Digital VLSI System Design

Prof. Dr. S. Ramachandran

Department of Electrical Engineering

Indian Institute of Technology, Madras

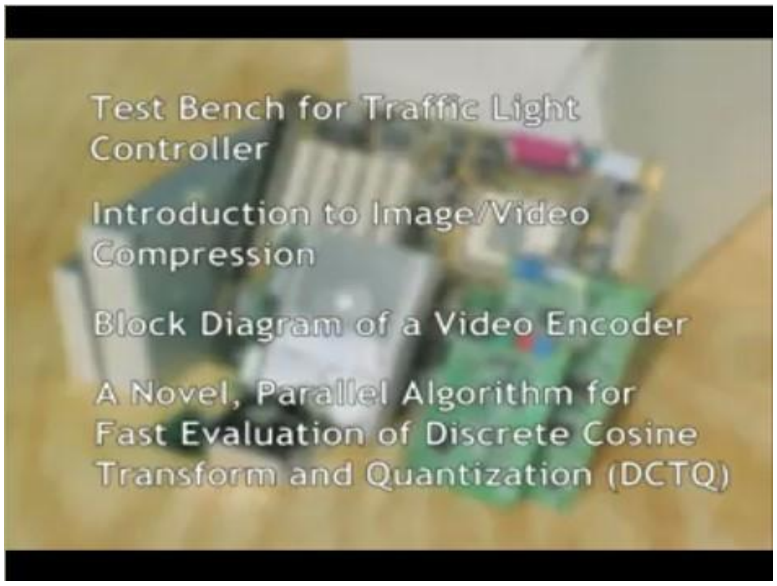
Lecture – 44

System Design Examples

(Continued)

Contents of Lecture 44:

(Refer Slide Time: 01:33)



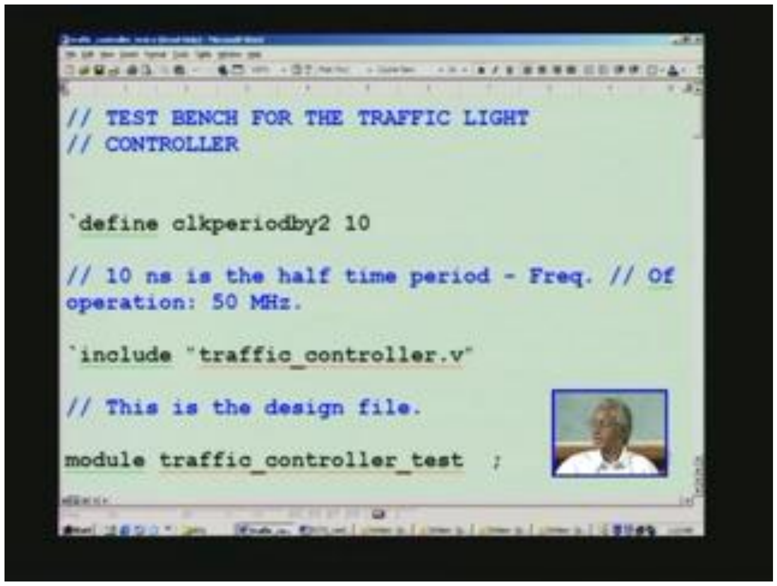
Test Bench for Traffic Light Controller

Introduction to Image/Video Compression

Block Diagram of a Video Encoder

A Novel, Parallel Algorithm for Fast Evaluation of Discrete Cosine Transform and Quantization (DCTQ)

(Refer Slide Time: 02:08)

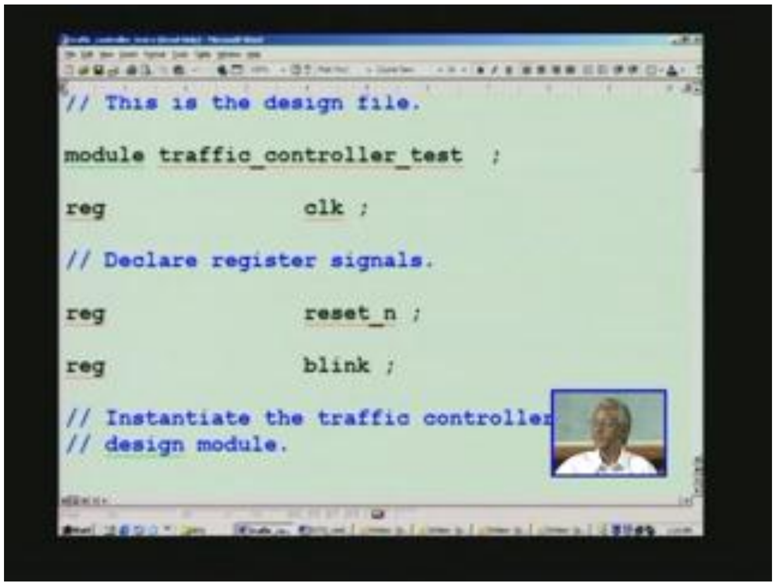


```
// TEST BENCH FOR THE TRAFFIC LIGHT
// CONTROLLER


`define clkperiodby2 10

// 10 ns is the half time period - Freq. // Of
operation: 50 MHz.

`include "traffic_controller.v"

// This is the design file.

module traffic_controller_test  ;
```

We were looking at one of the design applications, the traffic light controller for which we have seen design as well as the waveform for analysis. We found that the waveform confirms to the design we have actually made. We will now see the test bench for the same. The test bench goes something like this. This is the title for the traffic light controller test bench. As usual, we have a define statement which defines the clock for 50 mega hertz. We put 10 in this variable here, that being the half time period.

We include the actual design of the traffic controller here by the statement include this is the design file. This is the module we declare here. This is the traffic controller test bench. This is the module, which is the current file. We declare all the inputs as reg namely clock, reset underscore n, then blink. This is to blink the yellow lights beyond the traffic hours.

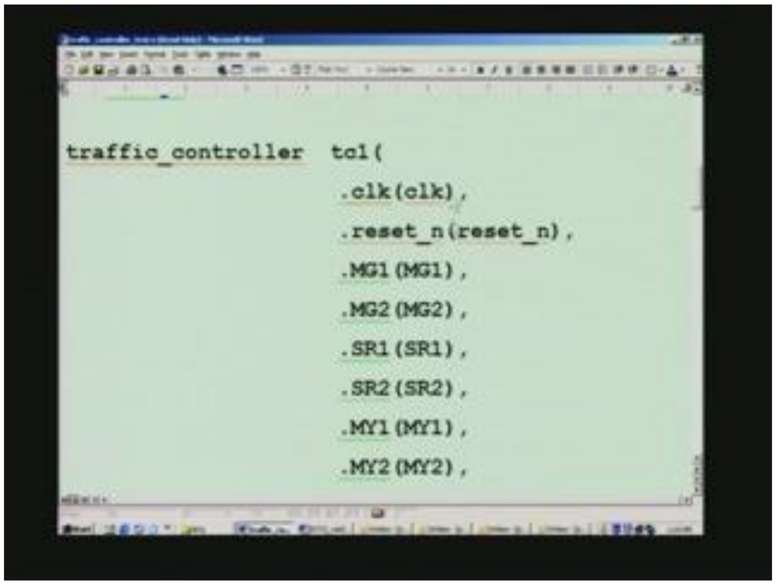
(Refer Slide Time 03:03)



```
// This is the design file.

module traffic_controller_test  ;

reg                clk ;

// Declare register signals.

reg                reset_n ;

reg                blink ;

// Instantiate the traffic controller
// design module.
```

We need to instantiate the traffic controller design module. That is being done here. The only instantiate is this tc1. This is similar to u1, u2 in the chip that you really design on a board. You need to declare ports by name. These are all the inputs we have already seen. These are all the output lamps, 16 in number here, starting with main green, side red, and so on. Y is for yellow.

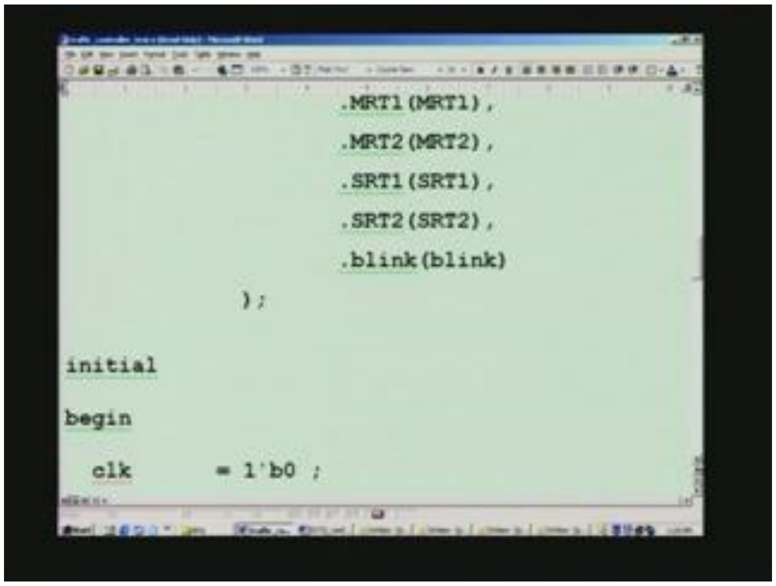
(Refer Slide Time: 03:30)



```
traffic_controller  tc1(
                     .clk(clk),
                     .reset_n(reset_n),
                     .MG1(MG1),
                     .MG2(MG2),
                     .SR1(SR1),
                     .SR2(SR2),
                     .MY1(MY1),
                     .MY2(MY2),
```

We also have right turn for both main as well as side and they are all green lights. We have a blink control. Beyond traffic normal traffic hours, we switch on this control. This is done either manually, or automatically using a timer, which can also be designed into this very traffic controller design that we had done before.

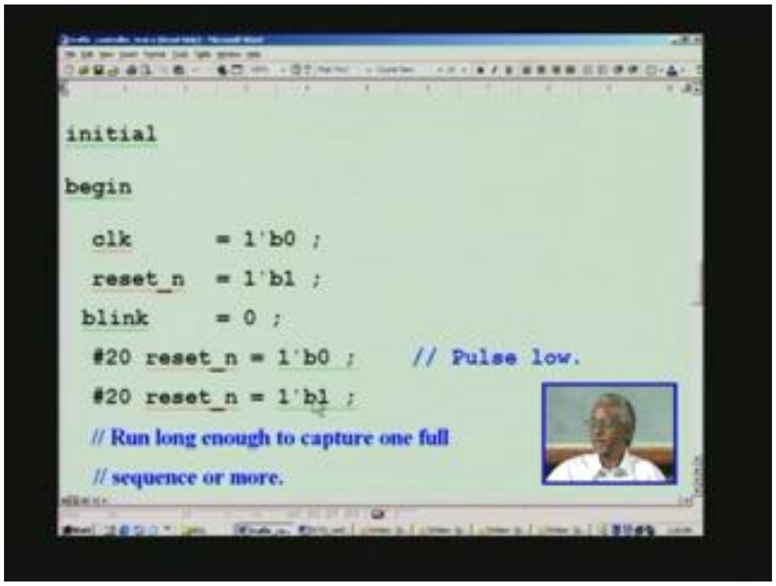
(Refer Slide Time: 04:06)



```
                        .MRT1(MRT1),
                        .MRT2(MRT2),
                        .SRT1(SRT1),
                        .SRT2(SRT2),
                        .blink(blink)
                );

initial

begin

  clk       = 1'b0 ;
```

We start the initial block right here in order that we may initialize all these input signals namely clock, reset n, and blink, all of them 0 except for the reset n because this is active low.

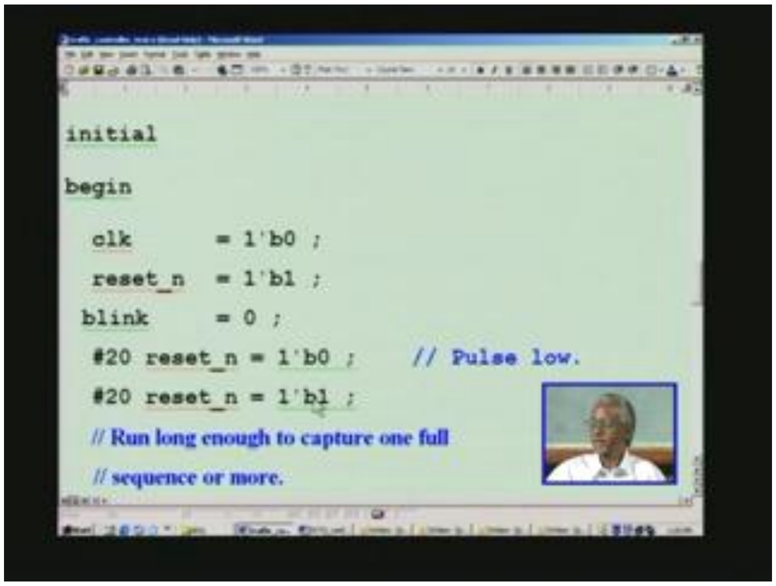
(Refer Slide Time: 04:16)



```
initial
begin
  clk      = 1'b0 ;
  reset_n  = 1'b1 ;
 blink     = 0 ;
  #20 reset_n = 1'b0 ;    // Pulse low.
  #20 reset_n = 1'b1 ;
  // Run long enough to capture one full
  // sequence or more.
```

We pulse this reset here for 20 nano seconds and, once again, we restore so that normal operation starts right here. We allow enough time for it go through at least 1 full sequence. I think there were some 8 sequences, which we have seen already earlier on a power point presentation.

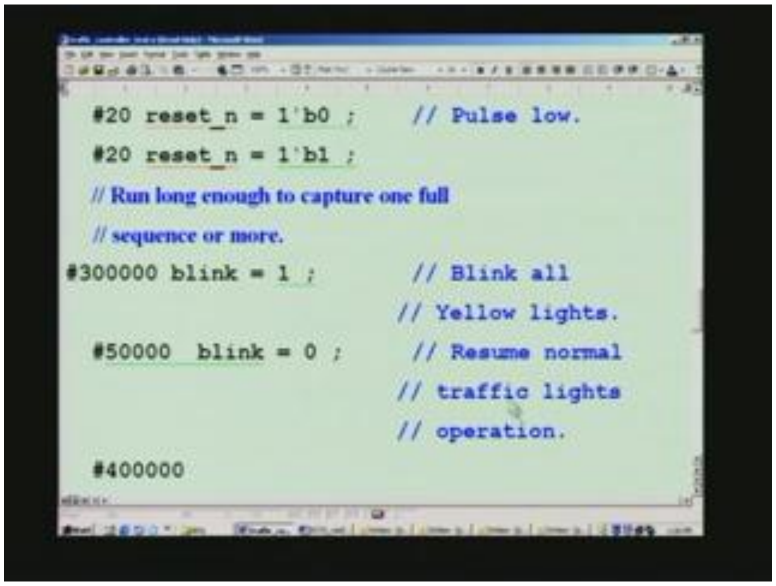
(Refer Slide Time: 04:40)



```
initial
begin
  clk      = 1'b0 ;
  reset_n  = 1'b1 ;
 blink     = 0 ;
  #20 reset_n = 1'b0 ;    // Pulse low.
  #20 reset_n = 1'b1 ;
  // Run long enough to capture one full
  // sequence or more.
```

It may be one sequence or more, for which we need at least these many nano seconds, and beyond which we will activate the blink needed in order to check whether all lights are blinking. In fact we have seen the waveform and confirmed earlier that it is really working. Once this is done, after a while we will switch off blink so as to resume the normal traffic light operation.

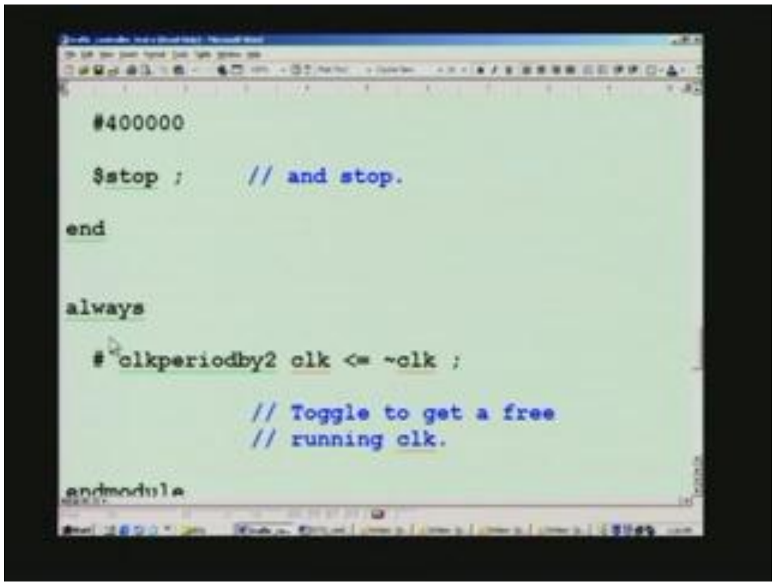
(Refer Slide Time: 05:04)



```
  #20 reset_n = 1'b0 ;     // Pulse low.
  #20 reset_n = 1'b1 ;
  // Run long enough to capture one full
  // sequence or more.
#300000 blink = 1 ;        // Blink all
                          // Yellow lights.
  #50000  blink = 0 ;      // Resume normal
                          // traffic lights
                          // operation.
  #400000
```

After some gap, it need not be such a huge value here, it can be even just 40 nano seconds, and then we stop here.

(Refer Slide Time: 05:29)

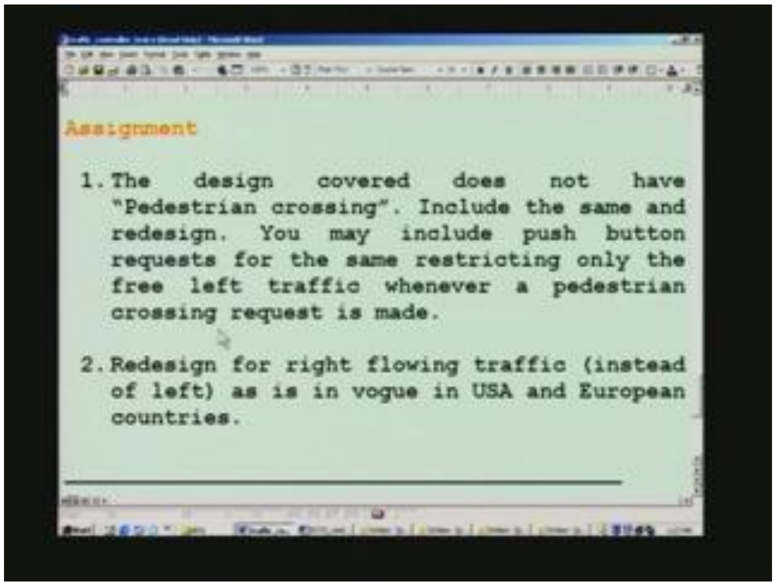


```
  #400000

  $stop ;     // and stop.

end


always

  #`clkperiodby2 clk <= ~clk ;

                // Toggle to get a free
                // running clk.

endmodule
```

The test bench for this is very simple and straightforward, which we have already seen I think. We activate, run the clock, by this statement. This is programmed for 10, right at the beginning of the code. This is to toggle the variable called clock. Every 10 nano seconds it will toggle, which means 20 nano seconds will be the time period and hence 50 mega hertz, which we have already listed earlier. This completes the test bench.

Before we wind up, we have, as usual, an assignment for you. I will read the assignment and explain what it is. The design covered does not have pedestrian crossing and we include the same and redesign. You may include push button requests for the same restricting only the free left traffic whenever a pedestrian crossing request is made. What we had earlier was the main flow as well as side flow straight traffic would imply that pedestrians can cross at the other road. For example, if it is main straight and there is no traffic on the side road therefore, it can be crossed. Pedestrian can cross here and even here. The complexity is that we have a free left traffic. That will be a hindrance for the pedestrian. So we will provide one more push button switch for each of the four corners and then allow the pedestrian to cross based upon that, in which case whenever a pedestrian crossing push button is pressed, that corresponding traffic free left will be suspended for the time being and for the duration of the programmed time. You have to add that in this assignment. That is what this is.

(Refer Slide Time: 06:00)



Assignment

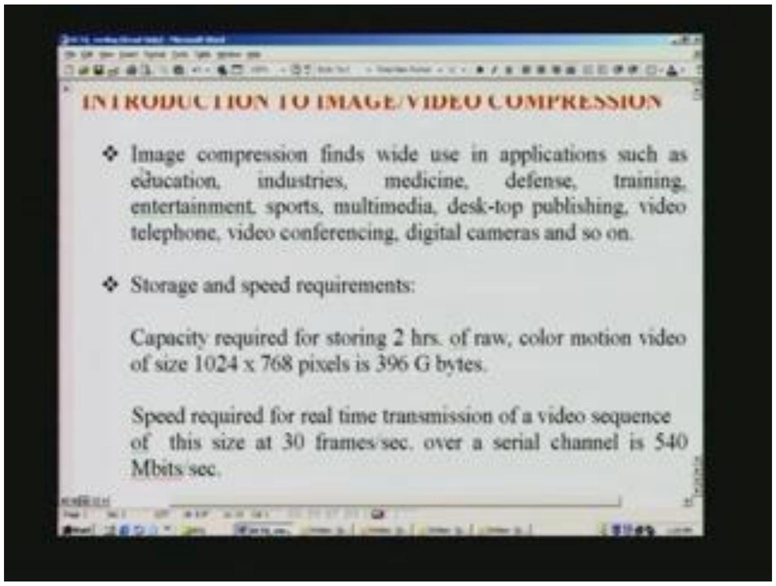
1. The design covered does not have "Pedestrian crossing". Include the same and redesign. You may include push button requests for the same restricting only the free left traffic whenever a pedestrian crossing request is made.

2. Redesign for right flowing traffic (instead of left) as is in vogue in USA and European countries.

The second assignment is, we have been doing for places like India, say Asia and closer countries, and that is left flowing traffic. On the other hand, in the USA and European countries, it is right flowing traffic. Redesign the whole thing that we have done including the pedestrian crossing and make it cater to right flowing traffic. This is the second assignment.

Now we will look into the design of DCTQ and we will process an image such as this. Before that, let us have an introduction to image and video compression. I will go through some of these points first and then explain later on. Image compression finds wide use in applications such as education, industries, medicine, defense, training, entertainment, sports, multimedia, desk-top publishing, video telephone, video conferencing, digital cameras and so on and so forth. That is an immense scope for applications, especially in image as well as video. The differentiation between image and video is that image always refers to a still picture, whereas video is a series of such pictures, one different from the other only by a slight amount.

(Refer Slide Time: 82:12)



INTRODUCTION TO IMAGE/VIDEO COMPRESSION

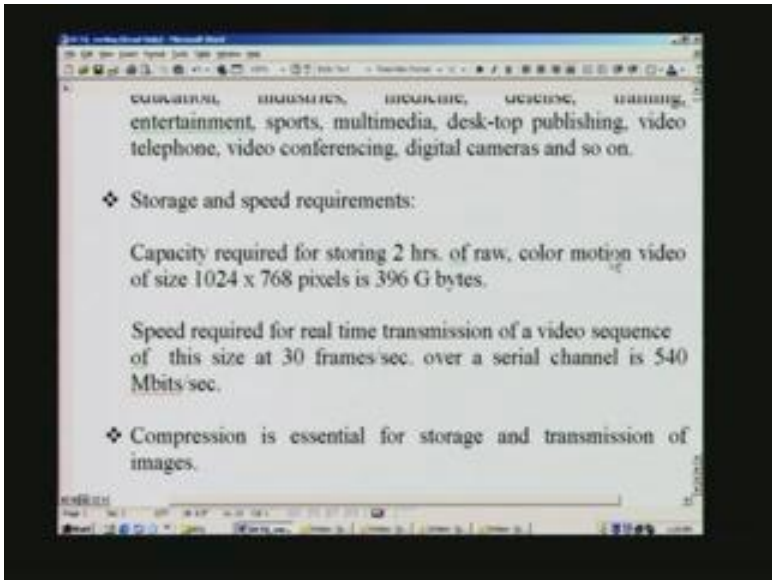
- Image compression finds wide use in applications such as education, industries, medicine, defense, training, entertainment, sports, multimedia, desk-top publishing, video telephone, video conferencing, digital cameras and so on.
- Storage and speed requirements:

  Capacity required for storing 2 hrs. of raw, color motion video of size 1024 x 768 pixels is 396 G bytes.

  Speed required for real time transmission of a video sequence of this size at 30 frames/sec. over a serial channel is 540 Mbits/sec.

Depending upon the actual motion involved in the video picture, and as such for raw video, let us say for 2 hours of color motion picture of very high resolution, say 1024 into 768 pixels, you need a very high storage of the order of 396 giga bytes. We also have much higher bandwidth requirements. Ultimately, the point here is that we need to compress and after compression we need to transmit over the serial channel.

(Refer Slide Time: 08:55)



education, industries, medicine, defense, training, entertainment, sports, multimedia, desk-top publishing, video telephone, video conferencing, digital cameras and so on.

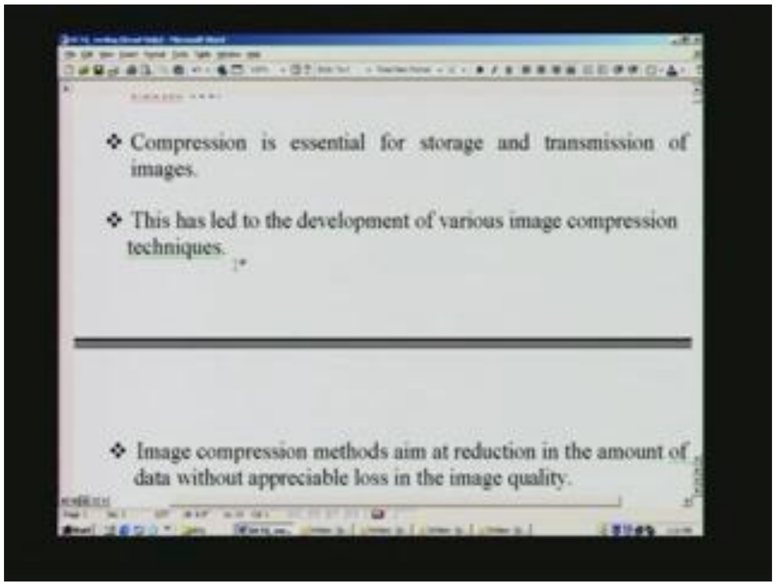
- Storage and speed requirements:

  Capacity required for storing 2 hrs. of raw, color motion video of size 1024 x 768 pixels is 396 G bytes.

  Speed required for real time transmission of a video sequence of this size at 30 frames/sec. over a serial channel is 540 Mbits/sec.
- Compression is essential for storage and transmission of images.

If you are to transmit this same thing without compression, namely the raw signal, you require a very high speed channel for that. Speed required for real time transmission of a video sequence of this size at 30 frames per second is very high. These 30 frames per second hold good in USA and other regions whereas in India and other places it will be 25 frames per second. If you are to transmit this over a serial channel it requires a stupendous bandwidth say 540 mega bits per second. This will restrict; put together with the high storage this will be a big constraint, hence the need for compression here. Compression is, so to say, inevitable in order that we may contain the storage and transmission in manageable limits. This is a hot field even as of today and has led to a number of developments among researches all over the globe for various image compression techniques, say development of new algorithms as well as implementation as VLSI.

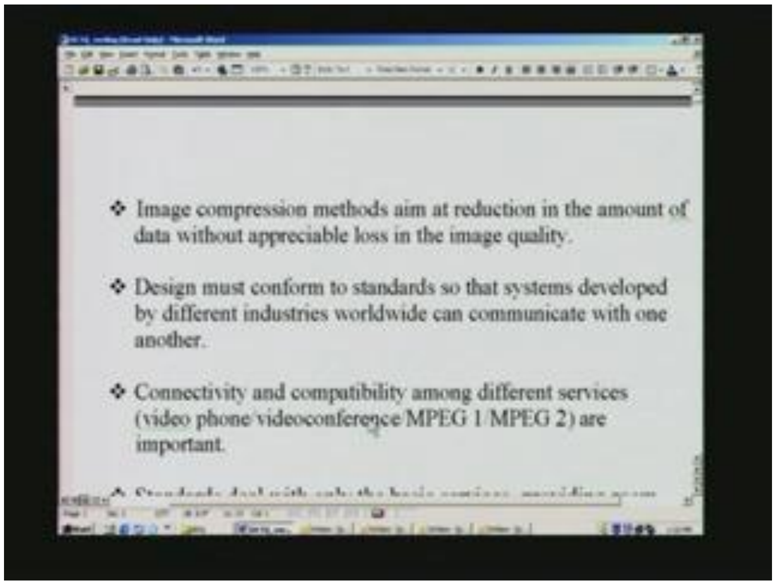
(Refer Slide Time: 10:34)



- ❖ Compression is essential for storage and transmission of images.
- ❖ This has led to the development of various image compression techniques.
- ❖ Image compression methods aim at reduction in the amount of data without appreciable loss in the image quality.

All these methods aim of image compression aim at reduction in the amount of data, without appreciable loss in the image quality. We need to achieve as much compression as possible and so that we can limit the storage as well as bandwidth requirements but, at the same time, you should not sacrifice too much of image quality. In fact any compression would involve sacrificing quality. The question is how much we can sacrifice. These designs will have to confirm to standards; otherwise there is no communication between one design and the other. What is also important is the connectivity and compatibility among different services, for example, video phone to video conference or to MPEG 2 and so on. We will see a little more in detail next. Design must confirm to standards so that systems developed by different industries worldwide can communicate with one another. Connectivity and compatibility among different services, video phone, video conference, MPEG 1, MPEG 2 are important as well.

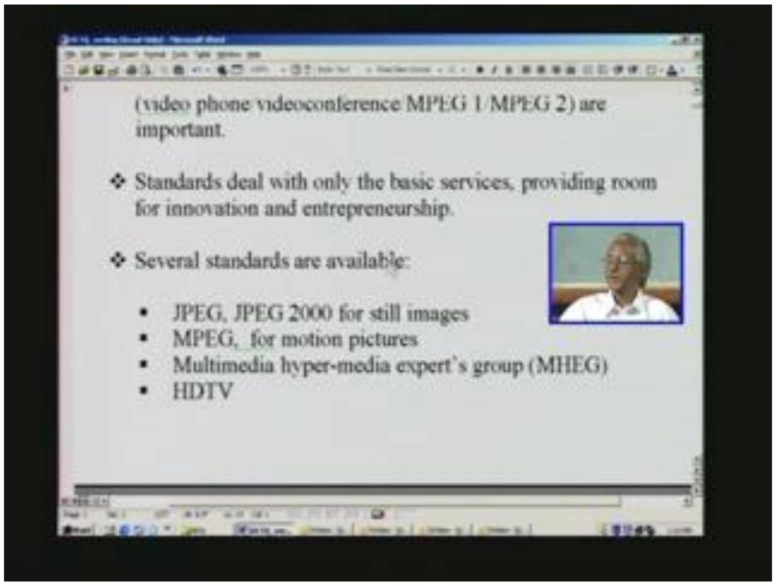
(Refer Slide Time: 11:30)



- ❖ Image compression methods aim at reduction in the amount of data without appreciable loss in the image quality.
- ❖ Design must conform to standards so that systems developed by different industries worldwide can communicate with one another.
- ❖ Connectivity and compatibility among different services (video phone/videoconference/MPEG 1/MPEG 2) are important.

All these standards we have named deal with only the basic services, providing lots of room for innovation in the form of developing new algorithms or the type of implementation, whether it is FPGA implementation or ASIC or any other implementation. It does not cripple your innovation or entrepreneurship.

(Refer Slide Time: 11:53)


(video phone videoconference MPEG 1/MPEG 2) are important.

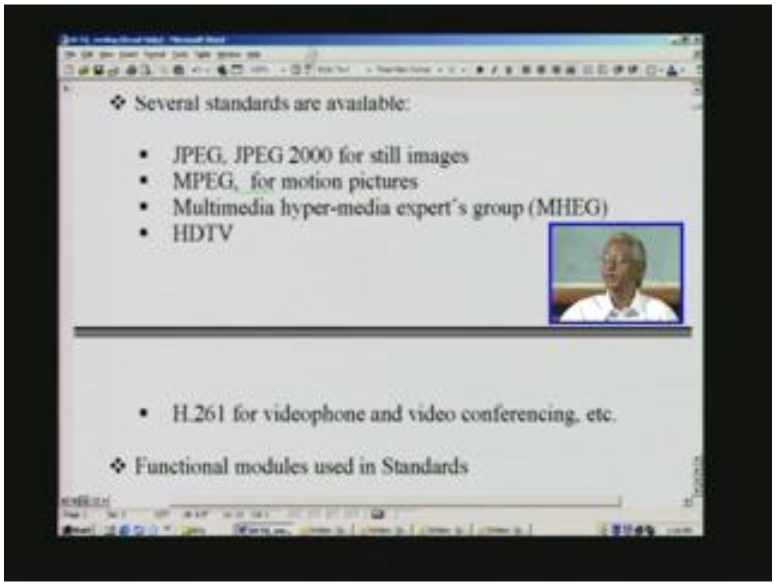
- Standards deal with only the basic services, providing room for innovation and entrepreneurship.
- Several standards are available:
  - JPEG, JPEG 2000 for still images
  - MPEG, for motion pictures
  - Multimedia hyper-media expert's group (MHEG)
  - HDTV

Several such standards are available and we will look into the basic features of some of these standards. For example, JPEG stands for joint photographic experts group. This is based on what is called discrete cosine transform abbreviated as DCT and this is based on a transform, in the sense that an image is taken and transformed to a frequency domain. That is what it implies here.

So also is JPEG 2000. In JPEG 2000, DWT is used instead of DCT. DWT stands for discrete wavelet transform. This is of a recent origin. Both are for processing still images.

In JPEG, if you compress too much, you have what are known as blocking artifacts. In JPEG 2000 these blocking artifacts do not manifest. In a way, regarding quality, JPEG 2000 is better than JPEG but you have to pay a price for it. This is very computationally intensive when compared to the DCT. Therefore you cannot achieve real time speeds for high resolution pictures of the order that we have already seen before, which can be in this design. We will see that very high quality, much greater than the picture size that we have mentioned before, can be achieved by using JPEG or for that matter MPEG1 and 2. MPEG stands for motion picture.

(Refer Slide Time: 12:52)



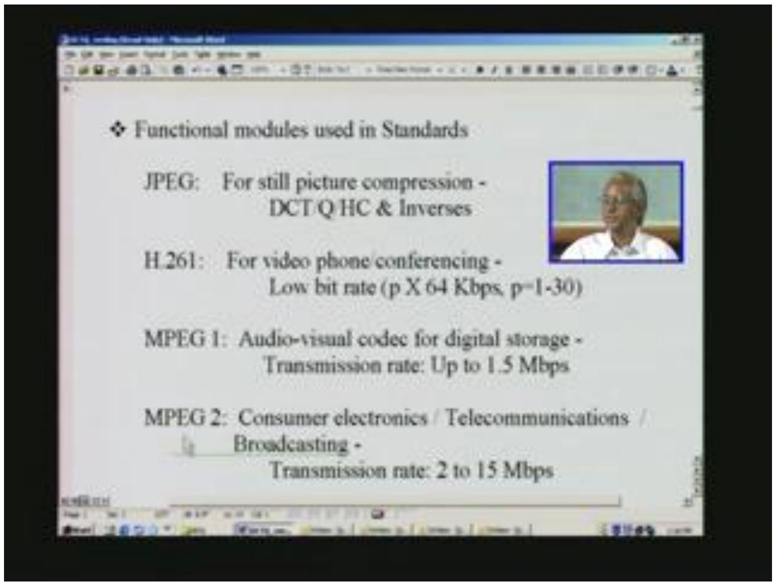
❖ Several standards are available:

- JPEG, JPEG 2000 for still images
- MPEG, for motion pictures
- Multimedia hyper-media expert's group (MHEG)
- HDTV
- H.261 for videophone and video conferencing, etc.

❖ Functional modules used in Standards

This is another standard. 1, 2 and 4, 7 are all there for multimedia that 4, 7 etc. can be used. You have one more for multimedia. It is called MHEG and there is also HDTV, which is high definition TV. They are all motion pictures. You also have one more standard for video phone and video conferencing that is H 261. I think the earlier version was 263. I suppose this is better than 263. This covers quite a lot of range right from still picture to motion picture and video phone, video conferencing to broadcast quality video, which we will be listing here.

(Refer Slide Time: 14:42)


❖ Functional modules used in Standards

JPEG: For still picture compression - DCT/Q/HC & Inverses

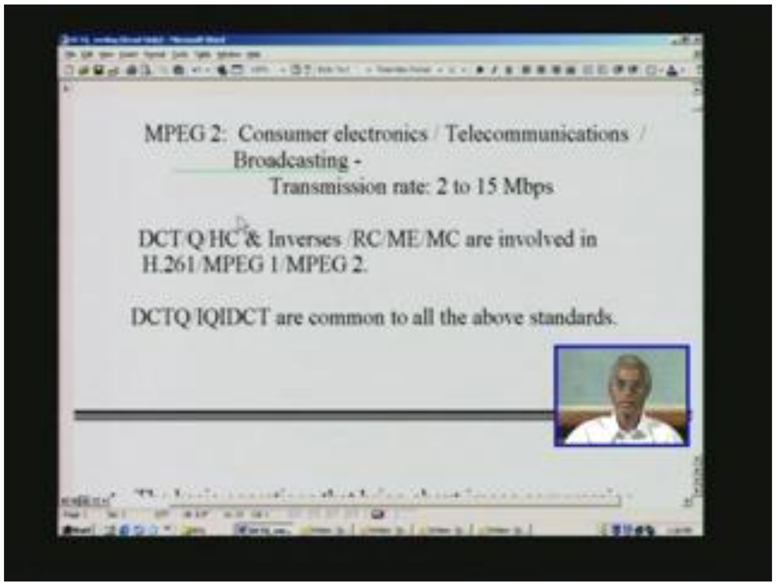
H.261: For video phone/conferencing - Low bit rate (p X 64 Kbps, p=1-30)

MPEG 1: Audio-visual codec for digital storage - Transmission rate: Up to 1.5 Mbps

MPEG 2: Consumer electronics / Telecommunications / Broadcasting - Transmission rate: 2 to 15 Mbps

These are the functional modules used in these standards, for example, JPEG. We will not cover JPEG 2000 because that is DWT based. We will concentrate on DCT based. All the standards listed are basically the same. DCT will be used. This is for still picture compression. The module set we will use and design will be the DCT then quantization. Next is what is called Huffman coding. It also goes by the name VLC, which is an abbreviation for variable length coding. This will be at the encoder stage wherein you take a raw image and compress using this technique and then send it over a channel. At the receiving end, which is called the decoder, we have to do inverse operations of this. For example, VLD will have to be applied here first followed by inverse quantization and followed by inverse DCT. These are all for motion picture. For video phone, video conferencing, the total transmitted bit rate is rather low and depending upon p value you can have right up to 3030 into 64 Kbps. This is kilo bits per second because this was serial transmission. We also have MPEG1 for digital storage, namely on the tape and any other storage media, and it can go right up to 1.5 million bits per second. MPEG 2 gives a broadcasting quality that you can use even for TV and transmission rate is quite high, 15 mbps. Here, what is said is DCT quantization, Huffman coding and their inverses as well as what is called rate control. Rate control is part of the VLC or VLD in the sense that depending upon the picture you may be compressing. We cannot predict how much will be the utility as far as the transmission channel is concerned.

(Refer Slide Time: 16:39)

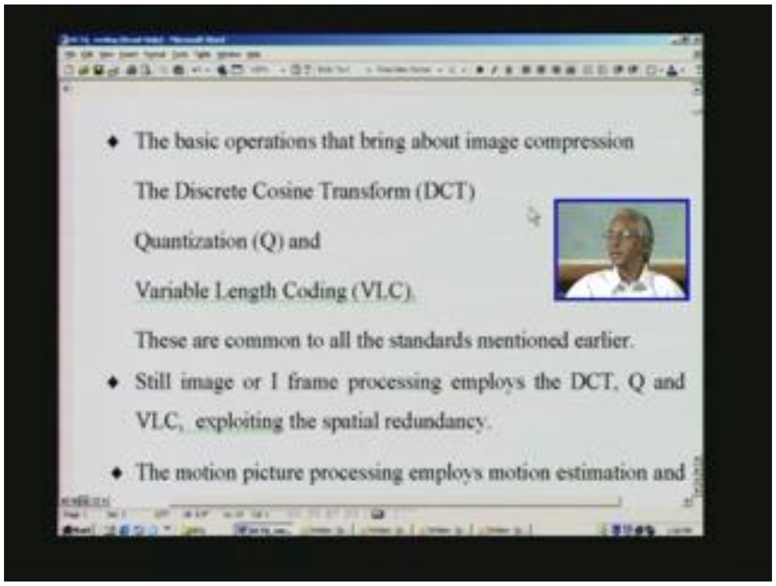

MPEG 2: Consumer electronics / Telecommunications / Broadcasting -
Transmission rate: 2 to 15 Mbps

DCT/Q/HC & Inverses /RC/ME/MC are involved in H.261/MPEG 1/MPEG 2.

DCTQ/IQIDCT are common to all the above standards.

It may not be utilized for a brief while. If you look at the bit stream coming out, it will not be uniform. In order to make it constant and uniform, you need to place what is called rate control, which is part of this VLC. We are not going to discuss VLC. What we are going to do is only DCTQ to start with. IQIDCT has also been done but we will use only the end result because design is quite involved and we will have to restrict it. Even this goes to some 124,000 gates. Then comes what is called motion estimation and motion compensation. A new algorithm has also been developed by the speaker, just as parallel Novel algorithm has been developed for DCTQ, and which we will be presenting a little later. We will not be covering this motion estimation beyond the scope of the present lecture series. All these things are involved in motion picture. For example, 261 for video telephone conferencing then for storage as well as for broadcast quality compression. DCTQ, IQIDCT are common to these standards and that is the reason I have taken DCTQ to start with. This also will be used although it will not be explained. It will not be explained also because it is exact counterpart of DCTQ; it employs the very same algorithm used for a DCTQ. In all this, the basic operations that bring about compression are these. For example, discrete cosine transform prepares the ground for compression; it does not bring about the compression all by itself. It only prepares the ground for that. We apply a block of image and transform it to frequency coefficients. Outcome some 64 coefficients if you process a block of size 64 pixels

(Refer Slide Time: 18:42)

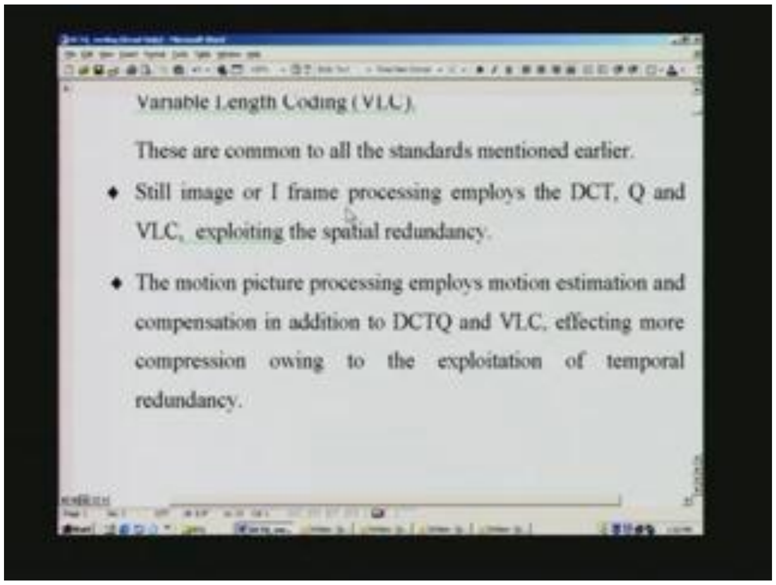


- The basic operations that bring about image compression
  The Discrete Cosine Transform (DCT)
  Quantization (Q) and
  Variable Length Coding (VLC).
  These are common to all the standards mentioned earlier.
- Still image or I frame processing employs the DCT, Q and VLC, exploiting the spatial redundancy.
- The motion picture processing employs motion estimation and

These coefficients pack varying energy of the image in different coefficients. For example, the very first coefficient stores 8 times the average of the image information, and the rest of the coefficients, namely 63 coefficients, store only the balance. This is followed by quantization. In quantization, we make an attempt to make this DCT coefficient 0, make as many coefficients as possible 0. The purpose of making it 0 is we do not have to code this quantized value. This coding is done in the Huffman coding or variable length coding abbreviated as VLC. Frequently occurring intensity value of any signal will get reflected as a discrete coefficient. Coefficient DCTQ coefficient value and that is taken as the input that DCTQ output is parallel say For example, it may be a 9 bit and that will be the input for the variable length coding and out comes just a single bit and it is a bit stream. This bit stream is a compressed stream, and when compared to the original input you will have a factor of 10 to 50 compression depending upon the algorithm we adopt and the techniques we adopt.

The architecture also applies a great role in that. The purpose of applying this variable length coding is to bring about compression. The compression effected in this stage is even more when compared to the quantization stage. This is bringing about compression in the sense that it succeeds in transforming most of the coefficients to 0, which need not be broadcast and need not even be coded. All these we have seen that bring about compression are common to all the standards mentioned earlier, namely JPEG through MPEG 2. Still image or I frame, any still image that you have is processed block by block using that DCTQ, which we will be seeing in video encoder block diagram shortly. Out comes, through the VLC, your bit stream. Before that, we can even perform the inverse operation namely IQIDCT and then form a separate frame that is known as I frame. This is useful for motion picture processing which I will explain a little later.

(Refer Slide Time: 21:23)



Variable Length Coding (VLC).

These are common to all the standards mentioned earlier.

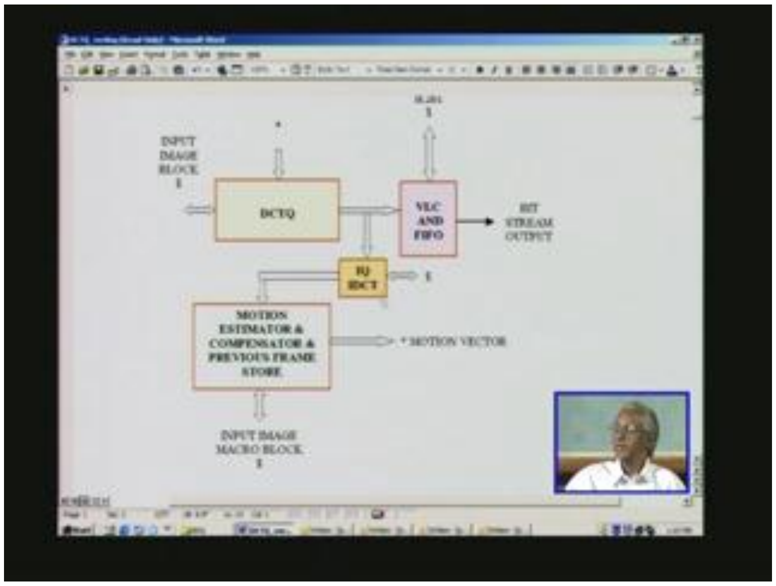
- Still image or I frame processing employs the DCT, Q and VLC, exploiting the spatial redundancy.
- The motion picture processing employs motion estimation and compensation in addition to DCTQ and VLC, effecting more compression owing to the exploitation of temporal redundancy.

This I frame employs DCT quantization as well as VLC and it exploits what is known as spatial redundancy. If you take a picture, as we have seen before, namely the parrot, what we have is a two-dimensional picture. The two-dimensions are width wise and height wise. That is nothing other than a space, and whatever redundancy is there in the intensity of information as far as adjoining pixels are concerned, they are removed. Thereby we bring about compression. It this DCT Q VLC all put together that accomplishes this task of spotting out redundancy in the spatial domain of the picture and then bringing about the removal of the redundancy and retaining only that which is required as far as the information is concerned.

In contrast with this I frame, we have what is known as p frame called predicted frames, or b frames, which are bi-directionally, predicted frames in motion pictures. In motion pictures, you would have seen the film rolls. If you inspect the roll, you will see that there is a between 2 sequences. There is only a small motion involved in adjoining frames. What is the need for coding it? Where there is no motion involved, we can, as usual, retain the same frame, I mean the previous frame. That is the basic idea in this motion picture processing. It is also the idea in motion estimation algorithm, motion estimation compensation as well, which we have already mentioned. Wherever there is a motion change, we need to code only there. Coding is obviously application of DCTQ followed by VLC and so on. Motion picture processing employs motion estimation and compensation. In addition to DCTQ and VLC we also need this DCT Q VLC because if you want to find the motion bit in 2 successive frames, you will have to take one block known as macro block and consisting of 16 by 16 pixels, and search in a window that is 4 times that area and in the previous frame so as to look for where the particular macro block has moved with reference to the previous thing. Lots of algorithms are invoked for this. The speaker too has developed a new algorithm and one of the fastest available. We will not be covering those and effecting more compression owing to the exploitation of temporal redundancy.

We speak in terms of frames, that is, frame to frame. It is a difference in time so temporal refers to the time. This DCT Q VLC removes whatever redundant information is there within the image. That is confined to the space, whereas frame to frame we speak in terms of time. That is why it is called temporal redundancy here. We expect much more compression than is possible here in the sense that we need to code only where there is some motion and majority of motion. For example, you are looking right at the image here and there is practically no change in motion so there is no point in coding the same thing, again and again, frame after frame. In such cases even if you see the speaker you will see only the lip movements or head movements, eye movements and so on. What is the need for coding the rest of the picture? That is precisely the background here and thereby you achieve much more compression there. A typical video encoder will be like this and these are all the signals pertaining to the system bus from or to host. We will design ipcores for DCTQ and IQIDCT and so on although we will see only the design DCTQ here. This is the total picture of the video encoder. The block diagram of a video encoder is shown here.

(Refer Slide Time: 26:25)

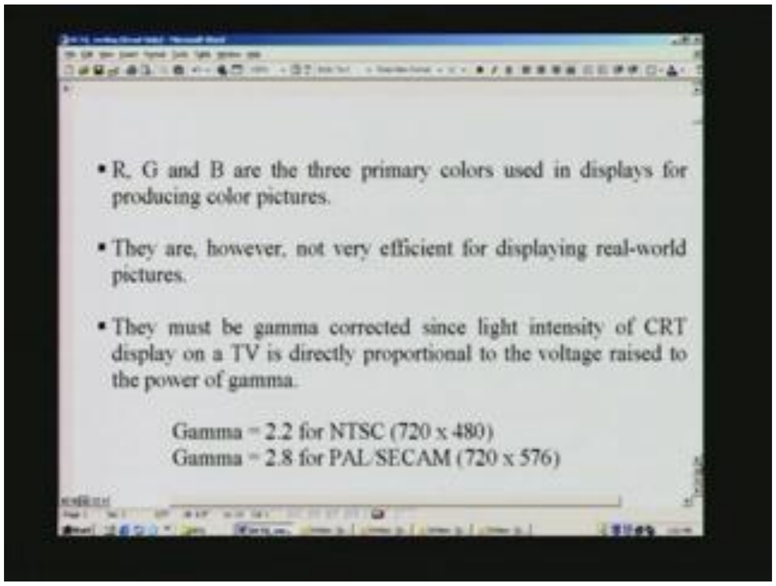


As explained before, we first take one block of image and then apply DCTQ. Output comes as the corresponding 64 coefficients here. If you do the inverse operation, you can recreate the very same image that you have put in. That is what is called reconstructed image that can be stored in RAM. That is what is designated as previous frame store. You need this one because, in order to perform motion estimation, we need the previous frame. Otherwise, we cannot see how to process the motion actually. So we take a macro block, as mentioned earlier, in the current frame and look for the shift in the previous frame and that is what this algorithm does. Finally, it will create what is known as motion vector, which will say it has moved by such and such place.

The x y coordinate will be transmitted, using which we will easily reconstruct that motion at the receiving end. There will be no need for sending the previous frame; we only need to send that particular small macro block where there is a motion change. Regarding this I frame, so we apply block by block outcomes the I frame here, which is stored. It is also transmitted after doing VLC and via first in first out a typically 16 kilo bits depth should be enough for FIFO storage. Out goes to the bit stream. If rate control is in place, then you will have a constant bit stream. This is the video encoder. At the receiving end, this will go right out at the channel as serial channel. Unfortunately, we cannot use the present day lan networks because you cannot process any real time image. Even in a still image, it will be difficult. We need a dedicated channel for that. At the receiving end, you receive this bit stream. There will be FIFO there also.

From that, we do inverse operation, namely VLD, (D stands for the decoder) and get parallel information that is DCTQ equivalent. There will be this element, this processor, at the receiving end. That will be processed and you will get a recovered signal. Ideally that signal will be same as this one. We will have a look at the images that we get out of running our design using both matlab as well as verilog later on. Is this clear to you? This is the encoder; its counterpart is in the decoder. What goes out is a highly compressed stream. For example, it may be whatever the image you compress by a factor of 10 to 50 anywhere and then only send there by reducing the bandwidth requirements as well as storage at the receiving end. As far as color processing is concerned, we have red, green, and blue as primary colors for a display on the TV monitor that you see or on the computer terminal that you have. It is CRT display but they are not efficient in terms of transmitting real world pictures. Processing real world pictures is not that efficient.

(Refer Slide Time: 29:46)


- R, G and B are the three primary colors used in displays for producing color pictures.
- They are, however, not very efficient for displaying real-world pictures.
- They must be gamma corrected since light intensity of CRT display on a TV is directly proportional to the voltage raised to the power of gamma.

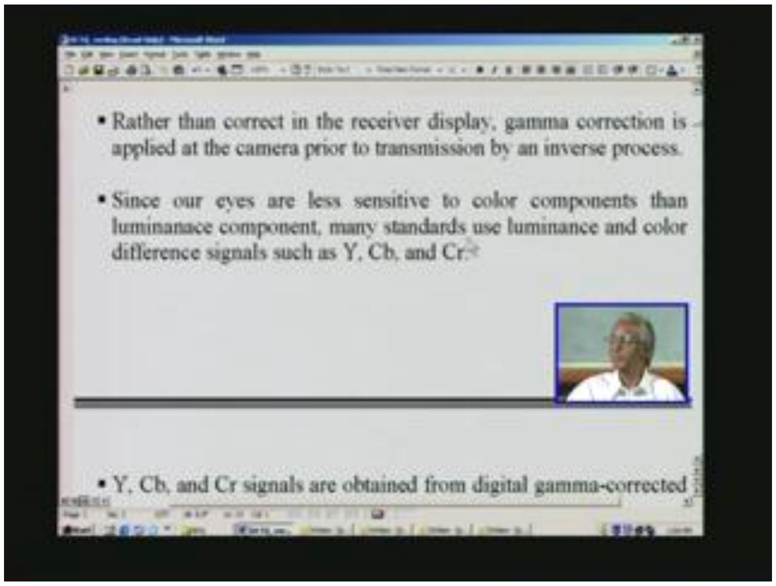
Gamma = 2.2 for NTSC (720 x 480)
Gamma = 2.8 for PAL/SECAM (720 x 576)

We first need to do what is known as a gamma correction. The reason for not being efficient is that in a CRT, electron beams form in the intensity corresponding to the image information that it receives. The intensity is directly proportional to a power such as 2.2.

Say voltage is the signal voltage v to the power of 2.2 will be the response here that will be the intensity of the on the CRT. This is a non linear thing so if you use gamma correction, which is the same as this gamma factor for this signal, such as NTSC signals. This NTSC signal is for USA and PAL/SECAM for Asian and Russian, I suppose. Picture sizes are also given there: 720 by 480 pixels here and 720 by 576 here.

There are what are called interleave. In order to avoid flickering, they have separated entire pixels into odd and even lines for that matter of frames, thereby reducing the flickering. That is called interleave. In CRT monitor etc., what is popular is a non-interleave so that means line after line it is processing all the 720 lines and 470 pixels in each line and so on. For NTSC, we have to achieve 30 frames per second picture rate because they are all concerned about motion pictures. For PAL, it is 25 frames per second. So we say that we need to incorporate gamma correction. If you are to do at the receiving end, we have a TV at every home and they will be too numerous. I think it is better, to save cost for the consumer, to take the complexity right at the camera end. Inverse of this, a factor of 0.45 will come v to the power of 0.45 will be the intensity that needs to be corrected at the camera end before transmission. That is what is mentioned here. As far as the standards are concerned, even this gamma corrected RGB is not adequate. It is not efficient. I will read this out. Since our eyes are less sensitive to color components than luminance component, many standards use luminance and color difference signals such as Y, Cb and Cr.

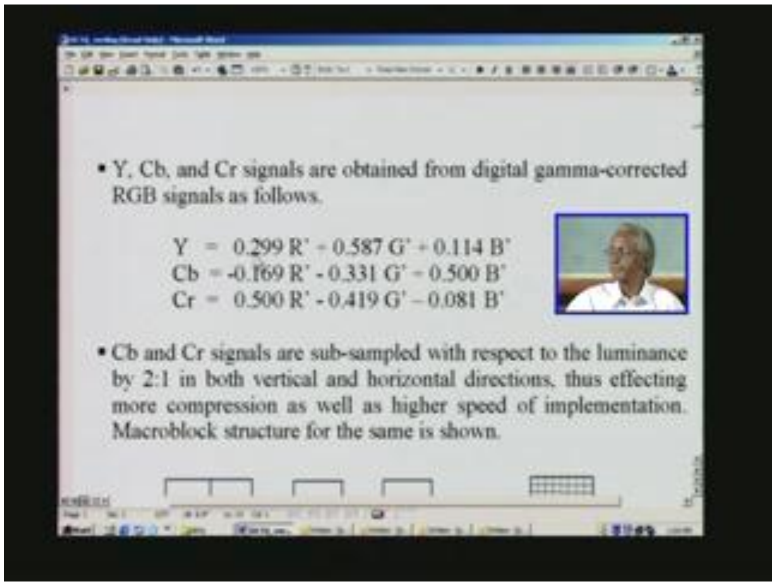
(Refer Slide Time: 32:35)



- Rather than correct in the receiver display, gamma correction is applied at the camera prior to transmission by an inverse process.
- Since our eyes are less sensitive to color components than luminanace component, many standards use luminance and color difference signals such as Y, Cb, and Cr.
- Y, Cb, and Cr signals are obtained from digital gamma-corrected

Y is known as the luminance portion and Cb and Cr are known as the color portion. Do not correlate this one to RGB straightaway. Suppose you take Cb, and say I want a color picture, you may get some outline of picture but not the true color; as also you will not get the monochrome picture. The reason is this. Y, Cb, Cr are governed by these expressions here, which are derived from RGB gamma corrector. Gamma correction is with a prime indicated here. These are all the expressions we need to confirm to. The standards, JPEG through MPEG 2 and even beyond, all demand the use of Y Cb Cr format.

(Refer Slide Time: 33:24)



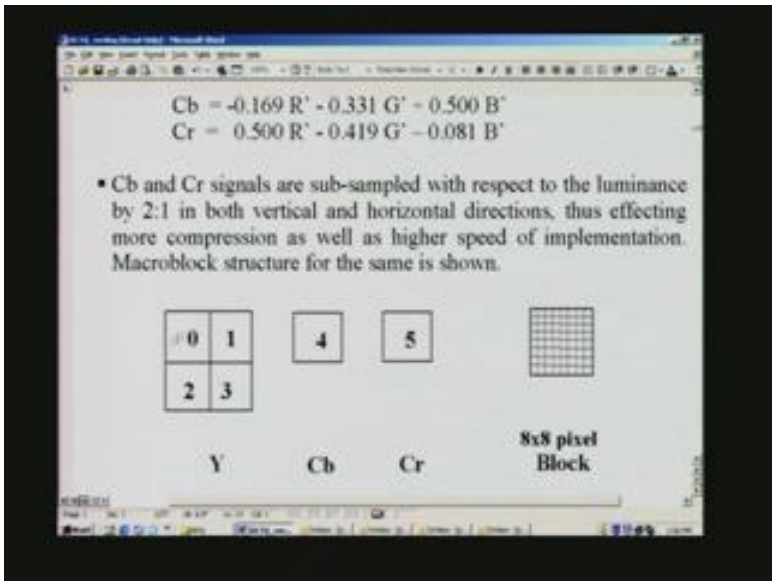
- Y, Cb, and Cr signals are obtained from digital gamma-corrected RGB signals as follows.

$$Y = 0.299 R' + 0.587 G' + 0.114 B'$$
$$Cb = -0.169 R' - 0.331 G' + 0.500 B'$$
$$Cr = 0.500 R' - 0.419 G' - 0.081 B'$$

- Cb and Cr signals are sub-sampled with respect to the luminance by 2:1 in both vertical and horizontal directions, thus effecting more compression as well as higher speed of implementation. Macroblock structure for the same is shown.

I will read out another important thing. Cb and Cr signals are sub-sampled with respect to the luminance by 2 is to 1 in both vertical and horizontal directions, (I will explain this in a minute), thus effecting more compression as well as higher speed of implementation. Macro block structure for the same is shown here. We have seen a picture and the entire picture (the picture that you saw right at the beginning, namely parrot) is actually 1024 by 768 true color. This means that you need 3 bytes for representing a color of Y, Cb, Cr if it is gamma corrector RGB, also the raw format. You need 3 bytes for representing; each byte for each of the color components. If you do that one you will need a macro block in order to process. It consists of 4 blocks. It is only for the sake of convenience that we have divided it as macro block. The standard demands this. Each one is a block. What is a block? A block is nothing but a collection of 64 pixels, arranged as 8 pixel by 8 pixel. We will look at what a pixel is. With respect to the actual image we saw before, this is what is termed as a block and four such blocks would constitute a macro block. That is what is here. Each of them represents a block.

(Refer Slide Time: 33:50)



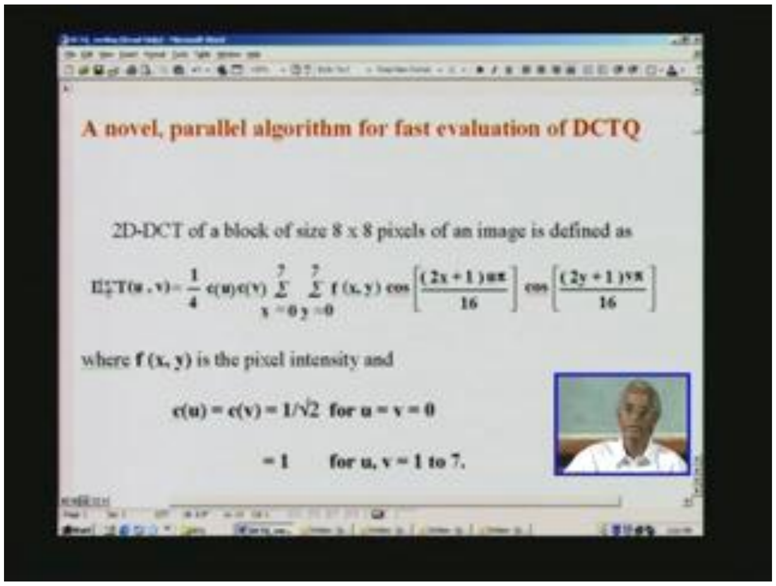
Cb = -0.169 R' - 0.331 G' - 0.500 B'
Cr = 0.500 R' - 0.419 G' - 0.081 B'

- Cb and Cr signals are sub-sampled with respect to the luminance by 2:1 in both vertical and horizontal directions, thus effecting more compression as well as higher speed of implementation. Macroblock structure for the same is shown.

Suppose you want to transmit a full color or process a full color. What you actually need is, instead of 4 blocks here and just 1 block shown here for color component, another 4 here and another 4 here. That would make a total of 12. You have to process 12 blocks. The processing of each block takes the same time. When compared to a monochrome, for which we need only 4, for processing true color we need 3 times of that. If you want 3 times the time you require, it means that the throughput is just 1 by 3 of what you get in a monochrome. In other words, resolution of the picture will get hit so you will have to scale it down by a factor of 3. Are you clear about this? If you process more blocks, it takes more time .What I am trying to say so you not to contain this one a trick played by and recommend by the standard is this. We have also already seen that our eyes are sensitive only for luminance and not for chrominance. A color component can easily fool our eyes by taking only alternate pixel color and processing them. For example, in this block, hoping that you see independent block, take the first pixel then the third pixel then the fifth pixel and arrange them. If you take the total macro block, which contains 16 by 16 pixels, you can extract only 8 by 8 block. That is what is shown. This corresponds to one component, color component Cb, and similarly there is one more component Cr. Cb, Cr are all derived by this expression that we have already seen. Why Cb, Cr? In fact note that all the RGB components are embedded in this. So do not seek for a 1 to 1 correspondence. Are we clear here? What we have here is instead of in this particular recommended by the standard, we need to process only 6 instead of 12, which means that we need to process only half the number of blocks, instead of full color processing. Double the speed can be achieved with this particular recommendation. It is just 1.5 times the monochrome speed.

Now what we will see is that we have developed a Novel parallel algorithm for fast implementation of DCTQ. We need to go into a little theory for a while. Follow this carefully. DCT is defined by this. It is not a very complicated affair. cus, cv are all constant, but it depends upon the spatial coordinate of the image. We take an image. The image is f x y. x y are the coordinates. We have already seen what a block of image is. Suppose you number those blocks arbitrarily down as 0 0 coordinate and, as you refer to any Cartesian coordinate, so each unit can be marked from 0 through 7 instead of 1 to 8. If you do that one, you can cover for. This sigma is for all summation. If you just apply for this alone, it simply implies summation of all the pixel intensity. To keep it simple, let us deal only with one component, color component, or just a plain monochrome. If you are processing color, we are dealing with only 1 of the components, either Y or Cb Cr. So this is a cosine transform that we apply here. There are 2 cos terms and this 3. This is because we have a two-dimensional picture and we need what is called a two-dimensional DCT for that. Similarly, you also have a 1D DCT, wherein there will only be these 2 terms. There will be only 1 sigma. There will be only one constant. This is the basic definition for DCT. uvs is also like wise 0 through 7. This is to say that you have a total combination of 64 because 0 through 7 is 8, so 8 into 8. So totally you can have DCT this is called a DCT coefficient. So if you take one block of image, consisting of 8 by 8 pixels, out come 64 coefficients here.

(Refer Slide Time: 37:55)



A novel, parallel algorithm for fast evaluation of DCTQ

2D-DCT of a block of size 8 x 8 pixels of an image is defined as

$$T(u, v) = \frac{1}{4} c(u)c(v) \sum_{x=0}^{7} \sum_{y=0}^{7} f(x, y) \cos\left[\frac{(2x+1)u\pi}{16}\right] \cos\left[\frac{(2y+1)v\pi}{16}\right]$$

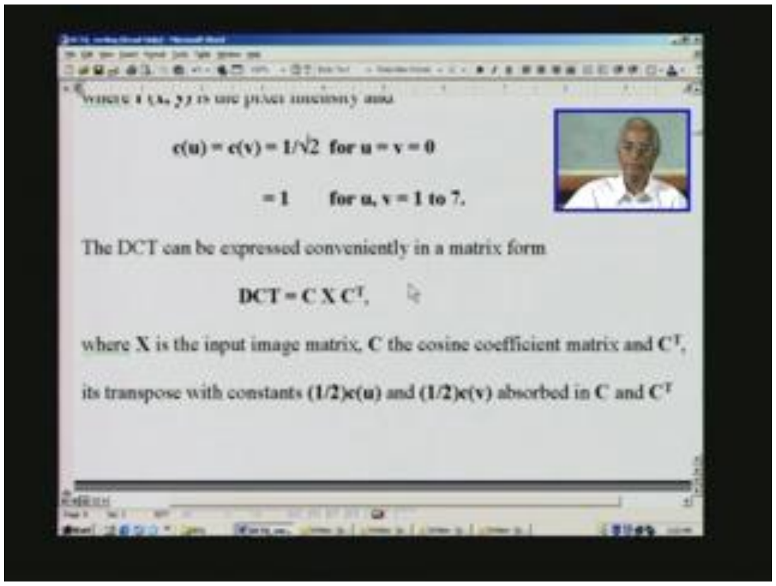
where **f (x, y)** is the pixel intensity and

$c(u) = c(v) = 1/\sqrt{2}$ for $u = v = 0$

$= 1$ for $u, v = 1$ to 7.

Mind you, it has no direct relationship to each of these pixels. You cannot say one pixel is related to one coefficient. In fact, each of the coefficients is a conglomeration of all the pixels information. For example, the very first coefficient is known as DC coefficient and that is obtained by assigning u v equal to 0. This will boil down to cos 0, cos 0, which will be 1 here. What is left here is only this. So this is nothing other than taking all the pixel intensity for the entire block, which means 64 pixels, and adding them together. That is what is implied by this. If you take a for u v it is 0, so what you have here is 1 by root 2, 1 by root 2 and this contributes to 8 together. In other words, DC coefficient, which is u v equal to 0 each is very first coefficient. We will compute later on. That is nothing other than all pixel intensity in a block divided by 8.It is not an average. Many people mistake it as an average of all the pixel intensity. It is simply not true. Average is 64 times division, whereas we deal with only 8 here, which mean that 8 times the average information is contained in just one coefficient. In other words, if you take just DC coefficient alone and code it, transmit, and receive at the other end and do the inverse operation you still recover the whole image, only details may not be that easy to see. You need much higher coefficients in order to create a detail especially at the sides, vertical, horizontal, or even through diagonal and so on. This is the basic DCT transformation. T this cu cv is 1 by root 2, only u v equal to 0. For all other values it is 1 here. Now we go into the new algorithm developed for the sake of implementing it as an asic or fpga and in general as a VLSI.

(Refer Slide Time: 42:15)



where f (x, y) is the pixel intensity and

$c(u) = c(v) = 1/\sqrt{2}$ for $u = v = 0$

$= 1$ for $u, v = 1$ to 7.

The DCT can be expressed conveniently in a matrix form

$\mathbf{DCT = C\,X\,C^T}$,

where **X** is the input image matrix, **C** the cosine coefficient matrix and $\mathbf{C^T}$, its transpose with constants **(1/2)c(u)** and **(1/2)c(v)** absorbed in **C** and $\mathbf{C^T}$

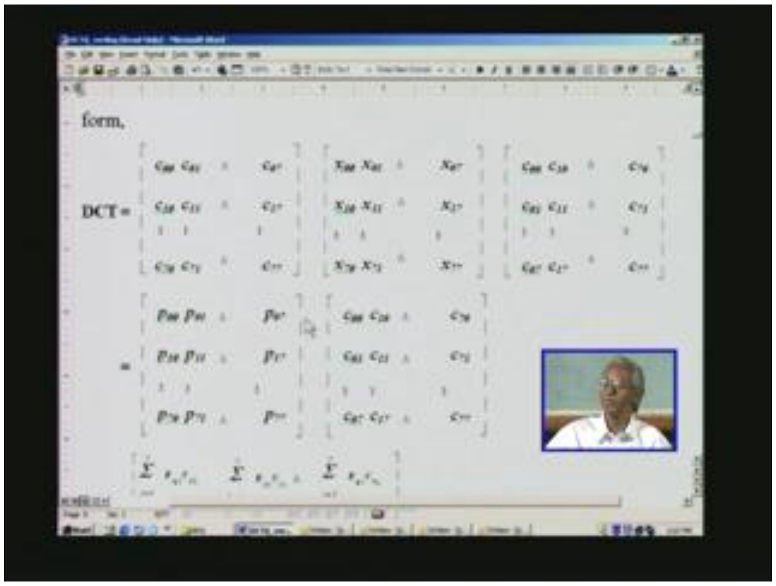
If you look at this expression here, we can see that it can also be expressed in the form of a matrix. For example, this is 0 through 7, which means that there are only 8 combinations. Similarly, if you take bifurcate half cu and then half cv and assign that one absorb that in this cos term half the thing here and half cv in second thing. You can then represent it as 8 by 8 matrices. There will be 3 such matrices. The first cos term will get reflected in C. This will get reflected in another C T which is nothing other than the same matrix but a transpose of the same. What you see as a row will be here as a column. That is what you imply by the transpose. So DCT can be conveniently expressed as a matrix here, where X is the input image. A block of image is being referred to here; therefore, it has to be 8 by 8 size. Similarly, we have seen because its varying from x y 0 to 7 and naturally C and Ct also will be on 8 by 8 matrices

We have already explained in the present implement algorithm this has been absorbed in C and other portion half cv has been absorbed in Ct here. This can be expanded so as to get an understanding. There were dots here. Unfortunately it has changed. Just ignore this.

The first matrix is C0 C101 and so on up to C07. This is the first row. You have 8 rows like this. The same is the case here except that this row becomes the column. That is how it is. X stands for the actual image block so you have each of them corresponding to 8 bits because we will process only one color component at a time. We saw that only 8 bits are required, that is, only 1 byte is required. So each of these will be an 8 byte information so, if you want to evaluate this DCT, you just see the number of calculations, multiplication involved. It is quite a stupendous value. Let us say the purpose of looking into the algorithm is to make it convenient for FPGA implementation that can yield a very high throughput. That is the purpose of deriving, I mean contributing, a new algorithm, which we will see presently. How do you multiply 2 matrices? You have 3 matrices. You have to multiply them. In order to multiply, you can only do 2 matrices at a time.

So row of a particular thing is a first matrix is taken and multiplied element by element with the column. Namely C00 multiplied by X00, then plus C01 into X110 and so on. That constitutes one particular element, which is P00. Note that we need 8 multipliers in order to accomplish 8 multiplications and one adder, which we need for adding all 8 terms together. So it is an 8 input adder. If you recollect, we have already seen the design for multiplier and adder and we will use precisely the same design here. Therefore, those designs will not be repeated here while treating the verilog code. In this case, you multiply this row by column. In the very first cycle, as far as the implementation is concerned, you need 1 clock. We can accomplish in one clock cycle. You may wonder how it is done. We will see that presently. Once it is done, we repeat the same thing. Retain the same row but change the column to the second column, and get P01, and so on. For 8 clock cycles, you would have got the entire row of this and we term this as a partial product. Once this row is complete, we can start multiplying the second one. We repeat the same multiplication here, element by element, P00 into C00 plus P01 into C01 and so on.

(Refer Slide Time: 44:03)



form,

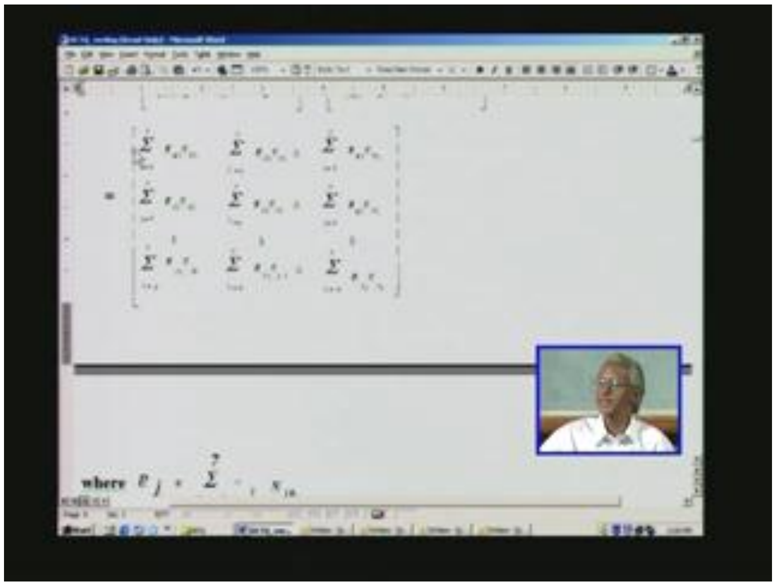
$$DCT = \begin{bmatrix} c_{00} & c_{01} & \cdots & c_{07} \\ c_{10} & c_{11} & \cdots & c_{17} \\ \vdots & \vdots & & \vdots \\ c_{70} & c_{71} & \cdots & c_{77} \end{bmatrix} \begin{bmatrix} x_{00} & x_{01} & \cdots & x_{07} \\ x_{10} & x_{11} & \cdots & x_{17} \\ \vdots & \vdots & & \vdots \\ x_{70} & x_{71} & \cdots & x_{77} \end{bmatrix} \begin{bmatrix} c_{00} & c_{10} & \cdots & c_{70} \\ c_{01} & c_{11} & \cdots & c_{71} \\ \vdots & \vdots & & \vdots \\ c_{07} & c_{17} & \cdots & c_{77} \end{bmatrix}$$

$$= \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{07} \\ p_{10} & p_{11} & \cdots & p_{17} \\ \vdots & \vdots & & \vdots \\ p_{70} & p_{71} & \cdots & p_{77} \end{bmatrix} \begin{bmatrix} c_{00} & c_{10} & \cdots & c_{70} \\ c_{01} & c_{11} & \cdots & c_{71} \\ \vdots & \vdots & & \vdots \\ c_{07} & c_{17} & \cdots & c_{77} \end{bmatrix}$$

$$\Big[\sum [illegible] \quad \sum [illegible] \quad \sum [illegible]$$

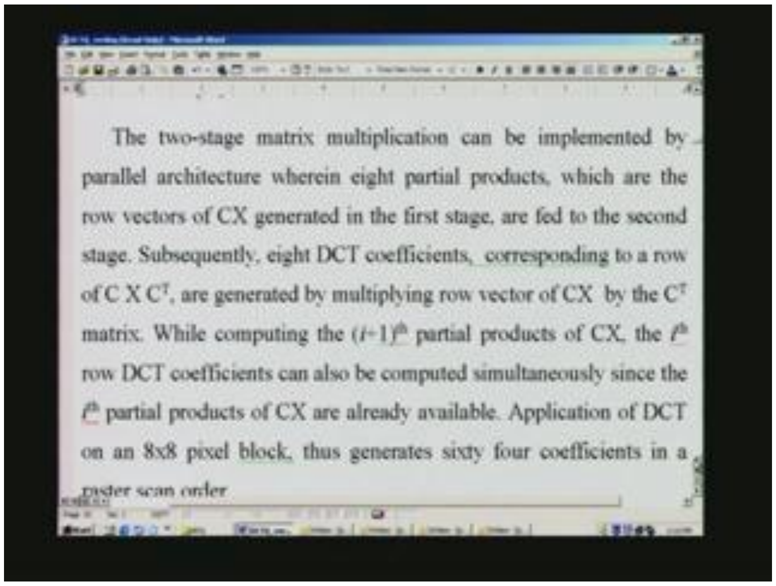
Again you require another 8 multipliers, and one more adder is required for adding those. When this is done, you would have got the very first DCT coefficient here. That is what is shown here. The crux of this algorithm is here. As you evaluate this DCT coefficient, concurrently you can also evaluate this coefficient, because this is quite free to execute this concurrently. Concurrently means simultaneously. So that is the contribution of this algorithm. Thereby you save time and the whole thing can be pipelined in such a fashion. You will get the output at the rate of 1 coefficient at every clock pulse. That is the best possible speed you can conceive of and it is because of this regular structure. In fact, I mentioned that there are over 100 technical papers in journals as well as international conferences, and all of them are oriented towards reducing redundancy in this by using what is known as butterfly structure. Thereby regularity of the structure is lost, whereas here we are retaining the regularity of the structure, and applying some other trick so as to get a very high throughput which cannot be achieved in the butterfly structures and other researches that have been done. This is the reason why this algorithm is claimed to be the fastest as of today. This is the basic DCT, wherein each of these terms is just put here, which we have already explained before.

(Refer Slide Time: 47:50)



I will just this complete the DCT. I will just summarize what we have done here. The 2 stage matrix multiplication can be implemented by parallel architecture, wherein 8 partial products that are the row vectors of CX generated in the first stage are fed to the second stage. Subsequently, 8 DCT coefficients, this is after using the partial product so corresponding to a row of CXCT, this is the DCT coefficient that emerges are generated by multiplying row vector of CX by the CT matrix. While computing the (i plus 1)th partial products of CX, the ith row DCT coefficients can also be computed simultaneously, since the ith partial products of CX are already available. This is the basic contribution in this algorithm, which we have explained before.

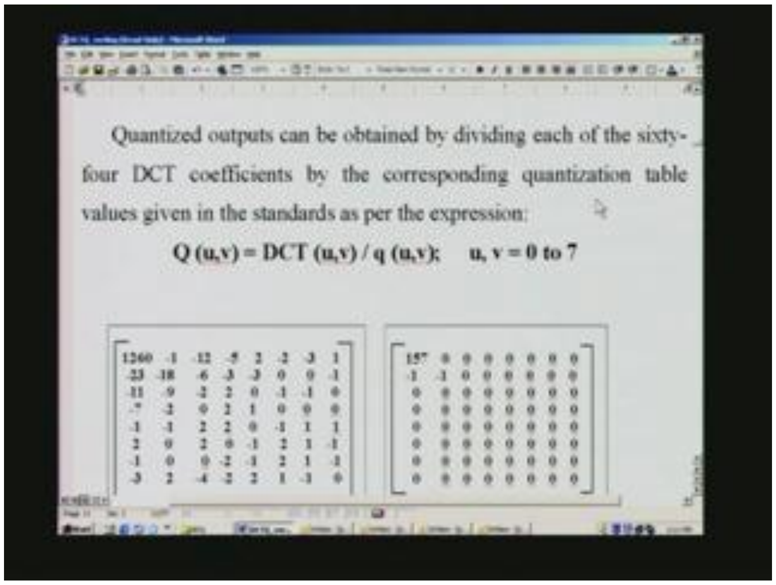
(Refer Slide Time: 48:48)



The two-stage matrix multiplication can be implemented by parallel architecture wherein eight partial products, which are the row vectors of CX generated in the first stage, are fed to the second stage. Subsequently, eight DCT coefficients, corresponding to a row of C X C^T, are generated by multiplying row vector of CX by the C^T matrix. While computing the $(i+1)^{th}$ partial products of CX, the i^{th} row DCT coefficients can also be computed simultaneously since the i^{th} partial products of CX are already available. Application of DCT on an 8x8 pixel block, thus generates sixty four coefficients in a raster scan order

Application of DCT on an 8 by 8 pixel block thus generates 64 coefficients in a raster scan order, which we have explained right at the beginning. The next step is what is called quantization. Quantization outputs can be obtained by dividing each of the 64 DCT coefficients that came in the previous step by the corresponding quantization table values given in the standards. This is part of the standards.

(Refer Slide Time: 49:46)



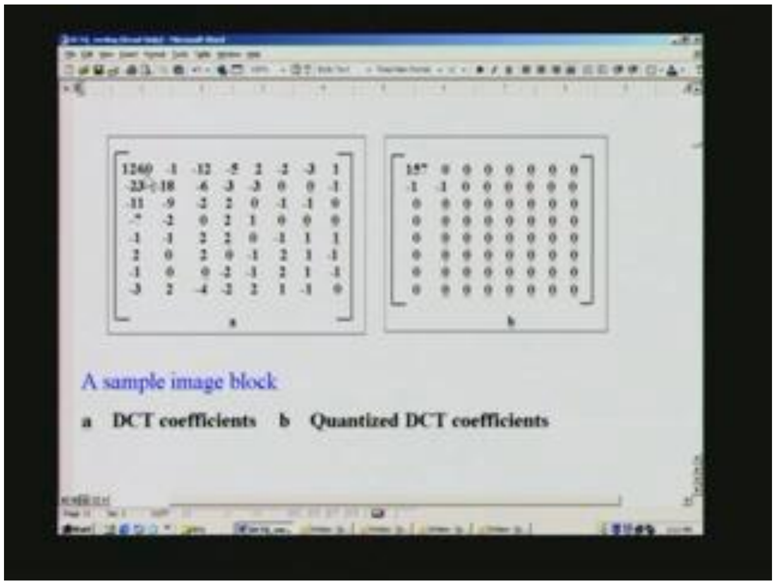
Quantized outputs can be obtained by dividing each of the sixty-four DCT coefficients by the corresponding quantization table values given in the standards as per the expression:

$$Q\,(u,v) = DCT\,(u,v) \,/\, q\,(u,v); \quad u, v = 0 \text{ to } 7$$

$$\begin{bmatrix} 1260 & -1 & -12 & -5 & 2 & -2 & -3 & 1 \\ -23 & -18 & -6 & -3 & -3 & 0 & 0 & -1 \\ -11 & -9 & -2 & 2 & 0 & -1 & -1 & 0 \\ -7 & -2 & 0 & 2 & 1 & 0 & 0 & 0 \\ -1 & -1 & 2 & 2 & 0 & -1 & 1 & 1 \\ 2 & 0 & 2 & 0 & -1 & 2 & 1 & -1 \\ -1 & 0 & 0 & -2 & -1 & 2 & 1 & -1 \\ -3 & 2 & -4 & -2 & 2 & 1 & -3 & 0 \end{bmatrix} \qquad \begin{bmatrix} 157 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

It is another 8 by 8 matrix with elements of 8, 16, 16, 19, 22 and so on. For example, if you take a DCT, if you evaluate the DCT for 1 block, sample block here, you get a 1260 DCT here. If you divide this by 8, which is one of the values in the quantization matrix table, you get 157 after rounding off. This is called a quantized output. As I mentioned earlier, the purpose of DCTQ is to convert as many 0s as possible, which need not be encoded in the VLC. That is how you bring about compression.

(Refer Slide Time: 50:14)



$$\begin{bmatrix} 1260 & -1 & -12 & -5 & 2 & -2 & -3 & 1 \\ -23 & -18 & -6 & -3 & -3 & 0 & 0 & -1 \\ -11 & -9 & -2 & 2 & 0 & -1 & -1 & 0 \\ -7 & -2 & 0 & 2 & 1 & 0 & 0 & 0 \\ -1 & -1 & 2 & 2 & 0 & -1 & 1 & 1 \\ 2 & 0 & 2 & 0 & -1 & 2 & 1 & -1 \\ -1 & 0 & 0 & -2 & -1 & 2 & 1 & -1 \\ -3 & 2 & -4 & -2 & 2 & 1 & -1 & 0 \end{bmatrix}$$

a

$$\begin{bmatrix} 157 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

b

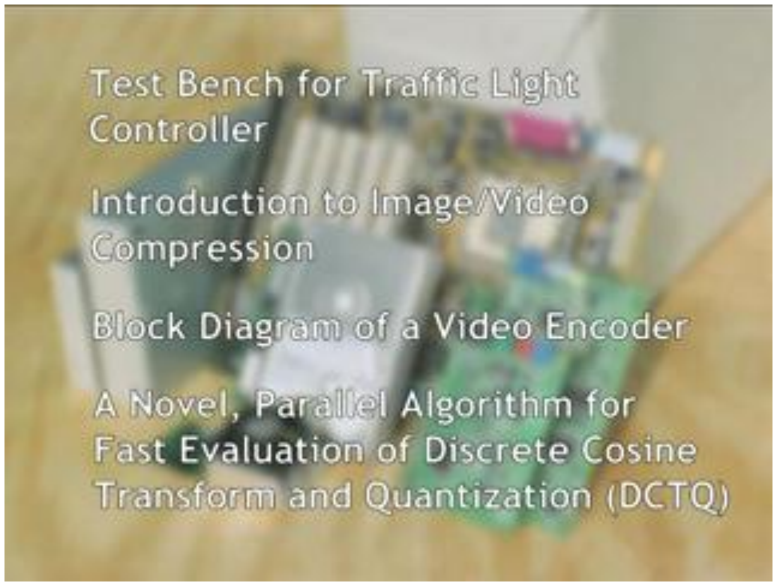
A sample image block

a DCT coefficients b Quantized DCT coefficients

We will see more in our next lecture. Thank you.

Summary of Lecture 44

(Refer Slide Time: 50:53)



Test Bench for Traffic Light Controller

Introduction to Image/Video Compression

Block Diagram of a Video Encoder

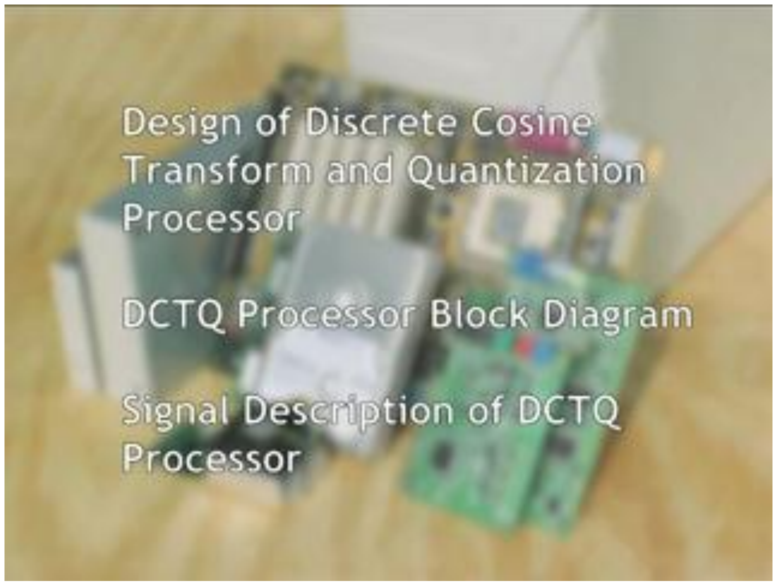
A Novel, Parallel Algorithm for Fast Evaluation of Discrete Cosine Transform and Quantization (DCTQ)

Next Lecture

System Design Examples

(Continued)

(Refer Slide Time: 51:24

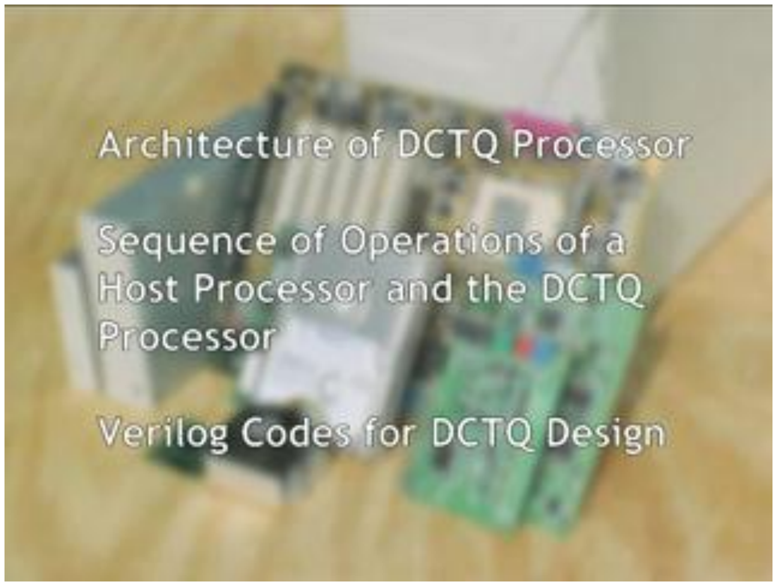


Design of Discrete Cosine Transform and Quantization Processor

DCTQ Processor Block Diagram

Signal Description of DCTQ Processor

Refer Slide Time: 51:58)



Architecture of DCTQ Processor

Sequence of Operations of a Host Processor and the DCTQ Processor

Verilog Codes for DCTQ Design