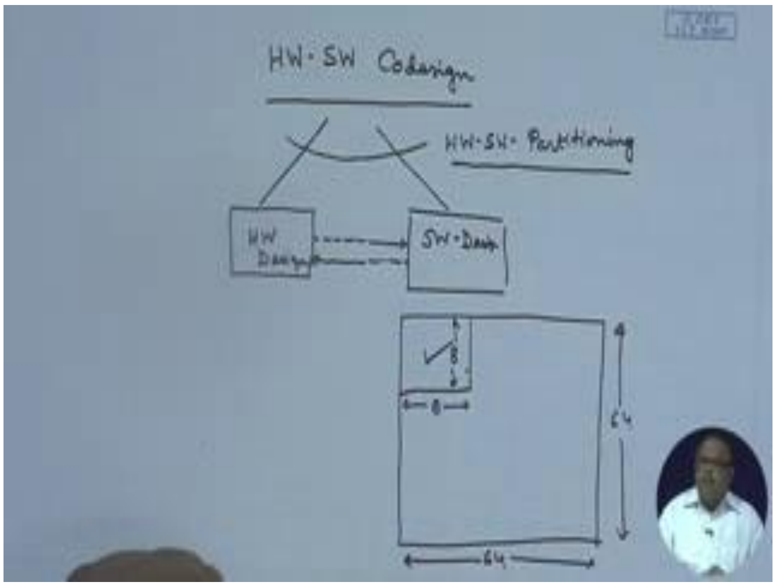
Embedded Systems Design
Prof. Anupam Basu
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 43
Digital Camera Design

See, in the earlier classes, we have seen in the last couple of lectures, we have seen how we proceed with high-level synthesis. We have in particular seen how we do profiling of the tasks and then look at the different scheduling steps, and we have in particular seen four scheduling algorithms by which we can schedule the tasks in the different control steps. Today, we will look at a particular example of an interesting example of a digital camera. A digital camera example I have selected this example from the textbook that has been referred for this course that is embedded system design I unified hardware-software introduction by Frank Vahid and Givargis. I have taken the examples straight away from there and it is a very popular example used in many embedded system courses.

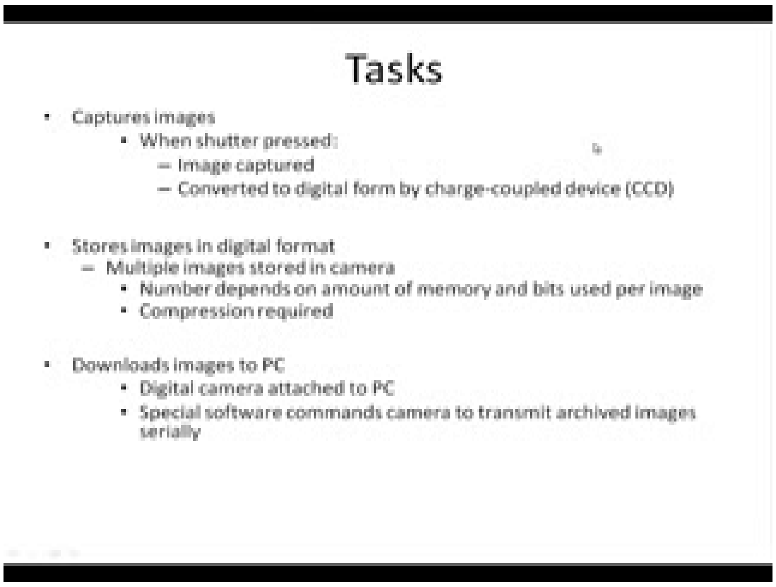
A digital camera will illustrate the design of a digital camera will illustrate the different steps that are involved in an embedded system design. In particular, I have selected this because I want to carry out the hardware show you the hardware software tradeoff. As I had said earlier that any embedded system design is not only hardware design is not only software design, but designing both, but more importantly I had used another term called hardware-software code design.

(Refer Slide Time: 02:02)



If you recall that is we carry out the hardware design, we also carry out the software design, and these two are dependent on the design decisions are dependent on each other. And try to that this arcing this forking that has happened here is known as hardware-software partitioning that is which part of the tasks or which subset of the tasks will I carry out in hardware and which parts I will carry out in software. That will be illustrated and how that affects the performance that will be illustrated in the digital camera example.

(Refer Slide Time: 03:14)



Tasks

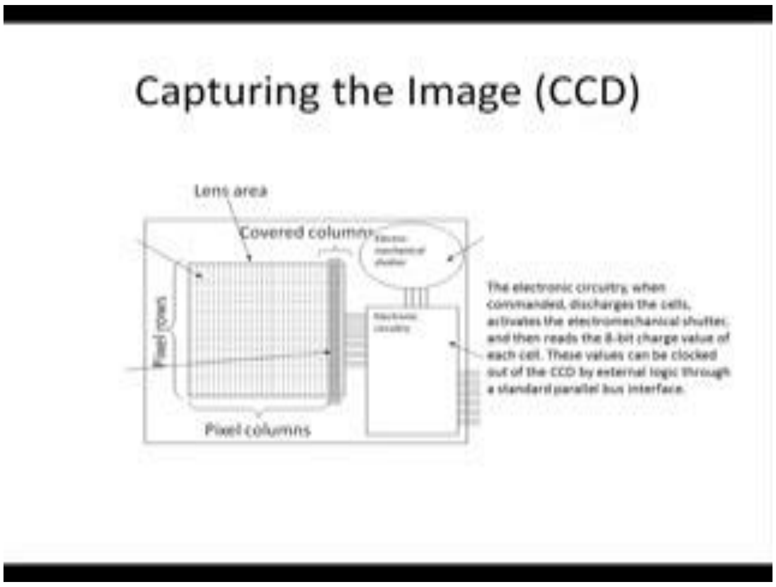
- Captures images
    - When shutter pressed:
        - Image captured
        - Converted to digital form by charge-coupled device (CCD)
- Stores images in digital format
    - Multiple images stored in camera
        - Number depends on amount of memory and bits used per image
        - Compression required
- Downloads images to PC
    - Digital camera attached to PC
    - Special software commands camera to transmit archived images serially

To start with let us think of the different tasks, there are digital camera is expected to perform. First of all, it will capture images; it will store the image in digital format. It will there should be a facility to download images to PC. Now, this is a simplified task besides that there are many other functions that can be carried out, but here we see that the when we capture images will capture we have got a CCD camera using with using CCD will capture the images that means, when the shutter is pressed the image is captured. And then that image is converted to digital form by charge coupled device. So, the image will be incident on the lens, and from there it will receive by the charge coupled device which will convert the image into the digital form.

Now the images have to be stored in the digital format. And obviously all of you use digital camera and you know that we can store multiple images we can store in the camera, there are no films required. And the number of images that we can store depends on the amount of memory - number one and the resolution of the images. When you go for high resolution that means we are using high higher number of bits per pixel. So, higher number of bits the number of bits use per image. Now, in order that I can store more number of images some sort of compression is required that is the part of the task.

The third is the downloading of the images to the PC. The digital camera will have to be attached to the PC, and there will be some software commands by which the camera will transmit the archived images serially to the PC. It is usually done to some u s b or other communication mechanism. Besides that there are many other functions like zoom in, zoom out, flash, auto and all those things that we are not considering in this lecture for the sake of manageability and simplicity.

(Refer Slide Time: 05:48)



Capturing the Image (CCD)

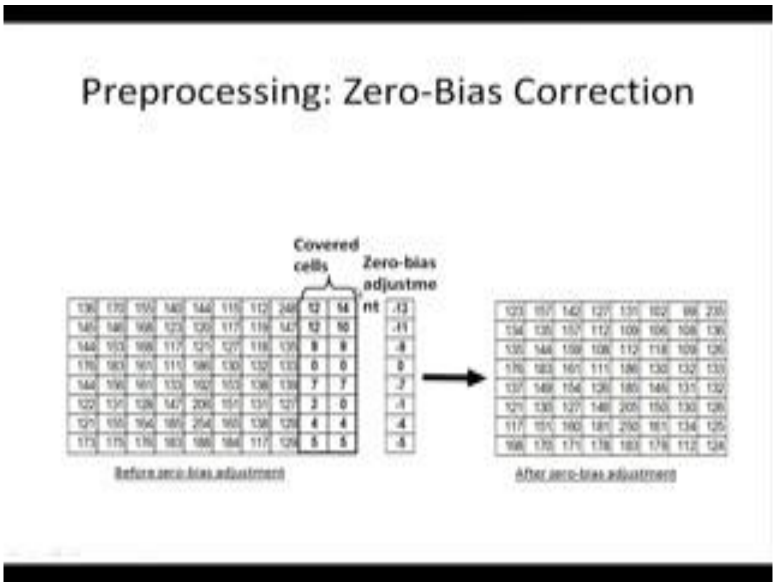
The electronic circuitry, when commanded, discharges the cells, activates the electromechanical shutter, and then reads the 8-bit charge value of each cell. These values can be clocked out of the CCD by external logic through a standard parallel bus interface.

So, let us start with how the image is captured. Now, we have got these dots that you can see here are the different cells - CCD cell. This is the lens area. And the image will be incident on this. And some of depending on the image, some of the cells will be activated, some will be not, some will have incident light some will have not. So, there is an electro mechanical shutter by which if we press that, then this lens or this area is exposed to the image or to the light. And there is an electronic circuitry as the lens is as the image is incident on this, the cells get charged that is why they are charge coupled devices, they will get charged.

Now, each cell will have a particular level of charge and that particular level of charge will be encoded using 8 bits. So, 255 will be the highest extent of incidence or intensity and 0 will be the least. And this electronic circuit will read those, and also it can be used to discharge this cells. Now, most importantly you have to look at this. Ideally, what is the scenario, ideally if there is no light incident then the charge should be 0, but in practice there are some manufacturing defects and so the charge is nonzero even for no incidence of light.

So, we have to take some corrective measure for that. In order to do that what is done is the two these two columns on the extreme right, you can see are covered black. Therefore, no light will be incident on this. And since no light is incident on this, the value should be 0. If the value is nonzero say 5 or something, then I have got a positive offset of 5 for all these, so I have to correct that image. If I have got 255 here that should be actually 250, so that is called zero correction or offset correction, zero offset correction so that is in essence the structure of the CCD part how we will capture the images.

(Refer Slide Time: 09:06)



Preprocessing: Zero-Bias Correction

Covered cells

Zero-bias adjustment

136	170	155	140	144	115	112	248	12	14		-13
146	146	166	123	120	117	119	147	12	10		-11
144	153	168	117	121	127	118	135	8	8		-8
176	183	161	111	186	130	132	133	0	0		0
144	156	161	133	192	153	138	139	7	7		-7
122	131	126	147	206	151	131	127	2	0		-1
121	155	164	185	254	165	138	129	4	4		-4
173	175	176	183	188	184	117	129	5	5		-5

Before zero-bias adjustment

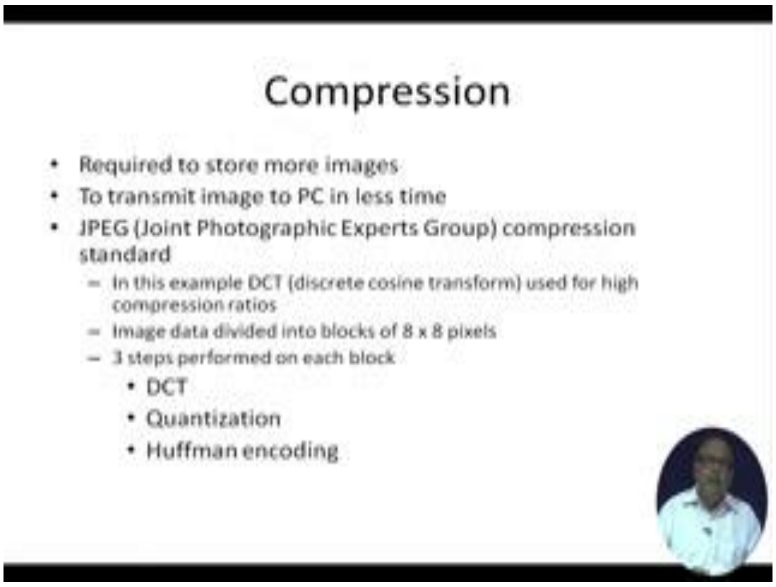
123	157	142	127	131	102	99	235
134	135	157	112	109	106	108	136
135	144	159	108	112	118	109	126
176	183	161	111	186	130	132	133
137	149	154	126	185	146	131	132
121	130	127	146	205	150	130	126
117	151	160	181	250	161	134	125
168	170	171	178	183	179	112	124

After zero-bias adjustment

So, as I had said after we capture the image, we can see that we have captured the image here. Then the next step will be pre processing. And the pre processing will start with 0 bias corrections. Here we have got a matrix 8 by 8 we are showing plus 2. Now, these are the covered cells, you have seen that there were two columns of those, so the two columns of covered cells. So, we can have we are looking at the intensity values that are coming in this two cover cells, so 12 and 14 that means, my zero-bias adjustment to be minus 13, 12 and 10 minus 11, 12 and 10 is 22. So, average of these is minus 11. Here I have got in that particular dot there was no manufacturing defect. So, no adjustment is required. It is also assumed that the variation of the manufacturing defect is paid along the rows in the way they are manufactured, but along the columns they say

So, next is what we do is we take each of these values and subtract the average of this. So, 136 minus 13 becomes 123; 146 minus 11 becomes 135, here you look at 186 and that is the first, second, third, fourth, fifth 554. So, I come to here. So, 186 corrected remains 186, because here for this row, it is zero all right. In that way we carry out the zero-bias adjust. So, first we have captured the image and then we have done the zero-bias. We will have to the CCD camera will have to do this tasks, so that is done zero-bias adjustment is done.

(Refer Slide Time: 11:39)



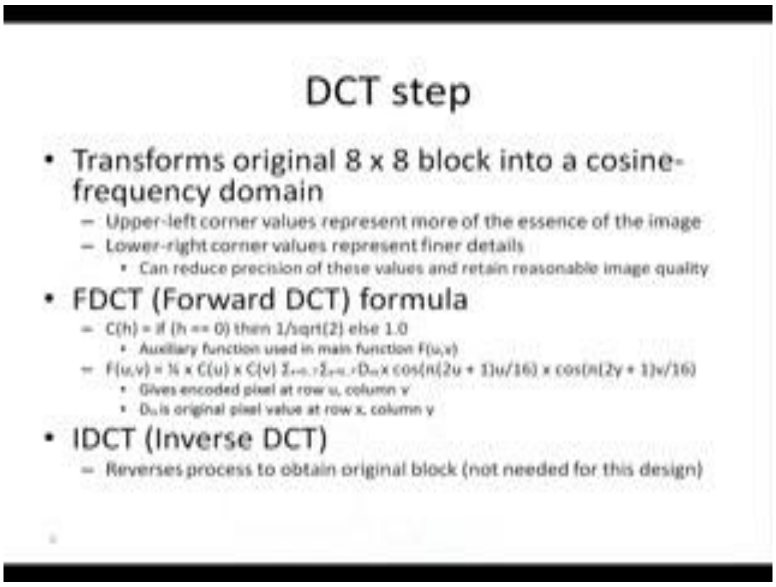
Compression

- Required to store more images
- To transmit image to PC in less time
- JPEG (Joint Photographic Experts Group) compression standard
  - In this example DCT (discrete cosine transform) used for high compression ratios
  - Image data divided into blocks of 8 x 8 pixels
  - 3 steps performed on each block
    - DCT
    - Quantization
    - Huffman encoding

Next, we need to do compression, and that is required to store more images to transmit images to PC in lesser amount of time, if I can compress, then I will have to transmit less number of bits. In digital camera, we are using the JPEG format of compression if you know that jpeg stands for joint photographic experts group there are different modes that JPEG allows. But in this example we are concentrating on a particular type of compression that is the discrete cosine transform. We will carry out the discrete cosine terms transformed because that gives us high compression ratios. The entire image we are assuming is 64 by 64 pixels 64 by 64 that entire thing we are dividing into 8 by 8 blocks.

So, we have got the image which is 60 for by 64 pixels, on this side also 64. We are may breaking them into 8 by 8 that means how many such blocks I will have 64 blocks, I will have 64 such blocks. And we will be carrying out the compression and everything on each of these blocks separately that means, at a time I will take one of these blocks and work on that. For each block, three types of operation, three operations are actually carried out. What are the three operations, one is discrete cosine transform, other is quantization, another is Huffman encoding we will have to encode them we will have to actually write them in different bit patterns. We have to do it in a way; so that the number of bits are also minimized at every stage we will try to minimize the number of bits.

(Refer Slide Time: 14:16)

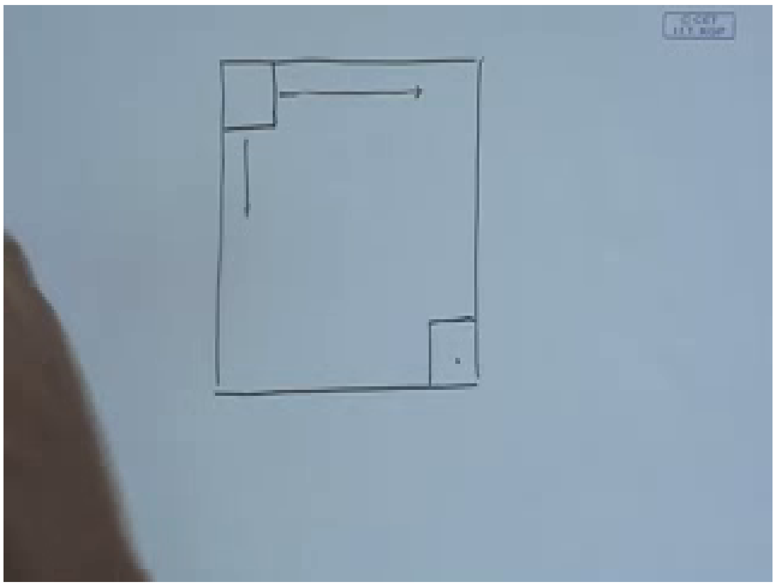


DCT step

- Transforms original 8 x 8 block into a cosine-frequency domain
  - Upper-left corner values represent more of the essence of the image
  - Lower-right corner values represent finer details
    - Can reduce precision of these values and retain reasonable image quality
- FDCT (Forward DCT) formula
  - C(h) = if (h == 0) then 1/sqrt(2) else 1.0
    - Auxiliary function used in main function F(u,v)
  - F(u,v) = ¼ x C(u) x C(v) $\Sigma_{x=0..7}\Sigma_{y=0..7}D_{xy}$ x cos(π(2u + 1)u/16) x cos(π(2y + 1)v/16)
    - Gives encoded pixel at row u, column v
    - D_{xy} is original pixel value at row x, column y
- IDCT (Inverse DCT)
  - Reverses process to obtain original block (not needed for this design)

So, let us star start with the discrete cosine transform step. I am not going into the details of discrete cosine transform that you can get in any signal processing book or also the book of Vahid and Givargis explains that. I will just mentioned the only with the parts which are required for our design. We are transforming the original 8 by 8 block or selected one 8 by 8 block, and I am trying to transform that to the cosine frequency domain, cos omega t type of domain. As I do that the beauty of this is that the upper left corners will represent more essence of the image, the lower right corners will represent the finer details.

(Refer Slide Time: 15:23)



So, what is happening actually let us I mean if I have a block this part will have the essence of the image mostly and as we go down either this way or this way, their essence comes down and here is the minimum. Now for those of you know about the information theory that any information and image is also information. So, whichever elements are very frequent, they have got less information; whichever a less frequent, they carry more information.

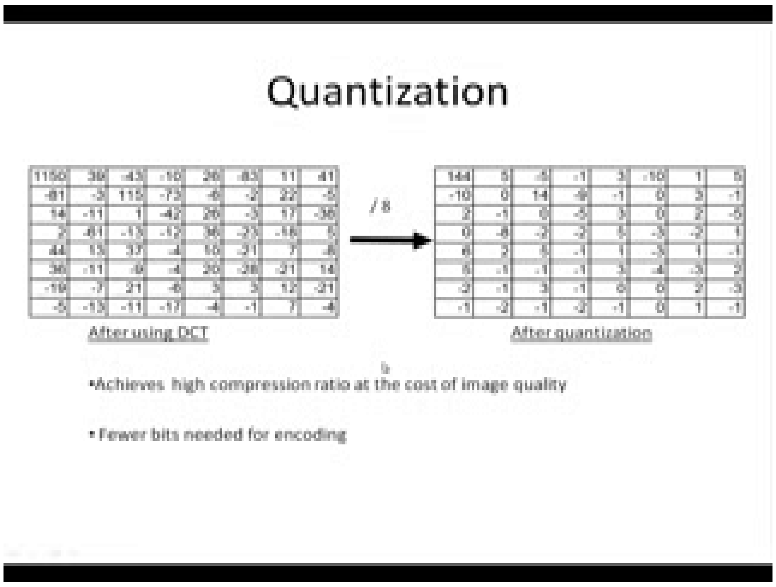
So, here actually the elements that will come ultimately we will see that later will be of lesser frequency. And here as we go down the higher frequency components will come. So, upper left corner represent more essence. And so consequently what happens I can take differ I can offer differential treatment to the different parts of the image whichever is carrying the actual essence, I will devote more attention, in our case more attention means modem maybe bits or actually we will put in lesser bits there. Because we will come to that we will have to do is take care of that resolution much more.

Now, we actually do say when we take an image and convert it to the frequency domain, cosine frequency domain that is forward transform. And later on, from there we have to retrieve the image when you want to print the image you do not want to have the transformed cosine frequencies spread on your photographic plate instead you want that to a brought back, and actually image should be retrieved that is inverse DCT. But for the camera we are not interested in the inverse DCT, we are interested in the forward

DCT. In this forward DCT, this I again I tell you that I am not going to devote much time on this just to show you what it is many of you might be knowing discrete cosine transform.

So, C h is basically a basis function that if h whatever h is it is 0 then I have C h becomes 1 by root 2; otherwise it is 1. Now, this function is used F u, v, u and v are say in the in the two dimensional image u and v are the two indices. So, for each pixel, we take different u's and v's, one-fourth C u C v there is a dot product all through and then sigma x; that means, row wise I go from 0 to 7 - all the 8 rows and then column 0 to 7 D x y is the original pixel value at row x column y. I am multiplying that D x with cosine frequencies with that I get the transform. So, this F u, v will give me the encoded pixel at row u column v; and D x y is the original image intensity, so that is what we are trying to do.

(Refer Slide Time: 19:33)



Quantization

1150	39	-43	-10	26	-83	11	41
-81	-3	115	-73	-6	-2	22	-5
14	-11	1	-42	26	-3	17	-38
2	-61	-13	-12	36	-23	-18	5
44	13	37	-4	10	-21	7	-8
36	-11	-9	-4	20	-28	-21	14
-19	-7	21	-6	3	3	12	-21
-5	-13	-11	-17	-4	-1	7	-4

After using DCT

/ 8

144	5	-5	-1	3	-10	1	5
-10	0	14	-9	-1	0	3	-1
2	-1	0	-5	3	0	2	-5
0	-8	-2	-2	5	-3	-2	1
6	2	5	-1	1	-3	1	-1
5	-1	-1	-1	3	-4	-3	2
-2	-1	3	-1	0	0	2	-3
-1	-2	-1	-2	-1	0	1	-1

After quantization

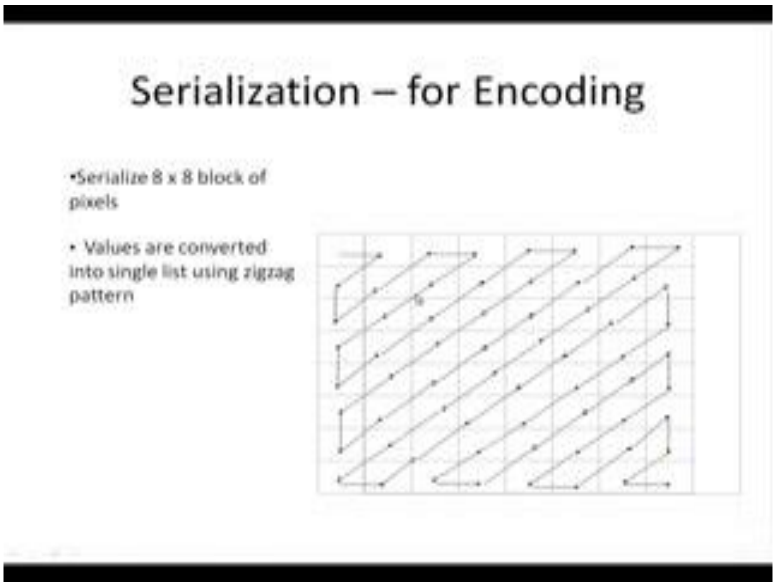
- Achieves high compression ratio at the cost of image quality
- Fewer bits needed for encoding

So, I have taken an image; taken an 8 by 8 block out of it and then applied DCT. After DCT, I get this 8 by 8 matrix, after using DCT. So, you can very well see that this part has got a value 1150, 39, minus 81 look at the absolute values of this. So, these are the essence of the image. As we go down here this is minus 4 in that we have got it. Now, this is in a continuous domain, I mean not continuous domain, they are discrete values, but I do not want to go by the value absolute value of this, I want to quantize it. So, some of the values are clubbed together.

So, when I quantize, I quantize with respect to some power of 2. So, here if I divide each element with 8 that is 2 raised 3, if I divide each of these elements by 2 raised 3; that means 8. I will get such values this is again quantized you can see 1150 divided by 8 is not exactly 144, but we get the say for example, 39 divided by 8 is four point something, but we will make it to the nearest rounding search 5. So, after quantization I get this thereby what did I sacrifice by quantizing, I sacrifice the quality of the image of the position. I will sacrifice that, but what did I save number of bits. So, therefore, I get the higher compression ratios here. And when well I can go out this way by multiplying by 8 right then there will be a little change. So, we need fewer bits for encoding. So, this is what is quantization.

So, once again, I go back, I start with capturing the CCD all right. I capture the image, this will be some distribution of intensity values, then I correct them with zero-correction. So, I get a corrected distribution of intensity values here, then I do some compression; and for compression, we selected here discrete cosine transform. As I do that, I get a matrix like this, which I am quantizing I am getting this matrix that is it.

(Refer Slide Time: 22:51)



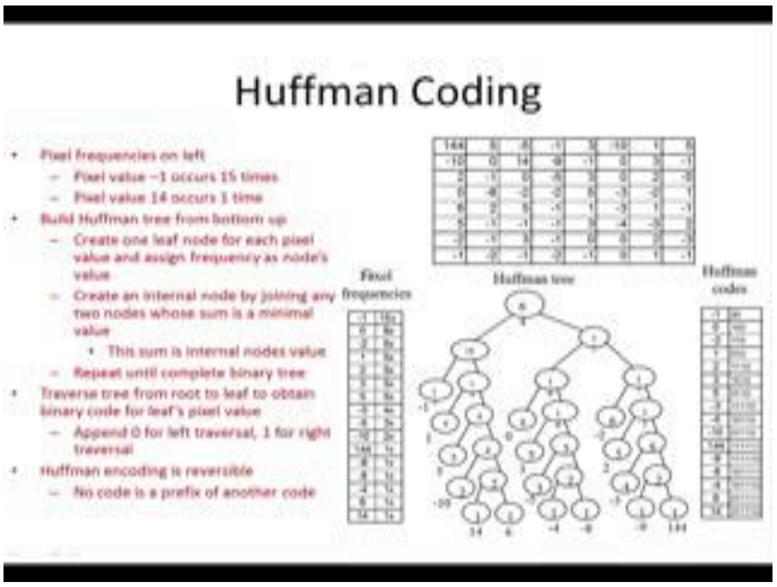
Serialization – for Encoding

- Serialize 8 x 8 block of pixels
- Values are converted into single list using zigzag pattern

Now, after quantization here I have to actually have the bits I will have to actually have to give the bits. I have to encode them; I have to encode them using some bits. Now, for encoding and I said that we will be using Huffman encoding. One of the first steps that is done before the encoding is serialization. Each of the 8 by 8 will be serialized with respect to some; so if I serialize this I will get an array of some integers.

Now, how do I see realize it? The serialization is done using a zigzag pattern, why, in a zigzag pattern, because yeah if I had straight way gone here, then from more information to less information, then I come again more information to less information, more information look similar. Or the other way if I start from here and go down more to less, more to less like that. So, then I would have lost the serialization of the information content, therefore a zigzag pattern is utilized so that I keep the more information content things together as far as possible right clear? So, that is the pattern that is done.

(Refer Slide Time: 24:38)



Huffman Coding

- Pixel frequencies on left
  - Pixel value −1 occurs 15 times
  - Pixel value 14 occurs 1 time
- Build Huffman tree from bottom up
  - Create one leaf node for each pixel value and assign frequency as node's value
  - Create an internal node by joining any two nodes whose sum is a minimal value
    - This sum is internal nodes value
  - Repeat until complete binary tree
- Traverse tree from root to leaf to obtain binary code for leaf's pixel value
  - Append 0 for left traversal, 1 for right traversal
- Huffman encoding is reversible
  - No code is a prefix of another code

Now, once that pattern I get, now here you let us explain a little bit for those of you who do not know Huffman encoding. I had this matrix, if you recall here after quantization, I had this matrix. In this matrix, I find out that these are different pixel values here minus 1, 0, minus 2 all the different pixel values that I have and I look at the pixel frequencies, how frequently they appeared. I can see that pixel value is one here pixel value one has appeared 5 times, pixel value minus 1 has appeared 15 times and pixel value 14, 14 here 14 has appeared only once less frequent. So, I sort them. So, this one has got more information 14, because it has occurred only once, 6 has occurred only once. So, I must be respect to them somebody who is regularly visiting does not get enough respect.

So, you look at this, now we will build the Huffman tree in a bottom up fashion. What will you do, this is the Huffman tree in sort of it; how do you do that? Create one leaf node for each pixel value; for each pixel value, for each of them 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, all these 16 pixel values are give are made the leaf node of this tree. All of them 1, minus 1, 1, 5 minus 10, 14, 6 all of them are made the leaf node. And we assigned the frequency to these leaf nodes. For example 14 has got the frequency 1.

Let us look at say minus 1o has frequency 2, because it occurred twice here minus 10 here and there must have been another minus 10 here. We can see that 4 has occurred 5 times let us check 5 here two, where three, four, four or five, five, so 5 times 5 appeared here therefore, the frequency of 5 is 5, and here I put the node 5 and marked that value of that node to be 5. Now, we create internal nodes by joining any two nodes whose sum is a minimal value. I can join these two, I could have joined one and one; I could have joined one the sorry I mean sorry I am actually joining minus 9 and 144. I could have joined minus 9 and minus 3, but that would have really lead to 5. If I join these two minus 9 minus 144 my frequency is coming to be 2.

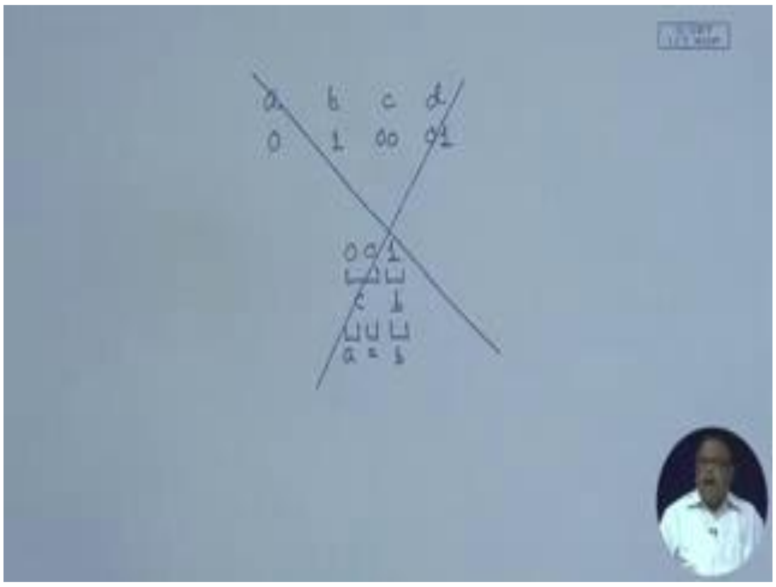
In that way at every stage I select the nodes and paired them up, so that their sum that internal node value is the minimal for that stage. And there I repeat this until I get the complete binary tree. So, I get the binary tree. I once again repeat how I got the binary tree. I take the leaf nodes. The leaf nodes are marked with their frequencies. So, I pair up two nodes and make an internal node such that that is a minimal combination. So, 2 and 1, I did not combine, but 1 and 1 I combine, because 1 and 1 gives me 2; 2 and 1 would have given me an internal of 3. Now, here I have got 2 2 5 I did not combine 2 and 5, I combine 2 and 2, I do 4. In that way I go on traversing go on combining until I get a complete binary tree.

Now, once I have got the complete binary tree then my task is to generate the code. Now, what has happened here is I will start from the root to the leaf to get the pixel value of the leaf. Now and as I traverse as I go to the left, it will be a 0, I will append a 0; as I go to write, I will append the 1. So, what happens I want to get the code of minus 1 here there is 1 minus 1. And why do I take the code of minus 1, because minus 1 is the more frequent and you can see that, here it is more frequent.

So, I come here. So, in order to start in order to reach minus 1 starting from the root, I have to traverse left. Once I traverse left, I take a 0. And then I traverse left again, I get this, so 0 0. Next node is 0. How do I get 0? 0 is somewhere here. So, I have to travel starting from here right that means 1 then I have to traverse left - 0 1 0, then again left - 1 0 0, so I put 1 0 0 there. Then minus 2, where is minus 2, minus 2, I will have to traverse right - 1, right - 1, left - 1 1 0. Consequently, in that way I go on.

Consequently, what is happening? You see for 14 I have got getting the maximum number of bits maximum number of bits. So, 14 is least frequent. So, something that is least frequent is being given more number of bits, because I want to capture more information there. And the number of bits and the amount of information is coupled the entropy relationship minus p i log p i. So, in that way, Huffman code is generated. Besides that there is another beauty in the Huffman code, it is also known as the prefix code, because no two codes will have the same prefix.

(Refer Slide Time: 33:31)

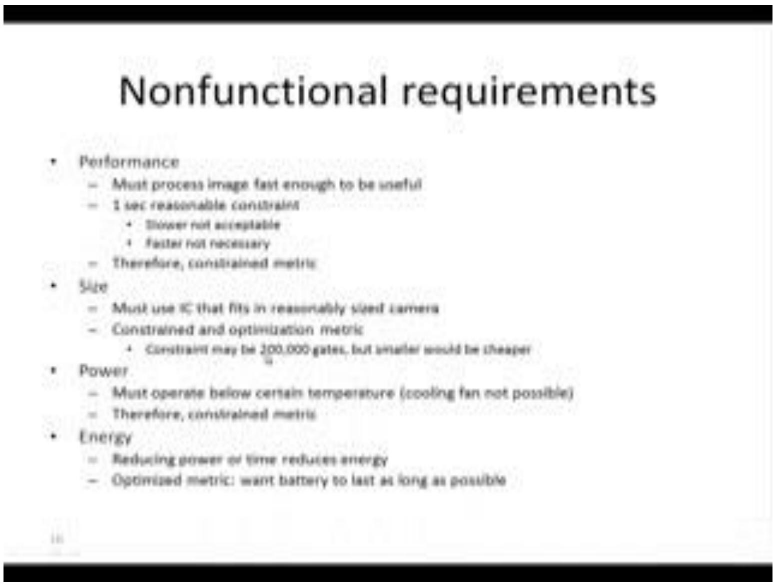


So, a, b, c, d if I had, and if I had given 0 1 00 01. Now, you see the code for c has got the code for a as the prefix. So, if I get some 0 0 1, I have got an ambiguity problem here. It can be c, b, it could be a, a, b and so and so forth a d many thing should have come, but that problem is not appearing in the case of Huffman coding. So, purely this is a unique encoding that we get and the other property of this encoding is that I am associating larger number of bits for the low frequency high information component.

And here say 144 is a very high frequency, sorry high information 144 is appeared once. So, what is the code for 144 – 1 1 1 1.

See the number of bits over here all these, which are high frequent minus 9 was 1. So, all these you see the beauty is all these which are having the same frequency have been given the same number of bits, so that is the Huffman coding. So, once again I repeat what we are doing, we are capturing the image compressing the image using DCT, I am quantizing them then serializing them, and I am encoding them. Now, in the next lecture, we will see how these functions can be captured through a design we have to implement it to an embedded system design.

(Refer Slide Time: 35:52)



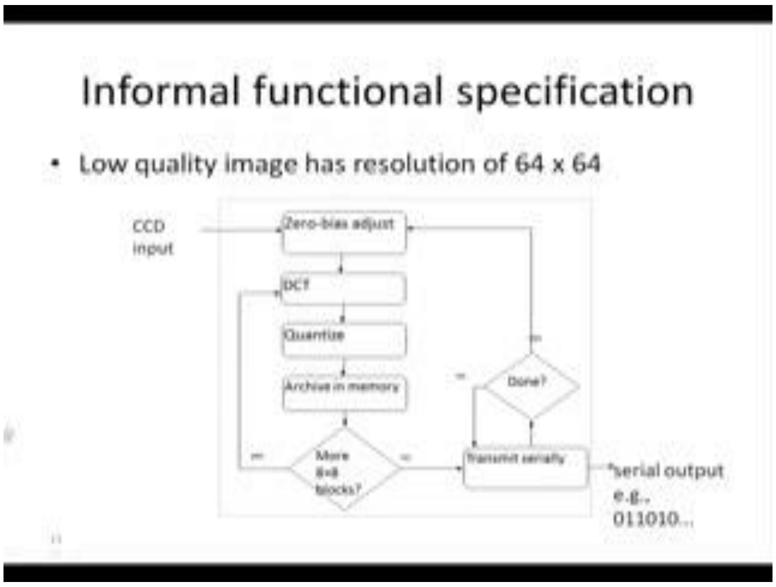
Nonfunctional requirements

- Performance
  - Must process image fast enough to be useful
  - 1 sec reasonable constraint
    - Slower not acceptable
    - Faster not necessary
  - Therefore, constrained metric
- Size
  - Must use IC that fits in reasonably sized camera
  - Constrained and optimization metric
    - Constraint may be 200,000 gates, but smaller would be cheaper
- Power
  - Must operate below certain temperature (cooling fan not possible)
  - Therefore, constrained metric
- Energy
  - Reducing power or time reduces energy
  - Optimized metric: want battery to last as long as possible

So, just to start with we are going to start a design in the next class, but before that let us look at what are the nonfunctional requirements. All those functions whatever we have talked about CCD, image compression, encoding those are functionally those have to be done. But what are the nonfunctional requirements, the performance is that we must process the images fast enough to be useful. The customer or the marketing group tells me that one second should be a reasonable constraint. Slower is not acceptable faster is not necessary for say low end products I do not need any more any faster. Therefore it is constraint metric, it is constrained. You can make it less than 1 second, but need not, but more than 1 second is not allowed.

The size is must use I see that fits in reasonably sized camera constraint and optimization matrix, because it is constraint, but I need to optimize because the less number of space it takes the bit eighties. So, constraint maybe around 200,000 gates, but smaller would be cheaper smaller would be cheaper it is therefore, this is an optimization constraint. Next comes the power - must operate below certain temperature, therefore that is the constraint some temperature will be given cooling fan is not possible, therefore again a constrained metric. And energy is reducing power we have to reduce the power or time. So, the energy is reduced this is an optimized metric, because we want the battery to last as long as possible. So, these are the nonfunctional requirements.

(Refer Slide Time: 37:59)



Informal functional specification

- Low quality image has resolution of 64 x 64

And typically the informal functional requirements for the camera would be take the CCD input, do zero-bias adjustment, do discrete cosine transform quantize archive in memory more 8 by 8 blocks, yes, go and fetch them do the same thing; otherwise the images is done you transmit serially that is the informal functional specification. Next, tomorrow we will see how we proceed with the design, and then we will show how the hardware-software trade off takes place.

But in order to do that we have to understand how the system works. You can be a, but that is not advisable that you are an embedded system designer, you are designing the thing, but you do not know what this thing is and how this is being used, because you have to take the optimization decisions, you have to take the hardware-software partitioning decisions. So, you must know the application thoroughly well. So, that is how we will today explain the application, next class we will be discussing on the design.