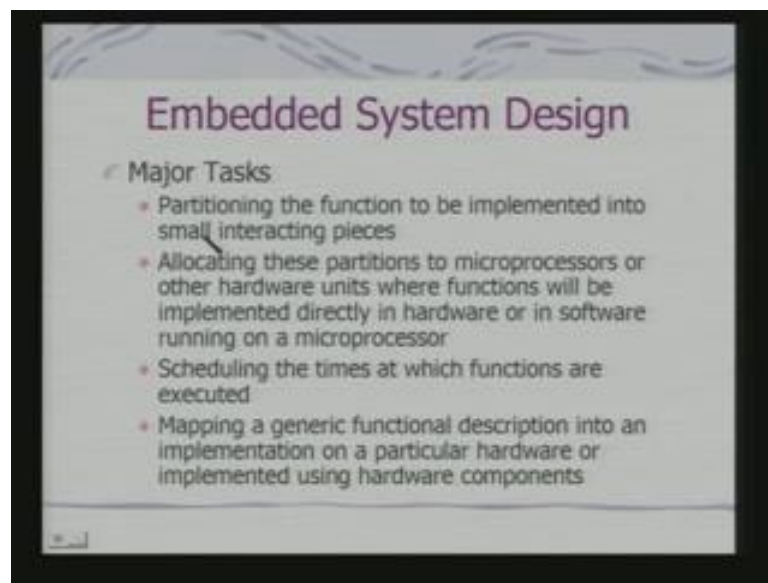


**Embedded Systems**  
**Dr. Santanu Chaudhury**  
**Department of Electrical Engineering**  
**Indian Institution of Technology, Delhi**

**Lecture - 31**  
**Embedded System Design – IV**

We were discussing, the different techniques been used for Embedded System Design. In the last class, we have looked at task graph. And how high level transformations can be applied to that task graph, to make it more amenable for hardware software co designing.

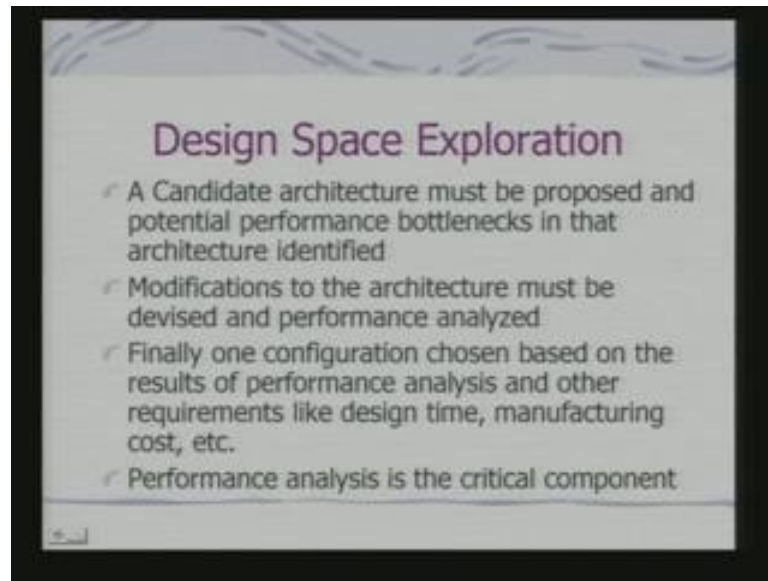
(Refer Slide Time: 01:24)



Today, we shall continue our discussions on hardware software co design. So, the major task to recapitulate; we know or that of partitioning the system function into small interacting pieces. Then, allocating these partitions to microprocessors. That means a pure software implementation or to other hardware units. That means we are using a dedicated hardware for implementation of these elements.

Then, scheduling the times at which this functions are executed. Then, it becomes a scheduling problem. But for doing all these things, we need to do mapping. Mapping a generic functional description into an implementation on a particular hardware or may be implemented using a set of hardware components. So, I need to have if I want to do this mapping. I need to know, what are the different possibilities? Depending on the possibilities, we shall do this mapping.

(Refer Slide Time: 02:23)

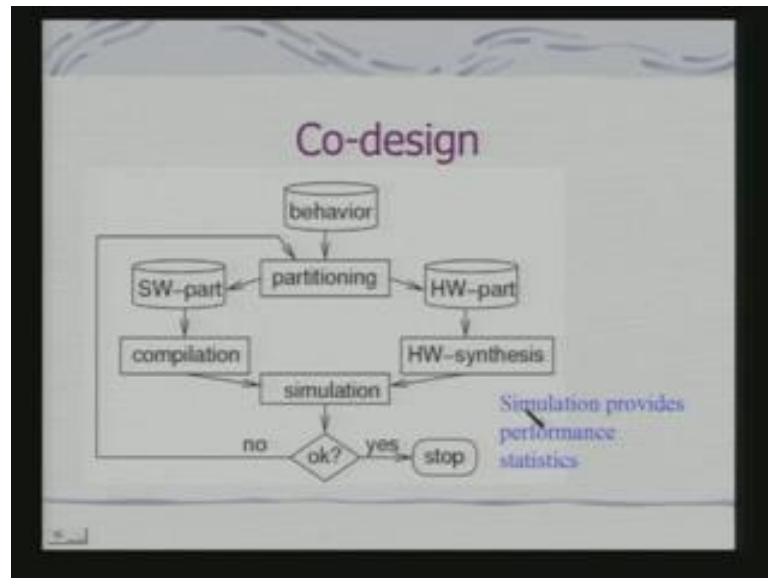


So, the design space exploration would be candidate architecture must be proposed. When you proposing candidate architecture. That means, we are making a choice among the possibilities of the target architecture in terms of processor. Also in terms of the hardware components which can be put together to make dedicated hardware. And can we shall find out the potential performance bottle necks in the architecture identified.

We shall do modifications to the architecture. Because, we would like and need to remove this bottle necks to meet performance constraints. Finally, we shall choose one configuration based on the results of performance analysis. And other requirements like design time, manufacturing cost, etcetera. So obviously, you can understand the performance analysis is the critical component.

Whenever we are proposing architecture, we need to evaluate that architecture on the basis of performance criteria. Now, this design space exploration can be done by actual implementing on trying out different alternatives. So, it process can be completely manual. Otherwise, we can also have automated schemes, formulating the problem as the formal optimization problem and then exploring the design space. We shall look at both these approaches in to this class.

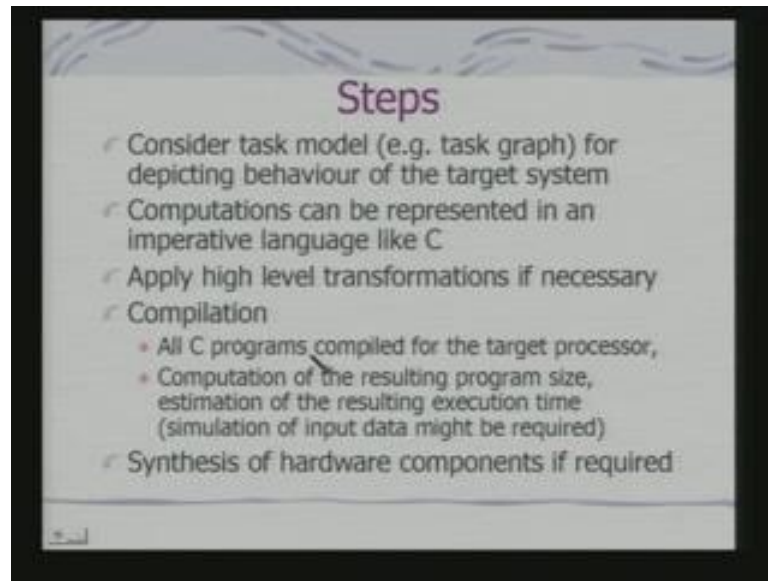
(Refer Slide Time: 04:01)



So, effectively what we are looking at is that, we have the behavior the task graph. We partition, we get software part hardware part. We can do the compilation, do the hardware synthesis. Then, do a simulation of the two things together. And that simulation can actually provide the performance statistics. If I find that, I am meeting my performance constraint.

Then, I can accept the design else I shall go back, try out alternate partitioning alternate architecture and repeat this process. So, these processes can be done through manual intervention or can be done in an automated fashion provided. We have enough statistics and computer aided tools to estimate the performance.

(Refer Slide Time: 04:54)



So therefore, the steps we shall involve that consider the task model. We have seen in the last class, a task model for depicting behavior of the target system. This task model can come from high level modeling, which can be represented through an UML notation. Then, we can come to the task model. And in the task model, the computations can be represented at each node in various ways.

One of the ways to represent, it could be using an imperative language like C. And apply high level transformations to this node or to the task graphs. We have already discussed the high level transformations in terms of computations. We have looked at different ways to deal with looks, different ways to the ((Refer Slide Time: 05:41)) looks. So that, we can have optimality in terms of cache access.

We have also looked at the possibilities of merging nodes or splitting nodes taking care of IO requirements. So, these are all high level transformations that we can apply on the task graph. And then, if we are looking at each compile at computation of each node. I can subject it your compilation. If I am doing this compilation and compiling it for the target processor effectively, we are doing as straight software implementation.

We can look at it in this way. If I am doing that way, then I can get the computation of the resulting program size, estimation of the resulting execution time. We might need to do a simulation of input data to do this estimation. And if required, we shall be designing the hardware components. In fact we can use system C or PHTL to represent the

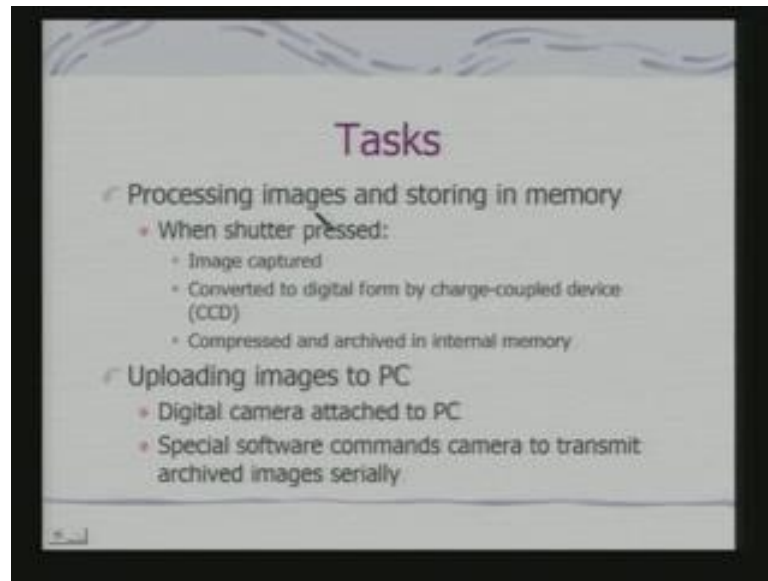
hardware specification of the nodes. That means hardware design of the nodes that is to be implemented. That specification can be a top level specification.

(Refer Slide Time: 06:57)



So, let us look at a simple example. A simple digital camera design, and through this we shall look at explorations of alternate design possibilities. Here, we are not talking about strictly and automated methodology for doing this alternate design exploration. So, design space exploration would be only on the basis of manual suggestions and trying out the different possibilities. We are not looking at a formal formulation of the search problem.

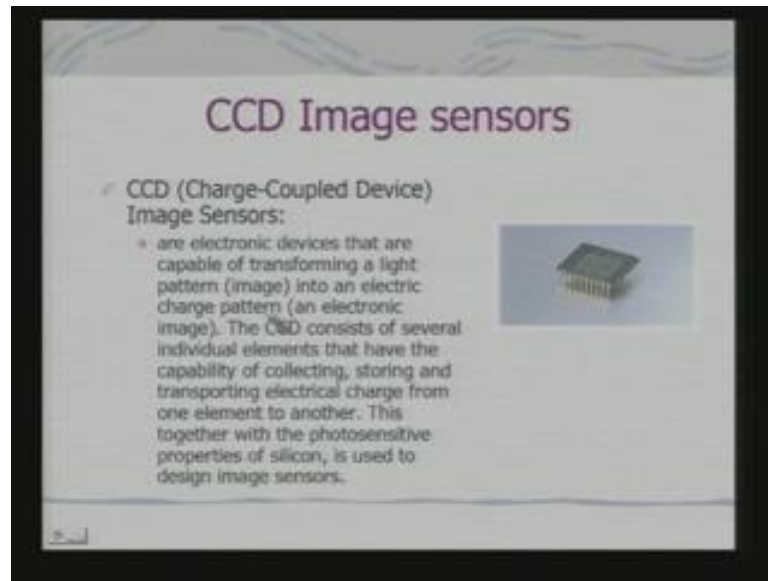
(Refer Slide Time: 07:36)



So, what are the tasks for a digital simple digital camera? A digital camera basic task is to process image and store them in memory. So, when the shutter is pressed the image gets captured. The image is converted to digital form by charge couple device. We are using the charge couple CCD sensors. And they are compressed and archived in internal memory. That is the memory internal to that of the camera.

We would also like the feature to support uploading of images to PC. So, the digital camera can be attached to pc via a very simple serial link. And you can have special software commands for the camera to transmit archived images serially. So, these are the basic tasks that had to be realized in a digital camera.

(Refer Slide Time: 08:35)



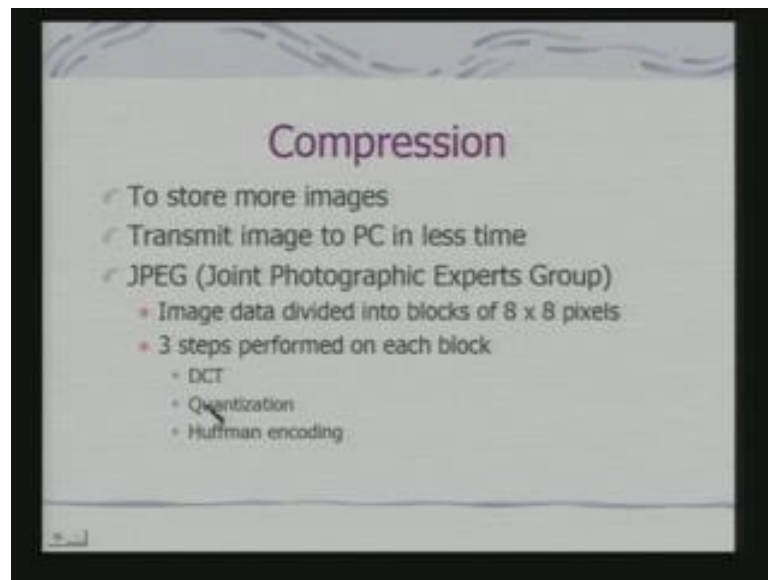
See let us look at the CCD image sensors. So, CCD is the Charge Coupled Device. They are electronic devices that are capable of transforming, a light pattern into an electric charge pattern; that is an electronic image. The CCD consists of several individual elements that have the capability of collecting, storing and transporting electrical charge from one element to other. The amount of electrical charge accumulated would depend on the intensity of incident elimination. That is, together with the photosensitive properties of silicon, is used to design image sensors.

And this CCD sensor typically arranged in the form of array, rows and columns. And from there, the data corresponding to the image is collecting it. So, each CCD site can correspond to a single pixel in your image. Now, when we are using a CCD there is one important issue, which we need to look at and considered, which is called zero bias error.





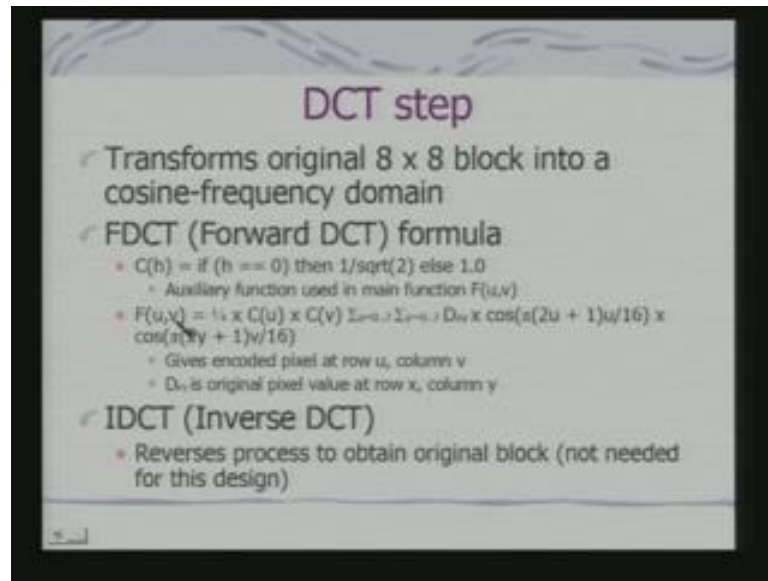
(Refer Slide Time: 11:15)



Then, the images have to be compressed. So, that they can be stored in the internal memory. The two motivations for this are to stored images and transmit image to PC in less time. Typically, what we used and what we are talking about here is the JPEG compression. You shall not go to the details of the JPEG compression.

But we shall briefly look at the states in order to understand the design. So, image data is divided into blocks of 8 cross 8 pixels. And three steps are performed on each block discrete cosine transform, quantization, Huffman encoding, which is nothing but, entropic coding.

(Refer Slide Time: 11:59)



So, how is the DCT done? In fact DCT is basically, that are transforming the data from the special domain to that of frequency domain. So, for each 8 cross 8 block, we apply the forward discrete cosine transform. The expression for the discrete cosine transform is this. So, this  $D \times y$  corresponding to pixel value. And what you get after transformation is  $F u v$ , which corresponds to the frequency domain representation.

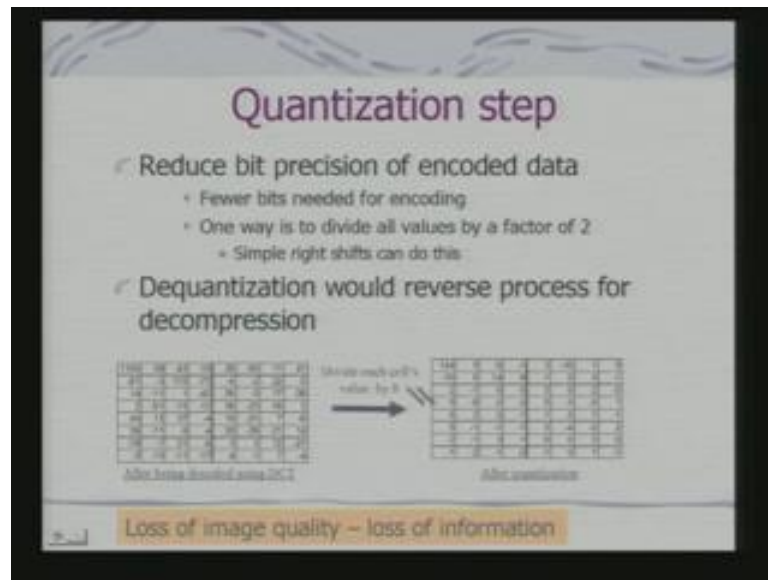
And this is actually we will find that, each of this elements is obtained over ((Refer Slide Time: 12:40)) over the intact 8 cross 8 block. So, when I have got  $F$  zero zero that basically means, I am looking at the DC component. And  $u$  and  $v$  corresponds to special frequencies in both iterations. Because, it is a two dimensional data. So, I shall have two frequency values. So once, we have these we get  $F u v$ .

Why we do this kind of a transformation? The whole idea is that, if I do this frequency domain transformation, I shall have the energy concentrated into few coefficients. See in an 8 cross 8 block, I have got 64 values. If I do a frequency domain transformation and consider for example, I am just considering an image block which consists of a single color. If I do a frequency domain transformation, then the energy would get concentrated for  $F$  zero zero and other values will be negligible.

So, I am left to it only one value rather than 64 values. So, that is the basic principle of energy concentration. And DCT provides being a good tool for concentrating the signal energy into few coefficients. That is why DCT is used in JPEG. When I would like to

decompress the image, I have to apply inverse DCT. The inverse DCT will give me back the original pixel values. Obviously, this computation you can see will take place in real domain. That means, it would really involve floating point calculations, if we do not get any special measure.

(Refer Slide Time: 14:30)



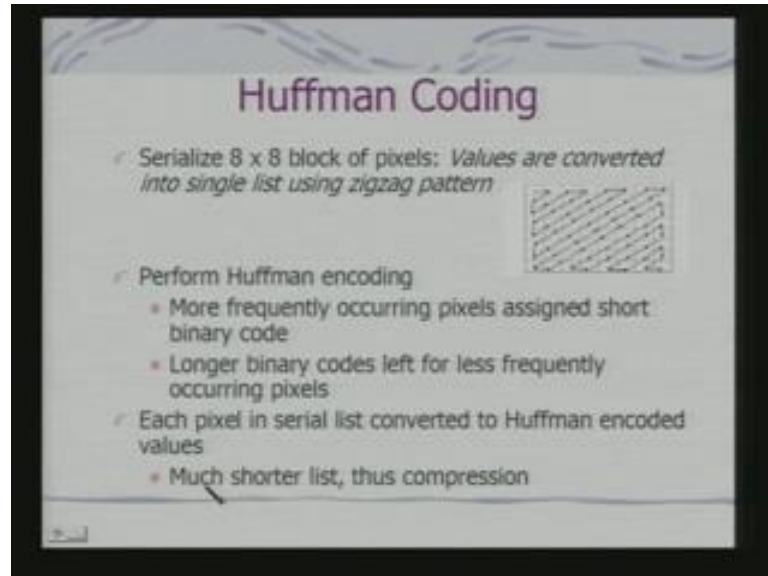
Next is quantization, because quantization reduces bit precision of encoded data. See if values are small, I can use a threshold and make them zero. If they are zero, then obviously I need not code them or store information about them. So, a very simple way is one way is to divide all values by a factor of 2. Simple right shifts can do this. If I divide by the value of 2 divide by a factor of 2, then what happens? The magnitude gets reduced.

Since the magnitude gets reduced, the number of bits that can be used for storing can be less. Here, I am giving you one example, that I am dividing each content of each cell by 8. Dividing the content of each cell by 8, this is the DCT value which have obtained and this is dividing by 8. If I divide by 8, you will see the range of values exchanging. Obviously I can use much smaller number of bits to represent these values.

This is the basic motivation for quantization step. And the lousiness of the image compression comes from quantization. Because of quantization, we are losing information. So, it is the glossy compression scheme that we are talking about. So, you can understand that this quantization would also involve computation, at least a division

operation. And I would like to do may be a divisions by factors of 2. Simplify because, that can be achieved by simple shift operations.

(Refer Slide Time: 16:10)



Then, we do Huffman Coding. For Huffman Coding, what we do? Serialize 8 cross 8 block of pixels. Values are converted into single list using a zigzag pattern. In fact the values are read in this pattern. Why it is done in this way? Because, if you look it look at it, see these coefficient and these coefficient are conceptually similar in nature. Because, this is representing the first component in x direction this is representing in y direction.

So, depending on the frequency content of the image, you can consider that, if there are no changes in x, there may be very low changes in y as well. So, these two coefficients may have similar values. And as I am moving ahead, I am actually considering coefficients corresponding to greater frequency. So, it may so happen. That in an image, if you see any natural phenomena if I am taking an image of natural external world, the changes are smooth changes are not really sharp, if we are not taking images of artificially generated patterns.

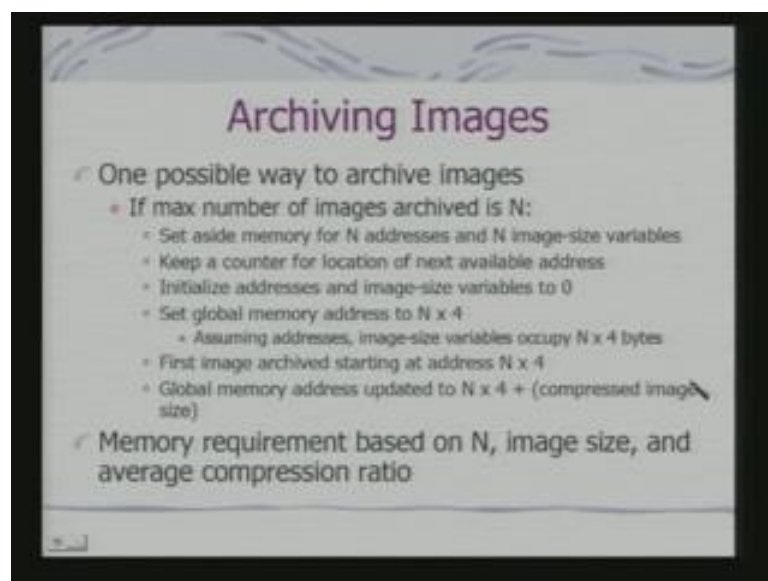
So, its moral less smooth major part it will be smooth. So, high frequency values can actually turn out to be very small. And through a quantization process, those high frequency values may become actually zero. So, then you do once you have therefore the sequence of these values, where many of them has zero because of quantization you

perform Huffman coding. In fact, you group them together to form symbols and form Huffman coding.

So, more frequently occurring pixels there are assigned short binary code. Because, that is the basic principle of Huffman coding and entropic coding and longer binary codes for less frequently occurring pixels. Because, depending on the entropy that codes are associated. So, each pixel in serial list is converted to Huffman encoded values much shorter list and thus compression.

Because, we are not using same number of bits for representing each and every pixel value. We are using different number of bits depending on probability of occurrence of that pixel value. In fact, actually you do not use individual pixel values, but you actually look at the group of this value for the purpose of Huffman coding. So, this is another computational task.

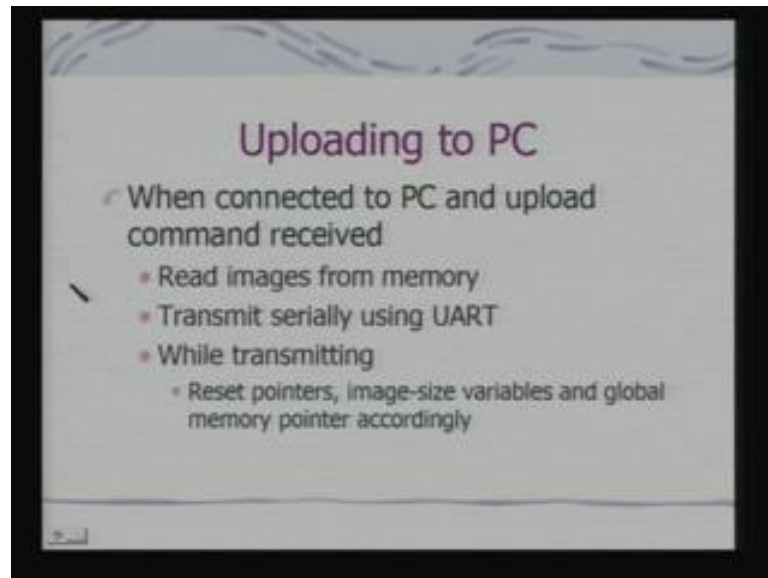
(Refer Slide Time: 18:47)



So, next if I have done the compression what shall I do? I shall be archiving the image. I would like to store the image. So, storing the image would be in the memory. And I can keep global memory address assigned and reserved for the different memories. So, we can set aside memory for N addresses and N image size variables. And the image in the image memory requirement would be based on N that is the image size and the average compression ratio.

So, that would give you the number of bytes required for storing an image and the average. So, you actually initialize the addresses and image size variables. These are the basic algorithm which will be involved. Assuming addresses image size variables occupy N into 4 bytes, allocating 4 bytes. And first image is archived starting at address N into 4. And subsequent will be at the offset of compressed image size.

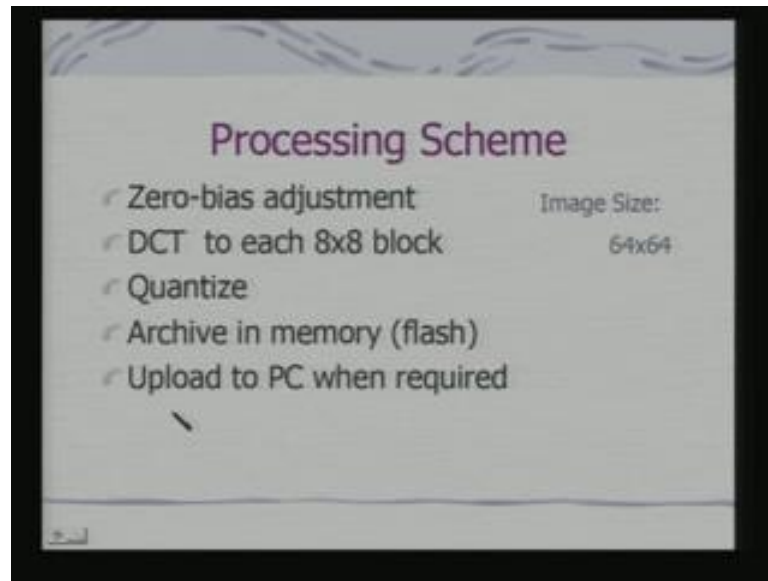
(Refer Slide Time: 19:58)



Therefore, once you have stored these images depending on the requirement, we shall be connecting it to a PC and upload command. So, read images from memory. So, that means it will transmit serially using UART. While transmitting, you need to reset pointers, image size variables and global memory pointer accordingly. Because, you have to keep track that you have finished transmitting a single image.

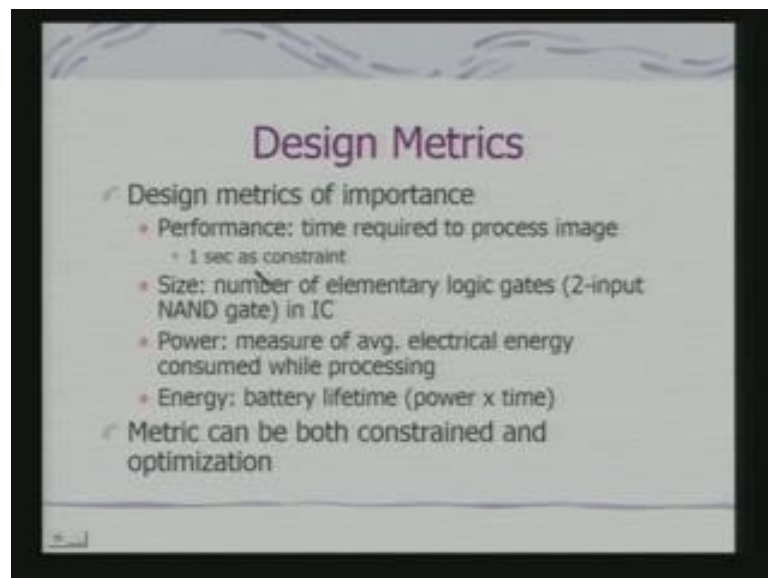
So, image after image is to be transmitted sequentially and the pointer adjustment have to be done by the software. So therefore, if we now summarize if we now remember the task graph, what are the different steps involved in a task graph?

(Refer Slide Time: 20:37)



Zero bias adjustment, DCT, Quantization and Entropic Coding, Quantization and Entropic coding, Archiving in memory and Uploading to PC, when required. These are the basic steps of the nodes in your task graph. And what I need? I need to map them to the processor or to dedicated hardware, depending on my design constraints. I have just specified the task. I would have not really looked at yet the design constraints. I have just considered one possibility, that image size. Let us considered 64 cross 64.

(Refer Slide Time: 21:27)



So, what are the different kinds of design metrics which would be feasible and possible? The most important is performance. The time required to process image is 1 second and these should be constraint. What is that mean? That means, I can not choose a possible architecture, while the time taken to compressed and stored a image would be greater than 1 second. Such architectures have to be rejected.

So, this is the constraint. The other design metrics are size, the number of elementary logic gates, 2 input NAND gate in IC. So, number of logic gates would determine the area or the size of the chip which is required. So, this is what we want to optimize. We do one to optimize power measure of average electrical energy consumed, while processing. And energy is battery lifetime that is power into time. What is this time? This time is the time taken to process an image.

So, that is the energy. We would like to optimize size, power and energy. And this is the constraint, which I might satisfy. So, you can see that actually exploration in a design space is therefore, searching a solution which meets the constraint and optimizes the parameters that we are looking for. So, what we say the metric can be both constrained and optimization. So, here I have got one constrained parameter and remaining a some kind of optimization parameters.

(Refer Slide Time: 23:05)

**Implementation 1: Microcontroller alone**

- ✓ Example : Intel 8051 microcontroller
- ✓ Total cost small
- ✓ Well below 200 mW power
- ✓ Time-to-market about 3 months
- ✓ However, one image per second not possible
  - 12 MHz, 12 cycles per instruction
    - Executes one million instructions per second
  - Reading out from CCD array require nested loops resulting in 4096 (64 x 64) iterations
    - ~100 assembly instructions each iteration
    - 409,000 (4096 x 100) instructions per image
    - Half of budget for reading image alone
  - Would be over budget after adding compute-intensive DCT and Huffman encoding

Let us look at possible Implementations. And I said at we are looking at a manual exploration of the design space. So, when we can use microcontroller alone. Let us take



an example of Intel 8051 microcontroller. What is the motivation for choosing this architecture? The total cost small, well below 200 mega watt of power consumption, time to market will be less because, you need to just simply write software.

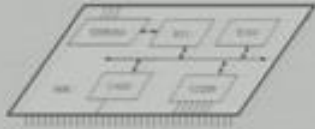
So, these are an assumption. However, look at these possibilities. One image per second is not possible. Because, it typically works 12 megahertz with 12 cycles per instruction. So, it will execute one million instructions per second. Now, reading out from CCD array required nested loops, resulting in 4096 iterations of the order of 100 assembly instructions in each iteration.

So, you have to execute 4096 into 100 instructions per image which consumes the half of the budget, because your budget was 1 second. Because, time constraint was 1 second. So, half of the budget is getting consumed, would be over budget after adding computes intensive DCT and Huffman encoding. So obviously, this is not a feasible or acceptable architecture.

So, we can see that if I simply, take the C code corresponding to the computations, subject it to a compiler which we generate 8051 code. And estimate the runtime of the code define, that the constraint cannot be met. So, if the constraint cannot be made I have to look at alternate possibilities. So, implementation 2 could be an SOC approach, Microcontroller and Dedicated Hardware.

(Refer Slide Time: 24:53)

**Implementation 2: SOC Approach  
Microcontroller and Dedicated Hardware**



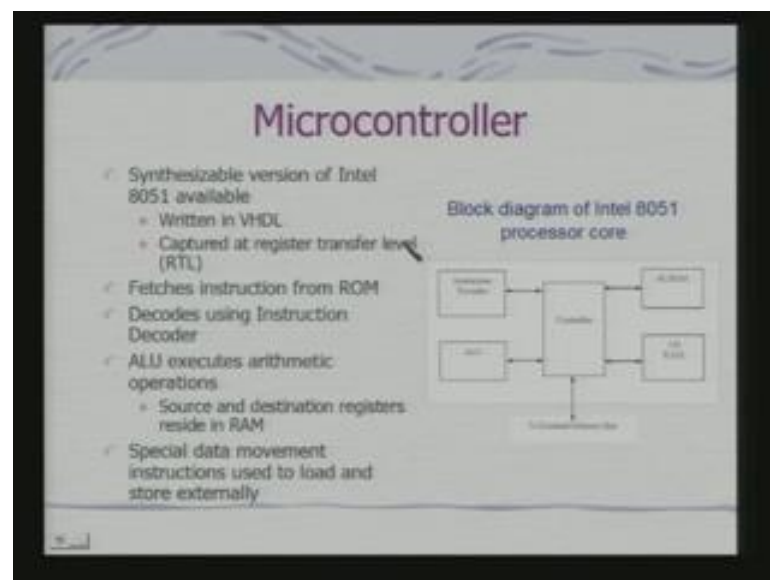
- ✓ CCDPP (image read) function implemented on custom single-purpose processor
  - Improves performance – less microcontroller cycles
  - Easy to implement
    - Simple datapath
    - Few states in controller
- ✓ Simple UART easy to implement as single-purpose processor also
- ✓ EEPROM for program memory and RAM for data memory added as well

Here, we are talking about say image read function, CCD read function implemented on a custom single purpose processor. It improves performance less microcontroller cycles, easy to implement because it just leads to do what? Read and Subtract. Simple data path, so that such a processor would have a very simple data path and few states in controller. In fact, the whole thing can be represented by a simple FSM and what do you need?

You need an implementation of 30 FSM in a dedicated hardware. Then, also we can also use UART that is Universal Asynchronous Receiver Transmitter as a single purpose processor. It should have along with it EPROM for program memory and RAM for data memory added as well. In fact, that can provide the storage for the intermediate processing and EPROM will have the program.

So, this is a kind of an SOC based approach that we are talking about. Here, we have 8051. We are putting in an UART here. We are putting in a special purpose processor which will communicate with the CCD and do the adjustments. This is a RAM and this is the EPROM. EPROM is the Programmable RAM which is called program to run the system.

(Refer Slide Time: 26:23)



So, how to do that design? In fact what we have the design is to develop the SOC? In fact, the other way could be using an 8051 is the discrete component and using all these as an external devices. What we are looking at? We are looking at a possibility of implementing a SOC. Now, if we are doing that, then we got to have the specification of

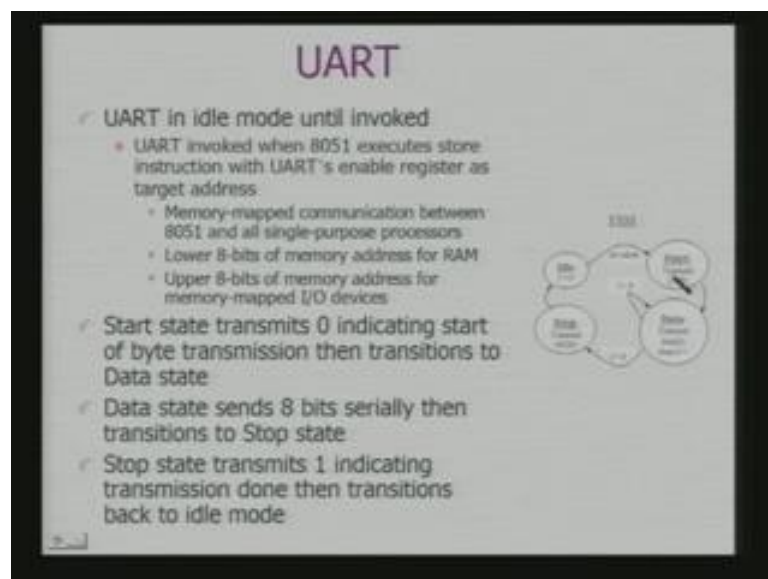
the processor also in a core form soft core form. I have talked about the whole design approach based on the soft core.

See if it is available in a soft core then 8051 specification, it should be possible for me to have in VHDL and captured at register transfer level. This obviously, would specify the complete architecture and its operations. So, we have already seen VHDL. So, basically I shall have description of the processor in terms of the VHDL code. So, it has got the controller, the ALU, instruction decoder, 4 k ROM and 128 RAM. This is an internal memory.

So, the whole description of the processor and how the processor works have to be captured through your VHDL code. Now, you can buy if we are talking about a soft processor code and somebody selling it out licensing out the core. It means, these VHDL code is being supplied to you. If the VHDL code is supplied to you, you can anywhere modify the code depending on your requirements.

And when you are designing an SOC, what is it required? I shall write the VHDL code for the additional hardware components. We shall also to specify the bus interconnect structure. We shall also specify the memory which can be used by both this 8051 as well as your other dedicated hardware.

(Refer Slide Time: 28:21)

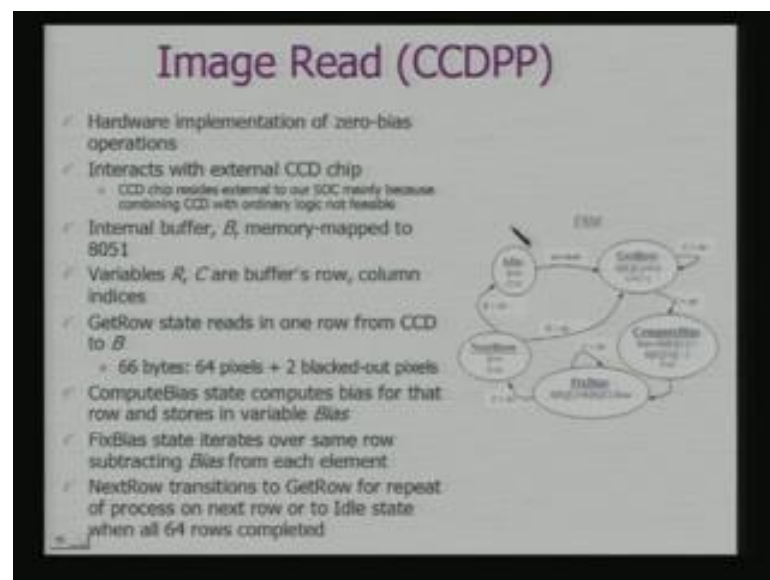


So, first let us look at an UART. So, these are the different states that UART can have, it is originally in an idle state. Now, when it is invoked that is when the transmission has to be done, its starts transmission. And typically, if you are familiar with RS 232 C kind of a protocol, what it will do? It will have a start bit bring the line low. So, this is a start. Then, it transmits the data depending on the data format.

So, it is said 8 bit data has been transmitted. Then it will go to the stop bit. And again go back, when it is transmitting a sequence of such bytes. So, this is a basic FSM which needs to be implemented with provision for appropriate data buffer registers, to store the data as well as to transfer the data. So, effectively what we are telling is that, I can implement this FSM, but how should it be interface to the 8051.

So, we are using memory mapped communication between 8051 or and all single purpose processors. What is that mean? The register of this processor would be the memory map of 8051. And lower 8 bits of memory address is for RAM, upper 8 bits of memory address for memory mapped IO devices. So, this is available to me because, now I can add the VHDL code corresponding to this UART implementation.

(Refer Slide Time: 29:55)

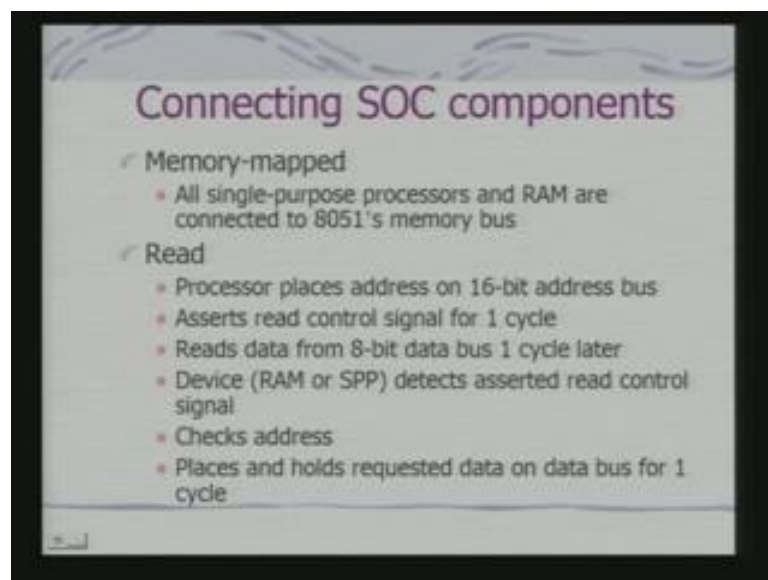


Similarly, the image read can be done in a similar way. So, we have got an internal buffer  $B$  which is memory mapped to 8051. The variables  $R$   $C$  are buffers row and column indices. Get row state reads in one row from CCD. So, if it is reading in one row

from CCD, it will get 64 pixels 2 blocked out pixels. The compute bias computes bias for that row and stores in variable bias. If there are 2 pixels, it would be an average.

And Fix Bias iterates over the same row subtracting bias from each element. And next row transitions to get row for repeat. So, this becomes effectively the FSM to do bias error adjustment on the memory read. So, this can be again mapped into VHDL. And I can have a dedicated hardware doing this function. And how does it interact to the external CCD and mainly because combining CCD with ordinary logic not feasible. So, that will communicate via this buffer.

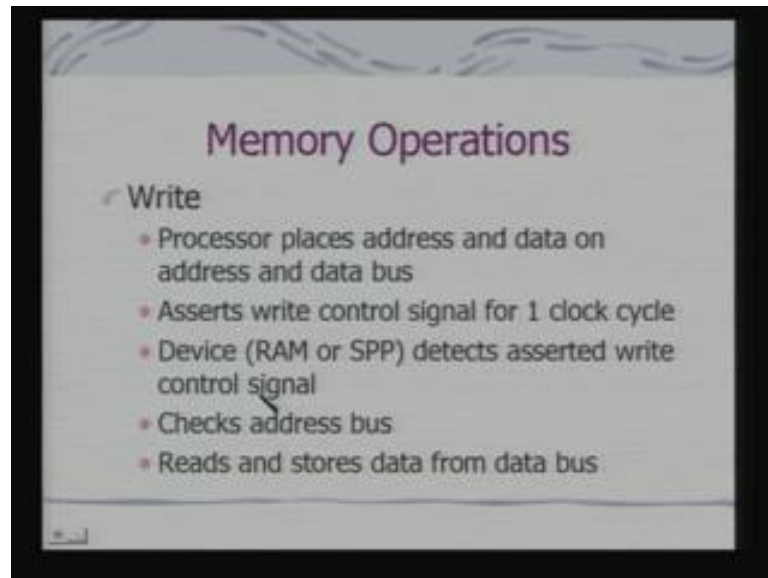
(Refer Slide Time: 31:03)



Now, we need to what we have got? We have got UART. We have got the special purpose processor to do bias error adjustments. Now, we need to connect this SOC component. In fact connection of SOC components is basically the glue logic using which the different components can get connected. So, all single purpose processors and RAM are connected to 8051 memory bus. So, what is the basic read operation?

The processor places address on the 16 bits address bus, asserts read control signals, reads data from 8 bit data bus 1 cycle later. The device can be RAM, a special purpose processor. Because, special purpose processor also memory mapped with respect to 8051 detects asserted read control signal, check address, places and holds requested data on data bus for 1 cycle. So, this is the glue logic which has to also going to the special purpose processors, which have been implemented.

(Refer Slide Time: 32:04)

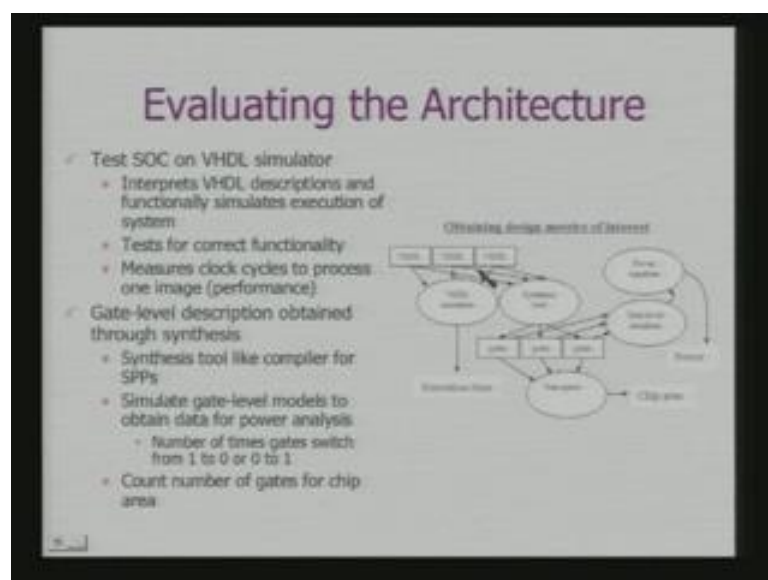


### Memory Operations

- Write
  - Processor places address and data on address and data bus
  - Asserts write control signal for 1 clock cycle
  - Device (RAM or SPP) detects asserted write control signal
  - Checks address bus
  - Reads and stores data from data bus

Similar thing will be done for write. The processor places address and data on address and data bus. Asserts control signal, the device detects asserted write control signal; checks address bus, reads and stores data from the data bus. So, this becomes the basic protocol, if you see why this becomes important? This defines a basic protocol for the special purpose processors to communicate with microcontroller 8051.


(Refer Slide Time: 32:33)



### Evaluating the Architecture

- Test SOC on VHDL simulator
  - Interprets VHDL descriptions and functionally simulates execution of system
  - Tests for correct functionality
  - Measures clock cycles to process one image (performance)
- Gate-level description obtained through synthesis
  - Synthesis tool like compiler for SPPs
  - Simulate gate-level models to obtain data for power analysis
    - Number of times gates switch from 1 to 0 or 0 to 1
  - Count number of gates for chip area

Obtaining design metrics of interest



```
graph TD
    VHDL[VHDL description] --> Simulation[Simulation]
    Simulation --> Synthesis[Synthesis]
    Synthesis --> GateLevel[Gate-level models]
    GateLevel --> Metrics[Obtaining design metrics of interest]
    Metrics --> CPU[CPU]
    Metrics --> RAM[RAM]
    Metrics --> SPP[SPP]
    Metrics --> Power[Power analysis]
    Metrics --> Clock[Clock cycles]
```

So, next question is comes that of evaluating this architecture. So, what I have got? I have got my processor in VHDL. I have got the special purpose processors also written

in VHDL. We have also defined, how they can communicate with each other. So, I have got the complete design of SOC in a VHDL form. So, Test SOC, so we can test SOC on VHDL simulator. We need not actually fabricate the system, which interprets VHDL descriptions and functionally simulates executions of the system.

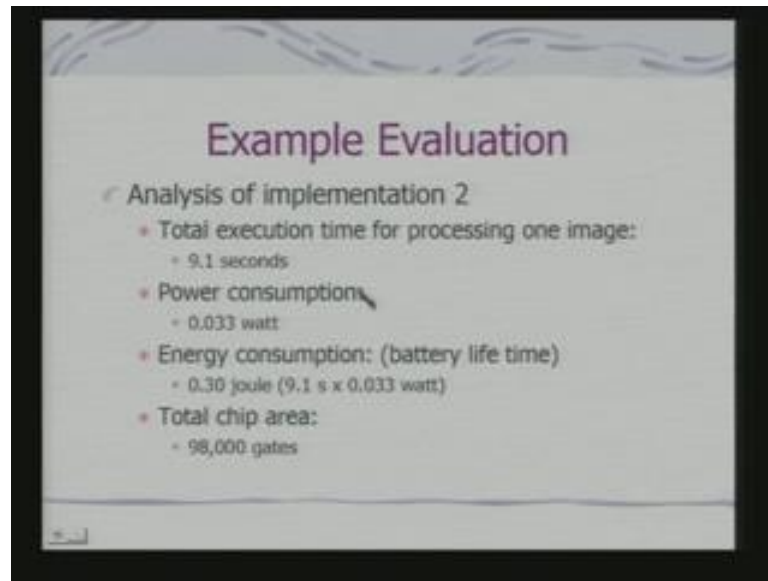
So, on the simulator one job is to test for correct functionality. That means, whether I have written the VHDL code correctly. And it can also measures clock cycles to process one image. So, this number of clock cycles would give you an indication of the performance. Next we can go to the gate level design, when you go to the gate level design, we are going through a processor synthesis.

When we are at register level design, we can get the clock cycles. But further down, we can get the gates through the processor synthesis. Once we have the gate, so what we can do? Simulate gate level models to obtain data for power analysis. There are various power estimation techniques. We can try to use some of them, to get an estimate of the power.

So, the number of times a very simple way of estimating is the number of time the gate switches from 1 to 0, 0 to 1. If you remember when we discussed this power our architecture we said, that this transitions is the source of power consumption. So, if I simply count this, this gives an estimate of power not accurately the power consumption. And you count the number of gates for chip area.

So, I can get therefore through this simulation and synthesis tools and estimation of the performance of the system. Obviously, this is the very simplify picture, whole all these performance estimation measures and how they have to be implemented at complex task. But, in principle we have tools to do this kind of estimation. So, once we have the estimation we can evaluate the architecture.

(Refer Slide Time: 34:46)

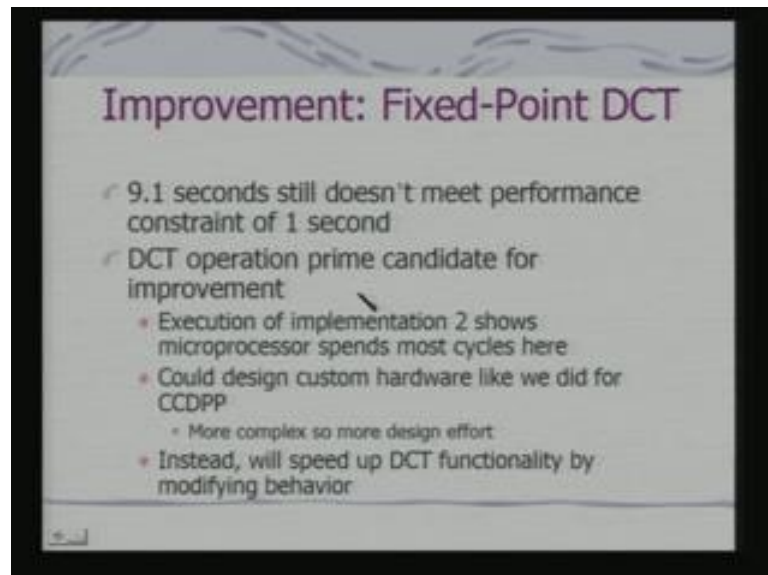


**Example Evaluation**

- Analysis of implementation 2
  - Total execution time for processing one image:
    - 9.1 seconds
  - Power consumption
    - 0.033 watt
  - Energy consumption: (battery life time)
    - 0.30 joule (9.1 s x 0.033 watt)
  - Total chip area:
    - 98,000 gates

So, what is the evaluation? We find the total execution time for processing one image trans out to be 9.1 seconds. Power consumption is .033 watt. Energy consumption is .3 joule because I am multiplying this and total chip area is this many gates. I am violating this constraint and hence this design is also not acceptable.

(Refer Slide Time: 35:12)



**Improvement: Fixed-Point DCT**

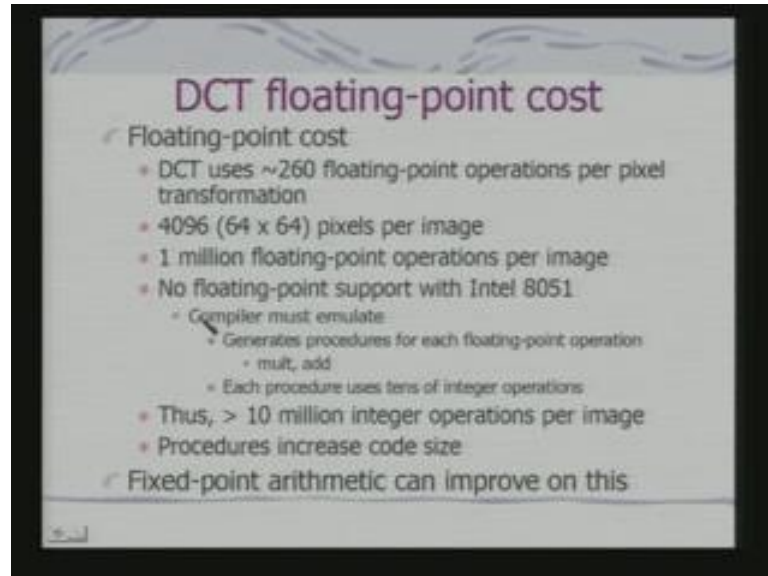
- 9.1 seconds still doesn't meet performance constraint of 1 second
- DCT operation prime candidate for improvement
  - Execution of implementation 2 shows microprocessor spends most cycles here
  - Could design custom hardware like we did for CCDPP
    - More complex so more design effort
  - Instead, will speed up DCT functionality by modifying behavior

So, what we do the next option is, the next option is to look at the DCT. If you remember we said one high level transformation was to go from floating point to fix point representation, in the task graph. So in fact, if you do a profiling we shall find the DCT



operation is the takes major time. And we could design custom hardware, it is more complex. So, design effort let us we can therefore try out alternate possibilities.

(Refer Slide Time: 35:46)



**DCT floating-point cost**

- Floating-point cost
  - DCT uses ~260 floating-point operations per pixel transformation
  - 4096 (64 x 64) pixels per image
  - 1 million floating-point operations per image
  - No floating-point support with Intel 8051
    - Compiler must emulate
      - Generates procedures for each floating-point operation
        - mult, add
        - Each procedure uses tens of integer operations
  - Thus, > 10 million integer operations per image
  - Procedures increase code size
- Fixed-point arithmetic can improve on this

In fact DCT uses 260 floating point operations per pixel transforms. And in fact 8051 does not support floating point calculations. So, that has to be taken care of by the compiler. Compiler needs to generate instructions to do the floating point calculations, using basically integer operations. So, each procedure uses 10s of integer operations. In fact at estimate is greater than 10 million integer operations per image of size 64 by 64 and the procedures increase code size. So, other possibility is, whether we can transform it into a fixed point representation.

(Refer Slide Time: 36:30)

**Fixed-point arithmetic**

- Integer used to represent a real number
  - Constant number of integer's bits represents fractional portion of real number
    - More bits, more accurate the representation
  - Remaining bits represent portion of real number before decimal point
- Translating a real constant to a fixed-point representation
  - Multiply real value by  $2^n$  (# of bits used for fractional part)
  - Round to nearest integer
  - E.g. represent 3.14 as 8-bit integer with 4 bits for fraction
    - $2^4 = 16$
    - $3.14 \times 16 = 50.24 \approx 50 = 00110010$
    - 16 ( $2^4$ ) possible values for fraction, each represents 0.0625 ( $1/16$ )
    - Last 4 bits (0010) = 2
    - $2 \times 0.0625 = 0.125$
    - $3(0011) + 0.125 = 3.125 \approx 3.14$  (more bits for fraction would increase accuracy)

There may be lack of precision, but when we are representing an image that lack of precision may not be critical enough. Because, in any case we are doing one digestion. So, how do you do fixed point arithmetic? In fact, what we do, we use fixed number of bits in the word to represent the fractional part of the binary number. So, if we put the real value by 2 raise to the power, number of bits used for the fractional part we are actually getting the representation.

So, this is the simple illustration only if you know this, that we can represent 3.14 as 8 bit integer with 4 bits for fraction. So, effectively if it is a 4 bits for fraction, then why I multiplying this? Because, here each bit is representing actually .0625. It will be 1 by 16. So, I am multiplying the value by 16, to get the actual integer representation.

So, that we can work with fixed point, so what we are getting effectively since this is the precision. We are getting depending on the number of bits we are using. So, that introduces approximation. So, 3.125 are actually representing 3.14. So, this is an approximation.

(Refer Slide Time: 37:52)

**Fixed-point arithmetic operations**

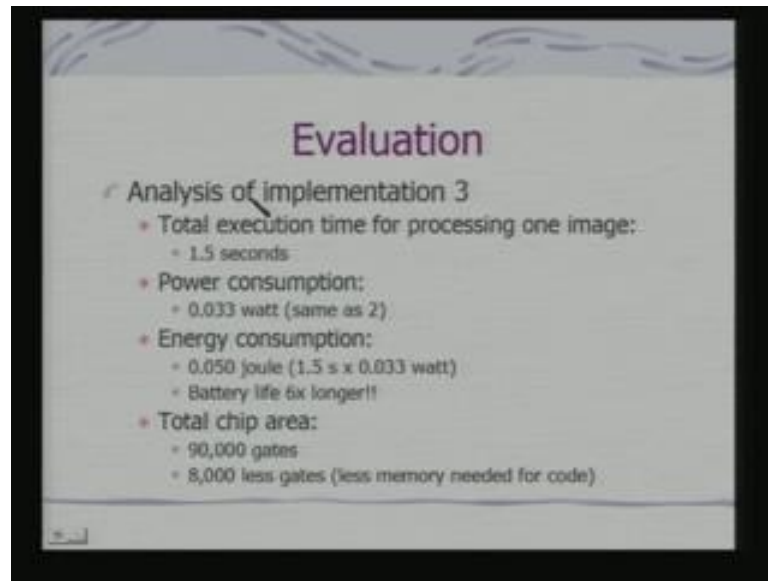
- Addition
  - Simply add integer representations
  - E.g.,  $3.14 + 2.71 = 5.85$ 
    - $3.14 \rightarrow 50 = 00110010$
    - $2.71 \rightarrow 43 = 00101011$
    - $50 + 43 = 93 = 01011101$
    - $5(0101) + 13(1101) \times 0.0625 = 5.8125 = 5.85$
- Multiply
  - Multiply integer representations
  - Shift result right by # of bits in fractional part
  - E.g.,  $3.14 \times 2.71 = 8.5094$ 
    - $50 \times 43 = 2150 = 100001100110$
    - $\gg 4 = 10000110$
    - $8(1000) + 6(0110) \times 0.0625 = 8.375 = 8.5094$
- Range of real values used limited by bit widths of possible resulting values

But what is the advantage? Advantage is, now I can do say operations like addition by simply add integer representations. So, this integer representation shows you, that when the result is 5.85 I get 5.8125. Because, here these bits are integer, remaining are the fractional bits. Even the multiplication can be done through integer operations. So, you shift result right by the number of bits in fractional part.

So, effectively what we get is these results. Because, that many shifting has to be done this points on to adjustment of the binary point. So, range of real values used is limited by bit widths of possible resulting values. So, what is for interesting to note is that, you are actually using straight forward integer operations to deal with real value calculations? So, that reduces the instruction count substantially.

And that is one of the reasons for we said that we might look like to do this kind of high level transformations of the task graph nodes, to get optimal solution. So, this is the transformation that we are doing, because we need it.

(Refer Slide Time: 39:09)



So, then we go into an implementation. So, this is what? This is the pure again we have not done any change in the architecture. We have simply done a transformation of a task level node. So, we get this kind of statistics. What do you find is that, since the time reduces the energy consumption is less, the battery life becomes longer. So, all this parameters in a various way are dependent.

The total chip area is this. And the gates are less why because, your less memory is needed for code. Because code size reduces your procedures goes off. You are using single instructions. So, your memory requirement reduces. You can realize why these transformations are important, when you will go into design for portable devices. So, your performance is close, but not good enough.

(Refer Slide Time: 40:01)

**Improvement: Codec in Hardware**

The diagram shows a block diagram of a codec implementation on a chip. It includes blocks for 'Encoder', 'Decoder', and 'Codec'. The 'Encoder' block is connected to the 'Decoder' block, and the 'Codec' block is connected to the 'Encoder' block. The 'Codec' block is also connected to the 'Decoder' block. The diagram is shown on a perspective view of a chip with pins.

- Performance close but not good enough
- Must resort to implementing CODEC in hardware
  - Single-purpose processor to perform DCT on 8 x 8 block

So finally, I have to go for a CODEC, a dedicated CODEC for implementation of DCT. That becomes the special purpose hardware.

(Refer Slide Time: 40:11)

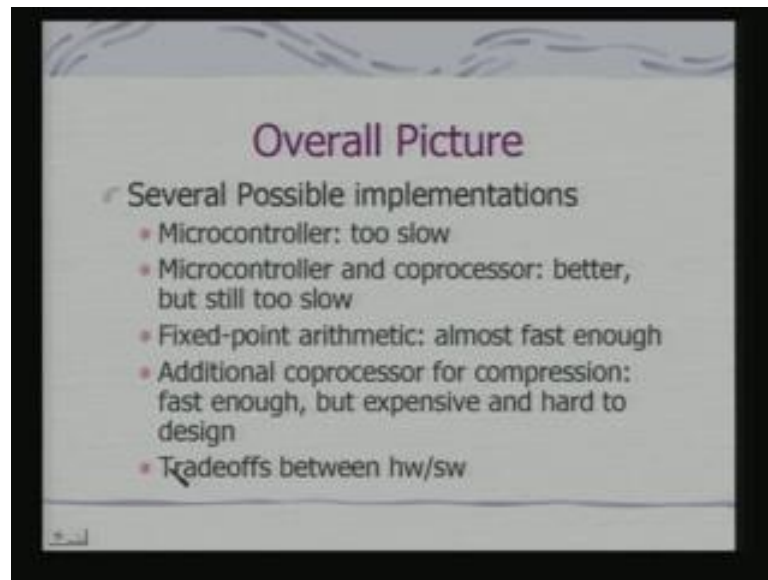
**Evaluation**

- Total execution time for processing one image:
  - 0.099 seconds (well under 1 sec)
- Power consumption:
  - 0.040 watt
  - Increase over 2 and 3 because SOC has another processor
- Energy consumption:
  - 0.00040 joule (0.099 s x 0.040 watt)
  - Battery life 12x longer than previous implementation!!
- Total chip area:
  - 128,000 gates
  - Significant increase over previous implementations

So, under that condition we can bring it down to .099 seconds. I am not going into the details of the design of the CODEC. But these standard CODECs are available. And that would be also available with VHDL code. You can generate a design the glue logic and get the system. So, here the power consumption is less increase over 2 and 3, because SOC has another processor.

So, the power consumption increases, but what is less is energy consumption. Why? Because, again the time goes down. So, the battery life becomes longer. So, total chip area is more because you are using another dedicated processor. So, this is the kind of a manual exploration of the design space.

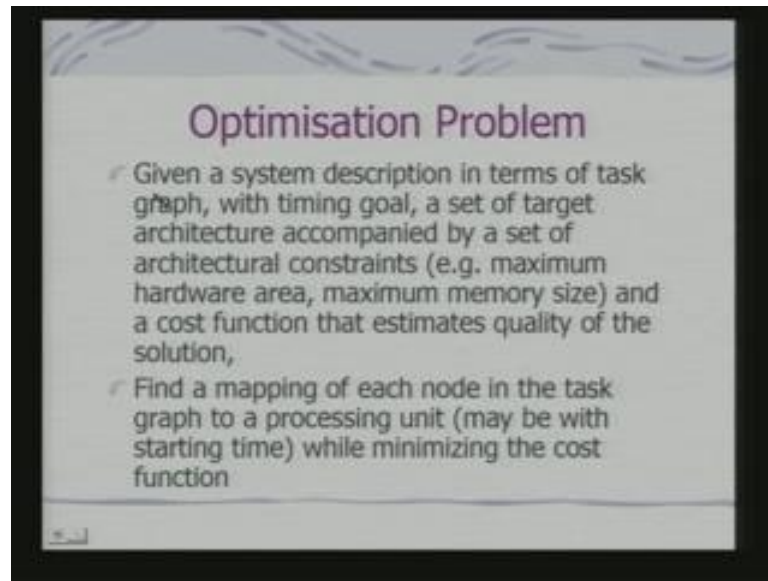
(Refer Slide Time: 40:56)



Now, we can also have the formal formulation of this problem. So, before going into a formal formulation, let us try to get the overall picture. So, what are the different architecture, we have tried? Microcontroller, pure software implementation, microcontroller in a co processor that is with additional hardware.

So, I have mapped some things to the hardware, but still slow. We have looked at fixed point arithmetic. Then, we have looked at additional coprocessor for compression. So obviously, what we find is the tradeoff between hardware and software to meet the performance constraints.

(Refer Slide Time: 41:36)



So, if we now look at more formal specification of the problem on the basis of our analysis of this example. So, what is that formal formulation? So, what we are telling is, given a system description in terms of task graph with timing goal, a set of target architecture accompanied by a set of architectural constraints. Target architecture was 8051 and possible implementation of FSM into with the help of VHDL.

And the cost function that estimates quality of the solution, the design problem or the partitioning problem is that of finding a mapping of each node in the task graph to a processing unit. A processing unit can be specially design or can be standard processor. And also the starting time, in the previous example we have not looked at the problem of scheduling, because it is a typical sequential operation while minimizing the cost function.

See if I have a set of possible architecture specified before and then the whole problem becomes the optimization problem, a standard optimization problem. And we can use standard optimization problem base techniques, for doing hardware software core design.

(Refer Slide Time: 43:02)

**Integer Programming**

Cost function  $C = \sum_{i \in X} a_i x_i$  with  $a_i \in \mathbb{R}, x_i \in \mathbb{J}$  (1)

Constraints:  $\forall j \in J: \sum_{i \in X} b_{i,j} x_i \geq c_j$  with  $b_{i,j}, c_j \in \mathbb{R}$  (2)

The problem of minimizing (1) subject to the constraints (2) is called an **integer programming (IP) problem**.

If all  $x_i$  are constrained to be either 0 or 1, the IP problem said to be a **0/1 integer programming problem**.

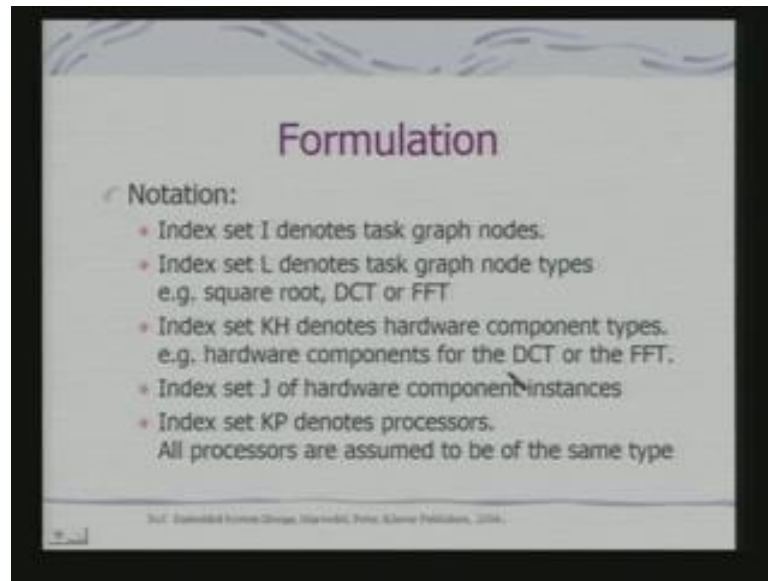
So, let look at a one such problem which is integer programming. I think many you and all of you are familiar with this integer programming as an optimization scheme. So, what we are trying to optimize? We are trying to optimize this cost function C subject to these constraints. In fact, we are looking at a minimization problem. And this is calling an integer programming problem, because the constraint is that these are all from the integer value.

And if  $X_i$  are constrained to be either 0 or 1, the integer programming problem is called a 0 1 integer programming problem. And in fact, there are standard techniques for solving integer programming problem, when even commercial packages are available. So, my job is therefore to do what? Map my hardware software core design problem to that of an integer programming problem. There are other optimizations schemes also.

See if I mapped to an optimization scheme, then I can run an optimization procedure to get an architecture generated automatically. And that is, what is the CAD tool? Completely Computed Aided Design tool for this developing an embedded system. So, this is an example of an integer programming. So, this is the cost function. These are basically the constraint and I can get the solution which is an optimal solution. And here, I am assuming that  $x_1 \times x_2 \times x_3$  are between 0 and 1.



(Refer Slide Time: 44:30)



So, let us look at a formulation because, that is a more interesting part of it. The index set I denotes task graph nodes, because I have to deal with task graph nodes. Index set L will denote the task graph node types. That is depending on the type of computation, say DCT, square root, FFT different kinds of computations. And index set KH, denotes hardware component types.

Because I said already, we can have hardware component for DCT a dedicated hardware component, so hardware component types. Index set J is of hardware component instances. And index set K P denotes processors. I can use multiple processors and all processors are assumed to be of the same type. So, I have got these sets. In fact what is the interesting and important to notice that, what we are doing is we are assuming this kind of an index sets.

So that means, I have got a library possible architectures. And what I am trying, what I shall try out through an optimization problem that of mapping, the task graph nodes to different elements of the library to get optimal cost. So, my solution is through a process of searching among the possible set of known architectures.

(Refer Slide Time: 46:02)

The slide is titled "IP problem" in a purple font. It contains a bulleted list of definitions and a cost function formula. The variables are defined as follows:  $X_{i,k}$  is 1 if node  $v_i$  is mapped to hardware component type  $k \in KH$  and 0 otherwise;  $Y_{i,k}$  is 1 if node  $v_i$  is mapped to processor  $k \in KP$  and 0 otherwise;  $NY_{i,k}$  is 1 if at least one node of type  $l$  is mapped to processor  $k \in KP$  and 0 otherwise. A mapping  $T: I \rightarrow L$  is also defined. The cost function  $C$  is the sum of costs for processors, memories, and application-specific hardware.

So, that searching is being done through integer programming. So, the problem is that we shall have  $X_{i,k}$ , the variable is one if the node  $V_i$ . This is  $V_i$  of the task graph mapped to hardware component of type  $K$ . So, it is  $X_{i,k}$  and  $K$  will be an element of  $KH$ . This set of possible architecture and 0 otherwise. And if node  $V_i$  is mapped to processor  $K$ , element of  $KP$  which is the set of processors and 0 otherwise.

So that means, this is indicating  $X$  and  $Y$  is indicating what? Whether I am designing a dedicated hardware or whether I am mapping it to software.  $NY_{i,k}$  is equal to 1 is another variable. It says, that if at least one node of type  $l$  is mapped to processor  $k$  equal to  $KP$ , this is one otherwise it is 0. That means, what we are telling is at least this task, one instance of the task is mapped to a processor.

$T$  is the mapping from task graph nodes to their types, because there are various types of computations. So, I need to know what type of computation is each task graph node is really represented. And I can use variety of cost function. This is one cost function, accumulates the cost of hardware units cost of processors cost of memories cost of application specific hardware.

(Refer Slide Time: 47:19)

**Constraints**

- Operation assignment constraints

$$\forall i \in I: \sum_{k \in KH} X_{i,k} + \sum_{k \in KP} Y_{i,k} = 1$$

All task graph nodes have to be mapped either in software or in hardware.

- Variables are assumed to be integers.

Additional constraints to guarantee they are either 0 or 1

Prof. Embedded System Design, Warwick, Peter, Elsevier Publications, 2006.

Then, we have to specify the constraints. This operation assignment constraint, what it means? It means either the operation has to be associated either with the dedicated hardware or with software. So, these constraints represent that. Variables are assumed to be integers. And additional constraints to guarantee, they are either 0 or 1.

(Refer Slide Time: 47:40)

**Constraints**

$$\forall l \in L, \forall i: T(v_i) = C_l, \forall k \in KP: NY_{i,k} \geq Y_{i,k}$$

- For all types  $l$  of operations and for all nodes  $i$  of this type:  
if  $i$  is mapped to some processor  $k$ , then that processor must implement the functionality of  $l$ .
- Decision variables must also be 0/1 variables:

$$\forall l \in L, \forall k \in KP: NY_{i,k} \leq 1.$$

Prof. Embedded System Design, Warwick, Peter, Elsevier Publications, 2006.

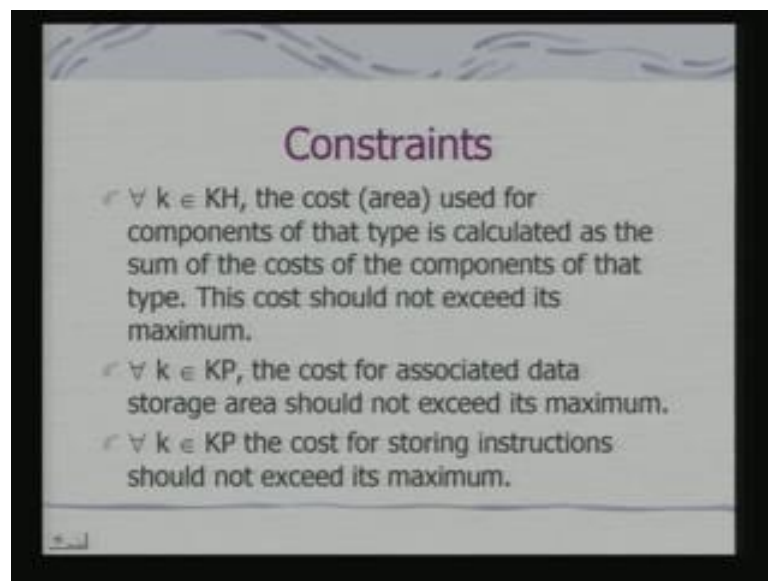
There would be other constraints as well. These constraints what does it say? For all types of operation  $l$  is the set of possible operations. For all types of operations and for all nodes stand of  $i$  of this type. If  $i$  is mapped to some processor  $k$ , then that processor

must implement the functionality of  $l$ . So, if it is mapped. So, what we say that if  $T V i$  is equal to  $C l$ . So, this is the corresponding mapping. Then, that corresponding processor has to implement this function.

And decision variables must also be 0 1 variables. And that is why for all  $l$  element of this set  $l$  types of computations and for all  $k$ , which is element of  $K P$  that is the set of processors  $N Y l k$ . It is mapping has to be less than equal to 1. In fact, the values could be either 0 or 1. So, what we are telling is that effectively I am telling that the nodes have to be mapped, constraints are telling you what? Nodes have to be mapped either to hardware or to software.

And if a particular computation is mapped to a processor, then that processor must implement that functionality. So, there has to be formally mathematically represented by this constraints. There will be other constraints now coming in.

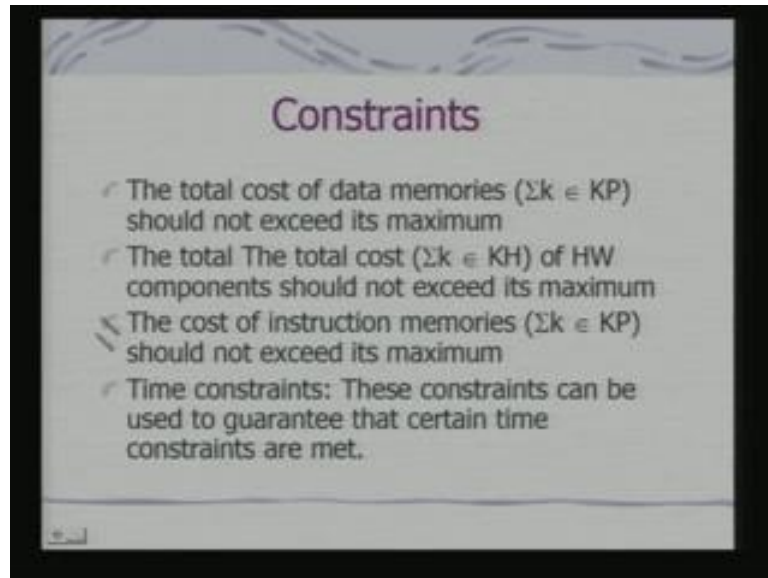
(Refer Slide Time: 49:10)



These constraints would come from design metrics. In the previous example, we have worked with only one constraint that was time. There could be other constraint as well. So, one constraint we are looking at the cost. That is the area used for component. In fact, of that type is calculated as the sum of the costs of the components of that type. That is the area be gets added up.

This cost should not exceed its maximum if I put a maximum area constraint. This keeps you a constraint on the data memory. I would not like to use data memory beyond. This keeps you a constraint on instruction memory, the program memory which is to be used.

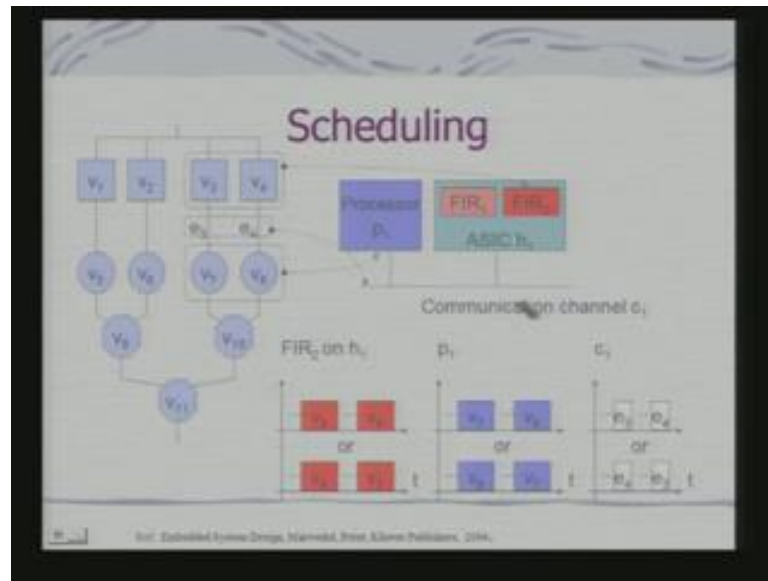
(Refer Slide Time: 49:58)



Then, you will have total cost of data memories should not exceed its maximum. The total cost of hardware component should not also exceed its maximum. And instruction memory should also not exceed its maximum. And we also talk about time constraints. There may be multiple time constraints used to guarantee that certain time constraints are met. So, what we have got? We have got constraints now on the area. We have got constraints in terms of memory usage.

We can have constraints in terms of time. So, these are all design constraints. So, what I am trying to do is now, that I have got a set of task nodes the graph representing the dependencies control dependencies. I have got a set of possible architectures. I have got the constraints. I have got the cost function. I need to find out the mapping which would minimize this cost function. Such that, all these constraints are satisfied, this is clear.

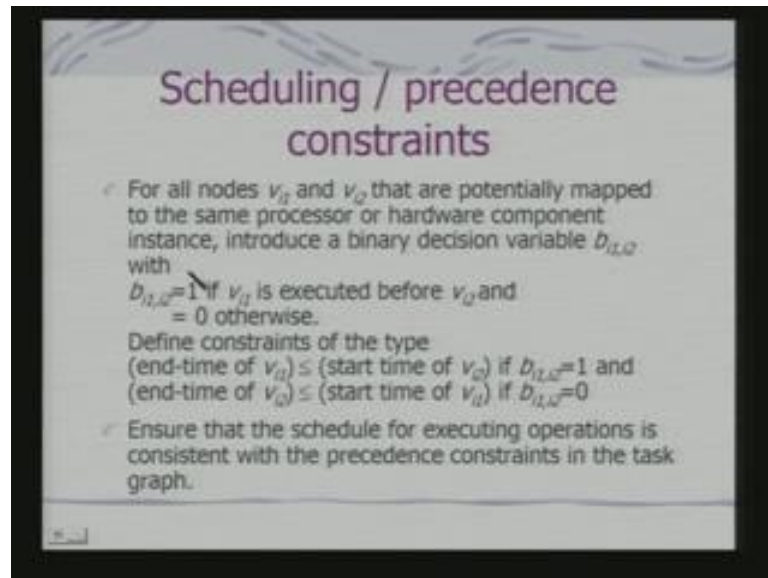
(Refer Slide Time: 51:03)



So, we can get if we take an example. Let us say, this is my task graph. I will just looking at a part of the mapping. So, what I can get? I can get, these are the nodes of the task graphs which can get mapped to an ASIC. These tasks, nodes of the task graph get mapped to a processor. This  $e_3$   $e_4$  are communication tasks. They get mapped to the communication channel or the bars by which the processor and the ASIC should get connected.

And the scheduling problem will come to what? How these tasks have to be scheduled? So, if there are two tasks which has been mapped on to, that is  $v_3$   $v_4$  has been mapped on to ASIC. Now, what is the order in which they should be scheduled? Similarly, if  $v_7$   $v_8$  getting mapped onto this processor  $p_1$ , how are these to be scheduled? And how communication has to be scheduled? So, that I satisfy the constraints of the task graph.

(Refer Slide Time: 51:59)



**Scheduling / precedence constraints**

- For all nodes  $v_i$  and  $v_j$  that are potentially mapped to the same processor or hardware component instance, introduce a binary decision variable  $d_{i,j}$  with  
$$d_{i,j} = \begin{cases} 1 & \text{if } v_i \text{ is executed before } v_j \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$
- Define constraints of the type  
(end-time of  $v_i$ )  $\leq$  (start time of  $v_j$ ) if  $d_{i,j}=1$  and  
(end-time of  $v_i$ )  $\leq$  (start time of  $v_j$ ) if  $d_{i,j}=0$
- Ensure that the schedule for executing operations is consistent with the precedence constraints in the task graph.

So, these conditions can also be formally represented. Say for all nodes that are potentially mapped to the same processor or hardware component instance, introduce a binary decision variable. And you say 1, if  $v_i$  is executed before  $v_j$  that means, I am depicting what the precedence solution, straight forward precedence solution. And you can define the constraints in terms of end time and start times.

Ensure that the schedule for executing operations is consistent with the precedence constraints in the task graph. I cannot do away with that. Because, that is the basic constraints which has come from the system model itself.

(Refer Slide Time: 52:43)

**Example**

Execution times:

T	H1	H2	H3	P
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

- ✓ HW types H1, H2 and H3 with costs of 20, 25, and 30.
- ✓ Processors of type P.
- ✓ Tasks T1 to T5.

© 2014 Embedded System Design, Warwick, Peter & Steve Parkinson, 2014.

So, let us take an example to understand this integer program. Let us say, this is the simple task graph and these are the execution times. Now, how we have obtained these estimates that is basically a key problem. But I am assuming such estimates are available because, if you remember I told you this performance estimation is the critical component for all these exercises. So, I have got this estimations may be through simulation or otherwise.

So, I have got hardware types H 1 H 2 H 3. And they have got cost associated 20 25 then 30. We have got a processor which is of type p and the tasks are T 1 to T 5. And for each task, these are the execution times on the different hardware components and that of the processor.



(Refer Slide Time: 53:39)

Operation assignment constraints

T	H1	H2	H3	P
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

$$\forall i \in I: \sum_{k \in H} X_{i,k} + \sum_{k \in P} Y_{i,k} = 1$$

$\Rightarrow X_{1,1} + Y_{1,1} = 1$  (task 1 mapped to H1 or to P)  
 $X_{2,2} + Y_{2,1} = 1$   
 $X_{3,3} + Y_{3,1} = 1$   
 $X_{4,3} + Y_{4,1} = 1$   
 $X_{5,1} + Y_{5,1} = 1$

© 2014 Embedded System Design, Marwan, Ben, & Steve Paliseras, 2014.

So, effectively what I need to find out? I need to find out a solution such that the cost would be optimized, satisfying the constraints. This is an example of the constraints. The first constraint is telling you that the task has to be either mapped to the dedicated hardware or to the processor. So, you will find that these constraints would translate to these kinds of expressions because, other possibilities it does not arise.

Because the task 2 cannot be mapped to each one, it can only be mapped to either H 2 or to the processor. That means, either pure software solution or mapped to this kind of a dedicated hardware. So, I have got these constraints, this is operational assignment constraints.

(Refer Slide Time: 54:22)

The slide is titled "Operation assignment constraints (2)". It contains the following text:

Assume types of tasks are  $l = 1, 2, 3, 3,$  and  $1$ .  
 $\forall l \in L, \forall i: T(v_i) = C_l \quad \forall k \in KP: NY_{l,k} \geq Y_{l,k}$

Below this, there are five inequalities:

$$NY_{1,1} \geq Y_{1,1}$$
$$NY_{2,1} \geq Y_{2,1}$$
$$NY_{3,1} \geq Y_{3,1}$$
$$NY_{3,1} \geq Y_{4,1}$$
$$NY_{1,1} \geq Y_{5,1}$$

A green callout box points to the third inequality and contains the text: "Functionality 3 to be implemented on processor if node 4 is mapped to it."

At the bottom of the slide, there is a small copyright notice: "© 2014 Intel Corporation. All rights reserved. Intel, the Intel logo, and Intel Inside are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. 2014.01.01.001"

So now, the other operational assignment constraint is what? Assume types of tasks are 1 1 2 3 and one these are the different kinds of tasks. So, what it says? If you look into it, the functionality 3 to be implemented on processor if node 4 is mapped to it. Because that is mapped to type of functionality 3, the node 4 is mapped to the type of functionality 3.

Since it is mapped to type of functionality 3, then the processor is it is implementing in it, because if node 4 is mapped to it. So, this is basically the operational assignment constraint which says, that if it is mapped to a processor, processor must implement that functionality.

(Refer Slide Time: 55:06)

The slide is titled "Other equations" and discusses "Time constraints: Application specific hardware required for time constraints under 100 time units". It features a table with 5 rows and 5 columns. The columns are labeled T, H1, H2, H3, and P. The rows contain numerical values representing hardware requirements. Below the table, a cost function is defined as  $C = 20 \#(H1) + 25 \#(H2) + 30 \#(H3) + \text{cost}(\text{processor}) + \text{cost}(\text{memory})$ .

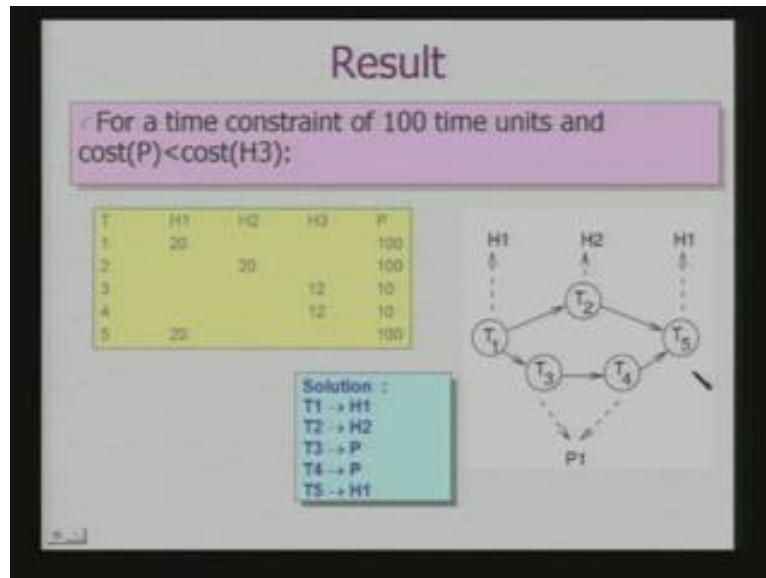
T	H1	H2	H3	P
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

Cost function:  
 $C = 20 \#(H1) + 25 \#(H2) + 30 \#(H3) + \text{cost}(\text{processor}) + \text{cost}(\text{memory})$

So, let us look at other equations. There will be time constraints because other constraints should now come into the picture. We are looking at a time constraint to one constraint for the timing. Application specific hardware required for time constraints under 100 time units. And we are defining this as the cost function; this 20 25 30 is the cost of this hardware.

And this should be multiplied by the number of this units been used, plus the cost of the processor plus the cost of the memory. So, this is what we would like to minimize, satisfying this time constraint and other operational assignment constraints.

(Refer Slide Time: 55:45)

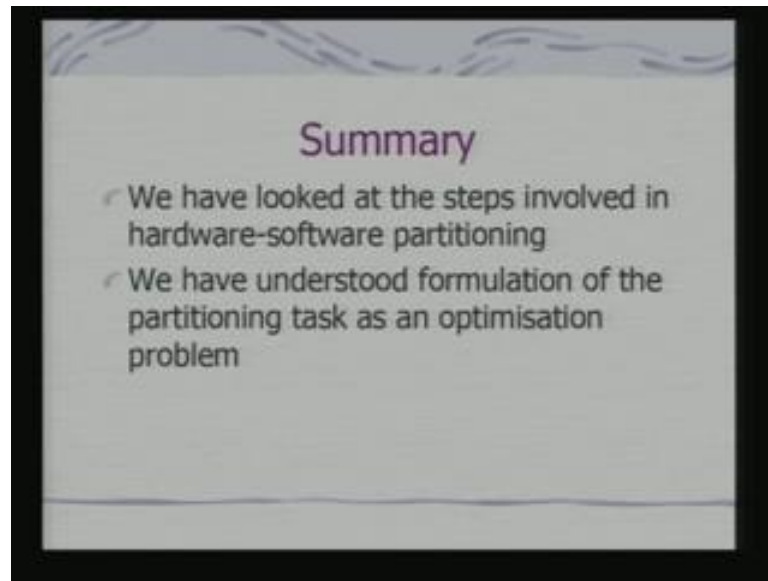


So, let us look at a result for a time constraint of 100 time units and cost of the processor p, if it is less than the cost of H 3. So, what I have done in this case you will find that I have mapped T 1 to H 1 T 2 to H 2 T 5 to H 1 T 3 and T 4 to P 1. Now, if you look at the time constraint, the time constraint will come from 20 plus 20. And this T 5 is mapped to H 1. So, I am using two units of H 1 that is 20.

So, I get a time constraint along this path less than 100, along the other path is also less than 100, because I am mapping T 3 and T 4 to the processor which is 10 plus 10, 20. So, I have got a solution which satisfies the time constraint. And why I said there is the cost of processor P is less than that of H 3. And that is the reason why I have used the mapping them on to the software, instead of using a dedicated hardware. So, I am optimizing on the cost, because I have to satisfy the constraints.

And after satisfying the constraints among the possible solutions, if I am choosing a solution which cost is less then I have to mapped the task T 3 and T 4 to the processor. So, I have got this assignment and this problem can be solved through an integer programming toolbox. So, what I am telling is I have got an automated mechanism for solving the problem. I am not doing a manual exploration of different possibilities. In fact the all these kinds of techniques are used for this kind of software hardware co design and partitions.

(Refer Slide Time: 57:41)



So, what we have looked at, we have looked at steps involved in hardware and software partitioning. And we have understood formulation of the partitioning task as an optimization problem. In fact, all through it was an optimization problem either you manually try it out or use an automated algorithm, with a set of possible target architectures to get a feasible solutions. Any questions?

Student: ((Refer Slide Time: 58:05))

The question is, if we are using a dedicated hardware for two functions whether I am using one of them or two of them. In this case, actually I have using two of them. Because use of both, use of two H 1 would act to the cost. So, when we are talking about a dedicated hardware the model that we have used for this example is that, it is implementing only one function. So, if I need two of them it will be two instances of that hardware element.