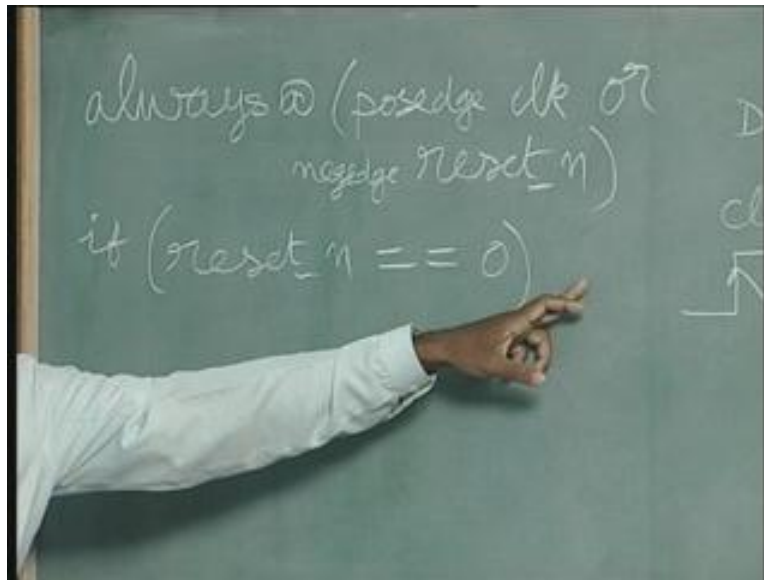
This is plus 5 volts operation, this is it. When a cable snaps it takes to a safe value. That is the reason why it is in industry especially this is the way very important signals has to be given due care.

So is the case with the clock although negative edge clock will also work; somehow positive edge seems to be more popular because that happens to be the first thing to arrive when compared to negative edge. Next we will take up reset condition; first in this case suppose if a reset appears then we have to take some action. I have forgotten one more thing; I said positive edge I have not mentioned edge here (Refer Slide Time: 08:15); this is a wrong statement what we have to do is put a negative edge.

In a sequential relation or flip flop or register, another name usually encountered is a register instead of a flip flop. Here after we may be referring to register which simply means either a “D” flip flop or any other flip flops. Fundamentally, when you see synthesis tool, finally get the circuit out of it for the code that you have written you will be seeing only “D” flip flop in most of the places in fact almost all the places D flip flop will be used. Coming to this we have already accounted for negative edge of reset but

when we write for reset we use a new term called 'if' statement, which is similar to again 'if', 'else', you have used in C.

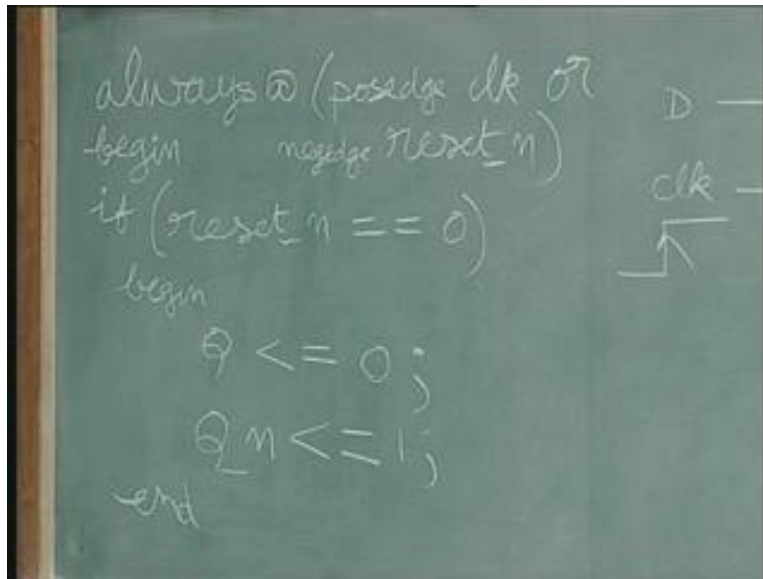
(Refer Slide Time: 09:43)



If you take this if reset underscore n we are going to see whether it is high or low. In this case, it is an active low we have to check for 0. Notice that if there is no comma or semicolon; this is a very important thing. After this condition is satisfied, then it starts executing this following statement; it may be one or more statements. If it is just one statement you do not have to put 'begin' and 'end'; otherwise you have to put 'begin' 'end', if more than one statement.

Now let us see how many statements - I do not know right now. As we go on we will come back put what is required. What we want is, Q is to be reset; that is Q must go to 0; Q underscore n must go to 1. Let us put statement in order to do this one we have only put Q. I will just give a space because we do not know at this stage we need a 'begin' or 'end'. I will just put Q in case, we will put that Q must be equal to 0. Here one statement is complete therefore a semicolon as in C.

(Refer Slide Time: 10:45)

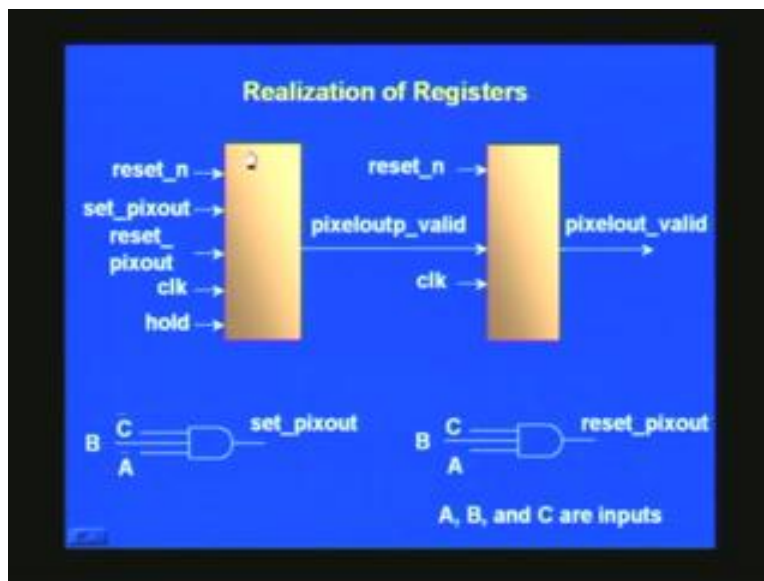


Here what we want is 1 to come there we will just put 1 because it is more than one statement we will put begin end this completes the resetting of the flip flop. Whenever a power on reset occurs or a system is being reset through a manual switch; as we have seen earlier we have forgotten one more thing here that is begin. We should not forget an end corresponding to that it is a good practice to follow indentation. For example, you can put here begin probably can start here if slightly with a try to avoid using the tabs. A good practice will be to use the manual spaces because editor to editor there will be difference. Unfortunately, you may have to migrate from one editor to another editor as you will see different tools later on. It is a good practice but otherwise do not worry much about it; in fact I am violating that by putting tabs I hope u will avoid that. Now what remains to be done is only d input reflecting on the output q. I better write here what we want is q, this must be same as d input q n must be inverse of this d. For inverse you can use either exclamation or this symbol tilda which I have mentioned earlier.

Again you need a begin a block you need to put a begin end the program is over provided you put one more end corresponding to this begin you just compare. How many begins it should be even numbers that should be a matching end for it. Now I am putting it as an indent offset. This completes a flip flop you would notice that it is almost one to one correspondence with what a flip flop does. First you do the resetting only thing is you

have to take the trouble of writing two statements herein fact you can avoid this take care in some other logic which you are going to do later on based on this. You can even dispense with this in which cases begin end can also be dispensed. Now I would like to point one more thing; you have had a new symbol here (Refer Slide Time: 13:55). It is not less; you have to read this together; this is equal to; this distinguishes assign statements; in assign you had used plain equal to or combinational you have to use what is called a non-blocking statement. The earlier one is called blocking statement if you do not put this one, again the tool will complain. I think with this I finished flip flop we will go on to using this as the basic concept. We will now see more and more complex registers. In the first line or vertical line towards the end add one more statement 'else'.

(Refer Slide Time: 14:35)



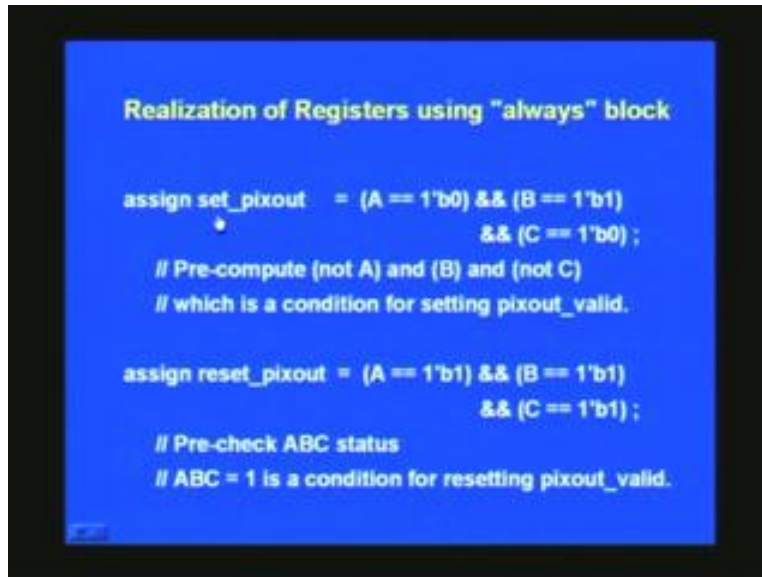
Here what you see in the picture is 2 registers. One register output is called pixel out P. P stands for 'previous', it is a valid signal. In microprocessors, you would have encountered data and its corresponding data value; this is precisely what it is - a valid signal not the data itself. Here, this is one of the designs made for video scaling; we are not going the details of the same; this is just an extract of a portion of it. When it is valid, pixel information is the image information. When it is valid it is indicated here; these are all basically the same signals what we have seen except for few differences. For example, you have used an asynchronous reset here (Refer Slide Time: 15:27), then a clock input.

Once again it is positive logic we will confine our attention to. There is one more signal called 'hold' we will come to that; set and reset are 2 new terms introduced in this. What it does is for example, this is just a register we want to either set or reset; accordingly you have to make a condition.

The condition is if $\bar{A} \bar{B} \bar{C}$ I have just simplified; it actually comes from the actual requirement field requirement. This is in between as I mentioned it is $\bar{A} \bar{B} \bar{C}$ if this is satisfied then this will be naturally 1; then only this flip flop will be set to 1 that is Q will go to 1. Similarly, if Q is to go to 0, we have to assert reset pixel out for that the condition is $A B C$ as depicted in this gate; naturally, $A B C$ are inputs. we have one more flip flop here that also has similar signals. For example, reset positive edge of clock just as we have a P we have a pixel out here actually this is the signal we want and pertains to a previous clock.

For example, in a train of clocks that you have at a particular clock this has become mature it has returned into this but, that is not the point of time that we want to extract this information. A data which is coming through another path normally is delayed because it has to come through several complex processes such as addition, multiplication etc. Naturally, you have got to have mechanisms of delaying. Because the data is delayed, we have to keep pace with it. We need to delay this pixel out valid which happens one clock cycle prior to what we want. That is the reason why we have one more register here this is actually to delay a register the output is pixel out valid for which the quotes are like this (Refer Slide Time: 17:45).

(Refer Slide Time: 17:45)



For the first flip flop we have set pixel output. We have already seen that it must A bar B C bar - this you are already familiar. 1 stands for, it is just a number of bits is what it signifies here together with this apostrophe. B stands for binary; that means what follows this number that is this 0 that is in binary. Suppose you want to have it in decimal what you need to do is just replace this by d, suppose you have 8 bits, you do not have to write 8 times 0s; just one 0 is enough it will take care. Suppose you want an octal representation replace d by o; you want to have hexadecimal replace it by h on.

Any bit precision you can use the same way note that two equals are there. That means, logic you could have straight away put A B C, that would have served the purpose; here this is again put deliberately not explained the alternate way. It is not the alternate way because, when we are going through hundreds of or even thousands of statements, we may not be in a position to go all the way back find out what the bit precision of a particular signal is.

It always a good practice to indicate what it does. Another important thing is, when you write a statement, it is a good practice to have good comments.

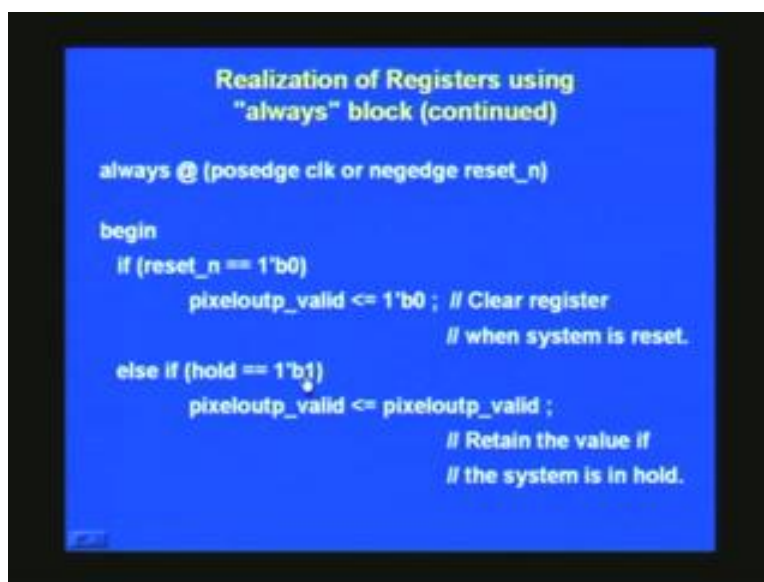
What is obvious in the statement do not state it, because it is already stating what you want to. Put it in plain English like language. For example, even this may not be that

plain; elsewhere you may see more plain language. Pre compute is a condition for setting this. All detailed explanation you can have. Suppose the explanation that you have offered is not adequate, you can use slash star followed by another star slash as in C; put any number of statements inside that; we have also seen reset pixel out here using assign statement.

Here (Refer Slide Time: 20:22) we put into what we have already seen here. The difference is A equal to 1, B equal to 1, C equal to 1, implying that it is A B C which we are handling together. Notice that in logical statement we need to put double AND. Of course, in assign statements even if you put just 1 AND it still accepts. But as a normal practice, we should use this because, in sequential circuits we can put the logic inside the always block there if you do not validate this 1, just put 1, the tool will complain.

Now we have seen that in the flip flop we have covered, an 'always', at positive edge of clock or negative edge of reset underscore n; there is 'begin' once again very same statement occurs here also, if reset is 0 here you could have put just 0 alone. When you do not put any other thing it implies that it is a decimal number that also is valid because, it is a single bit. If the condition is met you make that register output 0 that is clear register when system is reset.

(Refer Slide Time: 21:00)



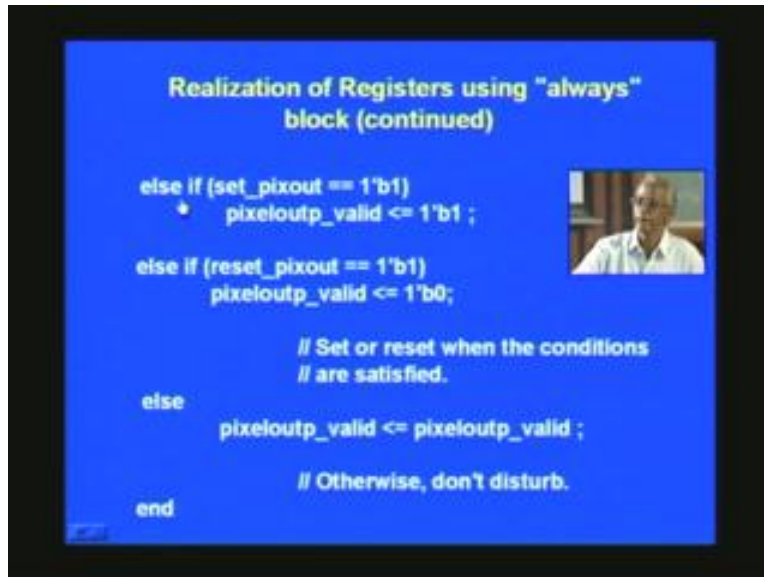
```
Realization of Registers using
"always" block (continued)

always @ (posedge clk or negedge reset_n)
begin
  if (reset_n == 1'b0)
    pixeloutp_valid <= 1'b0 ; // Clear register
                                // when system is reset.
  else if (hold == 1'b1)
    pixeloutp_valid <= pixeloutp_valid ;
                                // Retain the value if
                                // the system is in hold.
```

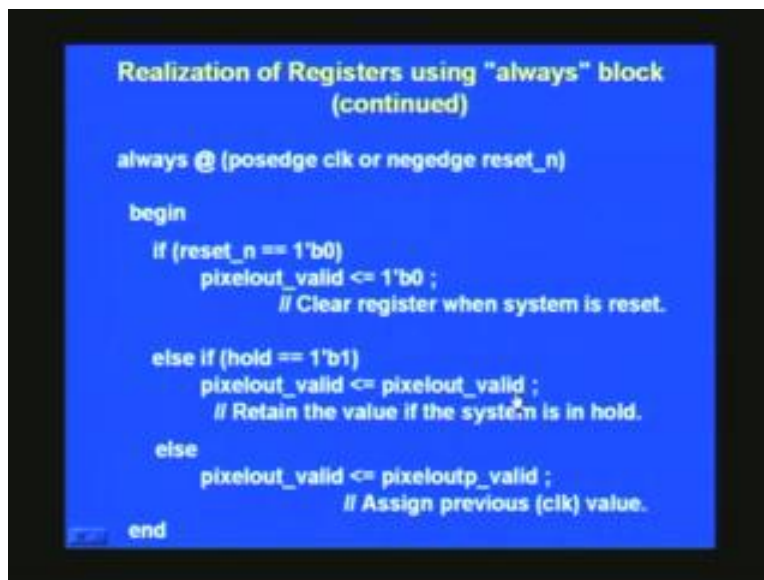
As I mentioned, you can write in plain language like this, do not repeat what is obvious here. If this condition is satisfied do this otherwise what should you do the otherwise condition is specified by 'else if' in this case; we have also seen another control here called hold. Whenever you have a system you may be processing at a very rapid phase there may be other host systems receiving this data. If the host is not as fast as your system, naturally, the host will have to hold you for the time being. Unless you have a hold pin there it cannot hold; that is the provision made for that. If hold is 1, it is asserted then what we have to do is just hold the previous value the pixel out P valid is again same here that means you are writing back its own value; in other words you are not disturbing the value. We have to do something at every point of time; you cannot remain without doing. You have to retain the value if the system is in hold. Or you can try; just remove this statement, find out what the tool reports.

If those two conditions are satisfied we are yet to cover set then reset that is, what we are doing here. If a set pixel out which was $A \bar{B} C \bar{D}$, if it is 1, we need to set this pixel out p as 1; if reset is asserted then we need to reset it. That is what we have put the comments as if none of these are satisfied then you just use one else because this indicates that it is the last in that category, 'else if'. In this case, again preserve the same value that, do not disturb the content. You have seen that 'if else', 'else if' and finally, 'else' structure; if you just count 1 2 3 here 4 5 from my experience, if you exit this file your speed of operation will get kicked. For example, if you are aiming at 100 mega hertz operation on a FPGA field, program gets arrested. Only by experience you get this; do not try to add more than these statements. If you are still forced to add this statement, combine this outside the 'always' block; then restrict this once again to 4 or 5. Smaller the number better it is, higher the speed of operation you get.

(Refer Slide Time: 23:20)



(Refer Slide Time: 25:50)



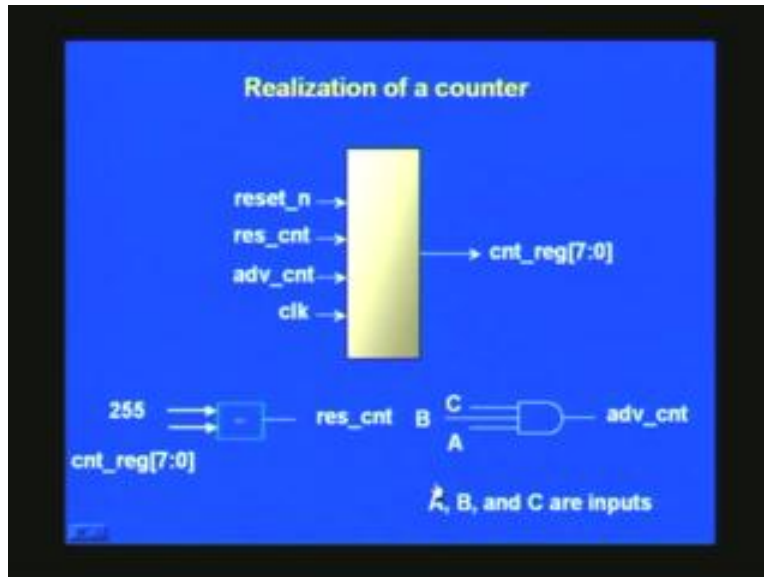
With this we have completed the first flip flop. With all the signals accounted for, this is the output we have this will be connected to the next in a pipeline; but the pipelining is not the aim here which we will be covered later on. This is only to delay the signal as we had mentioned earlier as to keep pace with another signal such as data actual data itself pixel out which is the data. We will now see this flip flop implementation which is very much similar to this. Once again we used 'always' at positive edge of clock or negative

edge of reset you noticed that when you go through the course you will get again and again very same structure all through; that is the beauty of this structure. Especially when you want to redesign something or even start on a new design, what you need to do is just save your earlier projects file simply rename to suit the new design that way it helps. Second thing is you are not bogged down by many signals.

If you do not stick onto this, what I have observed over the years, number of designers putting hundreds of variables in one 'always' block. The result will be chaos; you will get drowned in the myriads of the signals so avoid that one. You can put any number of 'always' you can rest assured that each of this always blocks will be functioning concurrently there is no need for you to put all together not make the system work. That will be a frustrating experience, especially for a beginner. Therefore, even for a beginner this is highly recommended here.

Once again what we have seen is when reset is asserted this pixel out is the actual signal we want earlier it was pixel out P indicating the prior signal for this is the actual signal we wanted - delayed signal. But the actual signal was already pre-computed in the earlier cycle now once again we reset because when reset is encountered we have to reset. That is what we have put here; if hold is there just hold on to the same value if not we do not you just notice that we have used only 3 statements, 'if else' in this case. The final output is pixel out valid; whatever is the input which happens to be the previous value, here in the diagram (Refer Slide Time: 28:00), that you have already seen. Note input is this P output is without the P.

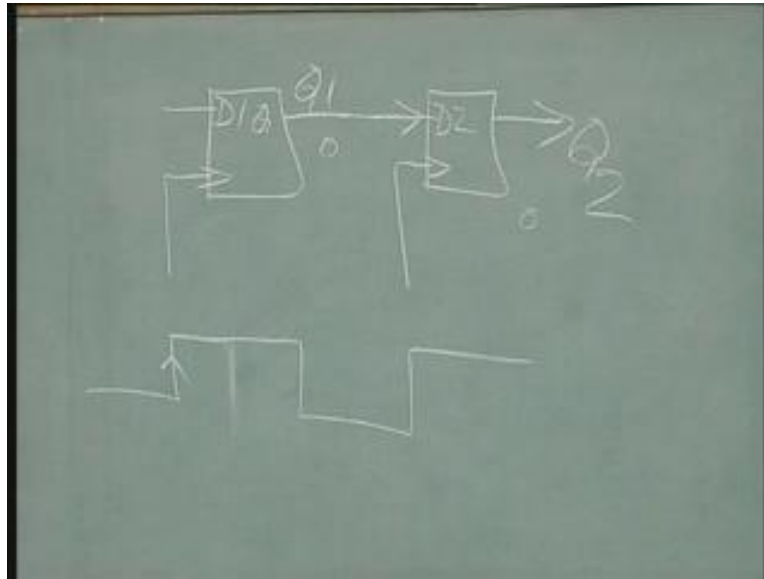
(Refer Slide Time: 27:52)



That is what we assign here; present value is assigned with the previous value. Note that this will be valid only at the following clock edge positive edge, the previous positive edge of the clock. This would have been set that gets reflected only in the next clock that is here. Earlier thing whatever was it keeps on just delaying the access signal; how do distinguish between present clock and next clock/ it is actually by the signal. This signal is derived from the previous signal the previous signal is set only in the previous clock; this is going to be set now which happens to be the previous status only in the present clock.

I will draw a small wave form; let us take the first flip flop we are going to. As per the 'always' block we have whenever a positive edge clock occurs whatever this D_1 value here it will get into this Q_1 suppose you have cascaded here D_2 receives this Q_1 , out comes Q_2 .

(Refer Slide Time: 30:17)



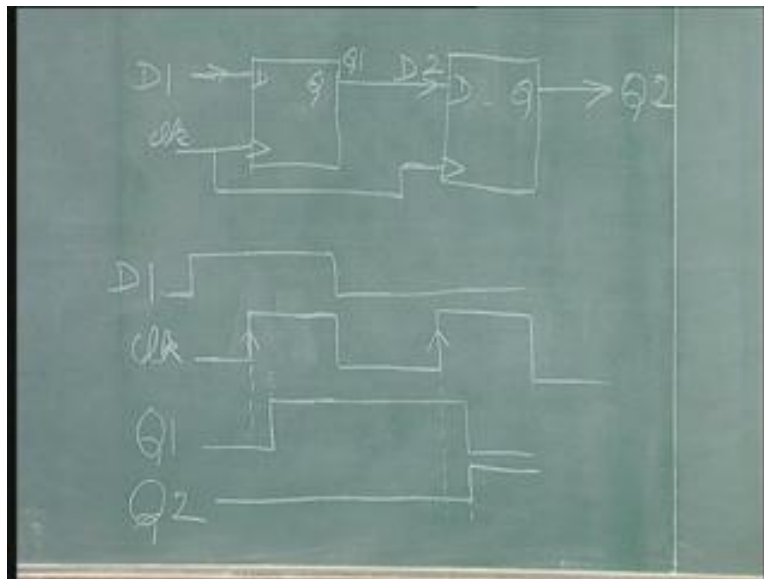
At this positive edge for that matter both flip flops will act whenever a positive edge clock occurs both will keep on acting. To start with, these both are 0s, let us for the time being that is a reasonable thing because we started with a presumption that reset is applied always the system is initialized. Both being 0, what is happening? Suppose there are trains of clock pulses coming earlier when you see, here whenever a clock strikes whatever is there here, it will go through. For example, this was 0 to start with; we said A B C which is actually not the actual field inputs it may be an intermediate outputs which is fed back maybe it is from a counter.

The whole thing is synchronous; wherever you have flip flops all of them will be reset. It is proper for us to assume even A B C condition is all reset for time being. So whatever be this input here at this clock also this flip flop it will come over here (Refer Slide Time: 31:15). For example, there was 0, the 0 would have come right at this point immediately after this. If you implement on the chip there will be a gate delay only here the data will be affected; but for simplicity we will just take here at this edge what you have here is.

This Q_1 being valid D_1 hence Q_1 now this forms the input for this when this was already 0 when the clock edge was encountered here it was also sensed here but it had seen a 0. Therefore it would have put a Q_2 also 0. Notice that because it was Q_1 is 0 it is going to

change state only a little later not here. Although we applied this here, it is going to change only after a delay that means to although we put the output right here it is not here actually it is slightly with delay. Therefore, when the clock struck this is going to sense 0 of Q_1 not 1, it will be sensing 1 only in this clock. This Q_2 will go only in this hence we have got a delay whatever is the first flip flop that only is being fed here that will take effect only with the arrival of the next clock pulse right although it is all synchronous circuit you have to have a system clock running all through. We will see more when we see a partial coding. Coming back to this, we completed wave forms for the D flip flops cascaded as shown.

(Refer Slide Time: 32:54)



I think the program is clear, self explanatory. Right now, before we go on to the next example we will just see what this statement means. **Conversation with students (Refer Slide Time: 33:15)**

Before we wind up this example, we will see what it implies, if statement what it puts is a MUX in hardware all of them are all put MUX; that means 1 MUX output cascades with another MUX and so on. That is the reason why I said you should restrict this to 4 or 5 based upon our experience as well, as that would ultimately decide what frequency of operation that you get that is an important thing you have to keep in mind. We will go on

to the next example. This is to realize a counter this is also based upon a flip flop, a register once again you have a reset here. It is same as previous reset asynchronous reset then we have in addition to this. Suppose you want to have this counter; this counter is lets an 8 bit counter - seven through 0; this counter output is named as cnt_reg.

Reg means to tell you as a ready reckoner, it is a register which means a flip flop this is 8 bit flip flop. That is what is implied here we want to reset this counter when the counter value has gone to a particular value let us say 255 then there will be another comparison circuit here another signal will be produced, we call it reset counter applied here; then once you counter you need to have another signal upon I mean to start the count its rather an enable of the count. That is what is implied here although it is advance count it is not a clock. The counter will be counting on the positive edge of the clock. Every time a positive edge of the clock is encountered, this will be counted this as the output. We have one more condition here A B C as the enable for the counter once again, A B C are inputs.

(Refer Slide Time: 36:20)

```
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.

assign cnt_next = cnt_reg + 1 ;
    // 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.
```

We have seen that reset counter is when counter register is equal to 255 once again we notice that 2 'equal to' be used here being a logical statement. It does the comparison if it is true then 1 is output; that 1 only is assigned to this reset count. It is not the 255 that is

coming over to this. This is only single bit the result is single bit once again which is assigned here the actual is all based on the counter value that is what we have written here as the comment (Refer Slide Time: 37:00). Another thing is, this is a line comment if I had not mentioned it earlier similar to reset count. We need to advance the count that is enabling the counting operation we put another assign statement this time it is A B C which we have already seen earlier. Another thing is noticed, that all these things are combinational circuit, this is appearing beyond this 'always' block which is for the counter.

This is the sequential circuit; prior to this one we need to do the incrementing of the counter whenever this advance count is asserted at the positive edge of the clock. In order to do this, we can do the incrementing right within this 'always' block; but, a better thing would be to pre compute it so that you do not waste time; when the clock strikes you start computing. You pre compute it be ready with it when the clock strikes you transfer the result. That would be a good design approach and that will once again speed-up of the operation of the system. What we need to do is we just increment counter register that is the output of the counter which we have already seen, just increment by 1 there is note that there is no for one stroke B and so on. It can be plain decimal number; the incremental value is assigned as the next value for the counter only this is the pre increment. Actually if it increments beforehand the incremental value will be assigned or not assigned depend upon this inside the always block.

Once again, we see begin final end an always block at positive edge of clock or negative edge of reset n these are all the standard things. As I said, it will keep on recurring later on. If a reset is this also you have already seen only difference is counter reg which is an 8 bit is reset to 0 value. You can even dispense with this 8 stroke D if you put 0 that also will work it will be easier. But good practice is to put this so that you will not get a doubt whether it is 8 bit or 12 bit precision later on it is a good practice to have this which naturally initializes the system.

(Refer Slide Time: 39:39)

```
Realization of a counter using "always" block
(continued)

else if (res_cnt == 1'b1)
    // Reset if terminal count is reached.
    cnt_reg <= 8'd0 ;

else if (adv_cnt == 1'b1) // If enabled,
    cnt_reg <= cnt_next ; // advance the
                          // counter once.

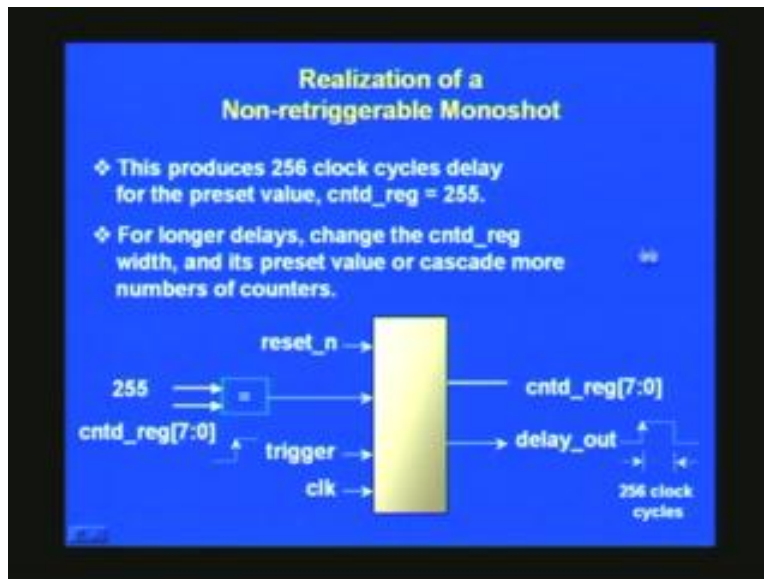
else
    cnt_reg <= cnt_reg ; // Otherwise, do not
                          // disturb.

end
```

Next what we need to do is when the counter was 25 we have already seen here. When counter is 255 then reset that is what we are going to do here. If it is one then reset the counter this counter output receives 0 signal here. Note that this is only number of bit precision that there in this number d stands for the decimal number. It is meaningless you can put any other thing you want or just dispense with it you want to. We have seen all one if then else if then one more else if else once again we see that it does not exist 4 or 5, if it exists we will consider another example also later on. We need to advance if it is 1 we have already pre-incremented earlier using the assign statement.

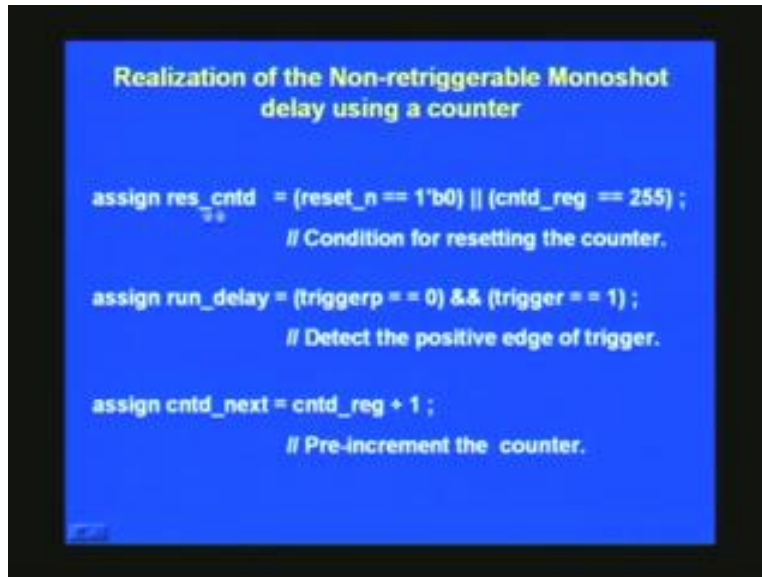
Now the clock will come only after delay because, clock strikes periodically at one particular point at that point when it does whatever was the pre incremented value it will be merely pushed into this counters. Another alternative way was to dispense with that assign statement there. Thereby remove one signal we are unnecessarily introducing one signal but that with the aim of increasing the speed of processing that is a good habit here we could have put the same counter edge plus one right here that also will work. But what is happening only when the clock strikes it starts computing right with this if this is not satisfied then we will not disturb the contents of the counter edge.

(Refer Slide Time: 42:25)



Next we will consider how to realize a monoshot it is a non-triggerable. With a decided delay of 2 digits clock cycle for this you need to reset. I mean set the value of 255 always bear in mind the delay you get is 1 clock cycle more than what you set if you want longer delays you can either cascade more number of counters or increase the bit precision of this counter. You have ample choice on that score. Once again you have a synchronous reset when the counter,... now this time I have put d because we are going to put together what all we have learnt all this is in a single program and then execute. If you just give the same name it will be overwritten deliberately putting different name for that here d stands for that this is 8 bit there is a trigger applied here. This is a non-retriggerable that means, once you have triggered the delay starts any amount of further triggering will not really have any effect. That is the idea here and there is naturally a clock; let us see how to code it.

(Refer Slide Time: 43:04)



Once again we have an assign statement for resetting that is also a register. Here we have combined earlier I said we should avoid 'else', 'if else', statements not to cross 5 here probably it crossed by 1. I have combined two such similar statements right outside here (Refer Slide Time: 43:22). Notice that there was a reset inside asynchronous reset when it is 0 then counter value also does the same thing of resetting the counter itself. Why not combine both in a single statement this is or that is doing here this is a logical realization each of them will return a 1 if the condition is satisfied. For example, counter 2 is to be defined then it returns a 1 or this will influence this reset signal as it this we have noticed that I mean input is trigger we want to sense the raising edge of the trigger. How do we sense? One way is to put always in positive edge clock there, you cannot have more number of clocks may be any system you need to restrict it to 2 or 3 clocks. Do not cross that number. What do we do if we have many other signals which also need to be sensed at the raising edge for that matter, falling edge? Then what you can do is we can have another signal which will store the previous value. Every time the clock strikes the current value you can push it into another flip flop then store it using that one in the next clock. You will know what is available in the previous clock that status here by ending this one for example previous value is 0 present value is one naturally you can sense that it has raising edge as far as this signal is concerned; this is once again pre incrementing the register.

(Refer Slide Time: 45:08)

```
always @ (posedge clk or negedge reset_n)
begin
    if (res_cntd == 1)
        // Initialize when the system is reset
        // or if the terminal count is reached.
        begin
            cntd_reg <= 8'd0 ;
            delay_out <= 0 ;
            triggerp <= 0 ;
        end
end
```

There is one more 'always' block; in this case we take a decision based upon that reset here it happens to be too blended together, that is system reset as well as the counter reaching 255; that is precisely the same here. If it is 1 as I said earlier there seems to be a mistake here (Refer Slide Time: 45:35), if there are mistakes it should have been 0 now let us cross check that. This one when it is 0, what it returns is 1. Similarly, here also this must be 1 only. If it is 1, then only signals should be initialized. For example, this counter output itself counter is although it is timer monoshot vibrator we have designed it as a counter; this is reset, here it is output. We have already seen as a delay out, that is also reset. There is one more signal that we set, we will have a previous status of the trigger which is also a flip flop. That also needs to be reset right at the beginning this being a block we have a 'begin' and 'end'.

(Refer Slide Time: 46:40)

```
Realization of the Non-retriggerable Monoshot
delay using a counter (continued)

else if (delay_out = = 1)
begin
    cntd_reg <= cntd_next ; // Advance the count
                           // by one if the timer
                           // is still running.
    triggerp <= trigger ;
end

else if (run_delay = = 1)
begin
    delay_out <= 1 ; // Start the delay if the positive
                   // edge of trigger is detected.
    triggerp <= trigger ;
                // Preserve the current state of trigger.
end
```

If we have another 'else if' statement let us if first we sense the output if delay is not yet 1 it will be 1 later on. But anyway we put this as the priority condition if this happens. If it is 1 what we do is we need to increment the counter because this implies the once the output has become 1, it means, the counter has started running. As long as it is running that long we can keep on incrementing the counter, we also should not forget to put this current trigger value as a previous value as to use it in the next clock, when it strikes this is previous status of trigger registered. We are remembering what the trigger previous status is by pushing it in current value into the previous value; that is what we have put here. Now, if this run delay which was the actual sensing of the raising edge, if that is 1 whenever this happens this is going to happen only once in a way if it happens what should you do, delay out must be set to 1. We have seen that in 'always' block all this are non blocking statements that you should not forget to push this current value into the previous value here (Refer Slide Time: 48:03).

(Refer Slide Time: 48:20)

```
Realization of the Non-retriggerable Monoshot
delay using a counter (continued)

else
  • begin
      cntd_reg <= cntd_reg ;      // Otherwise, don't
                                // disturb.
      delay_out <= delay_out ;
      triggerp <= trigger ;
  end
end
```

If that is not satisfied finally, you have one more block the last 'else if' will have only 'else' no 'if' here. If this happens do not disturb any of the contents we have used 3 flip flops or registers there is counter d, delay out-trigger each of them is 1 bit in size with the exception of counter d which is 8 bits. We have seen this delay out as 256 clock cycles that is what we have by equating it to 255, why 256? When we see the stimulation we will have explanation for one extra cycle. We have said earlier that MUX are put cascading each other. We will see one more point before we wind up today's lecture that is what this implies - it is a priority encoder.

We have 'if' statement occurring very first. If this condition is satisfied, then it will execute this process then quit the whole thing. It goes out of the 'always' block; no other statements occurring later on will be entertained. Similarly, if this is not satisfied only it will go to the next one. This in turn if it is satisfied it will go to this, once again come out of the 'always' block having executed the first one or it did not execute the first one. That is why it has come to this one having process this it quits this 'always' block. In the case for every other thing, now this implies it is a priority encoder; top priorities always put first this one because naturally the reset which is very important we had to take a decision not give priority to the advancing of the counter itself. That is why this is a priority encoder. Wherever you have priority encoded to be implemented such as in bus

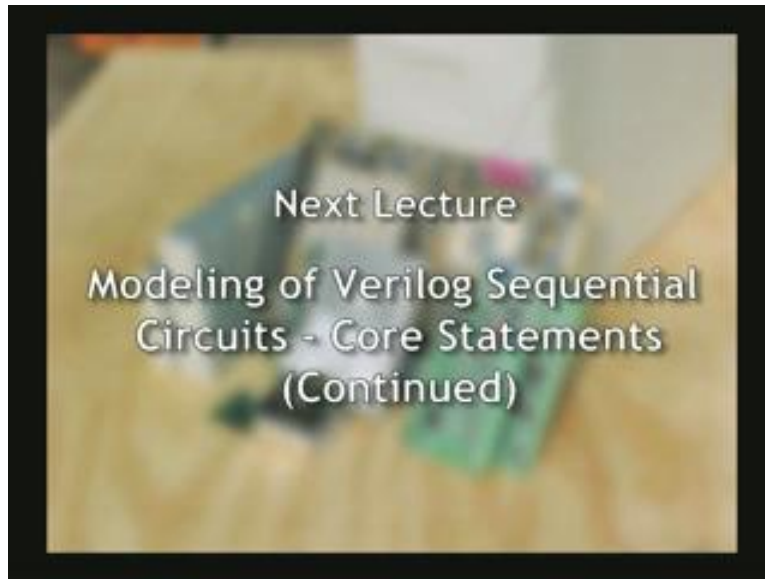
arbitration in multiprocessor systems etc and then you can use this way of coding. So far we have seen how to write a Verilog code for a simple D flip flop then go on with little more complexity introduced by adding cascading 2 registers then we spoke of non-retriggerable monoshot. Perhaps you can take as an exercise, how to do a retriggerable mono. We can amend the code for the existing ones we will continue with the sequential circuits in our next lecture.

Summary of Lecture 11

(Refer Slide Time: 51:50)



(Refer Slide Time: 52:14)



(Refer Slide Time: 52:19)

