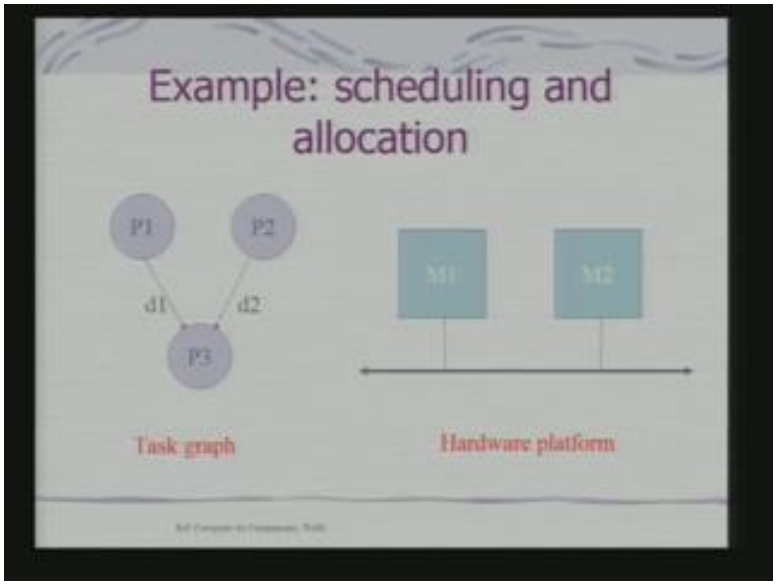
Embedded Systems
Dr. Santanu Chaudhury
Department of Electrical Engineering
Indian Institute of Technology, Delhi

Lecture - 32
Designing Embedded Systems – V

We are discussing the different strategies to be followed for Designing Embedded Systems. In the last class we had looked at the hardware, software partitioning problem. And we have seen, how the tasks can be mapped to different processing elements, depending on the cost constraints. Mapping of the tasks are simply scheduling of the tasks.

(Refer Slide Time: 01:28)

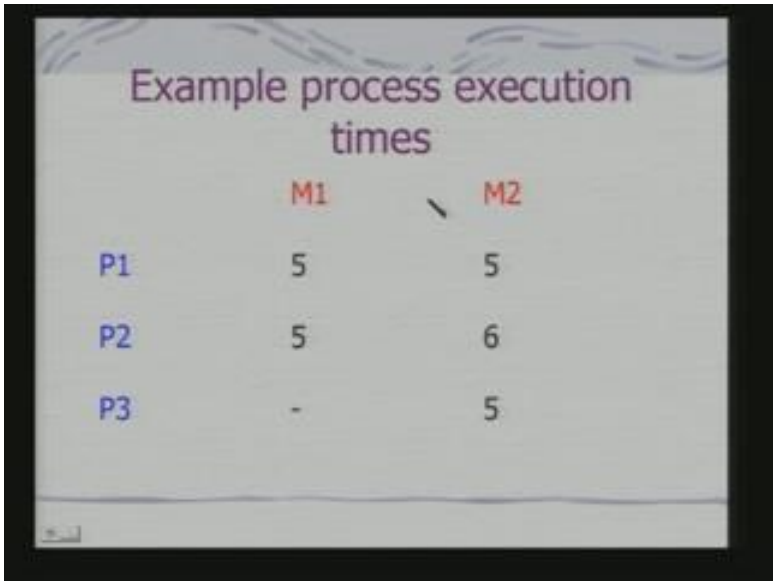


Trying to do partitioning and scheduling that is allocation and scheduling together is a complex problem. But, scheduling in a way is related to allocation as well. Here, we shall look at a simple example to understand this problem. Let us consider, this simple task graph in fact we have looked at similar task graph in the past as well.

Now, here this d 1 and d 2 indicate the communication cost between the tasks. And let us consider, that we have got a hardware platform. That is, M 1 and M 2 consisting of two processors. The problem is that of mapping, the tasks on to this hardware elements. And the question is, if we are mapping them optimally. That means, if we are doing the

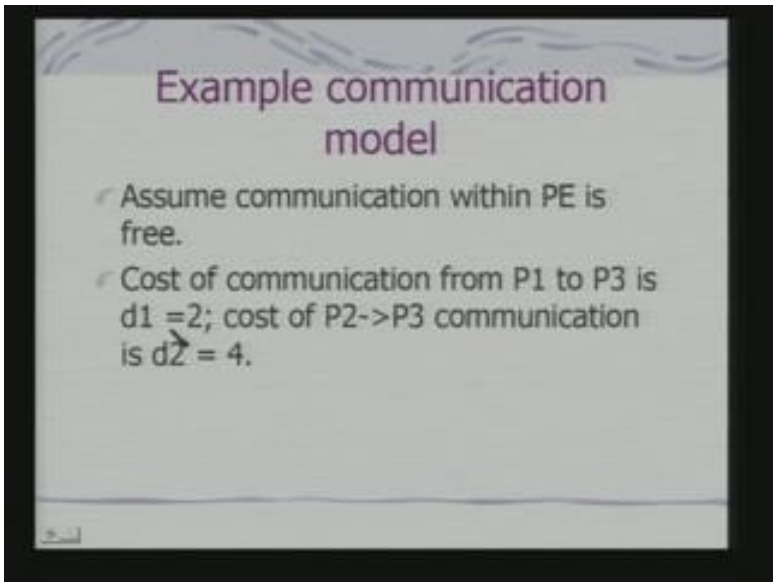
allocation of the task to processor optimally. Then, that would also determine the schedule. And hence provide the result in minimum time.

(Refer Slide Time: 02:46)



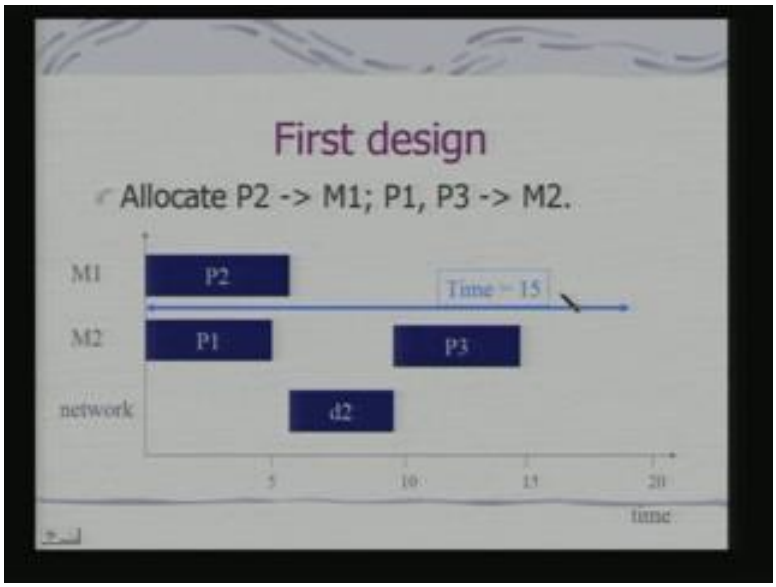
Let us consider, that this is the cost or this is the execution time parameter. That, we already have for the purpose of mapping. So, I have got p 1, which takes on both the processors time 5 units P 2 takes 5 on M 1 and 6 on M 2. And P 3 can only be executed on M 2 because of its special nature and it takes time 5 units. Now, what will be the optimal allocation, such that the complete task graph can be completed in minimum time. That means, it implies scheduling as well.

(Refer Slide Time: 03:30)



Assume communication within PE is free. That means, there is no delay involved there. And cost of communication from P 1 to P 3 is 2 and cost of communication from P 2 to P 3 is 4.

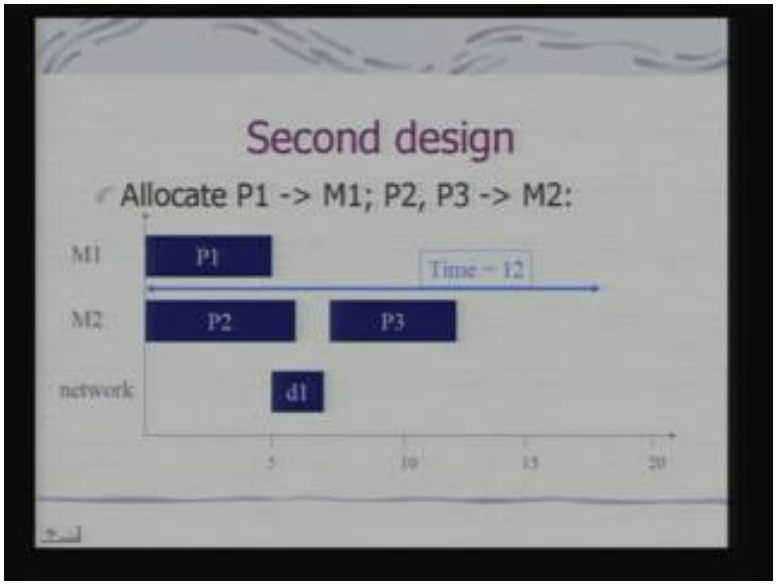
(Refer Slide Time: 03:50)



So, with this cost if you look into it we can have one design, where P 2 gets mapped to M 1 and P 1 and P 3 gets mapped to d 2. So, in that case we are paying for this communication cost because, I need to have the result from P 2 coming to P 3 for the

final computation. So, the total time taken would be summation of this path, whether and it is turns out to be 15.

(Refer Slide Time: 04:24)

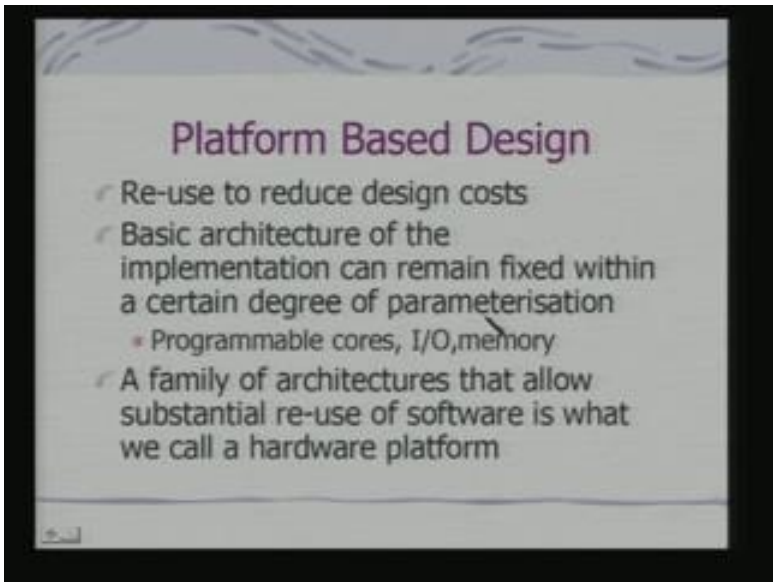


Now, if we look at another design here my allocation is that I allocate P 1 to M 1 and P 2 and P 3 to M 2. That means, effectively I have interchanged P 1 and P 2 in terms of it is allocation scheme. Here, we will find that the cost is much less for communication because, it is between the P 1 and P 3.

And so the total time required turns out to be 12, which is less compared to the previous one. So, the whole point is, that if we are trying to do the design, in terms of optimizing time as well as may be satisfying timing constraints. We need to look at allocation, as well as the scheduling problem in a integrated fashion.

In fact, partitioning, allocation and scheduling should be considered in an integrated fashion. And that provides as the basic frame work or overall picture of hardware, software co design and hardware, software partitioning. Now, picking up Q from there we shall look into a new methodology of design, which we called platform based design.

(Refer Slide Time: 05:49)



Now, here the basic motivation is to reuse elements or components, that has been designed to reduce design costs. It similar in philosophy with that of your object oriented design and component based design, where we say that if you already have software components, we shall block them in to develop an application.

Here, we are talking about components, which are just not software components. But, we are talking about both hardware as well as software components. So, what we say the basic architecture of the implementation can remain fixed within a certain degree of parameterization. So, we have programmable cores I/O memory.

A family of architectures that allow substantial reuse of software is what we call a hardware platform. In fact, this definition we had already considered, when we discussed SOCs. Now, again we have been going back to the platforms to understand the philosophy of embedded system design around platforms.

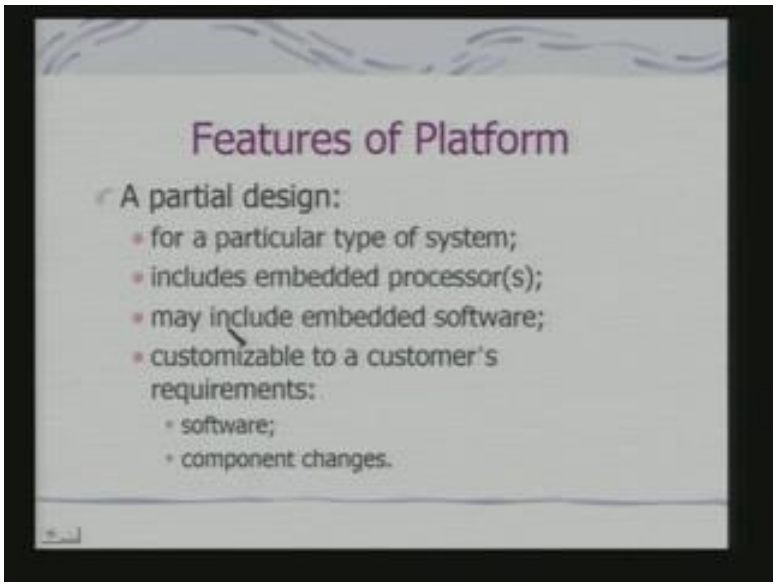
Now, in fact one point you should understand very clearly, that when we are talking about platform. We are talking about an architecture, with certain permissible parameterization. That means, when we are searching for that architecture, when we are searching for alternatives to do my task graph mapping the set of possible architectures. Therefore, gets defined on the basis of parameterization, that is available on a platform. And that gives as a basic benefit in terms of design around platform.

(Refer Slide Time: 07:41)

So, what is really a system platform because, platform cannot exist independent of the software. One part of the platform is; obviously, the hardware platform. On top of that, we got to have software layer, which wraps different parts of the hardware. You can have programmable cores, memory subsystems all these things being manages via RTOS.

I/O subsystem managed via device drivers network connection via the network communication subsystem. And these three things together can form the API. And this API can be used for developing application on the platform. And these layer is called software platform. And combination of both hardware and software platform is called the system platform.

(Refer Slide Time: 08:42)



So, what will be the features of platform, in a sense platform represents a partial design for a particular type of system. It includes embedded processors, may include embedded software three APIs actually, they do include software. And which can be customized to customers requirements by changing the software or through component changes.

(Refer Slide Time: 09:13)

So, once we have that; obviously, this would enable design reuse. Design reuse can be achieved by sharing components, among different products. This is called reuse in space, this can be also done across different product generations, which is called reuse in time.

In fact, the whole id of spaces that if you are simultaneously developing to different products, in some why differentiated in term of functionality, you can have your components shared between the products.

So, that is why it is a reuse in space else, if it use over different generation of products it would become reuse our time. So, effectively what if these products or what are these would not really the products, what are these components? This components in a way or nothing but, a library of functions and hardware and software implementations of these functions, which can be used across different designs.

So, effectively if I have got a platform, which talks about a parameterized hardware, along with the set of APIs define and top of that. And if we know, different kind of functions have been implemented on this platform. And if we have them in a library, then for a new design I can simply pick up those functions and use them in the design.

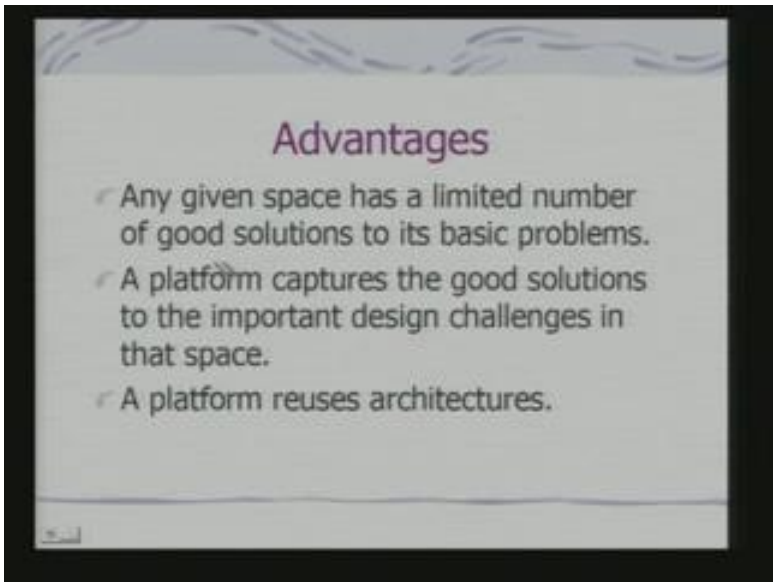
And what is the advantage of this. Since, they have been reused the estimation of the performance parameters, which is actually is the key point in doing hardware, software core design, becomes much more reliably calculated, generated optic. So, both at hardware and software level we can do design reuse.

By fixing hardware architecture, customization effort is entirely at software level. And basic software like RTOS and device drivers can be shared, if developed using proper methodology. And what is the interesting node is that, decomposition of system function is essential for design reuse. In fact, there is definitely tradeoff between the granularity at each you consider the system functions.

If you consider them at a smaller granularity then, these functions may be common across multiple applications. So, then you can use those functions straight away. But, if you are looking at a smaller granularity, then that be lead to increase complexity in the overall design. So, there is the basic tradeoff between the two.

But, the case is that each application is unique in shown way. So, until and unless you can define a proper decomposition of the system function, you cannot really reuse the library components. That is, why decomposition is an essential aspect of this kind of platform base design.

(Refer Slide Time: 12:40)



So, what are the advantages. Any given space has a limited number of good solutions to the basic problems. So, what is this space we are talking about an architecture space. That means, different possible architectures. And that may be related also two applications. So, any given space has a limited number of good solutions to it is basic problems.

So, when we are actually searching, if you look at hardware, software core design problem it is basically a search problem. Now, where are you searching, you cannot in an unbounded space. You would like to search among a set of possible good solutions. A platform captures, the good solutions to important design challenges in that space. And that is why you search gets small focused. And through that, you can actually reuse the good aspects of the good design.

(Refer Slide Time: 13:40)

So, this leads to fast time to market and you are reusing system design an hardware, both software and allows chip to be customized to add value. That means, using your software programmability or either hardware programmability, on the basic design, you can add value.

(Refer Slide Time: 14:01)

In fact standards have actually encouraged development of this platforms as well because, when we are looking at embedded systems, the standards a big role. Because, many of our appliances are get driven to say, wireless communication, multimedia

planes as like you say MP3 player or DVD player. As well as, network base applications and these three domains are completely standards driven.

So, standard defines the basic I/O requirements. Standard defines the basic processing aspects. So, you cannot really have a DVD player, which does not implement a DCT decoder, that is IDCT implementation because, DCT forms the basic part of your MPEG coding scheme.

So, in that sense standards provide you a basic framework of functionalities, which have to be implemented in the target of planes. So, standards that way actually motivates and supports platform based developments.

(Refer Slide Time: 15:25)

So, developer chooses implementation of standards functions, improved quality, lower power all these are issues. And products may be differentiated by added features. All of the cell phones to provide the basic, if they are working on the GSM standard provide the basic GSM speech decoded.

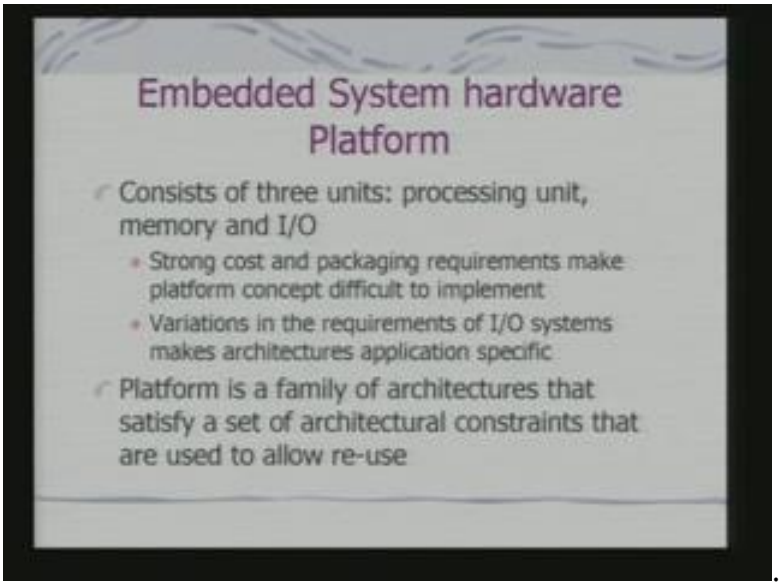
But, the user interface can be different, all of them can stored pictures in JPEG all of them has to stored if they are storing video in MPEG for ((Refer Time: 15:57)). So, they are the basic functionalities can remain same. But, the user interface can be different and this user interface can be implemented on top of your basic platform architecture.

So, standards in that sense encouraged platform based design. And today's whole approach for designing this kind of appliances or platform based.

(Refer Slide Time: 16:22)

So, the trade off in this case is platform has many fewer degrees of freedom, that is absolutely true. And it is harder to differentiate, but can analyze design characteristics that is something which is very important. On your other hand, if you want to deal the completely full custom design, it would imply long design cycles. In fact, this is the basic trade off, which really encourages this kind of development.

(Refer Slide Time: 16:50)



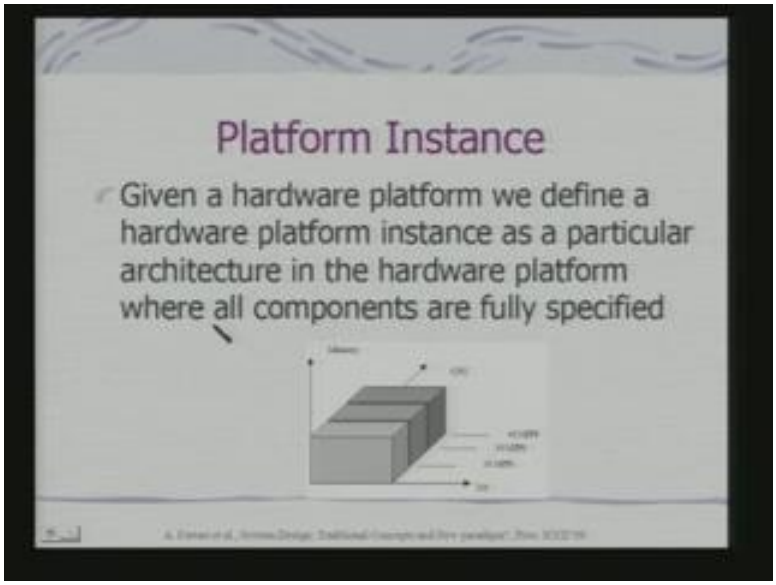
So, for embedded system hardware platform, what do you expect in such a platform. You would expect processing unit memory and I/O. But, strong cost and packaging requirements make platform concept, difficult to implement. Because, they can form different kinds of requirements, in terms of the cost and packaging because, packaging also decides the size.

Variations requirements of I/O systems makes architectures applications specific. So, you cannot have all source of I/Os implemented in a platform. Because, many of this appliances actually differ in terms of your I/Os can be support. In fact, standards in a way help you to handle this problem because, standards tell you what kind of I/Os to be handled, what kind of processing to be handled.

But, if you look at in a general purpose fashion then, these are two strong problems and typical problems, which a needed to be toggle for designing platforms. In fact, platform is a family of architectures, that satisfy a set of architectural constraints that are used to allow reuse. So, actually you want encounter or you want like to have may be a very general purpose platform, you cannot have that.

Because, these are the two factors, which makes that the platforms not possible for us to design platforms is that absolutely general purpose. So, many cases they will be applications specific. Now, applications specific will determine a set of constraints, which that architectures should satisfied.

(Refer Slide Time: 18:33)



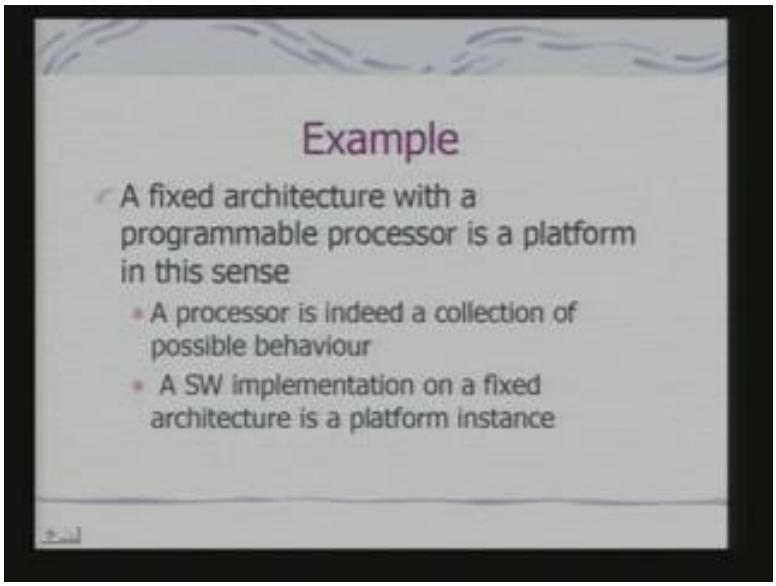
So, let us look an example, this is the mode of the conceptual example. Given a hardware platform we define a hardware platform instance, as a particular architecture in the hardware platform, where all components are fully specified. So, in a platform architecture some of the components may not be specified. Say for example, this is the conceptual model.

But, it says that we may have a platform on which we can have CPUs of different speeds, we can have different I/O characteristics and you can have different support from memory. And we can select an instance among this platform for implementation of my application. And instance will be what? An instance will be as CPU, which operates at a particular speed depending my application it can be 30 meet CPU with the certain memory. And capability of supporting may be 4 digital I/O channels.

So, that becomes application specific. And in this case what is the platform? In this case platform is not really achieved. If you look at do it platform provides you with the kind of a soft core kind of a framework. And using this soft core framework, you can actually get the platform instance to implement your function.

See, if you are searching for an architectural mapping, where we will be searching? You will be searching among, the different possibilities that you can have with these combinations. So, you have been provided with say soft core, which can be effectively parameterized, in terms of these parameters. You are searching for the best core, which would meet your requirement. And that is what is your application based platform instance, it is not a platform, but platform instance.

(Refer Slide Time: 20:52)

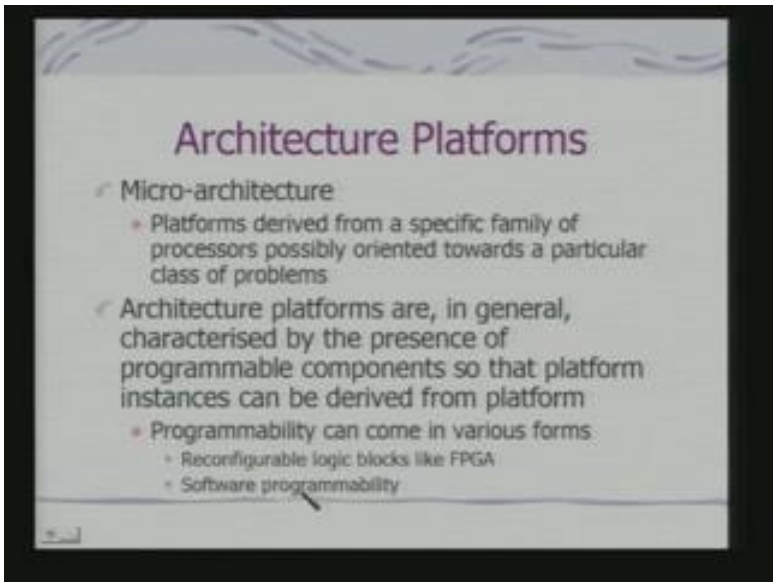


On the other hand, if you look into it. So, a fixed architecture with the programmable processor is a platform in this sense because, a processor is indeed a collection of possible behavior. And a software implementation on a fixed architecture is a platform instance. So, in that sense if you consider your PC using this definition PC itself is the platform.

Because, you can develop various applications on top of PC using pure software. A software implementation on a fixed architecture, becomes a platform instance. So, there are, so what I had shown in this two examples is that, in one case you can actually parameterize in terms of hardware characteristics, go through the synthesis process and build the platform instance, which is meeting your applications.

Other extreme could be, you can worked with the fixed architecture. And modify your software. And software will define a particular platform instance, meeting your functionality. Obviously, there would be variations of these which are in between.

(Refer Slide Time: 22:14)

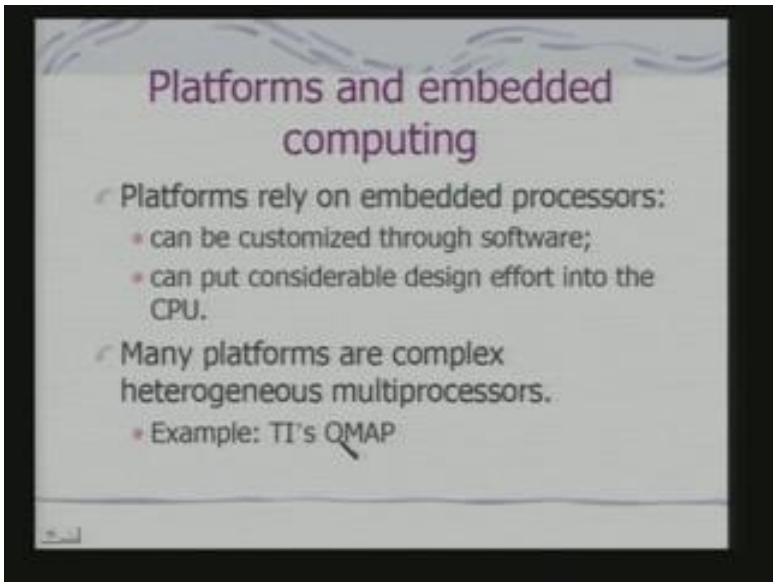


In fact architecture platforms typically at micro architecture based. In fact, what we say is that, platforms derived from a specific family of processors, possibly oriented towards a particular class of problems. So, the platforms which provide the basic hardware architecture. A typically build around some processor having it can be an arm family, it can be even around say other processors, like your big family.

So, there at typically build around processor family. And many of this platforms are characterized by the presence of programmable components. So, the platform instance can be derived from the platform. Now, this programmable component can come in variety of forms. One possibility is having a reconfigurable logic blocks like FPGA.

Other possibility is software programmability. So, obviously, if I can put in the software, the behavior changes and you get an instance. On the other hand I can program FPGA to implement some special functions, depending on my application need to suit the platform.

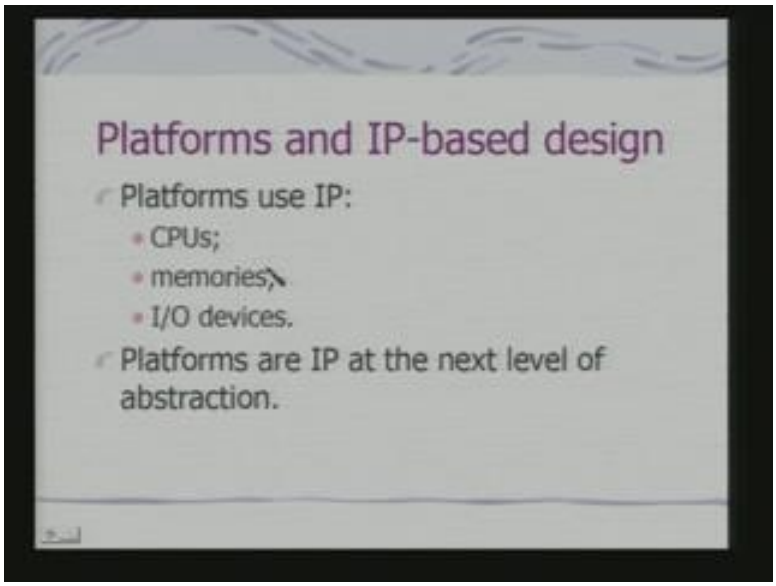
(Refer Slide Time: 23:34)



In fact, bending of the platforms rely on embedded processors and not just a single processor, but a number of processors. So, that there behavior can be customize to software. And that can put considerable design effort into the CPU. But, that design effort goes in by will making of the platform.

And many platforms are therefore, complex heterogeneous, multiprocessors. Example is TIs OMAP which we are already discussed, which has got arm 9, as well as a DSP processors sitting inside. So, the behavior of these will be determined by mapping software on to those to processors. And that would give a particular instance, suiting my application.

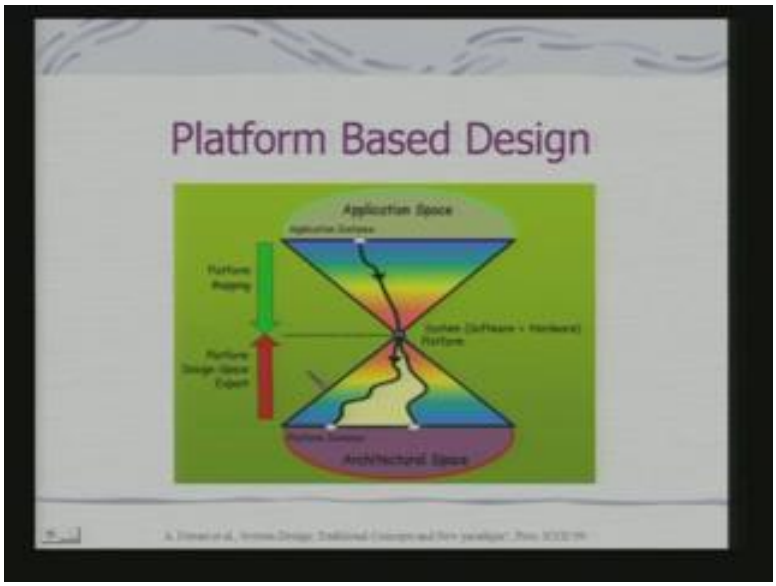
(Refer Slide Time: 24:33)



In fact platforms use various kinds of intellectual property and soft cores, like CPUs, memories, I/O devices. And in a sense platforms are IP at the next level of abstraction. So, if you are designing a chip using say pure soft core. So, we are using IP at a lower level to which I can add my own memory.

But, when we are talking about that use of CPU, memory and I/O devices, everything together we are using IP at a next level of abstraction. In fact, one example we are already talked about that, we can think in terms of a pure soft core based platform, where you may parameterize architecture in terms of it is processing speed, memory usage, I/O channels. And then, decide to fabricate that platform instance for your application.

(Refer Slide Time: 25:36)



So, conceptually what we are talking about is, say this is my application space, possible application. And I am dealing with an application instance. When, I am dealing with an application instance I know the basic properties and characteristics of the application. And then, I need to do what we call a platform mapping. That is, mapping in terms of system platform, that is pure software as well as hardware.

And we have got, if you look into this architectural space. So, this architectural space consists of set of possible architectures. So, once we have mapped this application instance to a set of software or hardware. In terms of here what we have got? We have got a set of functionalities depending on the application instance. So, on the basis of functionality, we shall try to match them with different platform instances.

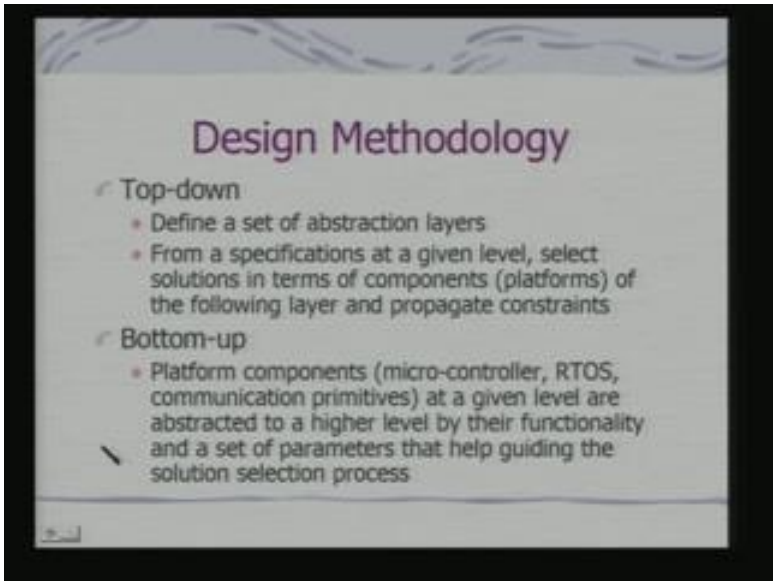
Because, the platform asserts would cover this architecture space. Because, platform has got a kind of a programmable component, parameterizable components. So, a platform correspond to a set of possible architectures. So, instances of the platform covers, this architectural space. And I have got therefore this platform instances.

So, this characteristics which is being mapped onto a system platform, that would lead to a platform instance. So, let us try to understand what we are talking about? We are coming from application got an application instance depending on an application instance we have got a set of functionalities. These, functionalities actually motivate or to make a choice about a system platform.

The system platform corresponds to a set of possible architectures. I search among the set of possible architectures to choose a platform instance, which will implement my application instance. So, in a sense if you see, that you design process is not strictly a top down design process.

In fact, it is a one way is platform mapping, that is the mapping to the platform. And next thing is the platform design space export. That is, there are different possibilities in the platform design space. And form there we are mapping and trying to get the meeting point here. So, this is the basic philosophy of what we called platform based design.

(Refer Slide Time: 28:30)



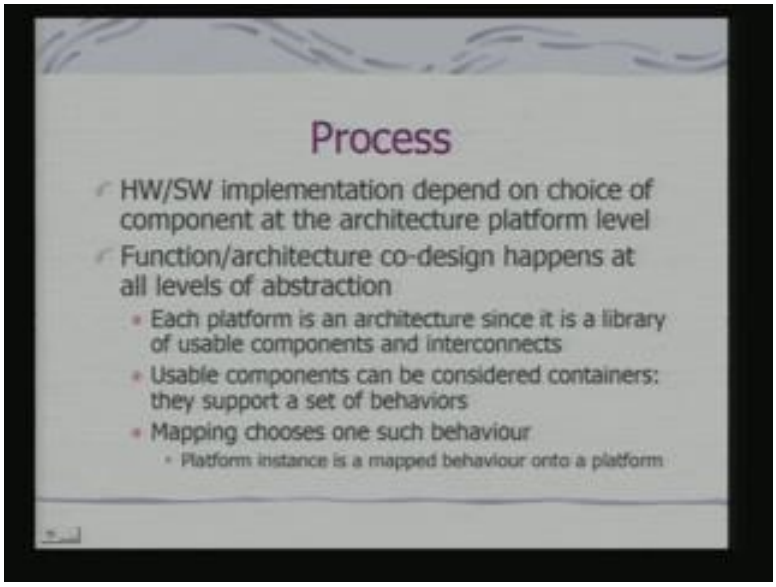
So, the design methodology is combination of top down and bottom up. The top down design define a set of abstraction layers from specifications at a given level, select solutions in terms of components, which a platforms of the following layer. And propagate constraints. That means, you got to low what are the components you required in terms of the functionalities, that your application required.

And those components would put in constraints. And that constraints you need to propagate to get to the actual instance. So, when you are looking at a bottom up part of it, the platform components, which we are micro controller, RTOS, communication primitives. At a given level are abstracted to a higher level by their functionality. And a set of parameters that help guiding the solution selection process.

Because, these two things have to match, here we are coming from a top level abstraction in terms of functionality. And for bottom we are going up in terms of raw components, the micro controllers, the basic RTOS, the communication primitives. And the bottom up process goes through a level of abstraction.

So, there these two has to match. If they do not match, there we cannot really to an implementation. So, a design methodology becomes a combination of a top down and the bottom processors.

(Refer Slide Time: 30:14)



In fact, hardware, software implementation depending on choice of component at the architecture platform level. And functional architecture co design, happens at all levels of abstraction. Because, each platform is an architecture, since it is a library of usable components and interconnects. And usable components can be considered as constrainers. And they support a set of behaviors.

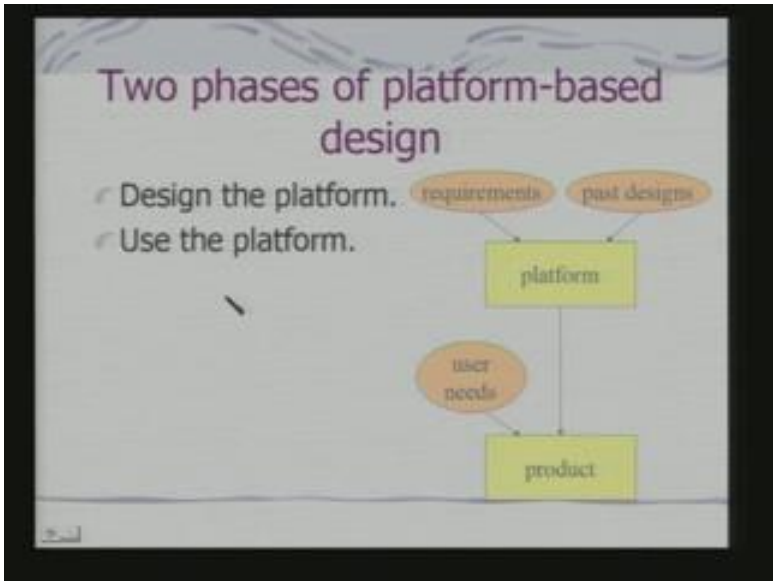
And mapping chooses one such behavior. And platform instance is a mapped behavior onto a platform. So, if we are coming from top, we are coming from application. Going up from the bottom we are going from platform and we need to do the mapping. So, that the functionality defines into behavior of a platform.

And that behavior of the platform makes it a platform instance. So, takes a very simple example from a software domain. Now, I can have an object representing a triangle. And

my problem is that of making a drawing and that defines triangle with a certain set of parameters. So, these parameters have to be mapped to the triangle object. And I have to created an instance of the triangle object, which realizes the drawing primitives.

It is exactly like that, from function what we are getting from application, we are getting functional specifications. And from platform, we are getting components, when we need to map functions to components. The mapping of function to components defines behaviors. And that could lead to a particular instance of the platform, which is actually your embedded systems that we design.

(Refer Slide Time: 32:26)



In fact, there are two phases of platform based design. So, what first phase is platform on the product. So, first thing is design the platform and use the platform. So, requirements on the phase designs can motivate development of platform. And depending on exact product, that you are developing you will be developing an instance of the platform.

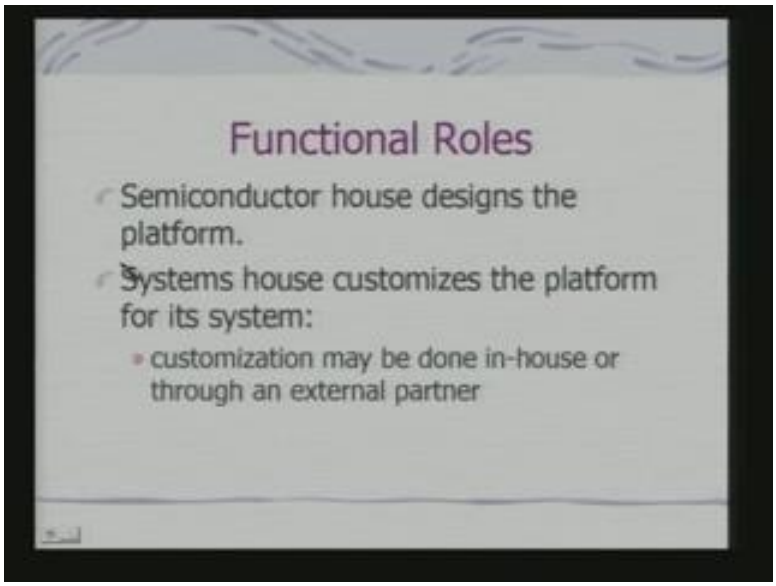
So, if I go back to TIs OMAP example, depending on the requirements of multimedia communication. The features, which are required has been analyzed, designs have been looked into and different components have been put together, to create the hardware platform, as well as with the software support for it.

And individual appliances, say cell phone from NOKIA. They, would use the TI OMAP say for example, with a particular set of functionalities. That is the particular set of behavior being implemented on top of that platform.

(Refer Slide Time: 33:27)

So, in a sense there is the division of labor. The platform design and platform based product design. So, platform design is choose characterize hardware units create the system architecture, optimize for performance power. And platform based product design is modify hardware architecture. And optimize your programs. So, that you get the best performance.

(Refer Slide Time: 33:54)



And in the ((Refer Time: 33:55)) the functional roles also diversify. Semiconductor house designs the platform. And systems house customizes the platform for it is system. Customization may be done in house or may be through an external partner. So, here the example says TI is designing the platform and NOKIA is developing the product.

(Refer Slide Time: 34:18)

So, what are the platform design challenges. Does it satisfy the applications basic requirements is it sufficiently customizable and in right ways, is it cost effective. These are basic issues, which have to be addressed. How long does it take to turn a platform

into a product? Because, if it takes lots of time to turn a platform into a product, then a platform will not be used. So, usability of the platform is also an important point. So, you cannot make your platform unnecessarily complex. So, that a user or an application developer cannot understand and use it.

(Refer Slide Time: 35:02)

So, what is the methodology, size the problem, what does that means. That means, you define an application look at the space of possible applications. And then, have an assessment of size of the problem space, size the problem essentially differs to sizing of the problem space. Because, an application can involve a number of problems.

And that problem space have to be assessed. Develop an initial architecture evaluate for performance and power. Then, evaluate customizability, improve platform after each use and get embedded design.

(Refer Slide Time: 35:50)

So, what we have telling in a way, that platform must satisfy constraints of the application domain. See, if you look at cell phone applications, it should support the wireless communication. It should support the signal processing capability, because I have to processed a speech signal I have processed a video signal.

And it should support also the various kinds of constraints, timing constraints of the application domain. And design of the platform is tradeoff between size of the application space, that can be supported by the architecture belonging to the platform. And size of the architecture space, that satisfies constraints embedded in architecture platform definition.

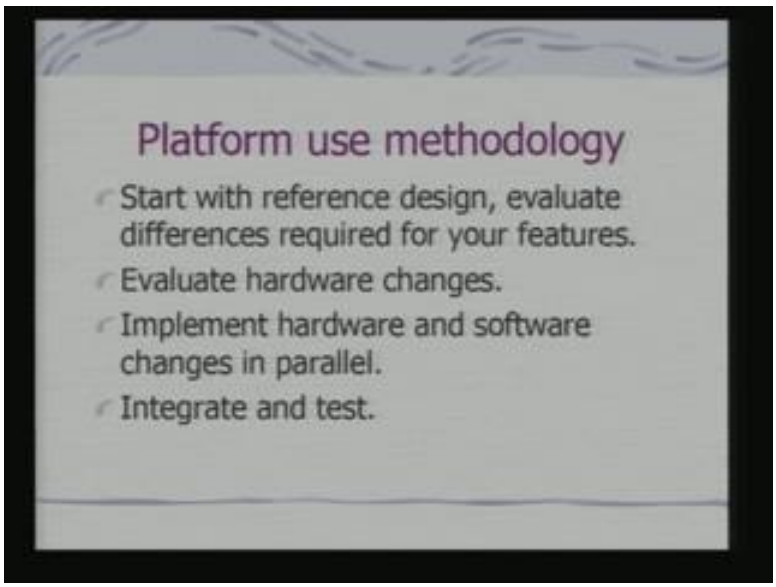
So, these are the two points, we should be clearly understood. This talks about coverage of the application space. What are the different aspects of the application space have been addressed by the platform. These, refers to the architecture space, what kind of flexibility you are giving to a product designer to explore alternate architectures for implementing functionalities.

So, there has to be a tradeoff between the two. Because, if we are trying to a address the entire application space, that you may not be left with the flexibilities, that can be provided to the final product of level. Because, your system may be complex enough to permit further flexible features. So, design is basically tradeoff between these two aspects.

(Refer Slide Time: 37:36)

Therefore, use challenges are what. How do I understand the platform's design? How do I modify it to suit my needs? And how do I optimize for performance, power, etcetera.

(Refer Slide Time: 37:50)



So, platform use methodology, start with reference design because, typically it will do with reference design. Evaluate differences required for your features, evaluate hardware changes. Implement hardware and software changes in parallel. Integrate and test. Wherever, the hardware changes are possible do the changes. And integrate your software and hardware and do the test.

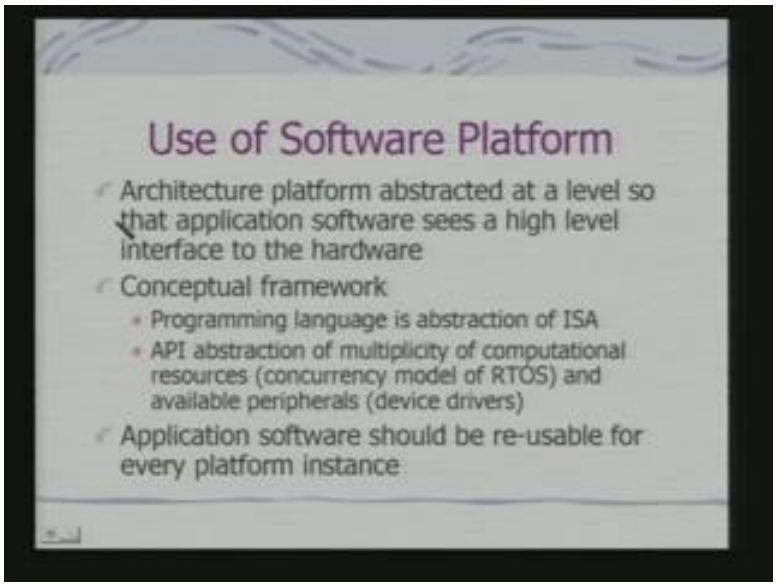
(Refer Slide Time: 38:22)

So, effectively what we are talking about it is the design exploration. Because, once an architecture platform has been selected, then the design process basically consist of exploring remaining design space with the constraints set by the platform. Constraints can be on components, as well as in communication.

So, if we are looking at an arm ways platform and if that uses amba bus, then amba bus can be a constraint. And you may look at the possibility of designing with an additional FPGA with the arm ways platform. You are exploring the design space in terms of designs that can be mapped on to that FPGA block.

And map next part comes in it map functionality of application on to platform instance. Mapping process includes hardware, software partitioning, which we have already seen. But, what is important again I am repeating the same point is that, your possibilities gets define by the constraints or the flexibilities that platform provide.

(Refer Slide Time: 39:33)



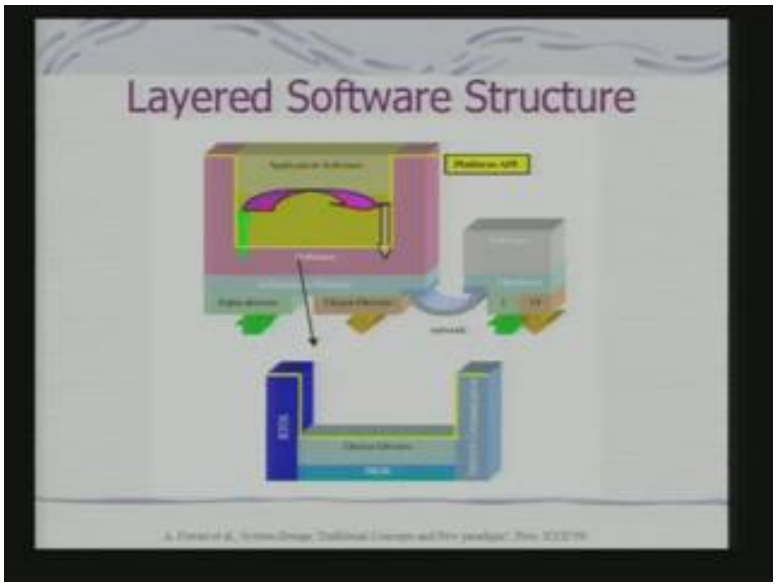
Next comes use of software platform. The architecture platform is abstracted at a level. So, that application software sees a high level interface to the hardware. So, what would be the basic conceptual framework. In fact, in a sense programming language is an abstraction of instruction set architecture.

See, if the even if the instruction set architecture changes, your application remains usable. If I am developing the application using any high level language. And API abstraction of multiplicity of computational resources. That is your concurrency model of RTOS and available peripherals the device drivers. That is abstracted through an application programmers interface or the layer.

So, that becomes the basic library of functions, which would be used by your application program. So, in that sense your application software should be re usable for every platform instance, this is very interesting to note. Because, you might be changing your platform instances. But, still if you are following these basic framework.

Then, your application remains reusable. See, you can optimize your application by considering different instances of the platform, without changing the software itself.

(Refer Slide Time: 41:00)



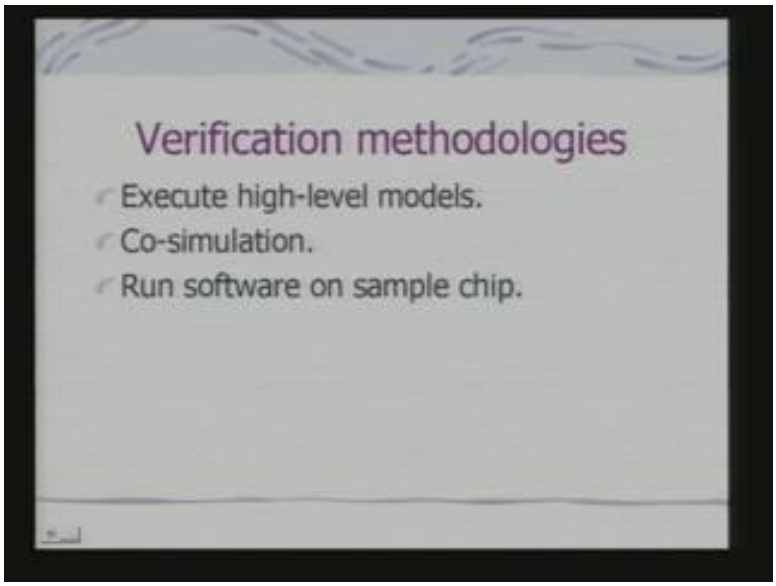
So, the layered software structure looks something like this. So, the top level you have got an application software, which is sitting on top of your platform API. So, you talk about a platform API, in fact TIS, OMAP provides a kind of a Linux spaced APIs are available.

And you have got the software. And this software layer of the platform would consist of device drivers. That is the abstraction over the I/Os, will consist of a active. That is the abstraction over the concurrency model, the model to supports threads, processors, etcetera. And if we are actually looking at a distributed system, it should also have the network communication support.

So, below that you have got your architectural platform, that is your hardware platform. And this hardware platform would be interfacing with the input and output devices. So, the whole application is on top of this platform. And you have the ability to modify the behavior by changing the software part of it, in terms of your application.

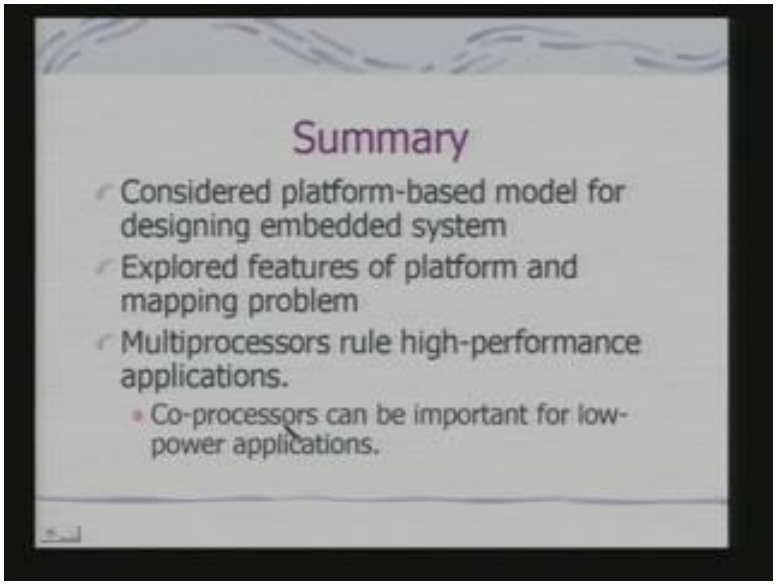
You can even have the ability to change the architecture platform, some aspects of hardware architecture platforms, through programmable or reconfigurable components. Even, you can change the behavior by mapping, when it is a multi processor, heterogeneous platform, mapping different parts of your application to different processors. So, these gives a layered software architecture for a platform based design.

(Refer Slide Time: 42:47)



And then, you do a verification execute high level models to co simulation, run software on a sample chip. And finally, check whether it is working or not co simulation means simulation of both hardware and software. In fact, all this platforms do provide you with the development environment, where you can do a co simulation to check your design.

(Refer Slide Time: 43:12)



So, what we have done today considered platform based model for designing embedded system. Explored features of platform and mapping problem. Multiprocessors rule high performance applications. And co processors can be important for low power

applications. In fact, we shall see more of these in next lecture. But, what is interesting to see that many of this platforms to support multiple processors in order to enable these kind of behavior.

So, a basic problem becomes that also, that of allocation. If you remember, you started today's class by considering the allocation problem. So, first such a platform based design, allocation and scheduling also becomes an important problem.