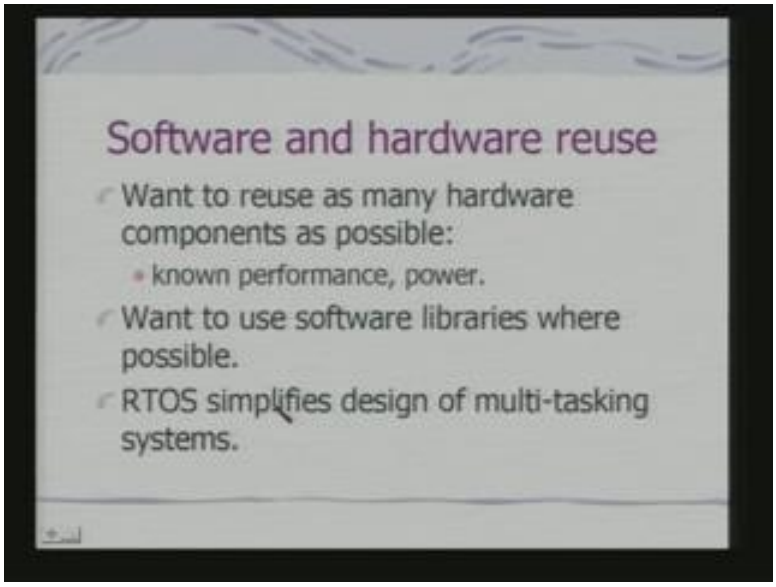
Embedded Systems
Dr. Santanu Chaudhury
Department of Electrical Engineering
IIT Delhi

Lecture - 33
Platform Based Design

We were discussing Platform Based Design, in the last class platforms providers with a set of possible architectures to be explode for designing an Embedded Systems. Today, we shall continue that discussion. In fact, platforms enable as we have already seen, software and hardware reuse.

(Refer Slide Time: 01:17)

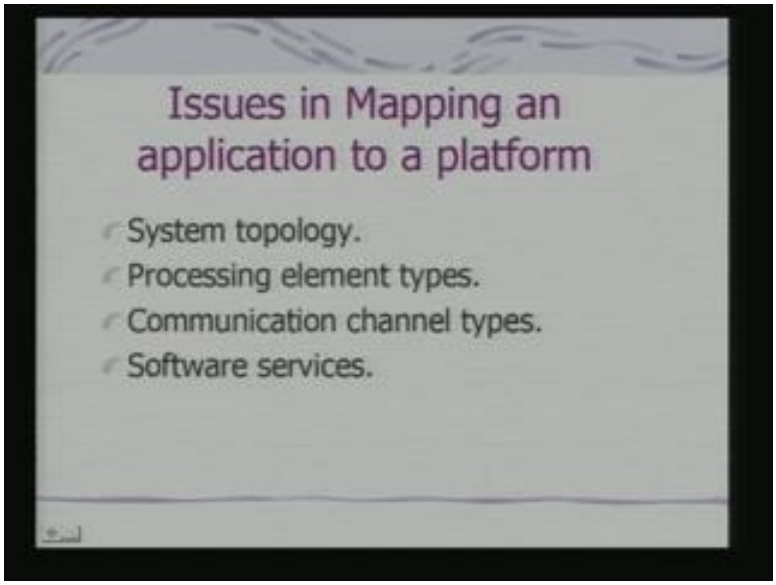


We would like to use as many hardware as components, because that give an advantage of knowing there performance statistics. Like power consumptions as well as the time taken to do a particular computation on the hardware block. We would like to use software libraries, wherever possible. And we can do that by making use of the software platform, which can be accessed for an application to an API.

And the software platform makes use of an underlying or task kernel, which simplifies design of multitasking system. So, these are the basic features of platform based design approach. So, related to this is that, when we are starting from a functionality and gone into a task graph. The question is, how do you map your task or your application to a

platform? And one aspect of the mapping is the mapping on to the hardware architecture of the platform.

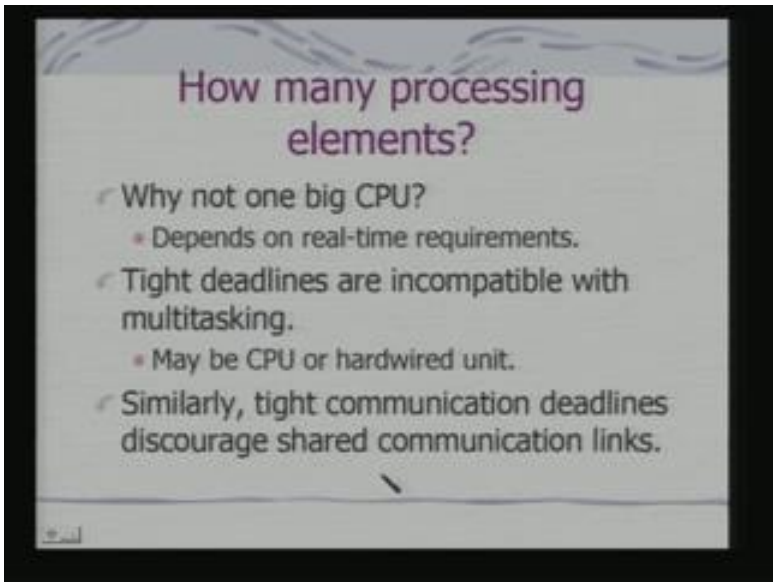
(Refer Slide Time: 02:33)



So, we have to keep track or know about the system topology, requirements about the different processing elements. What are the different kinds of communication channel available there may be constraints in terms of communication channels. In fact, we have seen in the last class an example, where we might designed a platform around arm, where AMBA bus provides the communication interface and that may be a constraint.

But, we have to do the design keeping in mind, I have got the AMBA bus for communication, on top of that there are software services. So, the software services have to be made use of to coordinate the different functions, which have been performed on the different hardware blocks, as well as for invoking those functions through your software services.

(Refer Slide Time: 03:48)



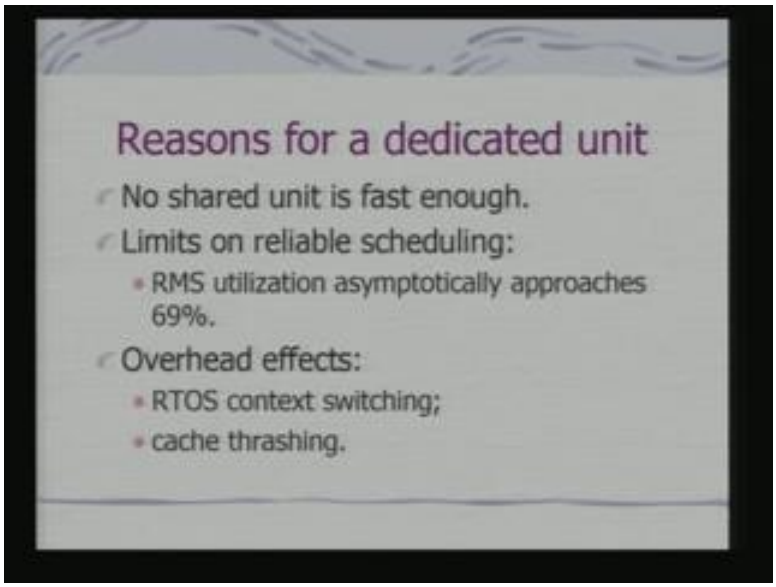
The first question that comes up is how many processing elements to you required for a given application. In fact, this might lead to also the choice of the platform, if you are talking about using one big CPU. Then, obviously, we might light to go for a platform which has got one CPU along with may be some amount of programmable logic, which may be through your FPGA block may be implemented or we might not required even that programmable hardware.

We can build in the behavior completely using software, so in that case the instance of the platform would be particular software running on a big CPU. But critically that depends on your time constraints. In fact, if you go back to the example of your camera design that we had considered, we are not being able to do the design using only 8051 simply. Because, I had the timing constraints that the camera should be able to take the next snap within one second and I cannot do compression of the image using your 8051.

Even when I am using additional logic on FPGA blocks. So, tight deadlines in many cases are also in compatible with multitasking, when we really have the deadlines and a deadlines are universally to be obeyed in a real time application. So, what we need, we need to mapped task now to different hardware units, some tasks may be to one CPU other task to may be a dedicated hardwired unit.

Similarly, tight communication deadlines discourage shared communication links. So, in that case we might need to build dedicated communication link using the configurable logic facility available on a platform.

(Refer Slide Time: 06: 04)



So, therefore what would be the reasons for having dedicated unit, we shall have dedicated units only when the shared unit is not fast enough to meet our timing constraints, as well as our deadline requirements. In fact, what we have found is that even when if I use rate monotonic scheduling. The utilization of the processor asymptotically approaches about 69 percent, so what it implies. It implies, that we really have hard deadline constraints using a simple CPU and trying to use that CPU optimally by following a proper scheduling approach will not giving results.

If, your constraints are tight enough, so under those circumstances you have to go for dedicated unit. Also there are overhead effects as I have told you in a software platform my RTOS kernel can be made use of for managing the concurrent model, but each task will correspond to a process in RTOS and there would be context switching. The context switching overhead can actually affect my timing constraints, because of the context switching overhead I might not be able to meet my timing constraints.

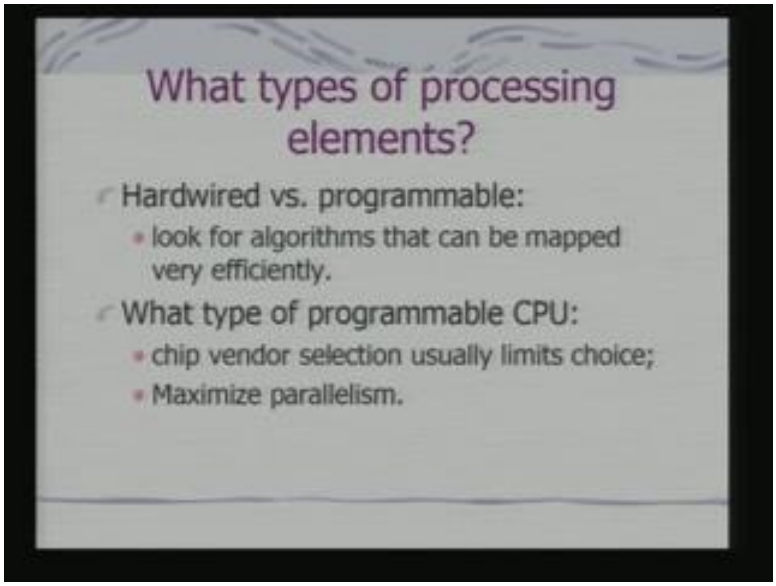
When we are using one CPU and mapping multiple tasks on that CPU. At the same time there can be cache thrashing; that means, when we are switching from one task to another I might need to reload the cache memory, so there will be miss and hit at regular

suggestion. When I am trying to access pages corresponding to one task, I shall encounter and miss, I have to reload and there for the task for the small time period I might have the pages in the cache.

But, whenever there is the task switch I might not get those pages on the cache and there will be a miss. The moment I have got cache thrashing coming into the picture, we are paying for performance and not only performance, we shall be paying for energy. Because I need to transfer the data from main memory to cache, so energy budget will also increased.

So, under these circumstances it may a good decision to go for a platform, which provides the support for multiple CPUs may be multiple ALUs, as well as you might like to have a dedicated FPGA block, where you would like to implement a specific functionality. So, that FPGA block in that case becomes the dedicated unit.

(Refer Slide Time: 09:12)



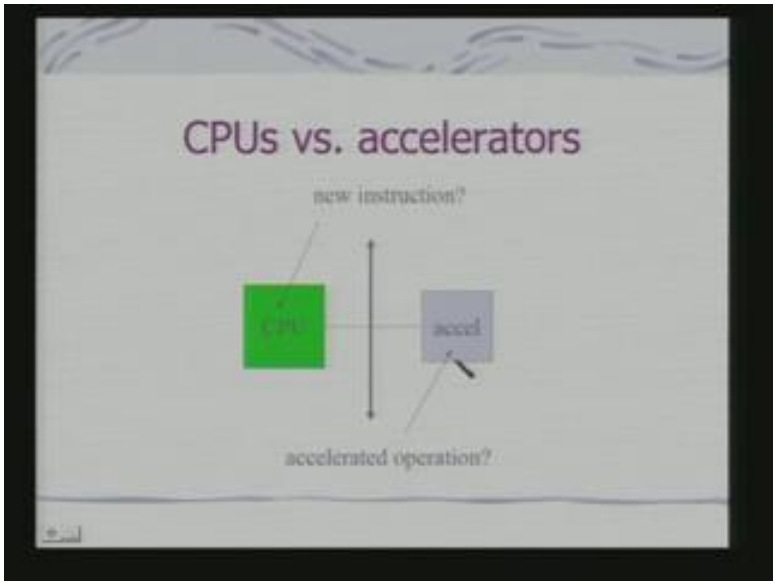
So, therefore the question comes up is, what kind of processing elements Hardwired versus programmable. Now, it depends on what kind of algorithms that you are making use of. In fact, in both that cases, if you look into it if it is a programmable option that means, if you have a multiple CPU and you would like to implement a certain functionality on another CPU the entire implementation is software based.

And there we need to have efficient algorithms, but if you looking for a dedicated hardwired to be built for that purpose then you got to have an algorithm, which can be translated easily to an efficient hardware implementation not all algorithms can be directly mapped on to hardware. It might requires some world to map it into a hardware and you will be using, possibly VHDL to code that hardwired algorithm for the target dedicated unit.

In fact, when we are using a programmable CPU, the chip vendor selection usually limits the choice. In-fact you cannot have all possible kinds of combinations, see if I go for TIs OMAP I know what kind of CPUs are available and how the heterogeneous task graph for an application should be mapped on to the CPUs taking into account there characteristics features. And what do you would to like, if you are going for a multi CPU platform to maximize parallelism and that would give you a maximum benefit.

But there also we have to keep in mind that when you are looking at a task graph, the alternate branches of the task graphs are getting map to different CPU. So, the total time delay would be decided by the time taken for the longest branch. So, when you are doing an allocation that point has to be kept in mind to make sure that finally, you satisfy the timing constraints. We are looking at one problem, one aspect of this platform based design, where we are having.

(Refer Slide Time: 11:41)

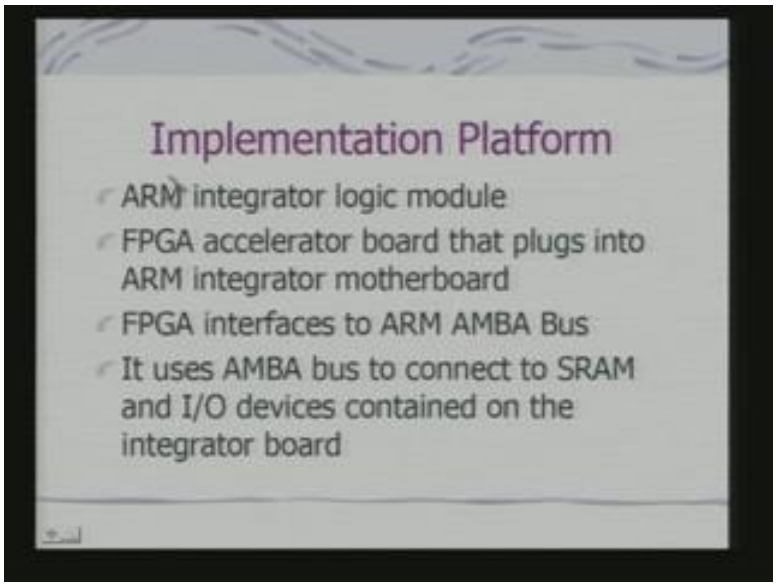


Let us say as CPU, along with some kind of a programmable logic the question is we need to have some functionalities to be implemented in hardware, depending on my timing constraints. Now, you have generically two basic approaches, if you really have a soft core, you would like to modify the CPU add on new instructions and get something done. On the other hand, your other approach could be if you are not having a soft CPU or a soft core based platform, you might be having a hard core CPU with a programmable logic on the platform.

Then you would like to design the accelerator that means a particular hardware function to accelerate one bottle neck that you figured out. See if you go back again to the camera example, we have found the DCT was the bottle neck. So, we might like to use this FPGA block to implement DCT to remove that bottle neck. So, you are not modifying the CPU by any sense, but we are implementing a special functional unit and that in many of times you refer to a accelerators.

In general purpose computers, you do use very commonly graphics accelerators, which accelerates, graphic state of processing for the purpose of realistic image synthesis on the screen, which you need for variety of applications including gates. So, accelerator provides accelerated options.

(Refer Slide Time: 13:38)

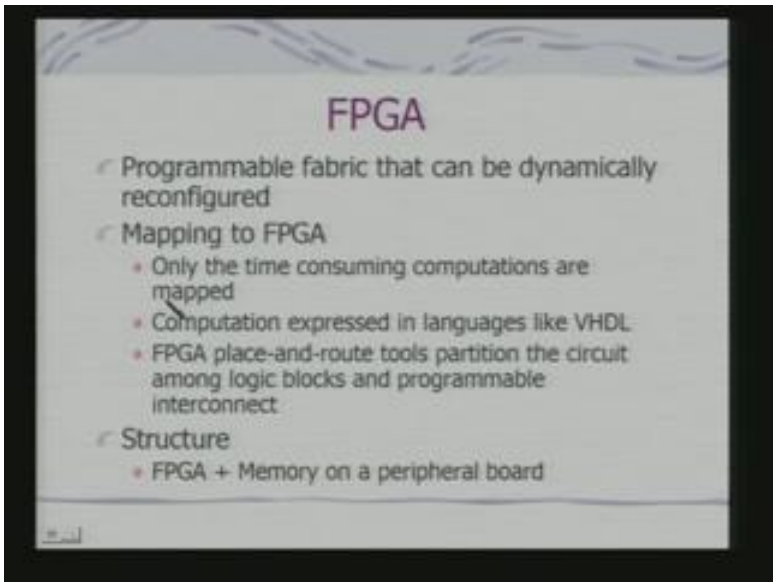


So, how do you go for such an accelerated implementation, we can look at a possible implementation platform and we are just looking at one example platform, which is

actually available commercial. In fact this comes from arm manufactures, so what we say arm integrated logic module. So, here we have got a basic arm processor based card, which may be an arm nine based card, if, you are looking at arm nine integrator to which an FPGA accelerator board can be plugged in.

The FPGA interfaces to the arm processor via AMBA bus via AMBA bus. So, here the communication is fixed, the FPGA can use AMBA bus to connect to SRAM and IO devices contained on the integrator board. So, the integrator board is basically your hard core host CPU and this accelerator board is the basic hardware frame work on which you would be implementing the required function. And these accelerator makes use of the same AMBA bus to communicate with the processor, as well as it make for making use of the memory and IO devices.

(Refer Slide Time: 15:16)



Now, what are these FPGAs really if you are trying to design a function, we should know what are these FPGAs, in fact FPGAs are in a way a programmable fabric that can be dynamically reconfigured. In fact, this point we have looked at when you looked at reconfigurable SOCs. So, basic job now turns out to be mapping of your task to FPGA. Typically, only the time consuming computations are mapped, that computations which are basically bottle neck.

So, one way to identify such computation is by which process, we can actually compiled the entire task by encoding it may be in C or a procedural language compile the entire

task, run it on the simulated of the CPU, profile the code, find out the parts of the code which had actually consuming maximum time and figure out whether the code is really violating the timing constraints. If it is violating the timing constraints you should identify from the profile code the bottle necks.

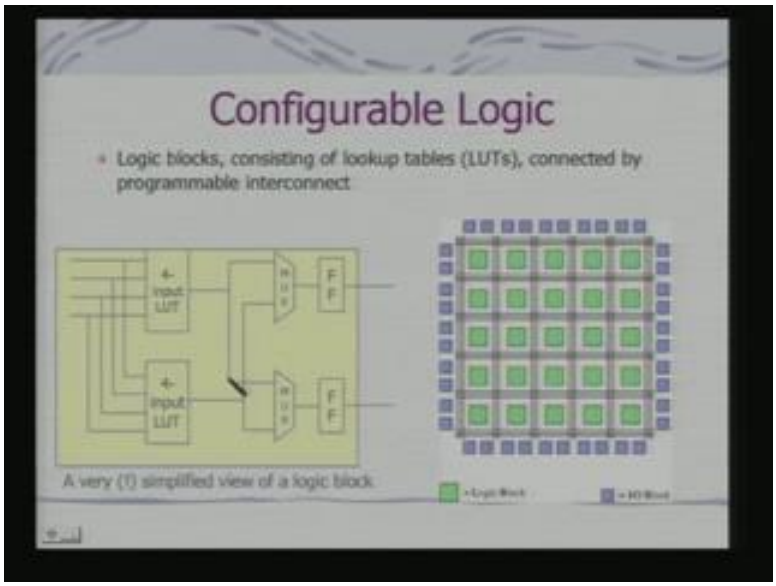
The functionalities implemented in the bottle neck should really be mapped to FPGA. The computations in this case have to be expressed in languages like VHDL. So, that means, if that is the software algorithm it has to be mapped to a sequence of hardware computations or hardware operations, which can be encoded in a language like VHDL. Which enables you a ((Refer Time: 16:54)) as well as register transfer level, also interconnect level specification of your computation.

But, the issue now turns out to be that you have expressed your computation, but that computation has to be mapped among the logic blocks of the FPGA. Because FPGA is what we said is the programmable fabric in a sense it contains, a set of programmable modules, we shall see examples of those soon. But let us try to understand what FPGAs are right, now conceptually there would be a set of logic blocks. So, when you are writing a computation in VHDL, it comprises of a set of functions.

Now, these set of functions have to be again partitioned and mapped into the FPGA blocks, for that we need to use what we called place and route tools for partitioning the circuit, which you can actually synthesis from your VHDL description of your computation to different logic blocks on the FPGA. And typically a structure of this kind of an FPGA base code is FPGA, plus memory on a peripheral board. This memory can have the data which is used by the computation that the FPGAs expected to do.

In fact, you can get FPGAs in various forms FPGAs can be some FPGAs can be programmed only once. So, once it is programmed it can be changed, there are other FPGAs which are reprogrammable FPGAs, which actually can form the reform configurable component in a platform.

(Refer Slide Time: 18: 44)

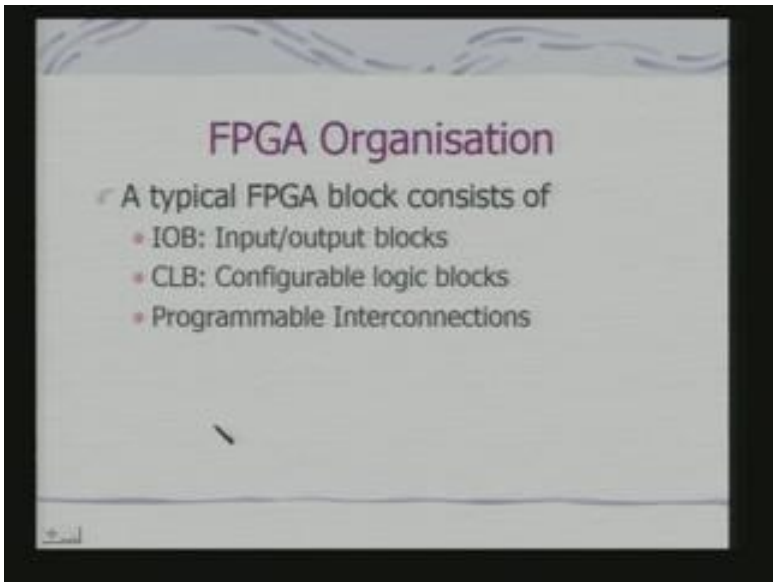


So, what are really FPGAs, FPGAs we said is the configurable logic and here we are showing a very simple example. In fact, this is a basic logic block, you have got a four input LUTs, so it is a kind of a combinational circuits, this is another 4 input LUT. So, depending on the four inputs you can synthesis any 4 input functions the basic idea is that and this is the multiplexer, depending on control inputs you can choose the functions from this two blocks and these are the flip flops.

So, this forms what is called a basic logic block, which consist in a way a set of look up tables connected by a programmable interconnect. In fact, this multiplexer tells you which one is to be connect to the flip flops and subsequently these blocks themselves can be connected in a programmable way to realize the computations. So, effectively what we have seen, we are seeing that we have got a set of combinational circuit blocks, we have got a set of flip flops registers.

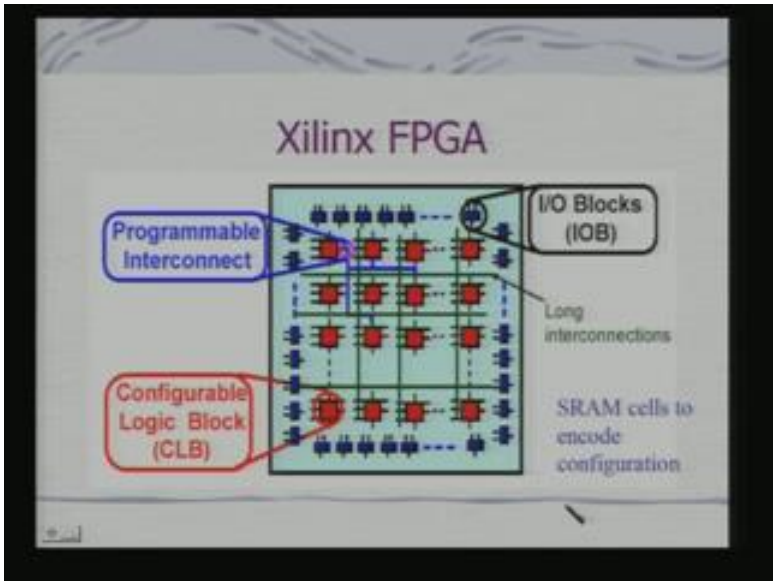
And if I have these set of blocks and if I have the freedom to connect them in any way I would like, I can synthesis any combinational or sequential logic functions. And any hardwired unit which is doing any kind of a function therefore, can be synthesized through this kind of FPGA blocks. In fact, this is showing you an organization, where we say that this is the logic block, these are effectively your IO blocks and these represents the interconnect pattern and this inter-connection patterns it should be possible for as to program.

(Refer Slide Time: 20:39)



So, typically therefore an FPGA consists of following blocks, what we call is IOB that is input output blocks. Then you have got configurable logic blocks, these are the basic units to which your functions are mapped and then you have got programmable interconnections.

(Refer Slide time 21:10)

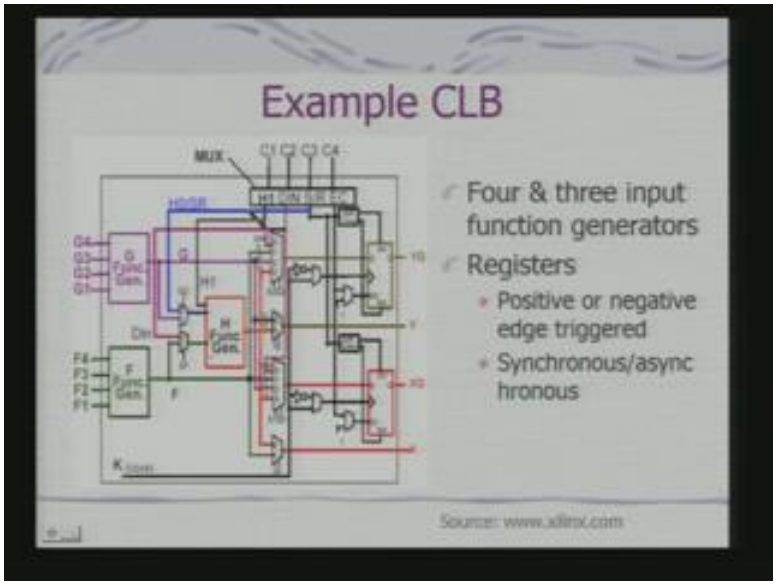


So, if we look at this is one FPGA there are various manufacturers we are looking a Xilinx FPGA organization. So, this is exactly what we have got, we have got your configurable logic blocks. These are your programmable interconnects. So, how this will

be connected these can be actually programmed and these are called long interconnects, these are your IO blocks depending on your interface to the external world this IO blocks can be set.

In fact, what is your programmable part of the logic you actually can synthesis what this logic blocks can actually do, you can synthesis how this logic blocks have to be connected. Also you can specify what kind of inputs should come into this logic block so that means, using these blocks you have the ability to synthesis any kind of the circuit. And in fact, SRAM cells at used encode the configurations, when I am using SRAM cells to encode the configurations, which has actually means that if I change SRAM data I can get another configurations for this FPGA block.

(Refer Slide Time: 22: 34)



Let us look at an examples, CLB If you look into here I have already given a simple example, here this is example from an actually FPGA block. See these are your 4 input look up table implementations, which can synthesis any 4 input function. This can synthesis another any 4 input functions and this is can synthesis a 3 input functions. This can synthesis a 3 input function and these are all multiplexers and these input is basically your control input that means, using these inputs, you can actually configure the system to synthesis different functions.

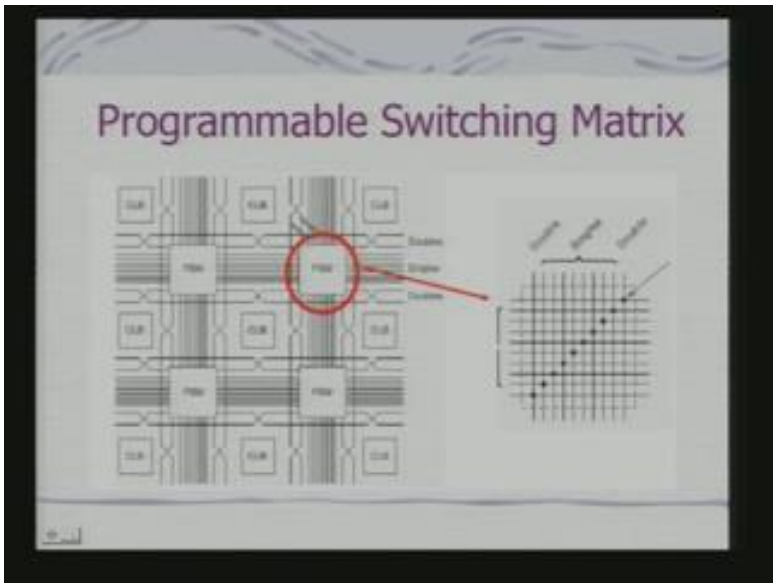
If you look into this path, this actually tells you that whether this H 1 is to be computed or not. In fact, the other information which are coming out, that is your controlled inputs

for your multiplexers and others that also can be provided through this inputs. Here, we have got the flip flops, this flip flops also have got set, reset logic, we have also got the set reset logic and they can be positive or negative edge triggered synchronous or asynchronous and you can provide the clock as well clock as well to this blocks.

So, effectively what you are finding here is that you can synthesis logic functions both combinational and sequential logic functions even using a single FPGA, block logic block. But and you can synthesis; obviously, more complex functions when you connect this logic blocks together. So, effectively what we have got, we have got therefore a kind of a programmable fabric to synthesis any logic functions. So, in a way we have got what we called hardware programmability.

Try to distinguish between the software programmability with respect to hardware programmability in terms of your platform based design. FPGA is providing with you a feature for hardware programmability. That means you can synthesis different functions by selecting, which four input functions to be implemented here, selecting how this muxers should be configured, multiplexer should be configured, figuring out or telling how this registers should be really behave.

(Refer Slide Time: 25:32)



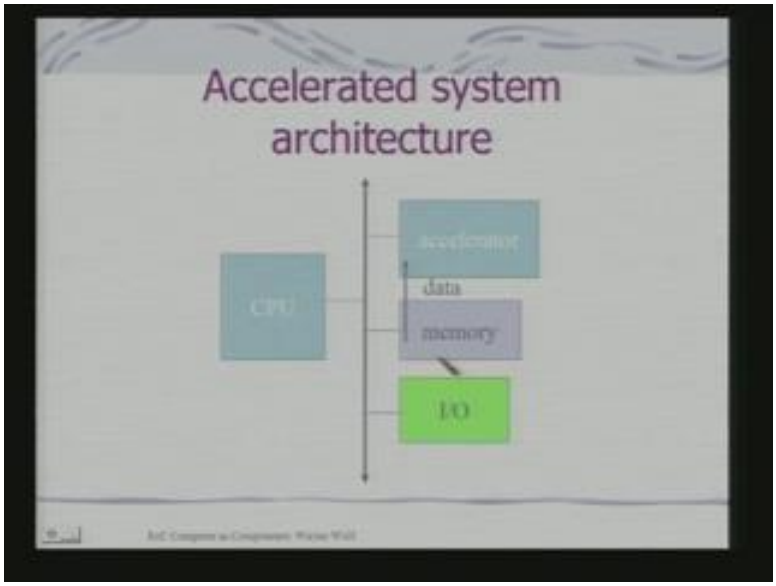
And these becomes the programmable switch this is the programmable switching matrix, which is connecting the different CLBs together and I can load. In fact, the data on to this matrix to decide on the inter-connection and as well as for configuring, may be the

CLBs because they have to provide various kinds of control inputs. So, it will come from one CLB to another and you can be an input as well.

So, the switching matrix determines the inter connectivity between the logic models so; that means, when we are looking at the problem of designing an accelerators it means what, the function which is to be implemented in hardware has to be mapped to FPGA. So, if you are using mapping tools straight away then what do you need, you need to encode your function in VHDL use the tools to simulate and find out first whether your VHDL code is correct or not.

Second thing, whether your VHDL code is synthesizing a circuit, which can satisfy the timing constraints, if you are satisfied with that then you do the mapping on to that FPGA block and once you have done in the mapping you have got an FPGA based implementation of the function.

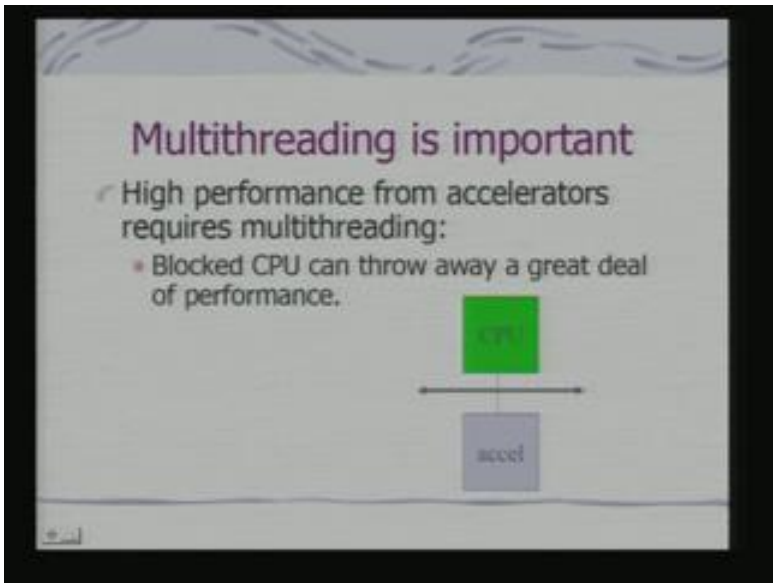
(Refer Slide Time: 27:03)



So, therefore, if I now have got that implementation, I have got an accelerator, the question is that implementation also implies that you have designed the glue logic, using which the accelerator knows how to communicate with the CPU and memory. So, typically what happens is the CPU puts in a request to the accelerators. Accelerator performs the function. In fact, it can access the data because to perform the function it would deep to access the data. So, it will access the data which is in the memory.

And can write the result back to the CPU, add the memory or you can write the data on to the memory and inform the CPU that it has written the data on to the memory. This is how typically and accelerated functions and this interface forms part of your glue logic.

(Refer Slide Time: 28:07)



Now, this takes as to the software aspect of the whole design, I told you that a platform based design is just not hardware design, but software design as well. The question is if you have got the CPU and accelerator, so how should the software be designed to manage these two things. So, typically what we say is that, these kinds of configurations would require a multithreading implementation. Why multithreading, one of the threads can manage the CPU task the other thread can actually look at to the accelerator.

Obviously, the thread is not really getting executed on the accelerator, but keeps track of the job being done on the accelerator. Because what would be really that communication modality between the CPU and accelerator, will it be a blocking communication. That means, the CPU we have found that CPU typically request the accelerator to do the job. So, if the CPU is requesting the accelerator to do the job will CPU get blocked

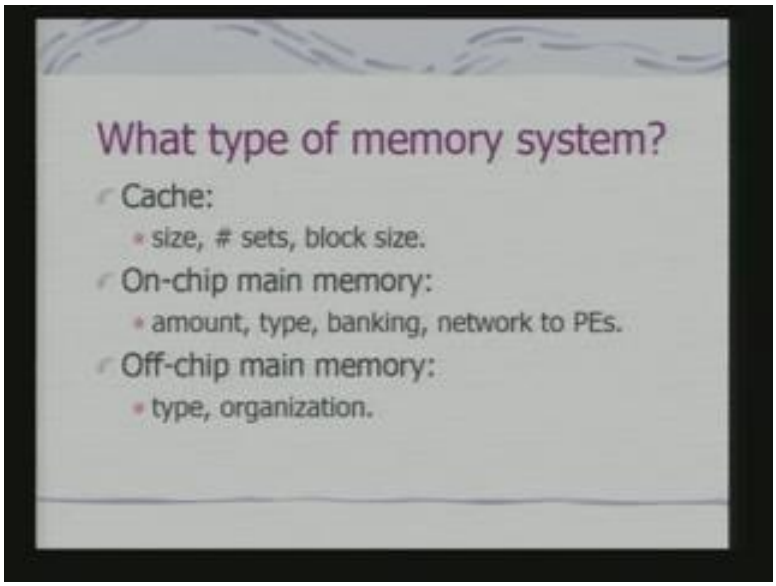
If, I really do not have a multithreading kind of an environment, that CPU can actually be blocked and wait for the result to come back from the accelerator. If you are really looking at a multithreaded implementation then thread can be blocked, but CPU can be used for servicing some other thread, CPU can be used for servicing some other thread.

So, typically a kind of an accelerator based implementation would like a software model which is the concurrency based and the thread based.

So, that configuration can come from your API definitions, because through your APIs you will be managing the concurrency models. And this concurrent threads you will be linked with the different tasks, which have been done on the CPU with the help of the accelerator, please try to keep this in mind that although this task is getting mapped on to the accelerator CPU has to keep track of the data. So, that task is being mapped to the thread which is running on the CPU.

So, actually CPU in a way is still sheared, although the actual computation is being done by the accelerator. Along with it also there had issues of memory design what kind of memory system you would like to have.

(Refer Slide Time: 30: 48)



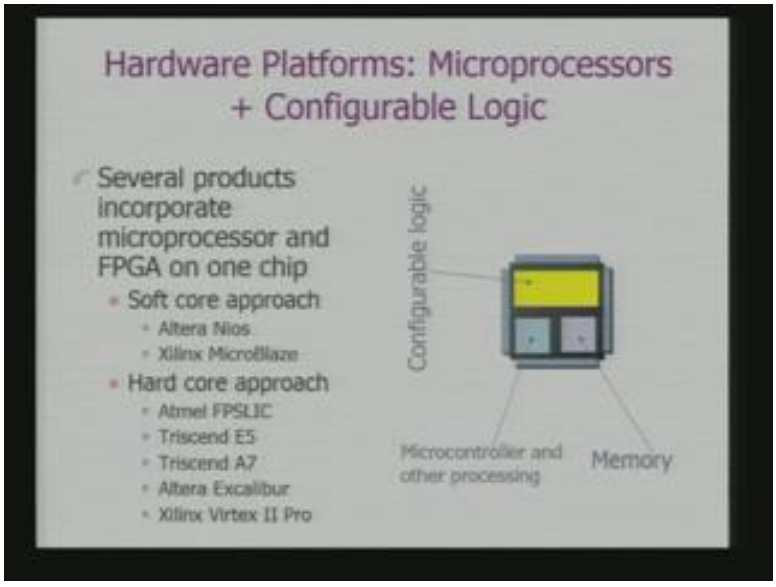
Cache if it is cache, then what is the size, what are the numbers of sets to be used, what is the block size to be used, because that has to be determined depending on the characteristics of your computation. Because you would like to optimally select this numbers such that your cache needs it minimized and in a way your energy consumption is minimized, as well as the cost does not go up. If you are using only say cache like memory and nothing else.

Then; obviously, your cost go up, so you have to optimized on the cost as well, also an on chip main memory what is the amount type because banking of the memory becomes an important issue, because when we discussed the power architecture we have told that if we can optimally arrange memory into banks then your power can be safe. Because you leakage current over individual banks are much less, your capacity of current to your capacity load also gets less because there are less number of lines to drive.

And there may be off chip main memory as well, which actually has got the code; that means, your flash memory, so what kind of flash to be used, what kind of organization to be adapted. So, these issues let us to what is called your memory design on the platform, but; obviously, the platform gives you a fix set of choice, you have make choice among those options only. But it may, so happen that platform has the facility to interface external memory.

So, you have the advantage to make the choice about your memory. So, then the memory architecture gives you another dimension of the exploration in the design space. We have looked at one aspect of exploration that is use of the CPU and the accelerator, the memory architecture provides you another space of design exploration. In fact let us with this back ground, let us look at some of the existing platforms on which this kind of strategies can be used and apply.

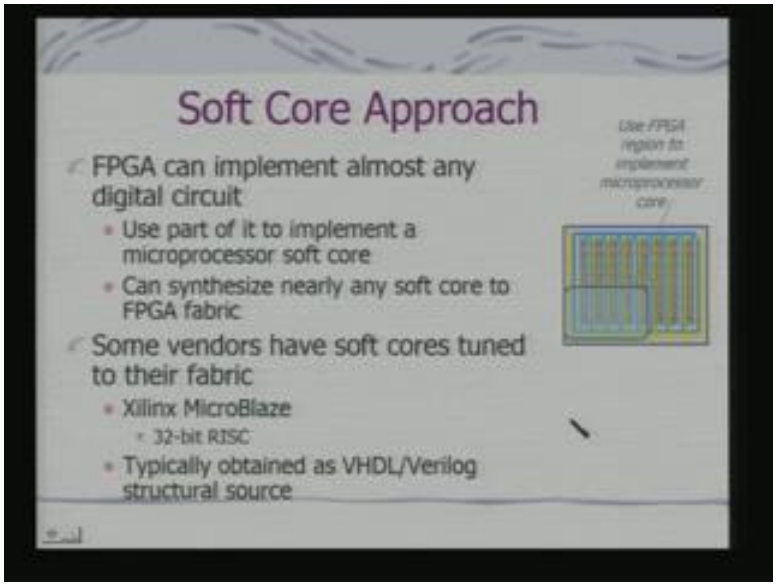
(Refer Slide Time: 33:11)



In-fact, there are number of these kind of hardware platforms which are today available, where you have got microprocessors and configurable logic. In fact, the arm example was arm integrated board with a separate FPGA board. Now, we are talking about may be a single device, which can have microprocessors as well as the configurable logic. In fact, there are two kinds of devices, which you get one is based on soft core approach another is based on hard core approach.

And in a typical soft core approach, the basic idea is that you have got the FPGA, on the FPGA you have actually synthesis the processor itself. And in a hard core approach you have the processor with the processor you have the FPGA block, just like the accelerator kind of a design that we had talked about. So, conceptually the block diagram looks something like this, you have got here the configurable logic block, you have microcontroller and other processing elements and these provides the memory.

(Refer Slide Time: 34:27)



So, let us first look at the soft core approach. In fact, if you remember we discussed that in the last class also that, if I am doing a platform based design with the soft core availability I have got certain freedoms available with me in terms of choosing the processor architecture, choosing the IO devices, choosing the register set which a processor can have. So, these could be the parameters of say VHDL base design, which can be with to make a choice about the architecture.

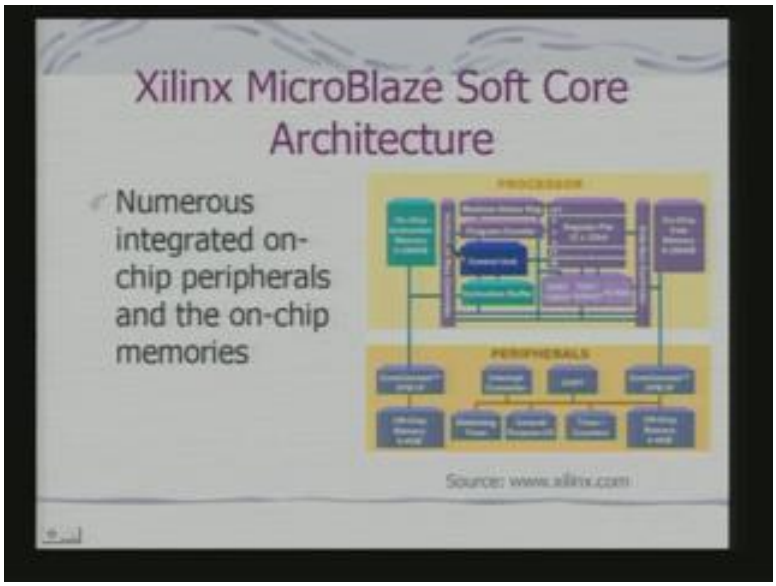
In fact, in this case when we have the soft core based approach, you basically get a big FPGA block. And you use a region of FPGA to implement the microprocessor core. In fact, again if you go back to the camera example, we said that 8051 is available with us in a VHDL form, if it is available in a VHDL form. If we have these kind of a soft core approach and if we have a parameterized form of 8051 architecture then we can play with the parameters to get variations of 8051 to meet our requirements and that gets interface here.

So, once we get one particular configuration, what we are getting we are getting an instance of the platform. The platform as such can implement if I start with the 8051 as the VHDL can implement a variance of your microcontroller family of 8051. Which particular instance you will implement will be dictated by your requirement. So, what we say we use a part of FPGA to implement a microprocessor soft core and can synthesize really any soft core to an FPGA fabric.

And some vendors, even when they provide soft core approach say provide also processors tuned to their fabric. I have just picked up one example, Xilins Microblaze which provides the soft core for a 32 bit RISC processor, for a 32 bit RISC processor. That means you can actually, if you are looking at a platform instance we will be searching around this basic architecture.

Some variant of this basic architecture to meet your requirements, because this is typically provided as VHDL or Verilog structural source, Verilog is another language that of your VHDL hardware description language. So, this is the typical way, this is done in a soft core approach.

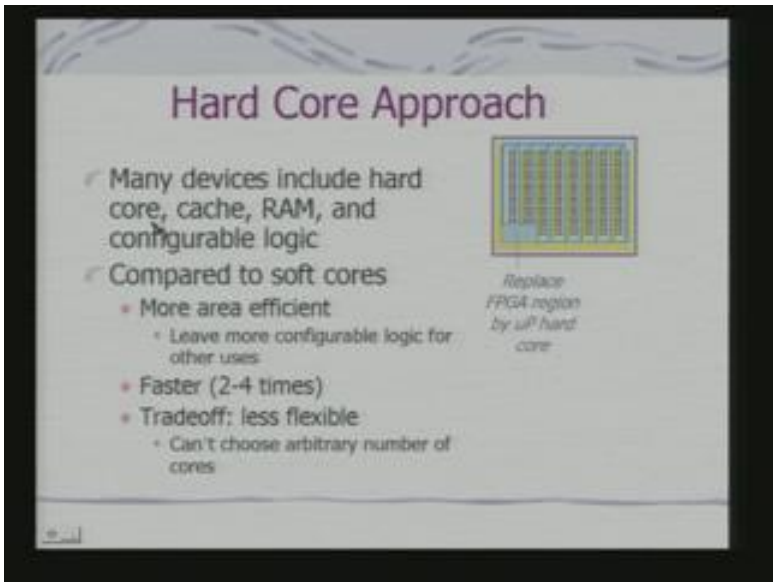
(Refer Slide Time: 37: 23)



So, if you look at the basic structure of this Microblaze soft core architecture, we will find that this is basically the processor configuration. This processor configuration on the processor logic how it would work, everything should be available in a VHDL form along with it, you have got peripherals. So, numerous integrated on chip peripherals and on chip memories have been provided here. So, here what you find that, there is an on chip data memory, you have got these part is your actual architecture the control unit, instruction buffer, program counter, machine status registers, register file.

And these are variety of peripherals which are available and you can configure the whole system has a soft core and then implement for your embedded systems. So, in this case the advantage is your search flexibility that means, flexibility of the platform is much more compared to a hard core approach. Because you can actually play with the processor core itself, to meet your requirements.

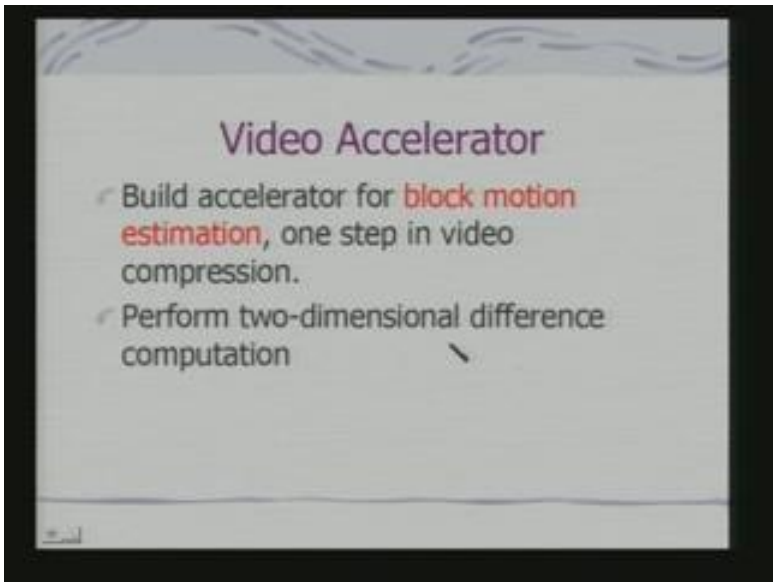
(Refer Slide Time: 38:47)



On your other hand the hard core approach, in fact the accelerator design is an example of exploitation of this hard core approach. Many devices include hard core that means, a processor, cache, RAM and configurable logic. So, in this case your memory architecture is also not available to you to decide, it is a kind of a fixed memory architecture what is available is primarily is configurable logic. Compared to soft cores it can be more area efficient, because you are deciding the processor once in the most efficient fashion.

And leave more configurable logic for other uses. Because in the other case the processor itself is consuming the configurable logic block, it can be faster because you can have an optimized implementation of the hard core CPUs. But tradeoff is obviously less flexible cannot choose arbitrary number of cores. In fact, in a soft core approach I can actually choose multiple CPUs and I might like to implement multiple CPUs, depending on my requirements and mapped tasks on to this multiple CPUs. But, those kinds of flexibilities are not really here because here, these CPU is expected to control the basic operations.

(Refer Slide Time: 40:29)

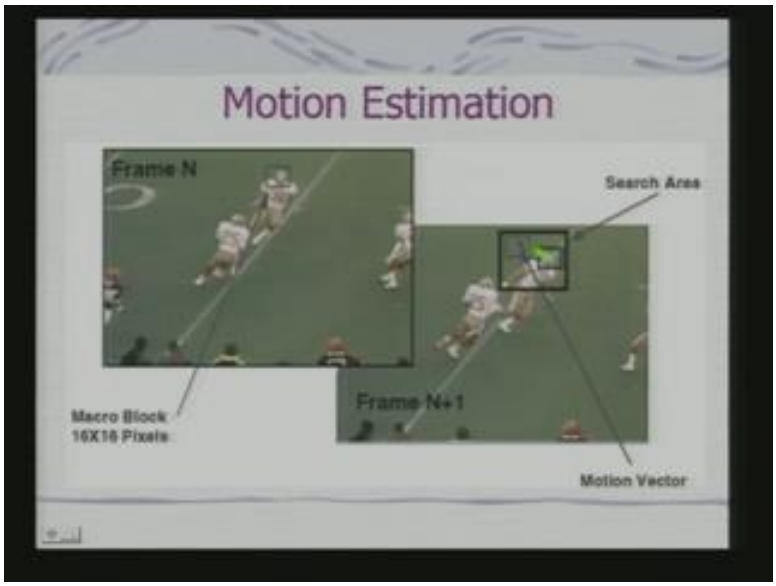


Now, with this back ground, let us look at actual design examples, with the platform based approach, first you shall look at an accelerator of video accelerator and this video accelerator is very common for any kind of an appliance, which is doing a video compression. In fact, with the camera we had discussed the basic principle of image compression, what we do we apply DCT to the image, then do quantize then we do entropic coding, when we are looking at that video analysis.

When we are having a video camera for that matter, we would like to design which would stored in the memory stick MPEG videos then we need to implement in hardware the MPEG compression. The MPEG compression scheme is conceptually similar to that of JPEG in various ways the only difference is the MPEG also has to exploit what we called temporal redundancies. And to exploit temporal redundancy, they need to perform what is called block motion estimation.

And that is the most compute intensive part of the video compression. So, we shall look at the problem of building an accelerator to do block motion estimation, in a way this would perform a two dimensional difference computation.

(Refer Slide Time: 42: 09)



Let us try to understand this problem, what is the problem of motion estimation, this is say this is the frame n and this is a frame n plus 1. Now, actually if you see these parts of the picture has moved almost in an un-changed fashion to these paths. So, if I can know this shift with x and y, delta x and delta y shift corresponding to this block then I did not encode this image refresh. I can reproduce this block from the previous frame, this is what is called the temporal redundancies in a video stream.

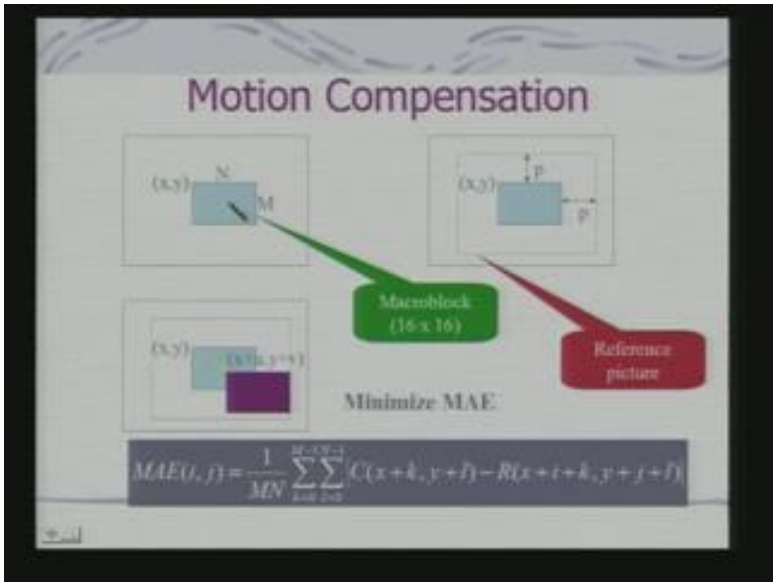
And we would like to exploit this temporal redundancy for the purpose of compression. In fact, MPEG compressions stream consist of what we called i p and b frames. In fact, i frames are nothing, but JPEG compress frames. P frames are what are called predictably compress frames; that means, for p frame we actually exploit this kind of temporal redundancy. That means the data in the current frame I can predict from the previous frame.

And what we stored in that case, we stored this shift vectors and the error. The error is stored again like a JPEG image we apply a DCT to the error, then we do quantization, then we do entropic encoding and stored the error corresponding to p frames. When we have b frames, b frames means bidirectional coding. So, in that case I can it may be possible for me to predict a window here from one of the feature frame; that means, from the frame n plus 2 I can predict also backward.

When we are using a combination of forward prediction as well as backward prediction, we get b frames. You can realize obviously, that computation of b frames for the purpose of compression is computationally most complex and in your video telephony, video conferencing kind of a applications. We typically do not use b frames, we use i and p frames.

So, when we use i and p frames you can understand the DCT is; obviously, a time consuming block and another time consuming block is this your motion estimation. Because I have to actually, search in this image to find a match for this block to exploit this redundancy. In fact, what I have shown here is the search area, the bigger block is the search area and I am mapping this block in the search area and I am moving around this block over the search area to find the best match. So, these computation, is what I would like to accelerate.

(Refer Slide Time: 45: 43)

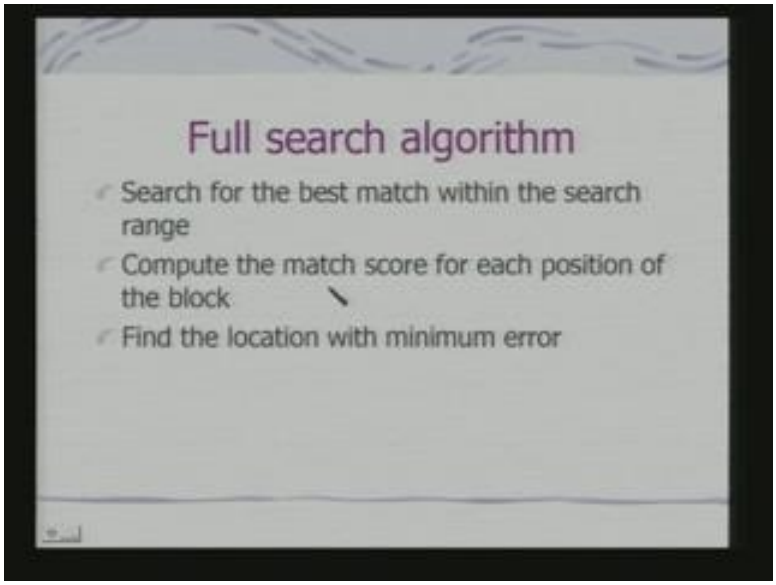


So, if we look at more technically what we have got is that in MPEG, the image is divided into what are called 16 cross 16 macro blocks, now the 16 cross 16 macro blocks have to be searched, in the search area to find the best match that means, I find the best match for the corresponding macro block. So, here in this image if we this is my reference picture, in the reference picture I, have to search over the search area, to find the best match for this block.

So, what we are doing is, this is the basic idea is if I consider this left most corner as the anchoring point for the macro block, then this is giving me the shift, x plus u and y plus v actually I am interest it to stored u and v which minimizes this error, I would like to stored u and v, which minimizes this error. So, that would gives me the motion option and I need not store, the data actual pixel data.

So, we have to search around this and at each position therefore, at each position of this anchoring point I need to compute this function, this is actually an absolute error. Summation of your absolute error over each pixel in a macro block, this is clear.

(Refer Slide Time: 47:28)



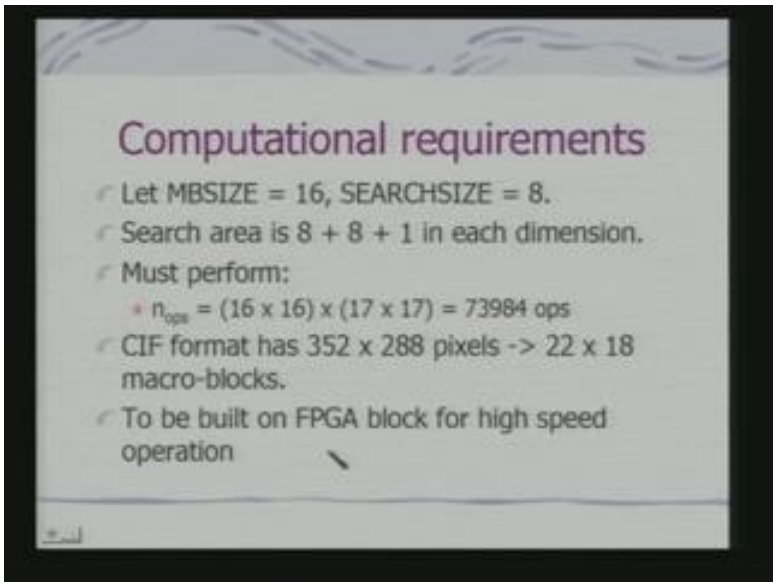
And I would like to find out u and v which would minimize this. So, what is the algorithm to be used, we are considering here the most inefficient algorithm, but we call of full search algorithm; that means, you search for the best match within the search range and search at each position; that means, compute the match score for each position of the block. So, I shall shift x y to each pixel position over the search area and compute the match score. Fine and then find the location with the minimum error. So, this is the algorithm and list efficient algorithm, which is called full search algorithm.

Now, one point you should keep in mind is that, this searching algorithm is not part of the standard. So, if you are making a video camera and if you can do this process well and first; that means, your camera is better compared to your competitor's camera and both the cameras are at the same time compared able to your MPEG standard. And that is

the one of the reason and the motivation behind looking at different algorithms as well as hardware implementation for solving this problem.

In-fact same thing is true for DCT computation also the standard talks about using DCT, but what is the algorithm what is the hardware algorithm to be used for computing DCT is not part of the standard. So, you have the option to innovate and get better design for developing this kind of embedded systems. And in fact, that whole idea terms from why we are talking about a platform based design platform pixels of certain aspects, certain aspects are open the standard pixels of certain aspects, certain aspects are open. So, you can exploit the open part get a better design on a platform.

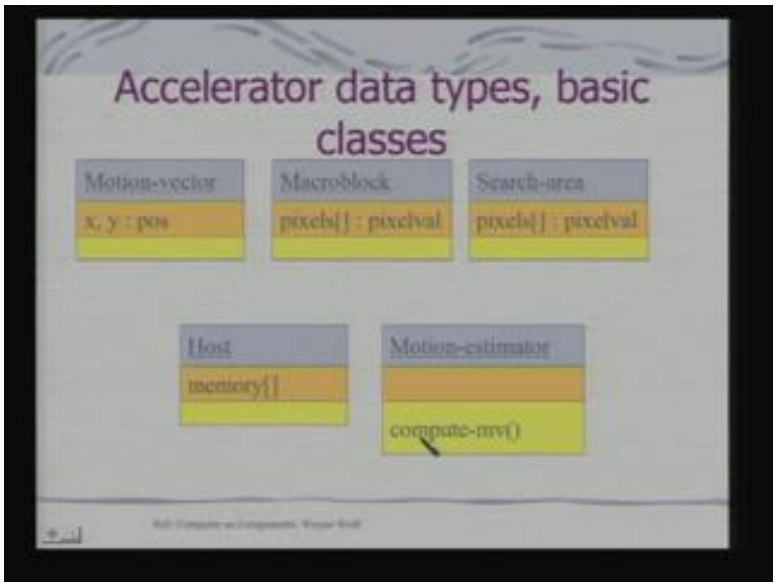
(Refer Slide Time: 49:28)



So, what is the computational requirements MBSIZE is typically 16 cross 16 and SEARCHSIZE is over 8 cross 8 window. So, your search area is 8 plus 8 in each dimension. So, therefore you need to perform how many operations, 16 cross 16 into 17 cross 17 operations this operations is actually difference computational operations. And typically a CIF format, which is a smaller format frames has got 352 into 286 pixels, which translates to 22 into18 macro blocks.

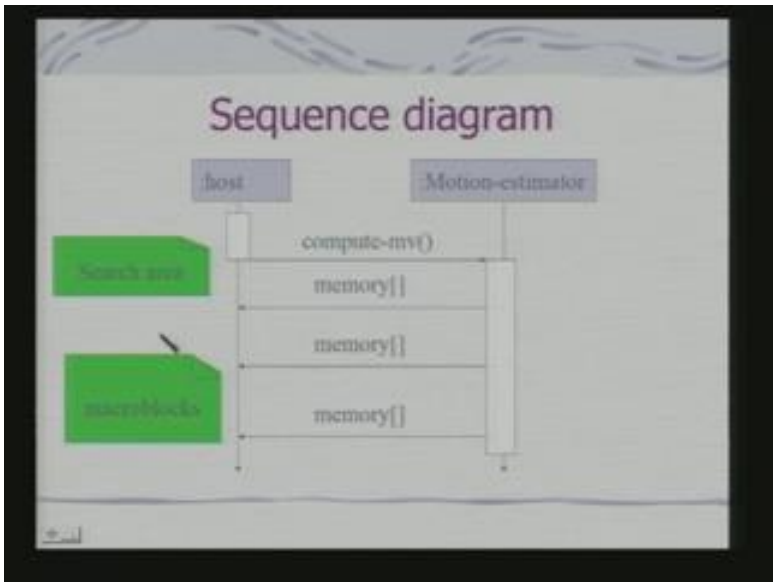
And if for each macro block, if this computation is to be done you can realize how many times therefore this difference has is to be there, that is the basic time consuming part of the compression scheme and this is to be built on FPGA block, for high speed operation.

(Refer Slide Time: 50:24)



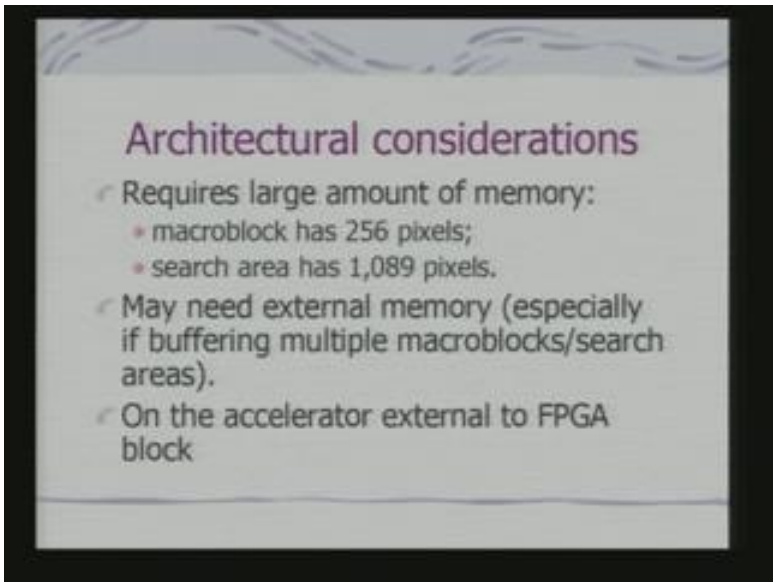
So, let us look at the basic design, if we are coming from a pure object oriented approach we have to identify the basic data types for doing this computation. Data types are what the motion vector, macro block and the search area fine, these are in-fact the classes are the objects, which will be using for doing the computation. This classes will also in capsulate the hardware resources. So, I am showing a top level resource, which is host processor along with the memory, where the data would be there and you have got a motion estimator, which is actually the accelerator block. That we are trying to design and the operation that is the request the processor will make will be compute the motion vector.

(Refer Slide Time: 51: 23)



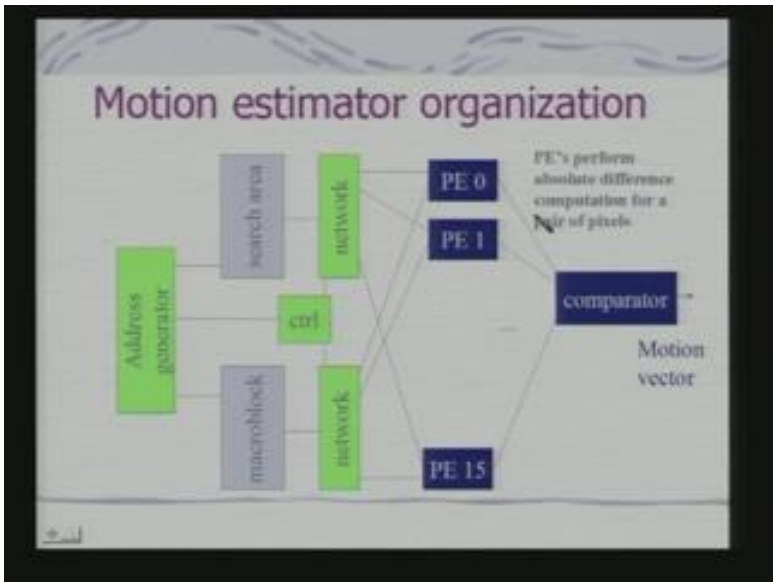
Now, these should be represented through a sequence diagram, so you have got the host this is an instance of a host, this is the instance of the motion estimator, this is the search area and the macro block, so what you have got. So, the host actually makes the request to the accelerator for computing the motion vector. On receiving the request, the motion estimator should get the data about the search area and the macro blocks, so I need to get the memory request on to the host. So, it implies that is from the sequence diagram itself you can realize, there has to be a substantial data movement between your memory and the FPGA block.

(Refer Slide Time: 52:09)



So, what we say that requires large amount of memory, macro block has got 256 pixels search area is 1089 pixels. And if such a large memory transfers has to be done that means, your accelerator needs to be changed as well. So, you need external memory, especially if buffering multiple macro blocks or search areas is required. And that would be on the accelerator external to the FPGA block, but that memory is not part of the host memory. So, you may have a DMA block, a DMA controller to transfer the data from the host memory to that of the memory on the accelerator block.

(Refer Slide Time: 52: 54)

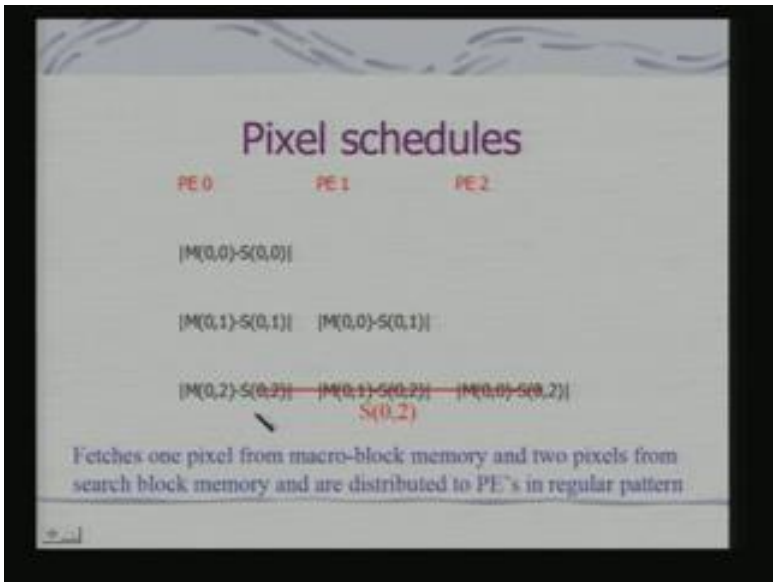


So, this is one of the possible architecture, if you see here this is the memory macro block and the search area this contains the data, this is the basic address generator and this PEs are implemented on the FPGA block. Each PE does what, perform absolute difference computation for a pair of pixels and your comparator takes the data from PE integrates it together and produces the motion vector corresponding to the best value.

The interesting thing is that this is the network which connects the data from the macro block and the search area to the PE, depending on how the scanning is taking place. Because you are scanning the macro block over the entire search area, depending on the scan this interconnection has to be done and which data is to be selected is given by the address generator. So, address generator is actually taking care of the scanning task.

So, if you want to implement a different algorithm from full search, the address generator logic has to be different. And that they need implement and different logic, you might not like to search at each pixel, you might like to sample and search. So, these become the basic organization, in terms of the FPGA block.

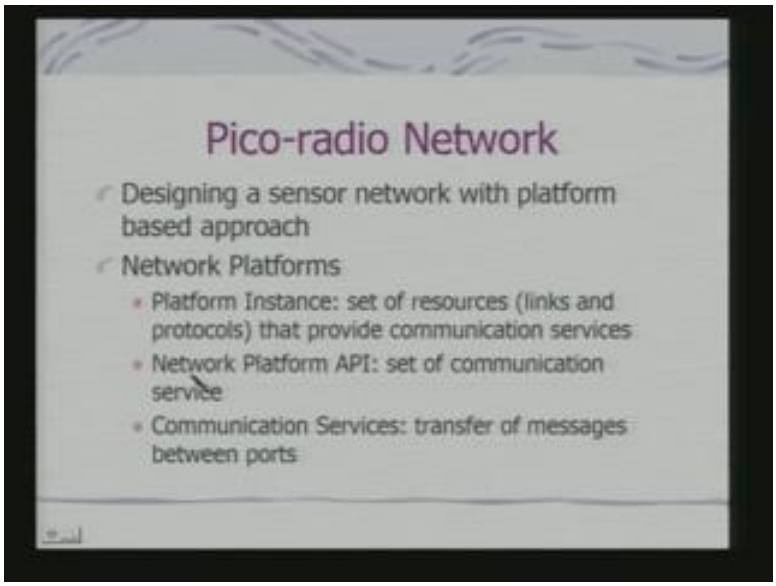
(Refer Slide Time: 54:14)



So, the pixels how they are schedule, because that scheduling is an important factor for doing the design, if you see that if I have PE 0 PE 1 and PE 2 from a macro block, I can have one pixel from macro block memory and two pixels from search block memory and if they are distributed these operations can be scheduled, simultaneously. So, I can have same pixels coming in and I can do this computation. And therefore, these pixels are distributed in a regular pattern and this distribution is done with the help of the network and that organization is a key issue in terms of this design.

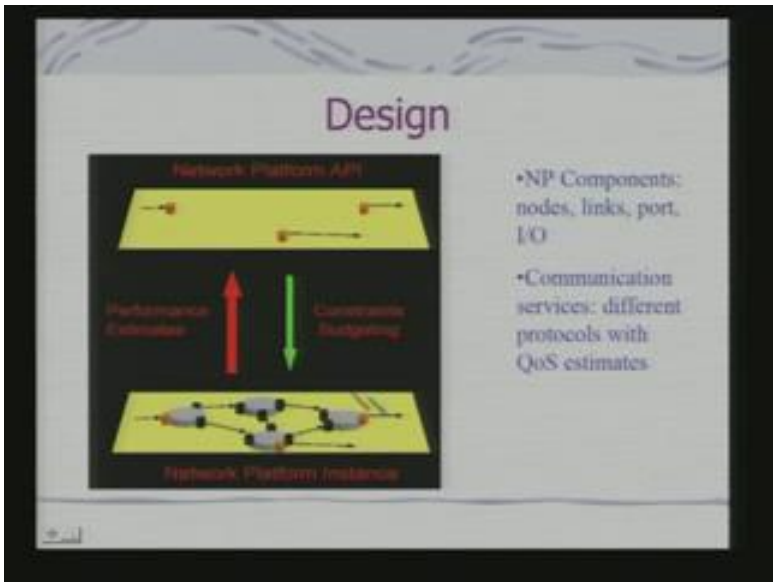
So, here you will find that S 0 2, how it is taking care of these same data, this is your same S 0 2 and I can use the same data over multiple macro block pixels and these macro block pixels, a traversing along the diagonal path over the PEs. So, that traversing and fetching regularity pattern gives you the efficiency in computation. Now, we shall look at very briefly another example in the network platform.

(Refer Slide Time: 55: 29)



This problem is that of designing a sensor network with platform based approach. So, if you are talking about a network platform, a platform instance will be a set of resources links and protocols that provide communication services, you got to have a network platform API for providing a set of communication services. And set of communication services is mechanism could be used to transfer messages between ports, because these are the elements, which have to be used to characterize to network platform. So, your network platform talks about a variety of possible instances of the resources that we will be using.

(Refer Slide Time: 56: 07)



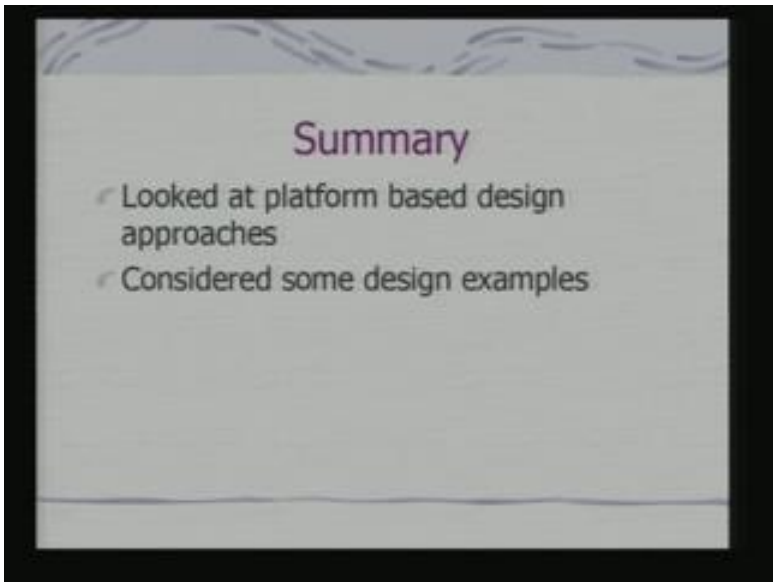
So, let us look at a very simple example, see if I have a network platform API, by which you can make a choice about your network components nodes, links and ports IO. What will be doing, you will be actually deciding ((Refer Time: 56:25)) configuration and the protocol using the API mapping on to that will give you a network platform instance, you do a performance estimates on that network platform instance. See whether your budgets constraints are getting satisfied or not.

So that means, when you are doing the API through the API specifying the architecture. Because, specifying the architecture primarily would been specifying the communication service of architecture that is the different protocols and with this protocols and the constraints budgets, you get one particular network platform instance. And on the basis of that you get the performance estimates. So, one possibility could be is it a single half network and a multi half network, if it is a multi half network, then you might have the better energy estimate.

You might have a better Qos, so those things can be checked up through this API and have the complete distributed embedded system design. So, in fact, the reason for looking at briefly this example is to understand the distributed systems can also be designed through this platform based approach. Because here your hardware elements are what your nodes, different parameters for the nodes, different parameters for your communication link.

The software elements are your protocols everything put together with the budget constraints can be tried out obtain the performance estimate. And that specifically your search in the possible space of alternatives formulations, to get an identify the optimal configuration for your network.

(Refer Slide Time: 58: 16)



So, what we have looked at therefore, at platform based design approaches and considered design examples this moral less brings us to the end of basic design schemes for embedded systems. In the next class, we shall look at some of the software related issues, particularly compiler related issues which can be used for optimal design.