# Digital VLSI System Design

**Prof. Dr. S. Ramachandran**
**Department of Electrical Engineering**
**Indian Institute of Technology, Madras**

## Lecture – 43

## System Design Examples

Contents of Lecture 43

(Refer Slide Time: 01:42)

We will consider couple of design examples as we go along, we will see why we take a particular example and how to solve using verilog code.

To start with, we will look into a traffic light controller. This is the result of observation of one of the busiest traffic light junctions in Chennai.

(Refer Slide Time: 02:18)

I have been passing through and observing that junction and for quite a long time, the result of which is what I have designed here.

(Refer Slide Time: 02:44)

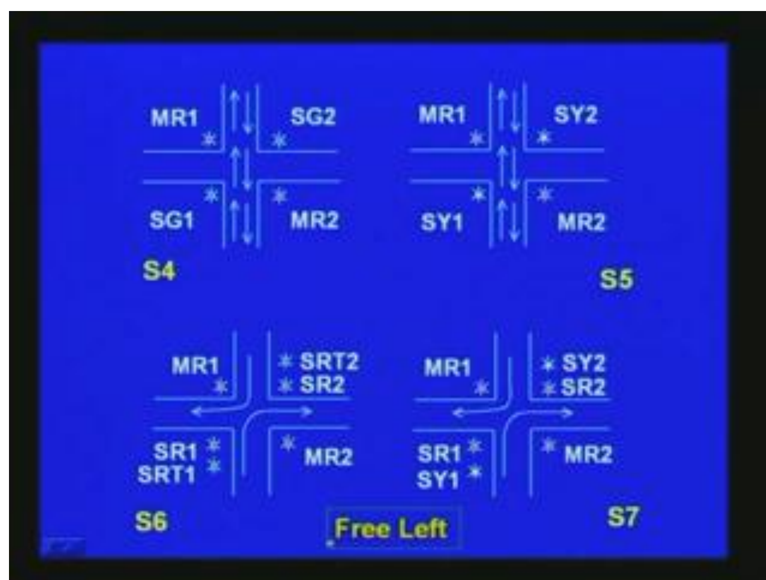
This is the traffic flow at that particular junction. There is a main road here and traffic flows in both directions. The very first sequence is there will be a green light here for both directions. There will be a red light preventing this traffic flow here. This is the main road and we have a time of 45 seconds. This is exactly what it is in that particular junction. After 45 seconds elapse, it will go yellow here for both the main, which means that green is withdrawn. That is the second sequence. In the third

sequence, you have a right turn here both ways here. There will be a red signal preventing this straight traffic flow and so is the case for the other direction. Naturally, there will be a side red in order to prevent this traffic and once again this will turn into yellow. That is what it is here. Perhaps they have made a mistake or I have made a mistake in one of these. This red should probably continue here. I will leave it as an exercise for you to change the code accordingly. That is the simple exercise.

All through, it is all free left and that is why we have put it here. There are further sequences. This is the fifth sequence here. This will go red preventing traffic here and it will allow the side road traffic to come on. In the earlier thing, I have forgotten to mention that these yellow lights will be on here, so also all other yellow lights, only for 5 seconds.

Once this is for 25 seconds flow and, once again, it will go yellow here and then switch over to the right traffic from this direction. Right turn will be allowed from the side roads in this case. So the nomenclature adopted here is M for main, then R for red, G for green, then you have Y for yellow, and then RT stands for right, and S having the same meaning as a side.

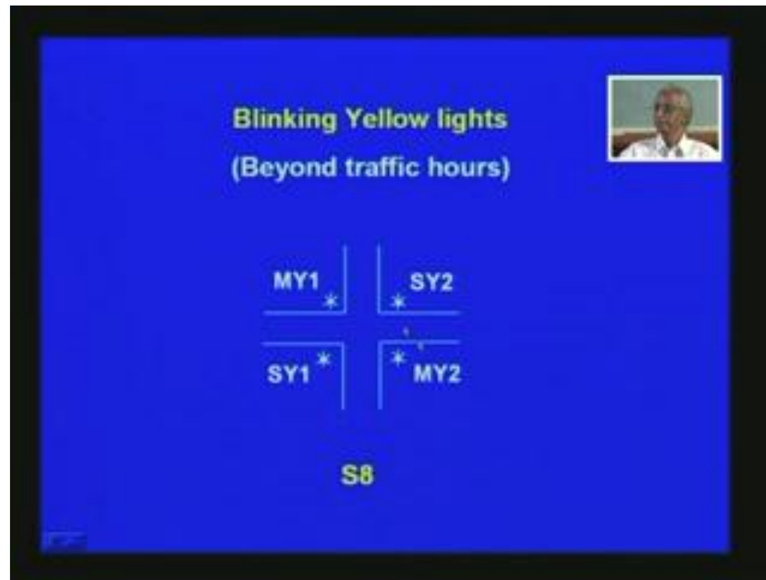
(Refer Slide Time: 04:15)




Once this is allowed, this also will flow for 25 seconds and, once again, it will go through a yellow light, naturally withdrawing the side right. This is a green arrow

here. So was the case for the main MRT1 and MRT2 earlier and so on. After the sequence, it will go back to S0 sequence and it keeps flowing in the sequence perpetually till the cop turns it into blink mode, which is the case in this particular junction. So I have observed it at around 9:30 or 10 in the night and it goes blinking. All yellow lights blink here.

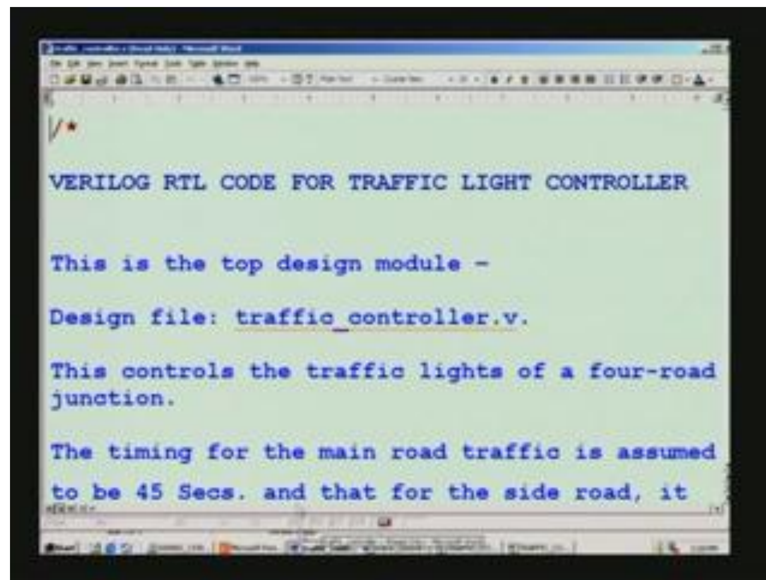
(Refer Slide Time: 05:59)




This is only to caution the traffic that they will have to be careful, exercise restraint, and go carefully. This is what we will look into.

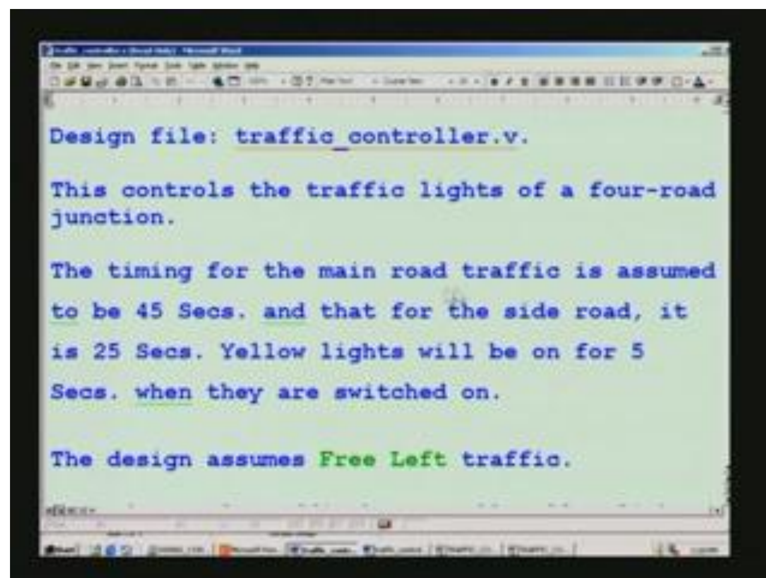
First we will see the code for this. This is the design. We will see the verilog code for this design. It is RTL compliant as I have been emphasizing all through the course and the top design module is traffic underscore controller dot v, some of which you have already seen.

(Refer Slide Time: 06:30)




These specifications control the traffic of a four road junction. The timing for the main road traffic is assumed to be 45 seconds, and that for the side road is 25 seconds. Yellow lights will be on for 5 seconds when they are switched on.

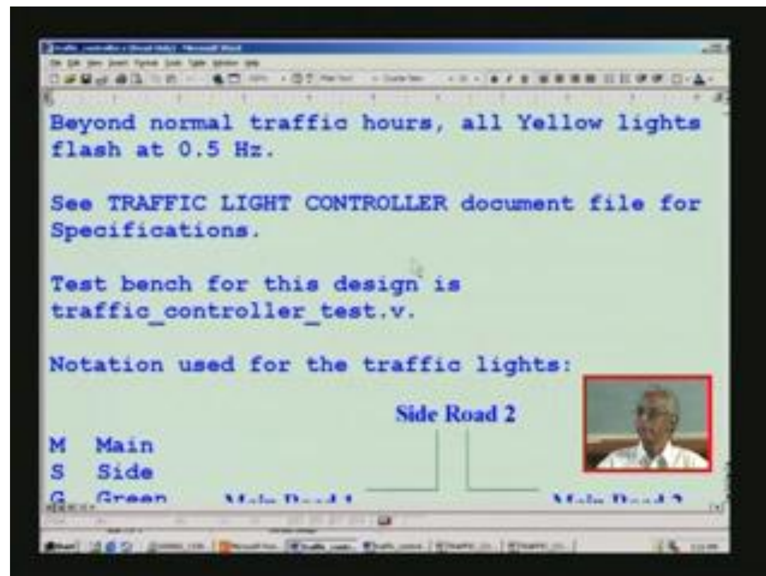
(Refer Slide Time: 06:51)




It is free left traffic. We have also seen yellow lights flashing when a blink control is switched on. This is for beyond normal traffic hours and it will flash at the rate of 0.5 hertz. There is sanctity for this. The actual traffic junction may actually have slightly faster thing. I have purposely turned it down so that lamp light will be better. You have to see the traffic controller document we should always generate a document that

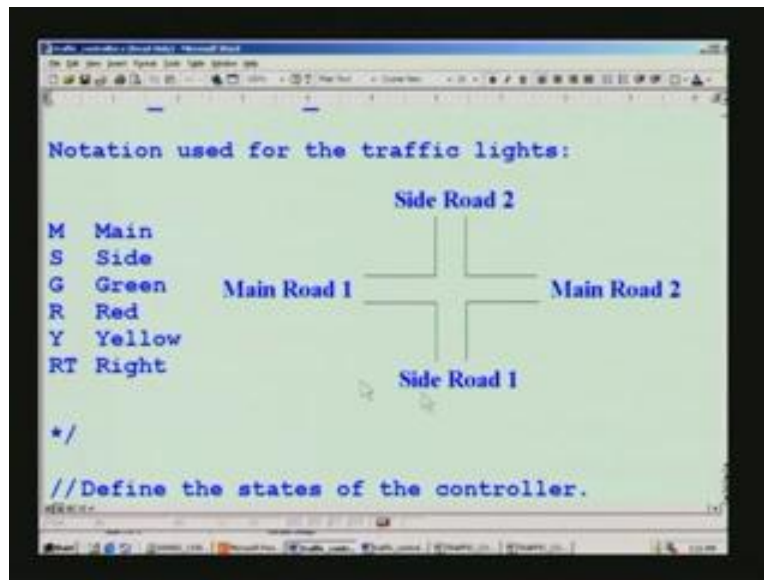
gives the specification instead of the document we saw in the power point presentation earlier and that may be regarded as the specification document. Test bench for this design goes by the same name with an extension of test. This is my style of coding and naming the files.

(Refer Slide Time: 07:31)



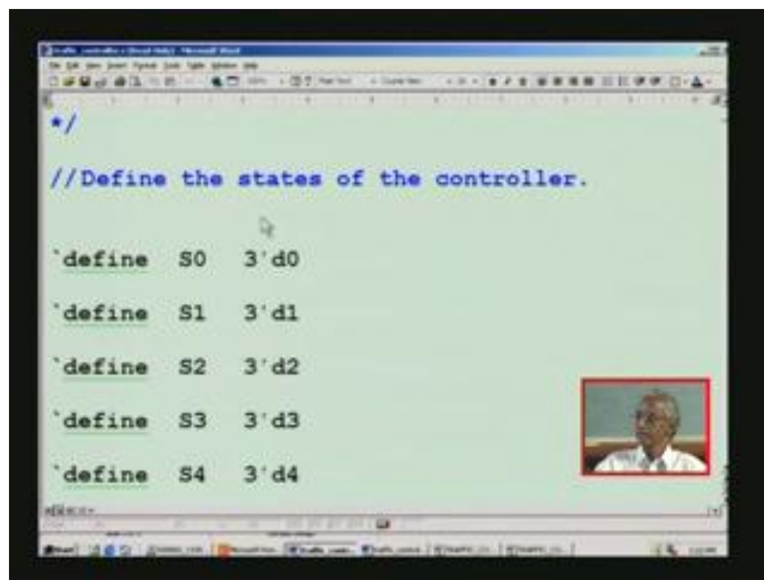

We have also seen this nomenclature. This is the notation used. Main, sides, green, red, yellow, and right are all abbreviated in this fashion right as well. This is the main road. Here it is called Anna Salai, and this is Venkat Narayanan Road or some such road so it is called Nandanam circle. I have, as far as possible, put the very same sequence here. This part of the road going to this side is the main road. It goes towards the airport this way and towards central station the other way. So this is main road 2 and we have side road 1 and side road 2. It is only a nomenclature; it is actually a single road. For the sake of convenience, we put numbers here. From that, when we come to the actual lamps, we will identify in terms of these numbers.

(Refer Slide Time: 08:02)



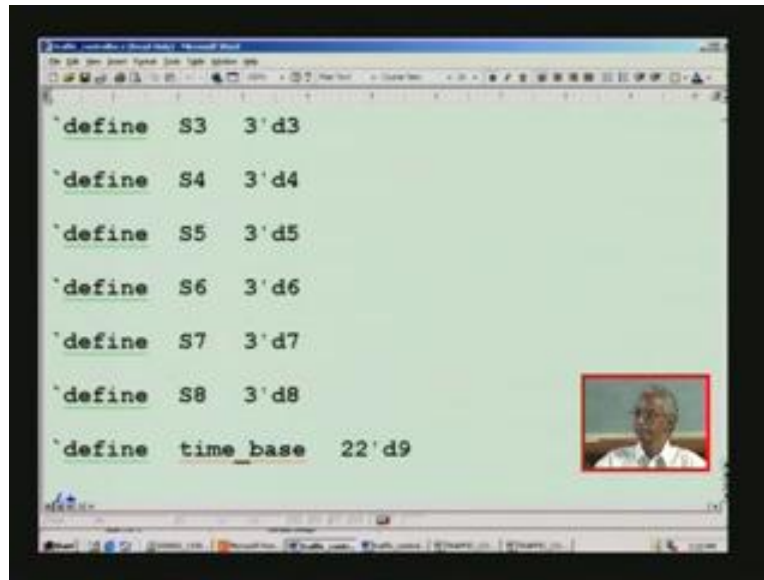

We have seen this is line comment and we had an fsm state machine realization. Each of the states is indicated by S0 through S8 and its decimal weight, you can say, is 0 through 8.

(Refer Slide Time: 09:00)



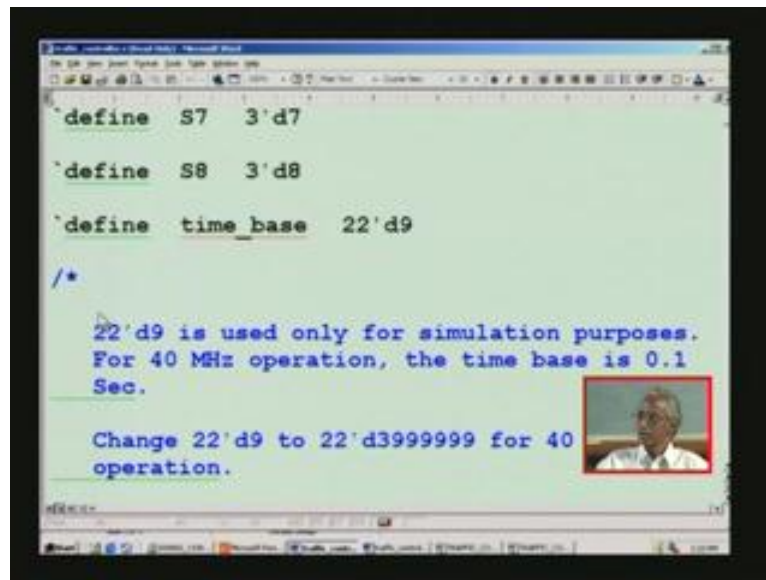

These are all the define statements that you have.

(Refer Slide Time: 9:18)



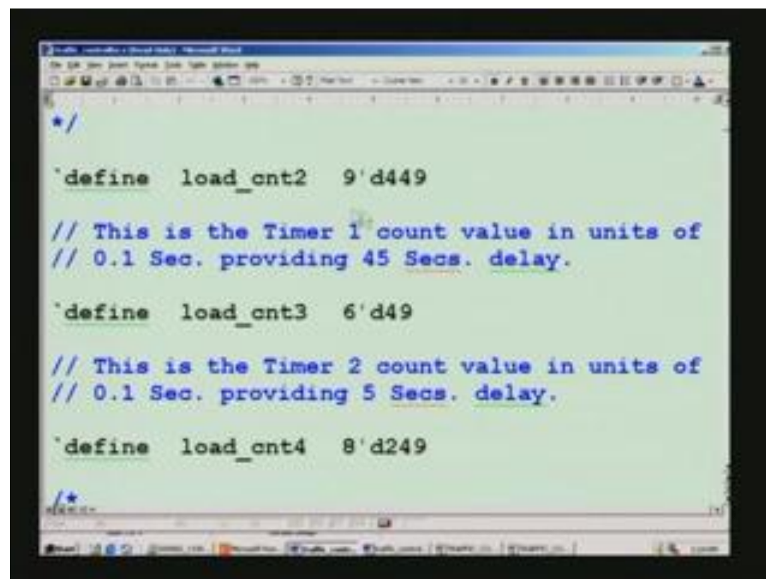

Note that we need 45 seconds, 25 seconds, 5 seconds as well as 1 second timer, 1 or 2 seconds timer. So you need a counter for realizing the same, and we will go into the design of timers as well. What we need for that is a time base. For any accumulating of count, if you are to interpret in terms of time, we should have a time base. Let us have a convenient time base, could be 0.1 second. You can trigger other counters based upon this and run to the precise 45 seconds etc. In this case, this time basis program for 9. This count takes place at 0, 1 2 right up to 9. This means that you have totally 10 clock cycles influencing this counter. So is the case with all the other counters. This 22d 9 is used only for simulation purposes, as I have mentioned here, because if we are to configure this with a frequency of operation at 40 megahertz, this is the clock frequency system clock. Then what we need is quite a big number to be loaded in this particular time base. This particular counter is counter 1and we need a very high value to be loaded. For example, this is equivalent to 4 million and 4 million and 40 megahertz. This means that 10 hertz will be the basic counter here that is actually 0.1 second in terms of time period.

(Refer Slide Time: 09:20)




That is how you generate a time base, which is 0.1 second.

(Refer Slide Time: 11:04)




Taking this as the basis for other counters, we can easily build 45 seconds counter by running count value of 449, always 1 less here. Earlier, in another type of timer we have considered, that is in re-triggerable mono shot we had considered, whatever we programmed it gave the very same time. In this case it is slightly different, because 0 is also reckoned here so 0 through 9. We also need another counter for keeping track of 5 seconds. That is what this is doing here. Similarly, this is the case for 25 seconds. If you take 250 into 0.1, you get 25 seconds. That is the meaning of this.

(Refer Slide Time: 11:43)

```
// This is the Timer 2 count value in units of
// 0.1 Sec. providing 5 Secs. delay.

`define  load_cnt4   8'd249

/*
   This is the Timer 3 count value in units of
   0.1 Sec. providing 25 Secs. delay.
*/

`define  load_cnt5   8'd9

/*
   This is the Timer 4 count value in units of
```

There is 1 more counter here. This is the counter that will do the flashing so another timer is required for that. We can probably reuse some of these counters. I did not reuse just to make it quite a simple design.

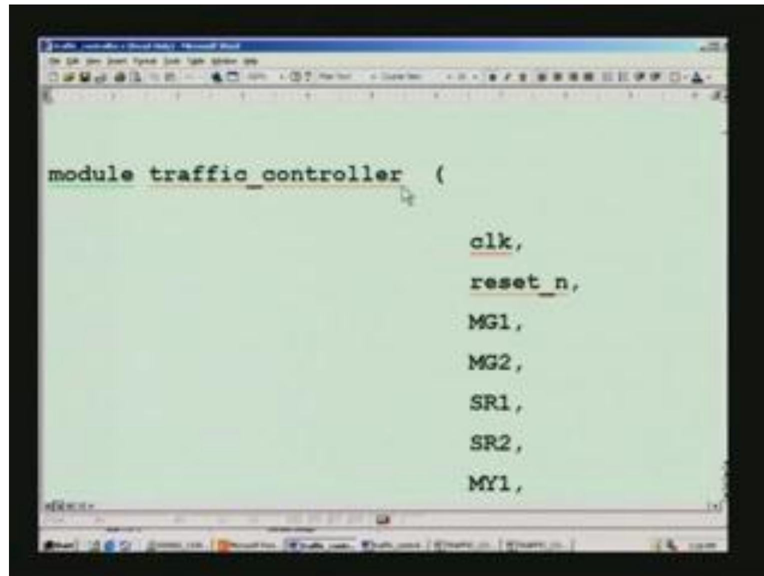
(Refer Slide Time: 11:50)

```
`define  load_cnt5   8'd9

/*
   This is the Timer 4 count value in units of
   0.1 Sec. providing 1 Sec. delay.

   Change these if you desire different
   timings.
*/

module traffic_controller  (

                                   clk,
```

If you reuse, hardware count can probably be very marginally reduced. So, it does not really warrant such exercise. If you want different timings, for example, you want to use the very same thing for some other road wherein you need something else instead of 45 seconds and all of them are in main roads instead of side roads, you can very easily amend by changing these values. That is what this statement says. We need to

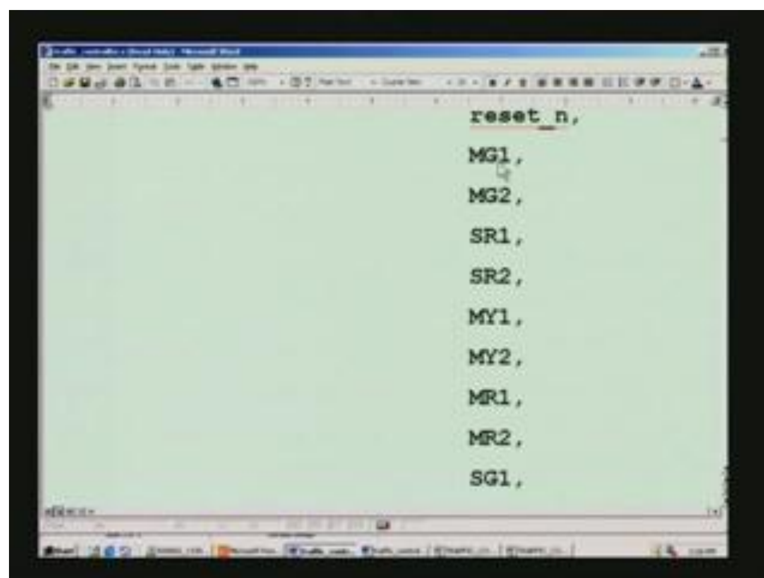
invoke the actual design, which is traffic controller and we will have to call it by this module name that you have already seen. This lists all the ios here.

(Refer Slide Time: 12:40)




For example, we have a system clock, then power on reset here. These are all just the lamps, the traffic light lamps. You can easily make out the meaning of MG1. M stands for main, G for green and 1 for main road 1. So is the case for all the others.

(Refer Slide Time: 12:59)



Finally, we have a blink as one of the input control. Towards the end of the traffic, the cop normally switches off. Alternatively, you can have a separate timer which will switch this on based upon the day. It will switch on, of course, during the night.

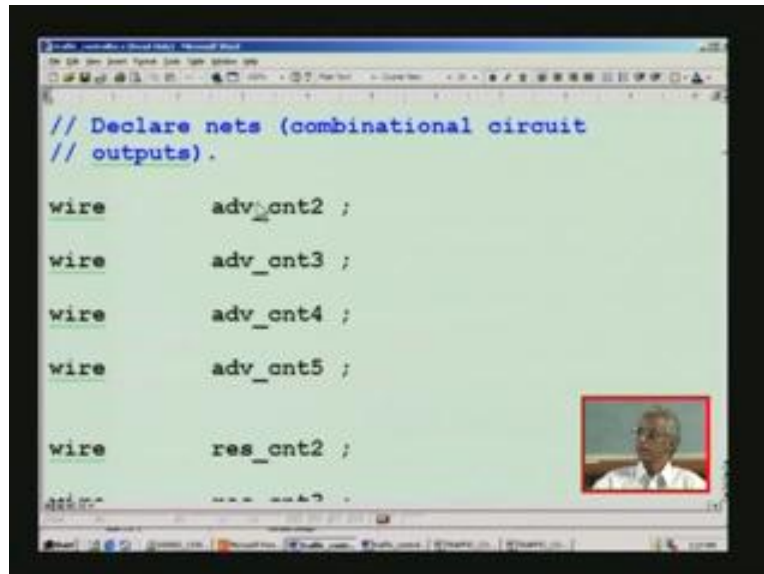
(Refer Slide Time: 13:13)

We have a listing of same I/Os here. All of them are single bit, therefore there is no width declared here.

(Refer Slide Time: 13:35)

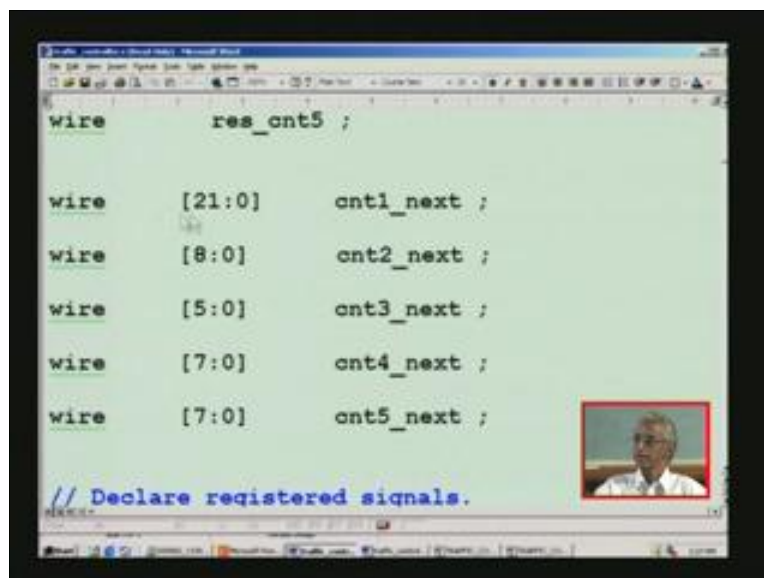
I/Os are all declared here. We also have some assign statements. All those would demand a wire declaration and that is what is here. This is the signal for advancing the counter. Similarly, I have adopted this nomenclature to reset the counter.

(Refer Slide Time: 13:50)




We also need next value for each of the counters, which we will use later on. That next value is designated by next with the same name of this counter.

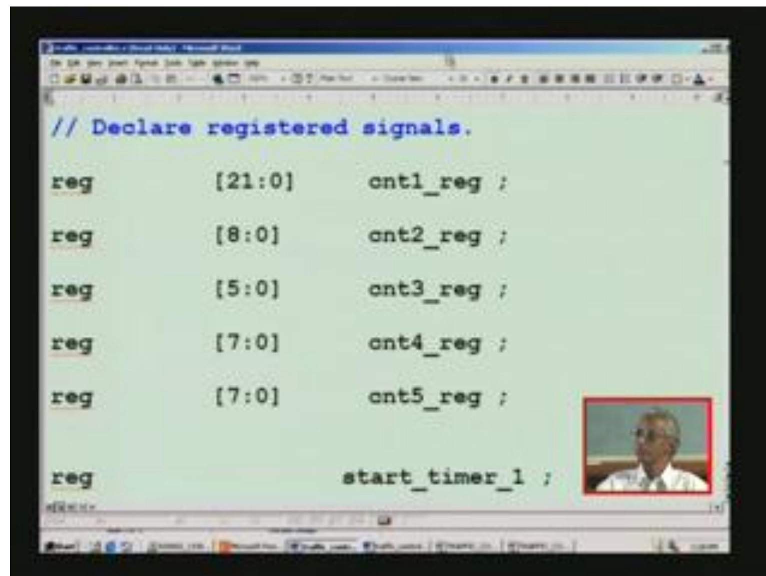
(Refer Slide Time: 14:19)




The width of each of these counters is here. For example, 22 bits are used for the very first time base because you have to store a very huge figure of 4 million. So is the

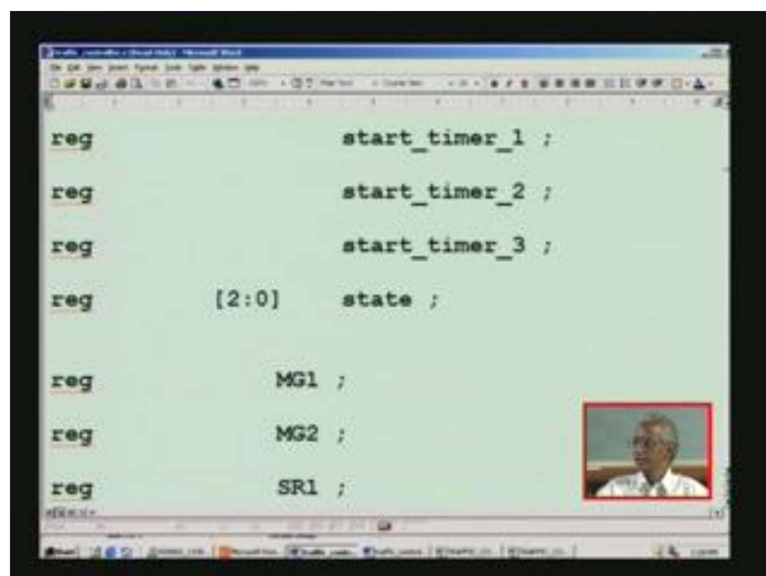
case for other counters. You can change depending upon how much time you need. If you want much higher timing or lower timing, you can always change this width. We declared the registers that what we have seen next comes as the actual register value here and same bits happened to be here.

(Refer Slide Time: 14:48)



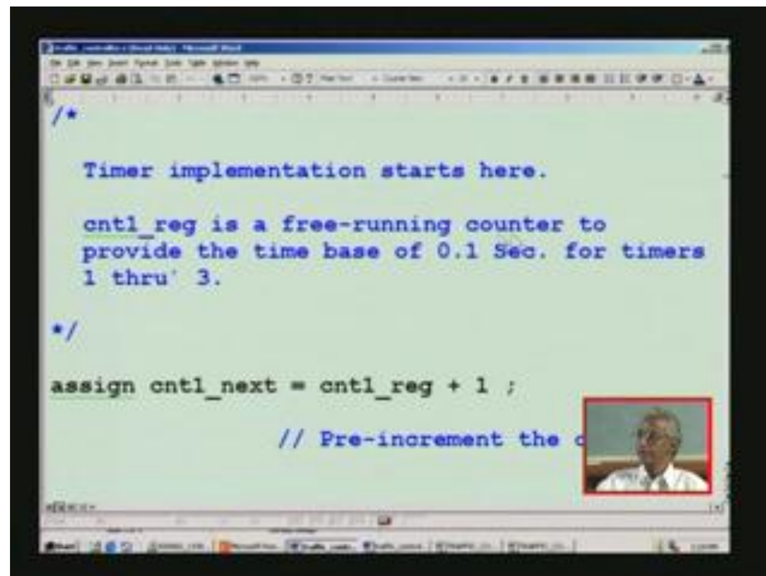

We also have 3 timers running and these can be started by this control. There are also reg because all these reg appear in always block, especially the clock portion. The positive edge of the clock block will be having this all reg as also these traffic lights here.

(Refer Slide Time: 15:10)



The actual timer implementation starts here. Counter reg is a free running counter to provide the time base of 0.1 second. We have already seen this for timers 1 through 3. We have 3 timers and there is 1 more also for the flashing. That is not really a timer; it is only a free running counter. We will see that later on.

(Refer Slide Time: 15:30)




To start with how we design the first counter. Let us have a look at the counter itself. It is a timer. The moment we cover this it will be apparently clear to you. First, we increment the counter. It is increment of the counter in advance so that any computation will defer it, I mean, put it outside the always block, so that during the event of the clock, we merely just assign whatever we have computed. We will do all the computation outside and only transfer the contents later on.

To start with, we have an always block here. Begin and corresponding end will be there and it is based upon the positive edge of clock. Only if the positive edge is encountered, will this always block be executed as also when reset pulse is applied and that negative edge of reset recants. This has been the very standard design we have had and that you are already familiar with all through this course.

(Refer Slide Time: 16:30)

```
always @ (posedge clk or negedge reset_n)
begin
   if (reset_n == 1'b0)
              cntl_reg <= 22'd0 ;
// Initialize when the system is reset.
   else if (cntl_reg  == `time_base)
// Reset if terminal count is reached.
          cntl_reg <= 22'd0 ;
```

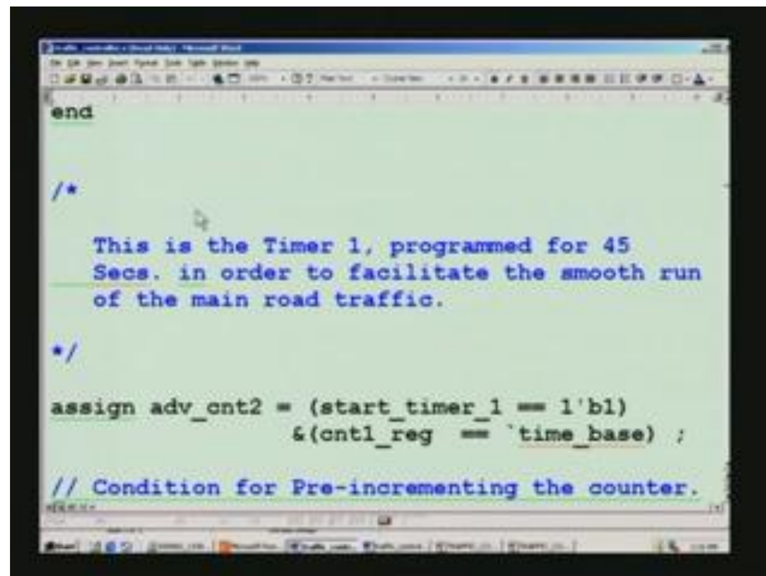
When reset is applied it is active low and we need to reset this particular counter. Otherwise, else if Is for otherwise, if counter 1 is equal to time base, it means that it has gone through all the counts so we only have to reset back so that, once again, it starts going round and round. That is the statement doing that job. If not, we have already pre incremented so we have merely to assign that incremented value here so that when the clock strikes, we do not have to start taking the count here. It is a good design practice to put all time consuming operations outside this always block because assign statements are always executed all the time.

(Refer Slide Time: 17:24)

```
   else if (cntl_reg  == `time_base)
// Reset if terminal count is reached.
          cntl_reg <= 22'd0 ;
   else
             cntl_reg <= cntl_next ;
// Otherwise, advance the count once.
end
```

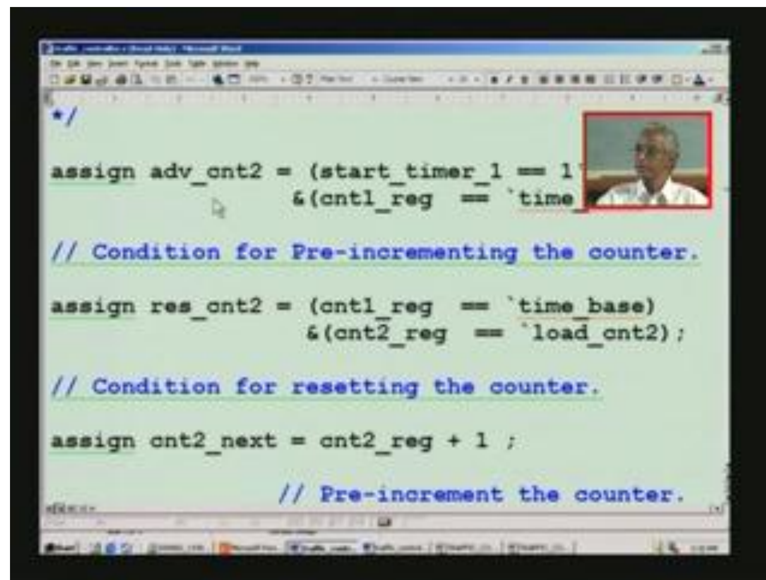
This is for the time based and such a simple thing for realizing the counter and we will see that other counters are also realized likewise except that the timing is different. This is the timer 1 programmed for 45 seconds in order to facilitate the smooth run of the main road traffic.

(Refer Slide Time: 17:50)



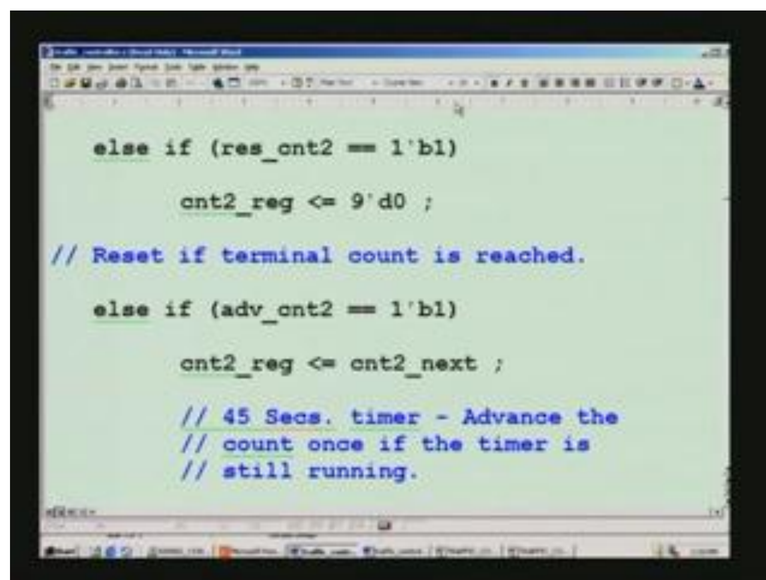

We use an assign statement in order to compute the advancing of the counter. Second counter will advance the counter only if start timer is asserted. This is 1 and counter 1 is equivalent to the time base, that is to say, we need to take action at every 0.1 second. Normally, the count will advance from 0, 1, 2, 3 and so on. Only when it touches 4 million minus 1 will this activity take place. Only then the counter advances. Similarly, for resetting, it is once again in the same pattern except that counter 2 is currently running so that will also have a load, which we have already seen using a defined statement, and when it is equal to that load value, then this also advances from 0, 1, 2, 3 and so on. Finally, it will touch the load count, which is fag end of chip. For example, for 45 seconds, we put it as 449 there. When it touches 449, which is equivalent to 4 million minus 1, only then will this counter be reset.

(Refer Slide Time: 18:41)




Once again, you have an advance increment for the counter. This is the always block wherein counter 2 is realized. Once again, you have a reset to reset this particular counter and also to reset if this condition, the condition that we have already seen earlier, is satisfied.

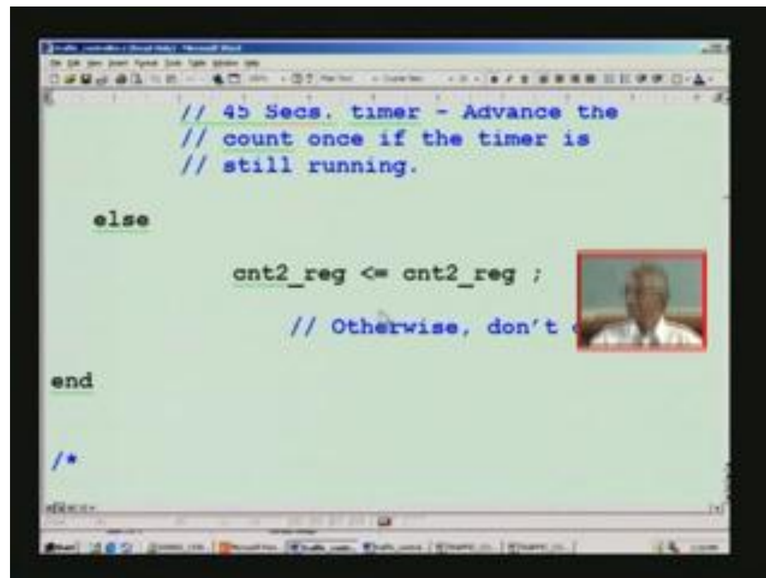
(Refer Slide Time: 19:45)




We have seen the condition for advancing also, and it is merely assigning into counter register here. Thus you have a 45 seconds timer, and advance the count once if the timer is still running. That is what the comment says. Otherwise, do a no action. You

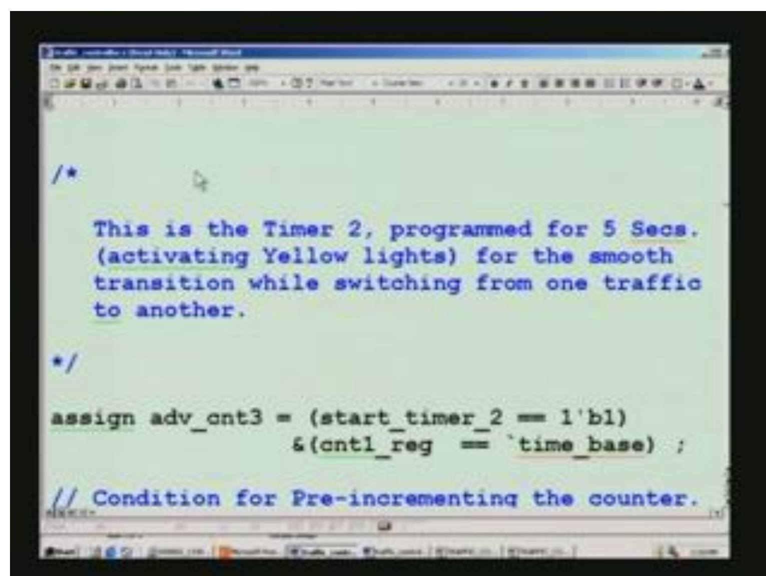
can just put a dummy statement, no statement at all, or just write the same content there.

(Refer Slide Time: 20:02)



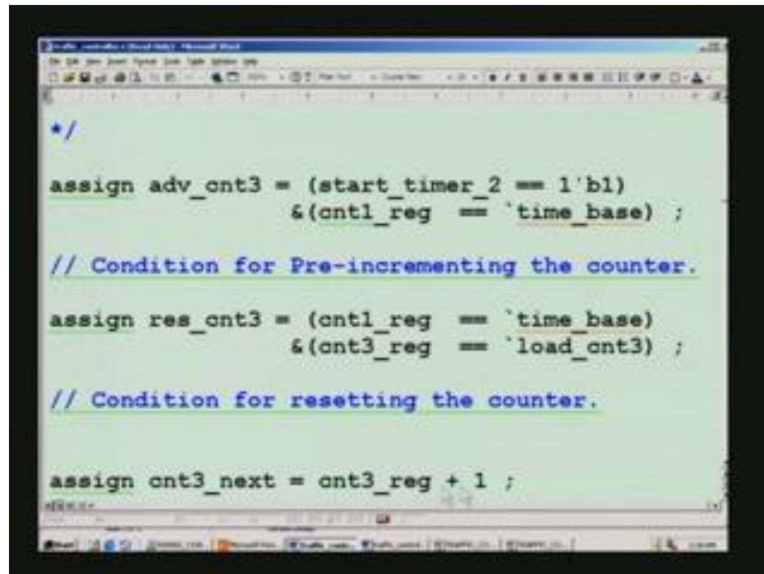

Likewise, we have to realize timer 2, timer 3. I will quickly go through this. It is exactly the same thing except that, instead of 45 seconds, we will realize only 5 seconds here. This is for the yellow lights, for the smooth transition while switching from one traffic to another. This is the advance count. Once again, timer 2 will be taken into account here and, as usual, counter 1 time base will also be taken into account because we need to take only when the time base is ripe.

(Refer Slide Time: 20:12)



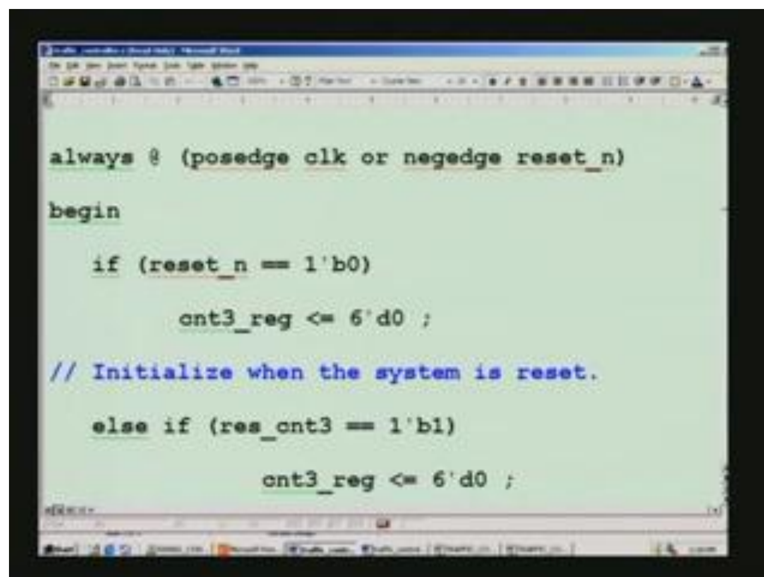
Just like counter 2, we have counter 3 here, and it compares with the corresponding load value. This is for advance, this for reset. Once again, you have advance increment for the counter.

(Refer Slide Time: 20:38)




In always block, realizing exactly the same thing that we have seen before.

(Refer Slide Time: 20:55)




For power on reset here, then reset and advance condition.

(Refer Slide Time: 20:59)

```
begin

   if (reset_n == 1'b0)

         cnt3_reg <= 6'd0 ;

// Initialize when the system is reset.

   else if (res_cnt3 == 1'b1)

               cnt3_reg <= 6'd0 ;

// Reset if terminal count is reached.

   else if (adv_cnt3 == 1'b1)
```

Resetting here and incrementing here and this realize the 5 seconds timer once again, a dummy statement here.

(Refer Slide Time: 21:03)

```
               cnt3_reg <= 6'd0 ;

// Reset if terminal count is reached.

   else if (adv_cnt3 == 1'b1)

               cnt3_reg <= cnt3_next ;

// 5 Secs. timer - Advance the count once if //
the timer is still running.

   else

               cnt3 reg <= cnt3 reg  ;
```

You have a timer 3 which is programmed for 25 seconds and is exactly the same as counter 2 as well as 3.

(Refer Slide Time: 21:16)

```
end

// This is the Timer 3, programmed for 25
// Secs. delay, used for the side road
// traffic.

assign adv_cnt4 = (start_timer_3 == 1'b1)
                  &(cnt1_reg  == `time_base) ;

// Condition for Pre-incrementing the counter.
```

We have an advance counter, reset counter, satisfying exactly the similar condition.

(Refer Slide Time: 21:22)

```
assign adv_cnt4 = (start_timer_3 == 1'b1)
                  &(cnt1_reg  == `time_base) ;

// Condition for Pre-incrementing the counter.

assign res_cnt4 = (cnt1_reg  == `time_base)
                  &(cnt4_reg  == `load_cnt4) ;

// Condition for resetting the counter.

assign cnt4_next = cnt4_reg + 1 ;
```

An advance increment here, then the always block, all precisely the same.

(Refer Slide Time: 21:28)

```
// Condition for resetting the counter.

assign cnt4_next = cnt4_reg + 1 ;

// Pre-increment the counter.


always @ (posedge clk or negedge reset_n)

begin

  if (reset_n == 1'b0)

                cnt4_reg <= 8'd0 ;
```

There is also a timer 4 programmed for 1 second delay, used for the blinking of all the yellow lights after normal traffic hours. This is what I have mentioned earlier. Here the advance counting is when blink equal to 1. That is the switch there so it will start blinking only when you say blink. The corresponding time base is, once again, taken into account here.

(Refer Slide Time: 21:41)

```
end


// This is the Timer 4, programmed for 1 Sec.
// delay, used for the blinking of all the
// yellow lights after the normal traffic
// hours.


assign adv_cnt5 = (blink   == 1)&
                  (cnt1_reg  == `time_base) ;

// Condition for Pre-incrementing the counter.
```

Reset is based upon the time base, once again, as well as the count value. Once again, it is exactly the same.

(Refer Slide Time: 22:09)

```
                    (cnt1_reg  ==  time_base) ;

// Condition for Pre-incrementing the counter.

assign res_cnt4 = (cnt1_reg  ==  `time_base)
                 &(cnt5_reg  ==  `load_cnt5) ;

// Condition for resetting the counter.

assign cnt5_next = cnt5_reg + 1 ;

// Pre-increment the counter.

always @ (posedge clk or negedge reset_n)
```

So any timer can be very easily realized by very simple counters like this and then always block as usual.

(Refer Slide Time: 22:16)

```
// Condition for resetting the counter.

assign cnt5_next = cnt5_reg + 1 ;

// Pre-increment the counter.

always @ (posedge clk or negedge reset_n)
begin
  if (reset_n == 1'b0)
               cnt5_reg <= 8'd0 ;
```

With reset, advance, and this gives 1 second timer.

(Refer Slide Time: 22:25)

So machine state starts right here. The fsm for the traffic light is here. Once again, you have an always block, considering at positive edge of clock. This is our standard format.

(Refer Slide Time: 22:40)

You take action if there is a power on reset, in which case you just reset all the lamps.

So, all the lamps are here.

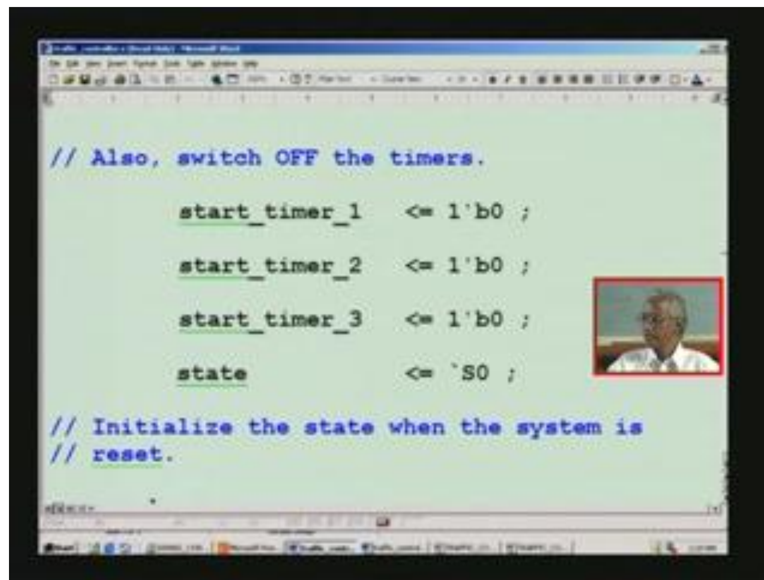
(Refer Slide Time: 22:54)

Note that all of them are made 0 here. This is for initialing phase.

(Refer Slide Time: 22:57)

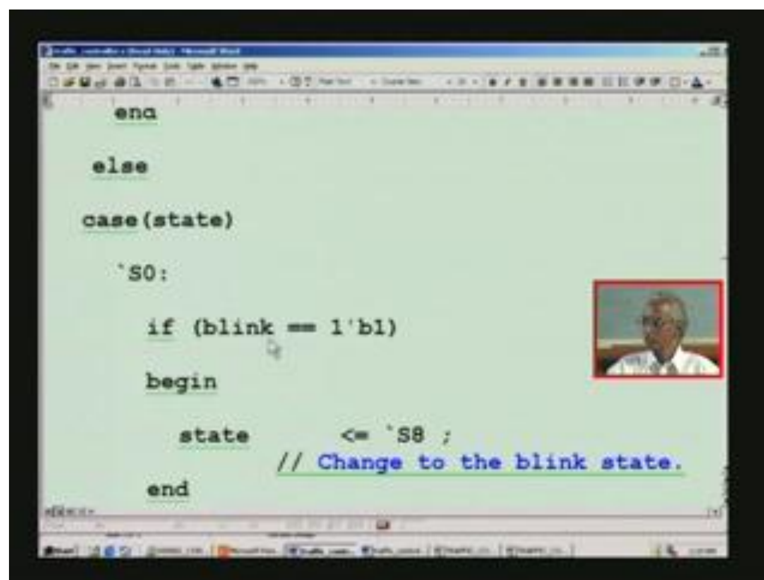
This is also with the timers, so that no timer will start without being called for. We have different states as we have seen in the power point earlier S0 through S8 states and that state is defined here as state itself. These have been defined as S0, S1 and so on. To start with, this will be 0 and decimal and state will be 0.

(Refer Slide Time: 22:03)



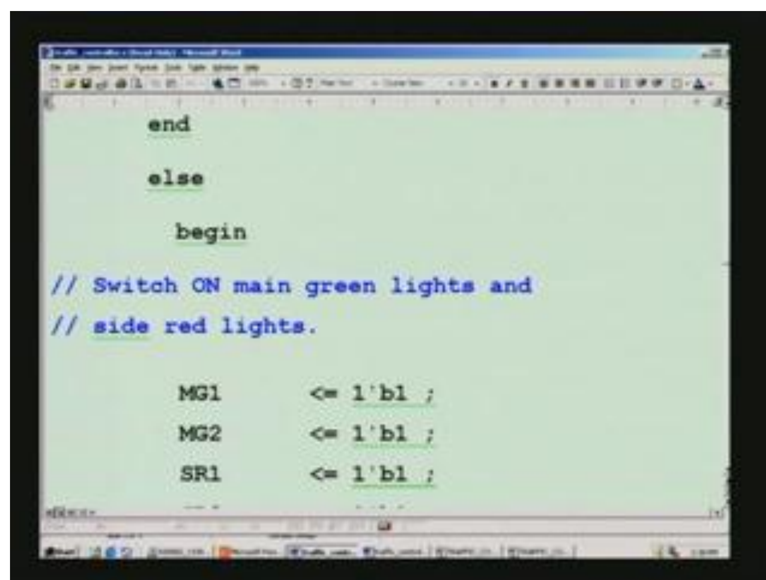

You can understand this clearly when you see the waveform. In fsm, what we will do is once we reset, if the reset is not present we will realize a case here. In this case we will have all the S0 through S8 states realized and if needed we will examine the blink. Right in the first state, we will examine whether blink is 1 and if it is 1 take it to the last state, S8. In power point, you would have seen that S8, the last one shown, is for the flashing of the lights. It will automatically take and this query will be made only during the S0 phase. So no matter when the cop switches it, only after going through sequence and touching S0, will the blink will be sensed.

(Refer Slide Time: 23:51)



The entire sequence takes 2 minutes and if you want to take immediately, you may have to incorporate this in every state possible. That is not really a requisite because we will turn it off only towards the end of the day. So that is not really required; this is quite adequate. So in the first S0 state, the actual lamp turning on or off will arise here. If that particular condition is satisfied so that is blink, then we do this 1 and if not satisfied we come to the normal functioning here. Note that if statement. Once it has executed that one, it will not execute any of this down below and it will straightaway go out of this, even out of this case. That is the beauty of this if statement.

(Refer Slide Time: 25:05)




We have examined priority coder several times earlier. We turn on the lights. For example, we want a main road. There are 2 green either way and we need to turn them on as also the side red because we do not want to allow the traffic for the side road. We should not forget to turn off all other lamps. That is what these statements are doing.

(Refer Slide Time: 25:42)

```
// Switch OFF all other lights not wanted.

        MY1      <= 1'b0 ;
        MY2      <= 1'b0 ;
        MR1      <= 1'b0 ;
        MR2      <= 1'b0 ;

        SG1      <= 1'b0 ;
        SG2      <= 1'b0 ;
```

Here, if reset counter 2 that is counter 2 is designated as 45 seconds and if that is one, that is it has touched the final value, we have to start the timer 1 here.

(Refer Slide Time: 26:06)

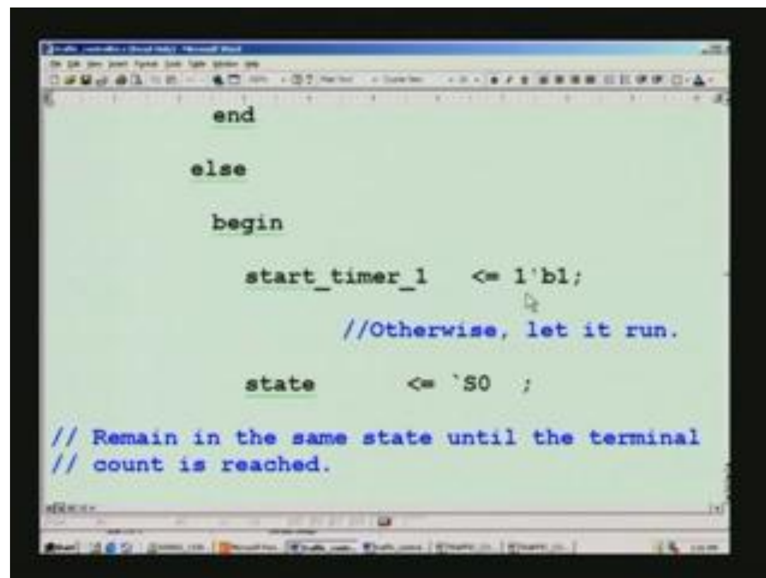
```
// This corresponds to 45 Secs. timing of
// timer 1.

          begin

             start_timer_1   <= 1'b0 ;

// Stop the timer if the terminal count is
// reached.

             state           <= `S1 ;

                         // Change the state.
```

I wonder whether there is 1 needed here. I will have a look at it, may not be required. What state are we in? S0 state and it goes to state S1.Why should I go to S1? I will have a look.

That is 45 seconds have elapsed so with the elapse of that you need to go to see. Did we switch on the timer earlier? No we did not switch on the timer. Let us see.

So we will come back to this. Anyway, we need to go to the second state after the elapse of 45 seconds. Otherwise, we start the timer only here so why is it so? Does it mean 45 seconds have not elapsed? You just verify that. Let us see. So we start the timer and allow it go to S0 state. That is remaining in the same state so this explains why we have done like that.

(Refer Slide Time: 27:48)




We need to start only if it is S0 state. This is the very first sequence. We need to start the timer and wait for the 45 seconds to be over. It keeps on revolving around this and when it is over it will take to S1 state. That is what the implication here is. We should probably have put the other way to avoid confusion, nevertheless it does the job. So that is for S0 state. In S1 state, we need to turn on yellow lights and red lights and turn off all other lights as you see here.

(Refer Slide Time: 28:31)

```
// side red lights.

        MY1       <= 1'b1 ;
        MY2       <= 1'b1 ;
        SR1       <= 1'b1 ;
        SR2       <= 1'b1 ;

// Switch OFF all other lights not wanted.

        MG1       <= 1'b0 ;
        MG2       <= 1'b0 ;
```

Once again, we check counter 3, which is programmed for 5 seconds, and this pattern will be similar to that. Again we do not start the timer here but later on. We will take the control to the S2 after expiry of 5 seconds.

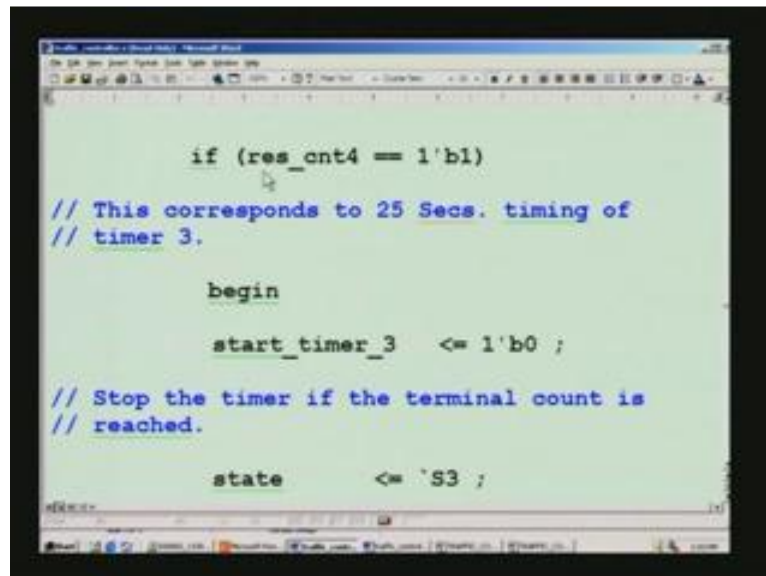
(Refer Slide Time: 28:50)

```
// This corresponds to 5 Secs. timing of
// timer 2.

          begin

            start_timer_2   <= 1'b0 ;

// Stop the timer if the terminal count is
// reached.

            state       <= `S2 ;

                        // Change the state.

          end
```

Expiry comes right at the beginning and starting of the timer comes late so S1 is the very same state. If you wish, you can always change it. Put it before this. It should also work then.

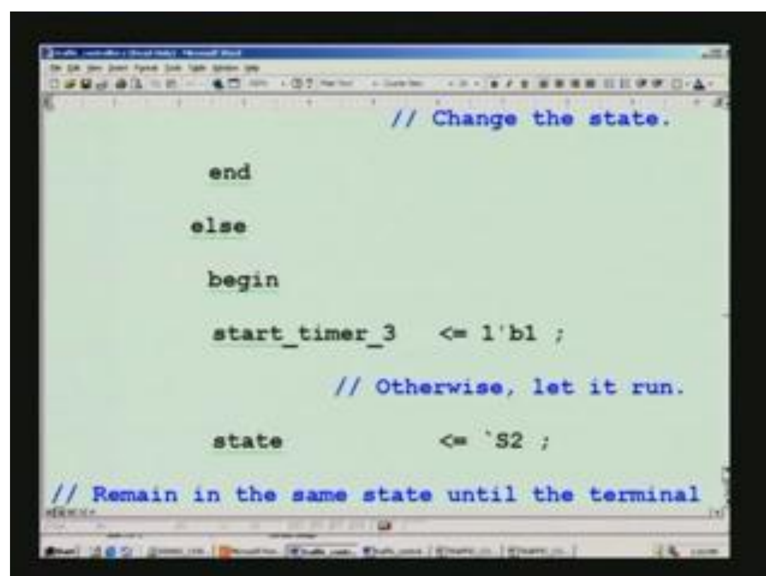
So in S2 state we need to switch on, that is, allow the right traffic to flow and all reds will have to be switched on in order to prevent the straight traffic going, and switch off all other lamps. Once again, for the next sequence, we check the counter 4 and this happens to be 25 seconds.

(Refer Slide Time: 29:39)




Once again, switch off the timer and remain in the same state, S3 state. Then switch on the timer, and this remains in the same state, and S3 is the next state.

(Refer Slide Time: 29:50)




This is S3.

(Refer Slide Time: 30:01)

```
            state           <= `S2 ;

// Remain in the same state until the terminal
// count is reached.

            end

        end

      `S3:

        begin

// Switch ON main yellow lights and
```

Here we have to switch on all the main yellow lights and side red lights, so side red and main yellow, and switch off all other lamps.

(Refer Slide Time: 30:05)

```
// Switch ON main yellow lights and
// side red lights.
          MY1       <= 1'b1 ;
          MY2       <= 1'b1 ;
          SR1       <= 1'b1 ;
          SR2       <= 1'b1 ;

// Switch OFF all other lights not wanted.
          MG1       <= 1'b0 ;
          MG2       <= 1'b0 ;
```

We inspect for 5 seconds and, once again, timer is switched off at this point of time and control branches off to the next state.

(Refer Slide Time: 30:20)

```
if (res_cnt3 == 1'b1)
// This corresponds to 5 Secs. timing of
// timer 2.
    begin
    start_timer_2   <= 1'b0 ;
// Stop the timer if the terminal count is
// reached.
    state           <= `S4 ;
```

Suppose you wanted the reversal of this. What you do is, if reset counter is not equal to 1, you can then put this group first here so that will avoid confusion a bit. Otherwise, we start the timer and remain in the same state S3.

(Refer Slide Time: 30:49)

```
else
    begin
    start_timer_2   <= 1'b1 ;
                // Otherwise, let it run.
    state       <= `S3 ;
// Remain in the same state until the terminal
// count is reached.
    end
```

All other sequences are exactly the same except that appropriate lamps are turned on. fsm realization is quite a simple one. You would have noted that we have started the design without any asm chart because it is so plain, simple and direct. You do not need the asm chart at all. That is the beauty of this particular example at least.

(Refer Slide Time: 31:04)

```
// Switch ON main red lights and
// side green lights.

        MR1     <= 1'b1 ;
        MR2     <= 1'b1 ;
        SG1     <= 1'b1 ;
        SG2     <= 1'b1 ;
// Switch OFF all other lights not wanted.
        MY1     <= 1'b0 ;
```

This design is quite simple as well as small. Later on, we will be looking into the video compression DCTQ. That will be much bigger; quite a large design. Then we will have both the extremes. We have seen that red light must be on and side green must be turned on at this point of time.

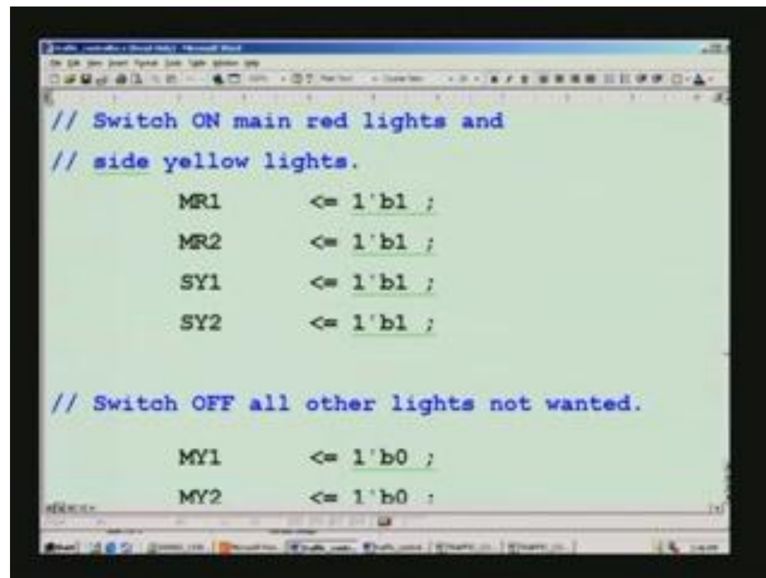
In S4 state, you have to switch on main red lights and side green. That is what it is here. Turn off all other lamps. Once again, for next sequence, we need 25 seconds and we check, this time, counter 4 and stop the timer and go to the next state or start the timer and remain in the same state.

(Refer Slide Time: 32:01)

```
        if (res_cnt4 == 1'b1)
// This corresponds to 25 Secs. timing of
// timer 3.
         begin
         start_timer_3   <= 1'b0 ;
// Stop the timer if the terminal count is
// reached.
         state      <= `S5 ;
```

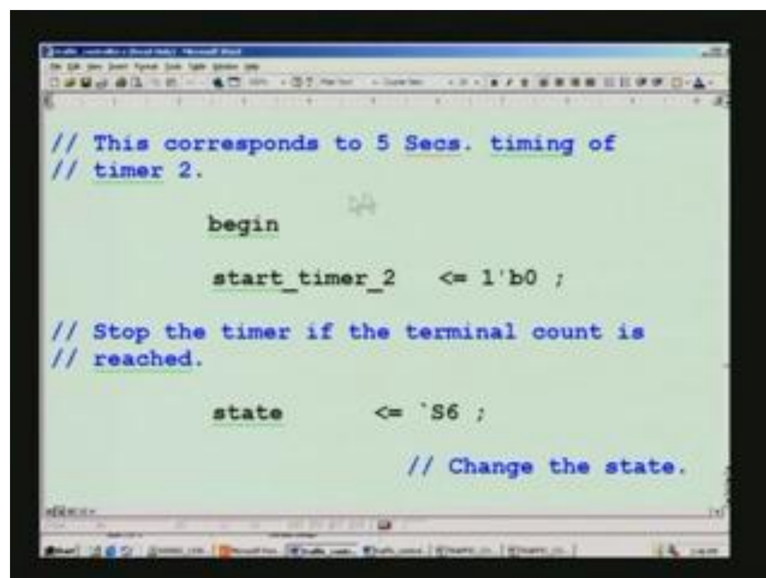
This is the very same thing that you get repeatedly. In the fifth state, you need to turn on main red lights and side yellow lights and switch off all other lamps.

(Refer Slide Time: 32:21)




Once again, a 5 second is coming.

(Refer Slide Time: 32:32)




You start the timer or the other way. Start it here and remain in the same state, otherwise go to S6. State S6 is here.

(Refer Slide Time: 32:42)

```
                    // Change the state.
    end
   else
    begin
    start_timer_2   <= 1'b1 ;
               // Otherwise, let it run.
    state       <= `S5  ;
// Remain in the same state until the terminal
```

In this case, main red and side red lights as well as side right turn lights are turned on so this is self-explanatory. Switch off all other lamps.

(Refer Slide Time: 32:50)

```
// Switch ON main red lights,
// side red lights, and side right turn
// lights.

        MR1      <= 1'b1 ;
        MR2      <= 1'b1 ;
        SR1      <= 1'b1 ;
        SR2      <= 1'b1 ;
        SRT1     <= 1'b1 ;
        SRT2     <= 1'b1 ;
```

Here we have 25 seconds and, once again, we go to the next state if timer is off. Then we make the timer on and then remain in the same state.

(Refer Slide Time: 33:20)

```
end
else
begin
start_timer_3    <= 1'b1 ;
              // Otherwise, let it run.
state       <= `S6 ;
// Remain in the same state until the terminal
// count is reached.
```

In S7 state, we need main red lights and yellow lights, as well as side red lights. Switch off all others.

(Refer Slide Time: 33:30)

```
// Switch ON main red lights,
// side yellow lights and side red lights.

MR1      <= 1'b1 ;
MR2      <= 1'b1 ;
SY1      <= 1'b1 ;
SY2      <= 1'b1 ;
SR1      <= 1'b1 ;
SR2      <= 1'b1 ;
```

Once again, a 5 seconds delay, which you can just have a glance, because we have encountered exactly the same thing. Now, comes the blink. This is for the blinking state, S8 state. If blink is on, this corresponds to 5 second timing of timer 2.

(Refer Slide Time: 33:52)

```
`S8:
        if (blink == 1'b1)
// This corresponds to 5 Sec. timing of
// timer 2.
         begin
          begin
// Switch OFF all other lights not wanted.
```

Is it correct? I think it should be timer 5 as words. In this case, we turn off all lights except for the yellow lights.

(Refer Slide Time: 34:17)

```
MR1      <= 1'b0 ;
MR2      <= 1'b0 ;
SR1      <= 1'b0 ;
SR2      <= 1'b0 ;

MG1      <= 1'b0 ;
MG2      <= 1'b0 ;

SG1      <= 1'b0 ;
```

So we have turned off all lights.

(Refer Slide Time: 34:28)

```
SRT1     <= 1'b0 ;
SRT2     <= 1'b0 ;

end

if ((cnt1_reg  == `time_base) &&
    (cnt5_reg  == `load_cnt5))
begin
SY1      <= ~SY1 ;// Blink all
```

We will blink only those yellow lights. Yellow lights are side as well as main for both the directions.

(Refer Slide Time: 34:32)

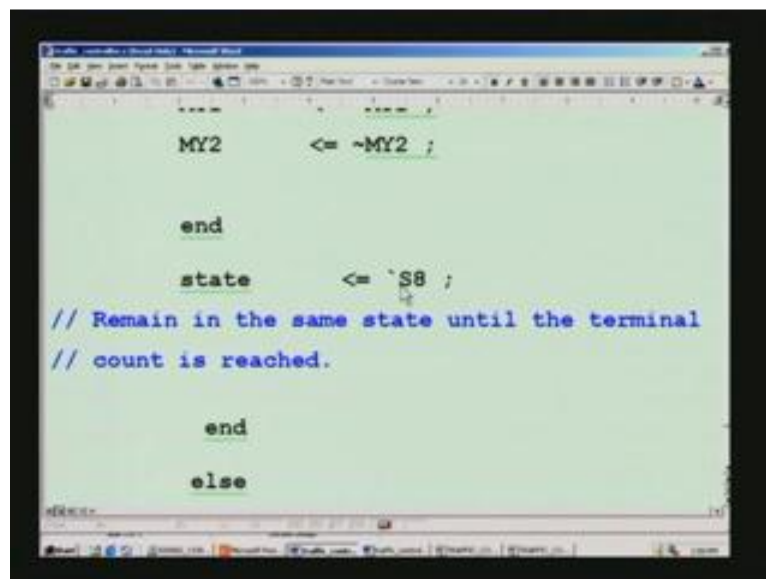
```
if ((cnt1_reg  == `time_base) &&
    (cnt5_reg  == `load_cnt5))
begin
SY1      <= ~SY1 ;// Blink all
                  // yellow lights,
SY2      <= ~SY2 ;
MY1      <= ~MY1 ;
MY2      <= ~MY2 ;
```

We take the very same signal and invert and assign it to itself, so if it were 0, it will be 1. Therefore, it will keep on toggling, which means that it is flashing light here, or what you call blink light, because of the inversion of the same signal. How and when should it happen? It should happen when counter 1 is equal to the time base, that is 0.1 second time, and counter 5 being equal to the load value of the count 5. I think I made a mistake there, as pointed out earlier. It should have been timer. It may be

correct. It is for 5 seconds. Okay, there is something wrong. I must have copied the previous comment and did not change this, so this should be read as 1 second and timer 5 or counter 5.

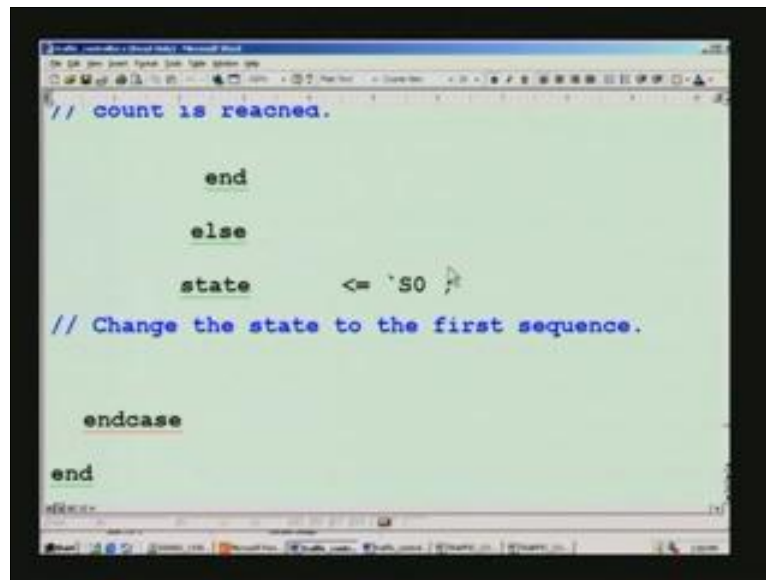
So that is the beauty of verilog design; you can always make a copy from one portion into another. In fact once the source codes are available, we do not have to key in any of the instructions. You can merely rename this to your design and change wherever your design needs arise. That is how you make quick designs. That is what we meant here. We need to blink only after the lapse of this happening so this will happen only when it is 0.1 second and into this 10. This is counter loaded by 9 so it advances 0 through 9, which means 10. 10 into 0.1 is 1 second. Every 1 second, this will be inverted, which means that every 2 seconds 1 cycle is completed, which means 1 by 2 that is, 0.5 hertz. That is what we said right at the start. If this is satisfied then, having done, this is remaining at the same state, S8 state, as long as the blink is still 1.

(Refer Slide Time: 36:44)



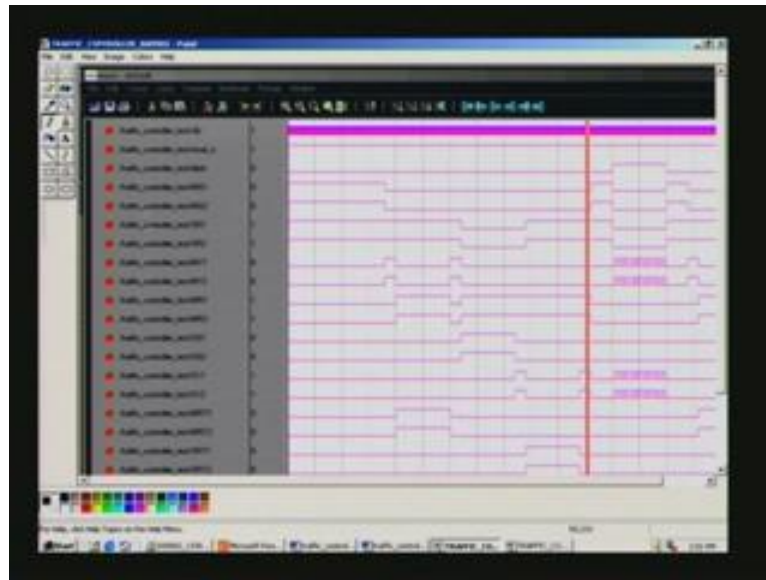

If the blink is not 1; else will take into effect, so, it will automatically take it to S0 state. That is what we said before. Blink is also examined in S0 state, as we have seen if you remember, and only at that particular state. Whenever it goes from S8 state it always lands at a S0 state. Similarly, at S0 state alone, that blink is examined. If your blink is switched on in any other states, it will be recognized only when the S0 state arrives. That is what I said before: if it is just missed when the cop switches on the blink switch, it will take about 2 minutes or so to process that.

(Refer Slide Time: 37:00)




2 minutes is nothing because the cop is just going home towards the night, right? Change the state to the first sequence. That is what it says. We started with the case; therefore there must be an end case. It was in always block; therefore begin and matching end. We started with a module declaration, and therefore, there must be an end module here. As usual, we have synplify results. First let us examine the actual outputs here. Before zooming into this output, I will just explain what these signals are. The first one is clock and it is 2 higher in frequency. You cannot see the clock because it is in megahertz and we are in the second regions here and 45 seconds etcetera we see here. Naturally, you cannot see the clock. There is also power on reset here and that is also half duration 20 nano seconds so that gets merged here. You cannot see that. These are all the usual signals that have been used in several designs before. This is the blink input here and it goes high at this point of time and you can readily note that some flashing is happening here. These 2 lights are yellow lights MY1 and MY2. The same is the case for SY1 and SY2, side as well as main. That is what we want here and the rest is precisely what it says.

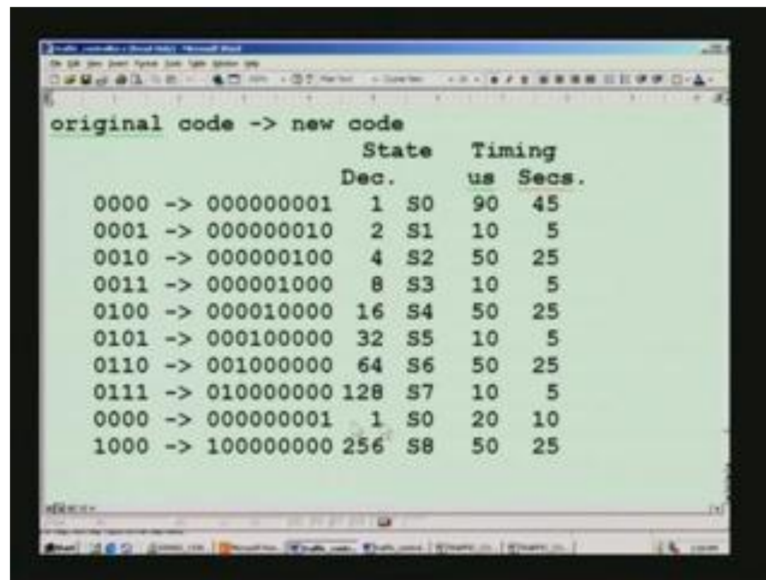
(Refer Slide Time: 38:08)



We have listing of all lamps. There are totally 16 lamps, main green, side red, and so on. Yellow, red, green, and yellow for side, then right then this are for side red then main.

Before we look into the fm, have a small diversion back to the synplify. Of course, within we did not start with synplify, let us have a look at that first. In synpflify results, what we had given S0, S1 etcetera are not a part of synplify, overwritten; some pertain to our design information so this synplify reports only these here and right up to this. I have added this just to make it clearer to you. Synplify is an optimizing tool so it normally allocates for a fsm state one hot assignment. This is state that we had requested but synthesis tool has made one hot assignment here. This 0 state corresponds 1 and this one in decimal I have put here and all these things are put bias here in order to explain this out.

(Refer Slide Time: 39:57)



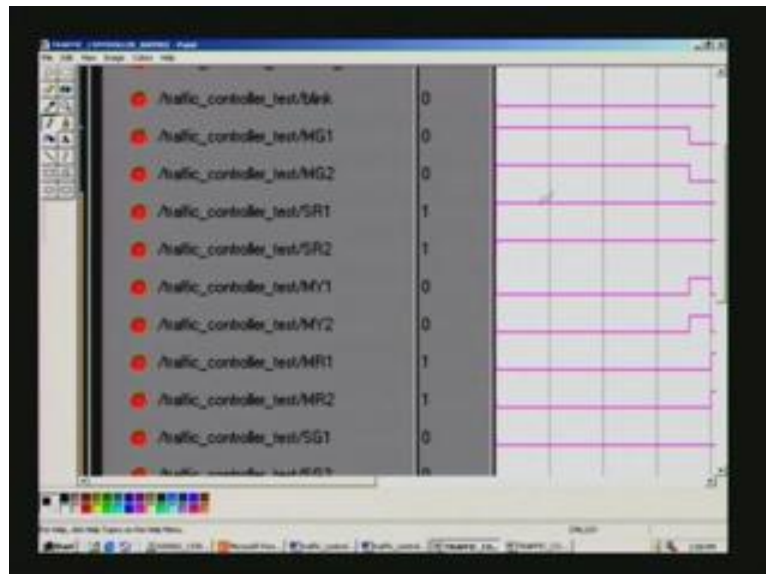

So 1, then this is 2, 4, 8 you can see follows this here, 16 then 32 all binary change with the exception of this. In synplify, you will not see this here. I have just added here in order to make clear to you the functioning of the traffic controller. The idea here is we are going from one state to another, S0 to S1 and so on. After S7, we need to come to S0, only then will blink be recognized. Only then can it go to S8 state, which is blink state. I have put it in this fashion in order to explain that. These are all the standing explains and this last one is 2 digits. When we see the actual waveform, you will see these timings there, 90 microseconds etc. As explained before, the time base was just limited to ten instead of 4 million, because it would be too enormous and we would not be able to have a look all put together. I made it into 10 there so that it can be confined into a single page. These are all the timings that we will see, and this corresponding time is the actual time that we need is 45 seconds this 90 microseconds, you can interpret as 45 and then you can see this ratio maintained all through.

This will prove that all the counters are working precisely. In fact, I suppose there is a 15 nano second error only for the very first clock. I am not very sure, but as far as the output is concerned, even that 15 nano seconds error is not there. I would recommend that you go through all these simulation signals after back annotation, all these results once again after back annotation. Back annotation is what we get after delays get incorporated. After Xilinx place and route, you have to do back annotation and then

simulate. That is what the real picture is. Otherwise, you will be mistaken thinking that it will give a high frequency of operation. That is not really correct unless you do the back annotation, as I have insisted all through. Before we go in to details let us finish the waveform. I have just mentioned that all the lamps are here. The final state is also mentioned here. I will just read out here I am going to zoom so that you can read it for yourself and verify.

This is state 1, and then 2, 4, 8, 16, 32, 64, 128, then 1, then 2 digits. This is then again 1 because the whole sequence is complete. This corresponds to S0, S1, S2, exactly the same thing that we have just examined in the synplify results. That was the assignment made by the optimization tool and, as it is after the back annotation that will get reflected here whereas merely after the simulation that would not have got reflected. So this clarifies that. We have one more waveform which will also clarify; I mean, prove to you the need of back annotation, especially the gate alley delay you can see. Let us have a look at different states, whether it is doing the right job or not. In this case, if you examine, I will first zoom in here. You can see MG1. This is the very first state so it is being high anything more than this, what is at bottom, you can recognize as 0 and what is on the top of this digit you can recognize as 1.

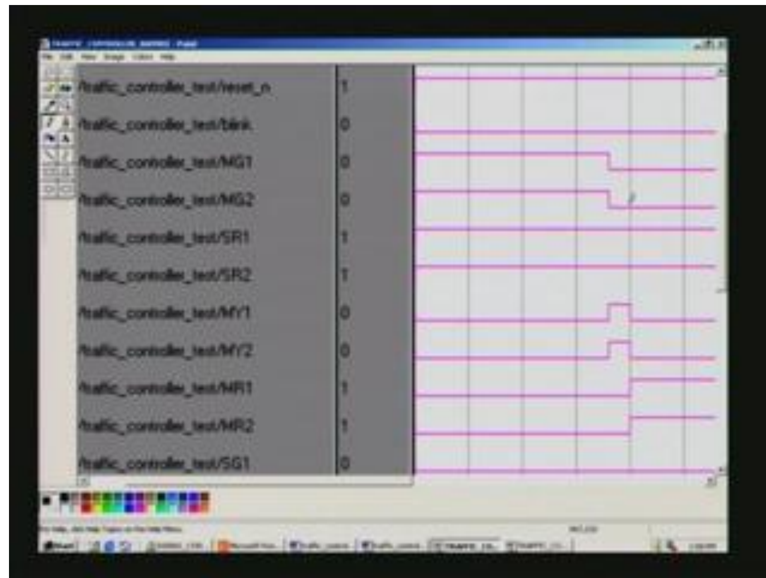
(Refer Slide Time: 44:30)



For example, this is 0 here so you can quickly see what is here. MG 112 as well as side road, all of them is 1. This is the very first state and all of the other lamps are 0s here. This is what we want here for the first state. So that is what this one is. Then it

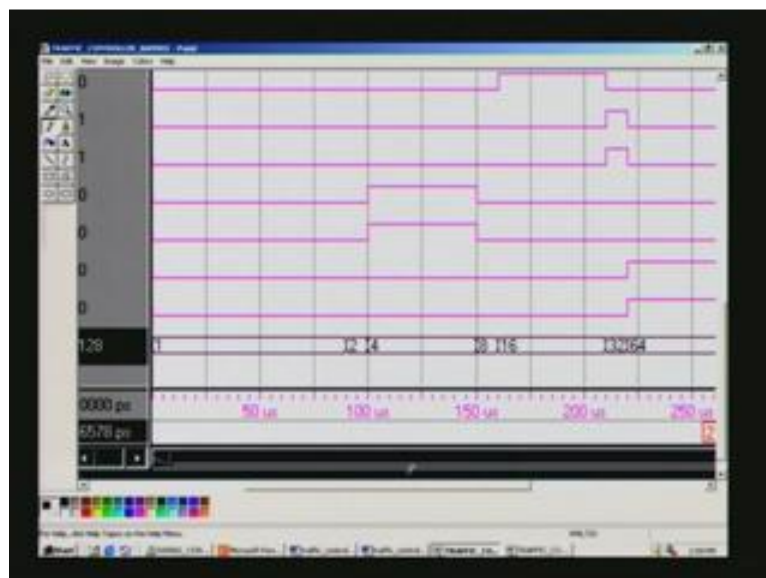
turns yellow. That is for second state here. That is here. This is small duration, 5 seconds. That is here. Unfortunately it is far off here, you can see yellow is turned here and main green goes slow and side road still red here.

(Refer Slide Time: 45:13)



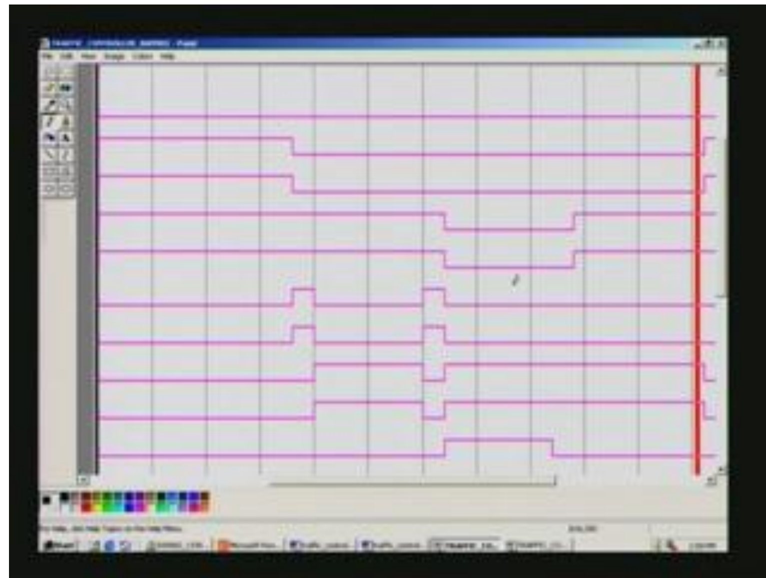

The next is 4. S0, S1, S2 states. That is here. You can see that these 2 are turned on. That is MRT. You have to turn right so MRT1, MRT2 are turned on. That is at this stage.

(Refer Slide Time: 49:25)



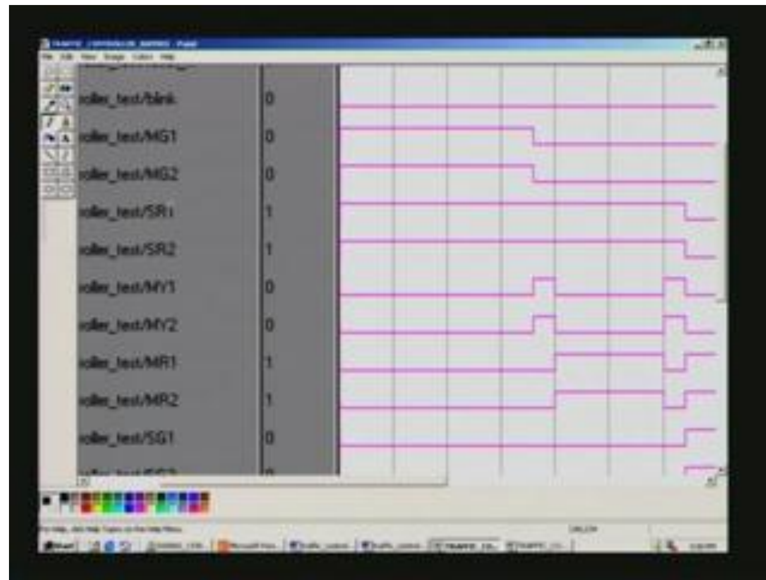
These are also turned on so this corresponds to red. SR1, SR2. That is this here. All of them and some more are here.

(Refer Slide Time: 45:53)

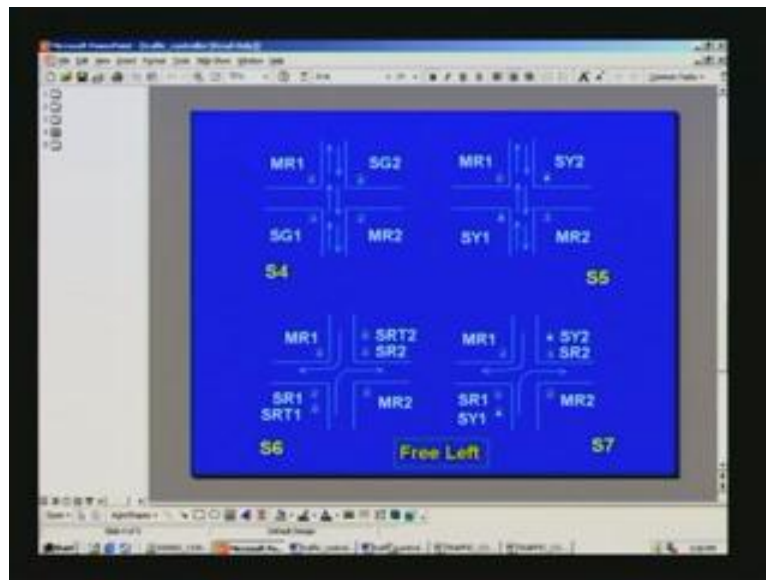


MR1 and MR2 are also turned on. Is this correct? Probably we look at the wrong thing it is a look here so only two are there I think see this is.., it is not S1 this one and MR one SR um both MR as well as SR MRT are there. The next one is main yellow side red here. Okay, let us have a look. That corresponds to 8. That is being yellow light here and it will be a small signal here so you can see this here, this four, so this four would mean this SR and MY that what we want SR and MY right, next time it should be MR and SG.

(Refer Slide Time: 46:57)

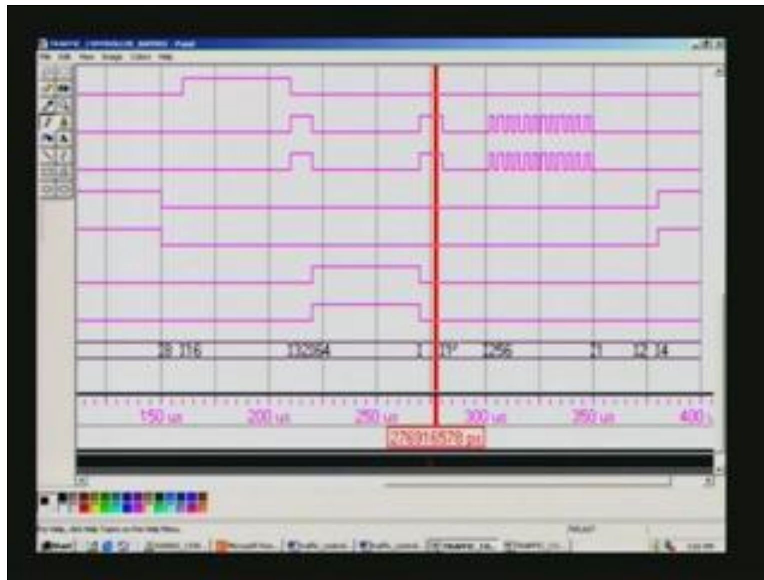


(Refer Slide Time: 47:07)




That I hope is 16 and MR and SG isn't so it is being high only here and here MR and SG, so this is 32 I will leave it as an exercise for you to look into. So is this case for sixty four then once again five seconds then it goes to one and we will examine that a blink here or one if you want.

(Refer Slide Time: 47:48)

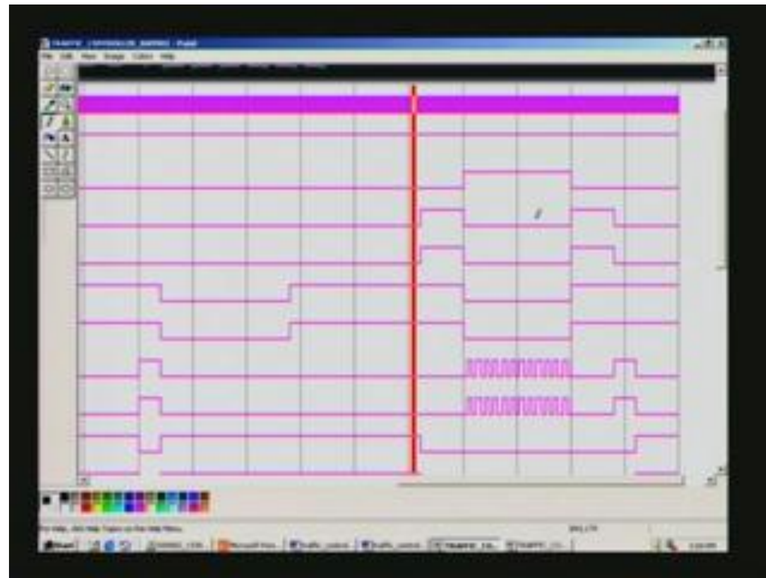


So in a one all are low except for this. So this two are there and this four are there. This is one twenty eight so one twenty eight state is S7. It is the right turn with yellow there: MR, SR, and yellow. Let us see. This is on here; let us see what is there. It is yellow and two more here that is this MR, there then should be one more MR. You have seen SY, SR did you see ?

Let us come from SR; that may be easy. SR is here. SR is 1 here; the cursor position is precisely that. Then it goes to S8 state, which is here. We have already seen that yellow lights are blinking here. It is all clean pulse. In fact, while doing this, I made a mistake in including the counter. I had explained earlier counter 1 as well as 5, so I had to take into account counter 1 too, but I did not take it into account. As a result, what happened is that there was a ringing here right at the raising edge of each pulse as well as at this edge. That was because this counter 1 is a much faster counter and each time counter 5 goes to 9, counter 1 can be anything. For all this, that particular frequency, I mean clock that is time base will come through, so they will see ringing there. It is not really ringing; that is a design flaw. Be on the lookout for such design flaws. It may not be matter much in traffic controller. For controller applications like programmable logic controller etc., it is vital that not a single unwanted spike must be there. This clarifies that the whole thing is working in back annotation. Then it goes to state 1 after this 1 once the blink is removed. You can see blink here, corresponding to

that blink is also high here. This is the blink here and it is withdrawn there so you can see right there as blink.

(Refer Slide Time: 51:22)



So this proves that the whole thing is working. Our design is working. We also have one more waveform here. This is just to tell you that, after back annotation, you have a delay here. This is the edge of the clock, which is really influencing the output. These are all some of the output.

(Refer Slide Time: 51:45)

You can just see the difference in delays here. Can you see the difference? See, this is much ahead when compared to this. Both are same but, once again, probably even ahead of this, so you cannot really predict how much delay will be in each of these signals because it has to come through several gates, and those gate delays get reflected as you would see when you finally put it on the FPGA board make it work it will be working almost with all this. Although this is a simulated signal, it is back annotated with the gate delays and that is how it is working here. So before we conclude this design example let us see the synplify results. We have given this operation 50 megahertz.  So what I have forgotten is did we cover the test bench? We did not, right? Okay, we will see this later. Thank you.

Summary of Lecture 43

(Refer Slide Time: 53:18)

Next lecture: System design examples (continued…)

(Refer Slide Time: 53: 38)