

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

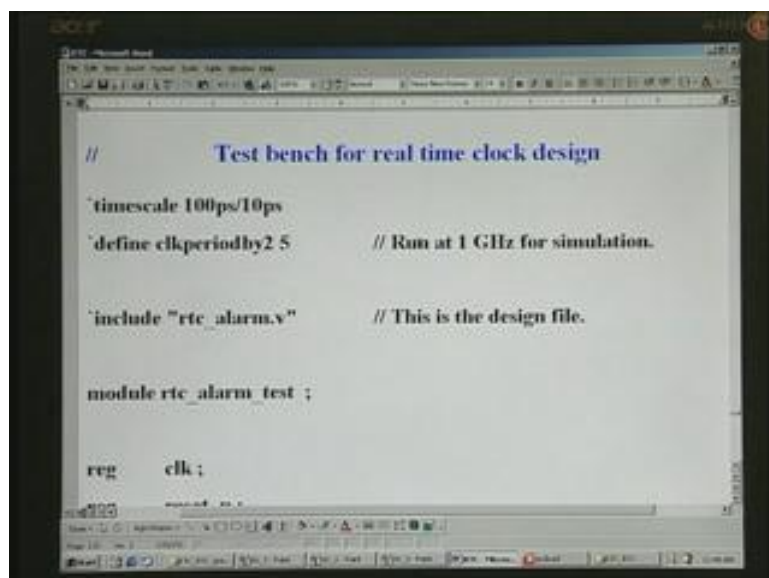
Lecture 54
System Design Examples Using FPGA Board (Continued)

(Refer Slide Time: 01:32)



Last time, we completed the design of a real-time clock and this time, we are going to touch upon the test bench.

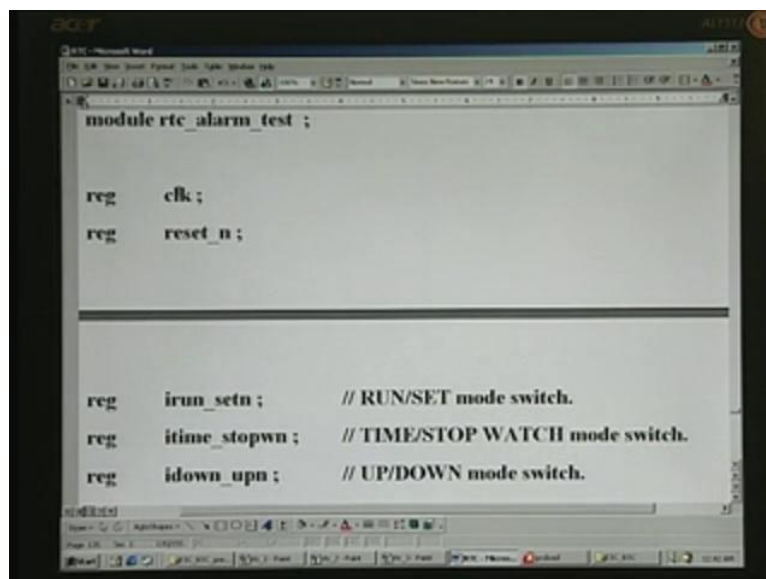
(Refer Slide Time: 02:11)



You are not going to write an elaborate test bench but only to test the basic running of the time. You can try other modes as well on your own. To start with, we need to declare the time scale. Let us say we wish to go for 1 Gigahertz simulation, for which we need to go for higher precision here. We will make the time base as 100 picoseconds and the accuracy is 10 picoseconds – we have already discussed this a number of times earlier. You also have a **clock period by 2** declared as 5. The basic unit is 100 picoseconds.

If you put 5 here, it means 500 picoseconds and this is on time. Likewise, equal number will be for the off time of the clock. The total that you get is 2 into 5 and that is 10 into 100, which is 1000 picosecond or 1 nanosecond. If you take 1 nanosecond, it is nothing other than 1 Gigahertz and that is how you fix the frequency in this fashion. The next thing we have to do is include the actual design, which is **rtc_alarm.v**. This is the design file and we need to declare the present module, which is the test bench – **rtc_alarm_test**. This is the nomenclature we have been following all through. We need to declare the inputs and outputs even on the test bench.

(Refer Slide Time: 03:45)



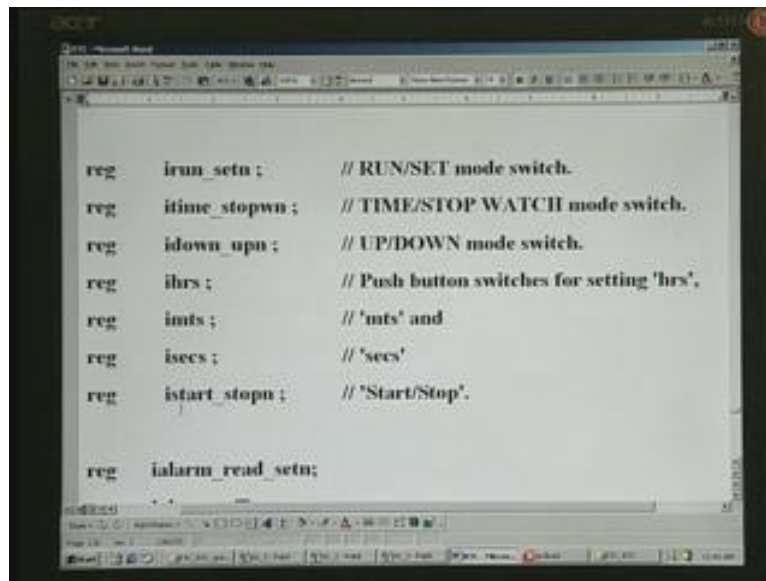
```
module rtc_alarm_test ;

reg    clk ;
reg    reset_n ;

reg    irun_setn ;    // RUN/SET mode switch.
reg    itime_stopwn ; // TIME/STOP WATCH mode switch.
reg    idown_upn ;   // UP/DOWN mode switch.
```

As you recollect, we have used reg for the inputs in the test bench and wire for the outputs once again in the test bench. This is different from what we have been using in the design. We need different inputs such as **clock, reset and run set switch**. This is **TIME/STOP WATCH**, then **UP/DOWN count mode switch**.

(Refer Slide Time: 04:18)

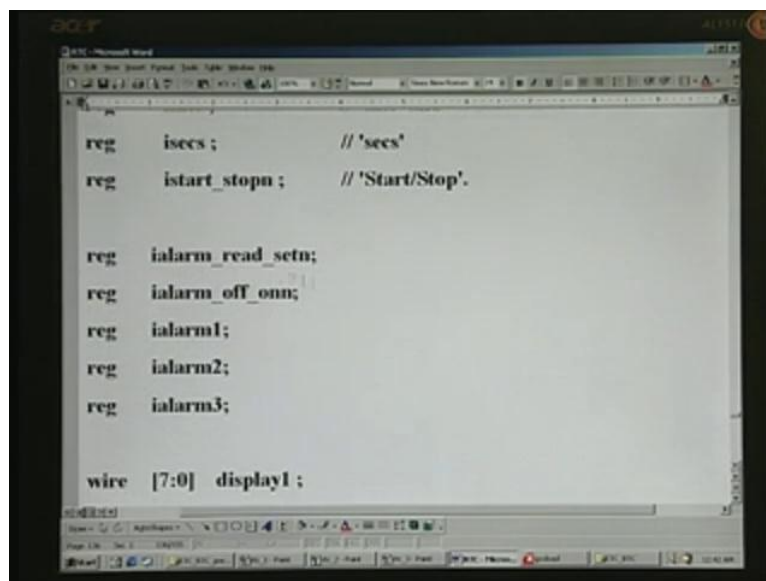


```
reg    irun_setn ;      // RUN/SET mode switch.
reg    itime_stopwn ; // TIME/STOP WATCH mode switch.
reg    idown_upn ;     // UP/DOWN mode switch.
reg    ihrs ;          // Push button switches for setting 'hrs'.
reg    imts ;          // 'mts' and
reg    isecs ;         // 'secs'
reg    istart_stopn ;  // 'Start/Stop'.

reg    ialarm_read_setn;
```

Then, push button switches for hours, minutes, seconds as well as push button switch for **Start and Stop**.

(Refer Slide Time: 04:20)



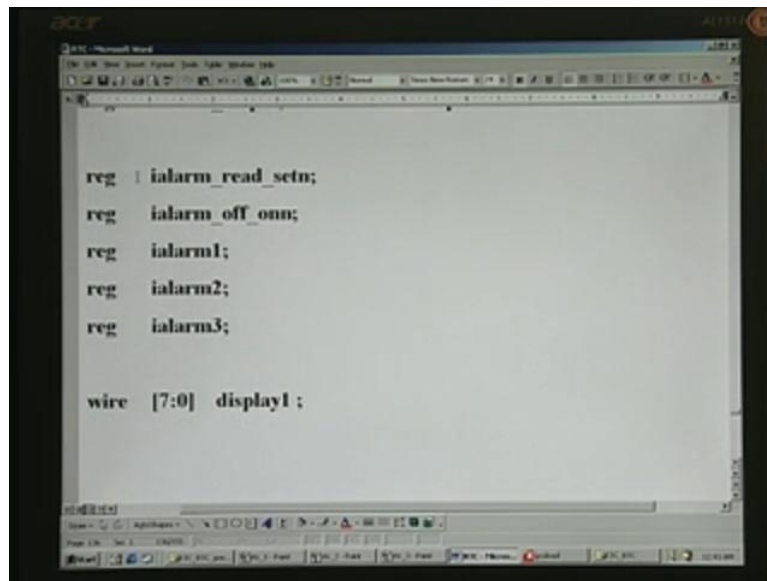
```
reg    isecs ;         // 'secs'
reg    istart_stopn ;  // 'Start/Stop'.

reg    ialarm_read_setn;
reg    ialarm_off_onn;
reg    ialarm1;
reg    ialarm2;
reg    ialarm3;

wire [7:0] display1 ;
```

We also need different switches such as **alarm read or set** or **alarm off or on**, then three different switches for the three alarms that we have and as we have seen before, i stands for the input.

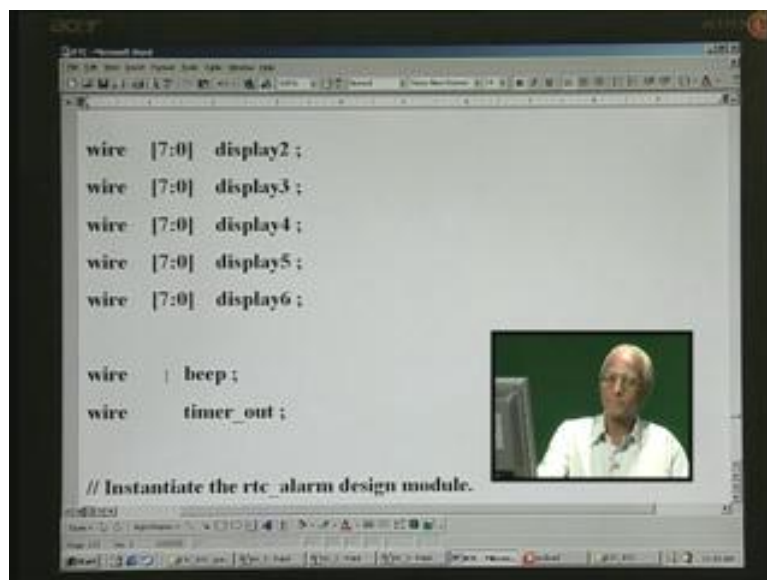
(Refer Slide Time: 04:37)



```
reg | ialarm_read_setn;  
reg ialarm_off_onn;  
reg ialarm1;  
reg ialarm2;  
reg ialarm3;  
  
wire [7:0] display1 ;
```

As far as the test bench is concerned, we are not concerned about **r alarm or alarm**, etc., which we have used in the design because they are all internal signals. As far as the outside world is concerned, it is **i that is the actual input, the display will be the output anyway** – display1 through display6, and each of which is 8 bits.

(Refer Slide Time: 04:58)

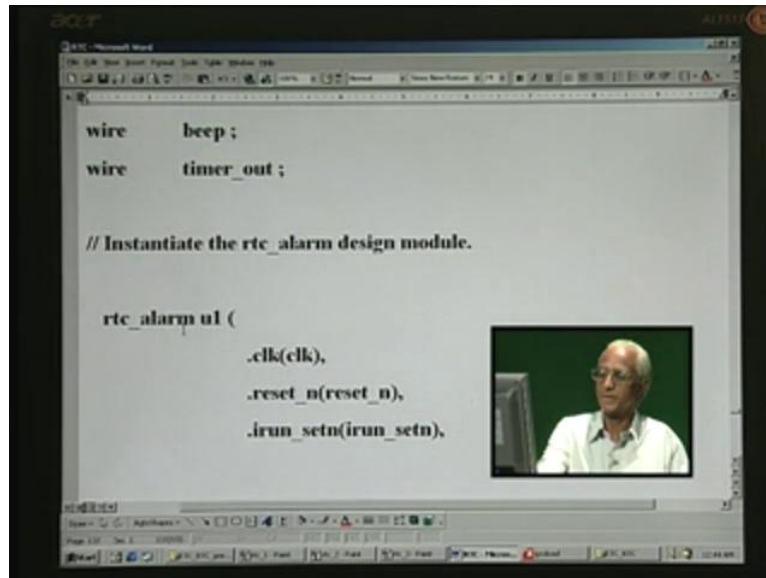


```
wire [7:0] display2 ;  
wire [7:0] display3 ;  
wire [7:0] display4 ;  
wire [7:0] display5 ;  
wire [7:0] display6 ;  
  
wire | beep ;  
wire timer_out ;  
  
// Instantiate the rtc_alarm design module.
```

We have seen that **a, b, c, d** is the order, **a is the bit 7** and decimal point is the very last 0 bit. We also have two more outputs: beep as well as timer_out. This is what we have used for setting a particular output when the count matches either in the up count mode or down count mode. Beep is the square pulse that we create in order to switch on a piezoelectric buzzer.

The buzzer will have a continuous sound and when you **output as a** square pulse switching on and switching off, you get a beeping sound and that is why we nomenclature it as beep. Note that in the test bench, we need to declare all the outputs as wire.

(Refer Slide Time: 05:53)



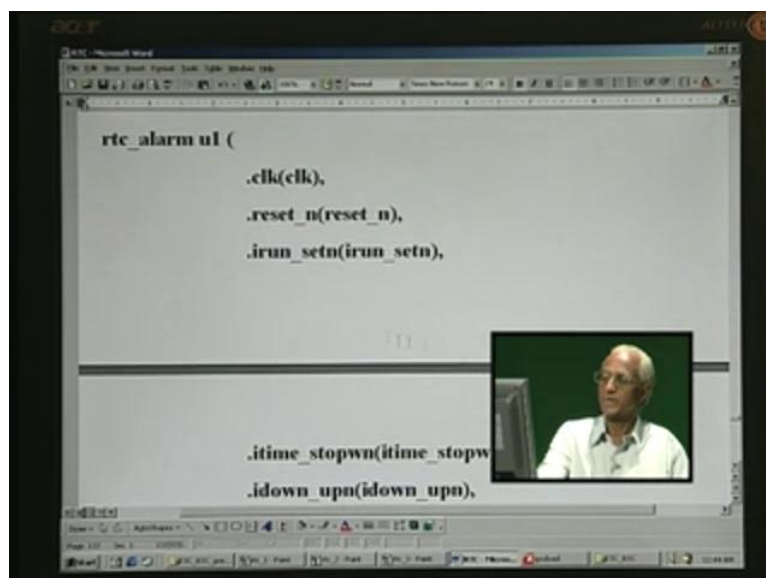
```
wire    beep ;
wire    timer_out ;

// Instantiate the rtc_alarm design module.

rtc_alarm u1 (
    .clk(clk),
    .reset_n(reset_n),
    .irun_setn(irun_setn),
```

Next, we will instantiate the rtc_alarm design module and that is this here and instantiate just once. This is like any other IC that you give instantiation for.

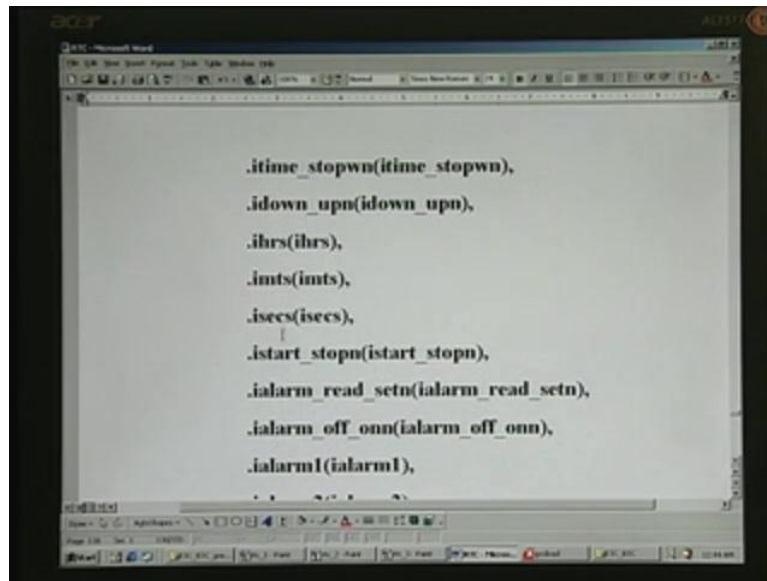
(Refer Slide Time: 06:01)



```
    .itime_stopwn(itime_stopwn),
    .idown_upn(idown_upn),
```

We declare ports by name as clock, reset, irun, **set time**, **stopwatch** – all these are the various inputs.

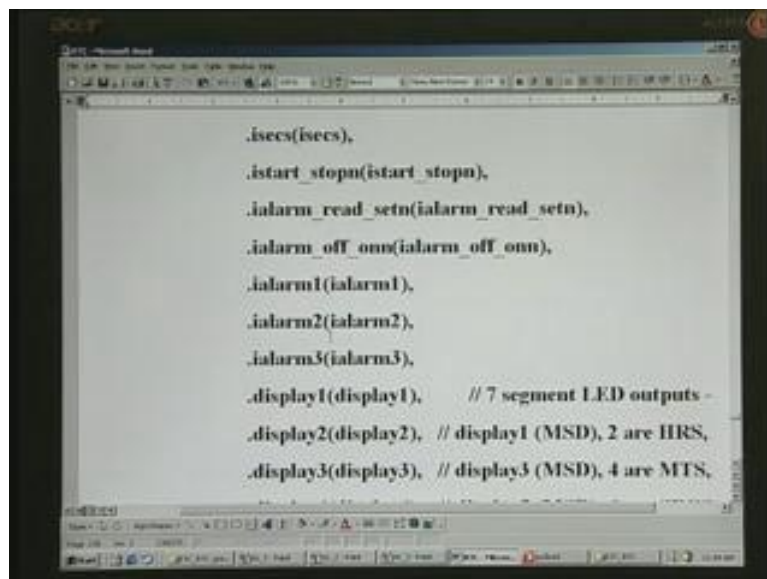
(Refer Slide Time: 06:12)



```
.itime_stopwn(itime_stopwn),
.idown_upn(idown_upn),
.ihrs(ihrs),
.imts(imts),
.isecs(isecs),
.istart_stopn(istart_stopn),
.ialarm_read_setn(ialarm_read_setn),
.ialarm_off_onn(ialarm_off_onn),
.ialarm1(ialarm1),
```

Down, up, then hours, minutes, seconds, there are the **start stop** push button switches and then **alarm read set**, **alarm off on** – the three alarms that you have.

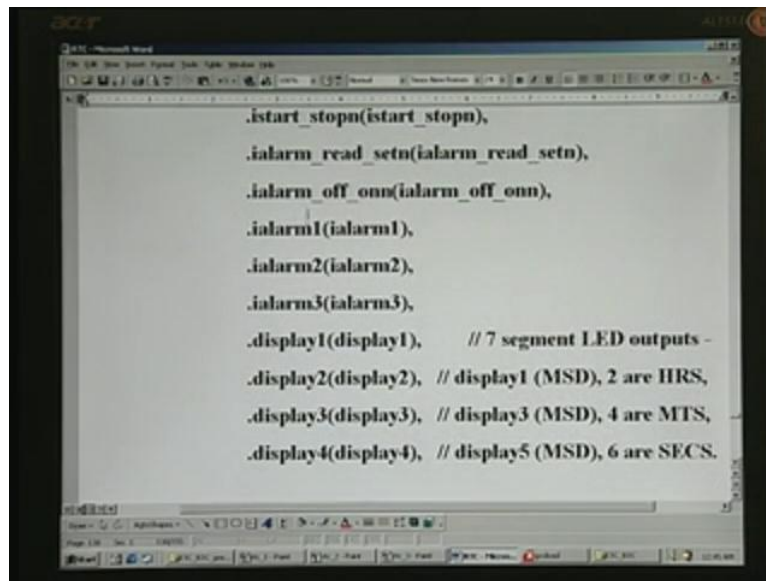
(Refer Slide Time: 06:22)



```
.isecs(isecs),
.istart_stopn(istart_stopn),
.ialarm_read_setn(ialarm_read_setn),
.ialarm_off_onn(ialarm_off_onn),
.ialarm1(ialarm1),
.ialarm2(ialarm2),
.ialarm3(ialarm3),
.display1(display1), // 7 segment LED outputs
.display2(display2), // display1 (MSD), 2 are HRS,
.display3(display3), // display3 (MSD), 4 are MTS,
```

Note that we have used very same name even inside this.

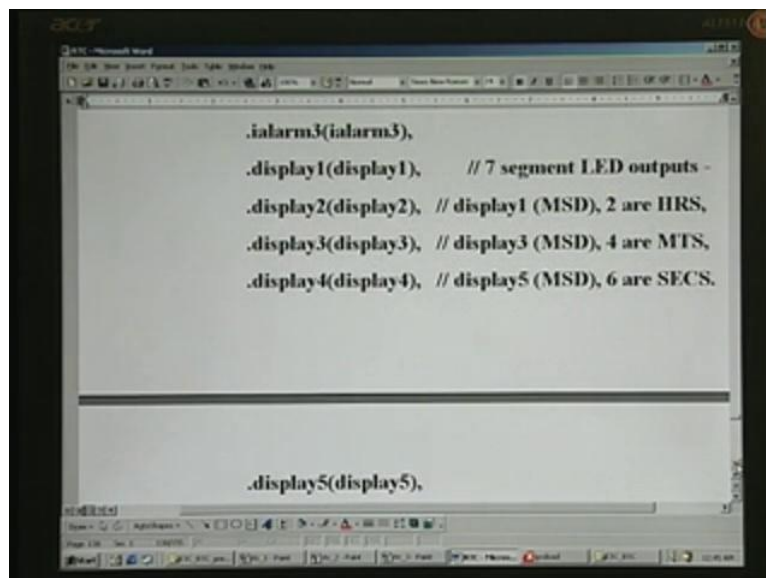
(Refer Slide Time: 06:30)



```
.istart_stopn(istart_stopn),  
.ialarm_read_setn(ialarm_read_setn),  
.ialarm_off_onn(ialarm_off_onn),  
.ialarm1(ialarm1),  
.ialarm2(ialarm2),  
.ialarm3(ialarm3),  
.display1(display1), // 7 segment LED outputs -  
.display2(display2), // display1 (MSD), 2 are HRS,  
.display3(display3), // display3 (MSD), 4 are MTS,  
.display4(display4), // display5 (MSD), 6 are SECS.
```

This implies that within the design module, this is the name and in the present test bench, these are all the names. Since both are same, we have declared it as wire, which actually means these symbols as such.

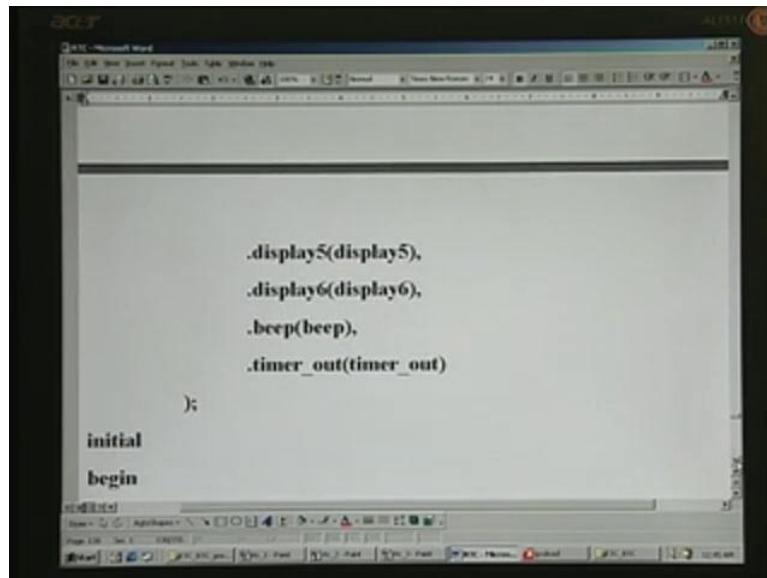
(Refer Slide Time: 06:53)



```
.ialarm3(ialarm3),  
.display1(display1), // 7 segment LED outputs -  
.display2(display2), // display1 (MSD), 2 are HRS,  
.display3(display3), // display3 (MSD), 4 are MTS,  
.display4(display4), // display5 (MSD), 6 are SECS.  
  
.display5(display5),
```

These are all the seven segment LED outputs.

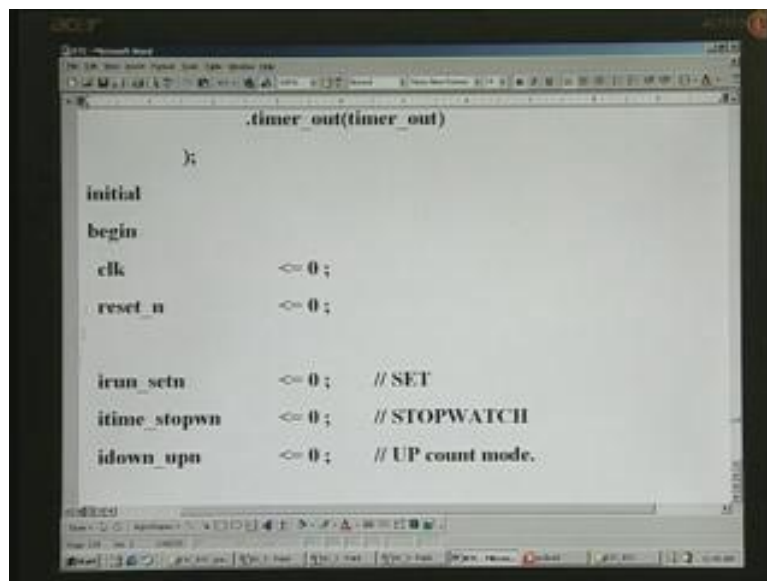
(Refer Slide Time: 06:55)



```
        .display5(display5),
        .display6(display6),
        .beep(beep),
        .timer_out(timer_out)
    );
initial
begin
```

display1 through display6. Beep and timer_out also will have to be declared.

(Refer Slide Time: 07:07)

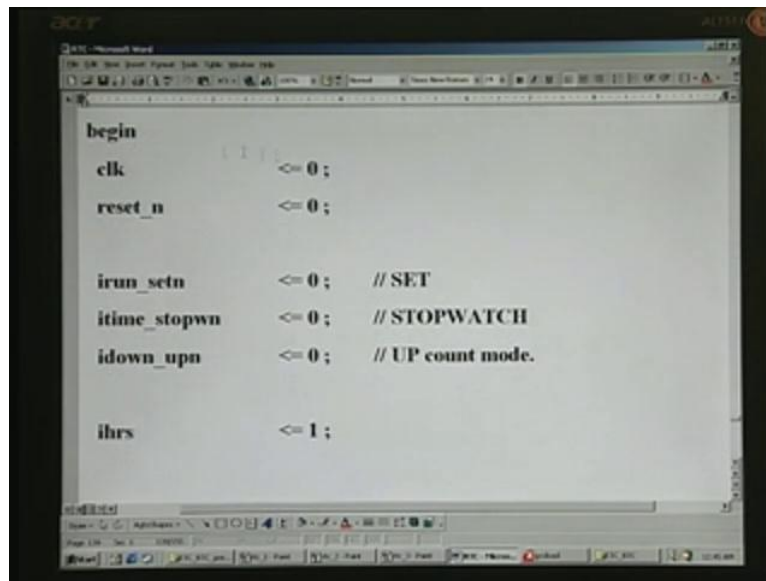


```
        .timer_out(timer_out)
    );
initial
begin
    clk          <= 0;
    reset_n     <= 0;

    irun_setn   <= 0; // SET
    itime_stopwn <= 0; // STOPWATCH
    idown_upn   <= 0; // UP count mode.
```

The next thing that we do is apply stimulants in the form of various inputs. For that, you need an initial block. There is a begin and there will be a corresponding end towards the end of this test bench.

(Refer Slide Time: 07:16)



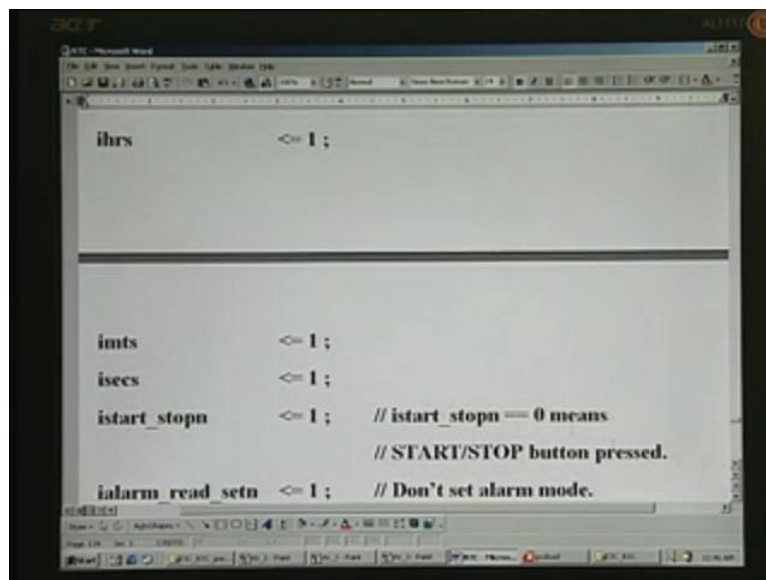
```
begin
  clk          <= 0;
  reset_n     <= 0;

  irun_setn   <= 0; // SET
  itime_stopwn <= 0; // STOPWATCH
  idown_upn   <= 0; // UP count mode.

  ihrs        <= 1;
```

First, what we will do is we will initialize clock to 0, then we will apply the reset pulse immediately and perhaps after 100 units of time, we can start the real work and some more switches are also initialized here, though not necessary at this point of time.

(Refer Slide Time: 07:36)

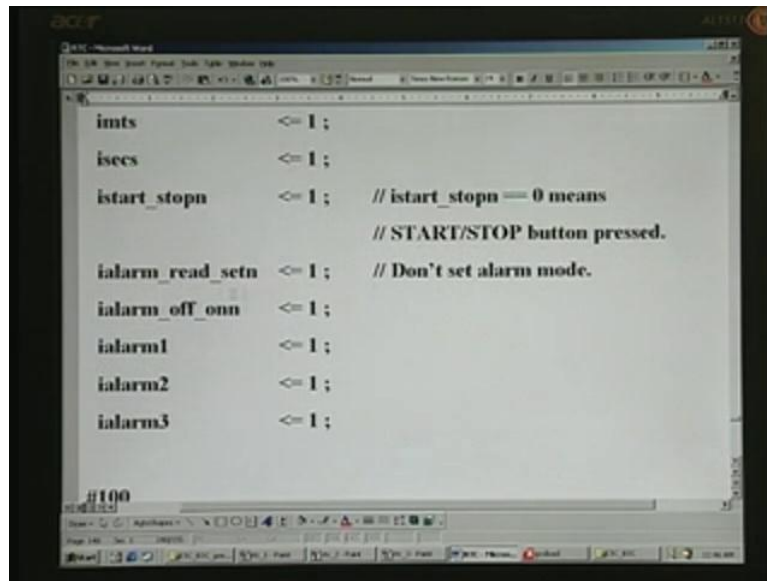


```
ihrs          <= 1;

imts          <= 1;
isecs         <= 1;
istart_stopn  <= 1; // istart_stopn == 0 means
                  // START/STOP button pressed.
ialarm_read_setn <= 1; // Don't set alarm mode.
```

Similarly, the hours, minutes, seconds, **start_stop** all have been initialized to the inactive state. The active state is **start stop** for example must be 0 **if you mean** start has been pressed. That is to say that it is in start mode if it is 0. Right now, it is in stop mode.

(Refer Slide Time: 08:00)

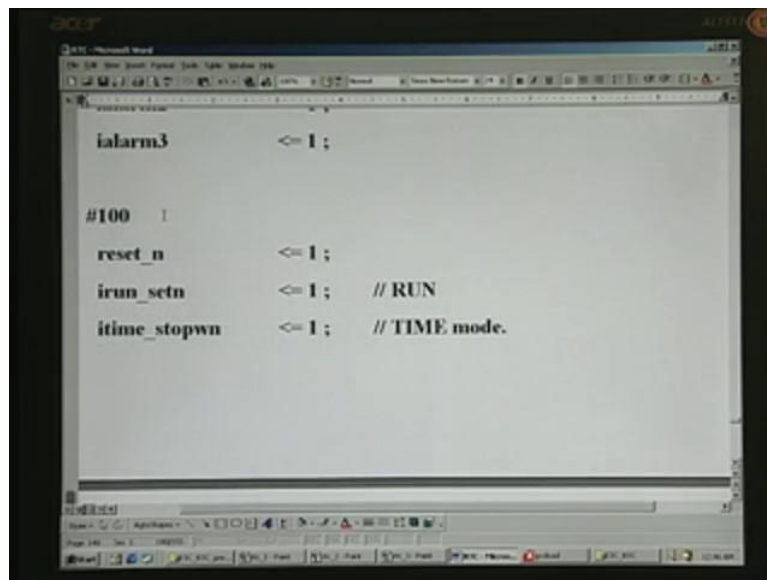


```
imts          <= 1 ;
isecs         <= 1 ;
istart_stpn   <= 1 ; // istart_stpn == 0 means
                  // START/STOP button pressed.
ialarm_read_setn <= 1 ; // Don't set alarm mode.
ialarm_off_onn <= 1 ;
ialarm1       <= 1 ;
ialarm2       <= 1 ;
ialarm3       <= 1 ;

#100
```

All other switches such as **alarm_read_set**, **alarm_off_on** and the individual alarms are all in the inactive state.

(Refer Slide Time: 08:08)

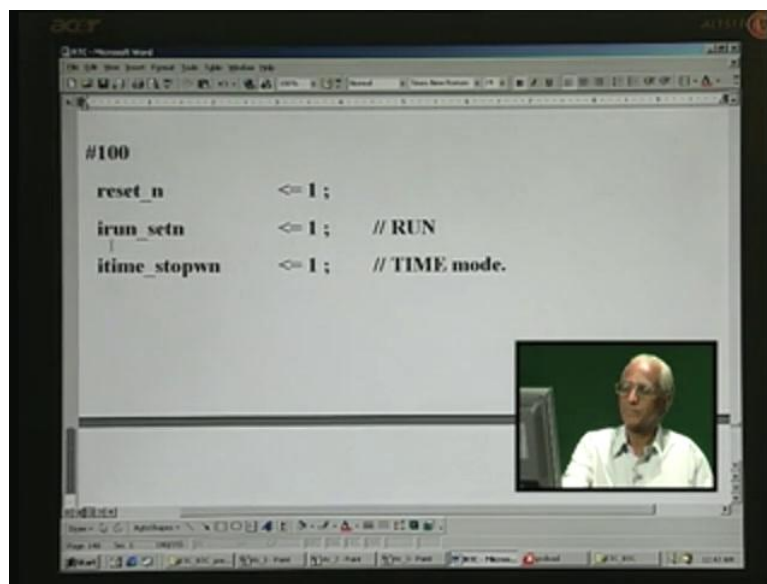


```
ialarm3       <= 1 ;

#100
reset_n       <= 1 ;
irun_setn     <= 1 ; // RUN
itime_stopwn  <= 1 ; // TIME mode.
```

After 100 units of **time...** The unit of time here, as you notice, is 100 picoseconds – we have declared using the time scale earlier. 100 into 100 picoseconds or 10,000 picoseconds or 10 nanoseconds will be the **actual application**, starting of the real-time clock. You notice now that reset has been taken to inactive state; 0 means active low. Now we are making it inactive so that we can start the design working.

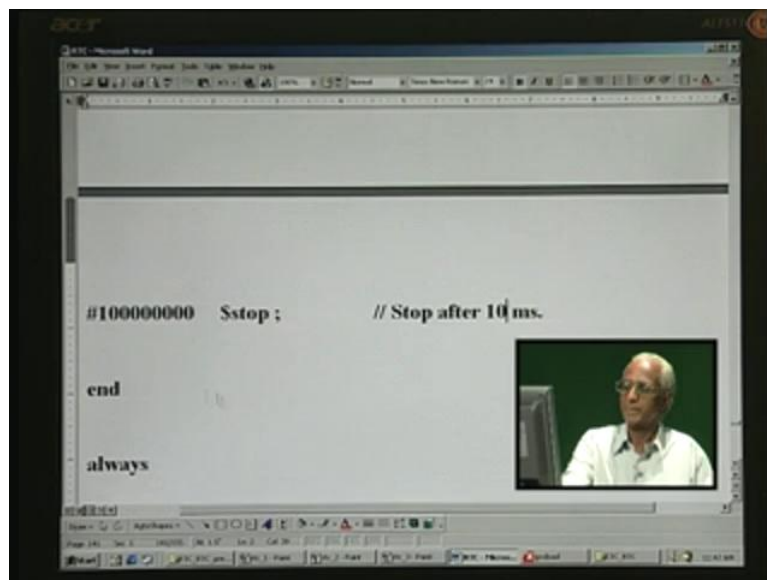
(Refer Slide Time: 08:43)



```
#100
reset_n      <= 1;
irun_setn    <= 1; // RUN
itime_stopwn <= 1; // TIME mode.
```

We also have `run_set` and let us set `that particular to run as well as to time`. That is what we meant by this comment `run, time mode` so that we can check only the running of the normal timing.

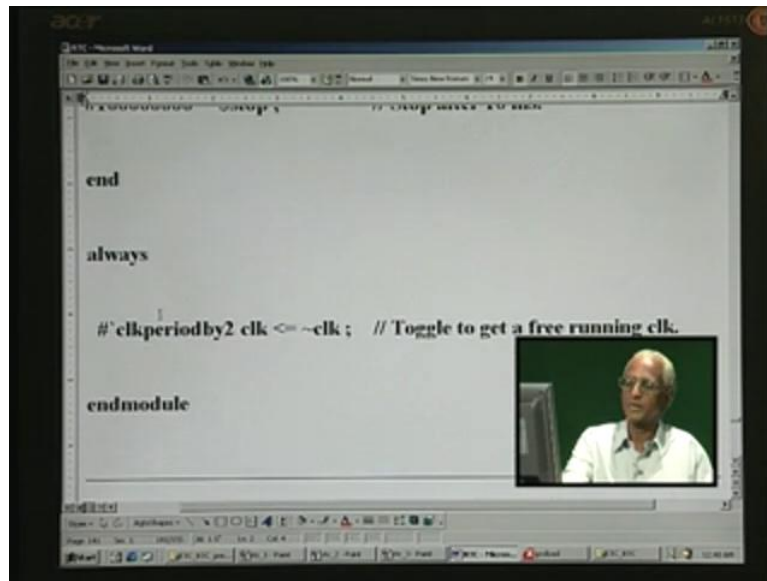
(Refer Slide Time: 08:57)



```
#100000000 Sstop; // Stop after 10 ms.
end
always
```

After quite a long time, we would like to stop. Now, let us analyze why it is 10 millisecond as commented here. We know that it is 100 picoseconds. In 100, two `ciphers` are there, you have one more 0 here. If you knock off this 0, what you will be getting is nanosecond; if you knock off another three, you will be getting microsecond; if you knock off further three, you will get 10 millisecond. That is how you account for 10 milliseconds here.

(Refer Slide Time: 09:31)



```
end

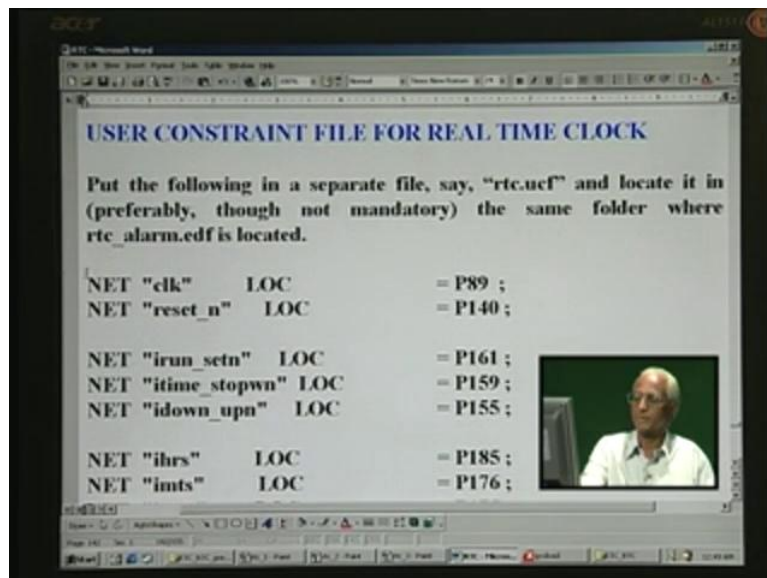
always

#`clkperiodby2 clk <= ~clk; // Toggle to get a free running clk.

endmodule
```

As usual, the clock will have to be toggled, we use an always block and **clock period by 2 after this much time** – that is every 100 picoseconds, what we do is invert the clock and assign it to itself. In other words, we are toggling to get a free-running clock and this ends the test bench.

(Refer Slide Time: 09:50)



```
USER CONSTRAINT FILE FOR REAL TIME CLOCK

Put the following in a separate file, say, "rtc.ucf" and locate it in
(preferably, though not mandatory) the same folder where
rtc_alarm.edi is located.

NET "clk" LOC = P89 ;
NET "reset_n" LOC = P140 ;

NET "irun_setn" LOC = P161 ;
NET "itime_stopwn" LOC = P159 ;
NET "idown_upn" LOC = P155 ;

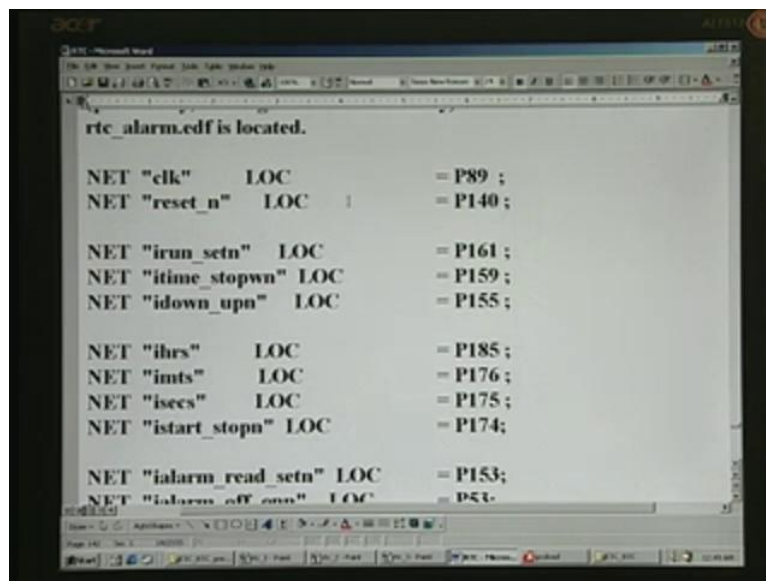
NET "ihrs" LOC = P185 ;
NET "imts" LOC = P176 ;
```

Now, what is to be seen is the actual user constraint file. Put the following in a separate file, say rtc.ucf. Having named it as rtc_alarm, you are free to give the same name or any other name – for brevity's sake, I just put rtc there. **Locate it in... preferably, though not**

mandatory. There is no compulsion for you to locate it in the same folder, but it will be convenient if it is located in the same folder as `rtc_alarm.edf`, which was created by using the Synplify tool and that is the synthesis tool.

This is the ucf file. From here onwards, you will have to put in this `rtc.ucf` file. This ucf file is required in Xilinx place and route when you wish to get the bit stream.

(Refer Slide Time: 10:47)



```
rtc_alarm.edf is located.

NET "clk"      LOC      = P89 ;
NET "reset_n"  LOC      = P140 ;

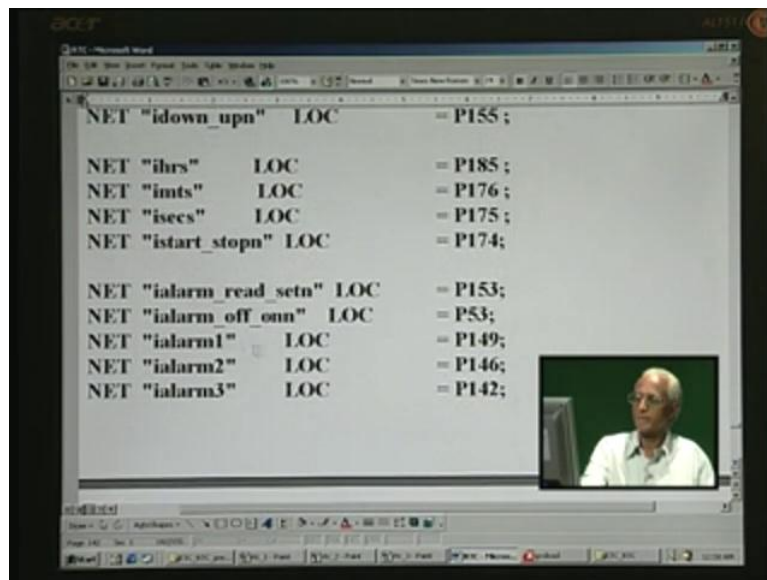
NET "irun_setn" LOC      = P161 ;
NET "itime_stopwn" LOC    = P159 ;
NET "idown_upn" LOC      = P155 ;

NET "ihrs"     LOC      = P185 ;
NET "imts"     LOC      = P176 ;
NET "isecs"    LOC      = P175 ;
NET "istart_stopn" LOC    = P174 ;

NET "ialarm_read_setn" LOC = P153 ;
NET "ialarm_off_on"  LOC  = P51 ;
```

The bit stream will be in a dot bit file by the same name `rtc_alarm`, which is the design. **Net clock location** is what you use for varying signals. For example, for clock, you declare it as NET here and then assign the pin here. For example, clock is assigned 89 pin in the **XCV-800** device, which we have used on the FPGA board and we have used that even for the traffic light controller. We are going to use the very same thing and we are going to see a demo shortly. These are all various pins for all the inputs: **run_set, time, down_up**, then hours, minutes, seconds, start_stop and then alarm_read_set, alarm_off_on.

(Refer Slide Time: 11:34)

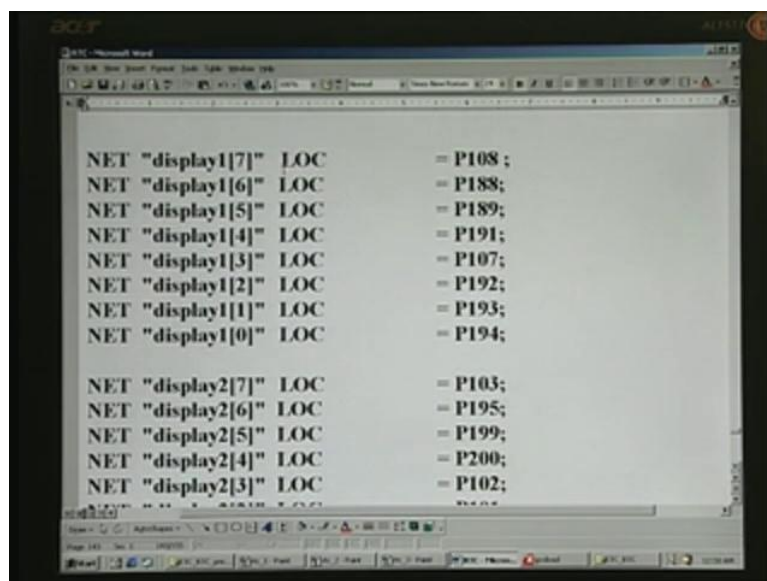


```
NET "idown_upn" LOC = P155 ;
NET "ihrs" LOC = P185 ;
NET "imts" LOC = P176 ;
NET "isecs" LOC = P175 ;
NET "istart_stpn" LOC = P174 ;

NET "ialarm_read_setn" LOC = P153 ;
NET "ialarm_off_omn" LOC = P53 ;
NET "ialarm1" LOC = P149 ;
NET "ialarm2" LOC = P146 ;
NET "ialarm3" LOC = P142 ;
```

Note that the pin numbers are different from each other. You have three alarms – 1 through 3.

(Refer Slide Time: 11:38)

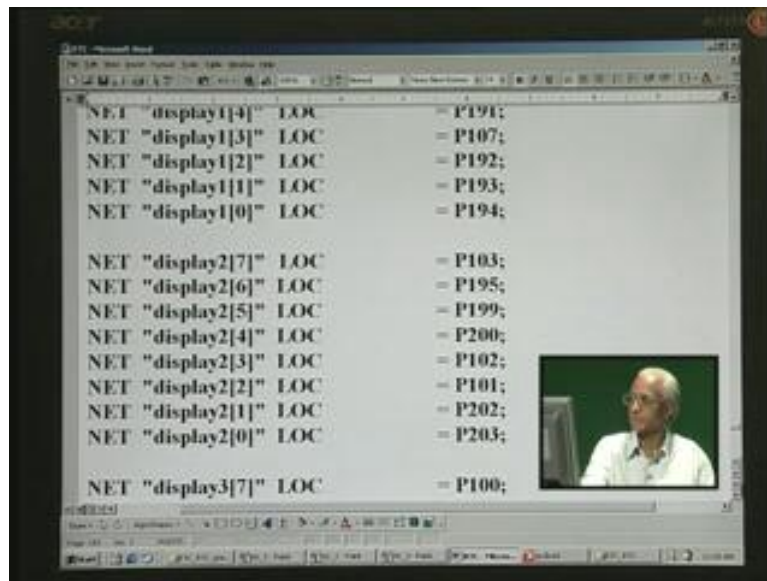


```
NET "display1[7]" LOC = P108 ;
NET "display1[6]" LOC = P188 ;
NET "display1[5]" LOC = P189 ;
NET "display1[4]" LOC = P191 ;
NET "display1[3]" LOC = P107 ;
NET "display1[2]" LOC = P192 ;
NET "display1[1]" LOC = P193 ;
NET "display1[0]" LOC = P194 ;

NET "display2[7]" LOC = P103 ;
NET "display2[6]" LOC = P195 ;
NET "display2[5]" LOC = P199 ;
NET "display2[4]" LOC = P200 ;
NET "display2[3]" LOC = P102 ;
```

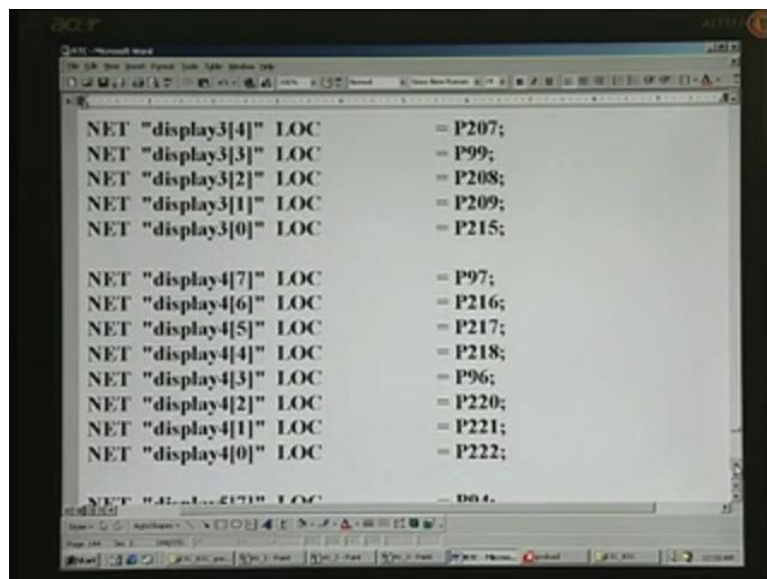
For the output also, you should have and these are all the names of the inputs and outputs. Here you note that you need to do bitwise because there are eight bits – the system cannot do it all by itself. You have to assign a unique number for every bit and note that these numbers are all not in order. It all depends upon how you have connected your extension port from the FPGA board, on the FPGA board.

(Refer Slide Time: 12:15)



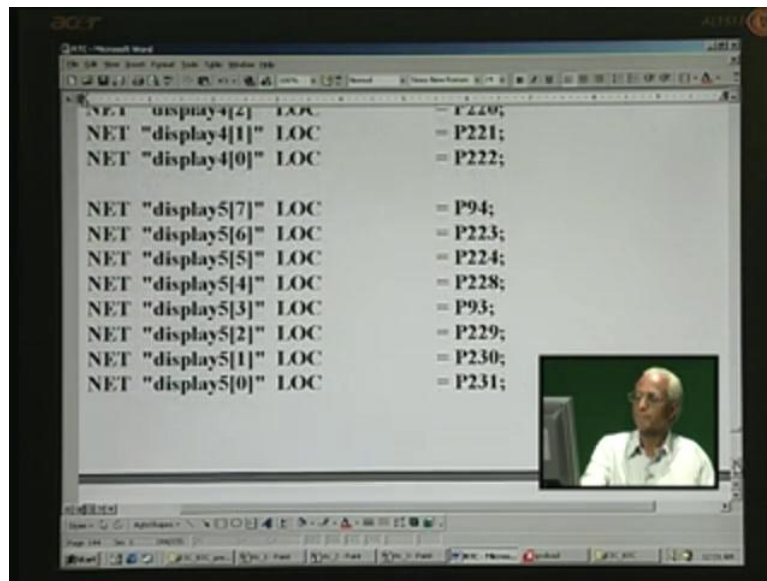
Likewise, we have six displays totally. display2 is here.

(Refer Slide Time: 12:15)



display3 is here, then display4.

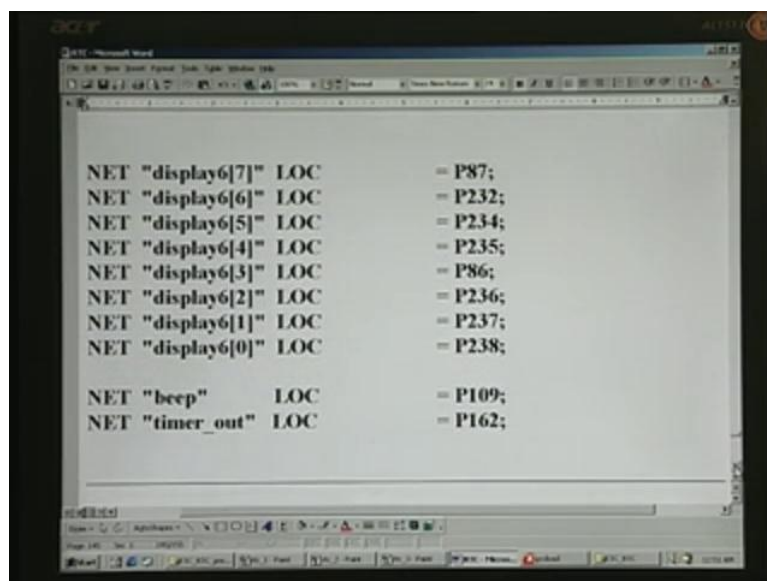
(Refer Slide Time: 12:26)



```
NET "display4[2]" LOC          = P220;  
NET "display4[1]" LOC          = P221;  
NET "display4[0]" LOC          = P222;  
  
NET "display5[7]" LOC          = P94;  
NET "display5[6]" LOC          = P223;  
NET "display5[5]" LOC          = P224;  
NET "display5[4]" LOC          = P228;  
NET "display5[3]" LOC          = P93;  
NET "display5[2]" LOC          = P229;  
NET "display5[1]" LOC          = P230;  
NET "display5[0]" LOC          = P231;
```

Followed by display5.

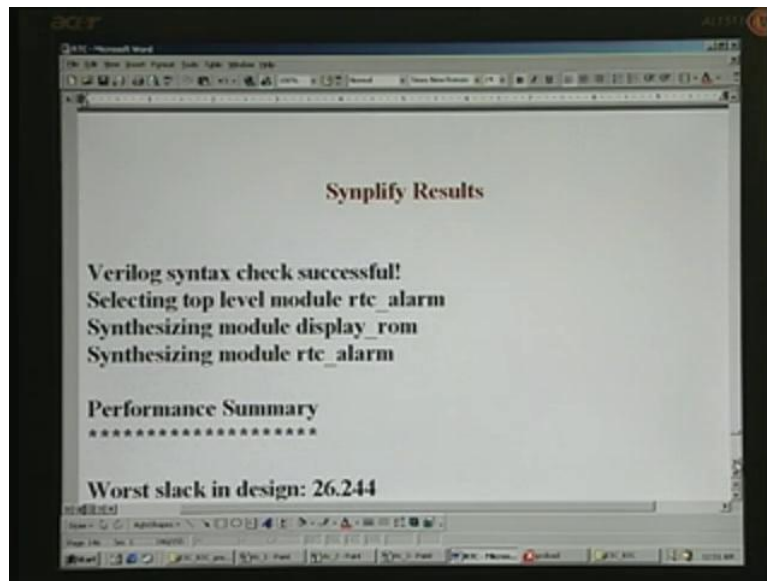
(Refer Slide Time: 12:30)



```
NET "display6[7]" LOC          = P87;  
NET "display6[6]" LOC          = P232;  
NET "display6[5]" LOC          = P234;  
NET "display6[4]" LOC          = P235;  
NET "display6[3]" LOC          = P86;  
NET "display6[2]" LOC          = P236;  
NET "display6[1]" LOC          = P237;  
NET "display6[0]" LOC          = P238;  
  
NET "beep"      LOC           = P109;  
NET "timer_out" LOC           = P162;
```

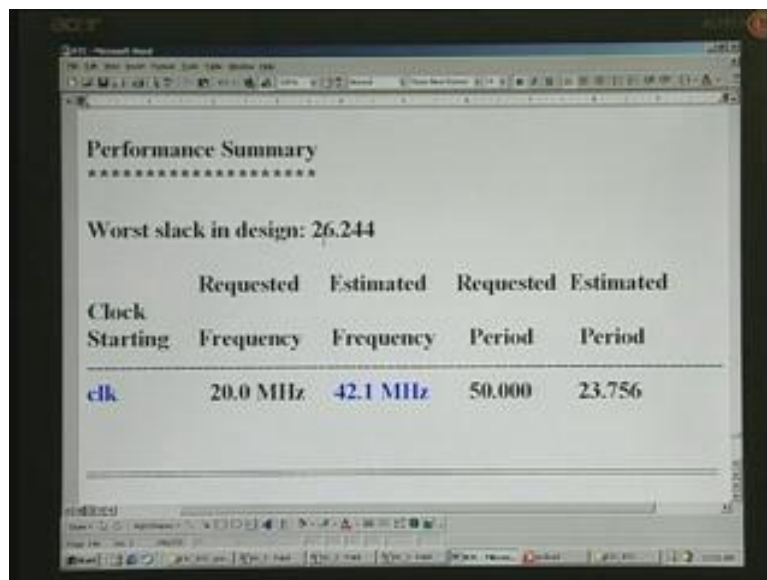
Then, finally display6. In addition to this, we also have two single-bit ports and you need only one pin to be assigned, whereas these are all eight-bit **output port displays** – each of these. Timer_out is just a single pin and that is assigned 162. P is also mandatory – you have to identify it as a pin.

(Refer Slide Time: 12:56)



Next, we will go into the actual Synplify results, so you can see the usual reporting here: Verilog syntax check successful and **top level is** the rtc_alarm, then synthesizing display ROM as well as the actual design.

(Refer Slide Time: 13:12)

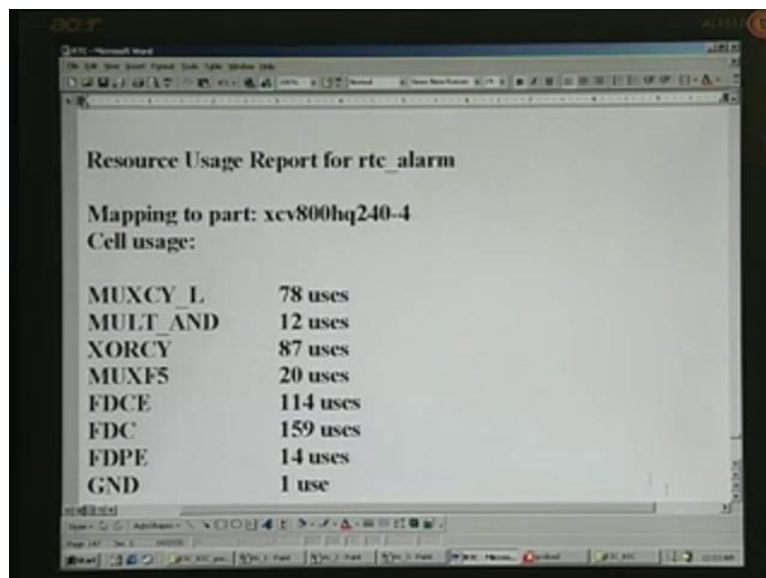


This is the performance summary that you have here. Slack means how much free time is available so that you can jack up your speed if you desire. For example, it would be enough if you run at 20 Megahertz instead of 1 Gigahertz. I have also cautioned you that you can run the simulation at any Gigahertz but after place and route, it will come down drastically. For example, it has come down to 42 Megahertz – this is the highest frequency that you can run.

It will be a misconception and you will be misguided if you just simulate at 1 Gigahertz and jump to the conclusion that you have achieved 1 Gigahertz – that is totally a wrong picture. Only after Xilinx place and route, you will know the true picture.

Even synthesis or Synplify will not reveal the exact frequency of operation – you have to go for the Xilinx place and route, which we are going to see next. Here in Synplify, we had requested only for a 20 Megahertz operation frequency and it has reported 42 Megahertz. This corresponds to 50 nanoseconds – 20 Megahertz would correspond to 50 nanoseconds – 1000 by 20. This is the shortcut for you to compute nanoseconds as we had seen before. If you take 42, corresponding to this, how much is the estimated period, that is here. If you add these two, what you get is this, because 20 Megahertz is the actual thing that we want.

(Refer Slide Time: 14:52)

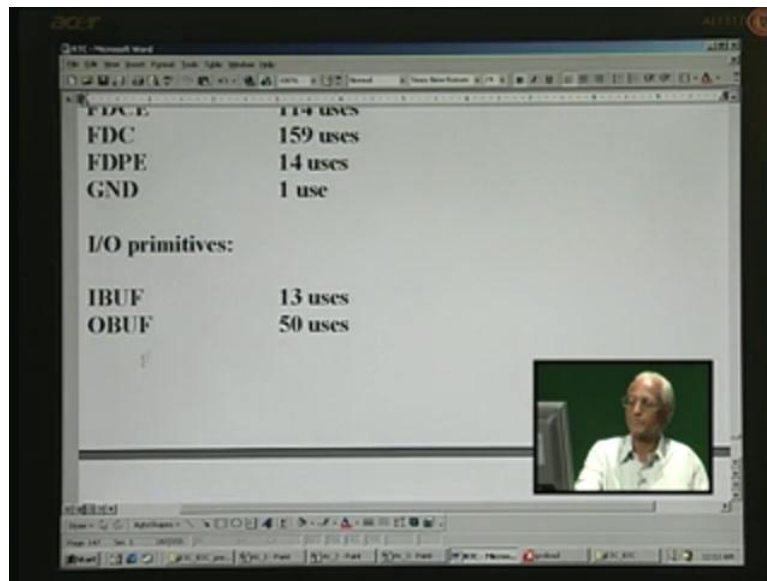


The image shows a screenshot of a software window displaying a "Resource Usage Report for rtc_alarm". The report is for a device mapped to part "xcv800hq240-4". It lists the following cell usage:

Cell Type	Uses
MUXCY_L	78 uses
MULT_AND	12 uses
XORCY	87 uses
MUXF5	20 uses
FDCE	114 uses
FDC	159 uses
FDPE	14 uses
GND	1 use

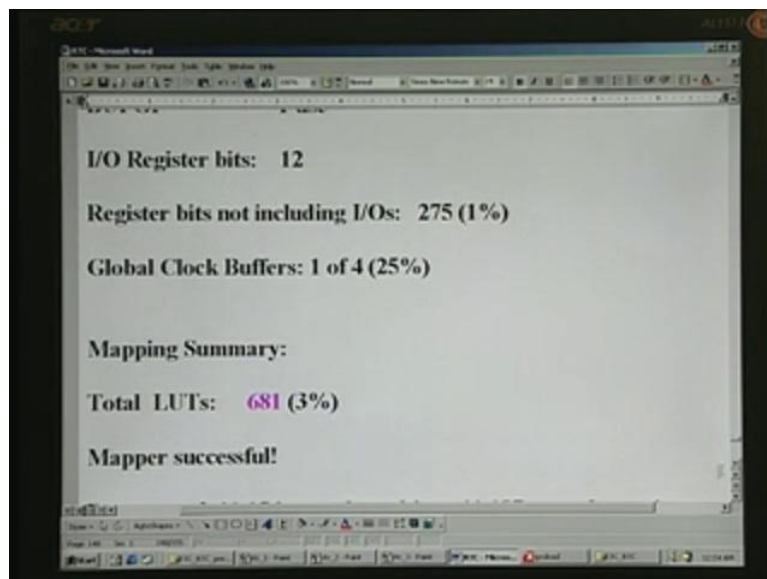
The device that we have used is xcv800HQ240–4. I think this is the highest speed in this particular package. What it reports here is the primitives that have been used in the FPGA. There is a MUX, then AND, exclusive OR, once again MUX, then flip-flops and so on. How many of them have been used or listed here.

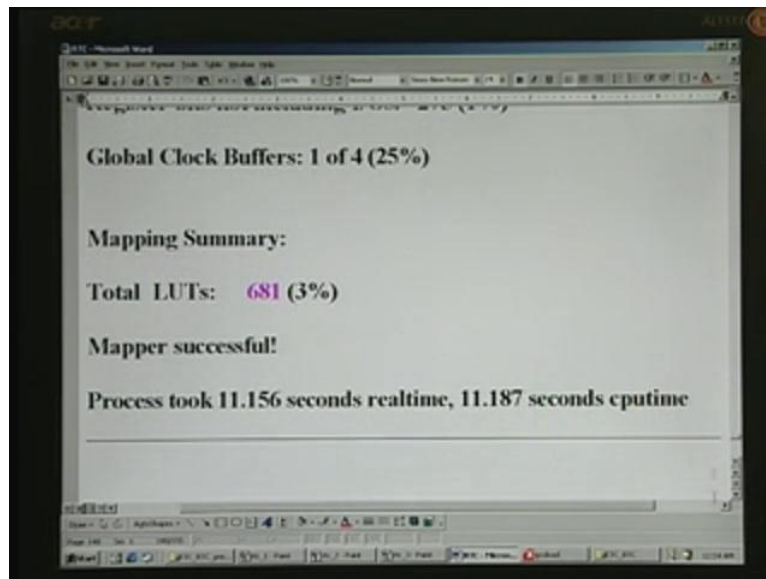
(Refer Slide Time: 15:19)



Followed by input buffers and output buffers that have been used in the design.

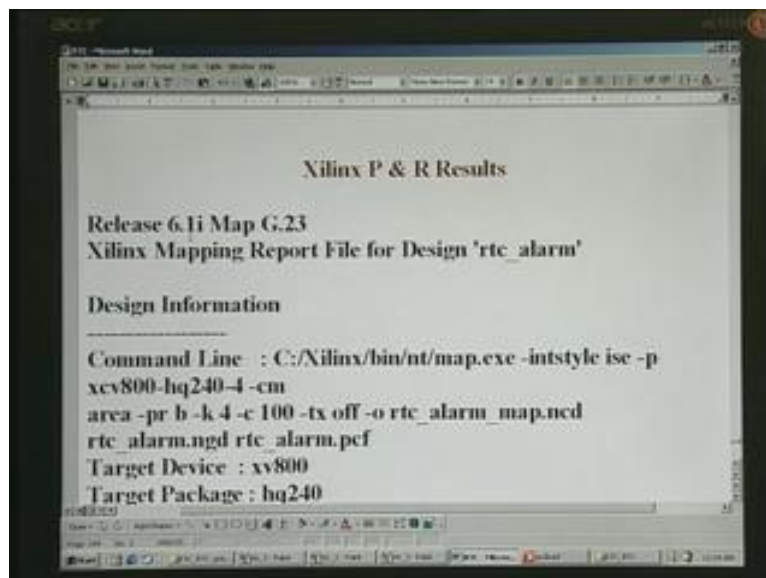
(Refer Slide Time: 15:25)





Finally, you have a report of how many LUTs have been taken by the design. It is just 3 percent of the total device because we are dealing with nearly 900,000 gates and this has taken away only 681 LUTs. Just remember 681 – we will crosscheck this figure. It was 42 Megahertz earlier. Let us crosscheck with Xilinx. It says mapper successful.

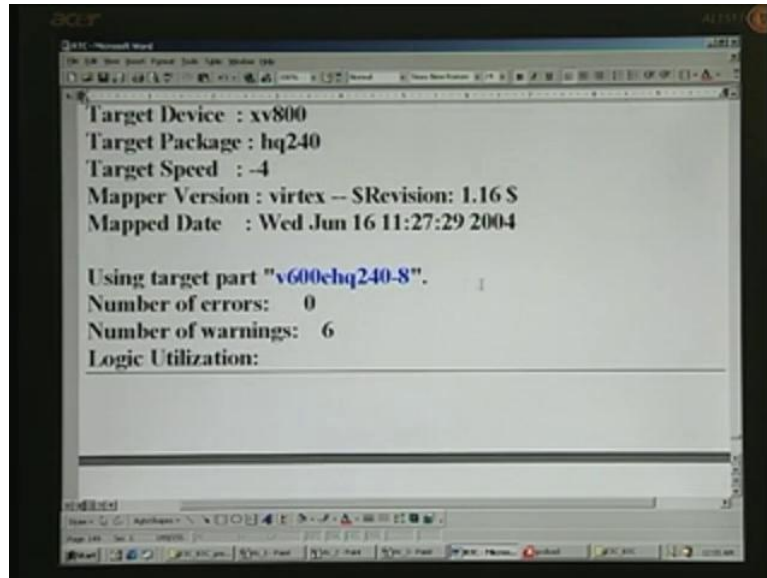
(Refer Slide Time: 15:58)



With Xilinx results, let us make a comparison. We have already seen the navigator for Xilinx using the 6.1i version. Both 6.1 and 6.2 are basically the same – not much of a difference. **Design it reports is the rtc_alarm.** The input for Xilinx place and route is the rtc_alarm.edf file, which was created by the Synplify tool. That is the synthesis tool **and it reports.** You did not have to specifically mention all this because this edf file itself will communicate to place

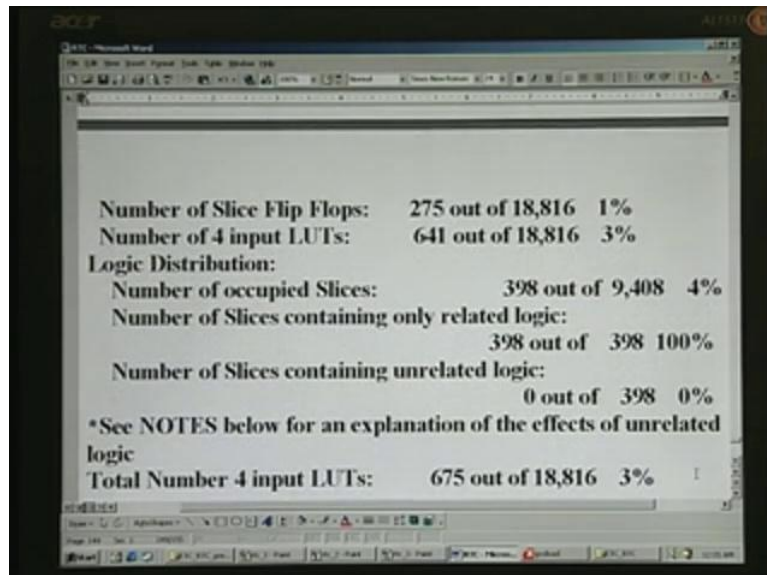
and route tool what devices have been has been chosen and so on. You can see precisely the same report here.

(Refer Slide Time: 16:43)



Once again, the target is reported here. It is also reported that there are no errors but some warnings are there. No errors anyway. We will have to look into warnings very carefully.

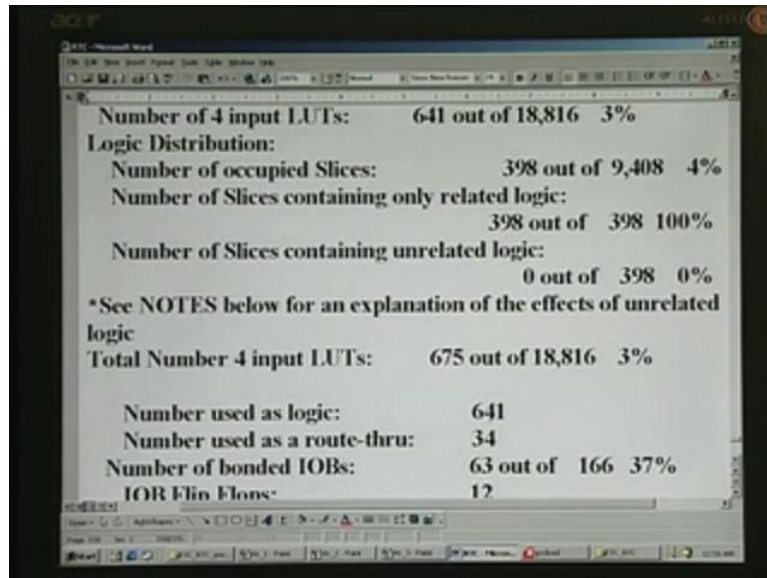
(Refer Slide Time: 16:56)



What is the logic utilization? There was nothing wrong with the warnings, it was only reporting that some bits are at 0 and therefore it has optimized and so on. Next is 4 input LUTs: only 641 have been used. This is the report of Xilinx place and route whereas it was

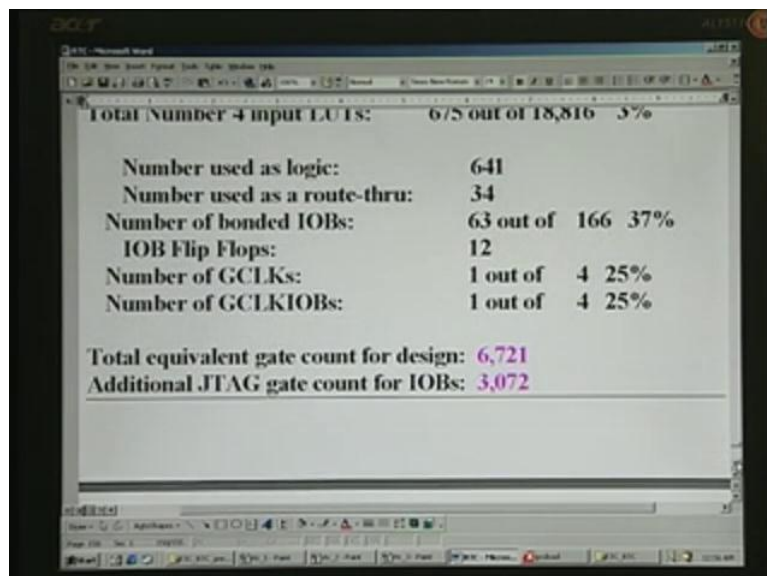
681 reported by the synthesis tool. The synthesis tool only gives a rough idea whereas this place and route tool will give the exact one because the actual FPGA **vendors' final tool** runs on that. It also has slices reported and how much it is reported there.

(Refer Slide Time: 17:43)



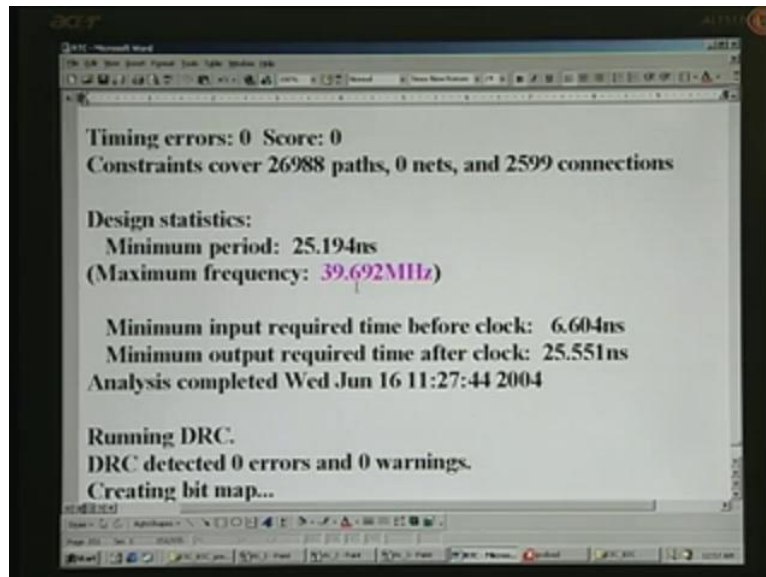
I think it was different here, 4 input LUTs, why is it reporting. It is same but some discrepancies appear here. I am not clear about it and we will have to find out why it is. Anyway, it is very close but this figure is the total number on the device.

(Refer Slide Time: 18:12)



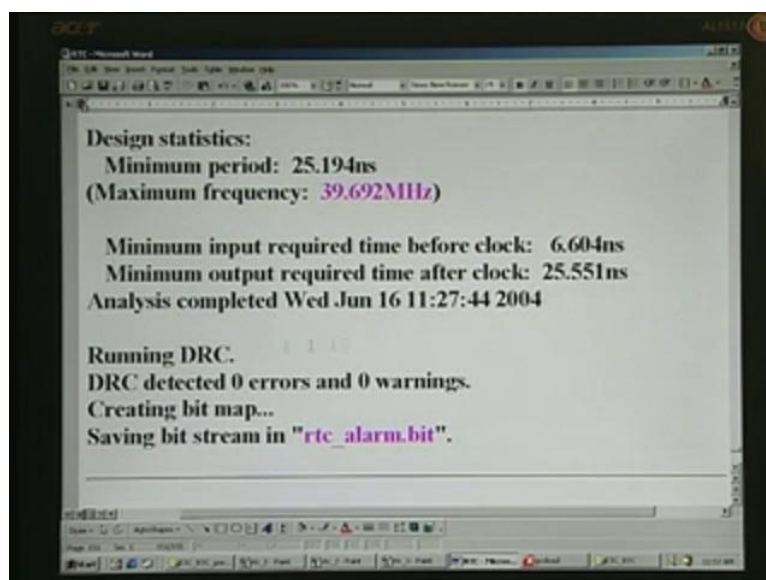
Finally, it also reports the total equivalent gate count for the design, which is nothing but 6,721 gates. In addition to this, you need JTAG IO gate count for IOBs. If you want JTAG communication for downloading bit stream through your parallel port of the computer, you also need this many additional gates. In total, it is hardly 10,000 gates that you can have.

(Refer Slide Time: 18:47)



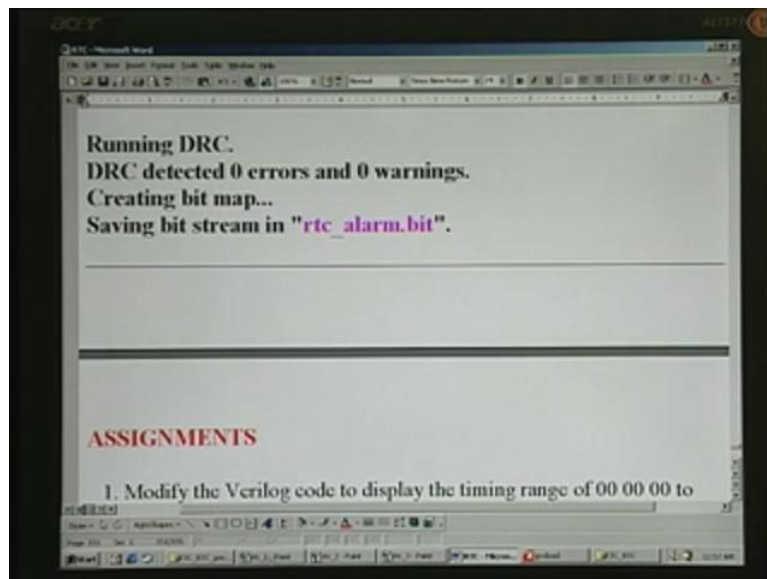
Finally, it reports on the maximum frequency of operation. Earlier, it was 41.2 or something, reported by the Synplify tool, but what is reported by Xilinx is nearly 40 Megahertz – very close to that.

(Refer Slide Time: 19:04)



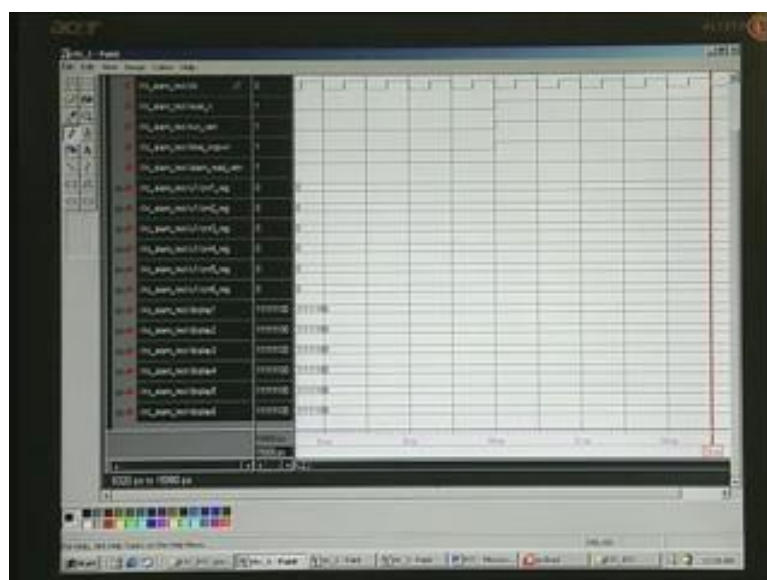
It also reports that DRC has been checked and it is saving the bit stream `rtc_alarm.bit`.

(Refer Slide Time: 19:21)



We have an assignment but before we come to the assignment, let us have a demo of this. Prior to that, let us have a look at the waveforms for this design.

(Refer Slide Time: 19:24)



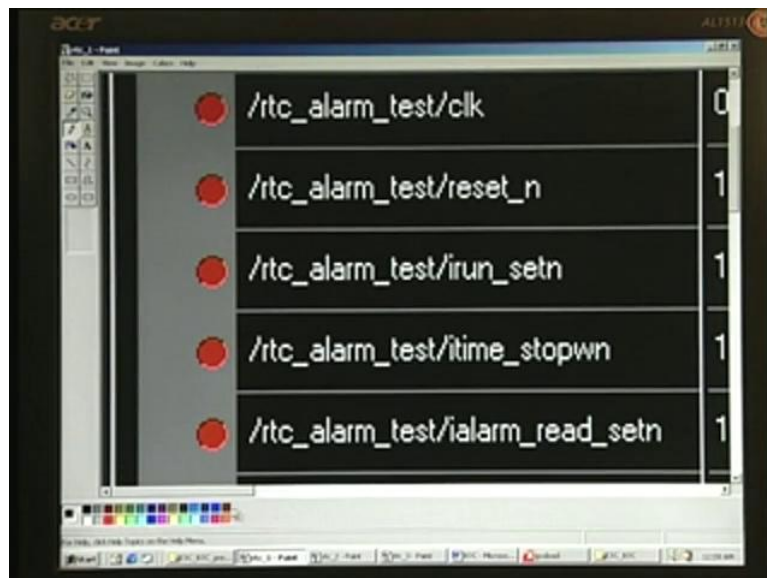
To start with, we have been **running...** the frequency of operation 1 Gigahertz. 1 gigahertz would mean 1 nanosecond. Let us see the clock. The first one is showing the clock. I will zoom it up a little later but first let me explain. You can see the clock pulse there and if you take this one, for example, it is 10 here and the next clock appears here – this is the falling

edge, so it is 1. If you take the rising edge also, it will be the same, you can see from here, **the rise** starts from here, this is also the falling edge – it makes no difference anyway. **You see that after 100 units of time only, we start the real operation.**

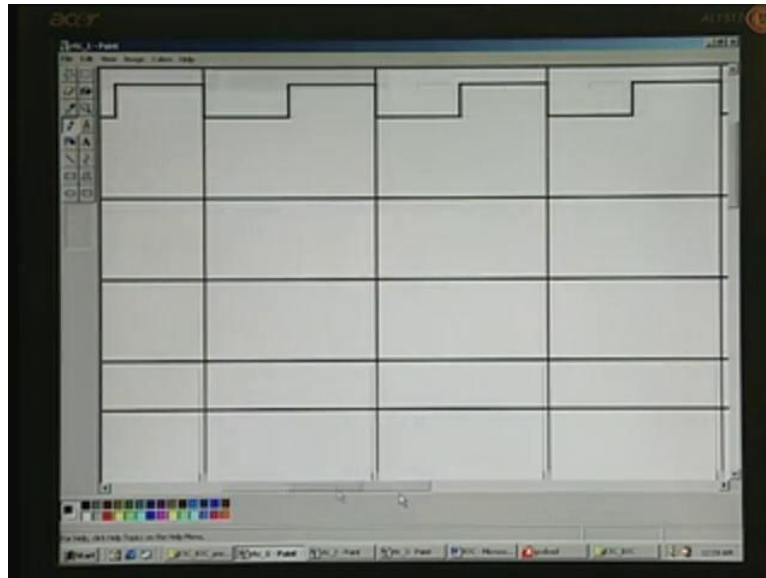
Then, reset was active here and **this was made inactive** so that the normal operation starts. Simultaneously, what we do is we set it to run mode as well as time mode – this is run and time, so it is set to 1 here. We should also set **alarm read set** to the read position – you should not put it in set mode. If you do this, you will not get this display and it will freeze at 0. You can notice that by changing your test bench. Remember that all these three will have to be 1 – that is run mode, time mode, then **alarm in read** mode. This is mandatory in order that time may run smoothly.

This happens at 10 nanoseconds. 10 nanoseconds because we have taken 100 picoseconds as time base multiplied by 100 – **we have given after 100 units of time**, that gives you 10 nanoseconds, this wave form is for that.

(Refer Slide Time: 21:20)

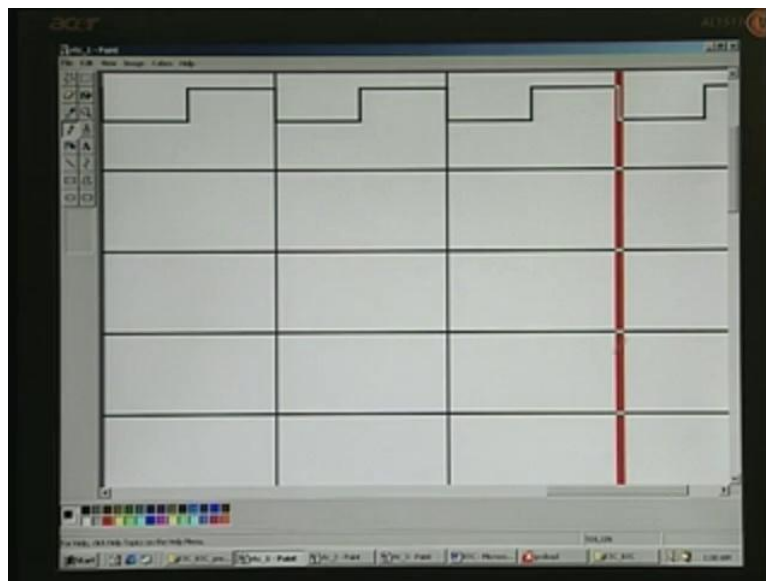


If you want the zoom version, you can see here.



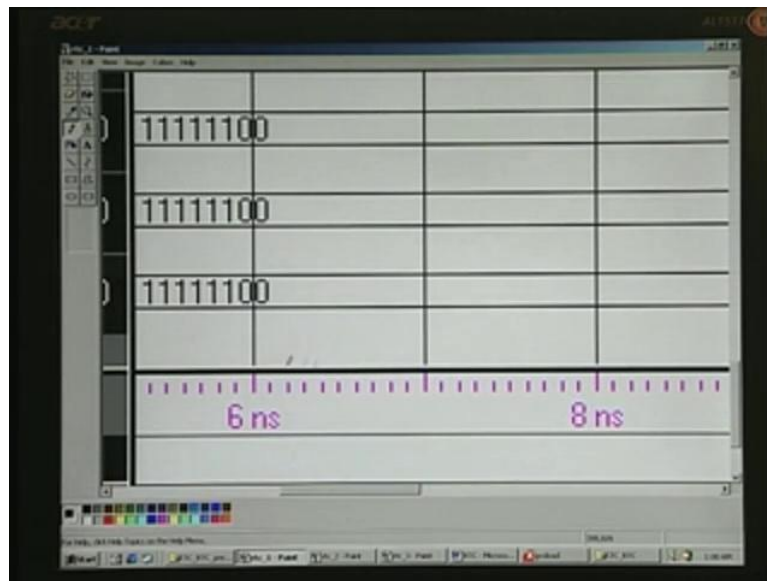
That is the clock – clock, reset, run, set, time, stopwatch, alarm. Here, you can see that all of them are 1 at the time of its working.

(Refer Slide Time: 21:37)



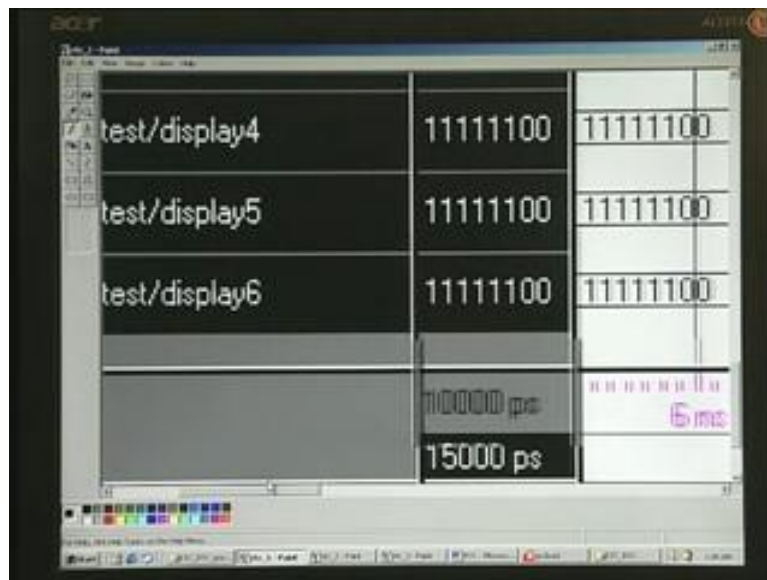
The cursor is somewhere here after it has started working – that is the point there. You can see here, every one clock is this.

(Refer Slide Time: 21:49)



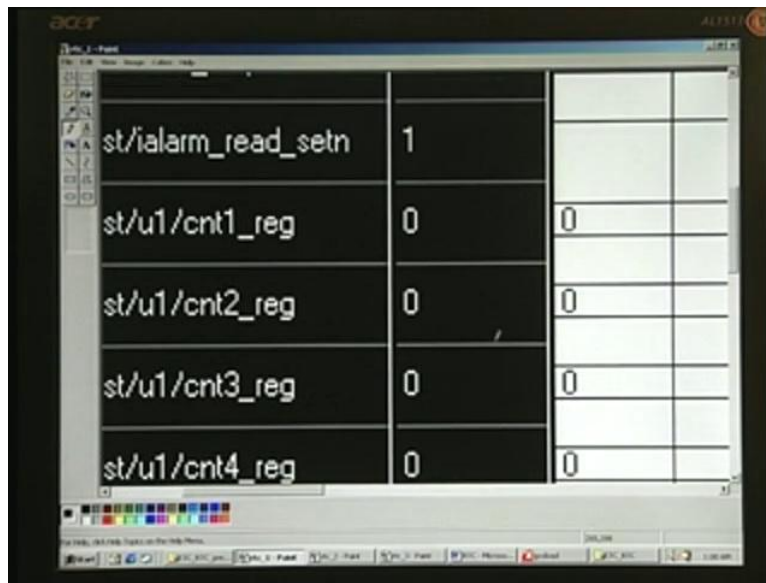
If you go to the time base at the bottom, you see that it is 6 to 7 nanoseconds. This is 1 nanosecond. In other words, it is running at 1 Gigahertz. Although it is running at Gigahertz, what has been reported by Xilinx place and route is 39 Megahertz, so it should not exceed this time.

(Refer Slide Time: (22:05))



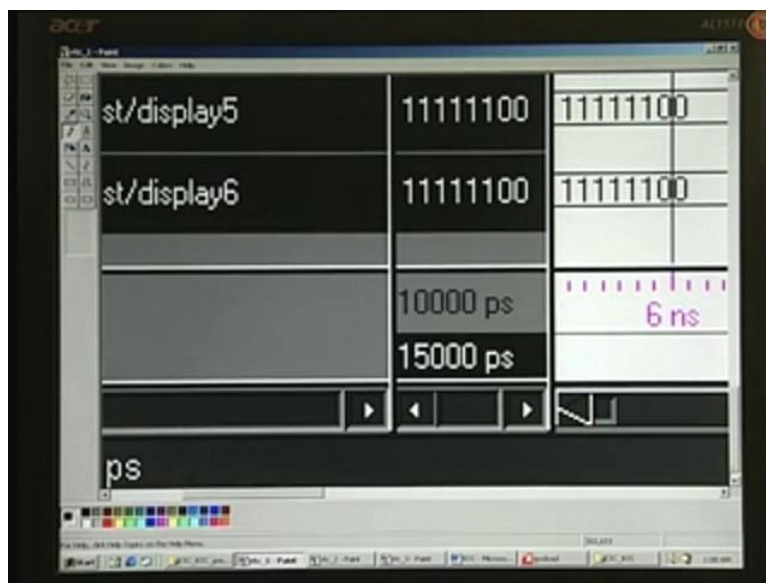
Another point **here is...** We have a display here.

(Refer Slide Time: (22:05))



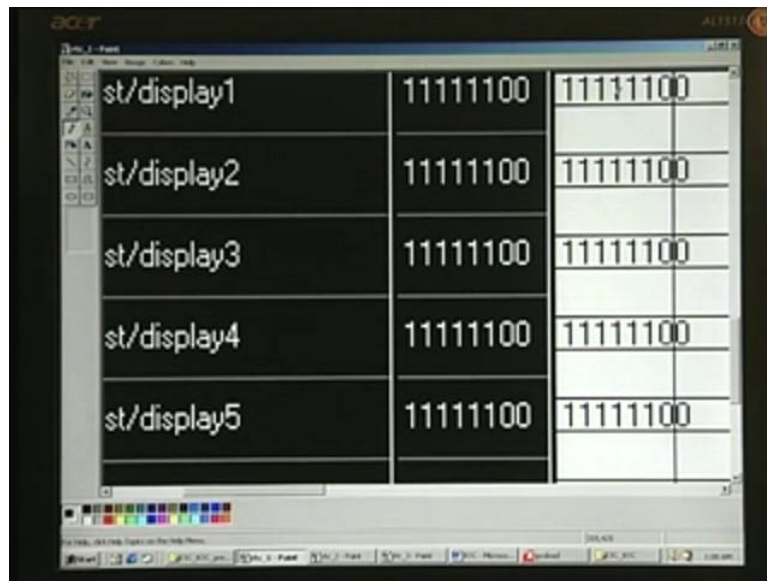
These displays are basically from the counter 1 through counter 6. They are all 0s to start with.

(Refer Slide Time: (22:18))



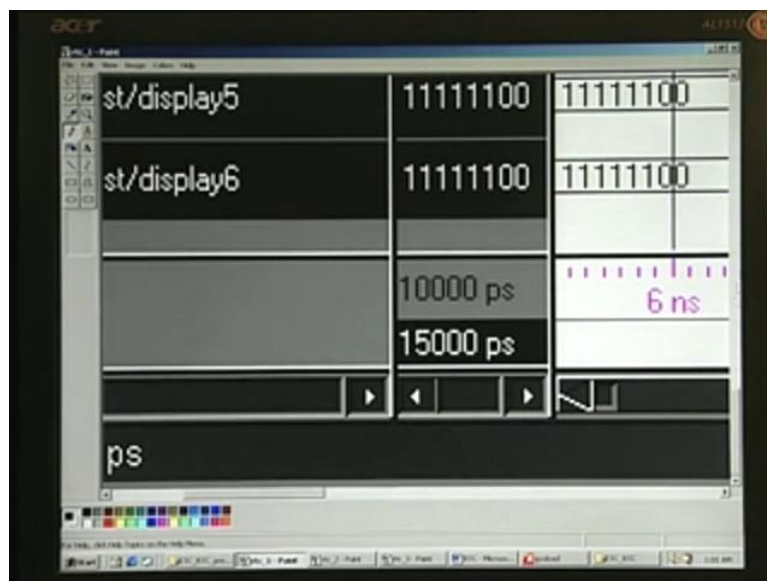
Naturally, you should have it reflected in the display.

(Refer Slide Time: (22:22))



Do you remember the displays? In all of them, you can see the same pattern right up to the 6 here.

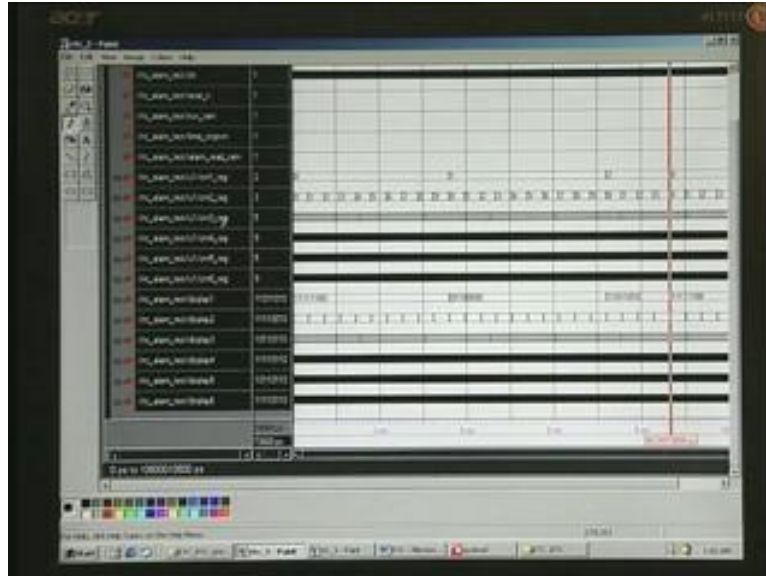
(Refer Slide Time: (22:25))



If you analyze this, these are all nothing but a, b, c, d, then e, f – up to that, all are 1. That means these segments are on. In other words, it is nothing but 0. This one is the g segment, which is the center segment of the seven-segment LED display and this is off. Naturally, it will automatically display, which is in agreement with counter 1 through counter 6 all having the same thing. When you get to the fag end of this waveform, you will see differences,

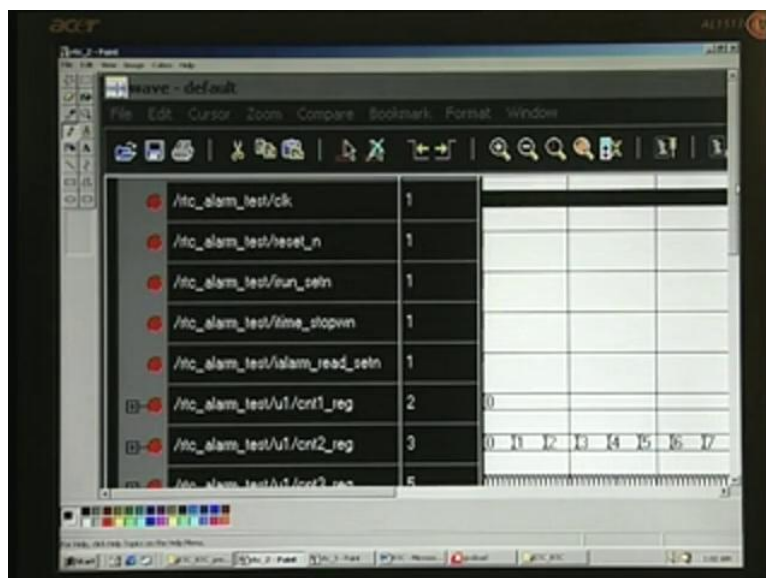
reflecting counter 1 on display 1, counter 2 on display 2 and so on. We will go for the next waveform here.

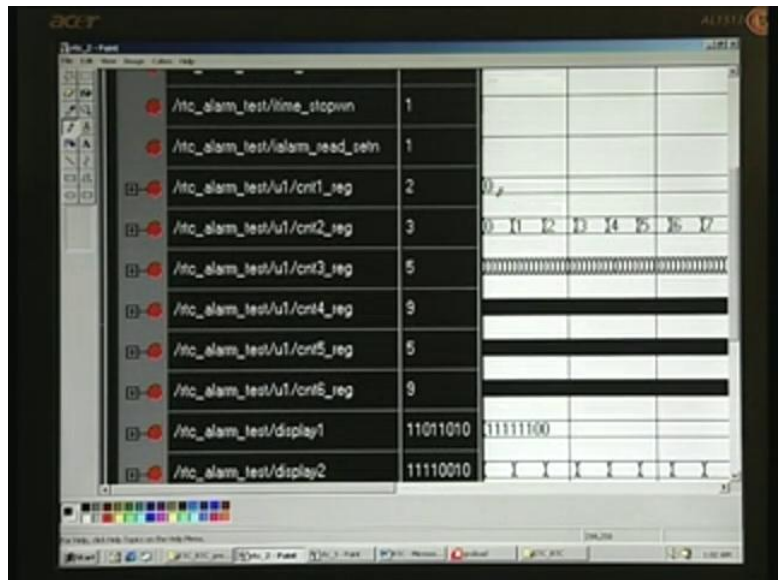
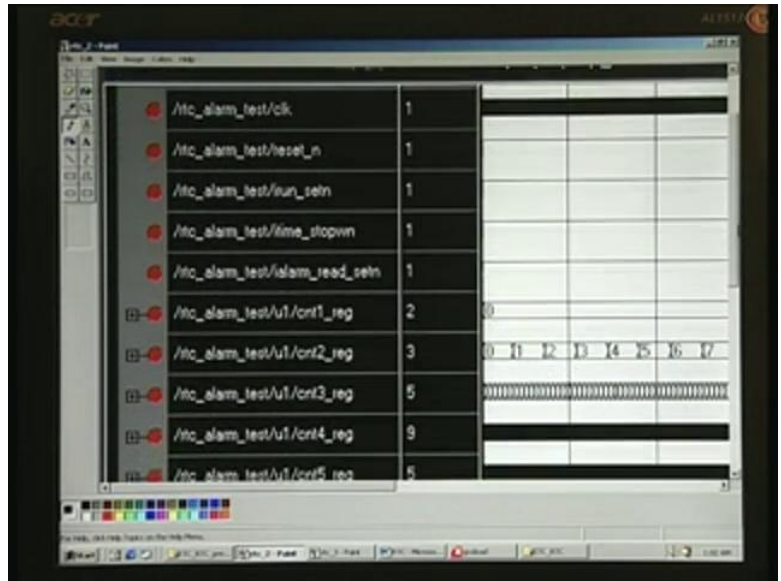
(Refer Slide Time: 23:04)



Here, what you get is the entire cycle completed. The real-time clock has run right from 0 through 1, up to 2. This one is right up to 23. We have 23, 59, 59 – 23 hours, 59 minutes, 59 seconds starting right from 0 here. These are all too high, you cannot notice this here, but you can notice this 23 here. Once again, if you see this a, b, c, d display... In the next waveform, we will analyze to see whether it is tallying with the counter 1 and so on. This is precisely what you are going to get in the next display output. I will zoom this before I close this.

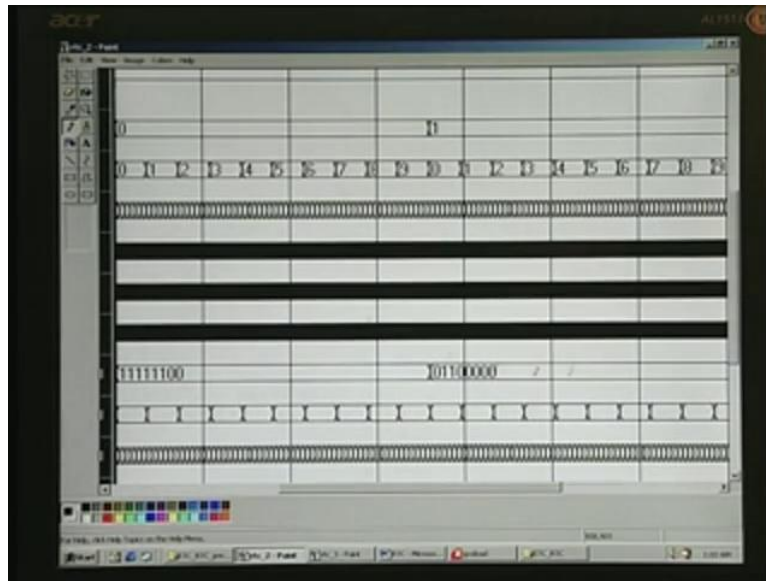
(Refer Slide Time: 24:06)





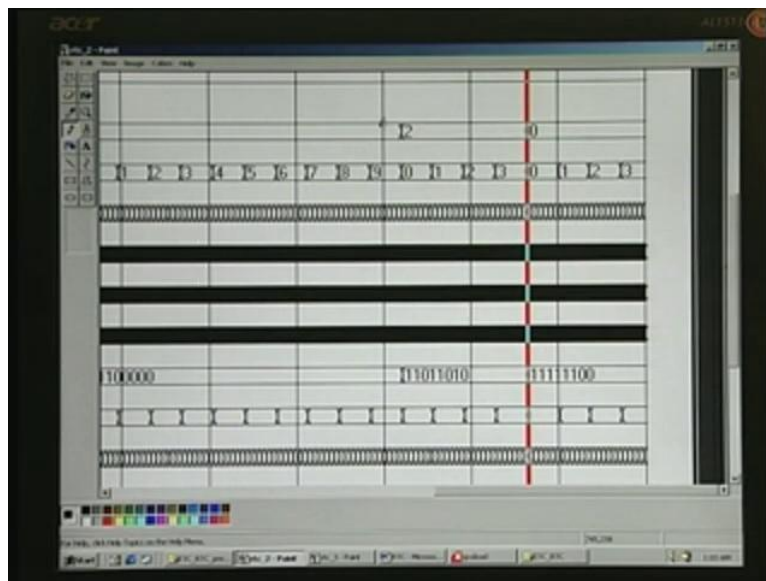
You can see clock, reset, run, set, etc. You can see clearly 0 through display 1 through 6 here.

(Refer Slide Time: 24:25)



There is a counter going right from 0 through 9. This is counter1, this is counter2 and it goes right up to 9. Once again, it goes from 0 through 9 when the counter1 is 1.

(Refer Slide Time: 24:38)



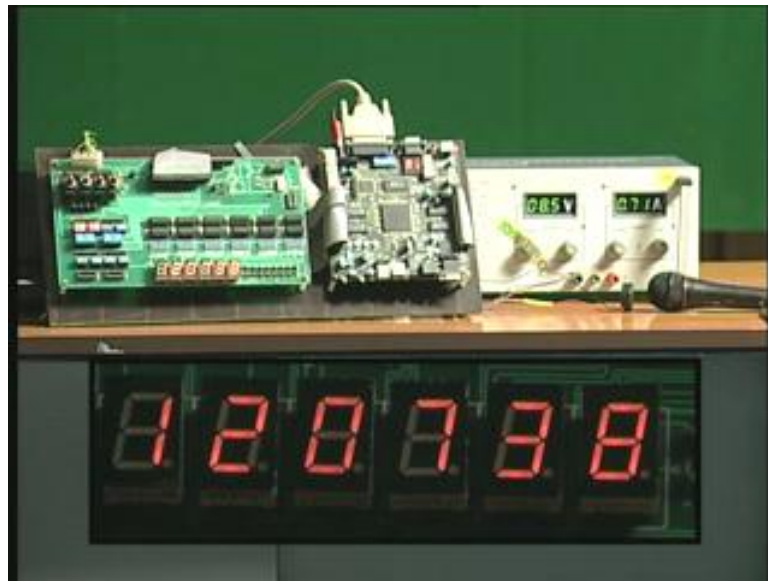
Finally, when it is 2, this goes only up to 0, 1, 2 and 3. After 3, it immediately resets to all 0s. The cursor is here.

that means it is displaying 0. This is in conformity with this 0 (Refer Slide Time: 25:44). The second 0 stands for the display2, counter 3 for display 3 and so on.

Let us take this 23, 59, 59 and then correlate. This counter1 must be having one-to-one correspondence with the display1. Let us say here a, b... c is 0, just keep track of this, then d, e, then f. c and f are 0, this should mean 2 because the counter1 value must be reflected. I will just draw it for you here. This is 2 here and if you take this, this is a segment (Refer Slide Time: 26:21), b here, then c is this one, c is 0 – that is what we have here, then d here, e here, f here, f is also 0. You can just see here f, then g; c and f are 0 here. If it is 3, what you should have is a, b, c, d, this is e and this is f. e and f must be 0. That is for a, b, c, d, here e and f – you can see this. This corresponds to display2 or counter2 – there must be one-to-one correspondence. This is a seven-segment display and 1 lights up the LED – remember that. 3 is also verified. Now, let us go for 5. Here, you see b is absent, this is d, e. b and d are off, that is 0, let us make sure about this. That is display3 and that is 5. display3 is here and b is off. Then c, d, e. b and e, is it correct? b, c, d, then b and e you can see. That is why 5 is displayed.

Now for 9, what you have here is you do not have.... This is c, d, e; e is 0, so let us make sure and crosscheck at display4. This is a, b, c, d, e. This alone is 0. Do not be confused about this 0 because this is only for decimal point. Anyway, we are not turning on the decimal point and that is why all of them are 0. This confirms that there is a one-to-one correspondence between counter1 and display1 and so on, right up to counter6 and display6. You can see that it has taken 8.64 milliseconds. Next, we will have a demo.

(Refer Slide Time: 28:21)



What is shown here is the hardware. You can see the FPGA board that we have used earlier. This is the FPGA board. The centrally located chip is the FPGA and towards your left is the digital I/O card. Note that the cable is going from here and getting connected to the extender I/O port of the FPGA. Part of the cable is here on the left, there is a connector there and another connector here. Both are extenders, which you will be connecting to the external world as far as the FPGA board is concerned. That means the I/Os are connected here.

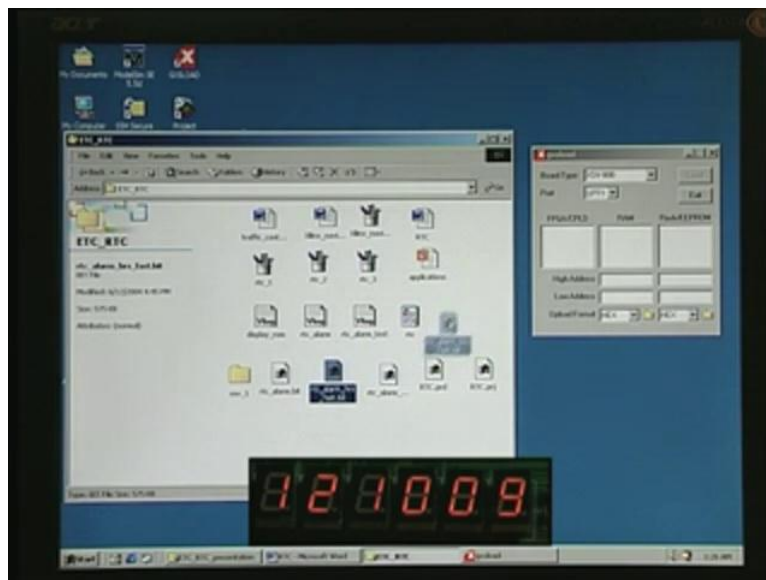
This is the display that we are going to use – two digits for hours, then minutes, then seconds here. Since it is not clear here, the exploded view is shown down at the bottom. Whatever you see there is actually functioning. Right now, the real-time clock is running here – hours, minutes and seconds here. Notice that there is a power supply, there is also another power supply not visible to your view – just at the rear end here. There is also a buzzer there – this is a piezoelectric buzzer. In front of that, there is a mike to pick up when the buzzer gets activated. We will see many modes – under which conditions the buzzer will be activated. To start with, what we need is a load. On the system that you see here, there is a **GAX load**. Just click on that one. It opens this window here.

(Refer Slide Time: 30:08)



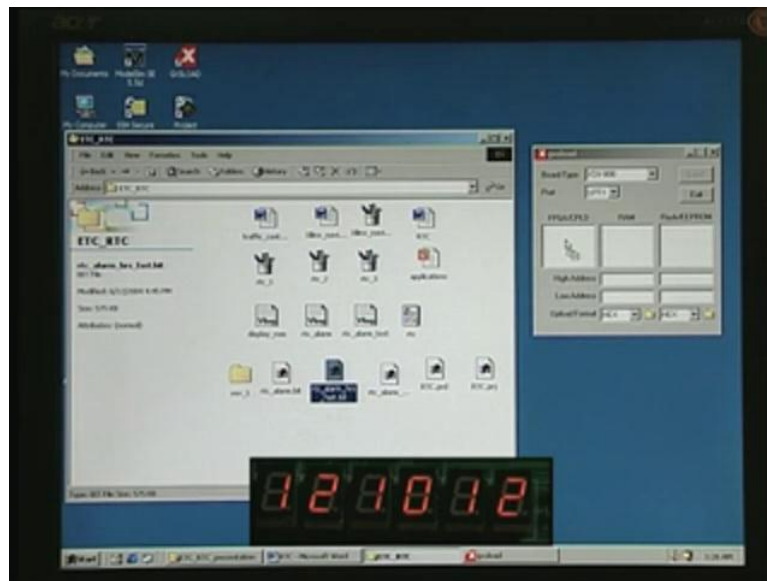
This window has different bit streams available, which is the result of Xilinx place and route. This is for the rtc_alarm.bit. I have also made 1 hour fast as well as **minute fast**. This is to help us quicken the demo, to test the real working by speeding up the clock, so that we can see hours as well as minutes very rapidly. For example, first, let us download these **hours fast**.

(Refer Slide Time: 30:53)



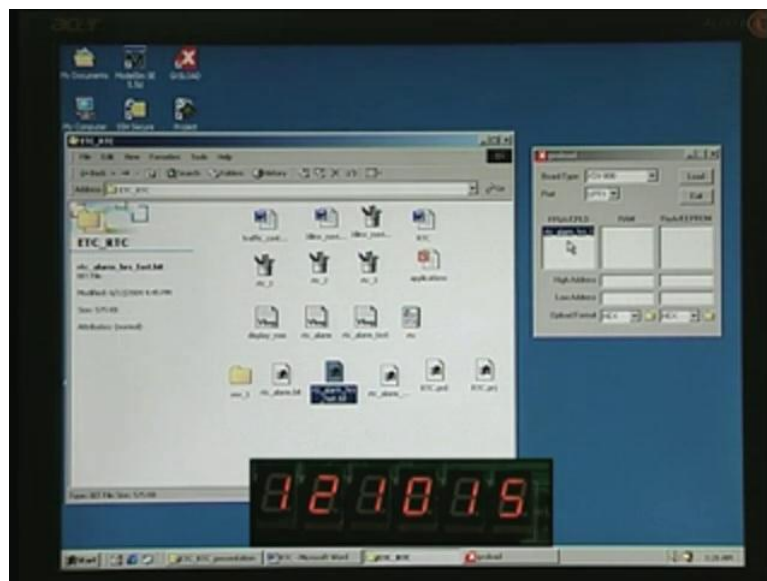
With this condition, let us see what will happen. You just hold. Do not release the mouse button.

(Refer Slide Time: 30:57)



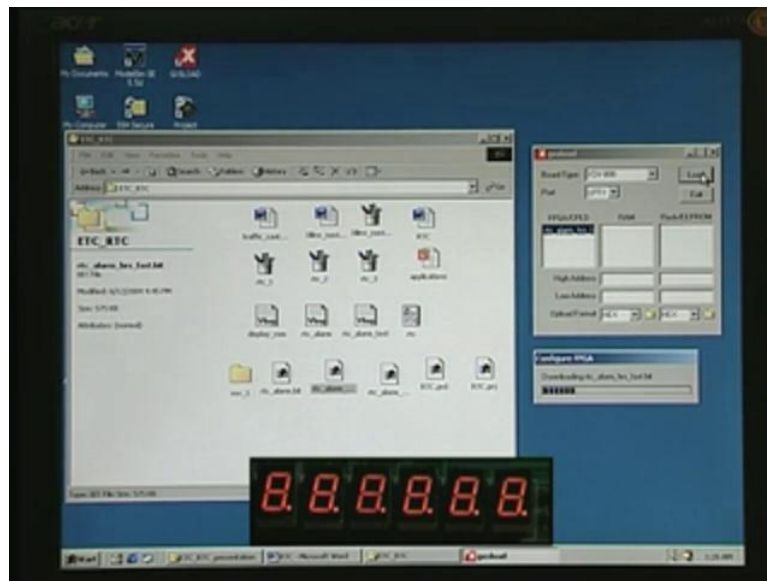
When it comes to this window, release that here.

(Refer Slide Time: 30:59)



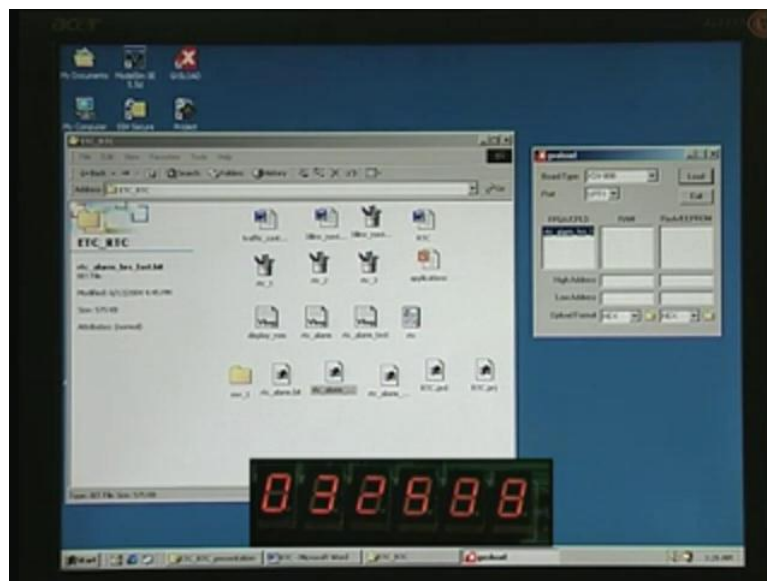
This is the bit stream we are going to download. There is one load button here, just press this.

(Refer Slide Time: 31:05)



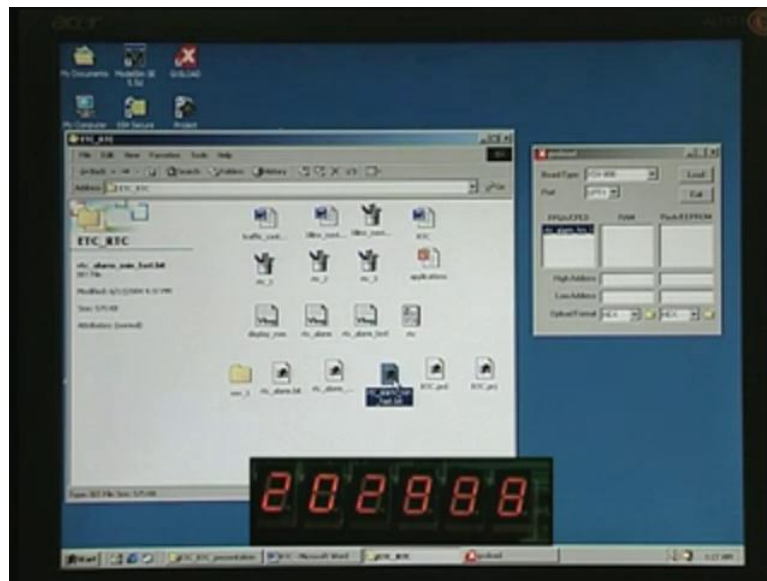
The buzzer is also sounding. The reason is at the time of configuring, the counter is all reset.

(Refer Slide Time: 31:16)



Now, you notice that the hours is running fast. You can see it started with 0s and it is counting 1 at a time. It will go right up to 23 and then roll back to 00. This will check these hours, which will normally take 24 hours for you to check but we have taken hardly half a minute. You have seen one revolution complete. Now in order to check the minutes, a similar thing will be observed, this is the 2 minutes display. For that, what you need to download is this rtc_alarm minute fast.

(Refer Slide Time: 31:57)



We will once again drag and drop it here and then load this.

(Refer Slide Time: 32:03)





It is downloading that particular **minute fast** bit here and **it starts with...** that buzzer was on once again. I hope it is audible to you. You can see now two digits just below this – two digits are going on, one after another and it will go right up to 59 seconds and then roll back. In addition to this, we have the usual **alarm.bit**. We will do the same thing.

(Refer Slide Time: 32:40)



We will drag and put it over here. You can observe the minutes going right up to 59 and then rolling back to 0. We will wait for some more time for the next thing. We will act on this **alarm.bit**, which is the normal running in seconds, minutes and hours and is the real-time clock. We will take it up soon after it completes one revolution. 59 now, 0 here and 1 hour it

has advanced. Like this, you have checked first the hours, the two digit whole sequence you have seen as well as the minutes.

Now, what is to be checked is only the seconds, but it is too fast for you to see. Maybe you can see the second digit of the seconds going. In the normal mode, we will see that. Now, what we have to do is download this [alarm.bit](#), which is the normal mode.

(Refer Slide Time: 33:43)



We are downloading now by clicking on Load.

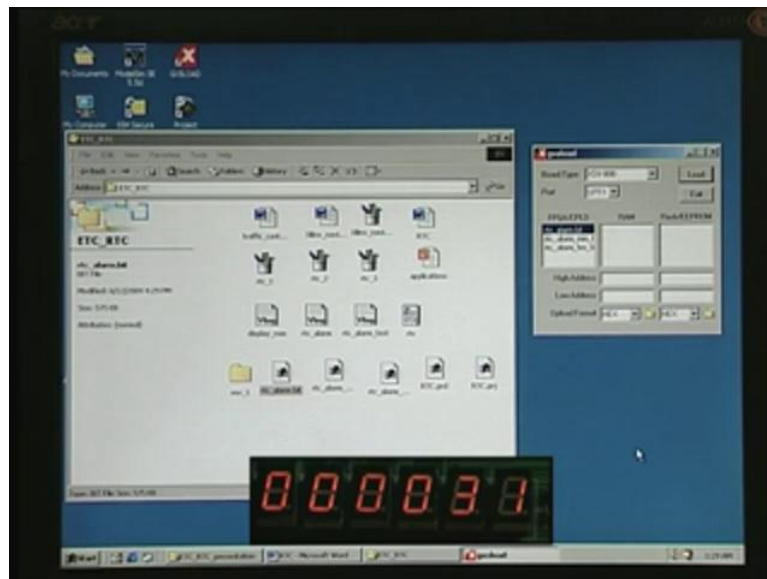
(Refer Slide Time: 33:54)



Now, it is normal time. You can see that for every one second it is making the change and the alarm is on. It is the beeping sound that as we have already seen in the design and this is programmed for 30 seconds.

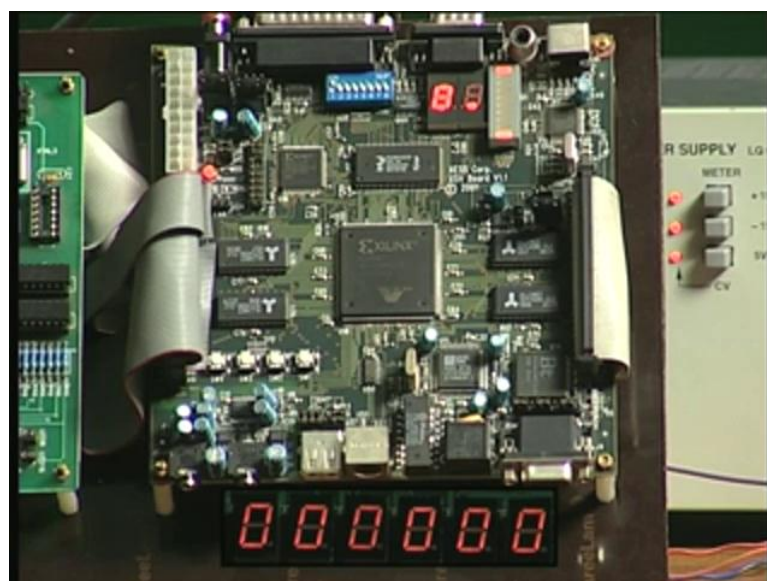
Now, what we have to do is we will do the setting for a particular time – real time is not this correct because all of them are 0.

(Refer Slide Time: 34:21)



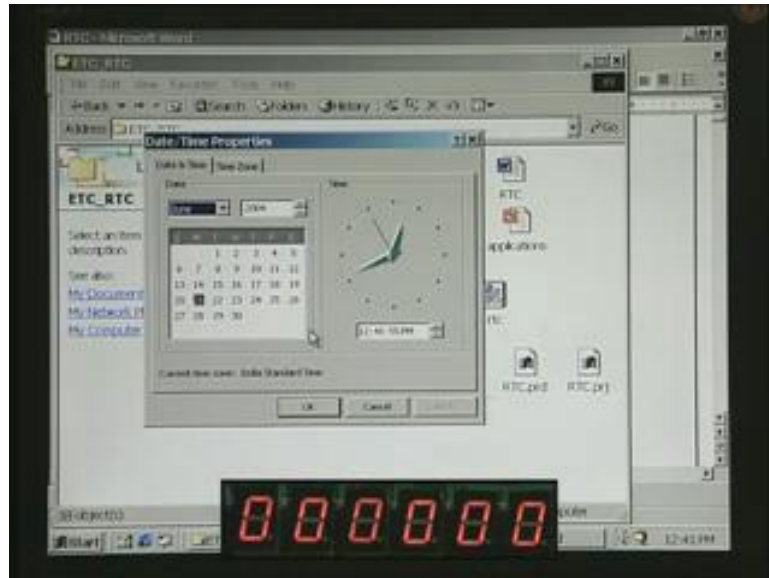
The buzzer has stopped after 30 seconds. Now, let us say we want to set it to the desired time. I will first invoke the time here.

(Refer Slide Time: 34:36)



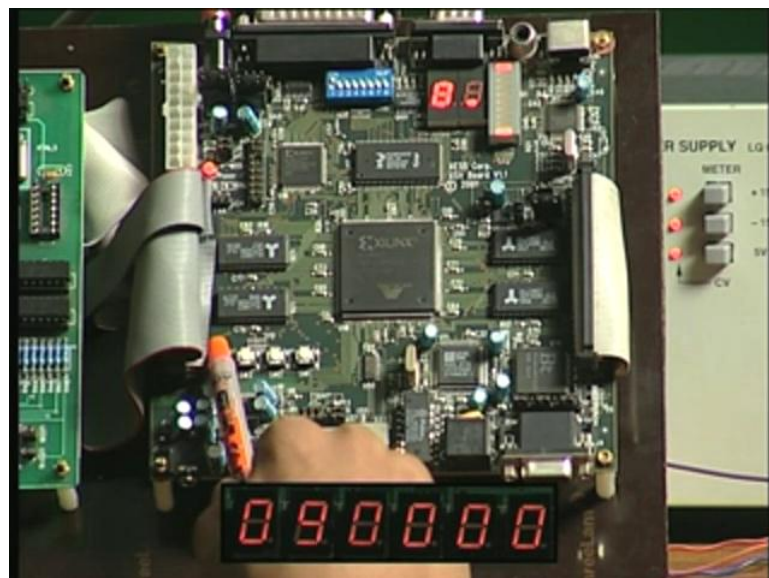
We will first set the hours setting here and it is right here. We have a DAP switch here and down, we have four push button switches. We need to use the left push button switch on the left in order to do the setting. We will now do the setting as per the system clock that you see right on the computer.

(Refer Slide Time: 35:02)



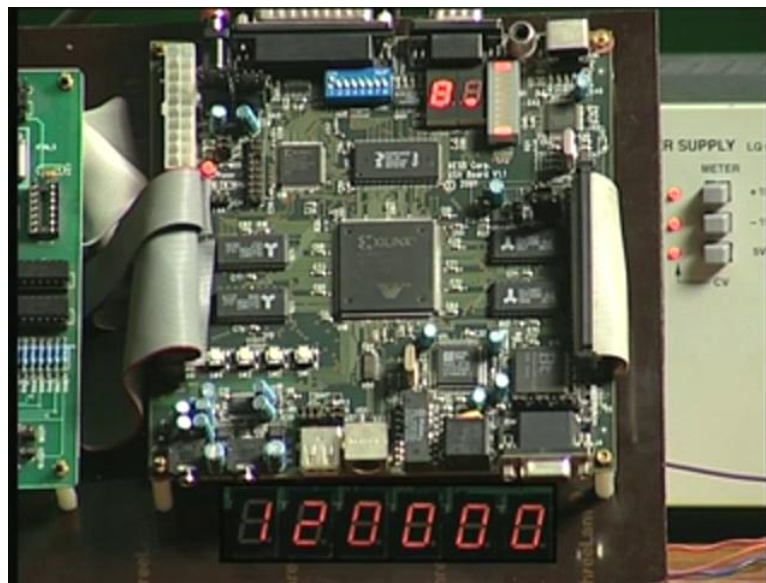
It is now 12:41, 12:42. We will set it for 12:43. Now, hours will be set.

(Refer Slide Time: 35:10)



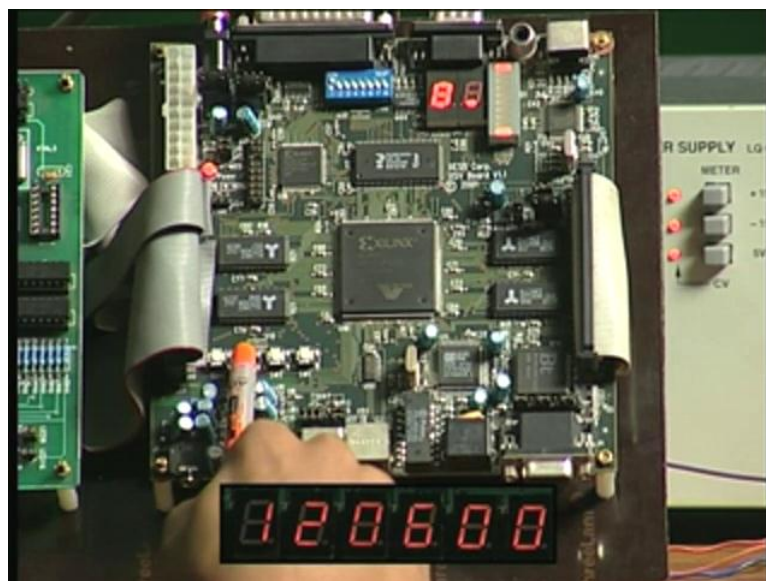
You keep it and hold it so that it runs.

(Refer Slide Time: 35:18)



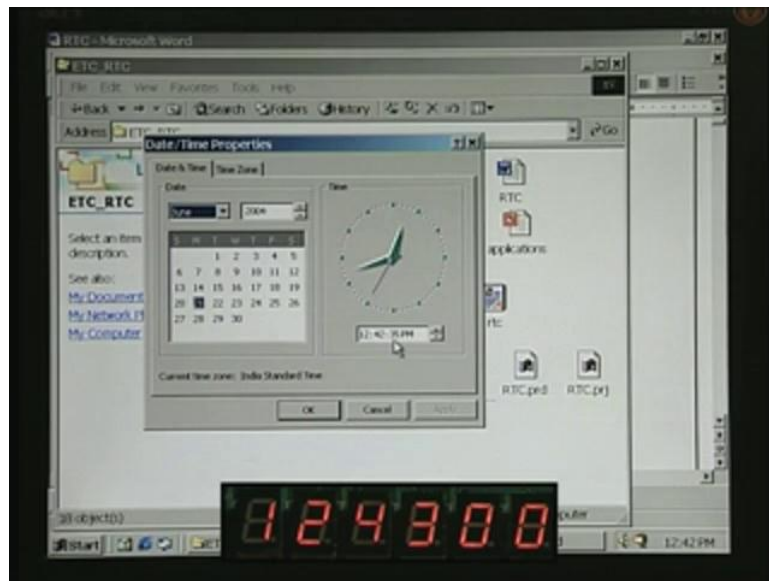
When it closes by, 12 is set. Now, minutes.

(Refer Slide Time: 35:18)



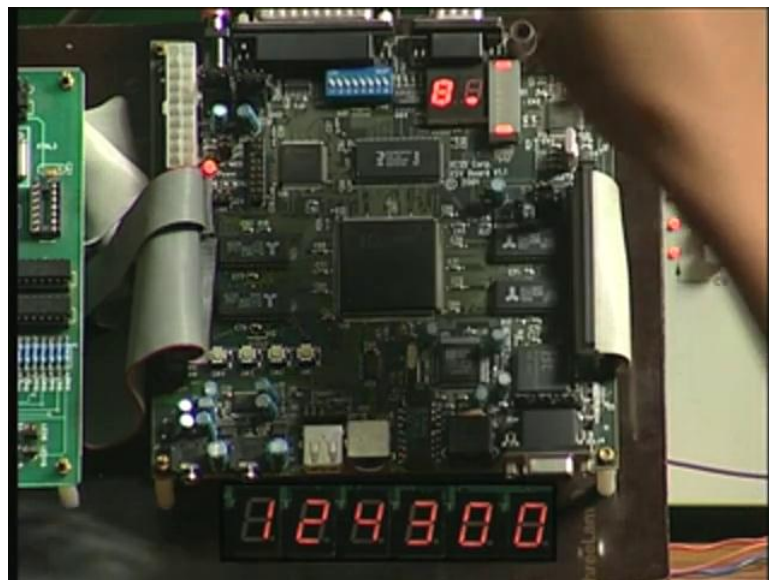
We keep it pressed so that it counts fast. After a delay of 2 seconds only, it started. When it [35:31], we will set there 43. Now, we have another 30 seconds. Look at the system clock there.

(Refer Slide Time: 35:38)



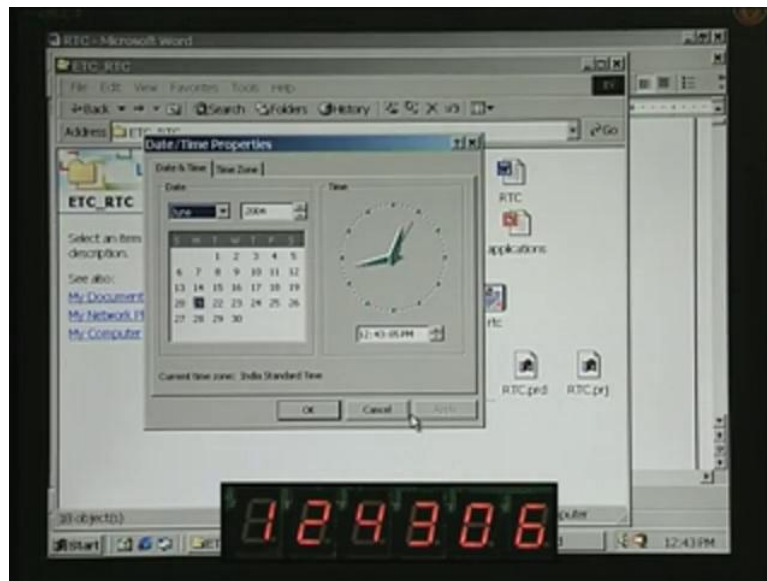
I will count. 35, 36, so after this 42, 38 now, right now. I will count and our friend will synchronously switch on that Start button and we will have a look, 51 now. I am counting now: 53, 54, 55, 56, 57, 58, 59, yes.

(Refer Slide Time: 36:03)



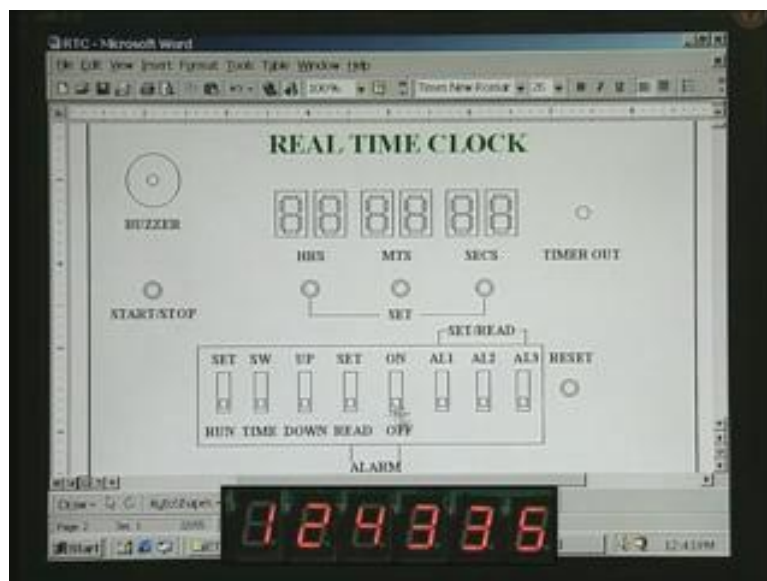
It is 43:04 there.

(Refer Slide Time: 36:09)



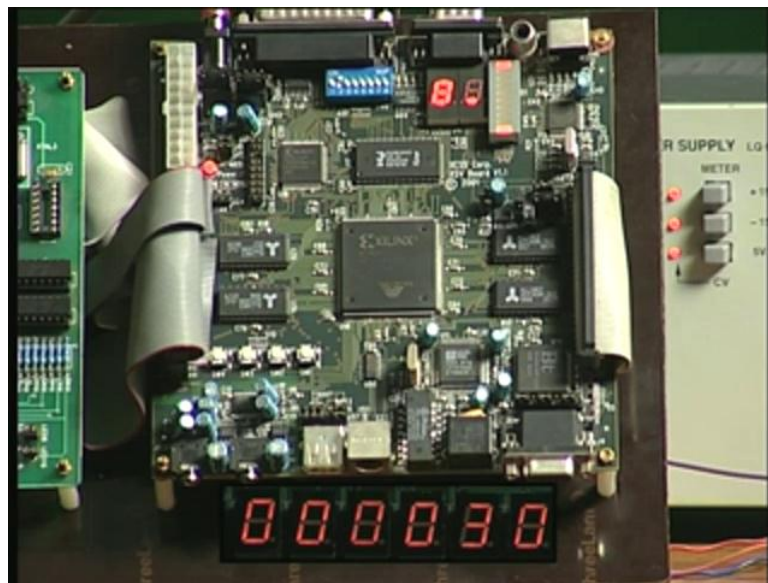
On the system also, 06, 07, 08, 09, 10. Some slight discrepancy is there because human beings cannot set that clearly and the control that we have is on this.

(Refer Slide Time: 36:26)



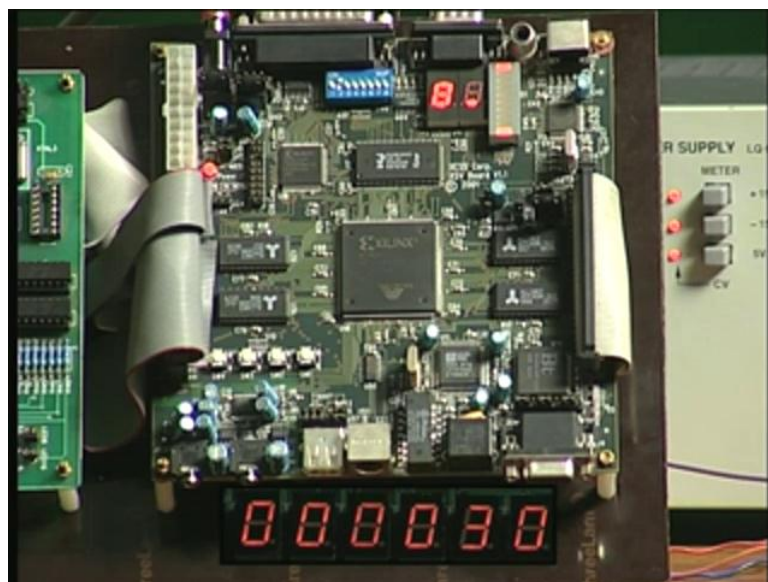
This is the control that we will have to operate. Hours, minutes, seconds is the display there. These are all the settings. It was put in set mode and time mode. Set time is what we have seen, using which we have set. The other modes we will see **once again come back to this**. You are seeing the timer running. Next, we will take down counting and we will set for the **down-counting**. In order to do this, what we need to do is put **set stopwatch mode**.

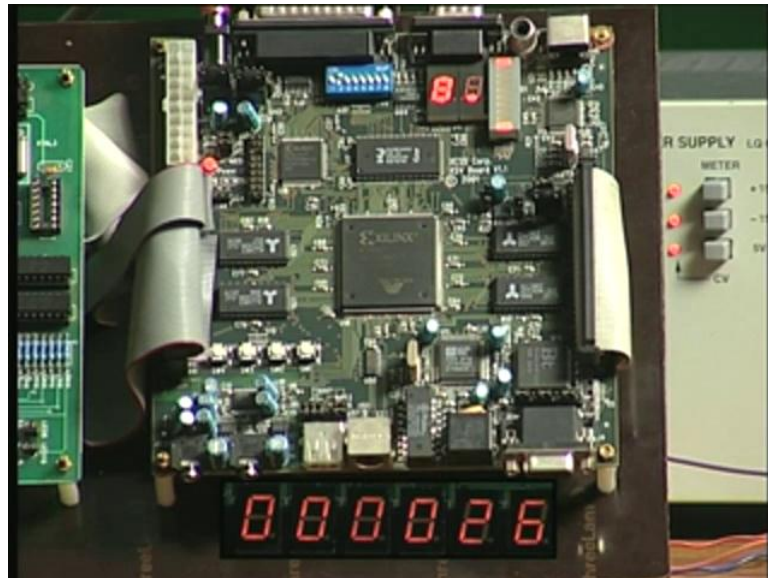
(Refer Slide Time: 37:03)



We will now set the stopwatch to 30 seconds and see what happens. It is in down mode and it has been already set by using once again hours, minutes, seconds push button switch – in seconds, we have used and it was in **set stopwatch mode**. Now, we will start the down-counting mode. By pushing the Start button, you can start counting.

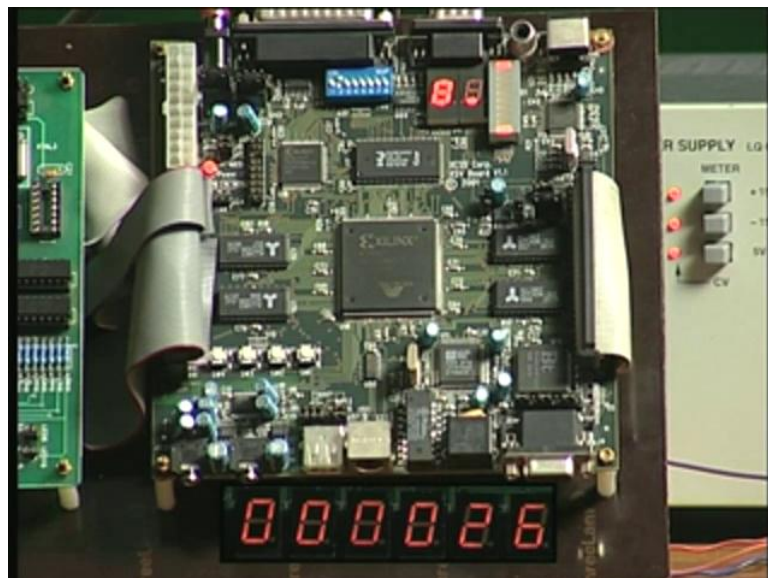
(Refer Slide Time: 37:34)





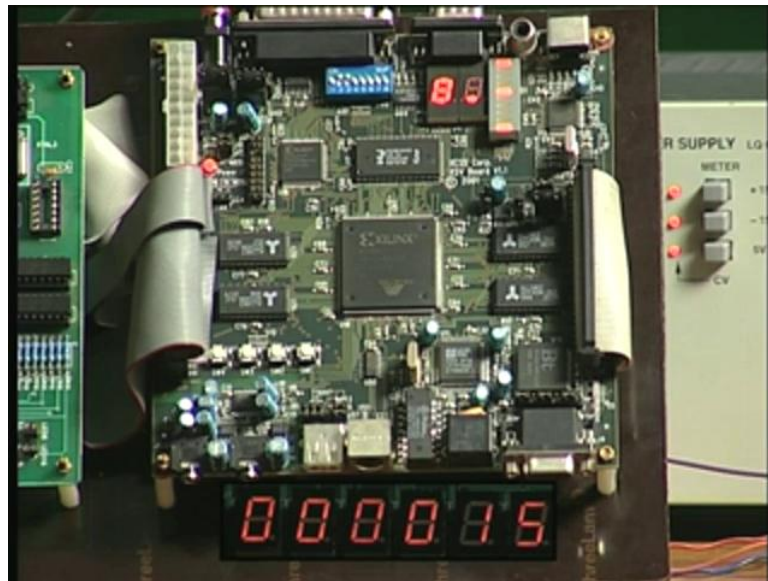
You can now see after it was started 27, 26 and so on. When it touches 0, it will freeze on that point and the buzzer also will be activated. Let us wait for some more time. After this, we will be taking up the up-counting mode. We will use the real-time setting for the up-count mode. Just wait for 5 seconds 2, 1, you heard the buzzer and it froze there. The buzzer sounds for 30 seconds, we will cut it short anyway. We will set 15 seconds for the **up-count** and now we will start in order to start the up-count mode.

(Refer Slide Time: 38:21)



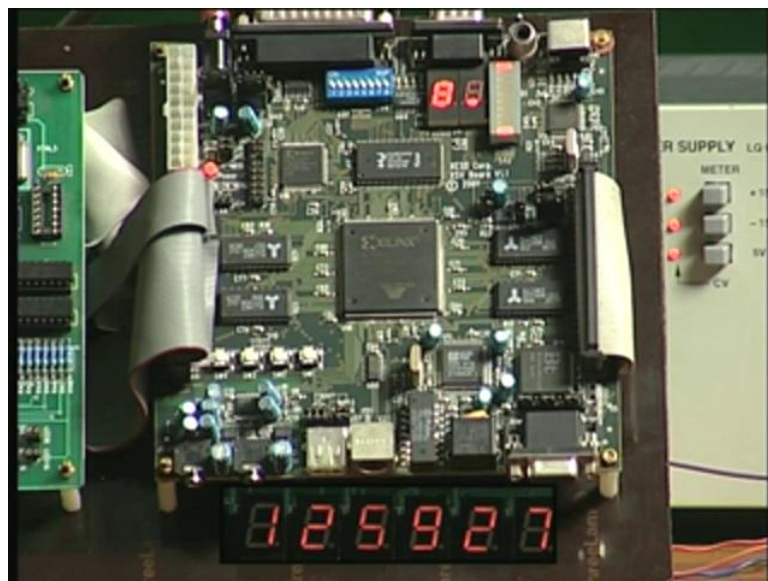
Now, it is counting 1, 2, 3, 4 and right on the top, **one bar display** is there. Watch for one small segment light up **after...** You can see that segment has come. In fact, I forgot to tell you on the down-counting mode – then also it came on.

(Refer Slide Time: 38:48)



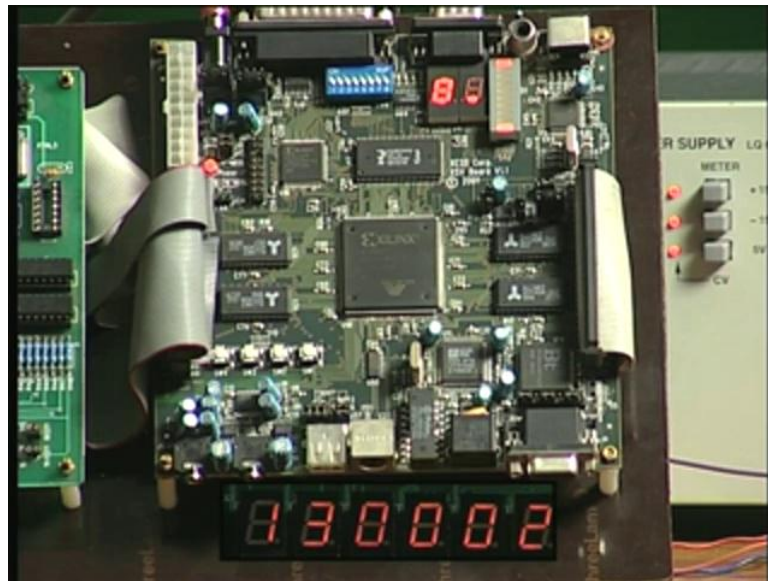
After 15 seconds are over, the buzzer has been activated and count display has frozen to 15 seconds.

(Refer Slide Time: 38:50)



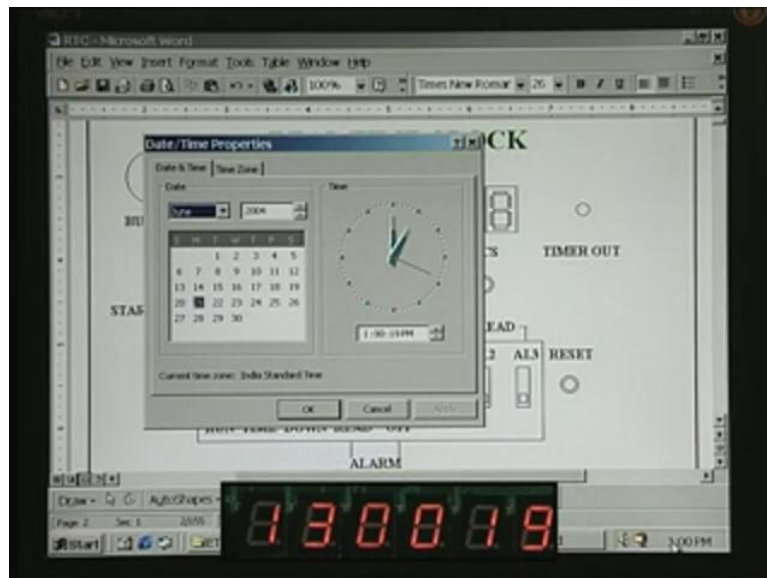
Three alarms have been set for 1 o'clock – afternoon of course and then 1, 1 – 1 minute I mean, then 1, 2, so 1 hour and 2 minutes – that means 13 hours. Looking at the actual real clock that is on your screen right now, it is 46, 47. When it touches 0, 0, 0, the first alarm must come on, the buzzer also must come on. Let us wait for this.

(Refer Slide Time: 39:26)



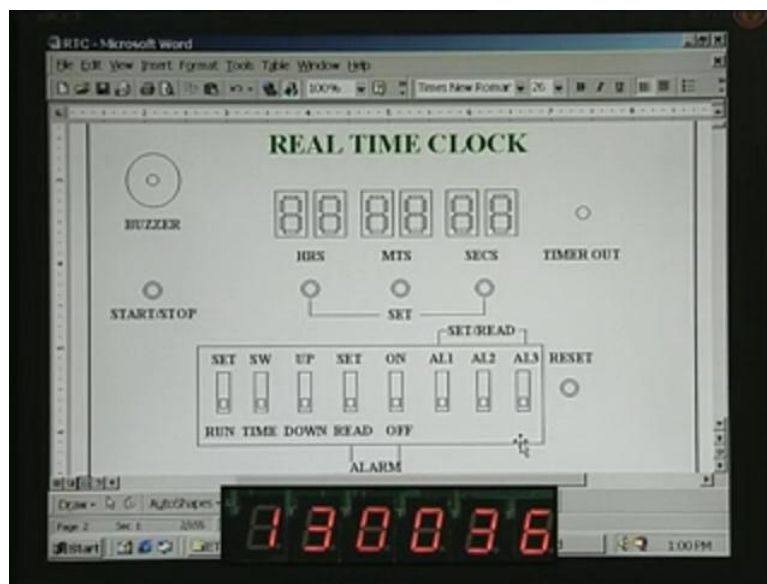
The alarm has come on precisely at 13 hours and it will go on right up to 30 seconds approximately. After this, you will be getting the second alarm – alarm 2 has been set and you can even see it on the computer.

(Refer Slide Time: 39:43)



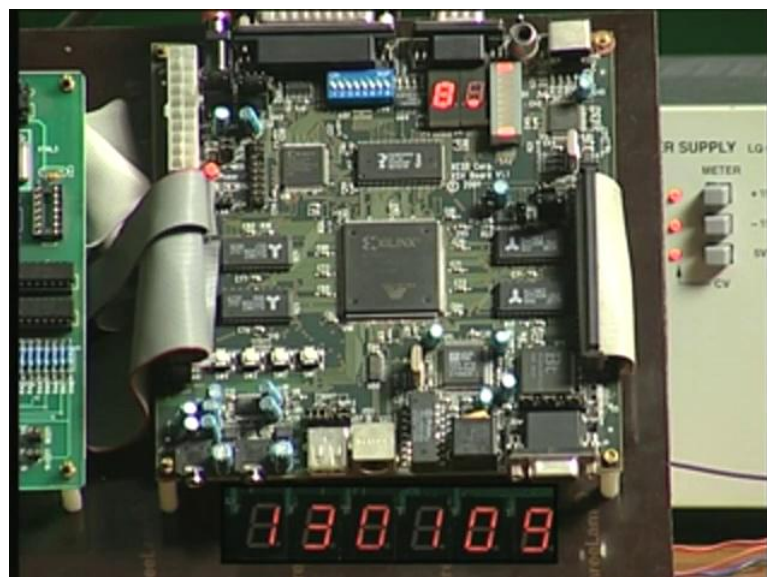
You can see the same time here. This proves that the timer that we have set right at the beginning has been running in spite of the fact that we went into up-count mode, down-count mode and so on.

(Refer Slide Time: 40:00)



Let us have a look in the meanwhile at the settings just to recollect how you had set. We had run and set and we are going to wait for the second alarm to come at another 15 seconds. Till that time, you see here. We have used **set stopwatch** earlier and now in order to set the alarm, we need **set position here on**. You see exactly that at 1 minute, the second alarm turned on – alarm 2.

(Refer Slide Time: 40:32)



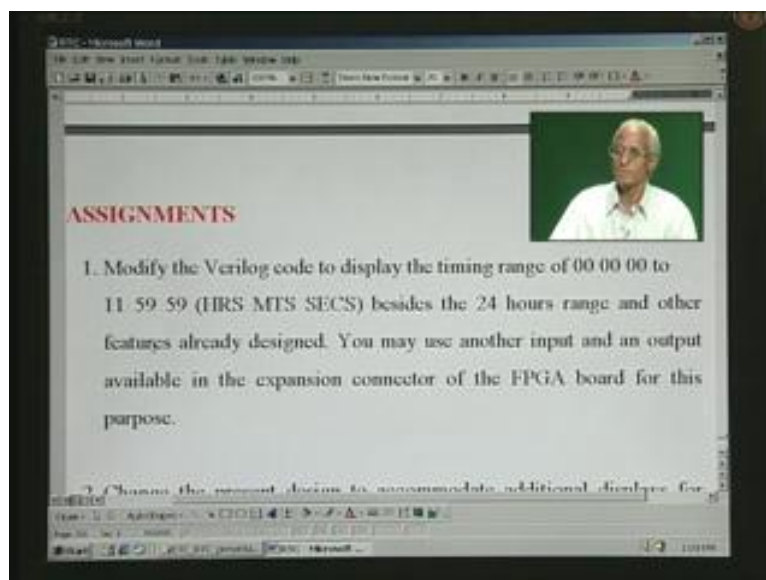
That was this one because using this you can set actually. This must be in set position and if you want to set alarm1, you should take it here. Then it will be automatically set to whatever was there on the display, **which you can advance only in the stopwatch**. Whether it is in

stopwatch or not, it will go on. Exactly after 30 seconds, the second alarm also stopped. It may not be exact, a delay of maybe half a second or 1 second is there – that is not a very important thing.

Let us wait for the third alarm to come, which will come in another 12 seconds. At 13 hours, 2 minutes and 00 seconds, you have to get the third alarm. Once again, it went on and for the next 30 seconds, it will sound. As I mentioned, this application can be used for epilepsy patients. If you do not administer the tablets, if they fail to take the tablets in time, say within half an hour or 45 minutes, they will promptly get fits. In order to avoid this, you can have this particular timer, which will keep warning say at morning 8 o'clock, then towards lunch time at 1 o'clock and then again at 7 o'clock dinner time.

They should not miss it for long. In fact, we can build in even more sophistication. In spite of your ringing in time, suppose they take much more time and suppose they have dozed off. What do we do? You can continue to sound the alarm intermittently after 5 minutes and so on till they press another button to acknowledge. If you want, you can add this feature in your assignment, which we are going to see next.

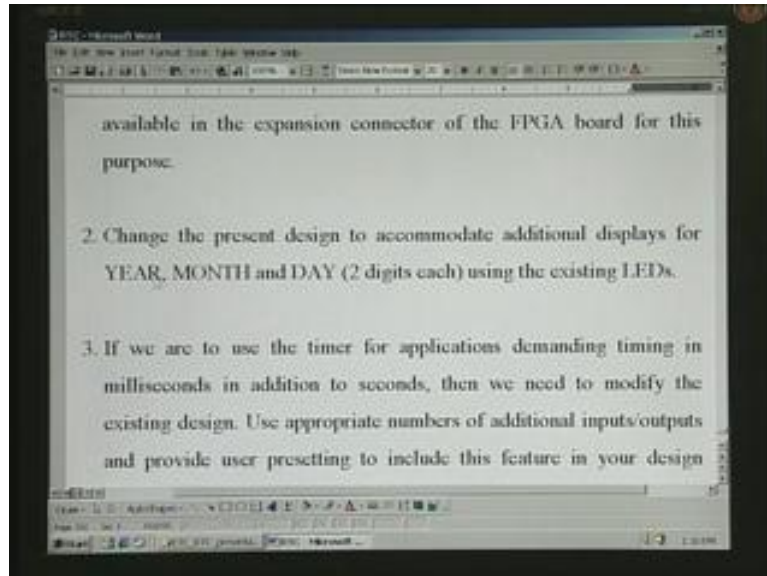
(Refer Slide Time: 42:29)



Let us discuss your assignments. The first assignment is we have used a 24-hour clock. What you have to do now is provide another switch and include a 12-hour clock as well. I will read it out. Modify the Verilog code to display the timing range **00 to** 11 59 59 (this is the maximum and after this, it has to roll back to 0s) besides the 24-hours range and other

features you have to retain as is. You may use another input and an output available in the expansion connector of the FPGA board for this purpose. I think this is clear to you.

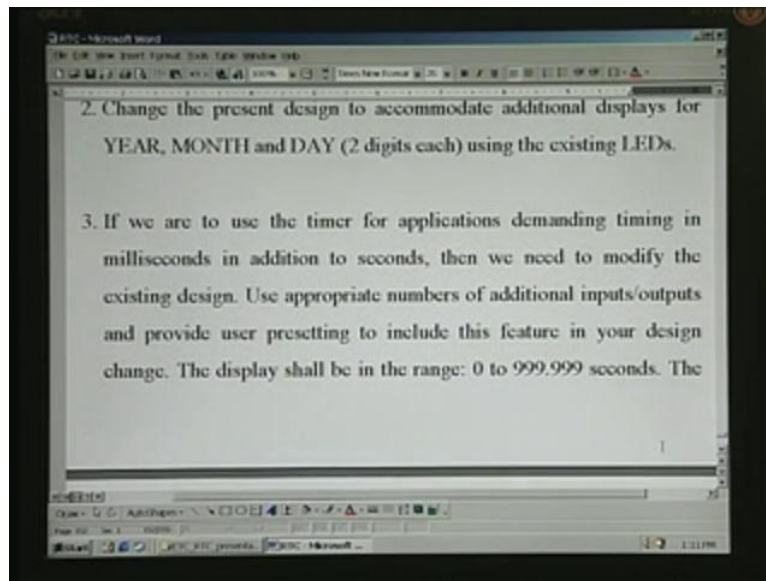
(Refer Slide Time: 43:16)



The second assignment is.... When you talk of real-time clock, you have to have year, month and day as well. Otherwise, it is not really a real-time clock. When you say year, it should at least be for the present millennium. You can have from 2000 to 2099. Make sure you have that and for that, you need only two digits – you do not worry about 2000, last 2 digits alone you need to take. The month of course is clear – it can go up to 12, 1 through 12. The day again depends upon whether it is a February or January and so on. You have to keep track of all the variations in the number of days in the month. You have to keep track of all this and then build the whole thing.

Basically, it is going to be different counters here. You do not have to worry about sounding the alarm with reference to any of this. Let the alarms be as they are, catering only to the hours, minutes and seconds. Without disturbing the hours, minutes and seconds, you should build this year month and day as well – 2 digits each. For this, you use the same display1 through display6. You need to use a different counter – remember that. That is why we say ‘using the existing LEDs’.

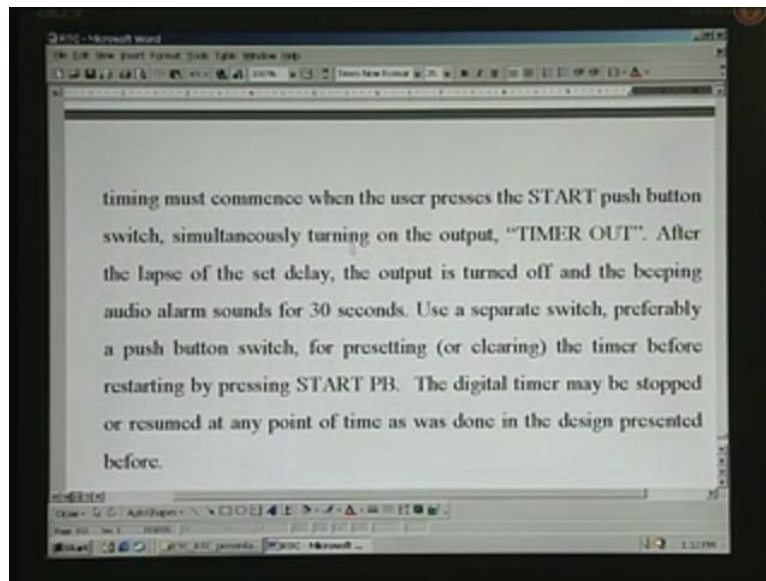
(Refer Slide Time: 44:46)



The third assignment is... I will read it out first. If we are to use the timer for applications demanding timing in milliseconds in addition to seconds... More often, we need for milliseconds. For example, you have a photographic exposure and you want to develop prints. Before that, you have to expose the film inside the dark room. You may need probably a millisecond resolution onwards, for which you need timing right from 0 to 999.999. After the dot, we have three digits, so this works out to be 1 millisecond, the very fag end on the right.

I will read it out. If we are to use the timer for applications demanding timing in milliseconds in addition to seconds, then we need to modify the existing design – that is obvious. Use appropriate numbers of additional inputs or outputs and provide user presetting to include this feature in your design change. You also need to have appropriate presetting. All these demand additional inputs. You can use push button switches preferably. The display shall be in the range 0 to 999.999 seconds.

(Refer Slide Time: 46:08)



The timing must commence when the user presses the START push button. This timer is **class different** from what we are used to earlier. In that case, it was an on-delay timer – after a set delay, the **output TIMER OUT** goes high. That is called on-delay timer whereas in this, what we want is interval delay timer. When you push a button, the time delay starts as well as **TIMER OUT** goes high. After the delay is over, the **TIMER OUT** again goes low and that is what you want.

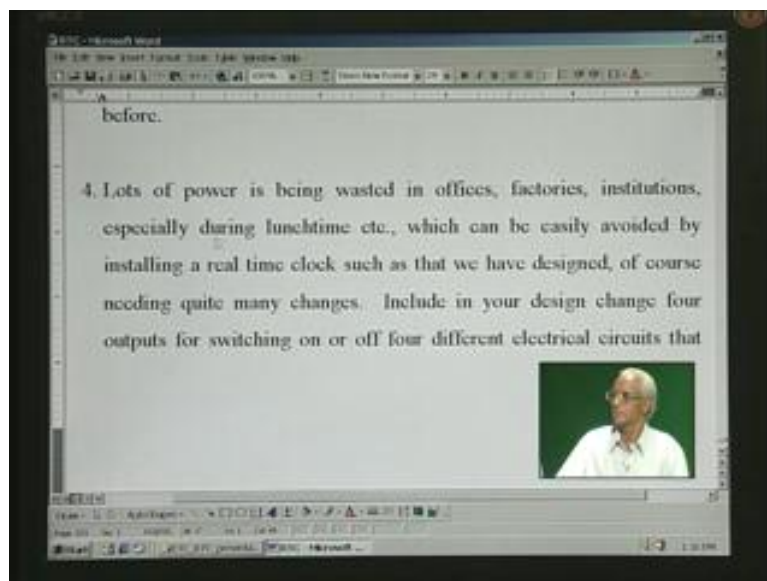
It is something like pushing the button, starting the time and after the set delay is over, for example, if you have set 999.999 seconds, after that time is lapsed, **TIMER OUT** goes low. Soon after you push the **START** button, it went high. That is what is called interval delay timer. This is what you are required to design now. On similar lines, we have done before for on-delay timer. The only thing you have to do is incorporate or change the logic aptly. After the lapse of the set delay, the output is turned off and the beeping audio alarm sounds for 30 seconds. This once again is the same thing.

Use a separate switch preferably a push button switch for presetting or clearing the timer before restarting by pressing the **START** push button. If you press the **START** push button, it should start. You need to either preset or clear depending upon what mode you want to use. If you use the down-counting mode, you would like to preset. For this, you need another input. Say for presetting, you can have another switch – a likely choice would be a push button switch.

You push this and whatever was set on the display, let us say 999.999, that will be transferred to the running counter at that point of time. But it will not start unless you press the START push button. Otherwise, if you want to use the up counter, you would like to clear it to start with. The preset value is inside the preset counters. Just like we had alarm counters separate for presetting them, here also you can have for up-counting or down-counting a preset counter separately – the running counter is different.

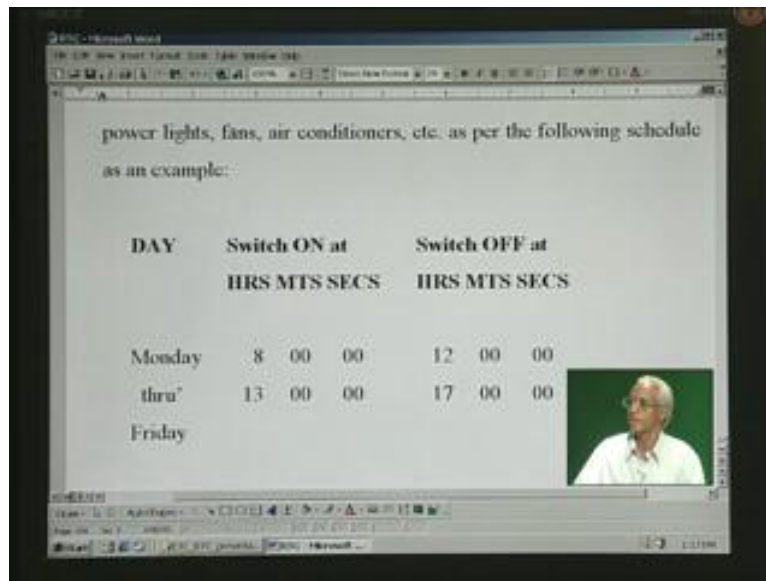
Whenever you push the button, if it is up counter, let us clear and if it is down counter, let us preset. Once you have preset, it will be cleared, all the display will be cleared. When you push the START push button now, it will start running right from 0, 1, 2, 3 and so on till the set value is got on the display. By that time, the TIMER OUT will go low. When you had started, it went high and it remains high for the duration of the delay that you have set. After that, TIMER OUT goes low. The digital timer may be stopped or resumed at any point of time as was done in the design presented before. These are the same features that we have seen earlier. Let us have the last assignment for you. I will quickly read this.

(Refer Slide Time: 49:33)



Lots of power is being wasted in offices, factories, institutions, especially during lunchtime, etc., which can be easily avoided by installing a real-time clock such as the one we have designed, of course needing quite many changes. We need to do quite a lot of changes in this.

(Refer Slide Time: 49:59)



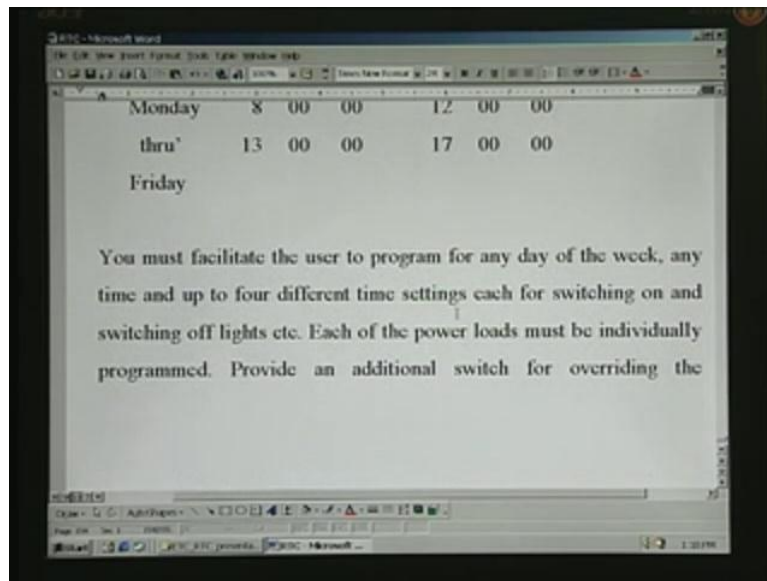
power lights, fans, air conditioners, etc. as per the following schedule as an example:

DAY	Switch ON at			Switch OFF at		
	HRS	MTS	SECS	HRS	MTS	SECS
Monday	8	00	00	12	00	00
thru'	13	00	00	17	00	00
Friday						

Include in your design change four outputs for switching on or off four different electrical circuits that power lights, fans, air conditioners etc., as per the following schedule as an example. For example, you want to set from Monday through Friday. If it is a factory or an office, the starting time will be 8 o'clock. Then, you have to switch on one of the circuits. I have mentioned four different circuits for lights, fans, air conditioners and some other load. Independently, you will have to be in a position to set all this.

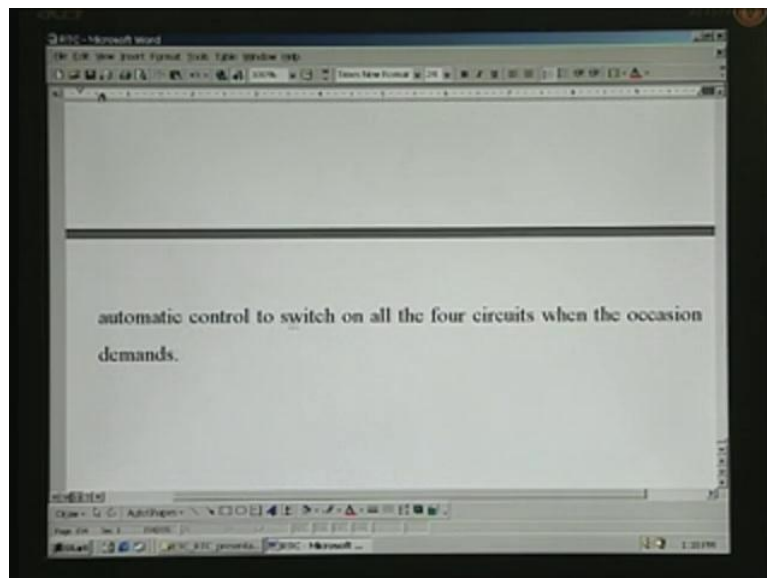
Once again, it is not merely Monday through Friday, it has to be right from Monday through Sunday – all you will have to cater to. As an example, what has been shown is only two pairs of timing here. It will be on at 8 o'clock, then off at lunchtime and after lunchtime is over at 13 hours, you have to switch on once again and switch it off towards the end of the current shift. This is not adequate, I want you to add two more. For example, you need teatime. Say at 10 o'clock, let us have a break for 15 minutes or 20 minutes. It should be programmable because you say on time is this, off time is this, so you do that accordingly. You are required to do in addition to this two more settings: teatime for morning at 10 o'clock and 14 hours **or something**, whatever you choose – towards evening.

(Refer Slide Time: 51:30)



You must facilitate the user to program for any day of the week, any time and up to four different time settings each for switching on and switching off lights, etc. This is what we have said. Each of the power loads must be individually programmed. It has to be individual – each **must be four different times**.

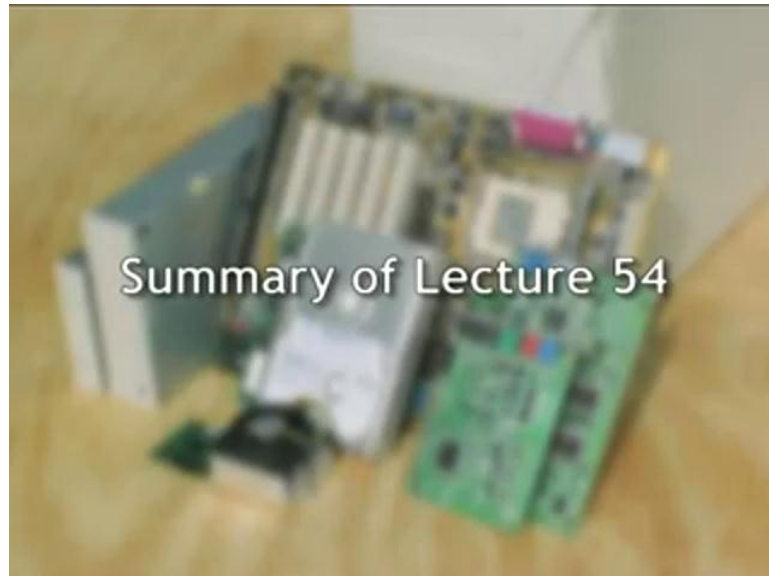
(Refer Slide Time: 51:51)



Provide an additional switch for overriding the automatic control to switch on all the four circuits when the occasion demands. You should also have a provision that will turn on. For example, it was a holiday and the timer would not have switched on. What do you do? There

must be an overriding control switch there in order to switch this. You can veto the automatic control that you have. Are you clear with this assignment? If so, thank you.

(Refer Slide Time: 52:16)



(Refer Slide Time: 52:50)

