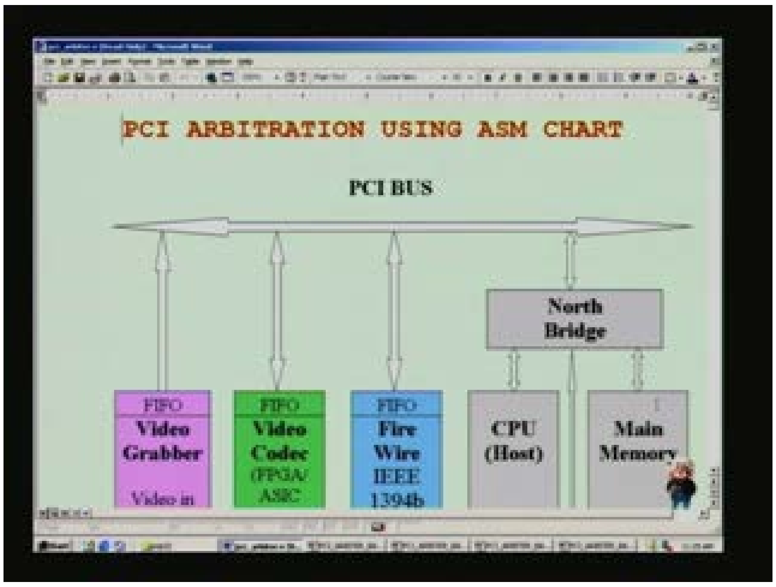
Digital VLSI System Design

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

Lecture No. # 36
PCI Arbiter Design using ASM Chart

We were looking into the PCI arbitration using ASM chart and basically, it is a design of arbitration and not pertaining to an application such as video compression. Although, it is for an illustration we mentioned about the video compression and sharing. This PCI bus here or 4 masters namely, video grabber which will get the raw image data - it may be PAL or NTSC sequence; or it can be SVGA format, any color motion picture can be processed; say at 30 frames per second or 25 frames per second.

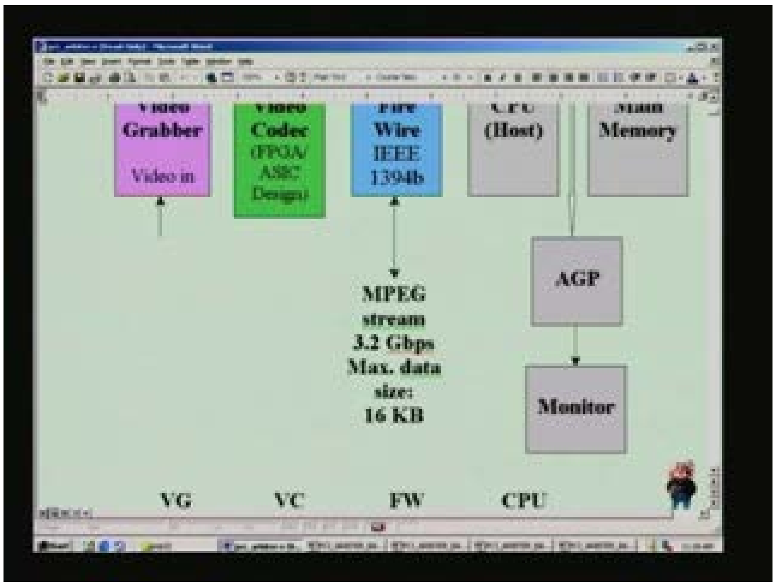
(Refer Slide Time: 02:32)



What we collect here using the PCI bus we communicate the raw data and to the codec whose function is to bring about the compressionals and we have actually 2 here. 1 is called encoder other is called decoder which brings about respectively the compression as well as the decompression and as an abbreviation it is mentioned as codec. This is the design that we will have to actually do using the Verilog and it can be an FPGA or ASIC platform and there will be a first in first out memory in all these matters. So that, whatever is processed here will be showed primarily and when it gets the chance in order to communicate to out to another master for example, this raw data I want to communicate here to the codec to start with, in order to bring about the compression. So, what we do is, we make a request to a bus arbitrator which is our present design from an independent masters.

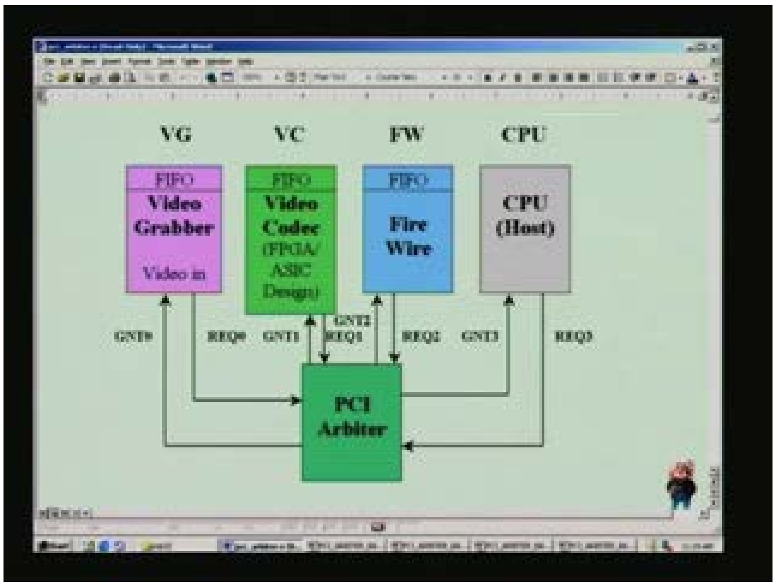
For example, this can be a request 0 and this can be request 1, then request 2 and then another which is the host. For example, a Pentium we have can be another master and we assign priority to these and the top we will divide this into 2 groups. One is video grabber and codec. We assign topmost to among the 2 groups and next group fire wire as well as the CPU and among this, the topmost priority goes to the video grabber and once again, among this second group higher priority is assigned for fire wire. And, fire wire is a serial bus which can be connected up to 64000 and odd serial buses here. That means to say, we can compress data here and then using the PCI we can send it over to the fire wire which will in turn serialize the data and send it over the channel.

(Refer Slide Time: 06:04)



Likewise, from any other computer we can get another motion picture concurrently and then process here and the reverse process can take place here namely, the serial to parallel conversion, then using the PC bus, once again communicate to the decoder port which decodes that means, brings about the de-compressional as such and passes over to the monitor which is the computer monitor that we have right here. Through an interface called advanced graphic port here and as far as fire wire is concerned, it’s through put is really very high and we can very easily use MPEG streams such as MPEG 2 which is the fastest of the MPEG group. And, we can also have JPEG or JPEG 2000 for a stream frame then MPEG 1 or MPEG 2 or MPEG 4 and so on and any other standards can also be incorporated here. what all we need to do is only a redesign the bus arbitrator as well as the core that we are developing for a particular application.

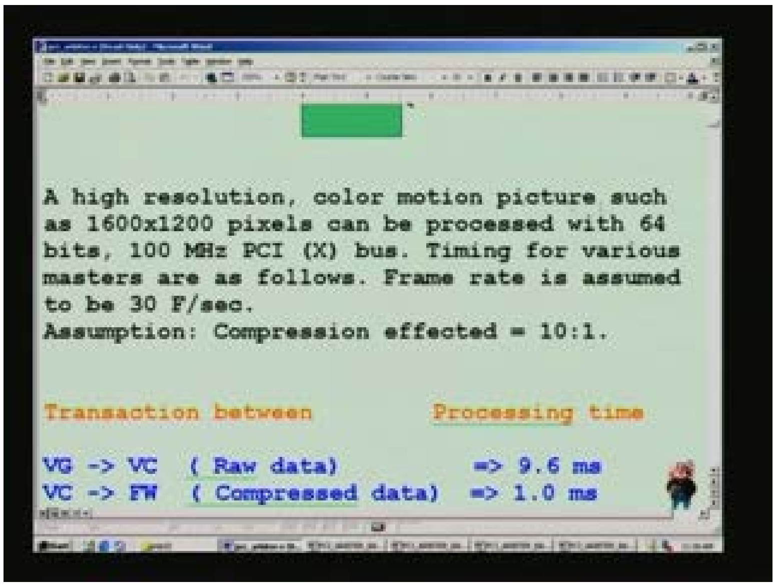
(Refer Slide Time: 07:17)



It’s a maximum data that it can handle at one-go is 16 kilo byte; that means to say, you can communicate to and fro from the fire wire up to this much size and this is not a handicap because, it will get a chance once again periodically as we will see right now and this CPU has, I mean, is connected to the PCI bus what is called north bridge and main memory is also connected to this; these are all the standard PC architecture.

So, in the case with AGP as well as the monitor and this is what we are going to design. This PCI arbiter has as I mentioned, 4 masters requesting the use of PCI bus and they are communicated by using the signals - request 0 through request 3 corresponding to the video grabber will abbreviate this as shown on the top. This PCI arbiter looks into the priority that we are going to see shortly and as per the priority it will allocate grant signals to either this video grabber or codec and so on using the signal grant - grant 0 through grant 3 is meant for this.

(Refer Slide Time: 08:04)

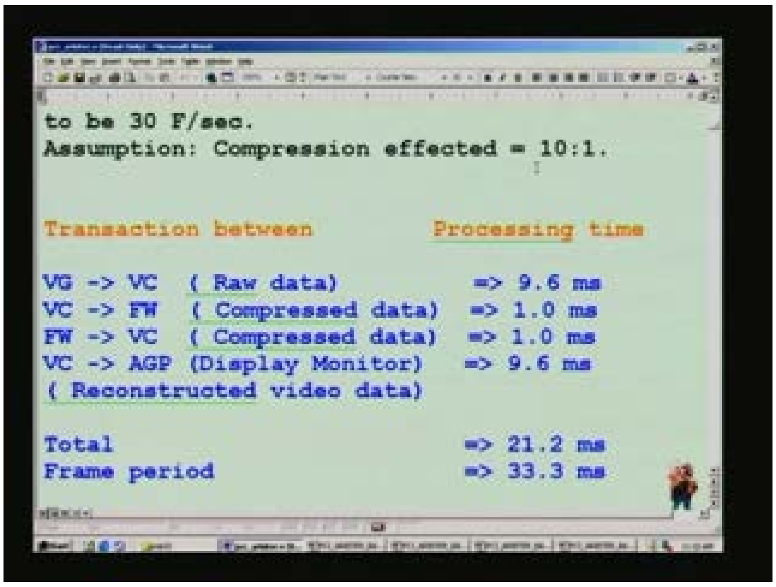


A high resolution, color motion picture such as 1600x1200 pixels can be processed with 64 bits, 100 MHz PCI (X) bus. Timing for various masters are as follows. Frame rate is assumed to be 30 F/sec.
Assumption: Compression effected = 10:1.

Transaction between		Processing time
VG -> VC	(Raw data)	=> 9.6 ms
VC -> FW	(Compressed data)	=> 1.0 ms

As far as the application is concerned you can have a high resolution color motion pictures such as 1600 by 1200 pixels and this pixel resolution is very high and it matches. I mean, it competes with the actual photographic prints and it is indistinguishable from prints and so even this sort of motion pictures can be processed with 64 bits and 100 megahertz PCI bus. Here, this 100 megahertz PCI bus is nomenclatures PCI x bus (Refer Slide Time: 09:00) and if you want much lower resolution, we can go either for PCI bus straight away a 64 bit or 32 bit or again speed either 33 megahertz or 6 megahertz and the frame rate it can process at this speed that you are going to discuss is 30 here.

(Refer Slide Time: 09:44)



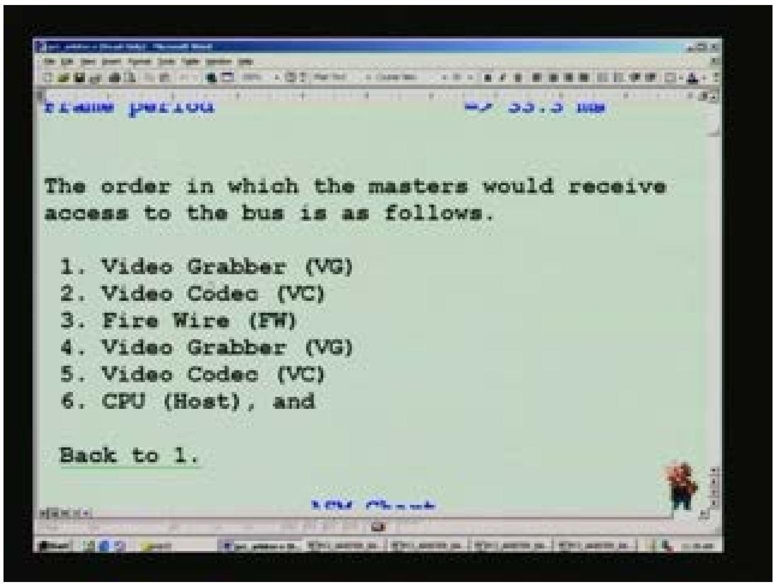
```
to be 30 F/sec.
Assumption: Compression effected = 10:1.

Transaction between               Processing time

VG -> VC   ( Raw data)              => 9.6 ms
VC -> FW   ( Compressed data)       => 1.0 ms
FW -> VC   ( Compressed data)       => 1.0 ms
VC -> AGP  (Display Monitor)        => 9.6 ms
( Reconstructed video data)

Total                               => 21.2 ms
Frame period                        => 33.3 ms
```

This is normally for the NTSC format, US format and parallel etcetera, you need to comply only with 25 frames per second and assuming we have a compression of 10 is to 1. That is to say, if you have 10 gigabytes of storage for a particular motion picture so what all we need is only 1 GB after the compression is effected in the codec that we have seen here.

The compression is brought about here and this is a very conservative assumption because, from my experience I have been seen even for I-frame processing in MPEG 2, it gives 15 to 20 compression factor. Now, let us see different timings here for example, from video grabber to video codec which is a raw data communication would take for this particular resolution 1600 by 1200 color motion picture around a 10 millisecond and from video codec after the compression is affected it will send it to the fire wire for onward transmission or a serial bus and this also requires the PCI bus and that would take only a 1 millisecond because, we have said 10 is to 2 is the compression factor. That means, it takes just one-tenth of this raw data.

So, next step is we send it out on the transmission channel serial channel and concurrently we may receive another motion picture from some other computer over the fire wire and that can be once again routed to the codec. In this case, it goes to the decoder and these both are compressed data as such and it again, takes only 1 millisecond because it is compressed fashion and once this is processed here it brings about decompression in the codec. Using decoder inside that and then what we need to do is, nearly send it to the display monitor. Why are this AGP here and this would again be same size or the raw data because, what we have done here is reconstructed the video data. That is how we can communicate on a network, PC network and up to a resolution of 16 bus 1200. And, this is rather a theoretical research stage; so will add to see the real implication when we really build the system and summing up all these you have only 21.2 millisecond whereas 1 frame time is 33.3 millisecond taking 30 frame per second and we have lots of room left. So, another 12 millisecond are so far the host. To do certain job such as configuration form JPEG to MPEG 1 or MPEG 2 and so on, or intermediate processing, also needs to be done which will use host for purpose and naturally, have lots of time available for the host in order to complete its task.

(Refer Slide Time: 12:40)



frame period => 33.3 ms

The order in which the masters would receive access to the bus is as follows.

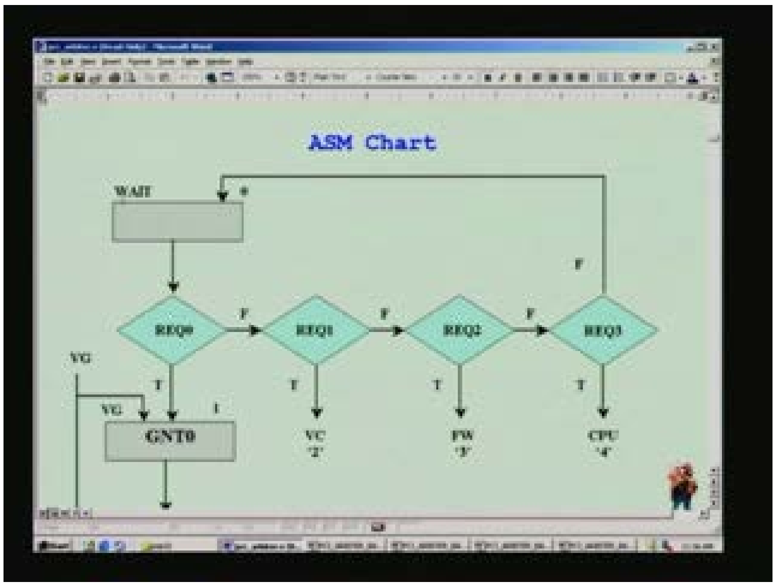
1. Video Grabber (VG)
2. Video Codec (VC)
3. Fire Wire (FW)
4. Video Grabber (VG)
5. Video Codec (VC)
6. CPU (Host), and

Back to 1.

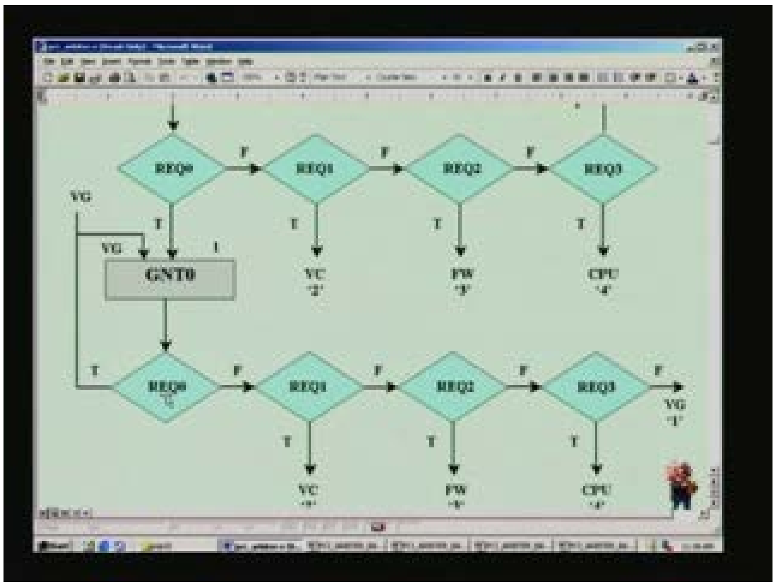
So, this shows that, this scheme is a viable scheme; and let us see the, in order of I mean, access to the PCI bus. This is the priority that we ascribe to different masters; for example, we said the video grabber and codec are in the first group. So, we will give in turn topmost priorities for the video grabber fallowed by codec and then we go to the second group wherein, give only one of the masters namely, the fire wire which is higher priority than the host. Once we have given a chance to the second group will come back to the first group and give chance once again for both video grabber and codec because, we need to process the raw data which cannot wait and we need to process it rapidly and so is the case with the compression here and once we give this and naturally, the last chance will be for the host.

As we mentioned before, host will not take much time for processing because, what all it has to do is drive the menu to start with and then configure from one state to another once in a way. What is topmost important is this video grabber followed by the codec and that is the reason why we gave the priority like this (Refer Slide Time: 14:00); and this is what we are going to implement in the bus granting scheme. Once we have given to the second group we come back to 1 and continue forever.

(Refer Slide Time: 14:11)



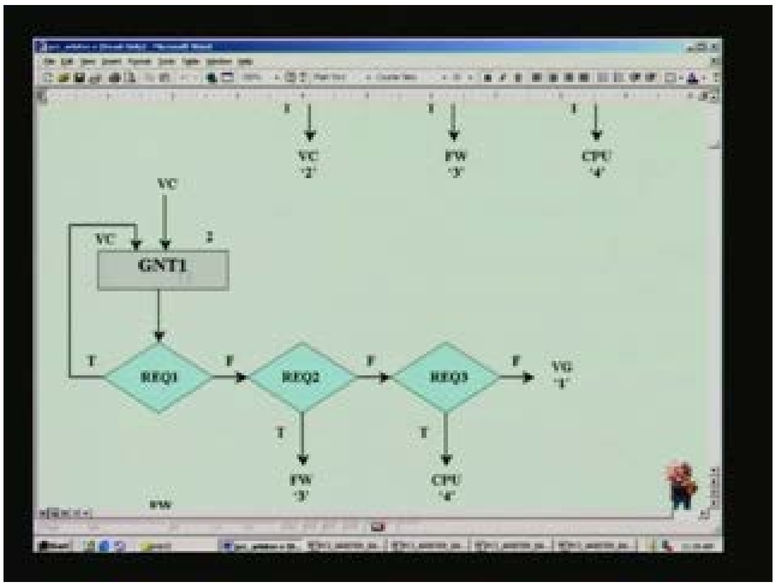
(Refer Slide Time: 14:53)



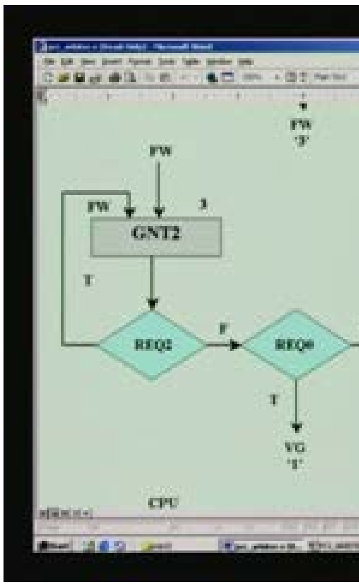
So, we will see how to make an ASM chart for this application such as bus grant. This is rectangular - is a straight indicator and actual value is put here. We are going to use the decimal value because, it will be easier for us to code it in Verilog and the descriptive state is on this. And this implies, we are going to wait for any of the request to manifest. So, as we mentioned before there are 4 masters and each of which is interrogated here whether a particular request is, has been got by an arbitrator (Refer Slide Time: 14:32). Well, we are questioning at this point of time and topmost priority is for the video grabber and therefore we question that first.

Suppose, the video grabber has not settled its request, so we will go on to the a next what is called a codec and followed by fire wire as well as CPU and if none of these requests are pending, so we go back to square 1 that is a wait state and continue wait till someone makes the request. Once request is got let us say, the topmost priority is the video grabber because that is the first thing that will happen in this compression system because, without grabbing the image there can be no further processing. So, naturally this video grabber is the one which is going to make the first request and it would take some time for it to process. I mean, dumping it’s data collected over the PCI bus on to the video codec; so that would require some time. So, it will the request 0 will be asserted for quite some time and till it is finished, the usage of the PCI bus and so long as this request 0 is asserted, grant 0 must be asserted and that is the loop here and next is this is also the entry point for VG state and if this is not asserted or it has completed its request usage of the bus. So, naturally the next contending master will get the attention. For example, we will see request 1, or request 2, or request 3, in that order; this is once again the same priority as we have seen before here.

(Refer Slide Time: 17:06)

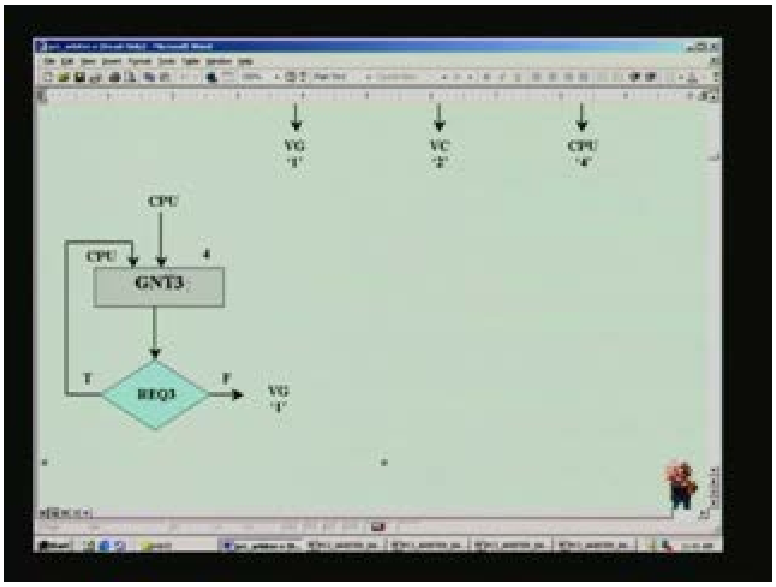


(Refer Slide Time: 17:52)



This state to which it goes after the request, is processed; it is asserted. It will be indicated here; so that is 2 corresponds to 2 here; so that means, this is connected here and once this is processed we go on to process the codec grant and this is request 1. I suppose yes, once the request 1 is asserted we will grant the signal 1 and once again, we stay here till this required v c has processed its codec; has processed all its transactions for current cycle. I mean, cycle means not the clock cycle; it may be several clock cycles and once it has finished its job, it will de-asset request 1. Therefore, control will go on to the fire wire and CPU and in case they have a request pending it will go to that respective routine. If none of this are present, it will naturally go to the video grabber because, all the 4 have been covered here and this is for fire wire grant. Once fire wire's request has come it will grant tech corresponding signal here - GNT 2 here. Once again, so long as its request is asserted it will keep revaluing around this asserting grant 2 all the time and once it has finished its job, it will naturally go to the first group because, this is the after fire wire we cannot go to the host because only one is allowed in the second group.

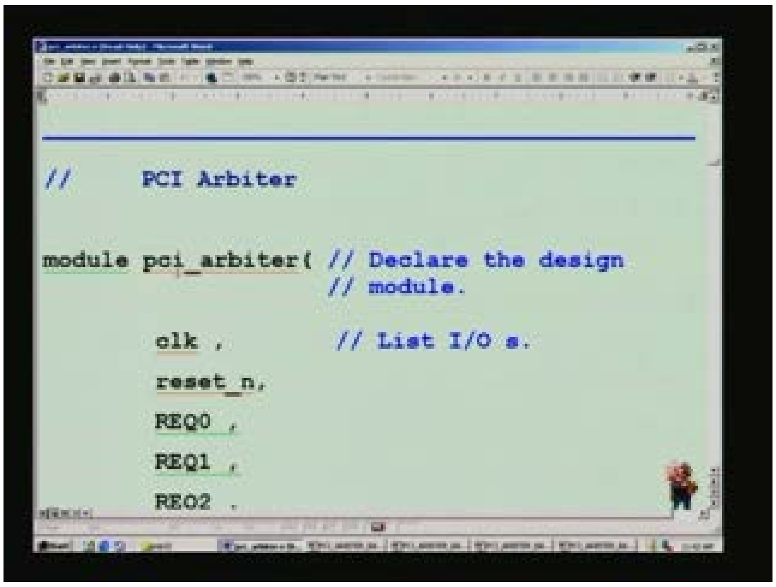
(Refer Slide Time: 18:54)



So, therefore the top next, automatically it goes to video grabber because that is the priority on which we had designed our systems and if this is not present naturally, it will go to the codec and thereafter, go to CPU. In case this is also not asserted and once again if none of this is present, it will go to VG and do the checking of request for all the 4 masters.

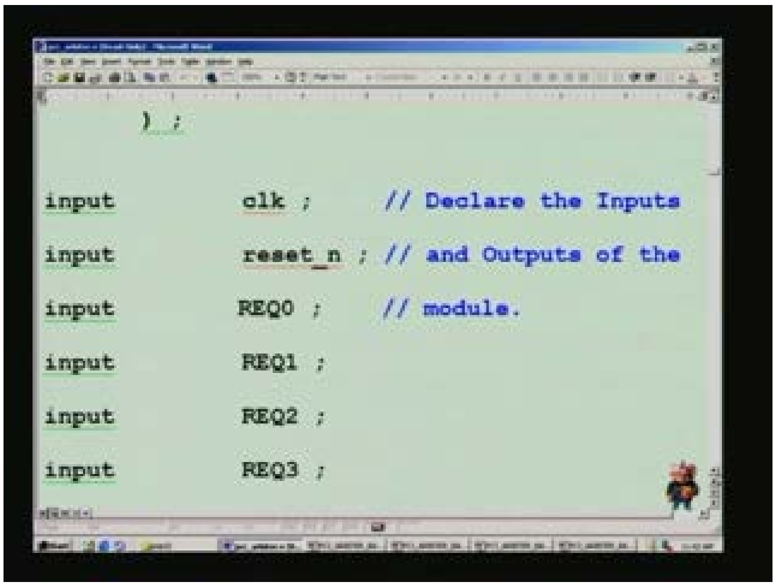
Similarly, finally for this CPU, this is the host here when host let us say, it has come from this path. It may come from any other branch also and suppose it has come here and grant 3 will be asserted here and it will continue to be asserted. So long as request 3 is asserted and once it has finished its job, it will go to the video grabber because this the second group second 1 and naturally, it has go to the first group and once again the video grabber being the topmost priority control passes on to this and it is a simple ASM chart and for which we can see the code now.

(Refer Slide Time: 19:34)



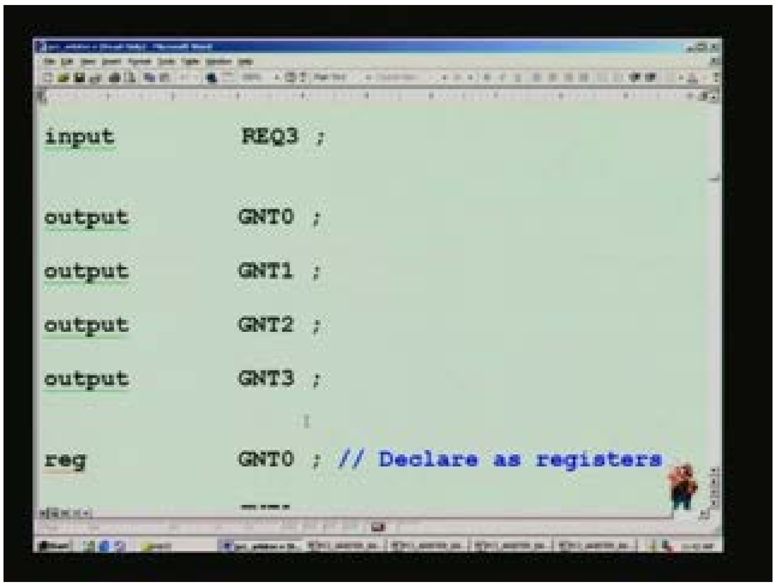
```
//     PCI Arbiter

module pci_arbiter( // Declare the design
                    // module.

         clk ,        // List I/O s.
         reset_n,
         REQ0 ,
         REQ1 ,
         REQ2 ,
```

(Refer Slide Time: 20:10)



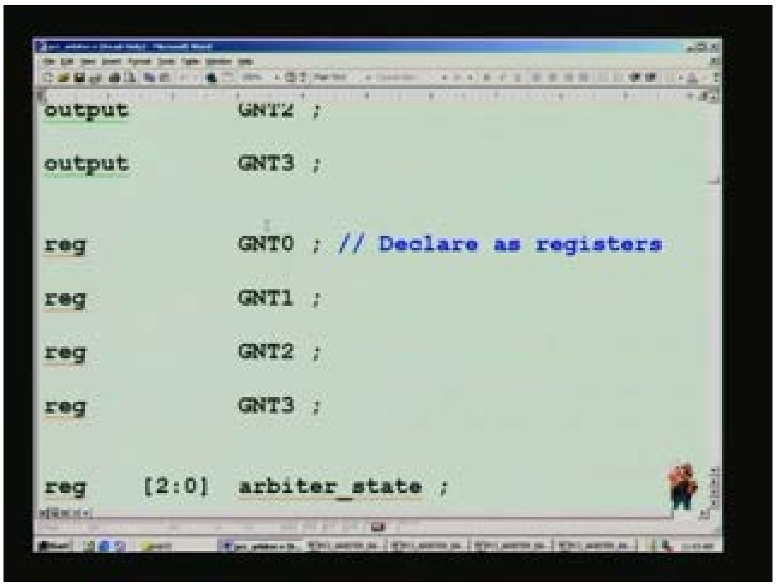
```
         ) ;

input          clk ;      // Declare the Inputs
input          reset_n ; // and Outputs of the
input          REQ0 ;     // module.
input          REQ1 ;
input          REQ2 ;
input          REQ3 ;
```

(Refer Slide Time: 20:16)



```
input        REQ3 ;

output       GNT0 ;

output       GNT1 ;

output       GNT2 ;

output       GNT3 ;

reg          GNT0 ; // Declare as registers
```

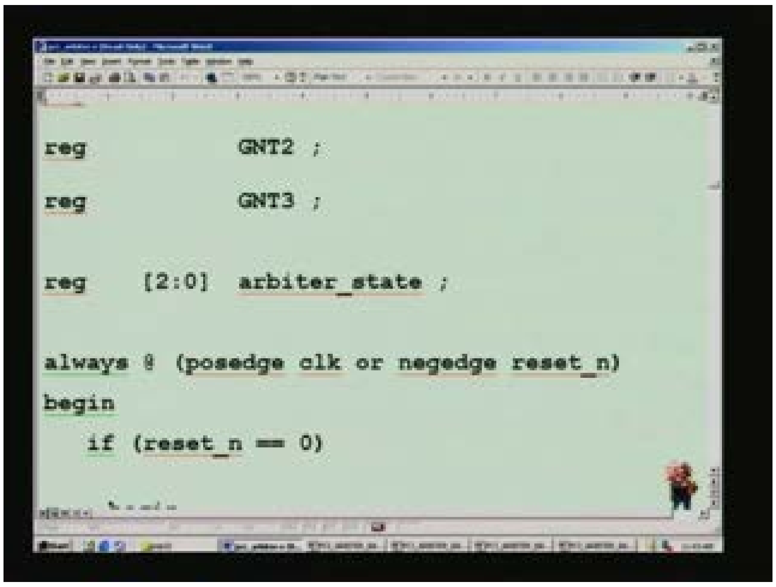
(Refer Slide Time: 20:23)



```
output       GNT2 ;

output       GNT3 ;

reg          GNT0 ; // Declare as registers

reg          GNT1 ;

reg          GNT2 ;

reg          GNT3 ;

reg    [2:0] arbiter_state ;
```

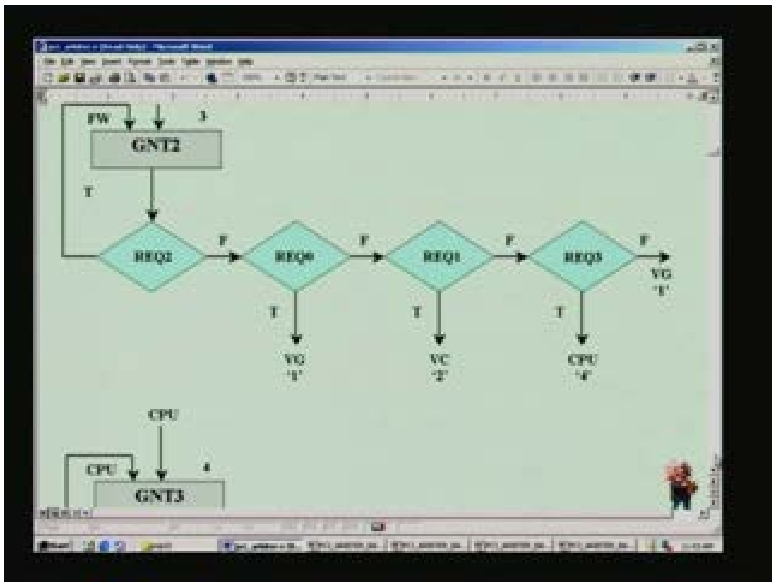
So, we give a line comment here and indicating this PCI arbiter is a design and we give the same name here for module and we declare the module here and collect all this IOs here and separator by comas. And, we have clock reset; so active low and 4 requests here - 0 through 3 and corresponding grant signal which will be output and this is what we have listed here. These are all the inputs here and these are all the outputs here and in addition to this, this output needs to be declared as register because, we will see that later on that this will be covered in always posedge clock block has.

(Refer Slide Time: 20:34)



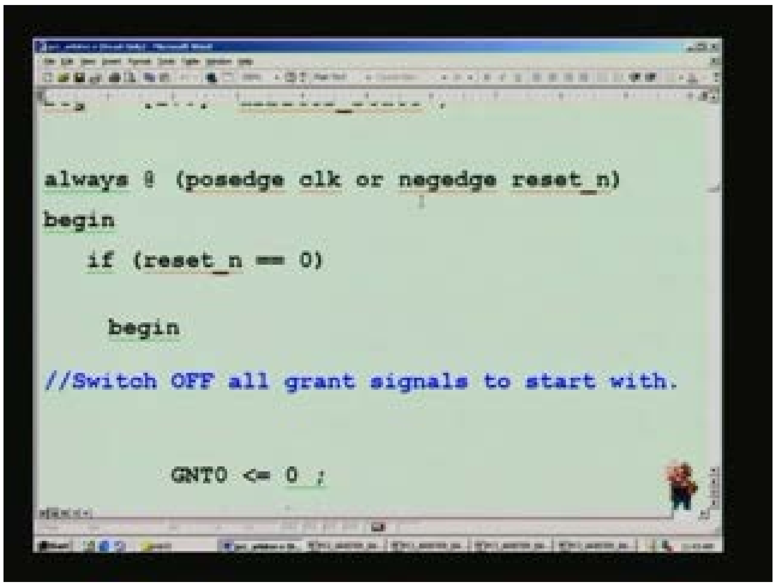
```
reg          GNT2 ;

reg          GNT3 ;

reg    [2:0] arbiter_state ;

always @ (posedge clk or negedge reset_n)
begin
   if (reset_n == 0)
```

(Refer Slide Time: 20:49)



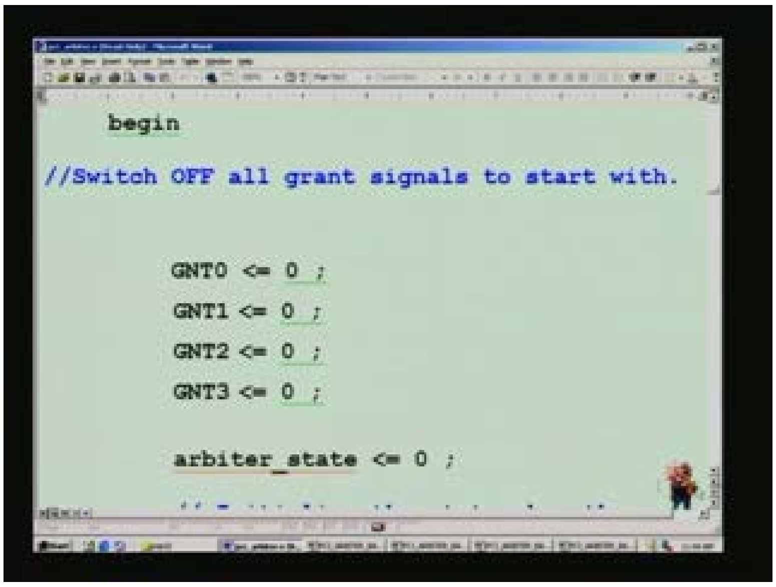
In addition to this, we have also the state indicated here and this arbiter state will call and this is what we have seen here. Earlier in the rectangles we have put the straight say for example 3 here or 4 here, this is what is implied there and will be using as we as used earlier a case statement in order to process this rectangular boxes in a sum.

(Refer Slide Time: 21:13)



```
always @ (posedge clk or negedge reset_n)
begin
   if (reset_n == 0)

     begin
//Switch OFF all grant signals to start with.


         GNT0 <= 0 ;
```

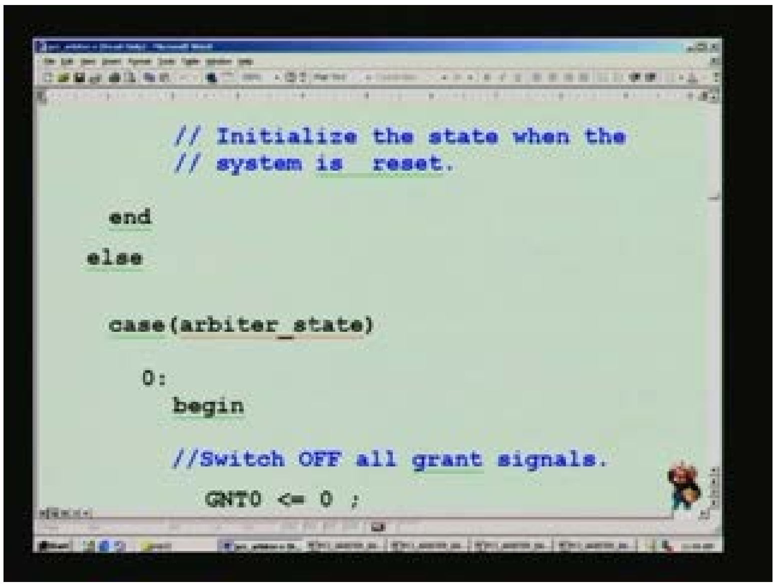
(Refer Slide Time: 21:28)



```
     begin
//Switch OFF all grant signals to start with.


         GNT0 <= 0 ;
         GNT1 <= 0 ;
         GNT2 <= 0 ;
         GNT3 <= 0 ;

         arbiter_state <= 0 ;
```

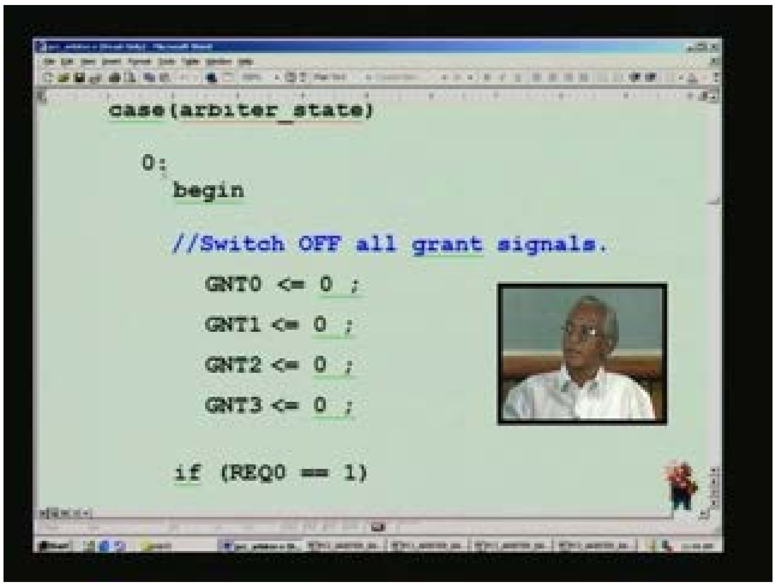
So to start with this is the first always block and it is recommended at posedge of the clock and we have once again as we show all negative edge reset. If reset is present this may be a system reset or a power on reset and if it is present, so what we need to do is we will switch off all grant signals to start with. This is, mind you, this a wait state 0 state and averring we do not had turn on any of the grants here and arbiter state also needs to be in the same state - wait state namely, 0 state.

(Refer Slide Time: 21:44)



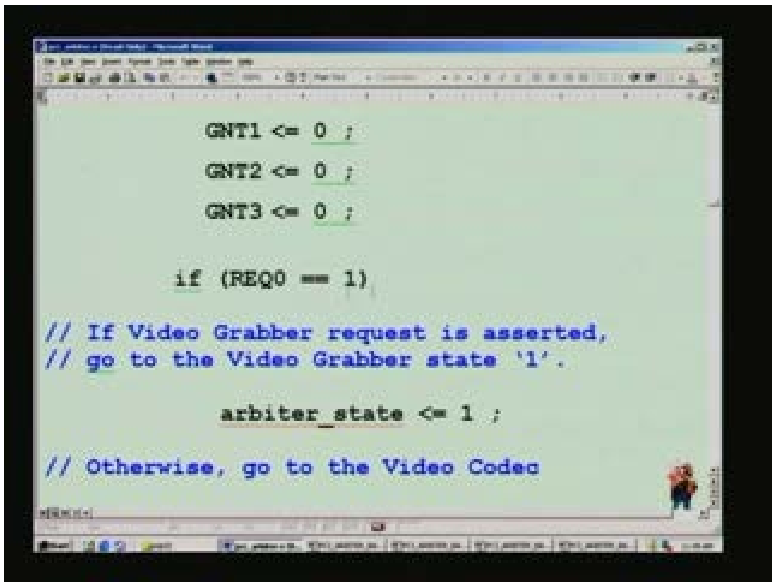
```
        // Initialize the state when the
        // system is  reset.

    end
  else

    case(arbiter_state)

      0:
        begin

        //Switch OFF all grant signals.
          GNT0 <= 0 ;
```

(Refer Slide Time: 22:10)



```
    case(arbiter_state)

      0:
        begin

        //Switch OFF all grant signals.
          GNT0 <= 0 ;
          GNT1 <= 0 ;
          GNT2 <= 0 ;
          GNT3 <= 0 ;

        if (REQ0 == 1)
```

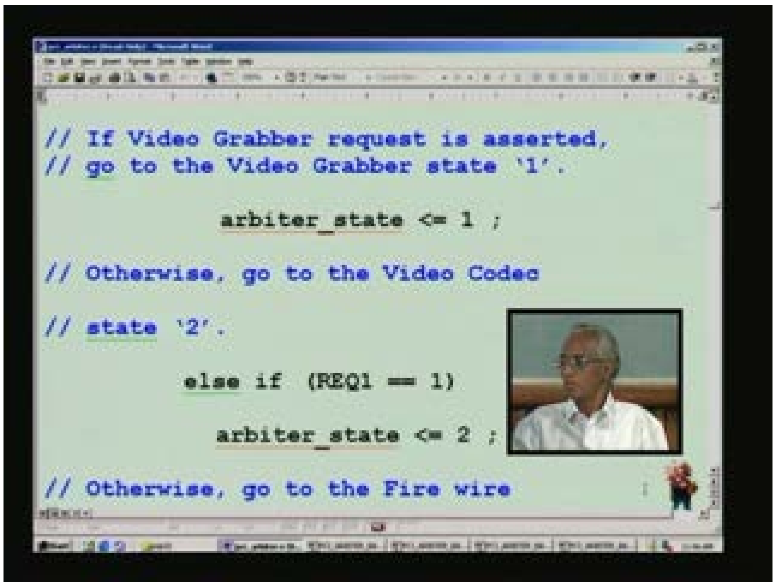
(Refer Slide Time: 22:26)



```
            GNT1 <= 0 ;
            GNT2 <= 0 ;
            GNT3 <= 0 ;

        if (REQ0 == 1)
// If Video Grabber request is asserted,
// go to the Video Grabber state '1'.

            arbiter_state <= 1 ;

// Otherwise, go to the Video Codec
```

We started with an ‘if’; so we have an ‘else’ here and a group of statements were encountered here. So, we need a ‘begin’, ‘end’ here and then we say ‘else’ and this is the case I have been referring to and this case depends upon the arbiter state and this will be 0 1 2 3 4 corresponding to the 5 rectangles that we had and this is the 0 state and these are all plain decimal as we have seen in earlier example also and in this state, it is nothing but a wait state. So, we do not have to do anything except turn off all the grant signals and now, we will examine corresponding to the very first interrogation block that we encountered in the wait state for request that was a diamond there on the sum chart and that translates here as an if statement here.

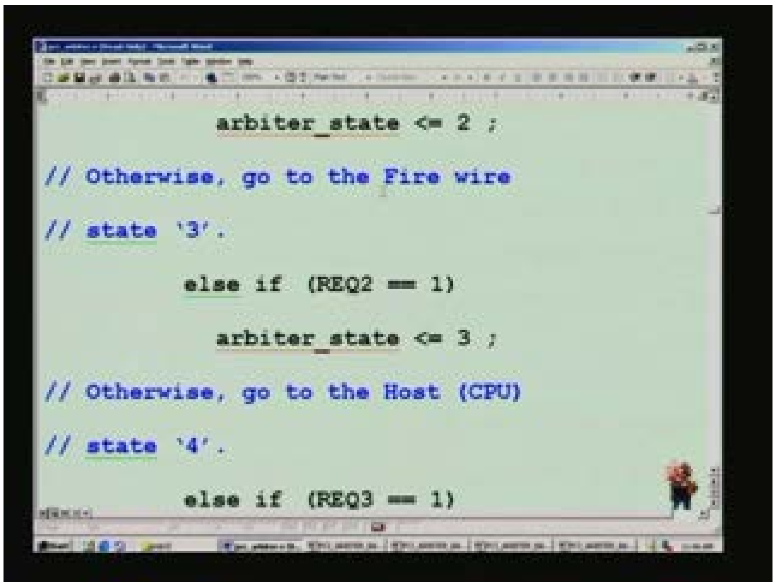
(Refer Slide Time: 22:57)



```
// If Video Grabber request is asserted,
// go to the Video Grabber state '1'.
              arbiter_state <= 1 ;
// Otherwise, go to the Video Codec
// state '2'.
            else if  (REQ1 == 1)
              arbiter_state <= 2 ;
// Otherwise, go to the Fire wire
```

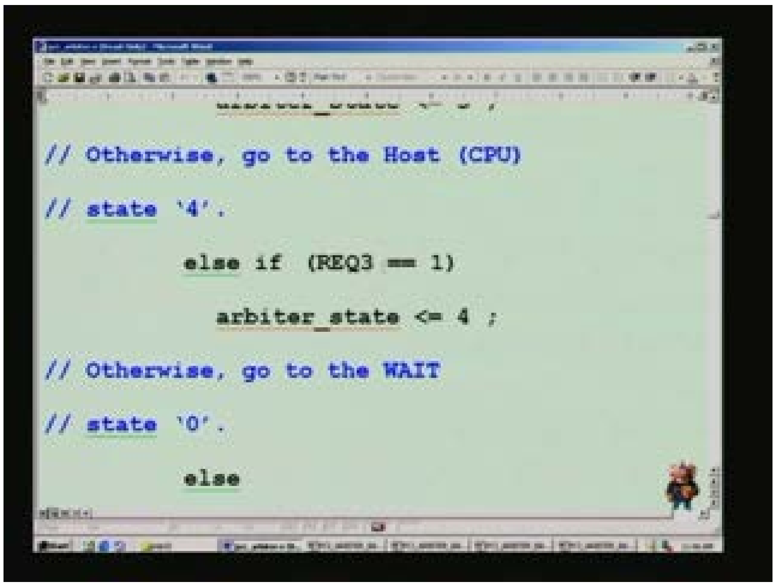
If request 0 is encountered and this is corresponding to the video grabber and what we need to do is, we had to take it to the state 1. We are presently in state 0; we will take it to state 1 because, in state 1 only video grabber is granted. It is a corresponding grant 1 signal and otherwise, go to the next waiting master that is video codec which corresponds to its state 2 and that is what we do here. else if statement is what you are already familiar; that we use and if request 1 that is corresponding to the video codec is asserted then take it to the arbiter state 2.

(Refer Slide Time: 23:27)



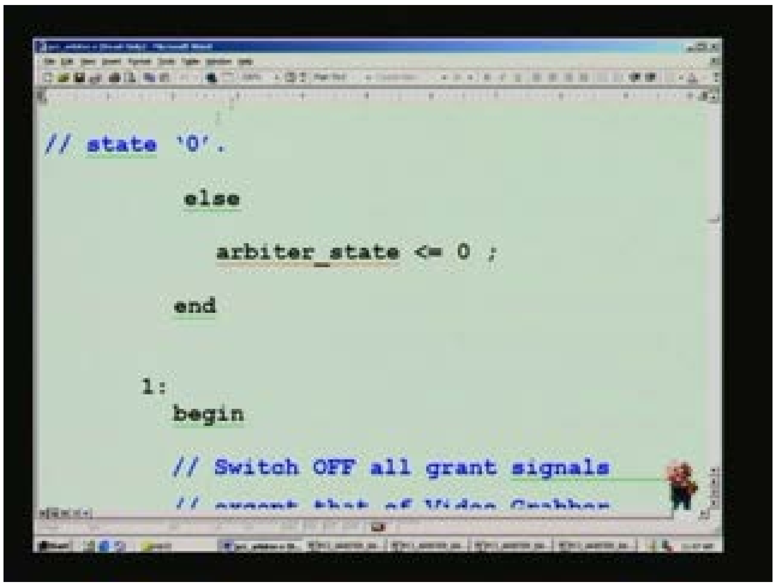
```
              arbiter_state <= 2 ;
// Otherwise, go to the Fire wire
// state '3'.
            else if  (REQ2 == 1)
              arbiter_state <= 3 ;
// Otherwise, go to the Host (CPU)
// state '4'.
            else if  (REQ3 == 1)
```

(Refer Slide Time: 23:54)



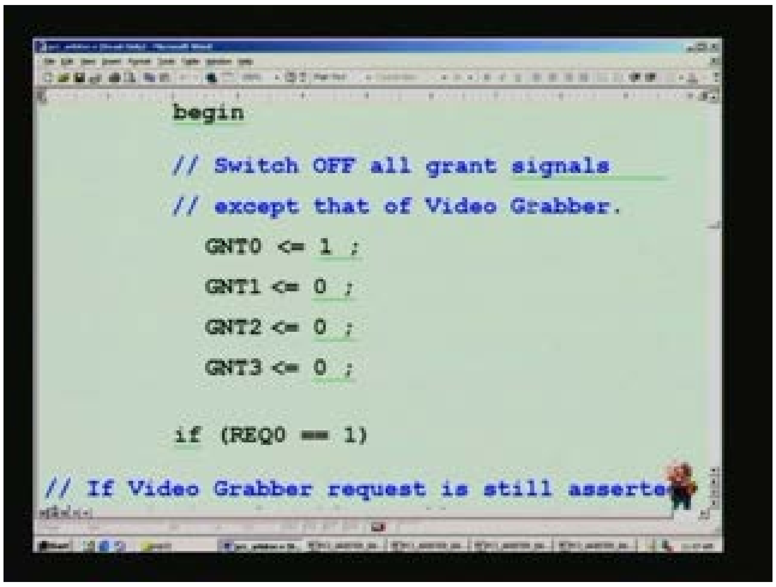
```
// Otherwise, go to the Host (CPU)

// state '4'.

          else if  (REQ3 == 1)

             arbiter_state <= 4 ;

// Otherwise, go to the WAIT

// state '0'.

          else
```

(Refer Slide Time: 24:20)



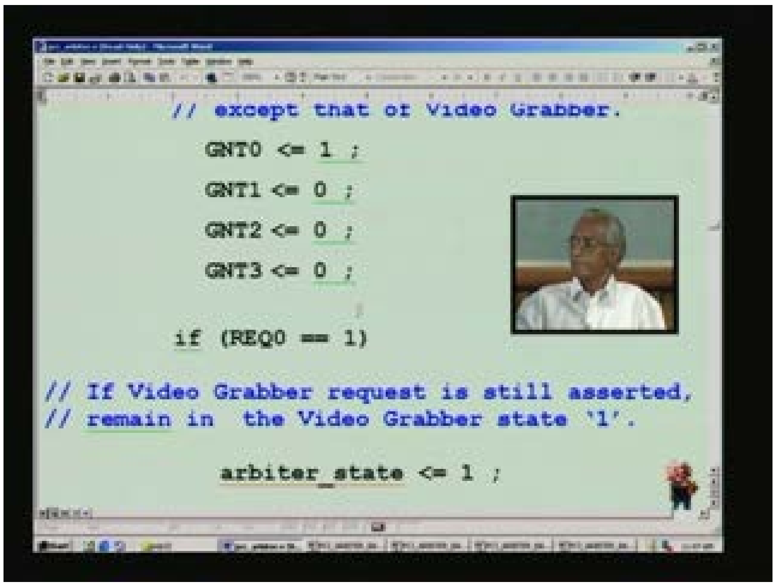
```
// state '0'.

          else

             arbiter_state <= 0 ;

         end


      1:
         begin

         // Switch OFF all grant signals
```

(Refer Slide Time: 24:33)



```
begin
// Switch OFF all grant signals
// except that of Video Grabber.
   GNT0 <= 1 ;
   GNT1 <= 0 ;
   GNT2 <= 0 ;
   GNT3 <= 0 ;

if (REQ0 == 1)
// If Video Grabber request is still asserte
```

(Refer Slide Time: 24:47)

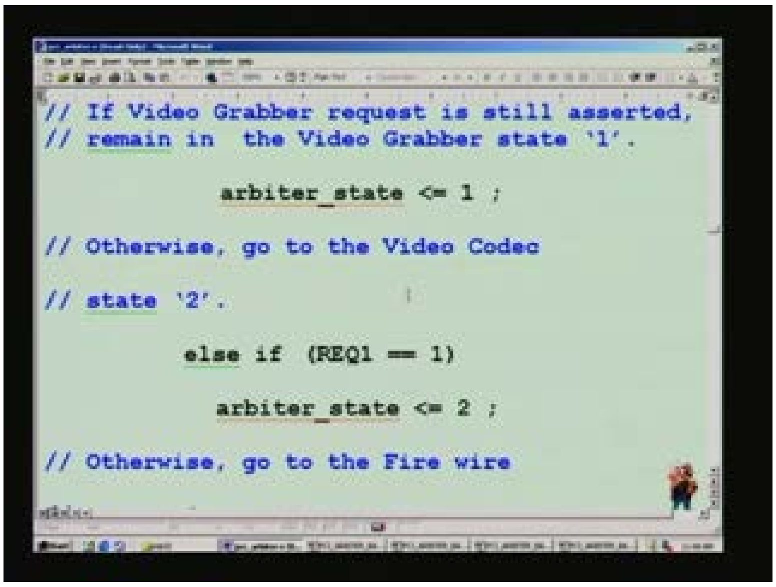


```
// except that of Video Grabber.
   GNT0 <= 1 ;
   GNT1 <= 0 ;
   GNT2 <= 0 ;
   GNT3 <= 0 ;

if (REQ0 == 1)
// If Video Grabber request is still asserted,
// remain in  the Video Grabber state '1'.

     arbiter_state <= 1 ;
```

If this is also not asserted, so we will naturally go for the next master which is fire wire and once again, we use an 'else if' state and this time we use the corresponding request 2 for the fire to be asserted and if it is asserted we take the arbiter state 2, state 3 which happens to be the state for the fire wire processing.

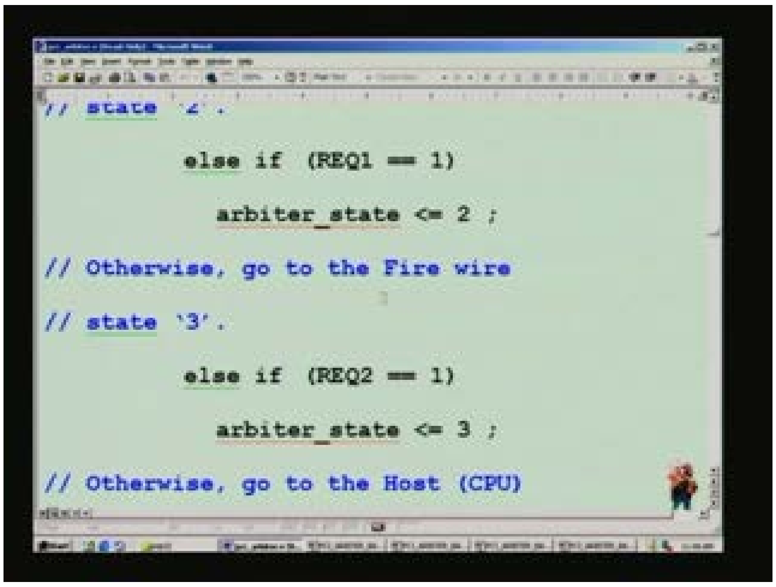
So, in the case for host processing whose state is 4 and its corresponding request is 3 here and that is what is here, is plain exactly same what we have seen before and if this is not encountered so we have it covered I think, all the states and what we need to do is go back to wait state 0. If none of these are encountered and corresponding state is 0 here, that is what is the arbiter state is for and there must be an end because we started with ‘begin’. We are processing only for case 0 and similarly, we had process for case 1 right up to case 4. So, for case 1 we need to switch off all other grand signals and switch on only 1 signal that is, grant 0 because, this happens to be the for a video grabber. This one important thing we should not forget to switch off all the unwanted grants and once again we look into the request 0 which corresponds to the video grabber state and whether, if the request is still pending which indicates that it has not yet finished the usage of the PCI bus and it should continue to being state 1.

(Refer Slide Time: 25:23)

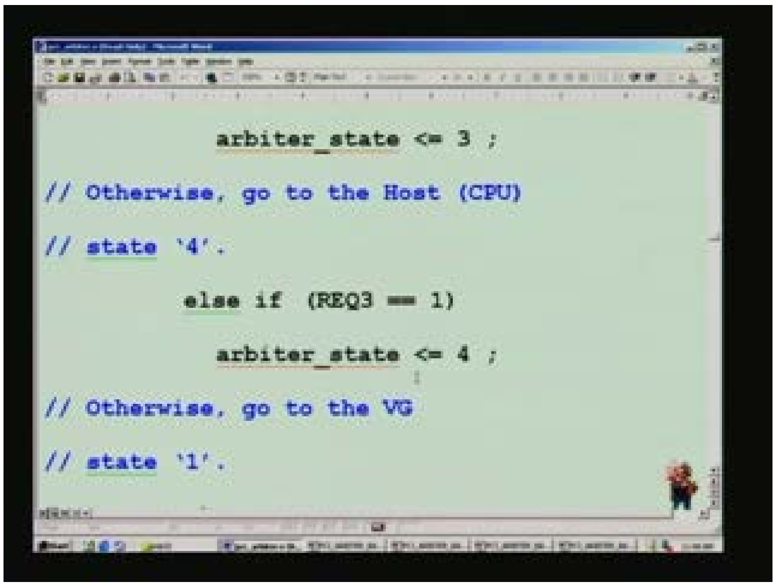


```
// If Video Grabber request is still asserted,
// remain in  the Video Grabber state '1'.

              arbiter_state <= 1 ;

// Otherwise, go to the Video Codec

// state '2'.

           else if  (REQ1 == 1)

              arbiter_state <= 2 ;

// Otherwise, go to the Fire wire
```

(Refer Slide Time: 25:46)


```
// state '2'.
          else if (REQ1 == 1)
            arbiter_state <= 2 ;
// Otherwise, go to the Fire wire
// state '3'.
          else if (REQ2 == 1)
            arbiter_state <= 3 ;
// Otherwise, go to the Host (CPU)
```

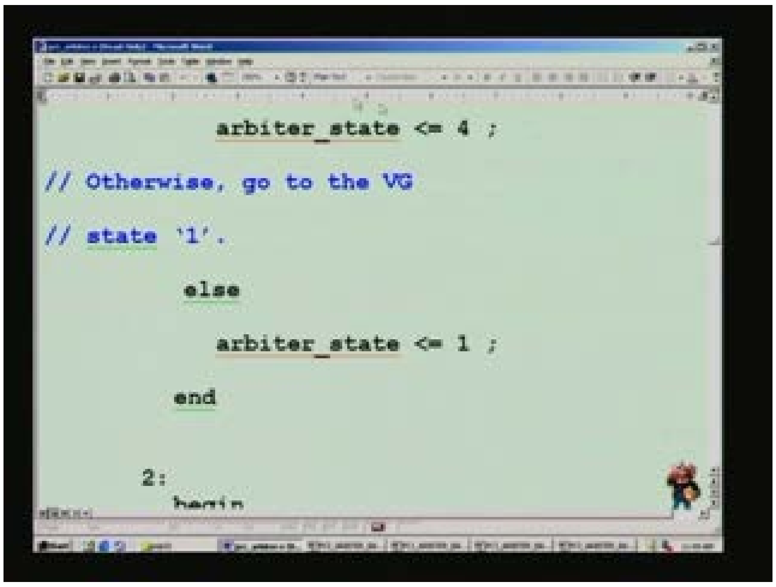
(Refer Slide Time: 25:57)


```
            arbiter_state <= 3 ;
// Otherwise, go to the Host (CPU)
// state '4'.
          else if (REQ3 == 1)
            arbiter_state <= 4 ;
// Otherwise, go to the VG
// state '1'.
```

In state 1 of course, we will be in same condition here where in grant 0 is continued to be asserted here and once it has finished its job, we need to check for request 1. This was request to 0 corresponding to the video grabber; corresponding to the video codec it is request 1 and corresponding state is 2. So, if this is asserted, go to the state 2; if this is also not asserted, so check for the fire wire status, its corresponding request is 2 and take it to the respective state 3. And so is the case for the host, which case it's state is 4 and we have seen that the same VG VC are given the priority first and then go out to the second group and in that give only 1 priority. We have just followed the ASM chart and

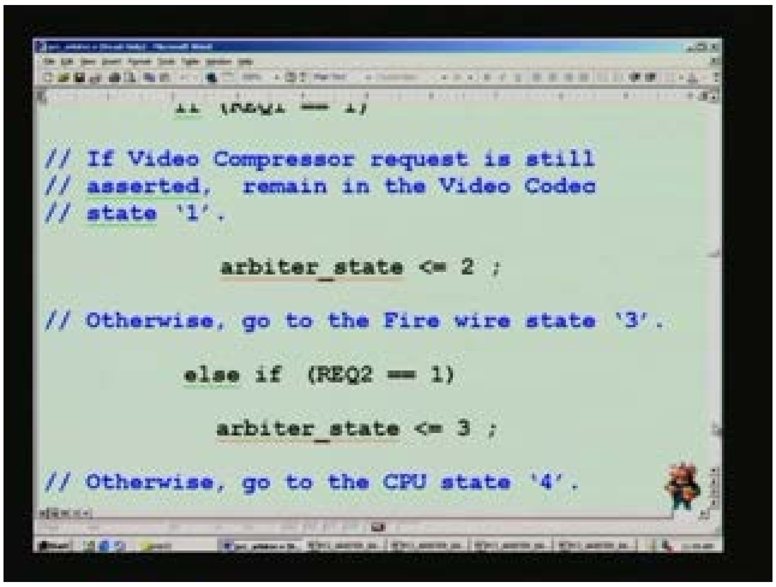
ASM chart anyway reflects the priority that we have already assigned and naturally, the code must follow suit.

(Refer Slide Time: 26:25)



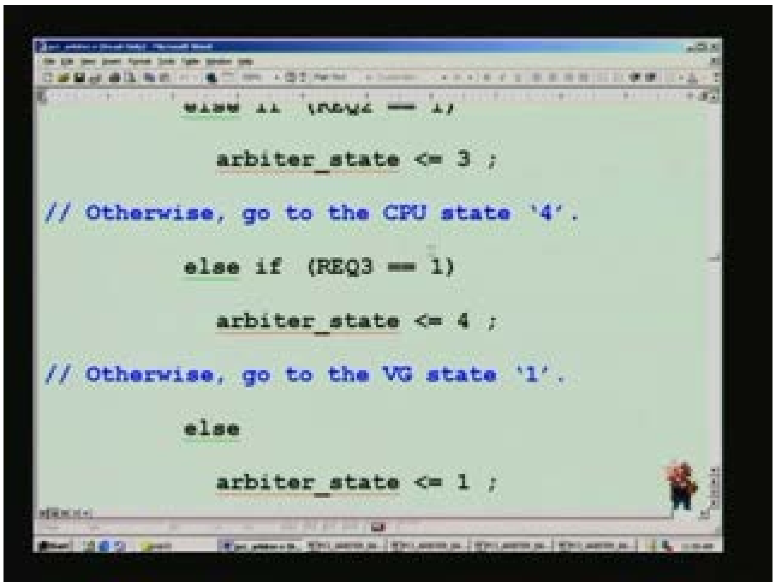
```
                arbiter_state <= 4 ;

// Otherwise, go to the VG

// state '1'.

            else

                arbiter_state <= 1 ;

            end


        2:
            begin
```

(Refer Slide Time: 27:03)



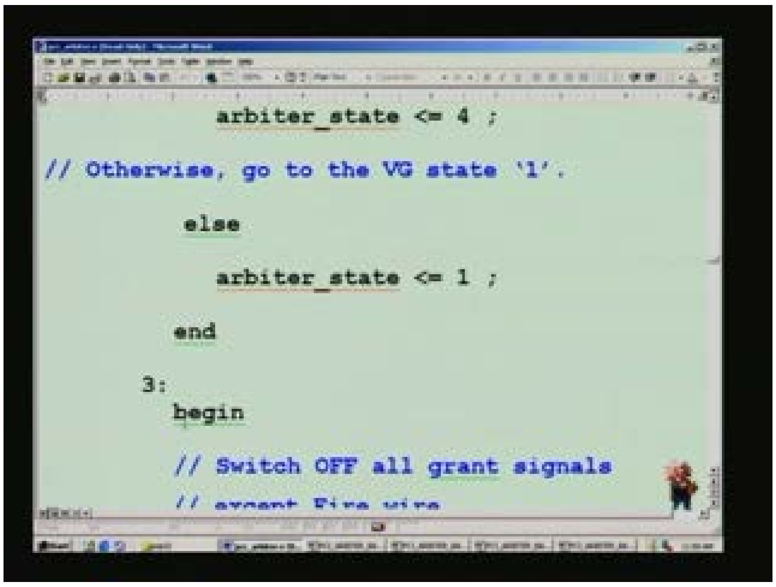
```
            if (REQ1 == 1)

// If Video Compressor request is still
// asserted,  remain in the Video Codec
// state '1'.

                arbiter_state <= 2 ;

// Otherwise, go to the Fire wire state '3'.

            else if  (REQ2 == 1)

                arbiter_state <= 3 ;

// Otherwise, go to the CPU state '4'.
```

(Refer Slide Time: 27:32)



```
            arbiter_state <= 3 ;
// Otherwise, go to the CPU state '4'.
          else if  (REQ3 == 1)
            arbiter_state <= 4 ;
// Otherwise, go to the VG state '1'.
          else
            arbiter_state <= 1 ;
```

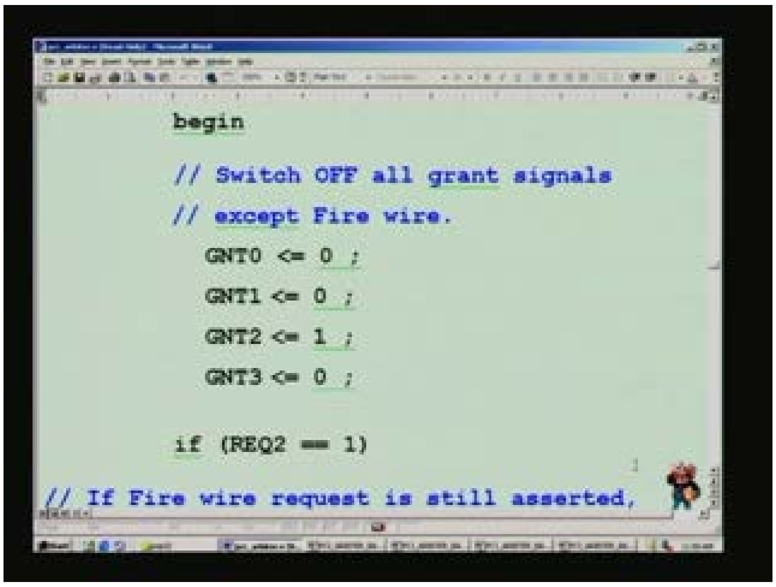
(Refer Slide Time: 28:06)



```
            arbiter_state <= 4 ;
// Otherwise, go to the VG state '1'.
          else
            arbiter_state <= 1 ;
         end
       3:
         begin
         // Switch OFF all grant signals
```

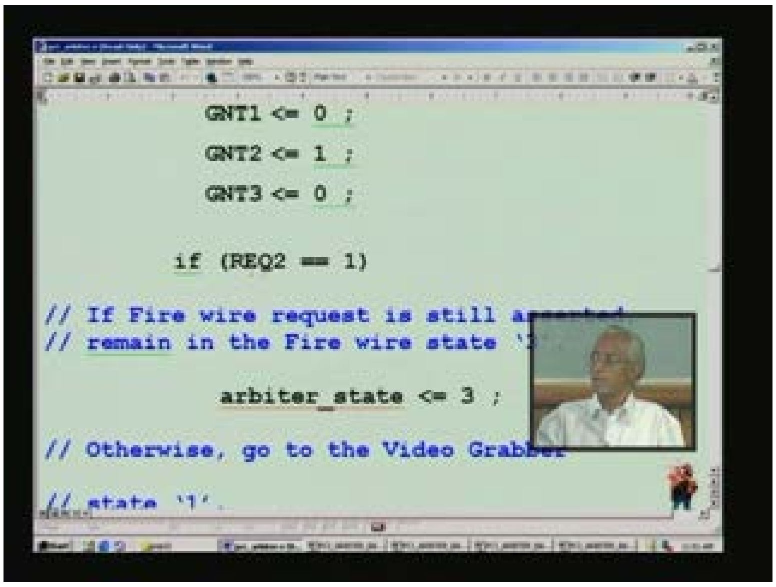
If we have already covered for all other conditions here, as far as video grabber state is concerned and if nothing else is remaining, we will say ‘else’ and then take it to state 1. The state 1 is the same we are in - state 1; so it continues to be in that state. So, in state 2 we had to grant 1 here and these are all exactly same here and you can see that it is in if a request 1 is made. Video compressor request is still asserted, remains in the video codec state here in 2 and if otherwise check for fire wire which is state 3 as well as the request is 1 and otherwise, you will check for the CPU and request 3 and state 4 corresponding. Once again, if none of this is present go to the state 1; so we always go to the video grabber state that was the right side end of the ASM chart that we have seen because we have examined all the conditions. So, no other condition is persisting and therefore, we go back to the VG state that is what we have seen before and once again, that is an ‘end’. Once again, that is a ‘begin’ and there will be an ‘end’; this is corresponding to the fire wire state.

(Refer Slide Time: 28:14)



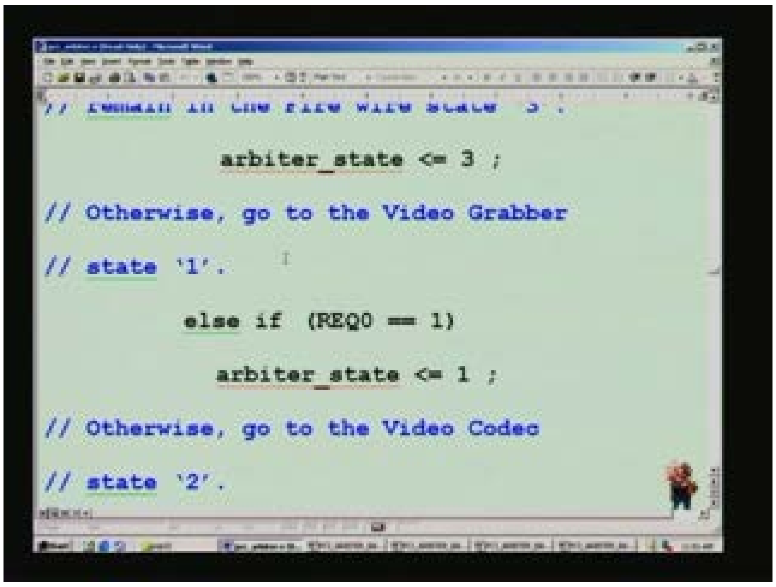
```
        begin
        // Switch OFF all grant signals
        // except Fire wire.
           GNT0 <= 0 ;
           GNT1 <= 0 ;
           GNT2 <= 1 ;
           GNT3 <= 0 ;

        if (REQ2 == 1)
// If Fire wire request is still asserted,
```

(Refer Slide Time: 28:26)



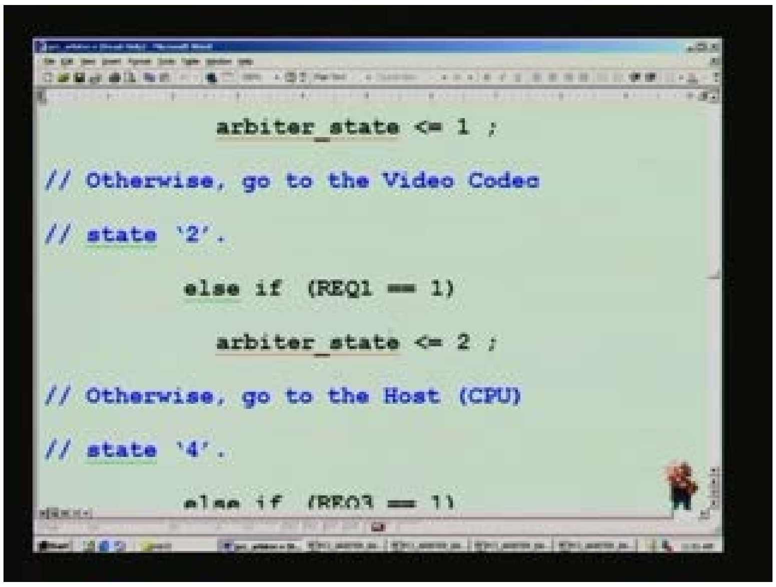
```
           GNT1 <= 0 ;
           GNT2 <= 1 ;
           GNT3 <= 0 ;

        if (REQ2 == 1)
// If Fire wire request is still a
// remain in the Fire wire state '

            arbiter_state <= 3 ;

// Otherwise, go to the Video Grabber
// state '1'.
```

(Refer Slide Time: 28:37)



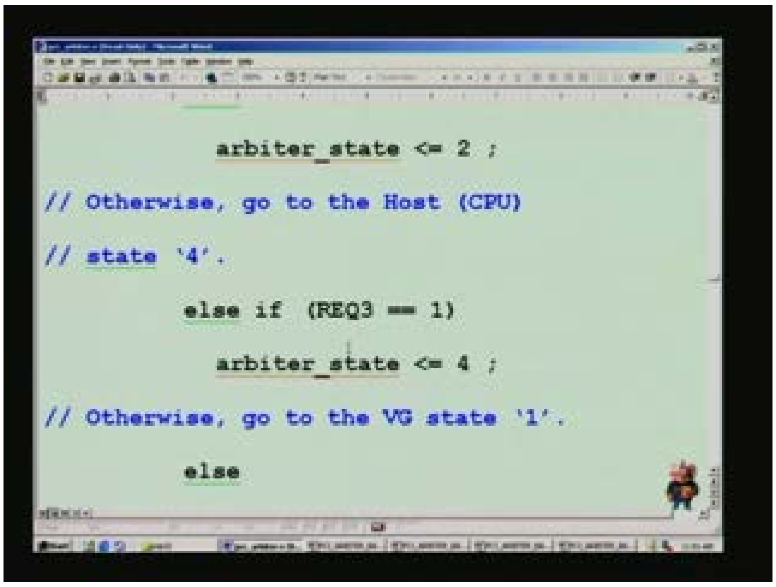
```
// remain in the fire wire state '3'.
                  arbiter_state <= 3 ;
// Otherwise, go to the Video Grabber
// state '1'.
              else if  (REQ0 == 1)
                  arbiter_state <= 1 ;
// Otherwise, go to the Video Codec
// state '2'.
```

(Refer Slide Time: 28:57)



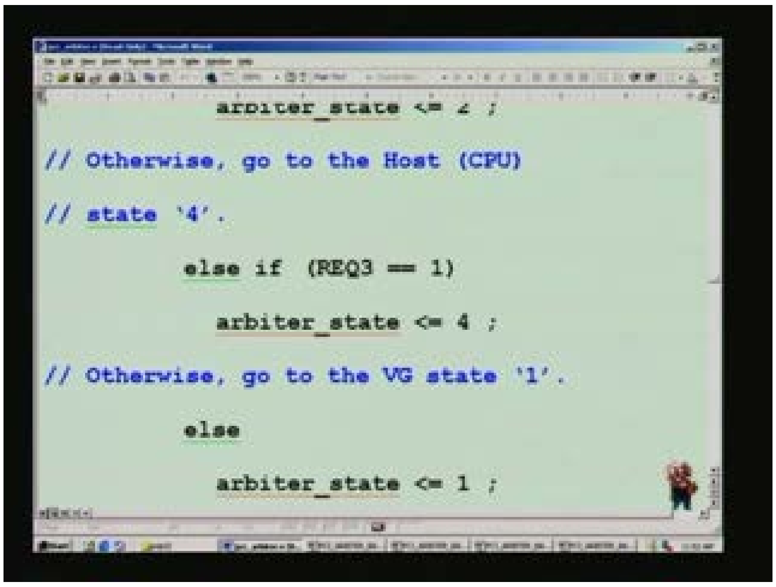
```
                  arbiter_state <= 1 ;
// Otherwise, go to the Video Codec
// state '2'.
              else if  (REQ1 == 1)
                  arbiter_state <= 2 ;
// Otherwise, go to the Host (CPU)
// state '4'.
              else if  (REQ3 == 1)
```

(Refer Slide Time: 29:05)



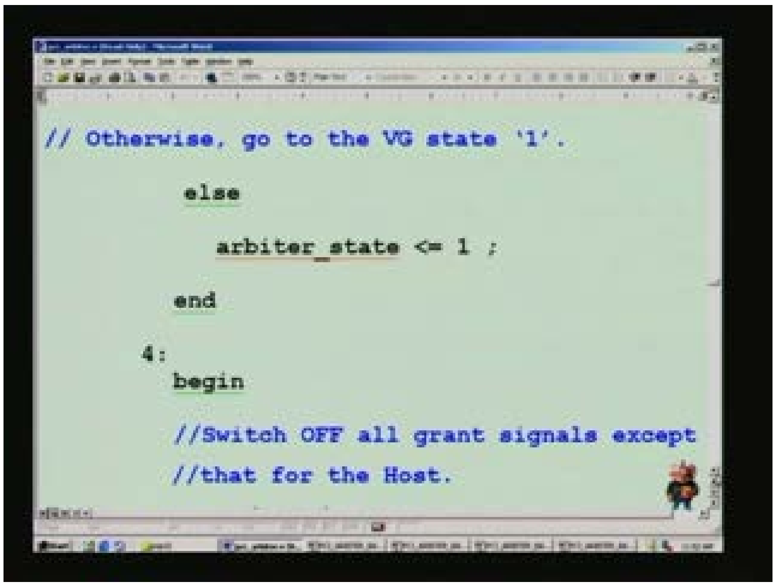
```
              arbiter_state <= 2 ;
// Otherwise, go to the Host (CPU)
// state '4'.
          else if  (REQ3 == 1)
              arbiter_state <= 4 ;
// Otherwise, go to the VG state '1'.
          else
```

(Refer Slide Time: 29:27)

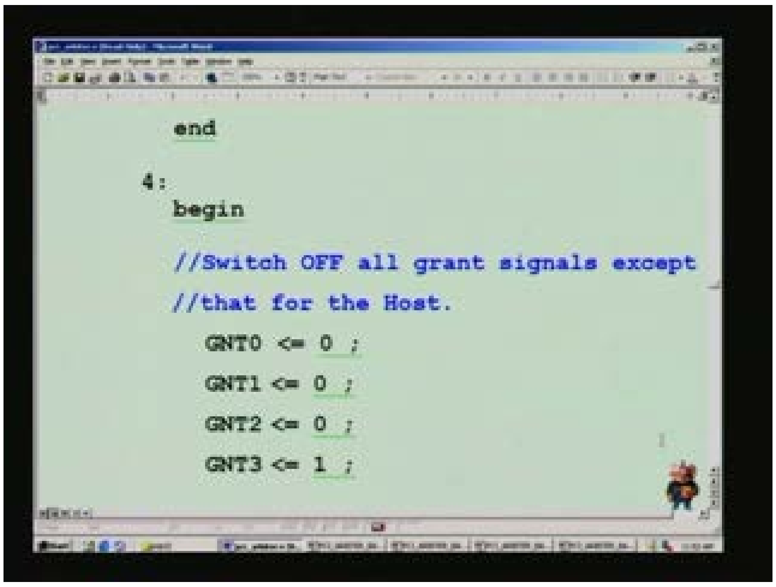


```
              arbiter_state <= 2 ;
// Otherwise, go to the Host (CPU)
// state '4'.
          else if  (REQ3 == 1)
              arbiter_state <= 4 ;
// Otherwise, go to the VG state '1'.
          else
              arbiter_state <= 1 ;
```

(Refer Slide Time: 29:54)

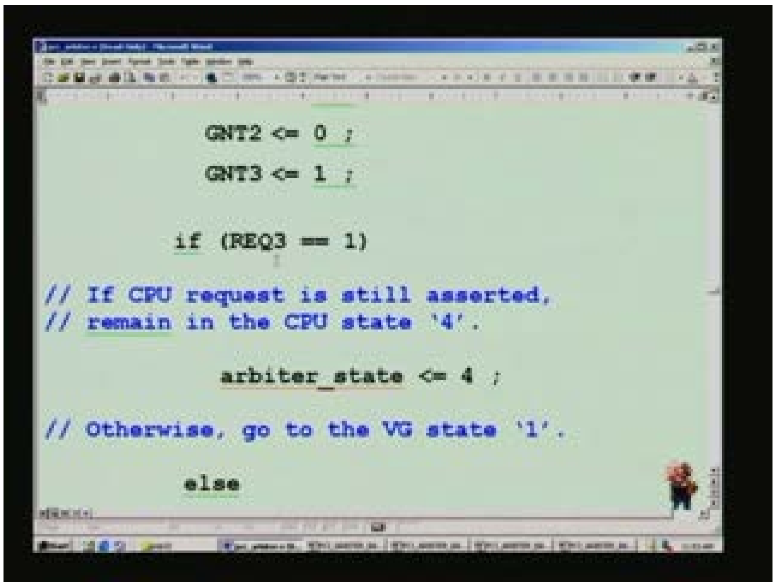

```
// Otherwise, go to the VG state '1'.
        else
          arbiter_state <= 1 ;
        end
      4:
        begin
        //Switch OFF all grant signals except
        //that for the Host.
```

(Refer Slide Time: 36:06)


```
        end
      4:
        begin
        //Switch OFF all grant signals except
        //that for the Host.
          GNT0 <= 0 ;
          GNT1 <= 0 ;
          GNT2 <= 0 ;
          GNT3 <= 1 ;
```

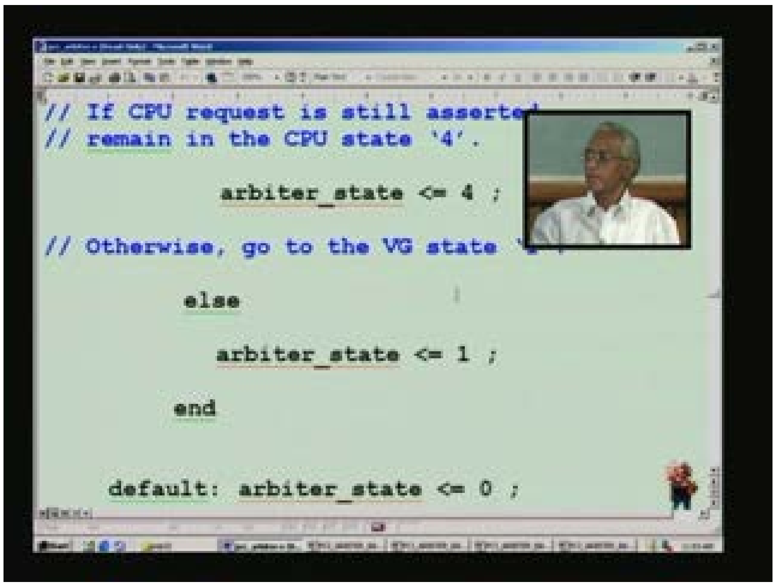
So, here also we switch off all grant signals and give grant only for grant 2. This corresponds to the fire wire and once again, we remain the same state so long as the fire wire request is still made asserted and we continue to be in the third state. That is what we are here and in this state if this is not encountered or if it has finished the usage of the bus, the fire wire has finished the usage of bus then, what we need to do is go back to video grabber and that is what we are doing here (Refer Slide Time: 28:38). Otherwise, go to the codec here; we have to examine request 1 and take to state 2 here. Otherwise, to the host here; request 3 and state are corresponding to this video; this host is it host on this fire wire and this is for the host here. If request 3 is still asserted remains in this host condition state for processing the same and once it has finished its job, the request 3 will become 0; then this will not be valid. So, control goes to the first group; top priority it is a video grabber here and corresponding state is 1 there and actually, we are in fire wire state. Examine different masters and here this is the state corresponding to the host and as you shell once you enter this state, you need to grant only 3 here which correspond to the host and de-asset all other grants.

(Refer Slide Time: 30:22)



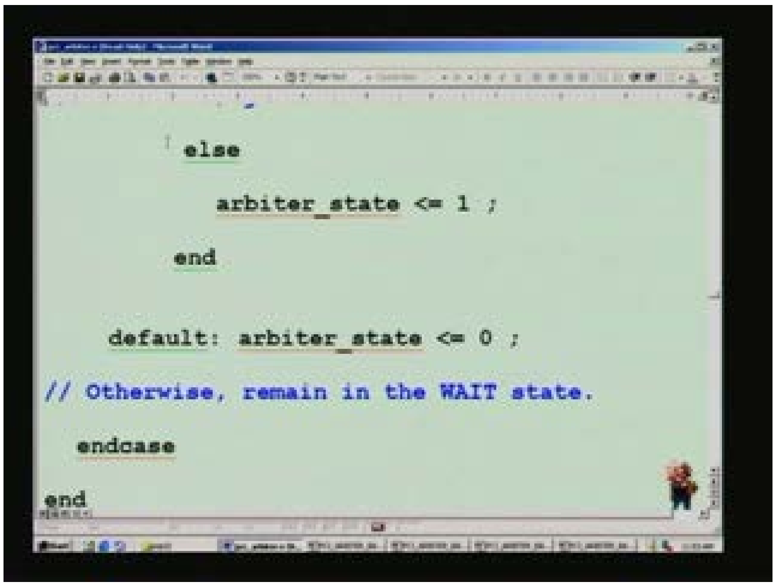
```
            GNT2 <= 0 ;
            GNT3 <= 1 ;

          if (REQ3 == 1)

// If CPU request is still asserted,
// remain in the CPU state '4'.

             arbiter_state <= 4 ;

// Otherwise, go to the VG state '1'.

           else
```

(Refer Slide Time: 30:44)



```
// If CPU request is still asserte
// remain in the CPU state '4'.

             arbiter_state <= 4 ;

// Otherwise, go to the VG state '

           else
             arbiter_state <= 1 ;
          end

      default: arbiter_state <= 0 ;
```

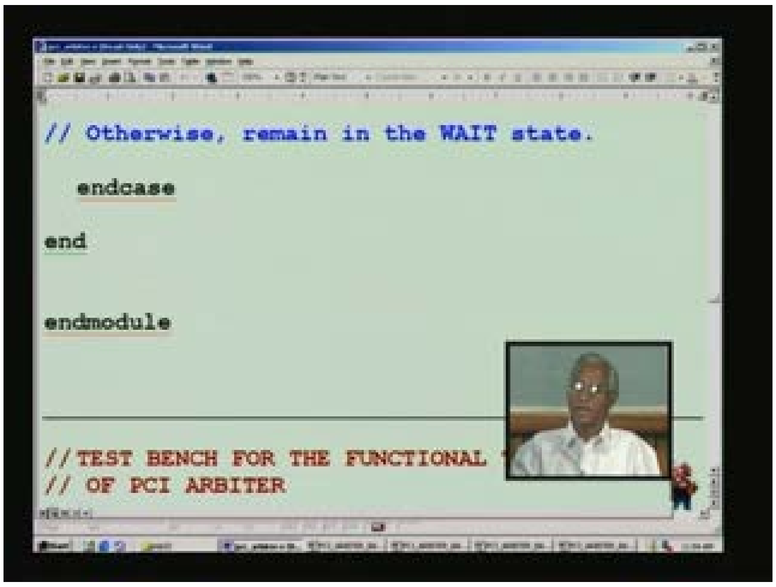
(Refer Slide Time: 30:50)



```
        else
           arbiter_state <= 1 ;
        end

     default: arbiter_state <= 0 ;
// Otherwise, remain in the WAIT state.
   endcase
end
```

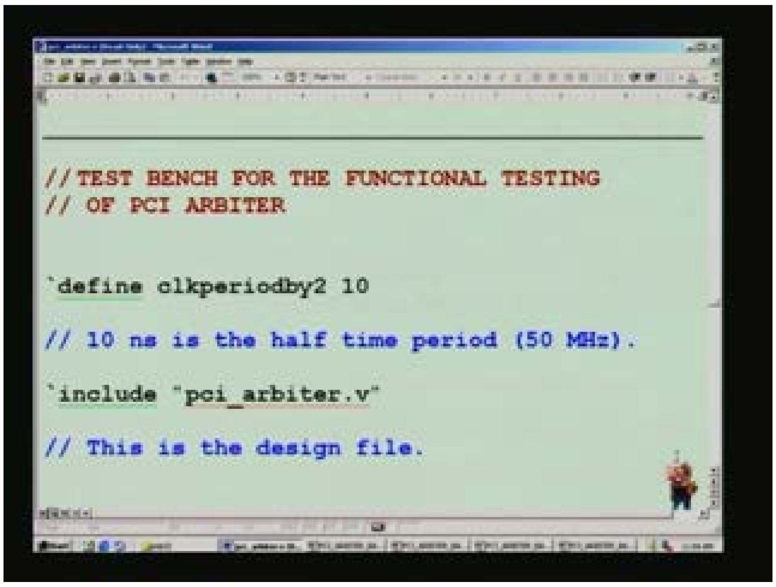
So long as this request 3 is still asserted and you need to remain in the same state always granting 3 here and naturally, it will be in state 4 and if it has completed this job this will go low and it will go to this condition here which is going to be the video grabber state and corresponding state is 1 here and this is the completion of case. We have covered all the aspects as mentioned in the ASM chart and we also need to take care for the do-not cares and tries it that may be encountered and corresponding in this arbiter state. So, in which case you need to take it a safe state; let us say, the wait state here (Refer Slide Time: 31:22) or you can video grabber state also you can also take it to.

(Refer Slide Time: 31:35)



```
// Otherwise, remain in the WAIT state.
   endcase
end

endmodule

//TEST BENCH FOR THE FUNCTIONAL
// OF PCI ARBITER
```

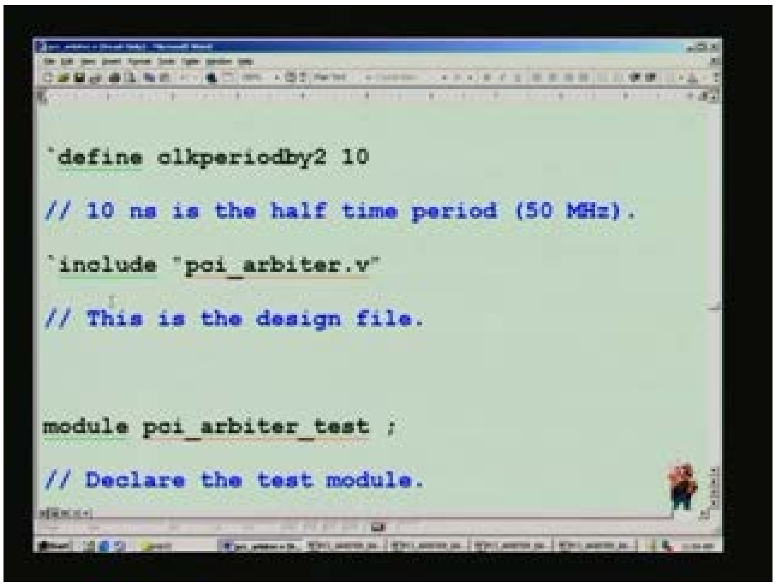
(Refer Slide Time: 31:43)



```
//TEST BENCH FOR THE FUNCTIONAL TESTING
// OF PCI ARBITER

`define clkperiodby2 10

// 10 ns is the half time period (50 MHz).

`include "pci_arbiter.v"

// This is the design file.
```

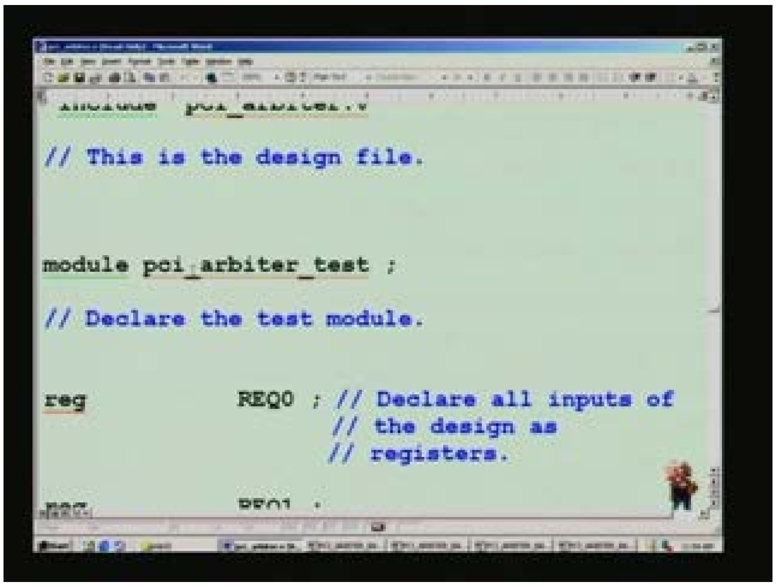
We started with case therefore, there is an endcase here and it was in ‘always’ block and there was a ‘begin’ at the start and their corresponding ‘end’ is this. We started with the module which is the design for the PCI arbiter and naturally, we will have to end with an end module; so we will look into the test bench for the PCI arbiter.

(Refer Slide Time: 31:58)



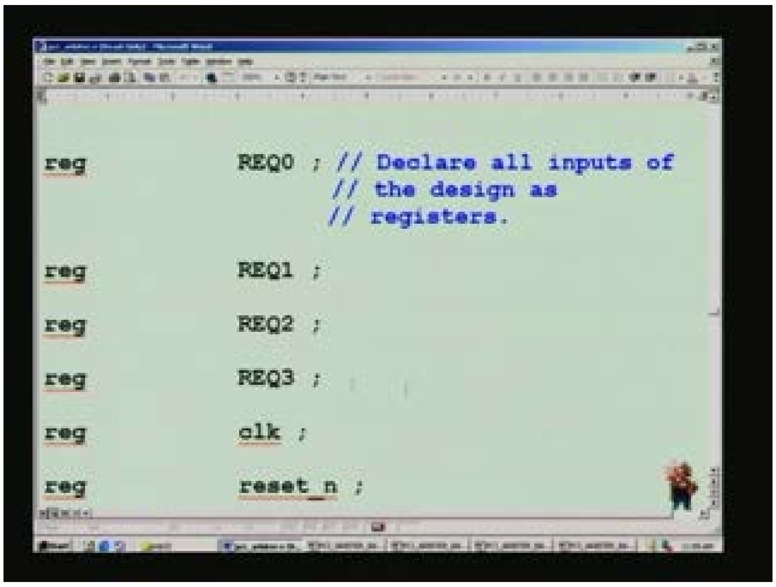
```
`define clkperiodby2 10

// 10 ns is the half time period (50 MHz).

`include "pci_arbiter.v"

// This is the design file.


module pci_arbiter_test ;

// Declare the test module.
```

(Refer Slide Time: 32:07)



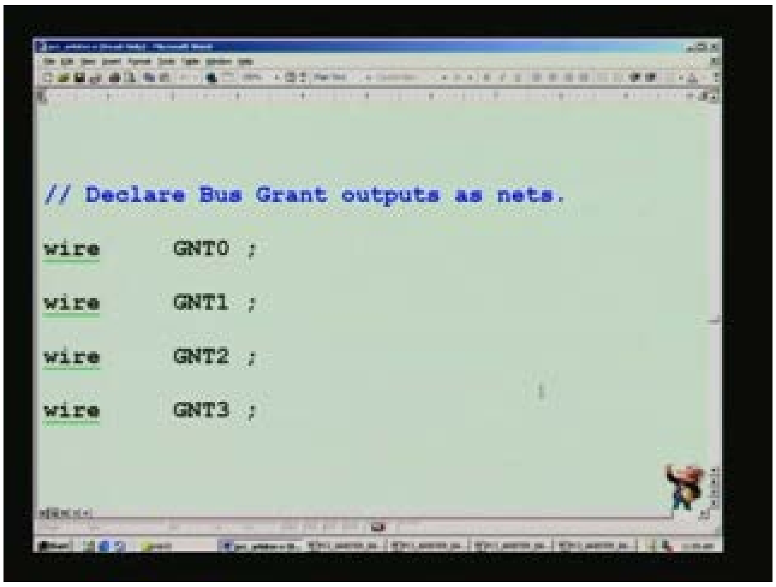
```
`include "pci_arbiter.v"

// This is the design file.


module pci_arbiter_test ;

// Declare the test module.


reg            REQ0 ; // Declare all inputs of
                      // the design as
                      // registers.

reg            REQ1 ;
```

(Refer Slide Time: 32:25)



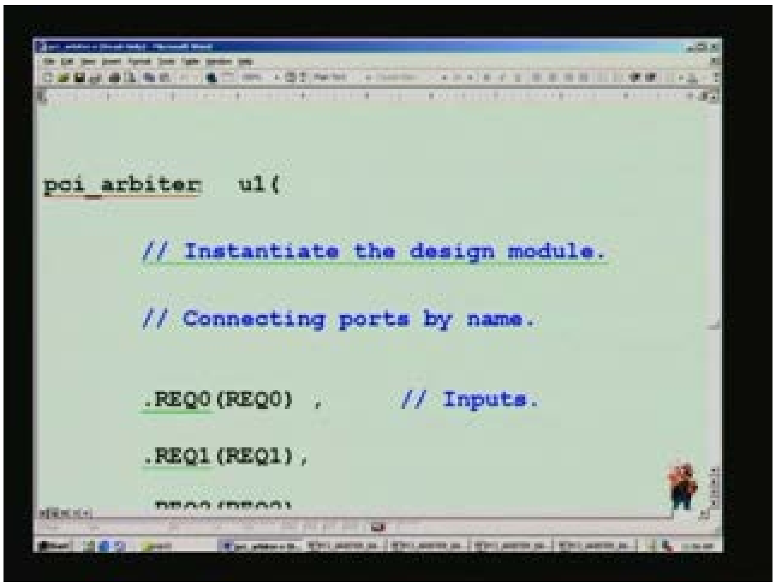
```
reg            REQ0 ; // Declare all inputs of
                      // the design as
                      // registers.

reg            REQ1 ;

reg            REQ2 ;

reg            REQ3 ;

reg            clk ;

reg            reset_n ;
```

(Refer Slide Time: 32:54)



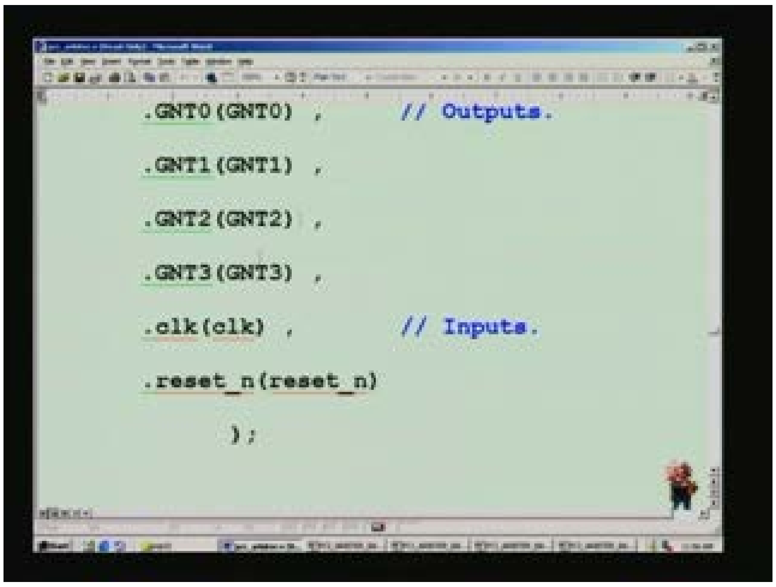
```
// Declare Bus Grant outputs as nets.

wire     GNT0 ;

wire     GNT1 ;

wire     GNT2 ;

wire     GNT3 ;
```

(Refer Slide Time: 33:19)



```
pci_arbiter  u1(

        // Instantiate the design module.

        // Connecting ports by name.

        .REQ0(REQ0)  ,      // Inputs.

        .REQ1(REQ1),
```

(Refer Slide Time: 34:00)

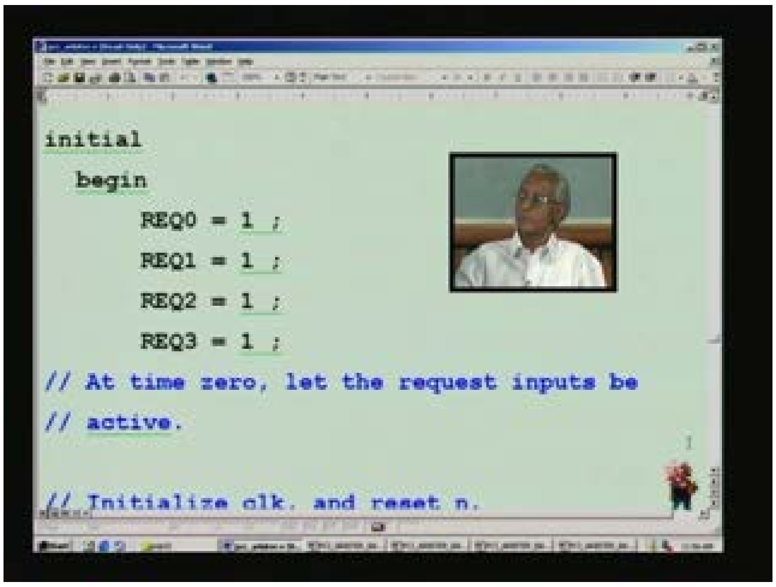


```
.GNT0(GNT0)  ,      // Outputs.
.GNT1(GNT1)  ,
.GNT2(GNT2)  ,
.GNT3(GNT3)  ,
.clk(clk) ,         // Inputs.
.reset_n(reset_n)
      );
```

So, in this case let us say, we want to run at 50 megahertz therefore, we say clock period by 2 is 10 and we need to include the actual PCI arbiter design and that is what is this statement is for. This is a test bench; so, we have to declare the test bench module name must be mentioned here PCI arbiter and the code test.

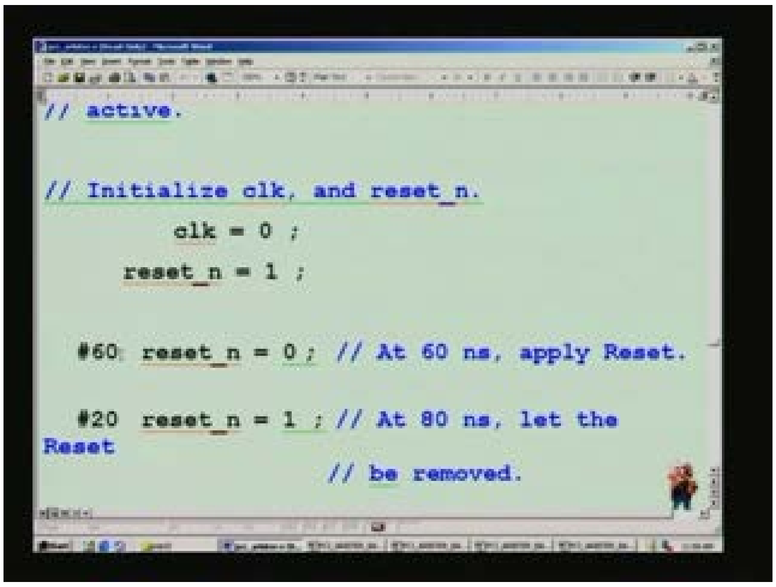
The same nomenclature we have been following all through, has been adopted here too and we have to naturally term it as a module here and being a test bench, we have seen that inputs are all reg and these inputs for the different request corresponding to video grabber, video codec, fire wire and host, these are all request 0 through request 3. This will have to be declared as register and which means, holding the value and so also the case for the other input such as clock - system clock as well as, the power on reset. We have also seen in the test bench, whatever output we have, naturally, the grants signals are the outputs here and they are grant 0 through grant 3 and they must be declared as wire because, we may have several such modules called here and we need to interconnect them. So, this interconnection implies it is a net or wire; that is why we declare it as wire and it has bench. Once we have done this, what we need to do is instantiate this design. This is the design; here we say arbiter and this instantiation we are doing and this happens only once and therefore, there will be just u1.

(Refer Slide Time: 34:15)



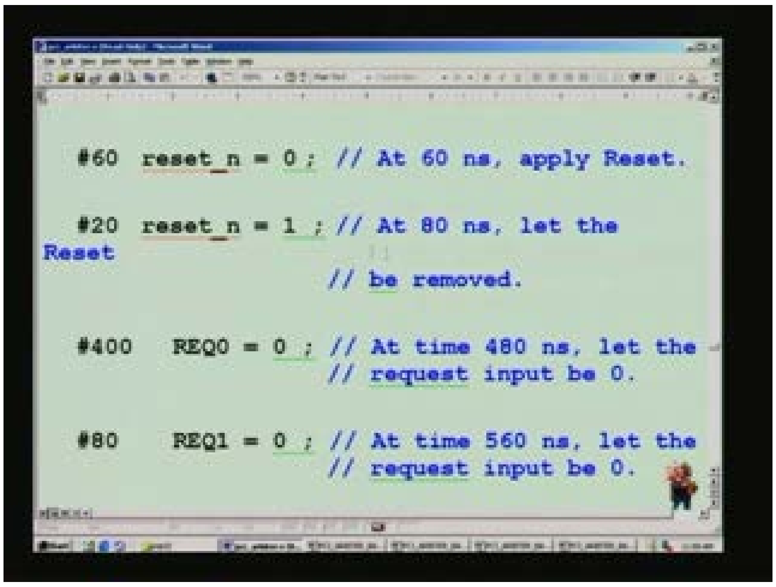
```
initial
  begin
        REQ0 = 1 ;
        REQ1 = 1 ;
        REQ2 = 1 ;
        REQ3 = 1 ;
// At time zero, let the request inputs be
// active.

// Initialize clk, and reset_n.
```

(Refer Slide Time: 35:05)



```
// active.

// Initialize clk, and reset_n.
          clk = 0 ;
      reset_n = 1 ;


  #60  reset_n = 0 ;  // At 60 ns, apply Reset.

  #20  reset_n = 1 ; // At 80 ns, let the
Reset
                      // be removed.
```

(Refer Slide Time: 35:41)

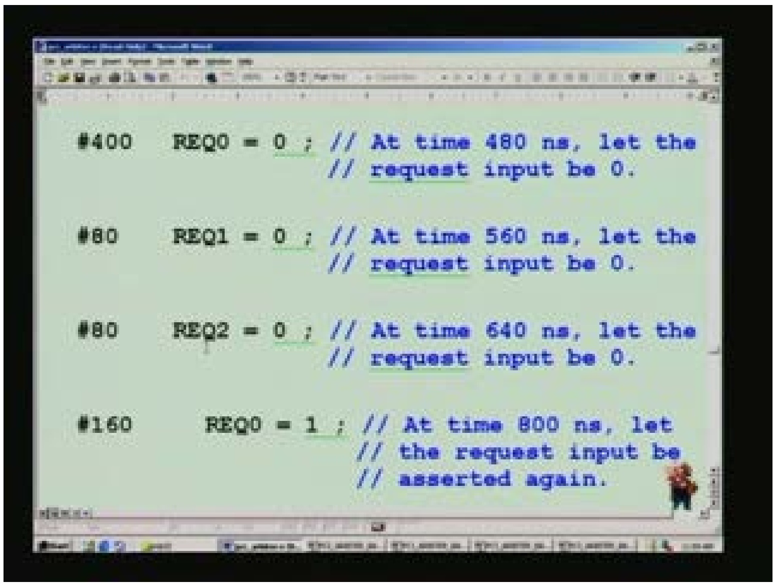


```
   #60  reset_n = 0 ; // At 60 ns, apply Reset.

   #20  reset_n = 1 ; // At 80 ns, let the
Reset
                      // be removed.

   #400   REQ0 = 0 ; // At time 480 ns, let the
                     // request input be 0.

   #80    REQ1 = 0 ; // At time 560 ns, let the
                     // request input be 0.
```

If you had several such designs called again, you need to only change this to u2 u3 and so on and once again, we have seen this connecting force by name and so that we can relocate any other order. These are all the inputs that we are listing here; followed by the outputs and inputs are also mentioned here. As I mentioned before, you need not separate out all the inputs and outputs and it can be any order and that is the beauty of using ports by name – nomenclature. So, the test bench is initialized at these points. Once you initialize it means, the time element is 0 here. So, we start with initial forward by ‘begin’ because there are going to be multiple statements here and at the end there will be an end and so the comment says a time 0, let the request inputs be active.

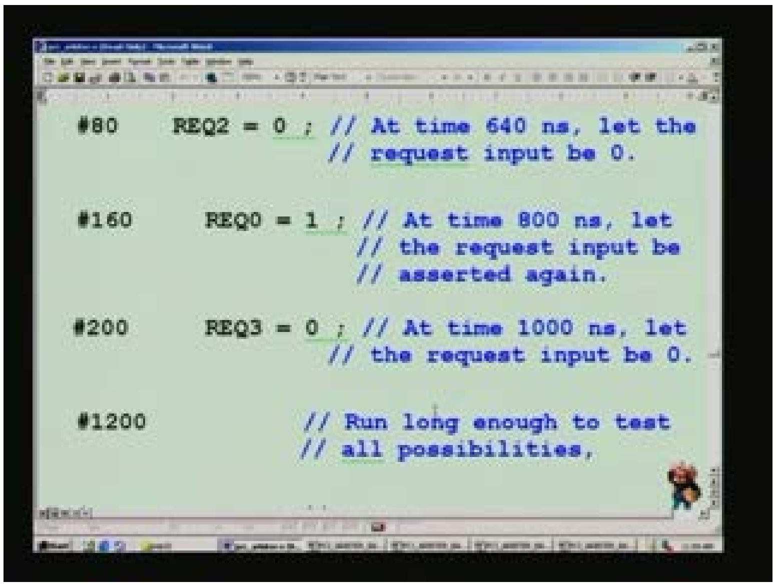
So, what we are going here is so we will make all the request 1. So, that means to say there are 4 masters and all the 4 masters are asserted. So, let us see what is the back connect wave forms we will see what will be given the grant. So, in spite of the fact that all the 4 are asserted and we have also to initialize clock and that is what we are doing here because, we are interested in posedge of the clock. We naturally start with 0 and this is a de-asserted reset condition because, it is an active load here. This n stands for negative or you can just say l also there implying it is active low. It is de-asserted here and we assert reset at after 60 nano seconds. So, that means to say for 20 nano seconds we are applying the reset pulse and that means, once again this is restored to normal condition which is not reset. So, once the reset is withdrawn, it is not processing from here onwards and let us say at after the lapse of 400 nano seconds and that is 480 to start with because, we have to add 60 plus 20 from 0 state and at 480 nano seconds what we do is earlier it was a request 0 so also, the other request where asserted that is 1.

(Refer Slide Time: 36:26)



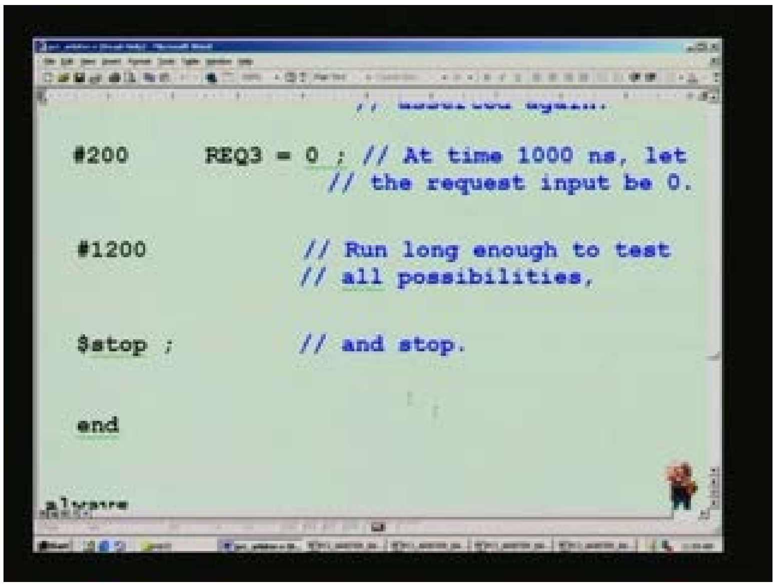
```
#400   REQ0 = 0 ; // At time 480 ns, let the
                  // request input be 0.

#80    REQ1 = 0 ; // At time 560 ns, let the
                  // request input be 0.

#80    REQ2 = 0 ; // At time 640 ns, let the
                  // request input be 0.

#160     REQ0 = 1 ; // At time 800 ns, let
                    // the request input be
                    // asserted again.
```

(Refer Slide Time: 36:47)



```
#80    REQ2 = 0 ; // At time 640 ns, let the
                  // request input be 0.

#160     REQ0 = 1 ; // At time 800 ns, let
                    // the request input be
                    // asserted again.

#200     REQ3 = 0 ; // At time 1000 ns, let
                  // the request input be 0.

#1200            // Run long enough to test
                 // all possibilities,
```

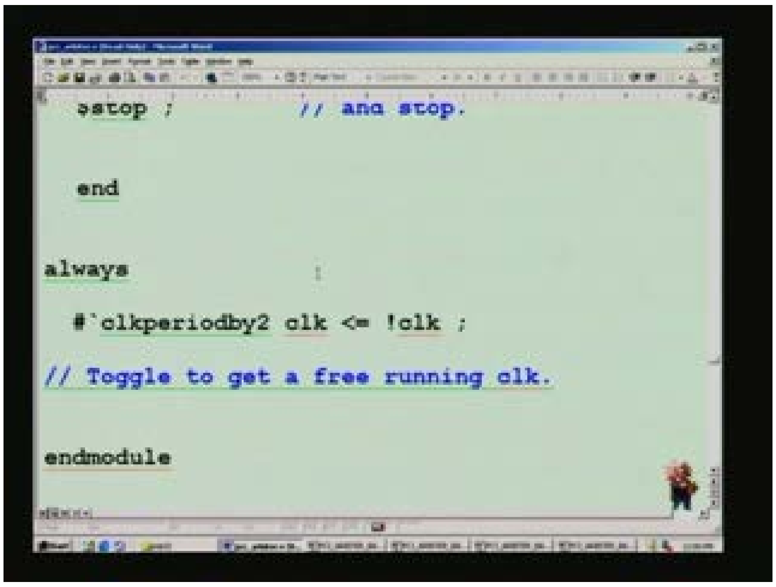
(Refer Slide Time: 37:08)



```
                      // asserted again.

#200      REQ3 = 0 ; // At time 1000 ns, let
                  // the request input be 0.

#1200               // Run long enough to test
                    // all possibilities,

$stop ;             // and stop.

end
```

Now, we are deliberately making these changes here at a 480 we will make it request 0 low; that means, withdraw the request. In other words, it has finished using the PCI bus and at 560 nano seconds we withdraw request 1 and so on request 2 and after at 800 nano seconds let us say, this is the top priority. That video grabber request and we assert it to once again here and when we view the wave form, we will come back to this. so, that we will see where we are and after further time 1000 nano seconds, we de-asset request 3 and what has been happened here is, we have only tested partially it is not full-fledged test as such and may be 7, 8 conditions we are checking from this and we will continue to run for some more time it need not be 1200 can be 100 or 200 it only is a cumulative here.

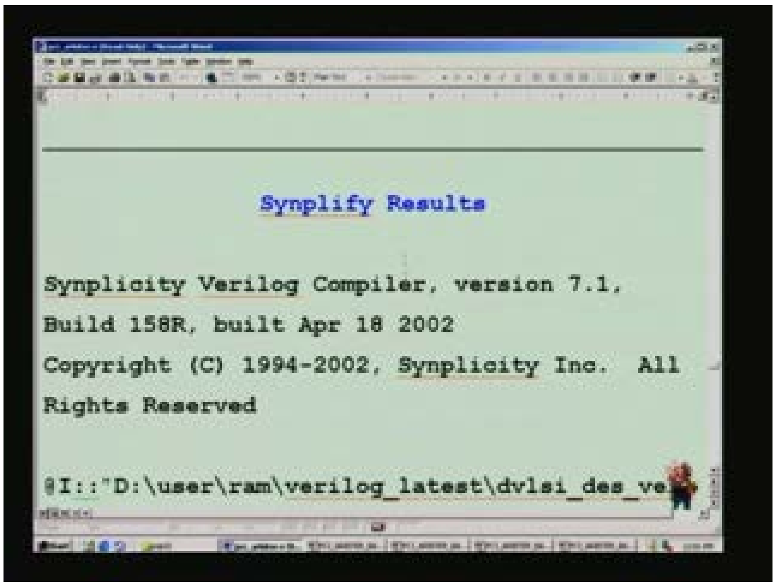
(Refer Slide Time: 37:35)



```
  $stop ;          // and stop.

  end

always

  #`clkperiodby2 clk <= !clk ;

// Toggle to get a free running clk.

endmodule
```

Once we are through with the testing you can just stop here; so I made it little longer here just to have some question here and so next 1 is we need to run the clock. So, that is possible only if you toggle a particular signal we call this clock we initialize it to 0 earlier at 0 nano second and this is being toggled by inverting and assigning it to the same signal and this happens every time after the elapse of this much time this we have set as 10 nano seconds. So, at every 10 nano seconds time clock is made as its inverse so if it is 0 it will become 1 every time nano second; that means to say every 20 seconds it will output 1 clock so that transforms to 50 megahertz and that is how we get the 50 megahertz.

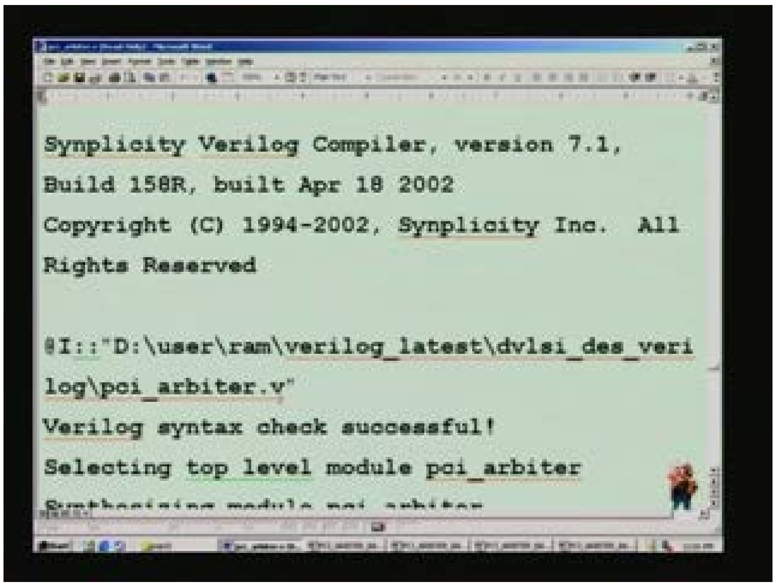
(Refer Slide Time: 38:30)



Synplify Results

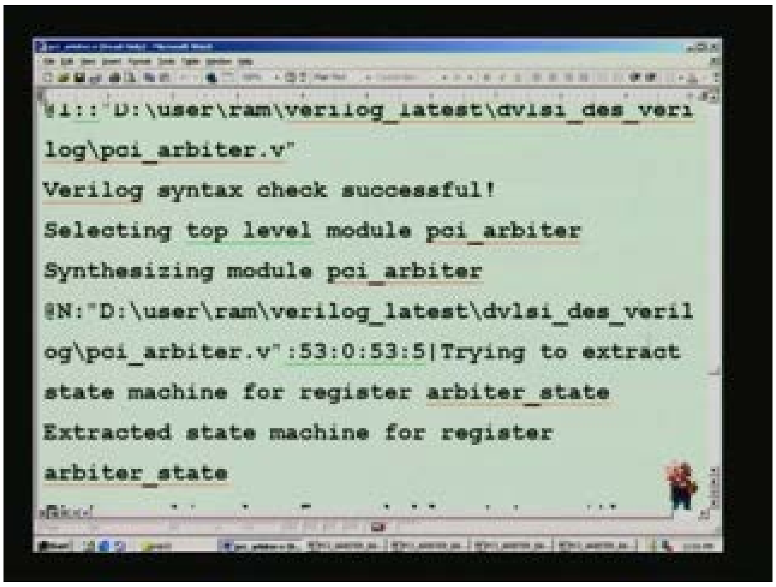
```
Synplicity Verilog Compiler, version 7.1,
Build 158R, built Apr 18 2002
Copyright (C) 1994-2002, Synplicity Inc.  All
Rights Reserved

@I::"D:\user\ram\verilog_latest\dvlsi_des_ve
```

(Refer Slide Time: 38:38)



```
Synplicity Verilog Compiler, version 7.1,
Build 158R, built Apr 18 2002
Copyright (C) 1994-2002, Synplicity Inc.  All
Rights Reserved

@I::"D:\user\ram\verilog_latest\dvlsi_des_veri
log\pci_arbiter.v"
Verilog syntax check successful!
Selecting top level module pci_arbiter
Synthesizing module pci_arbiter
```

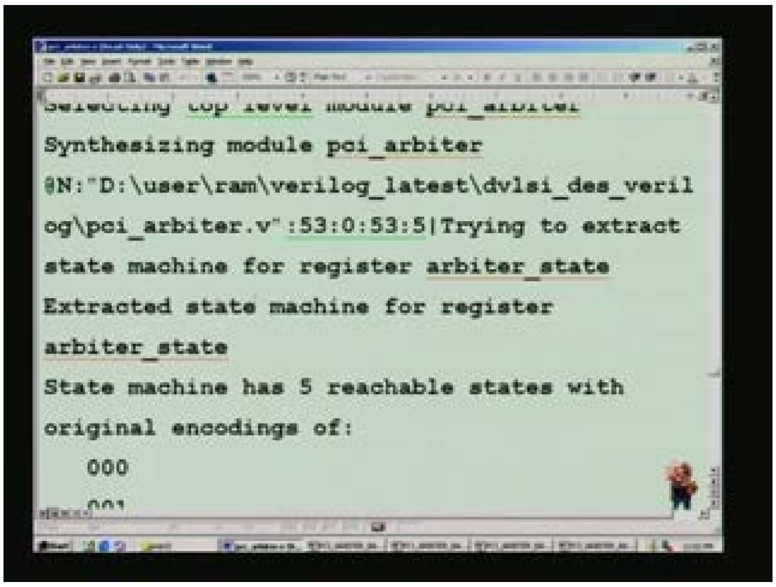
(Refer Slide Time: 38:45)



```
@I::"D:\user\ram\verilog_latest\dvlsi_des_veri
log\pci_arbiter.v"
Verilog syntax check successful!
Selecting top level module pci_arbiter
Synthesizing module pci_arbiter
@N:"D:\user\ram\verilog_latest\dvlsi_des_veril
og\pci_arbiter.v":53:0:53:5|Trying to extract
state machine for register arbiter_state
Extracted state machine for register
arbiter_state
```

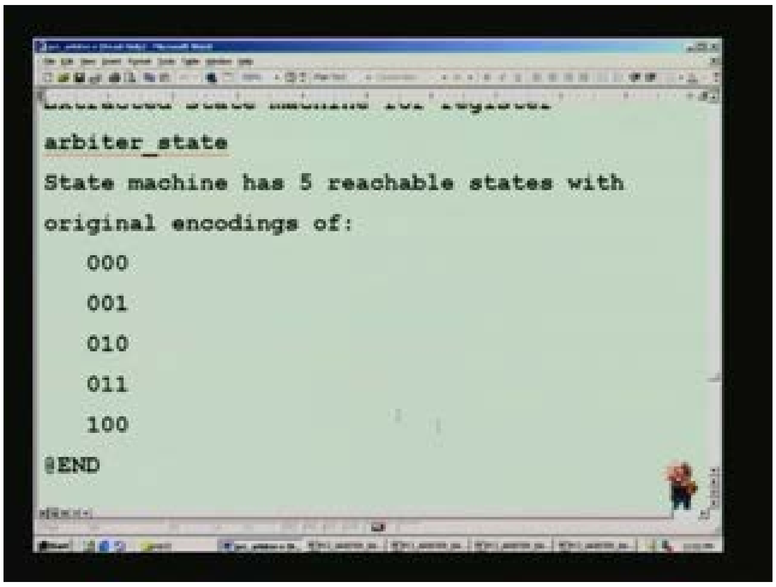
We started with module there must be an end module for the test bench as well and these are all the results for example, synthesis results using simplify is reported here and this is the PCI arbiter dot v is the actual design and its reports verilog syntax is successful. Remember, earlier what we had done is; so if this the ASM chart we have given 0 1 as the condition here 1 then 2 corresponding to the grants of the 4 masters that we have and this is the fourth master here and this is 0 1 2 3 up to 4 in decimal.

(Refer Slide Time: 39:29)



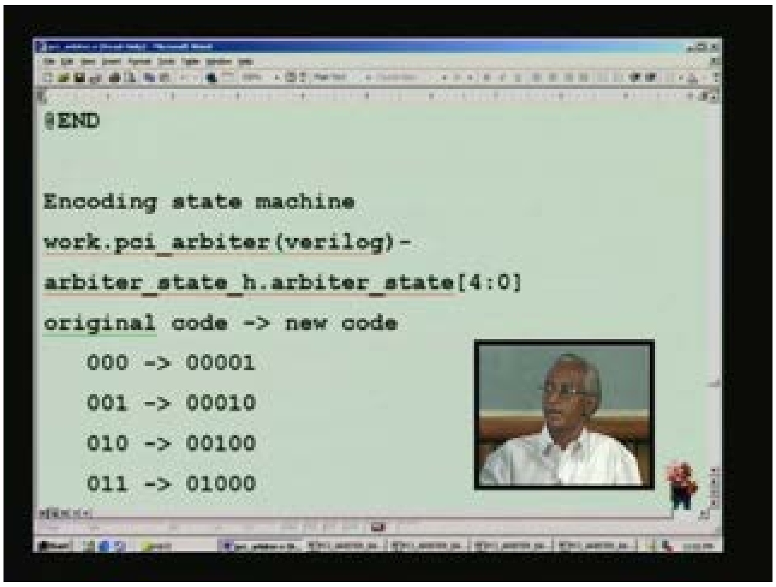
```
Selecting top level module pci_arbiter
Synthesizing module pci_arbiter
@N:"D:\user\ram\verilog_latest\dvlsi_des_veril
og\pci_arbiter.v":53:0:53:5|Trying to extract
state machine for register arbiter_state
Extracted state machine for register
arbiter_state
State machine has 5 reachable states with
original encodings of:
   000
   001
```

(Refer Slide Time: 39:45)



```
arbiter_state
State machine has 5 reachable states with
original encodings of:
   000
   001
   010
   011
   100
@END
```

(Refer Slide Time: 40:05)

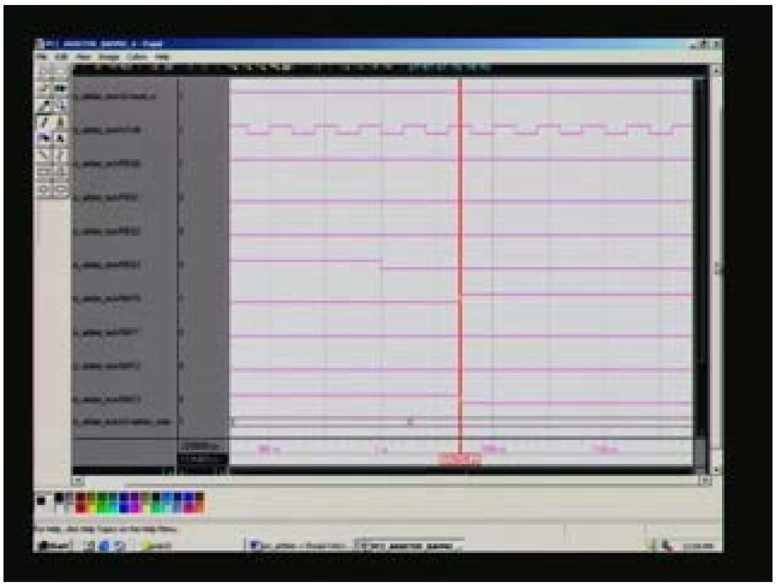


```
@END

Encoding state machine
work.pci_arbiter(verilog)-
arbiter_state_h.arbiter_state[4:0]
original code -> new code
   000 -> 00001
   001 -> 00010
   010 -> 00100
   011 -> 01000
```

Now, let us see what this tool does. So, this is what you had here and these are all the usual reports. What it says here is what is important here is, it is trying to extract state machine for registrar arbiter state and says it has extracted state machines for an arbiter state. So, what it means is it inspects although we have given 0 1 corresponding these are all the states here and we had as for 0 1 2 3 and 4 that is what we have seen the ASM chart.

Now, although we have as for this it requires only 3 bits here that means 3 flip-flops are required. Whereas, the tool has assigned different manner say for example, what it has done is the original rules what was requested was this and what we designed was this state actually what the tool after optimization is different. Now, it has taken 5 bits although it has increased number of flip-flops it has he would recognize this as 1 hot machine, you can see there is only one entry here in hot machine. All the states should have just 1 entry there. So, 1 here next is 2, here this decimal weight if you take and 4 here then 8 here 60 and so on so corresponding to all this.

(Refer Slide Time: 41:17)



So the tool has that much intelligence to map it to more better optimization and from that point of view the tool must have allocated this and if you see the simplified optimal coding we would precisely see what has been cooperated here. Before we go on to the details of this, what we will see is the wave forms after back annotation. See for example, I will just explain before zooming here we have a reset here and that is applied at this point of time. So, you remember that at 60 nano second so we had and here it is mark 50 nano second and each will be duration is 5 nano second. There are 10 here and 50 by 10 is a 5 nano second here and this corresponds to around here. So, that is 60 nano seconds that is what we have here; exactly 60 nano seconds and it remain for 20 nano second and there after it goes high.

So, the system starts working only at the following edge of the posedge of the clock that is at this edge and so here we had, will zoom this, you aim for that we had all the 4 requests made. So, you can see this wave form just on the top of the number employing that they are all 1 although it is showing 0 it was click at some other point that is why it is showing 0. But, actually what you should see is this is always on the top of the member; so, which means all are them are (( )).

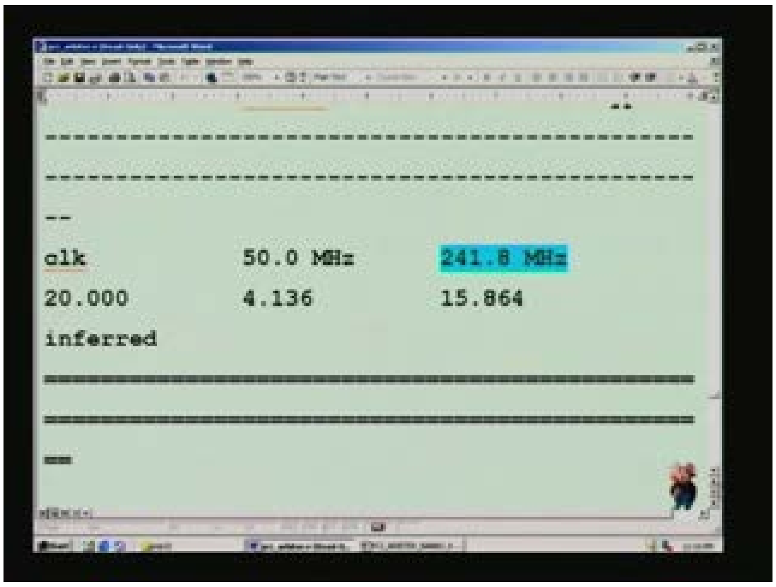
So all are 1; we started with the 1 and let us see where the first grant is who will be given the first grant. Say, for example, here these are all the grants; 4 grants here and this is the initialization state and where in it is made 0 here and very first grant, video grabber grant is awarded here and that is because this request 0 will be sent only at the posedge of this clock. That is at this point after the reset is removed and the grant will be automatic; I mean will be issued only at the following edge of the clock because it cannot change instantaneously. This, we have seen here before in mono shot also same thing; so, it should happen here because it is back annotated at file we should get the grant even delayed from with respect to this edge. For example, you can see here, so this is corresponding to this point somewhere here where as it is here. So, this will be around some 5 nano seconds or so. We will see in next wave form; so grant 0 is a asserted here and this what here design PCI arbiter has done and all the 4 requests were made here and naturally, it changes state here from 0 to 1 because 1 corresponds to the video grabber state in which case grant 0 is given here.

So, in the next wave form what we have is so grant 0 which is high here and request 0 has gone low here which will be sent only at the posedge clock and it will take effect. Grant will be removed; grant 0 will be removed only at the following posedge of the clock and once again, because it is a back connected thing you see a time delay here and you can see this grant 0 is removed at this point of time; that is the delay is also mentioned here that is 5.4 nano seconds.

Here, grant is always late whereas the state itself now you see that because of simplified overriding our states after state 1 2 which has gone to state 4. So, that is what we have pointed out before and here the actual state changes much faster this is only 1.768 menu second and with reference to the postage of the clock and this grant 0 is withdrawn here and grant 1is avoided here and request 1goes low here. Therefore, after the second clock from there and once again after delay a 5.4 nano second and grant 1 must go low this is corresponding to the request 1 going low here and simultaneously, without any time log you see even at this point you can see that grand 0 is withdrawn grant 1 is immediately given. So, this is the case for grant 1 and grant 2 and going to this third wave form you see here after 4 we have got a 8 here in this case grant 2 is de-asserted and grant 3 is issued here because grant 3 was already waiting for quite a long time and request 2 is also gone here. In the last wave form you can see request 3 going low here and in between in the earlier wave forms request 0 has been asserted again. Therefore, it is high here and although top priority has come here unless the lowest priority has passed completely, its usage of the bus it would not release the bus.

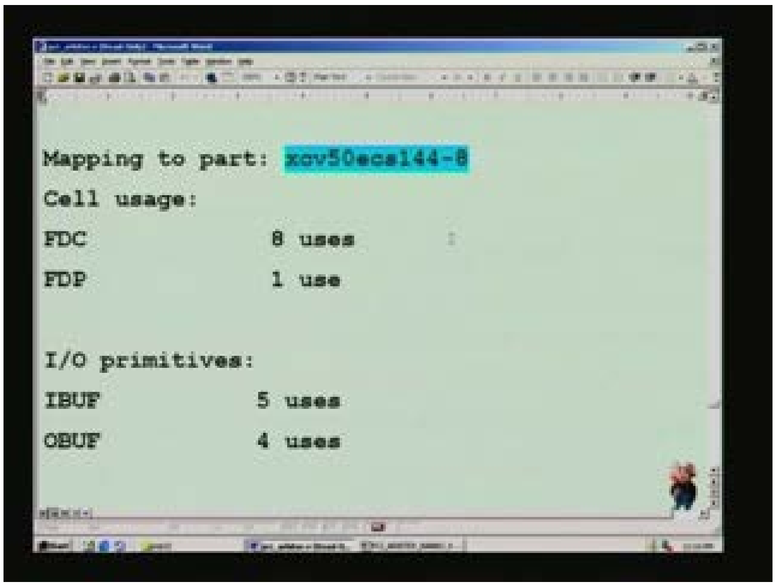
So, only after the release of the bus this top priority can get and that is what is happening here. It is released here; second clock h and once again 5.4 nano second delay grant 0 is avoided here and simultaneously, grant 3 is withdrawn here and this completes these checks the functionality write up to the back annotation.

(Refer Slide Time: 47:18)



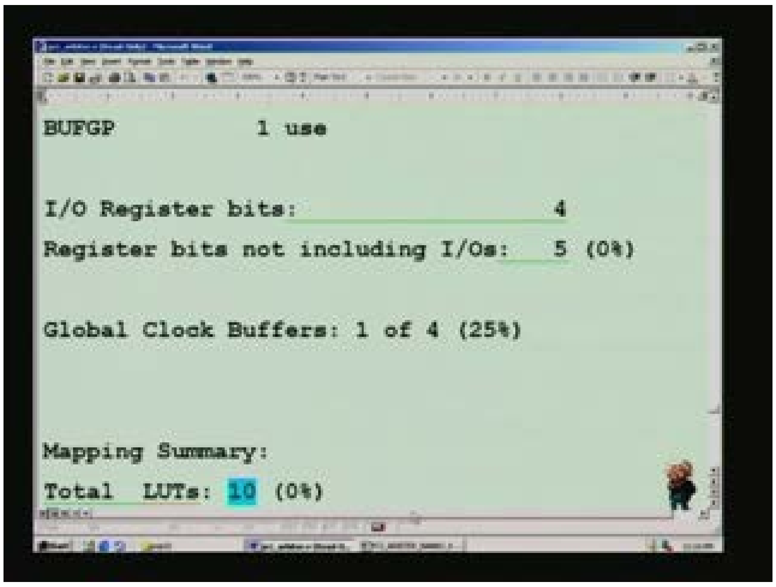
```
-----------------------------------------------
-----------------------------------------------
--
clk            50.0 MHz       241.8 MHz
20.000         4.136          15.864
inferred
===============================================
===============================================
===
```

(Refer Slide Time: 47:29)



```
Mapping to part: xcv50ecs144-8
Cell usage:
FDC                8 uses
FDP                1 use

I/O primitives:
IBUF             5 uses
OBUF             4 uses
```

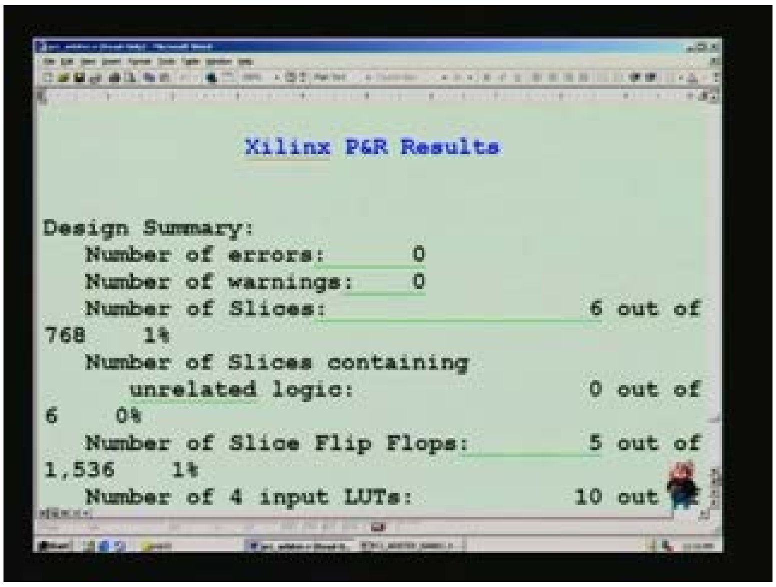
(Refer Slide Time: 47:47)



```
BUFGP            1 use

I/O Register bits:                     4
Register bits not including I/Os:      5  (0%)

Global Clock Buffers: 1 of 4  (25%)


Mapping Summary:
Total  LUTs: 10  (0%)
```

We have been looking into this; 1 2 4 8 16 are the states assigned by the synthesis tool and you will be surprised to see it is reporting very high frequency of 242 megahertz all though we have request for 50 megahertz. This is because we have selected the fastest available device in vertex each series and values and lowest capacity which is 50000 and lowest package and highest PDF and that is the reason why you get some 242 megahertz there and it has taken (( )) 10 number of LUTs here. So, the design is so compact and probably optimization has done a good work there.

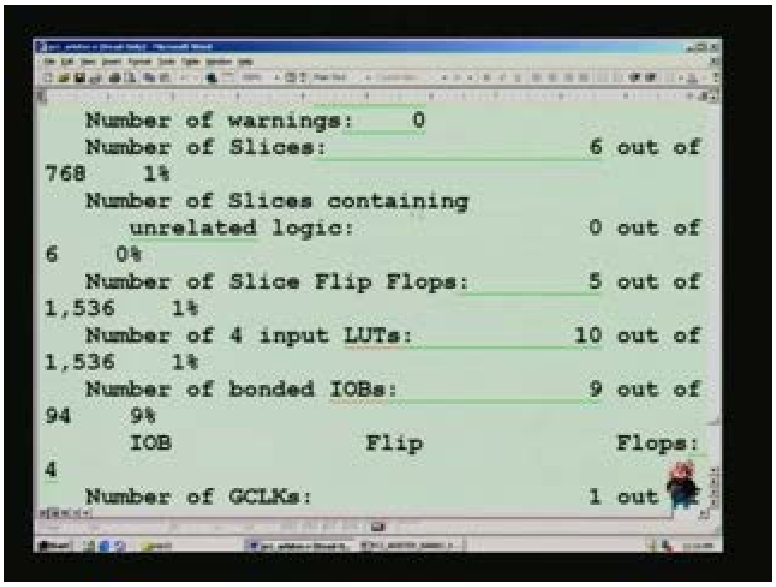
(Refer Slide Time: 47:59)



Xilinx P&R Results

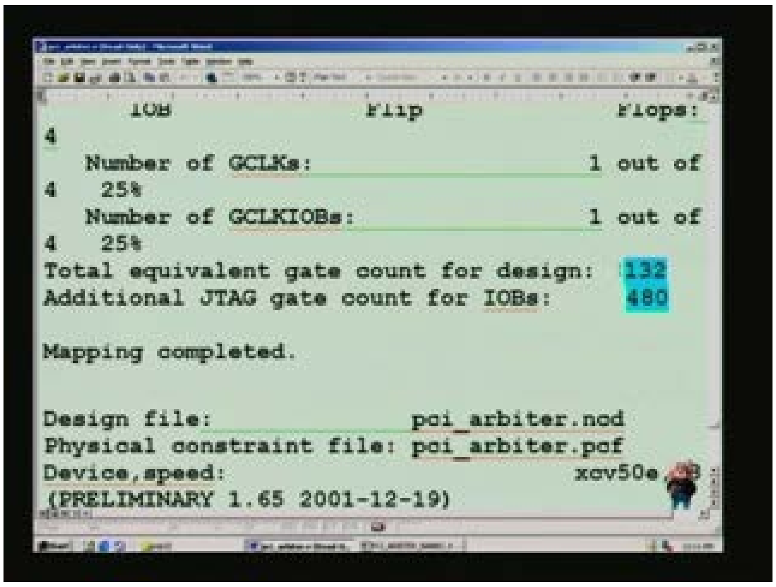
```
Design Summary:
   Number of errors:        0
   Number of warnings:      0
   Number of Slices:                         6 out of
768    1%
   Number of Slices containing
      unrelated logic:                       0 out of
6    0%
   Number of Slice Flip Flops:               5 out of
1,536    1%
   Number of 4 input LUTs:                  10 out
```

(Refer Slide Time: 48:02)



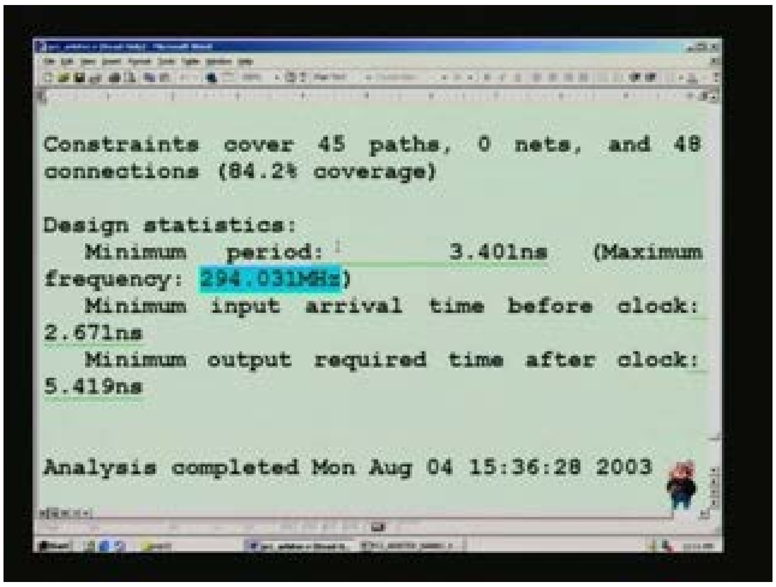
```
   Number of warnings:      0
   Number of Slices:                         6 out of
768    1%
   Number of Slices containing
      unrelated logic:                       0 out of
6    0%
   Number of Slice Flip Flops:               5 out of
1,536    1%
   Number of 4 input LUTs:                  10 out of
1,536    1%
   Number of bonded IOBs:                    9 out of
94    9%
      IOB              Flip              Flops:
4
   Number of GCLKs:                          1 out
```

(Refer Slide Time: 47:08)



```
      IOB                    Flip                  Flops:
4
   Number of GCLKs:                          1 out of
4   25%
   Number of GCLKIOBs:                       1 out of
4   25%
Total equivalent gate count for design:   132
Additional JTAG gate count for IOBs:      480

Mapping completed.

Design file:              pci_arbiter.ncd
Physical constraint file: pci_arbiter.pcf
Device,speed:                             xcv50e
(PRELIMINARY 1.65 2001-12-19)
```

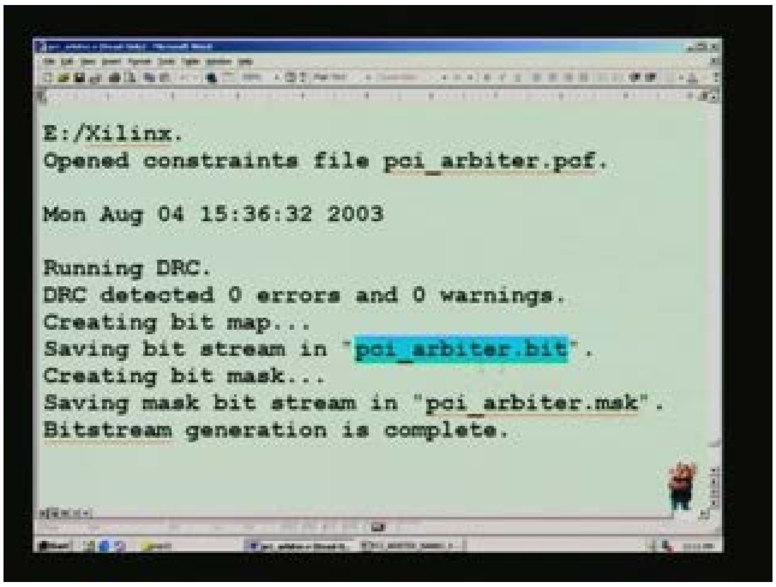
(Refer Slide Time: 48:25)



```
Constraints cover 45 paths, 0 nets, and 48
connections (84.2% coverage)

Design statistics:
   Minimum  period:          3.401ns   (Maximum
frequency: 294.031MHz)
   Minimum input arrival time before clock:
2.671ns
   Minimum output required time after clock:
5.419ns


Analysis completed Mon Aug 04 15:36:28 2003
```

And, Xilinx place and route results for same or number of slices, etcetera it reports and you can see the gate count here. It is just 132 gates and if you want JTAG compatibility of IOBs then you need to sacrifice more gates here and here. You will be surprised to find 242 megahertz reported by synthesis tool, is enhanced to 294 megahertz here. But, actually this will be misleading because this only a small portion of the total design.

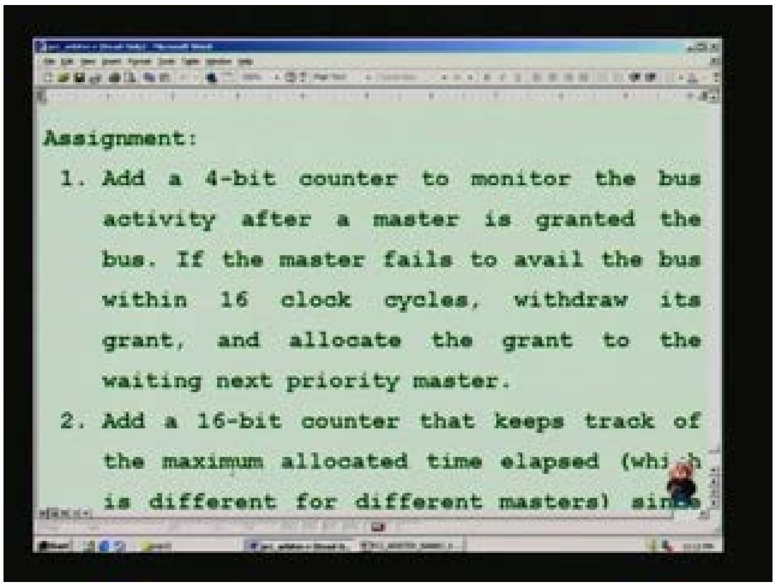
(Refer Slide Time: 48:56)



```
E:/Xilinx.
Opened constraints file pci_arbiter.pcf.

Mon Aug 04 15:36:32 2003

Running DRC.
DRC detected 0 errors and 0 warnings.
Creating bit map...
Saving bit stream in "pci_arbiter.bit".
Creating bit mask...
Saving mask bit stream in "pci_arbiter.msk".
Bitstream generation is complete.
```

The total design is when you make the entire codec and put everything in the same chip or multiple chips. So, that will be the true picture in which case this may come down crashing to anywhere from 50 megahertz to 100 megahertz. Finally, it will generate this bit stream and this is what we will be using for downloading the hardware or few are in open core - IP core supplier will be supplying only this along with the documentation.

(Refer Slide Time: 19:17)



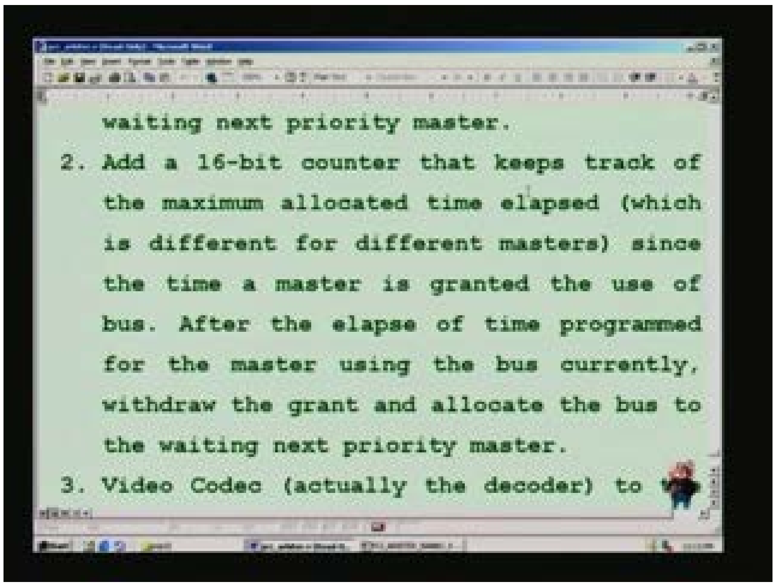
Assignment:

1. Add a 4-bit counter to monitor the bus activity after a master is granted the bus. If the master fails to avail the bus within 16 clock cycles, withdraw its grant, and allocate the grant to the waiting next priority master.
2. Add a 16-bit counter that keeps track of the maximum allocated time elapsed (which is different for different masters) since

So, finally we have some assignments for you. What we have done is I will just read out these assignments. Add a 4 bit counter to monitor the bus activity after a master is granted the bus. So, if the master fails to avail the bus within 16 clock cycles withdraw its grant and allocate the grant to the waiting next priority master. This means, you have granted to a particular master which has requested and even after 16 clock cycles it has not made use of the PCA bus.

So, what should we do? We should remove the grant for this and without even warning because this is the penalty made thing agreed protocol for all the masters. So, you include a 4 bit counter in your design; so that you count 16 clock pulses after it is granted and if no activities found on the bus which will be reported by another signal which we press you it is available to you and then process this.

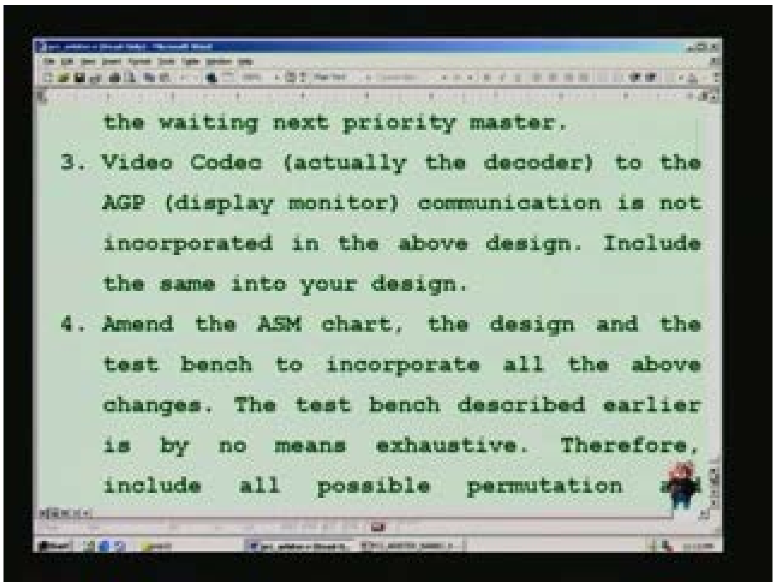
(Refer Slide Time: 50:20)


waiting next priority master.

2. Add a 16-bit counter that keeps track of the maximum allocated time elapsed (which is different for different masters) since the time a master is granted the use of bus. After the elapse of time programmed for the master using the bus currently, withdraw the grant and allocate the bus to the waiting next priority master.

3. Video Codec (actually the decoder) to

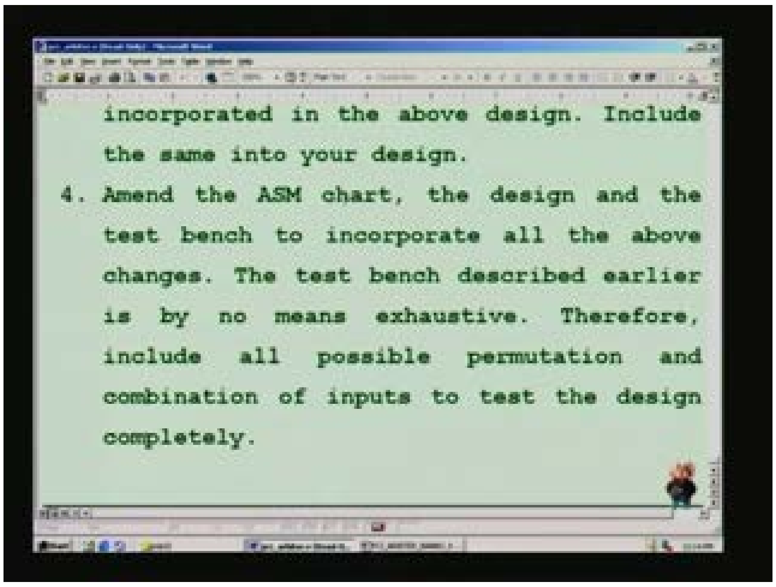
(Refer Slide Time: 50:50)


the waiting next priority master.

3. Video Codec (actually the decoder) to the AGP (display monitor) communication is not incorporated in the above design. Include the same into your design.

4. Amend the ASM chart, the design and the test bench to incorporate all the above changes. The test bench described earlier is by no means exhaustive. Therefore, include all possible permutation

(Refer Slide Time: 51:21)


incorporated in the above design. Include the same into your design.

4. Amend the ASM chart, the design and the test bench to incorporate all the above changes. The test bench described earlier is by no means exhaustive. Therefore, include all possible permutation and combination of inputs to test the design completely.

Also, add a 16 bit counter that keeps track of the maximum allocated time elapsed which is different for different masters. So, if the time a master is granted use of a bus, so once master has taken the bus and started using, there is no guarantee that it will surrender the bus after elapse of its time. So, you need to keep track of that also for which you need one more counter and design this as well after the elapse of time. Program for a master using a bus currently; withdraw the grant and allocate the bus to the waiting next priority master. And, here in design, we consider only the encoder portion for as far as bus grant is concerned, I have not taken into account the decoder aspect and video codec; actually, decoder to the AGP this is for display monitor and a motion picture is displayed on the monitor using this. AGP communication is not incorporated in the above design that is what is said. So, you have to include this into your design; that means, you may have add one more bus request signal and in order to do this and naturally this calls for amendment of the ASM chart; the design as well as the test bench to incorporate all the above changes.

The test bench described earlier is by known is exhaustive because, we have a 4 request. So, the total combination will be 16. Whereas, we have seen the test batch we have hardly used some 6 or 7 types of test. So, you look into that aspect and whatever is not covered in the test you make it full fledged. Therefore, include all possible permutations and combinations of inputs to test the design completely and I hope, you will make a sincere attempt to solve these problems. If you have any question you can ask.

Thank you very much.