

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

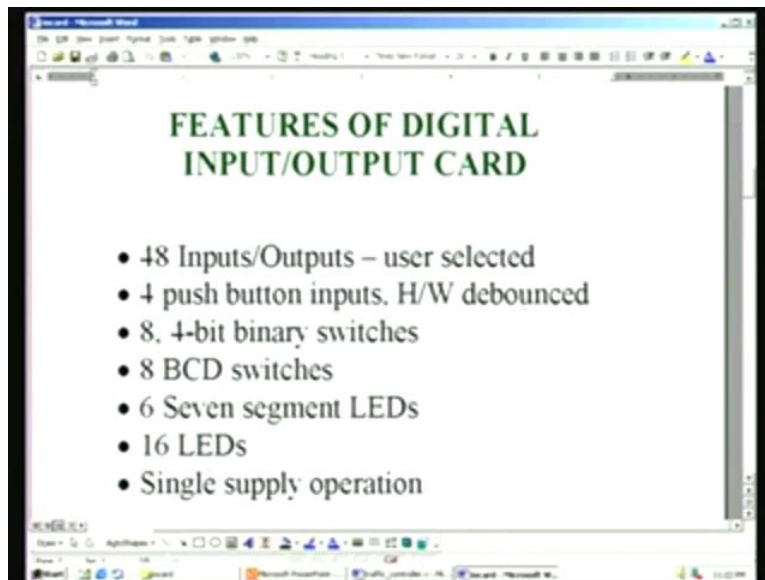
**Lecture No. 49**  
**System Design Examples using FPGA Board**

(Refer Slide Time: 01:37)





(Refer Slide Time: 02:23)

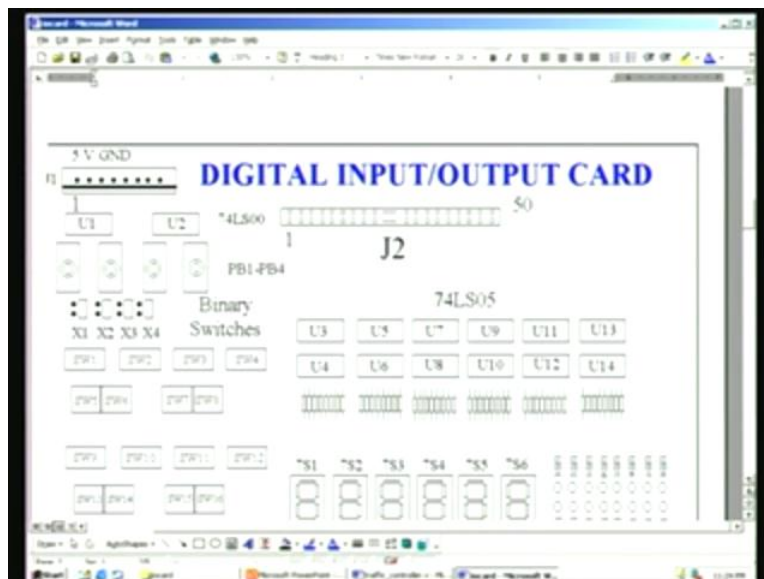


We have been looking into the design applications using the FPGA board. We also need different cards such as digital input/output card in order to increase the resources. For example, the features of digital input output card are as follows. This is the card that we are going to use in addition to the I/Os that are on the FPGA board. The first feature is it has a total number of 48 discrete inputs and outputs. Each of these inputs and outputs can be configured by the user and therefore, it is called user selected. In addition to this, we have four push button switches, each of which is hardware debounced. We use a 7400

NAND gate for debouncing this circuit. This is right there on the board. We also have eight four-bit binary switches and eight of BCD switches. This is very handy if you want to straightaway select in decimal values. For example, 0 through 9 are graduated on the BCD switches and you can be at home with decimal settings.

If you still insist on a binary setting, you can use this. In fact, all these eight binary and BCD switches are in parallel. You can select one of the two – one binary switch or one BCD switch, which is connected in parallel. You can select them in any combinations. In addition to this, we also have outputs. We have six seven-segment LEDs. They have a right decimal point display, which is not available in the FPGA board – we have already seen this earlier. In addition to these seven-segment LEDs, we have 16 discrete LEDs. These are all the LEDs that we are going to use in the first application, namely the traffic controller. This will be the very last of the six seven-segment LEDs. These seven segment LEDs and discrete LEDs are once again in parallel. This is parallel as far as the last two seven-segment LEDs are concerned. The whole board works on a single supply namely +5 Volts. We will have a look at the layout of the I/O card.

(Refer Slide Time: 04:56)



This is the I/O card. This card has a connector with 50 positions and a [05:16] male has been put on this board. Naturally, you need a female FRC connected to mate with this

and interconnect it to the FPGA board on the expansion headers. If you remember, we have two expansion headers. The cable will have to be bifurcated and then connected to the two expansion headers, since this is the only single connector at this end. We need to supply +5 Volts, which is done through the male connector on the board – this is called a Molex connector. Normally, this is popular for connecting power supplies.

The first two pins are +5 Volts and the next two pins are grounded. We also have a provision for 3.3 Volts here. Also, two or three crystal oscillators can be connected on the board, which is not shown here, because we are not going to use them for this particular application. In addition to this, we have four push button switches. This is a push button shown here (Refer Slide Time: 06:28). You also have four jumpers here. Right now, it is shown as a right-installed jumper but if you install the jumper on the left, it will automatically select this push button switch and it is debounced by using 74LS00.

You need cross-coupled NAND gates, which we will be covering later on. Out comes the debounced digital value, which is connected to pin number 3. This is the point we will have to connect. The second push button is connected to pin number 4; pin number 5 and pin number 6 are respectively used for PB3 and PB4. You see some more switches. All of them are connected in the very same order, left to right, like a raster scan order.

Parallel to these four bits, we have a switch SW1, which is a binary switch. There will be four discrete switches. The leftmost of the switches is parallel to this switch here and the second is parallel to this, third parallel to third position and fourth parallel to PB4. Parallel to this switch 1 is the BCD switch, which is SW5. This is graduated as I mentioned earlier from 0 through 9. You can use either the push button switch or this DIP switch for binary setting or the BCD switch for the decimal setting. All these are BCD switches – four here and four down here; all are symmetrical and very easy to find.

Similar to SW 1 to SW 4, there is also a group of binary switches SW9 through SW12 – four of them here. Since each has four bits, in order to cater to eight numbers, you need 32 bits. These 32 bits are connected in exactly the same order that you see into his fashion – first these four, which are parallel to these four and after this, SW4 is connected to SW9 again in the same order. Once again, SW13 is parallel to SW9 and so on. SW16

is parallel to SW12. This completes all the inputs. They would naturally take 32 bits and they are connected starting right from pin number 3 and counting 32, you will land up somewhere here. You can keep track of how many pins. It is in exactly in the same order that we have seen the switches positioned here.

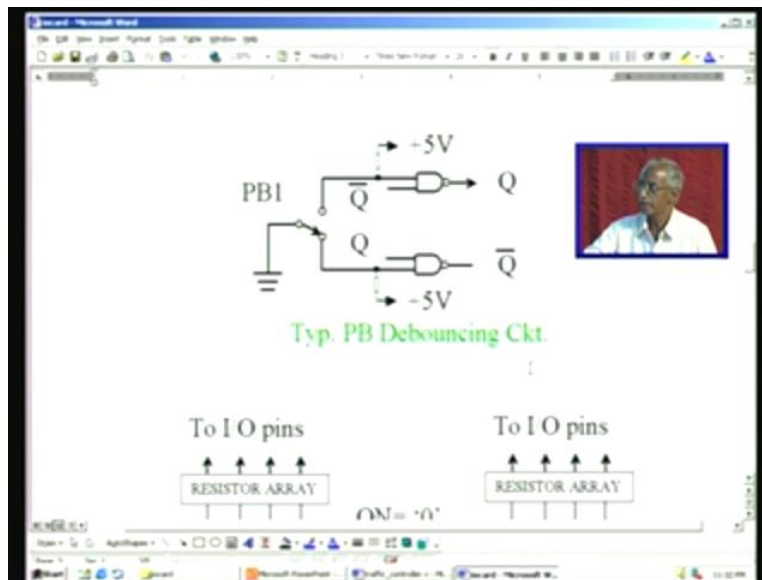
In addition to this, we also have these seven-segment LEDs connected to the very same pins. If you notice, there are six seven-segment LEDs. What is not shown in this figure is a decimal point, but we will be seeing that separately when we have a zoomed version later. Seven segments plus one for decimal point would take you to eight bits. Once again, this is connected from pin number 3 as it is in the case of push button switch or SW1 or SW5. The order in which this is connected is ABCDEFG and then decimal point, in that order starting from the left.

For instance, segment A is connected to pin number 3, segment B to pin number 4 and so on. After you have exhausted all the pins for this seven segment, which is eight in number, segment A for 7S2 will be connected immediately next to that and so on in the very same order, you will have it totally. If you see six seven-segment displays, you have actually have 6 into 8 including decimal point. That means 48 outputs are available here. If you notice this, we have a 50-pin connector of which we need for  $V_{CC}$  ground – +5 Volts is connected to pin number 1 and pin number 2 is ground. Pin number 3 starts with segment A and it goes on till the very end, 50. You can see that difference is 2 – if you knock them off, what you have is 50 minus 2, 48. That is precisely what you have here – 6 into 8 is 48 and it is the very same order.

In addition to this, we also have 16 discrete LEDs. For example, in this row, you can see LED1, LED 2 and so on right up to LED 8. Here starts LED9 through LED16. The first row LED1 to LED8 is connected in parallel to the last-but-one seven-segment display, that is, 7S5 is connected to this one in parallel. LED1 is connected to segment A, B segment is connected to LED2 and so on. So is the case for the seven-segment S6. A is connected once again to LED9, B connected to LED10 and so on. The decimal point will be naturally connected to LED16. These are all the resources that you need in order to do

any considerable amount of application that you wish to have. The resources on the FPGA board are clearly not adequate for this purpose.

(Refer Slide Time: 12:20)



Next what we will do is we will just have a look at the circuitry on the board. What we have here is a push button debouncing circuit. A traditional NAND gate is used. Instead of NAND gates, you can also use a NOR gate. Note the cross-coupling here. For example, Q bar is this output and Q is this top output. Q is connected as the input for the next NAND gate. This output Q bar is connected to the first NAND gate input. This is the traditional cross-coupling. The two other inputs of the NAND gate are connected to the push button switch. One of them is connected to the normally open connection and the other is connected to the normally closed connection. Note that to start with, when you do not push any push button, it will be returned to the ground, that is, this signal is forced to ground here.

Notice two pull-up resistors used here. For a pull-up resistor, a typical value used is a 4.7K or 10K – normally, these are all the industry standards. If you use a higher value, you cannot guarantee the threshold level – the high or low will be at the threshold level. Avoid going for very high values and do not go for [13:45] region as far as the pull-up resistor is concerned. The typical value as I mentioned is around 4.7K or 10K – this has

been proven in industries all over the globe. If you put a smaller resistor and if similar circuitry is involved, the current consumption will also be more. Accordingly, you can limit this by selecting 10K. Do not go beyond 10K. That is the price you may have to pay. Otherwise, you will get into trouble as far as the noise and performance are concerned; the level of 0s and 1s will also give a problem.

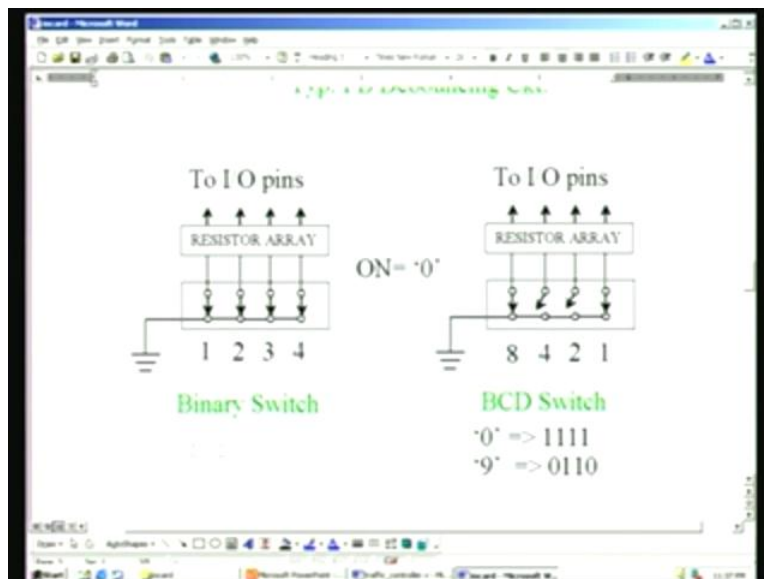
Let us see how the circuit works. To start with, this is 1 because it is pulled high and this is forced to 0. You know that 0 into anything is 0. Just before the bubble of the NAND gate, you get only 0 and after the bubble, you get 1. So Q bar is 1 for this particular position of this push button. Let us see what the case is. This Q bar is connected as the input – just remember it is 1 here. This is pulled high and so 1 into 1 is 1 before the bubble and after the bubble, it is a 0. That means it is 0 here and 1 here. Therefore, we have used the notation Q and Q bar because if it is 0, it will be 1, and if it is 1, it will be 0.

Right now, when the push button is not pressed, you get an output of 0. It is this that we will take as an output. Let us see what happens. When you press the push button, this connection is broken. It is going to travel all the way and strike at this point. Let us have a look at what happens before that. As the connection is broken here, this one, which was forced to 0, will be 1 and because this is not yet connected to this ground, this is also 1. So 1, 1 would mean store mode in SR flip-flop and that is precisely what we are exploiting here.

Remember it was 0 and 1. This 1 was fed here and this is still 1 only. 1 into 1 and naturally, the NAND output is 0. That is how the store mode works. In between, it is always in the store mode. You can analyze here also – since it is 1 here and Q was 0 there, it is naturally 1 here. So 0 and 1 are preserved, that is, it is in store mode. Now, when this contact is made here, this is forced to 0 for the time being. Once this is 0, this will be 0 here, therefore 1 here – the Q output has gone to 1. Obviously, I do not have to analyze this one because this will be the inversion of this. This has gone to 1 and this has gone to 0. Now what happens when the switch strikes here, it will bounce back because of elasticity – any contact that you make will bounce back.

This keeps on going for a while before it settles down. It will settle down because you are applying pressure to that. Before that, it will keep on making a number of makes and breaks here. When it is made 0 here, we have already seen it is 1. The very first time that you made a contact, even if it bounces back, it is in store mode and therefore, 1 is still preserved. That is how it is debouncing – the circuit debounces in a very simple manner. Like this, we have four push button switches. Each LS00 will have two such gates. We need two such LS00. That is the reason why we saw in the layout earlier two ICs for four push button switches.

(Refer Slide Time: 17:53)



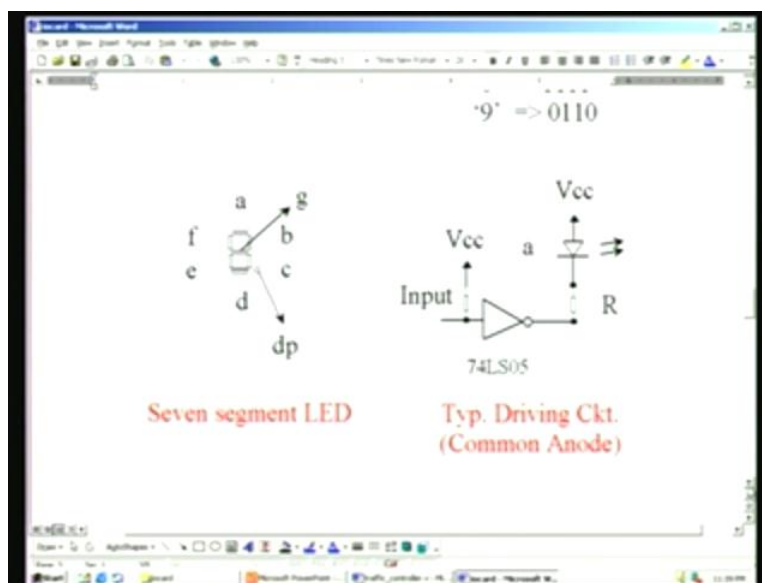
Here, for a binary switch, there are basically discrete switches – 1 to 4. They are all grounded here. The other end of the switch is returned to the I/O pins in the order that we have already discussed. Each of them is pulled high by the resistor array. So is the case for a BCD switch, which is also a resistor array. In fact, this resistor array and this are the same because all the switches are in parallel. The ON of a particular switch input will return to 0 for obvious conditions and it is grounded here. This will be pulled low and this is how you make a binary setting.

The BCD switch is also similar to this except that some inversion is involved. For example, if you set 0 on the BCD switch, the BCD switch is graduated from 0, 1, 2 and 3



right up to 9. You can take a small screwdriver and turn that to any desired position and you can thereby have a BCD or decimal setting. Notice that for 0, you will be reading all 1s, which means that all the switches will be off. All 1s are read because of the pull-up resistor array here. If it is 9, it will actually be reading 0110 because it will make a contact corresponding to 1001, which is the usual thing you can see – 1001 means making a contact is 1 here but in terms of logic, it will be inversion because of this configuration. That is the reason it is inverted here. You know that this is nothing but 8421 code. One can easily derive what you have to put here but just remember to invert this. These conditions are required if you are to recode, which may be given as an assignment towards the end.

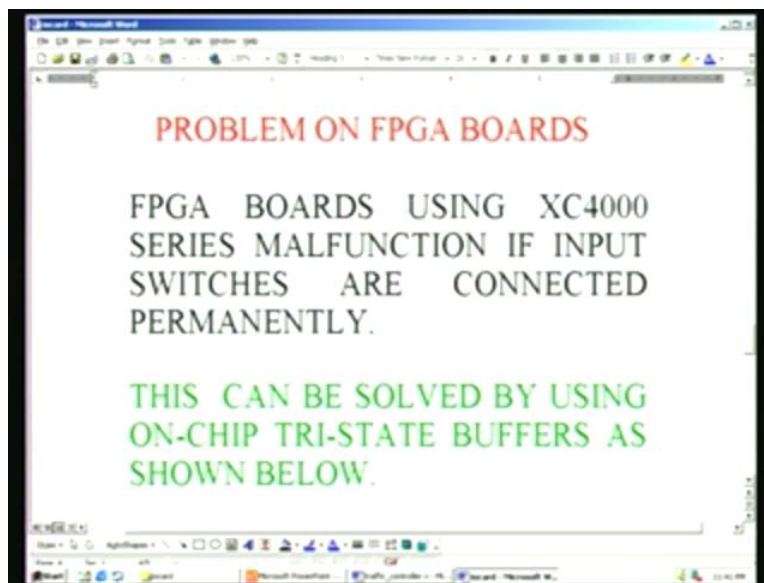
(Refer Slide Time: 19:43)



Coming to this, a seven-segment LED is shown here. The segments are a, b, c, d, e, f and this is the last one. In addition, there is also a decimal point. The center one is g. The typical driving circuit for each of these LEDs or for that matter even the discrete LED is quite simple. You have a 74LS05 and that input is once again connected from the I/O pins that we have already seen – 48 of them. This is once again pulled high by a pull-up resistor there. When it is not connected, 1 will be pumped in here because of this open collector output, which can sink current. The LED is connected in this fashion. The seven-segment LED that we have used on the board is a common anode type. That is why

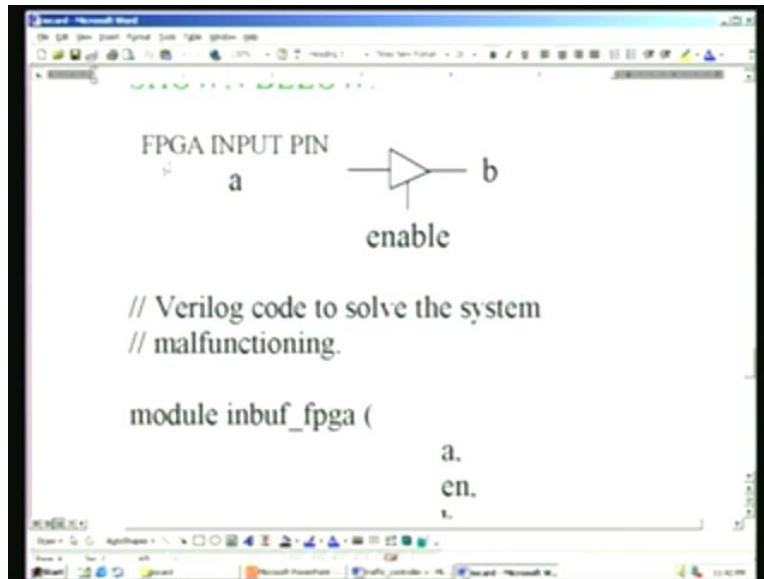
all the seven segments are combined together as far as the common anode is concerned. When it is 1 here, it will be 0 here. This is the current-limiting resistor. Naturally, the LED will conduct. The typical resistance is around 200 Ohms, which you can easily find. The conducting LED will take about 1.5 Volts and this will be 0.3 Volts with reference to the ground, which means that the drop of 1.8 Volts is accounted for by the +5 Volts supply and the difference will appear across R. Just allow for some 10 milliamps to 20 milliamps. Typically, 10 milliamps is enough to light up the LED.

(Refer Slide Time: 21:20)



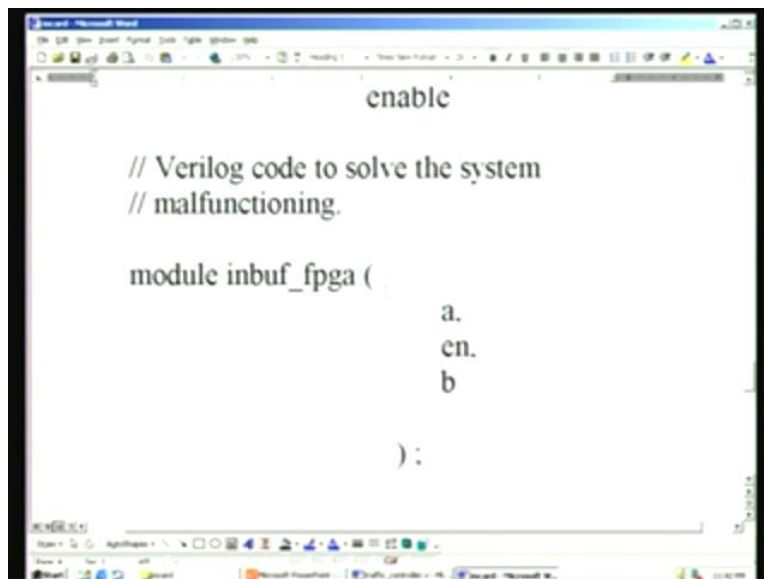
We will be showing one of the FPGA boards. In some of the boards, especially based on XC4000 series, there are problems associated. We have experienced a practical problem in the course of development of different applications. The problem is FPGA boards using XC4000 series malfunction if input switches are connected permanently. We have already seen that. In digital I/O, we have directly connected the switches. They are to be used with the FPGA board XSV-800, that is using XCV-800 type of FPGA. This problem is not encountered with that. This problem is only in the XC4000 series. The way to overcome this problem is by using on-chip tri-state buffers as shown below.

(Refer Slide Time: 22:21)



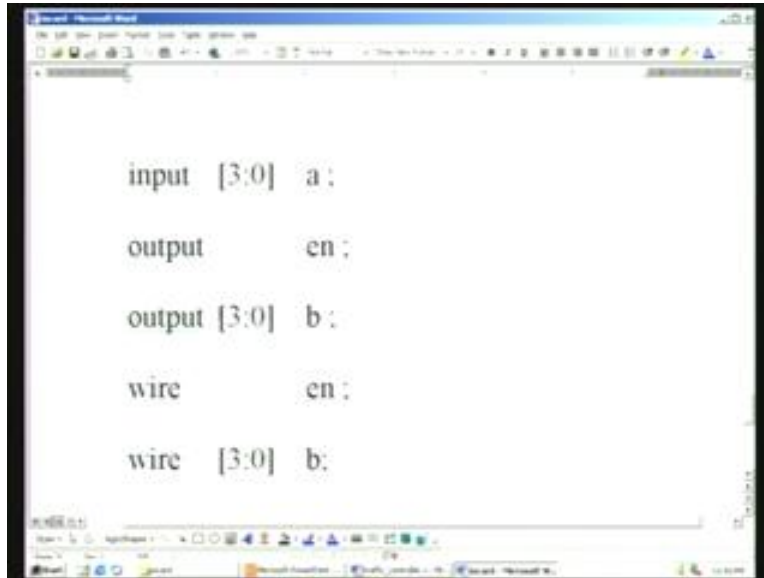
This is the FPGA input pin. Here, you may be connecting a DIP switch. In such a case, if you do not use this tri-state buffer and straightaway connect to the FPGA pin, when you switch on and download the bit stream, the system really goes mad. It does not function well and so it malfunctions. In order to overcome that, we have either to use an external tri-state buffer or a better option would be to use the on-board resources. The corresponding Verilog code for the same is given here.

(Refer Slide Time: 29:59)



It is a very simple Verilog code. This is the module declaration, a and b are what we have already seen. enable is the enable for this of course, it is bigger here you can remove this, actually it is enabled there. When enable is on, then a will be communicated to b. a will be connected to a DIP switch or BCD switch, thereby actually connecting it to the FPGA pin after the tri-state buffer.

(Refer Slide Time: 23:35)

A screenshot of a Microsoft Word document window. The document contains the following Verilog code declarations:

```
input [3:0] a;  
output      en;  
output [3:0] b;  
wire      en;  
wire [3:0] b;
```

The code is centered on the page. The window title bar shows "Microsoft Word" and the taskbar at the bottom shows various system icons and the Windows Start button.

The code for this is here. Input output declaration is here. We have used only four bits here. You will have to use as many number of bits you require. For a four-bit binary switch, four is enough. If you have multiple bits, accordingly you will have to increase this. Instead of three, you can increase it to as many as you want.

(Refer Slide Time: 23:52)

```
output      en ;
output [3:0] b ;
wire        en ;
wire  [3:0] b ;
assign en = 1 ;
assign b = en ? a : 0 ;
```

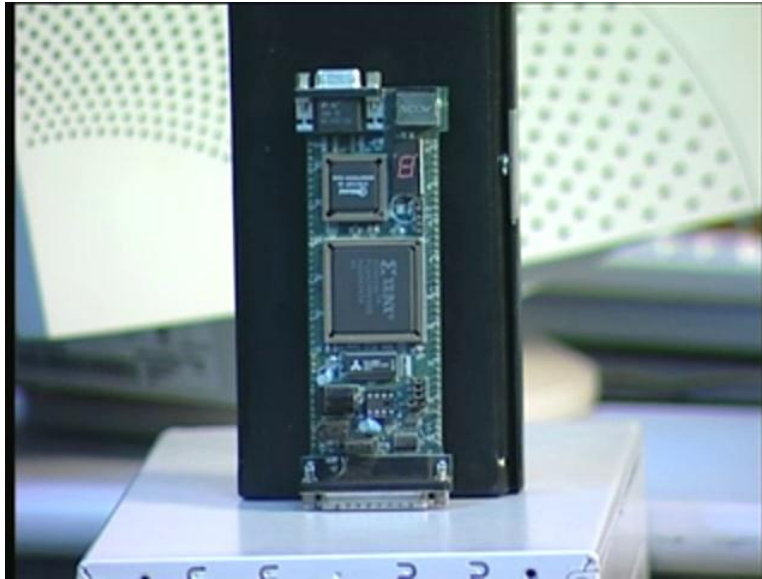
```
wire  [3:0] b ;
assign en = 1 ;
assign b = en ? a : 0 ;

endmodule
```

We are going to use only assign statements. They are declared as wire. There are two simple instructions that will do the trick. For example, we enable first here and the concurrent statement is another MUX here. This is the output assign statement. b is derived from either the actual DIP switch, which is connected to the input, or it is forced with 0. It is forced with 0 if it is in disable mode – if enable is 0, 0 will be forced to b. This is at the time of configuring. When you configure the bit stream, this is the case normally because when you switch on, all the FPGA pins are normally cleared.

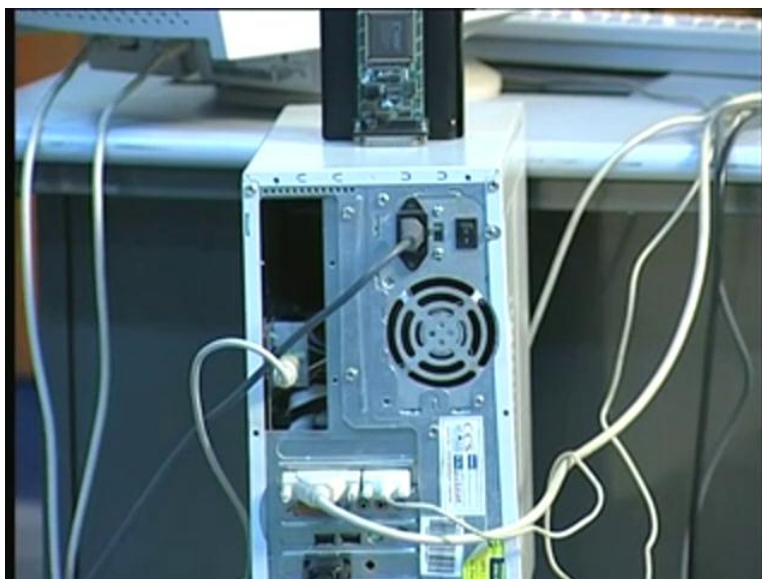
Therefore, we have written it this way. Very simple, just two codes in essence will easily solve the problem that we have explained earlier. This is the end module.

(Refer Slide Time: 24:57)



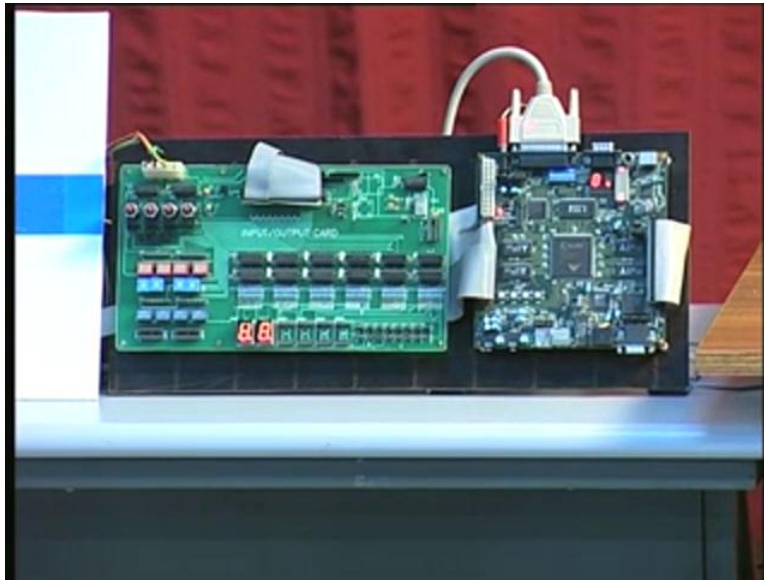
We have seen this 4000 series board, wherein a problem is encountered. This board is shown here. This is the Xilinx chip. This is actually XC4010. We have seen and analyzed how to solve the problem associated with this.

(Refer Slide Time: 25:28)



Next, we will see the traffic controller demo using different boards that we have already considered, namely, the FPGA board as well as digital I/O card. We need to connect this FPGA card to a parallel port. That is shown here. This is the parallel port connection. From here, it trails there and gets connected to the FPGA board right on the top here (Refer Slide Time: 25:45).

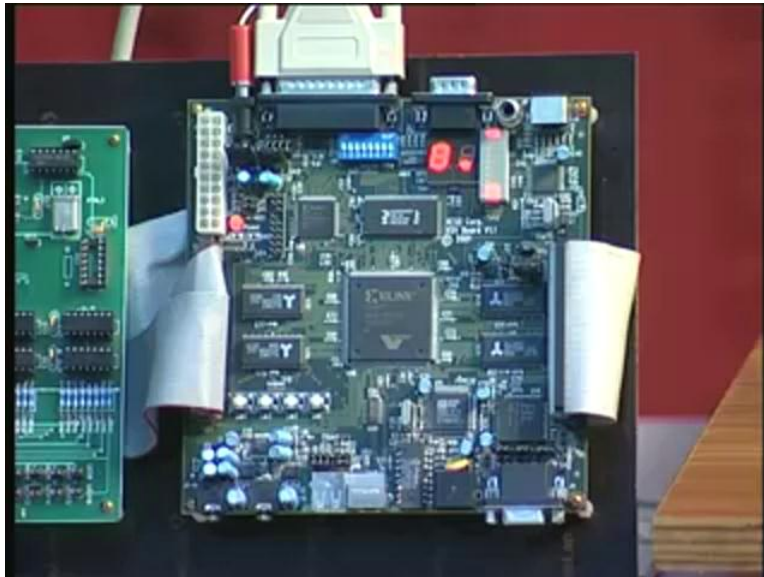
(Refer Slide Time: 25:46)



This is the connection.

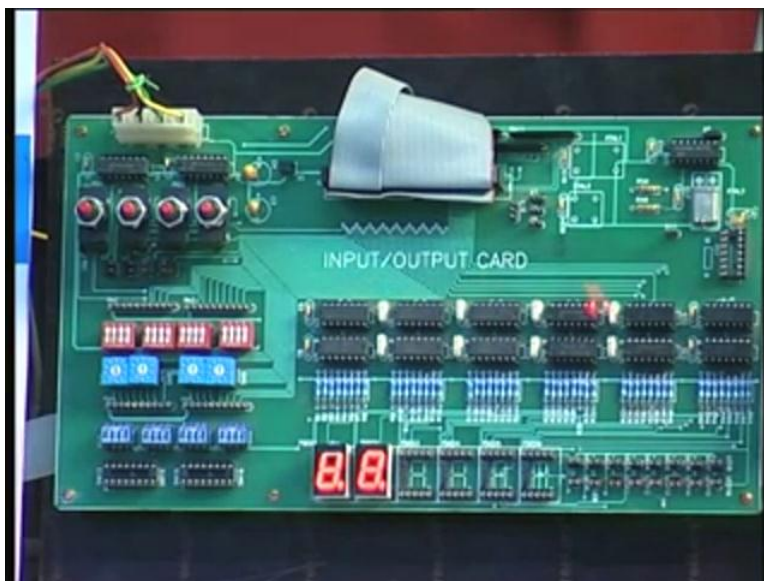


(Refer Slide Time: 25:49)



This is the FPGA board. A zoomed version is available right now. The main FPGA is here. This is an XCV-800 device. We have already seen descriptions of all the other things. These are all the flat cable connection expanders. These cables are connected to the digital IO card, which is next.

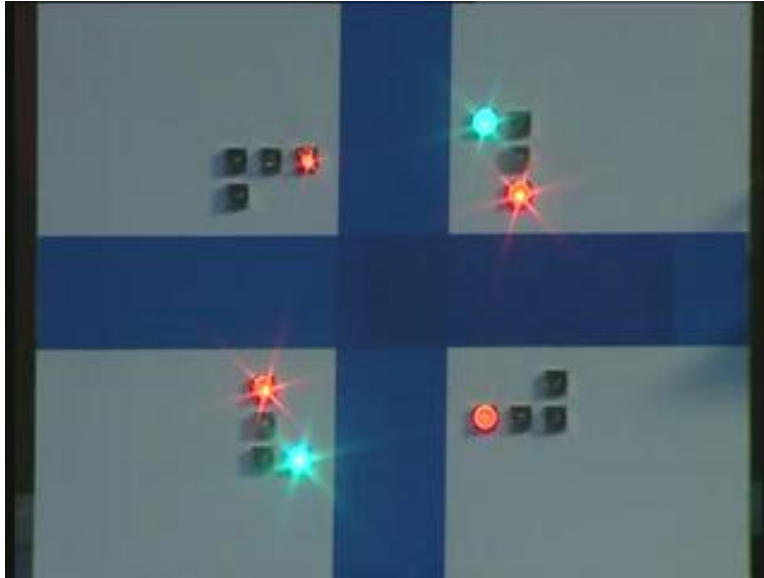
(Refer Slide Time: 26:17)





You can have a look at a zoomed version. The cable connection is right here. We have already seen the 50-pin connection. This is the BCD switch. It is graduated as 0, 1, 2, 3 up to 9. Eight such provisions are available, eight DIP switches are available and push button switches are here. Now we will see the working of the traffic controller. Right now, it is on. You can see the traffic controller board.

(Refer Slide Time: 26:43)

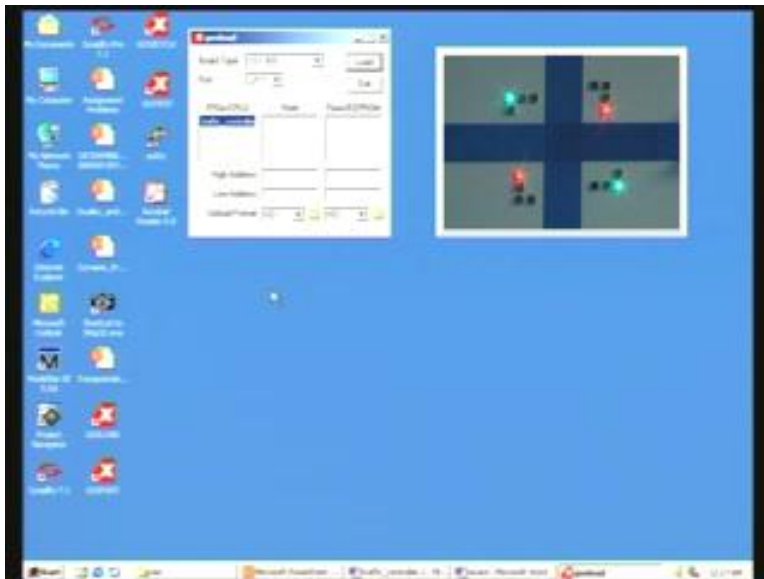


The zoomed version is here. Just have a look at part of this sequence for a while.

(Refer Slide Time: 27:00)

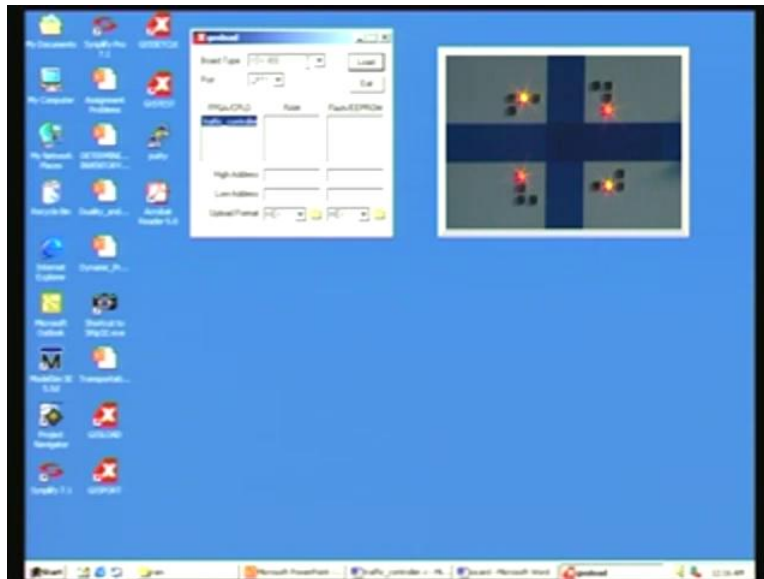


(Refer Slide Time: 27:03)



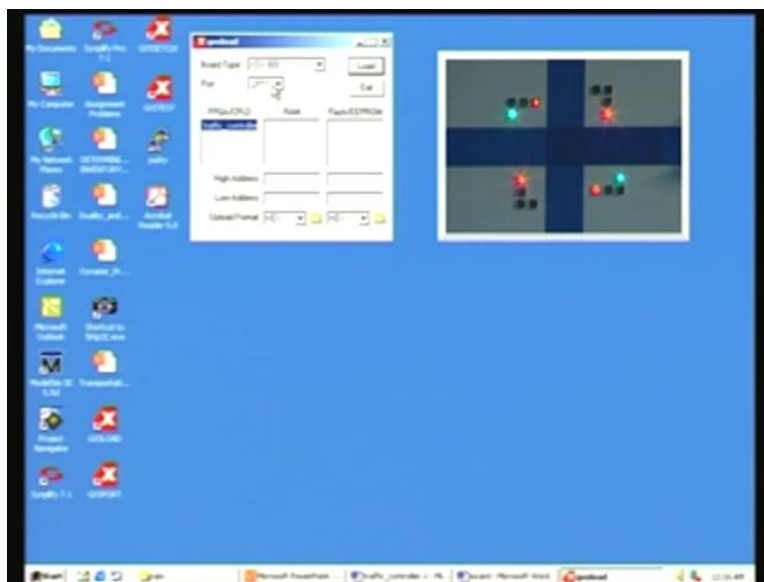
The traffic controller bit stream is loaded by using **GXSLOAD**. We have already seen this in the manual covered earlier. The board type XSV-800 is selected here.

(Refer Slide Time: 27:15)



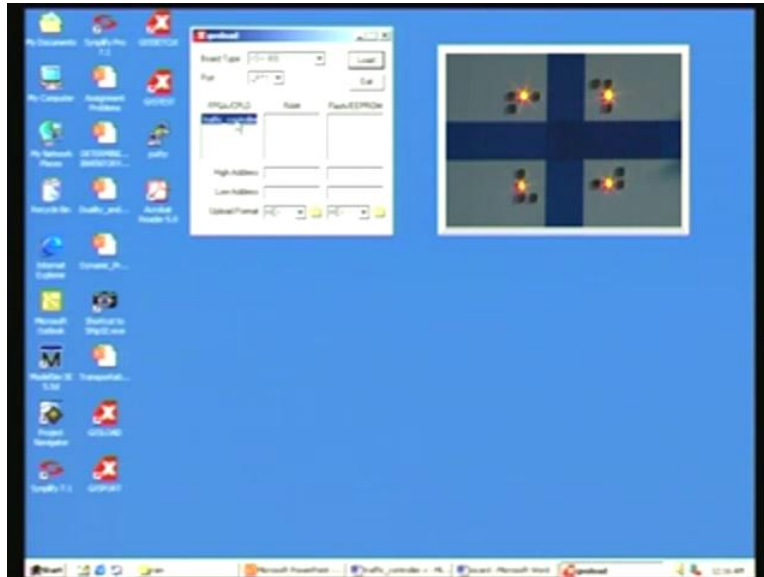
The connection is an LPT 1 parallel port and that is what is put here.

(Refer Slide Time: 27:20)



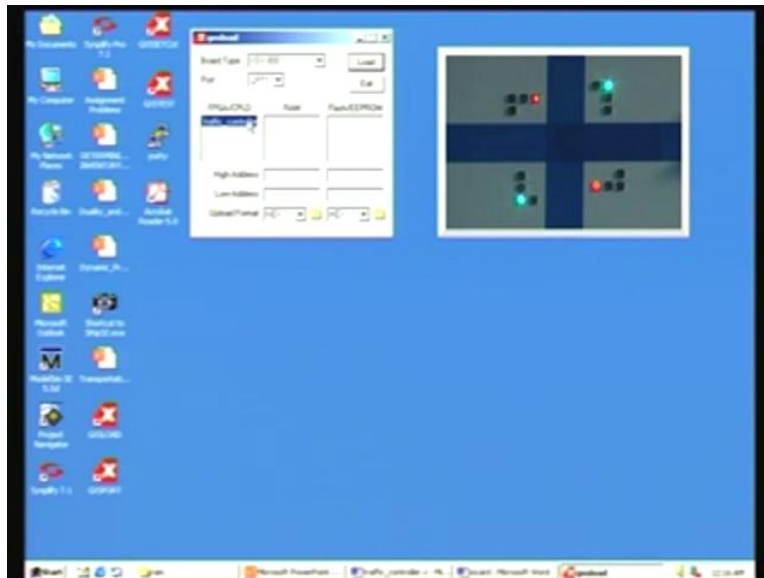
The bit stream that you need is for example, traffic\_controller – it is **actually a .bit**.

(Refer Slide Time: 27:29)



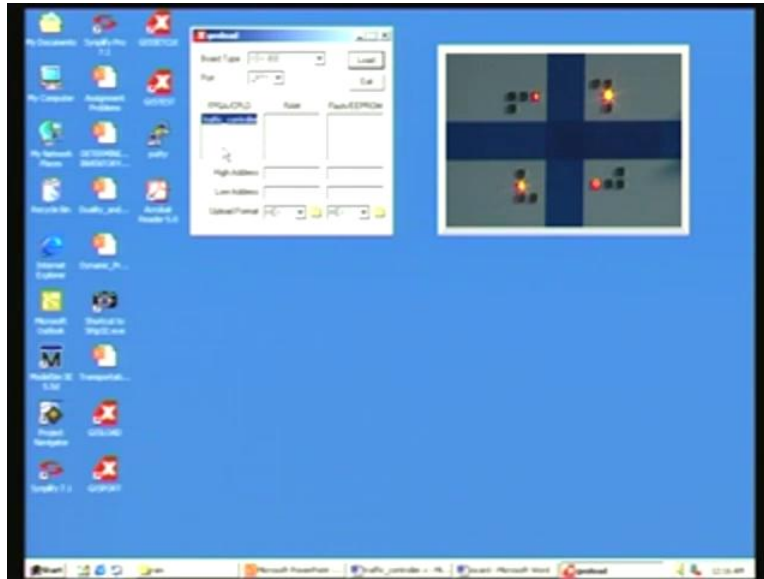
We have to get this field by merely copying and dragging from the folder in which it is available.

(Refer Slide Time: 27:37)



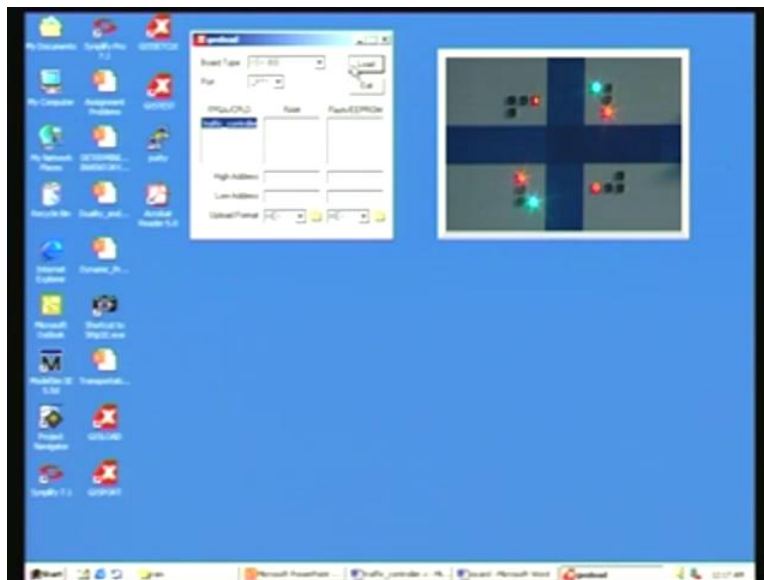
This is what we have already seen in the previous design of a traffic controller.

(Refer Slide Time: 27:45)



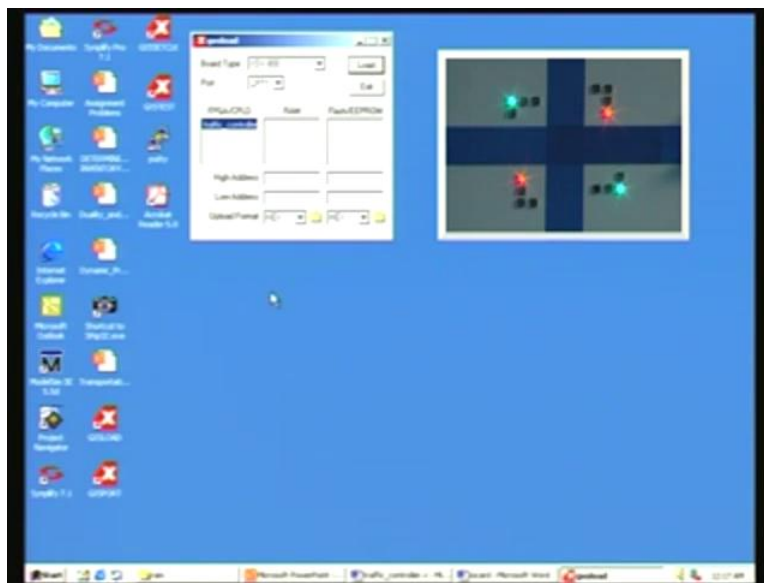
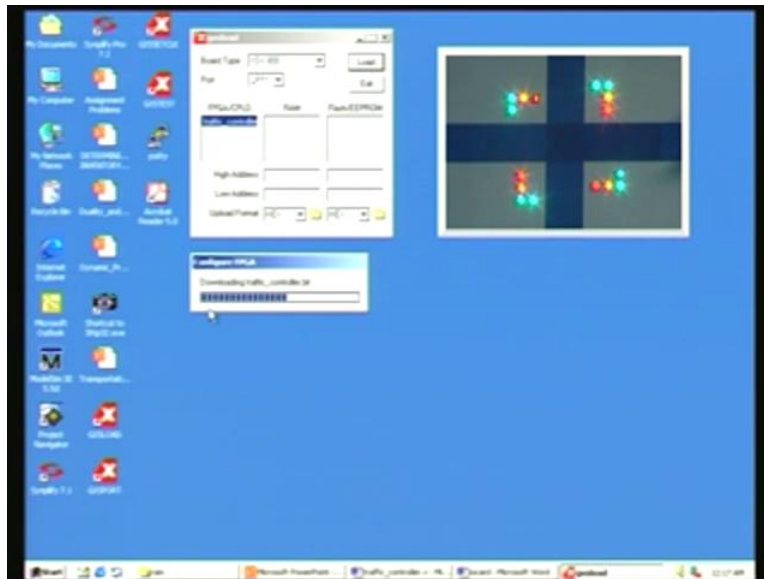
Once you have got the desired file on this, you have to load that.

(Refer Slide Time: 27:48)



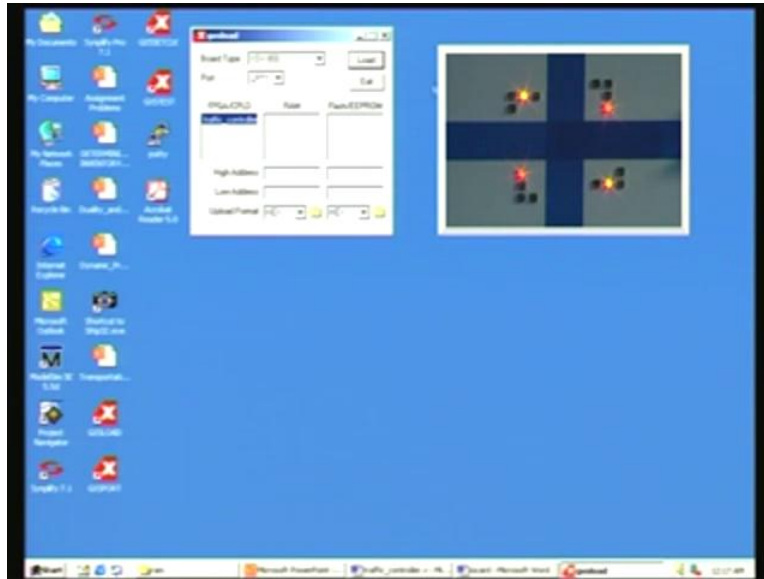
This is done by pressing this button on the right side. Watch what happens while doing the load.

(Refer Slide Time: 27:53)



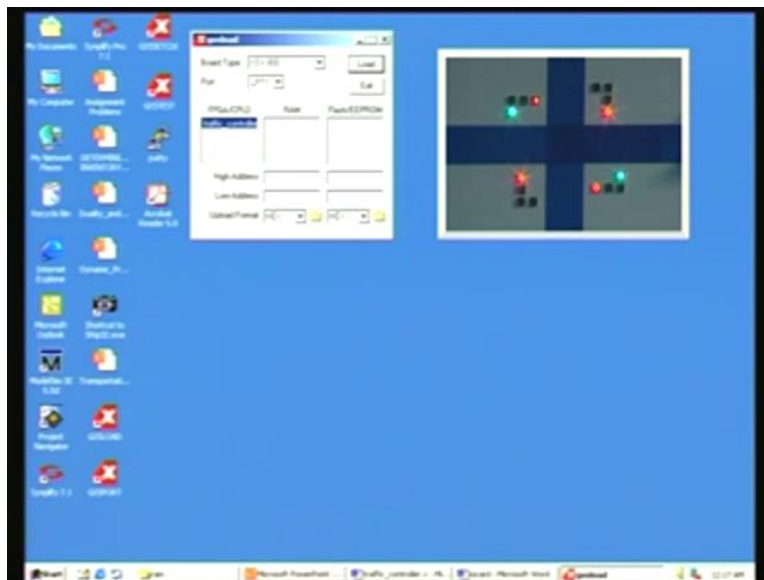
You see another small window opened down and it is actually loading the bit stream here. If you see that, it is traffic\_controller.bit. Immediately, what did you see here? It has initialized the traffic controller display starting with this is the main road, right from here left to right.

(Refer Slide Time: 28:13)



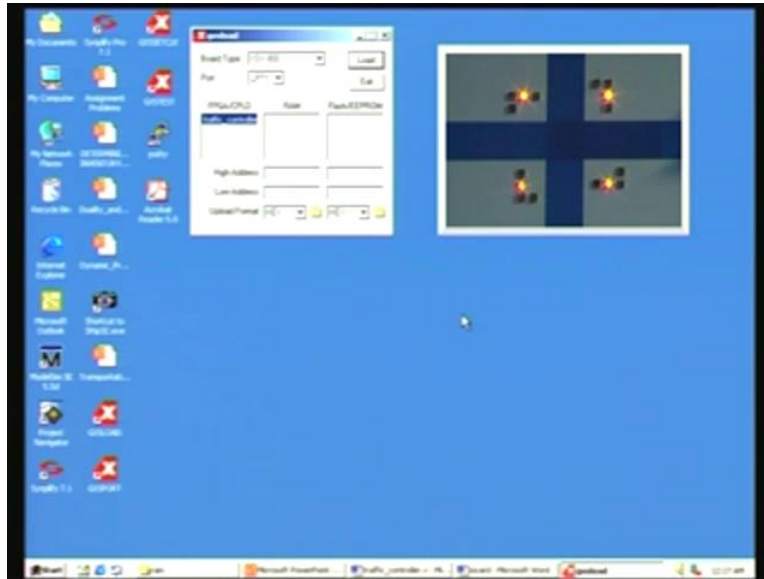
What you saw was a green transiting to yellow.

(Refer Slide Time: 28:18)



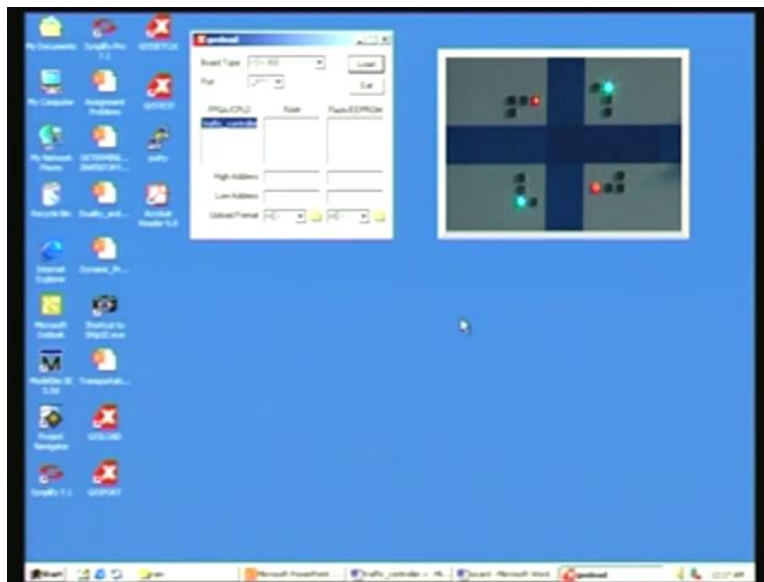
The vertical road is the side road. You can see the entire sequence.

(Refer Slide Time: 28:30)



Earlier in the design, we have used green for 45 seconds for the main road.

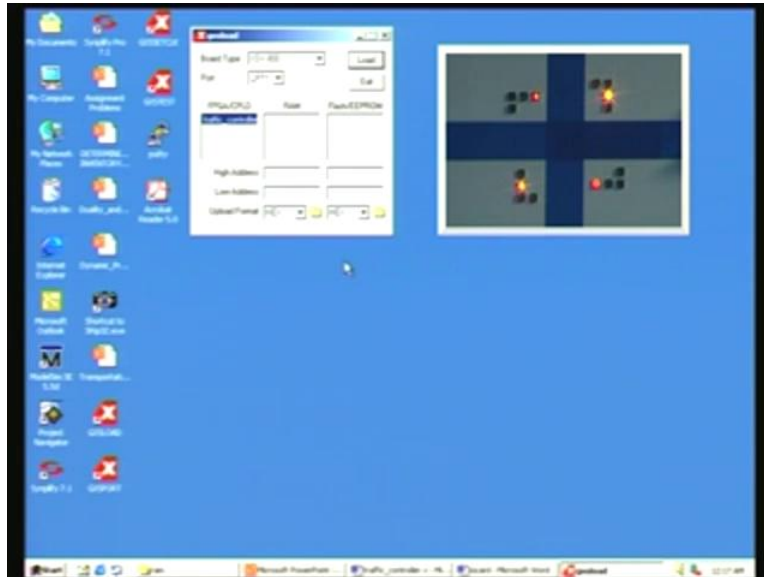
(Refer Slide Time: 28:32)



But this has been lowered to 15 seconds in order to get a good demo here.

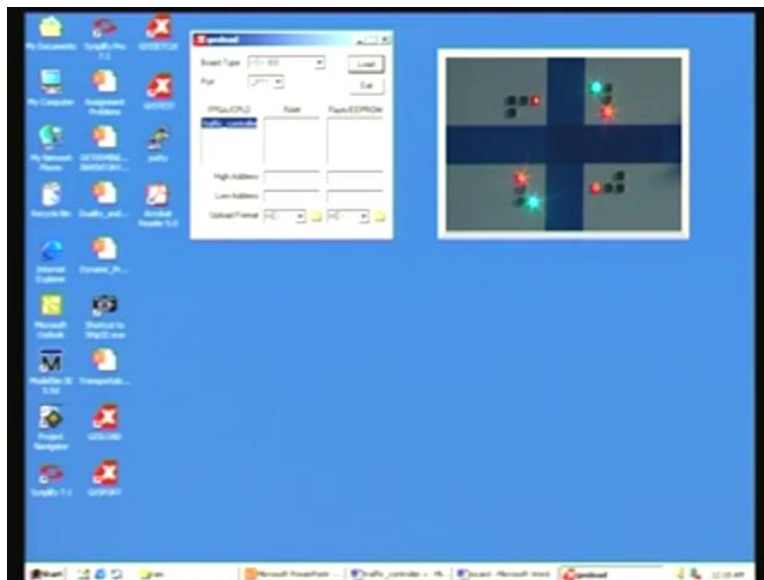


(Refer Slide Time: 28:42)



You can probably keep track of how much time it actually takes for each of them and then make sure that they are really working.

(Refer Slide Time: 28:47)

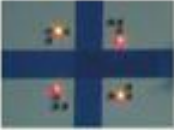


This will be there in one of the corners all the time. Now, let us see what we have in the traffic controller.

(Refer Slide Time: 29:03)

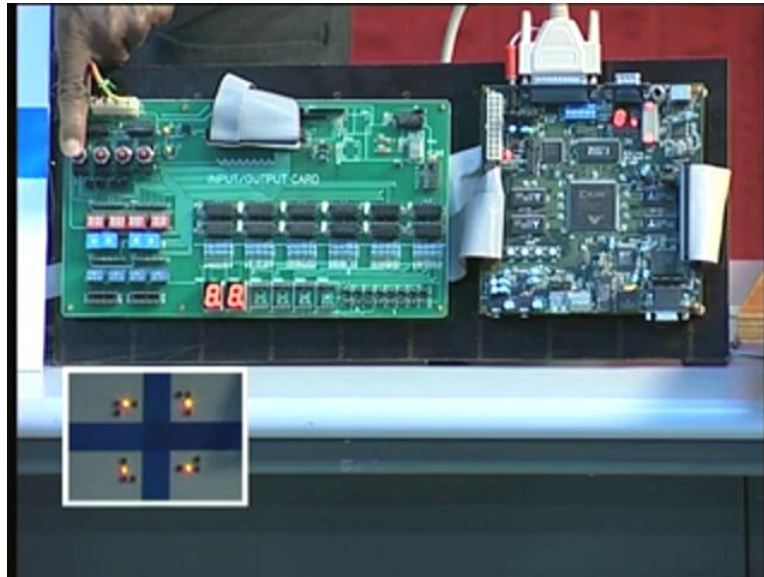
```
/*  
  
VERILOG RTL CODE FOR TRAFFIC LIGHT CONTROLLER  
  
This is the top design module -  
  
Design file: traffic_controller.v.  
  
This controls the traffic lights of a four-  
road junction with Pedestrian Crossing.  
  
The timing for the main road traffic is  
assumed to be 15 Secs. And that for the side
```

```
-----  
  
This controls the traffic lights of a four-  
road junction with Pedestrian Crossing.  
  
The timing for the main road traffic is  
assumed to be 15 Secs. And that for the side  
road, it is 10 Secs. Yellow lights will be on  
for 5 Secs. when they are switched on.  
  
The design has includes Pedestrian  
so free left turn cannot be consid
```



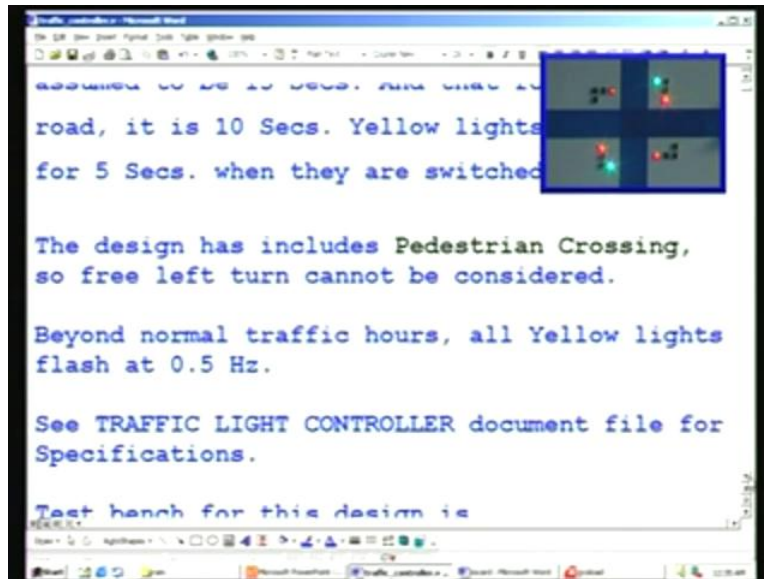
This is the traffic controller that we are already familiar with. I will not go through the detailed description of the Verilog code, which we have already seen earlier. What we have done in this is added some more features – pedestrian crossing, which was given as an assignment earlier, is actually put in this code. This is not in the demo that you are seeing. While running through the normal sequence of the traffic lights, you also have the provision for pressing a blink control. You see PB1 through PB4 and one of that is used, namely PB 1. If I press this, just watch what happens to the display.

(Refer Slide Time: 29:55)



You can see that all the yellow lights are blinking as long as this is switched on. This is equivalent to blinking of the lights during the night for cautious driving. When you release, it reverts to the very first sequence that you see there. We will see the traffic light controller code amended to cater to blinking covered in every state instead of just one state that we have covered earlier. That is one change that has been incorporated in this revised traffic controller design. In addition to this, we had given an assignment for pedestrian crossing and that has also been incorporated in this. This controls the traffic lights of a four- road junction with pedestrian crossing. We earlier had 45 seconds for the main road traffic, which has now been amended to 15 seconds and 10 seconds for the side road. The other things are the same – yellow lights for 5 seconds and blinking at 1 second rate. This is what we have already seen.

(Refer Slide Time: 31:19)



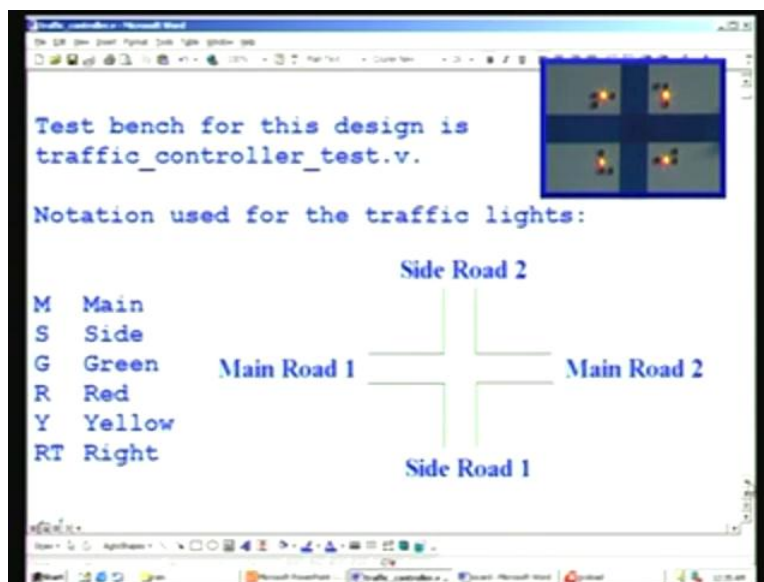
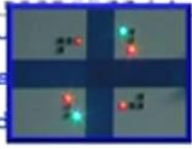
assumed to be 10 Secs. And that to  
road, it is 10 Secs. Yellow lights  
for 5 Secs. when they are switched

The design has includes Pedestrian Crossing,  
so free left turn cannot be considered.

Beyond normal traffic hours, all Yellow lights  
flash at 0.5 Hz.

See TRAFFIC LIGHT CONTROLLER document file for  
Specifications.

Test bench for this design is



Test bench for this design is  
traffic\_controller\_test.v.

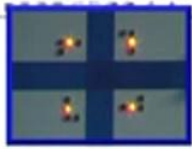
Notation used for the traffic lights:

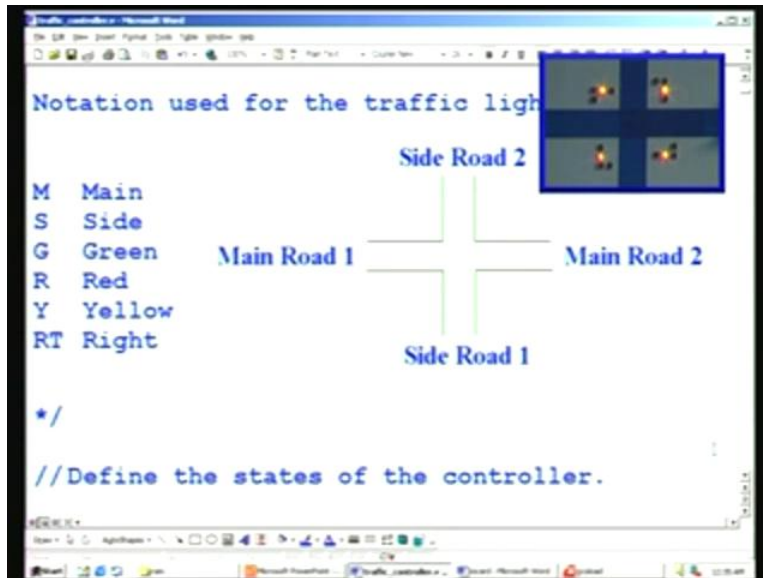
M	Main
S	Side
G	Green
R	Red
Y	Yellow
RT	Right

Side Road 2

Main Road 1      Main Road 2

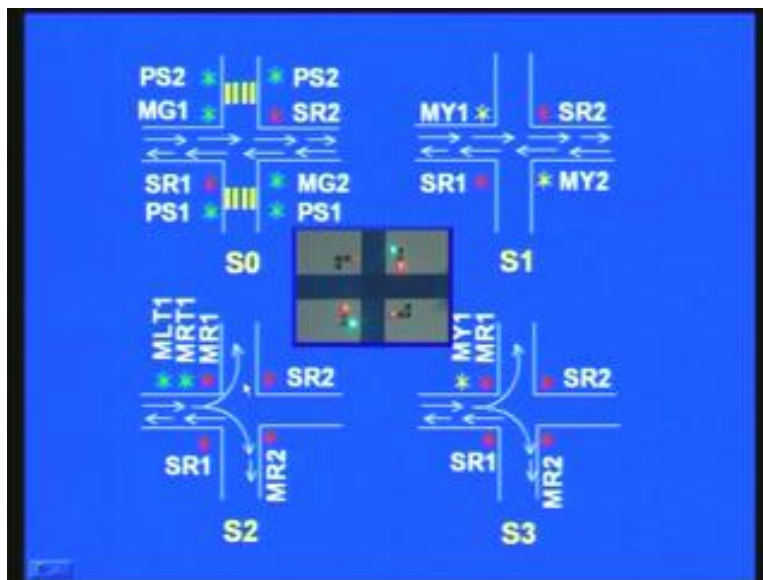
Side Road 1





This is the road here. Before that, let us have a look at the PowerPoint presentation for the same with the amended code.

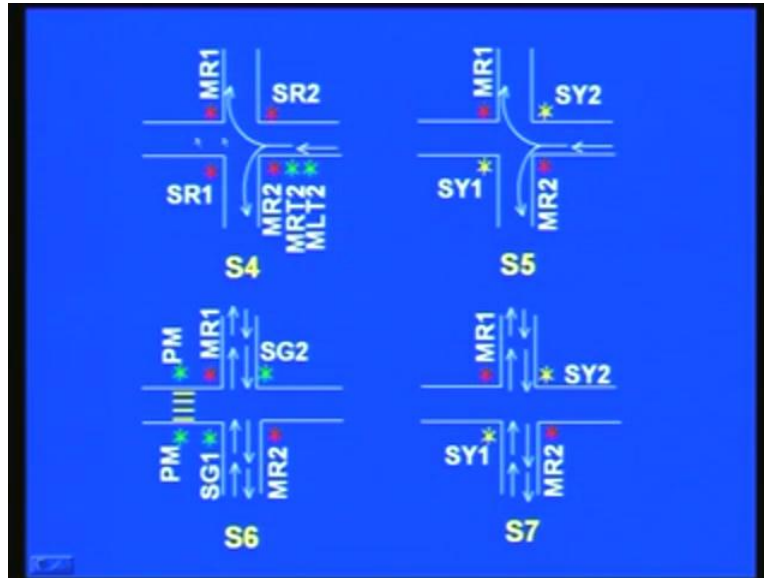
(Refer Slide Time: 31:25)



To start with, what we have is the straight-flowing traffic with MG1 green, MG2 green. While this happens, we can also allow the pedestrian to cross over here. PS stands for the pedestrian signal. 1 and 2 is the same nomenclature we have adopted earlier – 1 for this side road and 2 for the other side of the side road. S stands for the side road. You have this signal, which is ON. The next sequence is that it converts into yellow. When that

happens, we have removed the pedestrian crossing because it is much safer before the oncoming traffic, which is S2. It happens to be the left turn as well as the right turn here. The corresponding signals are all lit here. We now have a left turn – this is an extra addition here. This was already there. Once again, it will go through the yellow as far as main traffic is concerned – that is the S3 sequence.

(Refer Slide Time: 32:37)



In the S4 sequence, the reverse for the same main road happens – right and left traffic is allowed here in this case. Once again, you can see the left indicator here. Once again it goes through the yellow sequence, in which case SY1 and SY2 are on here because it is going to be green here in the next sequence – SG1 as well as SG2. That is why we have provided this. In case you are not happy, you are free to change because it is a question of changing a few Verilog codes.

Here, you can see that the main pedestrian crossing is lit, this signal is lit, allowing the pedestrian to cross because this going to be straight traffic. In addition to this left, we can also provide here. In fact, we can duplicate the same thing here also if you want this crossing – you do not need any extra signals for that. Next is once again the yellow traffic transition here.

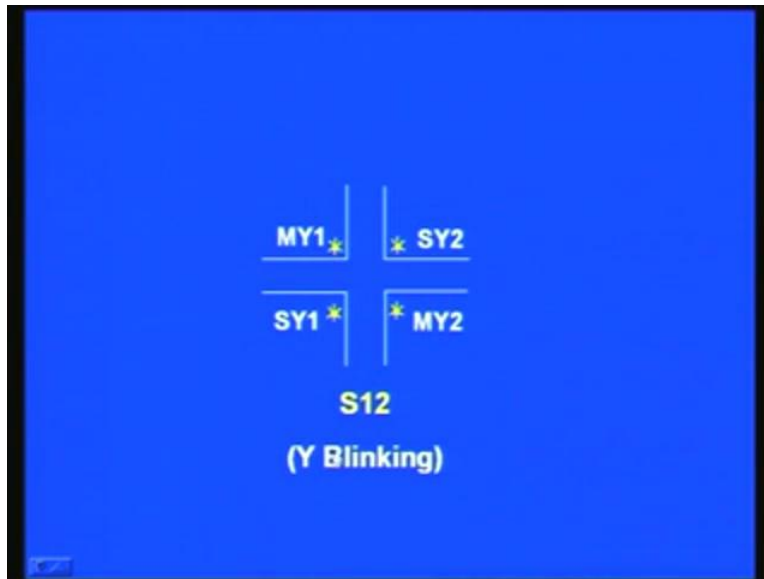
(Refer Slide Time: 33:43)



After this, we have the S8 sequence, which will be the same case for left and right traffic from the side roads. Side road 1 is allowed here and once again, yellow transition. In this case, yellow is given only for this because next is going to be on this side – right traffic as well as left traffic flow as far as side 2 is concerned. All the other things I do not have to explain here – red and whatever is to be done has been done. As far as this is concerned, both left flow as well as the right traffic signal must be ON. Once again, it goes through the yellow in this sequence.



(Refer Slide Time: 34:29)



Next is the very first sequence here (Refer Slide Time: 34:36). That is why yellow has been marked for the main here. S12 gives the blinking. Earlier if I remember correctly, it was S8 state and now, it is S12 state because we have added some more states in between, instead of eight plus one, nine states were there earlier, I think. Is it starting from S0? S0 to S11 is the normal sequence, that is twelve sequences plus one more sequence for the blinking.

(Refer Slide Time: 35:21)

```
`define S0 4'd0
`define S1 4'd1
`define S2 4'd2
`define S3 4'd3
`define S4 4'd4
`define S5 4'd5
`define S6 4'd6
```



```
`define S7 4'd7
`define S8 4'd8
`define S9 4'd9
`define S10 4'd10
`define S11 4'd11
`define S12 4'd12

`define time_base 22'd4999999
```

```
define S11 4'd11
`define S12 4'd12

`define time_base 22'd4999999

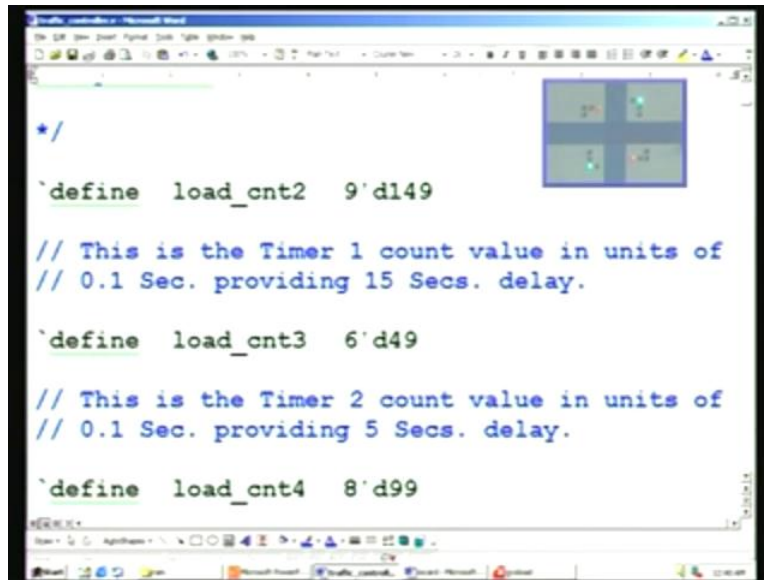
/*
22'd9 is used only for simulation purposes.
For 50 MHz operation, the time base is 0.1
Sec.

Change 22'd9 to 22'd4999999 for 50 MHz
operation.
```

Reverting to the code, these are all precisely the same definitions and we are not going to repeat it. Wherever there is a change, we are going to cover that. We had a time base for 40 Megahertz earlier, if I remember correctly. Now, what we have is 50 Megahertz because this is what is demanded by the [35:39] board. As I mentioned earlier, there was a divisor we programmed and that was in terms of 1, 2, 3 and so on we had put a divisor of 2. That means we get from 100 Megahertz, which is the basic clock frequency. A factor of 2 will give you 50 Megahertz and because of the board requirement, we change

its frequency, for which, instead of 3999 for 40 Megahertz, we need to update it to 499. That is all the change here.

(Refer Slide Time: 36:12)

A screenshot of a code editor window showing Verilog code. The code defines three timer constants: load\_cnt2 (9'd149), load\_cnt3 (6'd49), and load\_cnt4 (8'd99). Each definition is followed by a comment explaining its purpose in terms of timer counts and delay. The code is as follows:

```
*/  
  
`define load_cnt2 9'd149  
  
// This is the Timer 1 count value in units of  
// 0.1 Sec. providing 15 Secs. delay.  
  
`define load_cnt3 6'd49  
  
// This is the Timer 2 count value in units of  
// 0.1 Sec. providing 5 Secs. delay.  
  
`define load_cnt4 8'd99
```

We also have to change the timing. This is for the 15 seconds timer – always 1 count less, as we have discussed before. 0.1 second is the time base. Therefore, the decimal point is reckoned here. So it is actually 15; 14.9 means 15 because this is 1 count less. 0, 1, 2, 3 is the count and that is the reason for 1 less. So is the case for 5 seconds, 4.9 here.

(Refer Slide Time: 36:42)

```

`define load_cnt3 6'd49

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

`define load_cnt4 8'd99

/*
   This is the Timer 3 count value in units of
   0.1 Sec. providing 10 Secs. delay.
*/

`define load_cnt5 8'd9

```

Similarly, for the side road you need 10 seconds so it is 9.9.

(Refer Slide Time: 36:15)

```

/*
   This is the Timer 3 count value in units of
   0.1 Sec. providing 10 Secs. delay.
*/

`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.

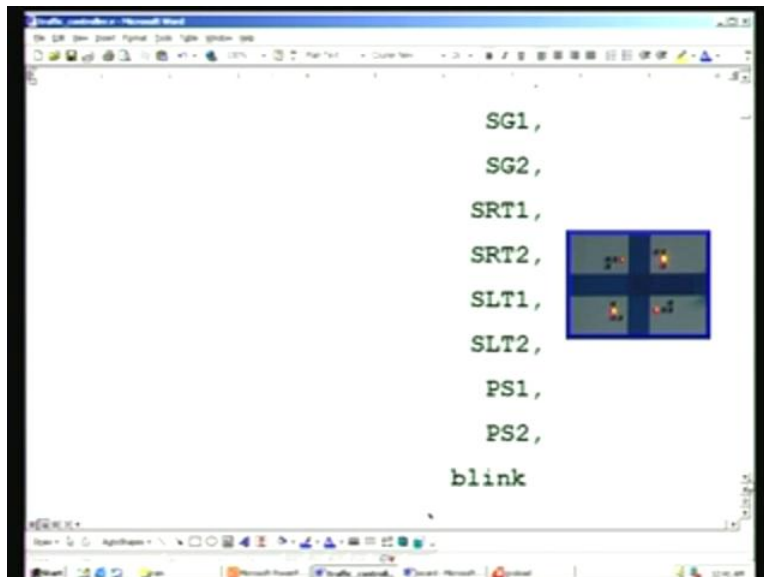
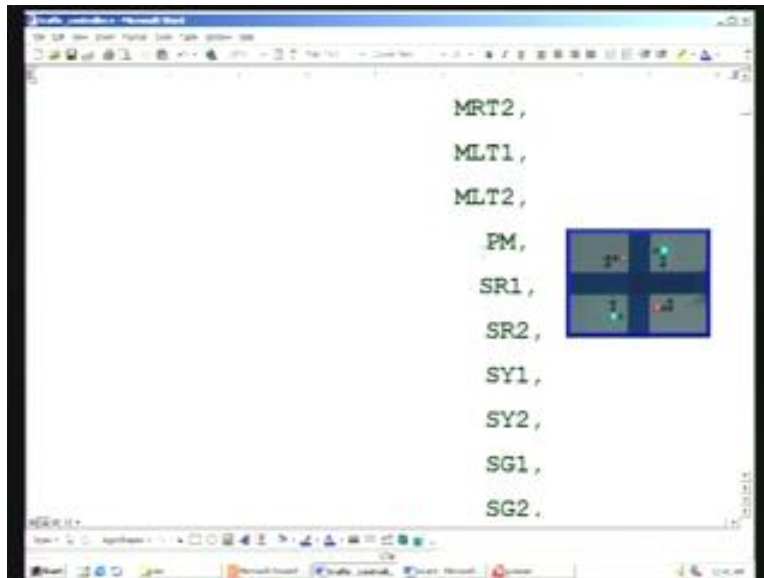
```

For 0 through 9 it goes as far as the basic time base is concerned, which gives a 0.1 second delay – that is here, for which we need instead of 10, 9 actually – 1 less. So this will give you a 0.1 second delay.

(Refer Slide Time: 37:04)

```
module traffic_controller (  
  
    clk,  
    reset_n,  
    MR1,  
    MR2,  
    MY1,  
    MY2,
```

```
    MY1,  
    MY2,  
    MG1,  
    MG2,  
    MRT1,  
    MRT2,  
    MLT1,  
    MLT2,  
    PM,  
    SR1.
```



These are all the declarations here. For example, left turn extra here and pedestrian main. Similarly, pedestrian side and similarly left are all extra here, then pedestrian signal. Then blink happens to be the very same thing.

(Refer Slide Time: 37:25)

```
// Declare Inputs/Outputs.

input    clk ;

input    reset_n ;

input    blink ;

output   MR1 ;// Traffic lights main green,
          // etc. are declared as outputs.
output   MR2 ;
```

```
output   MR1 ;// Traffic lights main green,
          // etc. are declared as outputs.
output   MR2 ;

output   MY1 ;

output   MY2 ;

output   MG1 ;

output   MG2 ;

output   MRT1 ; //Outputs for Main roads
```

```
output MG1 ;
output MG2 ;
output MRT1 ; //Outputs for Main roads
output MRT2 ;
output MLT1 ; //Outputs for Main roads
output MLT2 ; //left introduced.
output PM ; //Output for Main roads
//Pedestrian crossing
```

```
output MG1 ;
output MG2 ;
output MRT1 ; //Outputs for Main roads
output MRT2 ;
output MLT1 ; //Outputs for Main roads
output MLT2 ; //left introduced.
output PM ; //Output for Main roads
//Pedestrian crossing
```

Once again, we have I/O declarations and once again, you see this – left output and pedestrian main.

(Refer Slide Time: 37:38)

```
output S12 ;
output SG1 ;
output SG2 ;
output SRT1 ;
output SRT2 ;
output SLT1 ; //Outputs for Side roads.
output SLT2 ;

output PS1 ; //Outputs for Side road
```

```
output SG2 ;
output SRT1 ;
output SRT2 ;
output SLT1 ; //Outputs for Side roads.
output SLT2 ;
output PS1 ; //Outputs for Side road
//Pedestrian crossing.
output PS2 ;
```

Once again left for the side, then pedestrian crossing output. The wire happens to be the same thing, then for timer.



(Refer Slide Time: 37:57)

```
reg MLT1;  
reg MLT2;  
reg PM ;  
reg SR1 ;  
reg SR2 ;  
reg SY1 ;  
reg SY2 ;
```

```
reg SG2 ;  
reg SRT1;  
reg SRT2;  
reg SLT1;  
reg SLT2;  
reg PS1;  
reg PS2;
```

Once again, we have to declare registers. This is pedestrian main and once again, left side turn as well as pedestrian. They have to be declared as reg.

(Refer Slide Time: 38:16)

```
/*  
  
Timer implementation starts here.  
  
cnt1_reg is a free-running counter to  
provide the time base of 0.1 Sec. for timers  
1 thru' 3.  
  
*/  
  
assign cnt1_next = cnt1_reg + 1 ;  
  
        // Pre-increment the counter.
```

This is essentially the same. This is to create 0.1 second time base. These codes are exactly the same. You cannot take it left, you can see the first.

(Refer Slide Time: 38:40)

```
/*  
  
this is the timer 1, programmed for 15  
secs. in order to facilitate the smooth run  
of the main road traffic.  
  
*/  
  
assign adv_cnt2 = (start_timer_1 == 1'b1)  
                &(cnt1_reg == `time_base) ;  
  
// Condition for Pre-incrementing the counter.  
  
assign res_cnt2 = (cnt1_reg == `time_base)  
                &(cnt2_reg == `load_cnt2);  
  
// Condition for resetting the counter.
```

Although the first character is appearing on the monitor, it is not appearing on the screen. I hope it is visible on the TV. Anyway, we are not really interested in the actual. We have already covered these codes and so we need not really worry about that. I am not

describing the code. This is exactly the same thing; I am pointing out only the difference. For example, it was 45 seconds but now, the comment is for 15 seconds.

(Refer Slide Time: 39:12)

```
/*
This is the Timer 2, programmed for 5 Secs.
(activating Yellow lights) for the smooth
transition while switching from one traffic
to another.
*/
assign adv_cnt3 = (start_timer_2 == 1'b1)
                  &(cnt1_reg == 'time_base) ;

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

This is the timer 2, catering to yellow lights, for 5 seconds. There is no change here.

(Refer Slide Time: 39:21)

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

else

    cnt3_reg <= cnt3_reg ;

    // Otherwise, don't disturb.

end

// This is the Timer 3, programmed for 10
// Secs. delay, used for the side road
```

This is for 5 seconds.

(Refer Slide Time: 39:29)

```
else if (adv_cnt4 == 1'b1)
    cnt4_reg <= cnt4_next ;

// 10 Secs. timer - Advance the count once if
// the timer is still running.

else
    cnt4_reg <= cnt4_reg ;

// Otherwise, don't disturb.

end
```

This is for 10 seconds here.

(Refer Slide Time: 39:33)

```
// 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.

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

This is for 1 second for blinking. The same thing continues. Different counters are running, which we have already seen.

(Refer Slide Time: 39:48)

```
always @ (posedge clk or negedge reset_n)
begin
    if (reset_n == 1'b0)
    begin
        // Switch OFF all lights to start with.
        MR1    <= 1'b0 ;
        MR2    <= 1'b0 ;
        MG1    <= 1'b0 ;
    end
end
```

This is the initialized condition. This is also the same. Wherever the extra sequences are involved, we will cover that.

(Refer Slide Time: 39:58)

```
case(state)
`S0:
    if (blink == 1'b1)
        state    <= `S12 ;
        // Change to the blink state.
    else
        begin
            MG1    <= 1'b1 ; // switch ON main
            // green lights and
        end
endcase
```

The first one is also the same.

(Refer Slide Time: 40:05)

```
// green lights and
MG2   <= 1'b1 ;

SR1   <= 1'b1 ; // side red lights.
SR2   <= 1'b1 ;

PS1   <= 1'b1 ; // switch on side
        // pedestrian lights.
PS2   <= 1'b1 ;

// Switch OFF all other lights not wanted.
MR1   <= 1'b0 ;

MR2   <= 1'b0 ;
```

Here, side pedestrian is included. You can cross when the main green lights are on. We have turned PS1 and PS2 to 1 here. All other lights are exactly the same – we will see that all through, in each of the extra sequences that we have added.


(Refer Slide Time: 40:28)

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

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

SLT1  <= 1'b0 ;
SLT2  <= 1'b0 ;

if (res_cnt2 == 1'b1)
```



For example, we have to turn off an unwanted left signal and that is what we are doing here.

(Refer Slide Time: 40:35)

```
if (res_cnt2 == 1'b1)

// This corresponds to 15 Secs. timing of
// timer 1.

begin

start_timer_1 <= 1'b0 ;

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

state <= `S1 ;
```

This is for the 15 seconds timer.

(Refer Slide Time: 40:41)

```
if (blink == 1'b1)

state <= `S12 ;
//Change to the blink state.

else
begin

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

MY1 <= 1'b1 ;
MY2 <= 1'b1 ;
```

Hereafter, after every sequence, we will be noticing that we have included the blink state, so that we do not have to wait till the end of all the sequences, which we have done before. In this case, we are taking at every sequence – S0, S1, etc. We are sensing the blink at all sequences.



(Refer Slide Time: 41:06)

```
// Switch OFF side pedestrian lights and all
// other lights not wanted.


PS1  <= 1'b0 ;
PS2  <= 1'b0 ;

MG1  <= 1'b0 ;
MG2  <= 1'b0 ;
MR1  <= 1'b0 ;
MR2  <= 1'b0 ;
```

Once again, you can see this pedestrian is not wanted here and therefore turned off.

(Refer Slide Time: 41:14)

```
MRT2 <= 1'b0 ;
MLT1 <= 1'b0 ;
MLT2 <= 1'b0 ;
PM   <= 1'b0 ;
SY1  <= 1'b0 ;
SY2  <= 1'b0 ;
SG1  <= 1'b0 ;
SG2  <= 1'b0 ;
SRT1 <= 1'b0 ;
```



Then main pedestrian is also turned off here.



(Refer Slide Time: 41:29)

```
end

`S2:
  if (blink == 1'b1)

    state    <= `S12 ;
    //Change to the blink state.

  else

begin
```

That was for two sequences, S0 and S1. This is the S2 sequence. Once again, blink control is taken into account. If blink is 1, then it will go to the last state – S12 state, which happens to be the blink state. This is what we have already encountered earlier also in the other two sequences.

(Refer Slide Time: 41:52)

```
right, main road1 left and side red lights

MR1    <= 1'b1 ;

MR2    <= 1'b1 ;

MRT1   <= 1'b1 ;

MLT1   <= 1'b1 ;

SR1    <= 1'b1 ;

SR2    <= 1'b1 ;

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

In this particular sequence, we have MLT1 coming into the picture as well as MRT1.

(Refer Slide Time: 42:04)

```
SR2      <= 1'b1 ;

// Switch OFF all other lights not wanted.
MY1 <= 1'b0 ;

MY2 <= 1'b0 ;

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

MRT2 <= 1'b0 ;
```

```
// Switch OFF all other lights not wanted.
MY1 <= 1'b0 ;

MY2 <= 1'b0 ;

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

MRT2 <= 1'b0 ;
MLT2 <= 1'b0 ;

PM      <= 1'b0 ;
```

Naturally, we need to switch off all other unwanted lights, including MLT2 here and the main pedestrian crossing.

(Refer Slide Time: 42:16)

```
SG1 <= 1'b0 ;
SG2 <= 1'b0 ;
SRT1 <= 1'b0 ;
SRT2 <= 1'b0 ;
SLT1 <= 1'b0 ;
SLT2 <= 1'b0 ;
PS1 <= 1'b0 ;
```

```
SRT2 <= 1'b0 ;
SLT1 <= 1'b0 ;
SLT2 <= 1'b0 ;
PS1 <= 1'b0 ;
PS2 <= 1'b0 ;

if (res_cnt4 == 1'b1)

// This corresponds to 10 Secs. timing of
```

Again, SLT1 and SLT2 for left traffic of the side road, as well as pedestrian crossing for the side road.

(Refer Slide Time: 42:34)

```
state `S3:
    if (blink == 1'b1)
        state <= `S12 ;
        //Change to the blink state.
    else
        begin
            // Switch ON main red, main road1 yellow light
            and
```

Once again, for S3, we have a blink control here.

(Refer Slide Time: 42:39)

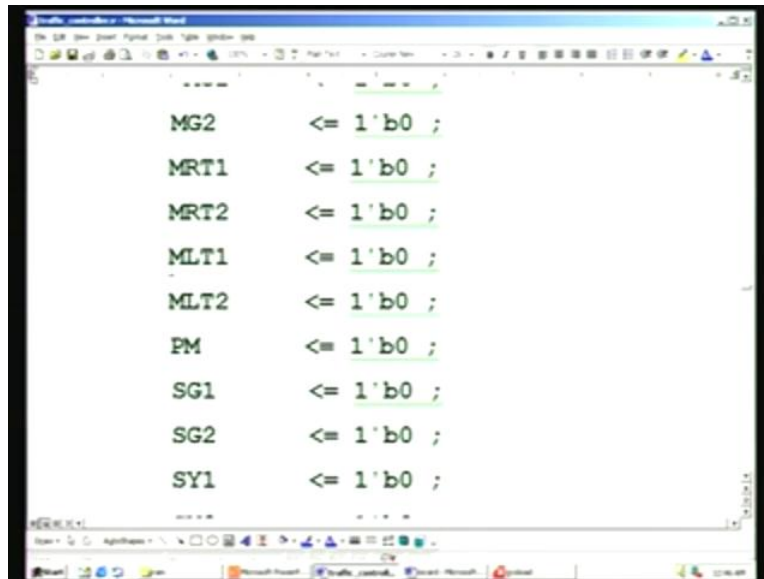
```
// side red lights.
MR1    <= 1'b0 ;
MR2    <= 1'b0 ;
MY1    <= 1'b1 ;

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

// Switch OFF all other lights not wanted.
MY2    <= 1'b0 ;
```

Once again, the lights are different here.

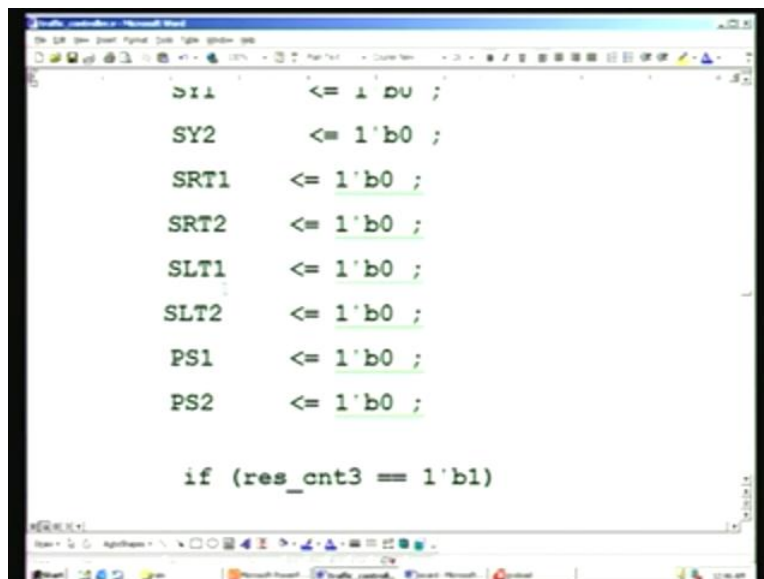
(Refer Slide Time: 42:45)



```
-----
MG2    <= 1'b0 ;
MRT1   <= 1'b0 ;
MRT2   <= 1'b0 ;
MLT1   <= 1'b0 ;
MLT2   <= 1'b0 ;
PM     <= 1'b0 ;
SG1    <= 1'b0 ;
SG2    <= 1'b0 ;
SY1    <= 1'b0 ;
-----
```

Once again, you can see the left flow for the main are all switched off here including the pedestrian crossing.

(Refer Slide Time: 42:55)

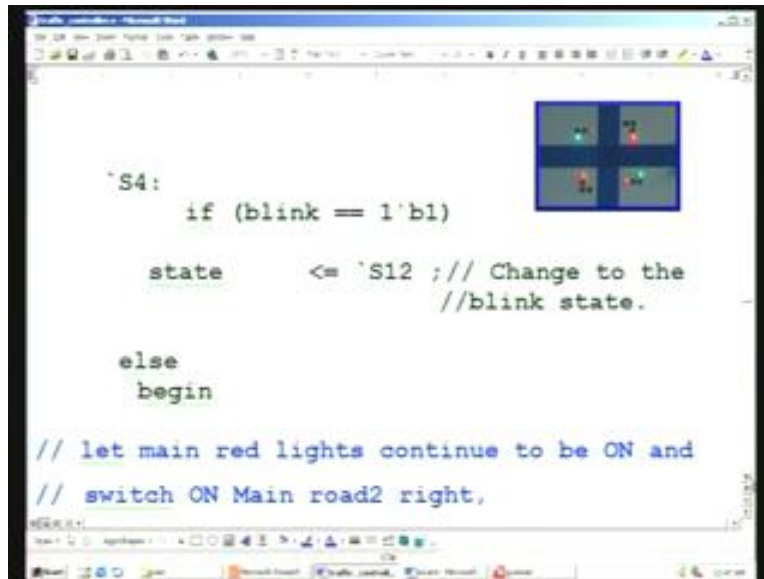


```
SY1    <= 1'b0 ;
SY2    <= 1'b0 ;
SRT1   <= 1'b0 ;
SRT2   <= 1'b0 ;
SLT1   <= 1'b0 ;
SLT2   <= 1'b0 ;
PS1    <= 1'b0 ;
PS2    <= 1'b0 ;

if (res_cnt3 == 1'b1)
```

The side road left are also all off here and so is the case with the pedestrian crossing.

(Refer Slide Time: 43:12)



```

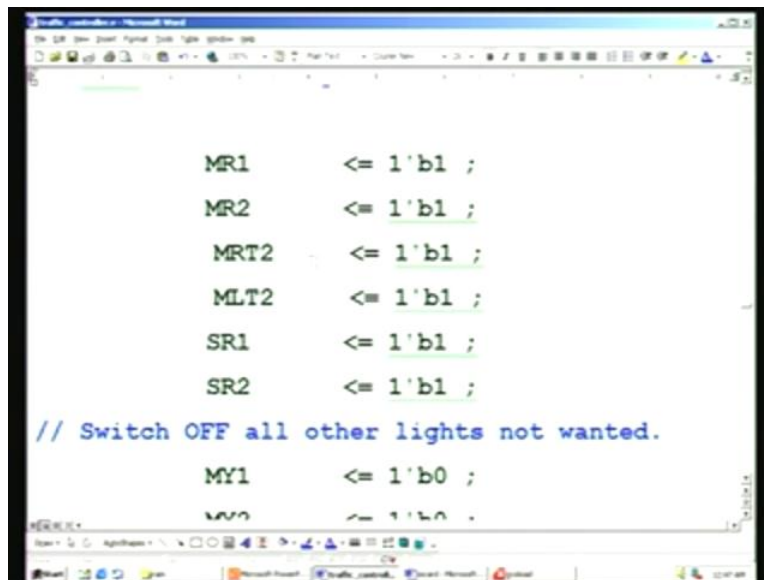
`S4:
    if (blink == 1'b1)
        state    <= `S12 ;// Change to the
                        //blink state.
    else
        begin

// let main red lights continue to be ON and
// switch ON Main road2 right,

```

This is the S4 state. If you want to recollect what that S4 is, you can have a look here (Refer Slide Time: 43:18). This is right flow traffic from the main road itself.

(Refer Slide Time: 43:32)



```

MR1    <= 1'b1 ;
MR2    <= 1'b1 ;
MRT2   <= 1'b1 ;
MLT2   <= 1'b1 ;
SR1    <= 1'b1 ;
SR2    <= 1'b1 ;

// Switch OFF all other lights not wanted.
MY1    <= 1'b0 ;
MY2    <= 1'b0 ;

```

You can see that right is allowed here; right as well as left flow from the second of the main road. All other lights are off.

(Refer Slide Time: 43:51)


```

SRT1    <= 1'b0 ;
SRT2    <= 1'b0 ;
SLT1    <= 1'b0 ;
SLT2    <= 1'b0 ;
PS1     <= 1'b0 ;
PS2     <= 1'b0 ;

if (res_cnt4 == 1'b1)

// This corresponds to 10 Secs. timing of
// timer 3.

```



You can see that this left side as well as pedestrian signals are all off.

(Refer Slide Time: 44:03)

```

`S5:
  if (blink == 1'b1)

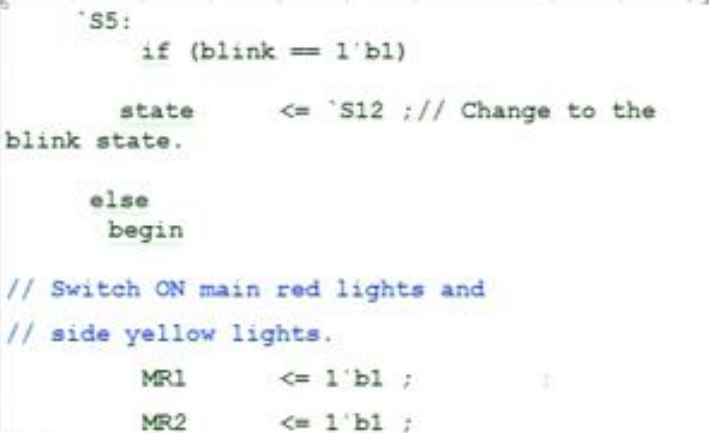
    state    <= `S12 ;// Change to the
    blink state.

  else
    begin

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

      MR1    <= 1'b1 ;
      MR2    <= 1'b1 ;
    end

```



The S5 state is just the yellow transition in the same manner.

(Refer Slide Time: 44:17)

```
MR1      <= 1'b1 ;  
MR2      <= 1'b1 ;  
  
SY1      <= 1'b1 ;  
SY2      <= 1'b1 ;  
  
// Switch OFF all other lights not wanted.  
  
MY1      <= 1'b0 ;  
MY2      <= 1'b0 ;  
MG1      <= 1'b0 ;
```

```
MG2      <= 1'b0 ;  
  
MRT1     <= 1'b0 ;  
MRT2     <= 1'b0 ;  
MLT1     <= 1'b0 ;  
MLT2     <= 1'b0 ;  
PM       <= 1'b0 ;  
SR1      <= 1'b0 ;  
SR2      <= 1'b0 ;  
SG1      <= 1'b0 ;
```



```
SG2 <= 1'b0 ;
SRT1 <= 1'b0 ;
SRT2 <= 1'b0 ;
SLT1 <= 1'b0 ;
SLT2 <= 1'b0 ;
PS1  <= 1'b0 ;
PS2  <= 1'b0 ;

if (res_cnt3 == 1'b1)
```

You can see the yellow is on here and all others are off including MLT, and SR, PM, then SLT as well as pedestrian crossing.

(Refer Slide Time: 44:39)

```
`S6:
    if (blink == 1'b1)
        state <= `S12 ;
    // Change to the // blink state.
    else
        begin
            // Switch ON main red lights, main Pedestrian
            // lights and side green lights.
```

The next state is S6 here. Once again, the blink is taken into account. This is the S6 state.

(Refer Slide Time: 44:46)

```
MR1    <= 1'b1 ;
MR2    <= 1'b1 ;
PM     <= 1'b1 ;
SG1    <= 1'b1 ;
SG2    <= 1'b1 ;

// Switch OFF all other lights not wanted.

MY1    <= 1'b0 ;
MY2    <= 1'b0 ;
```

Here, S6 is the side main traffic flow. There is a PM here (Refer Slide Time: 44:56). Only SG1 and SG2 are alive there.

(Refer Slide Time: 45:05)

```
MR1    <= 1'b1 ;
MR2    <= 1'b1 ;
PM     <= 1'b1 ;
SG1    <= 1'b1 ;
SG2    <= 1'b1 ;

// Switch OFF all other lights not wanted.

MY1    <= 1'b0 ;
MY2    <= 1'b0 ;
MY3    <= 1'b0 ;
```

You can see SG1, SG2 and PM also there. All others are off. The side main traffic yellow lights are shown here as the S7 sequence (Refer Slide Time: 45:26).

(Refer Slide Time: 45:29)

```


//Let Main road red be ON,
MR1    <= 1'b1 ;

MR2    <= 1'b1 ;

//Side road yellow ON and
SY1    <= 1'b1 ;
SY2    <= 1'b1 ;

MY1    <= 1'b0 ;
//Switch OFF all unwanted lights.
MY2    <= 1'b0 ;


```



```

MY1    <= 1'b0 ;
//Switch OFF all unwanted lights.
MY2    <= 1'b0 ;
MG1    <= 1'b0 ;
MG2    <= 1'b0 ;
MRT1   <= 1'b0 ;
MRT2   <= 1'b0 ;
MLT1   <= 1'b0 ;
MLT2   <= 1'b0 ;

```



The corresponding code here is MR1, MR2 are 1, then side is 1 each. All other lights are off.

(Refer Slide Time: 45:50)

```
end

`S8:
  if (blink == 1'b1)
    state <= `S12 ;// Change
              // blink state.

  else
    begin
      MR1 <= 1'b1 ;//Let Main road red
              //be ON itself and
```



```
MR1 <= 1'b1 ;//Let Main road red
              //be ON itself and
MR2 <= 1'b1 ;
SR1 <= 1'b1 ;//Side road right ON
SR2 <= 1'b1 ;
SRT1 <= 1'b1 ;
              //Side road1 right ON,
SLT1 <= 1'b1 ;
              //Side road1 left ON,
MY1 <= 1'b0 ; //Switch OFF all
```



The next sequence is S8. Once again, blink is taken into account. In this case, you can see all reds and side right as well as left traffic is ON here (Refer Slide Time: 46:12). Next is yellow followed by this side. S10 is from the top.

(Refer Slide Time: 46:40)


```

`S9:
  if (blink == 1'b1)
    state    <= `S12 ;
  // Change to the blink state.

  else
    begin

      MR1     <= 1'b1 ;
  //Let Main road red be ON itself and
      MR2     <= 1'b1 ;

```




```

    begin

      MR1     <= 1'b1 ;
  //Let Main road red be ON itself and
      MR2     <= 1'b1 ;
  //Side roads red be ON
      SR1     <= 1'b1 ;
      SR2     <= 1'b1 ;

  //Side road2 yellow be ON

```




```

//Side road2 yellow be ON
    SY2      <= 1'b1 ;

    MY1      <= 1'b0 ;
    MY2      <= 1'b0 ;
//Switch OFF all unwanted lights
    MG1      <= 1'b0 ;
    MG2      <= 1'b0 ;

```



In S9 once again, blink is there. We have seen this. Both the main red as well as side red are ON. So also the side yellow. All other lights are off.

(Refer Slide Time: 47:00)


```

`S10:
    if (blink == 1'b1)

        state    <= `S12 ;
// Change to the blink state.

    else
        begin


```



```

MR1      <= 1'b1 ;
//Let Main road red be ON itself and
MR2      <= 1'b1 ;
SR1      <= 1'b1 ;
//Side roads red be ON
SR2      <= 1'b1 ;
SRT2     <= 1'b1 ;
//Side road2 right ON
SLT2     <= 1'b1 ;


```



```

MR2      <= 1'b1 ;
SR1      <= 1'b1 ;
//Side roads red be ON
SR2      <= 1'b1 ;
SRT2     <= 1'b1 ;
//Side road2 right ON
SLT2     <= 1'b1 ;
//Side road2 left ON
MY1      <= 1'b0 ;

```



This is S10 sequence. Once again, blink is taken into account. In this case, main road red as well as side road are all 1, right and left are ON here – you can just have a look at this (Refer Slide Time: 47:22). You can see this 10. The right as well as left are ON here and all others are red. This turns into yellow here – MY1, MY2 and then red for S11 sequence.


(Refer Slide Time: 47:50)

```
end


`S11:
  if (blink == 1'b1)

      state    <= `S12 ;
  // Change to the blink state.

  else
```



```
    MY1    <= 1'b1 ;
  //Switch on Main roads yellow and
    MY2    <= 1'b1 ;
    SR1    <= 1'b1 ;
  //Side roads red be ON itself
    SR2    <= 1'b1 ;
    SY1    <= 1'b1 ;
  //Side yellow ON
    SY2    <= 1'b1 ;
```

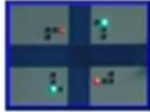




```

SR1      <= 1'b1 ;
//Side roads red be ON itself
SR2      <= 1'b1 ;
SY1      <= 1'b1 ;
//Side yellow ON
SY2      <= 1'b1 ;
MR1      <= 1'b0 ;
MR2      <= 1'b0 ;
MG1      <= 1'b0 ;
MG2      <= 1'b0 ;

```




S11 is here. Once again, blink is taken into account. You have MY1 and MY2 as 1, then side road red here and side yellow on. Is this correct? This is for S11 sequence. S11 is MY1, MY2 and SR1, SR2.

(Refer Slide Time: 48:36)

```

MY1      <= 1'b1 ;
//Switch on Main roads yellow and
MY2      <= 1'b1 ;
SR1      <= 1'b1 ;
//Side roads red be ON itself
SR2      <= 1'b1 ;
SY1      <= 1'b1 ;
//Side yellow ON
SY2      <= 1'b1 ;

```



MY1, MY2 and SR1, SR2. I think there is some problem in this. SY1, SY2 should actually be 0, is it not? This is for S11 case. I think this should be 0, S11, is it not? We will correct this. This is 0, this is also 0. We will just go through this. All other lights are off.

(Refer Slide Time: 49:31)

```


`S12:

    if (blink == 1'b1)
    begin

        begin

            MR1    <= 1'b0 ;
            MR2    <= 1'b0 ;
            MG1    <= 1'b0 ;

```




```

    begin

        begin

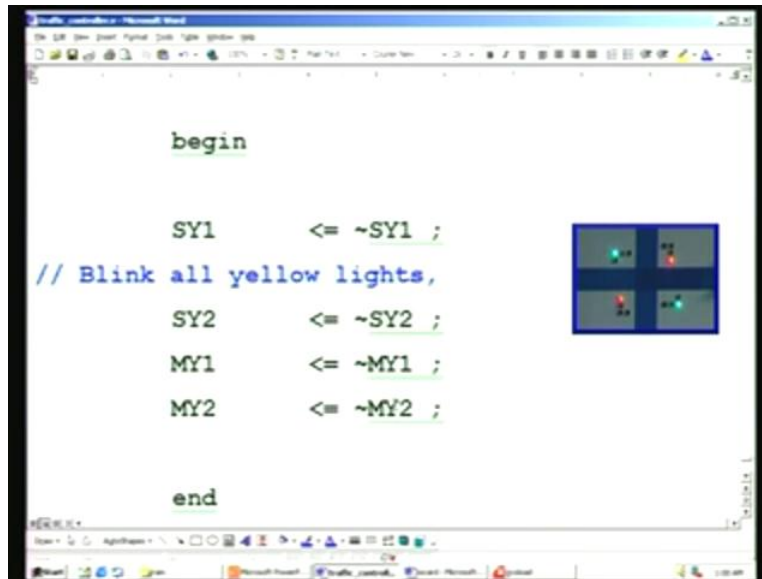
            MR1    <= 1'b0 ;
            MR2    <= 1'b0 ;
            MG1    <= 1'b0 ;
            MG2    <= 1'b0 ;
            MRT1   <= 1'b0 ;
            MRT2   <= 1'b0 ;

```



This S12 is the blink state. We check the blink also here and we switch off all lamps except for the yellow lights, which are going to blink.

(Refer Slide Time: 49:47)



```
begin

    SY1    <= ~SY1 ;
// Blink all yellow lights,
    SY2    <= ~SY2 ;
    MY1    <= ~MY1 ;
    MY2    <= ~MY2 ;

end
```

That is covered only here. This is precisely the same as we had in the S8 state in the earlier design – this is exactly the same thing. We are inverting the SY, all yellow lights condition so that it keeps toggling. This is happening every 0.5 seconds, so you have a 1 second time period or 1 Hertz. That is for all yellow, which you can see from the PowerPoint here (Refer Slide Time: 50:16). In addition to this, what we have is an assignment for you. The timings are all programmed for the main side road traffic as well as yellow lights. What you do is you take these switches SW7 and SW8, which are all BCD switches and you can program right up to 99. You use this as input for the timing. You have only one two-digit setting. Each time you want to set for main traffic delay, you can use two more jumpers. In the SW1 switch, the last two bits you allocate for selecting which of the three timers you are going to select. Let us say you select 00 for the first main timing, 01 for the side timing and 10 for the yellow timing. You also have to set the corresponding timing in the BCD switch. Whenever you want to enter this, you can just press push button 2. This is the assignment for you, so that a cop can change it right on the field. Thank you.

(Refer Slide Time: 51:55)

