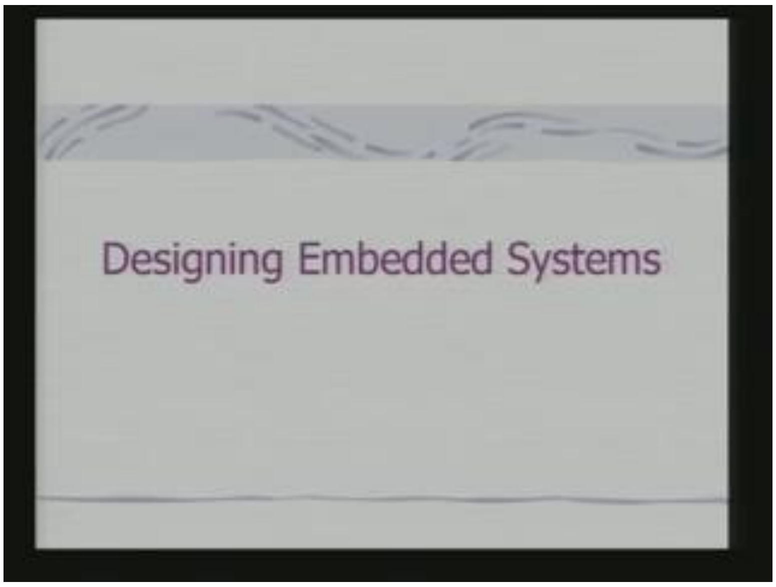
Embedded Systems

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

Lecture – 28
Designing Embedded Systems

Today, we shall look at the problem of designing embedded systems.

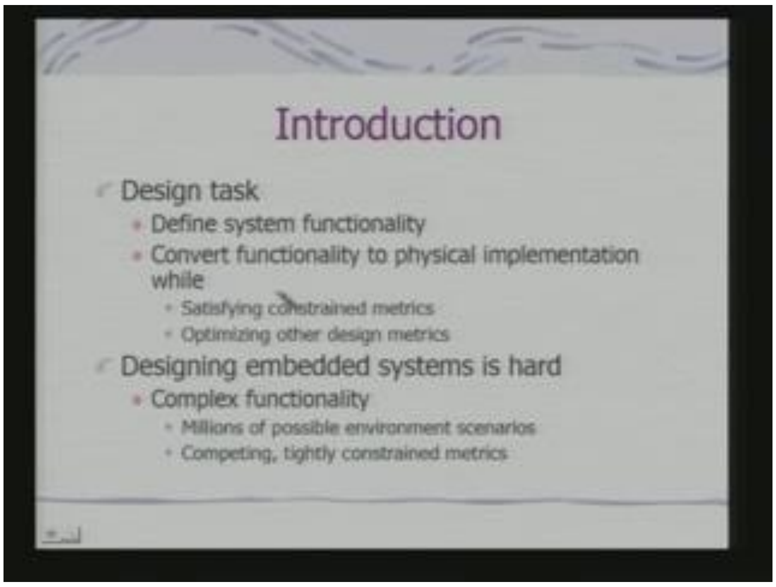
(Refer Slide Time: 01:01)



Designing Embedded Systems

In next few classes, we shall go through the different processors and different issues which are involved in designing embedded system.

(Refer Slide Time: 01:18)



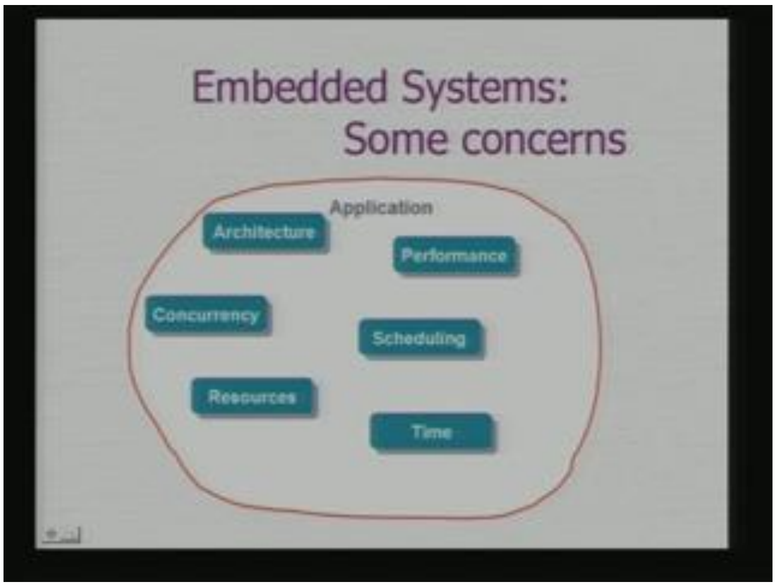
Introduction

- Design task
  - Define system functionality
  - Convert functionality to physical implementation while
    - Satisfying constrained metrics
    - Optimizing other design metrics
- Designing embedded systems is hard
  - Complex functionality
    - Millions of possible environment scenarios
    - Competing, tightly constrained metrics

The design tasks basically start by defining the system functionality. And the design process leads to translate the functionality to physical implementation satisfying the constraints which may be expressed in the form of variety of metrics in terms of performance, power consumption etcetera. And other design metrics optimizing other design metrics. So, you should understand the difference between the two aspects. One is when you specify the constraint a design is not acceptable until and unless the constraint gets satisfied. That means, if you say that the data acquisition should be done with in a time period of 40 millisecond then that is the constraint and that constraint has to be satisfied by the system.

Similar constraint can come from the requirements of power consumption. On the other hand there would be few parameters which you would like to optimize which may not have a constraint explicitly specified. And when I get a design for which this parameter better optimized; obviously, I shall prefer the design over another. Say for example, we may have two designs which satisfy the timing constraints. But for one design the energy requirement or energy consumption profile may be less compare to other then; obviously, we shall prefer the design for which energy consumption profile is more attractive. Obviously, designing embedded system turns of to be a complex problem simply, because it has got complex functionality and we have to look at this constraints as well as optimization issues.

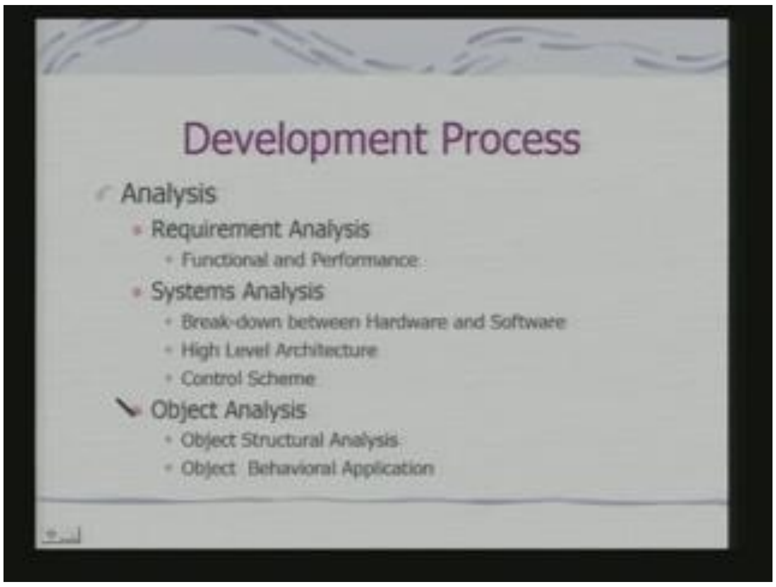
(Refer Slide Time: 03:35)



Embedded Systems: Some concerns

So, the concerns of the design are architecture and related to the architecture comes in the performance. Obviously, the architecture is also related to what is called concurrency what are the concurrent processes? Because for an embedded system they need for concurrency comes from concurrency of the physical phenomenon. So, how the concurrency has to be actually modeled and if they are to be modeled then depending up on the periodicity as well as the deadline associated we need to formulate proper scheduling policy. And based on the scheduling policy we shall be using variety of resources. Now, all this things are linked not in a general purpose sense they are linked via the application requirements. So, the determination of architecture its performance constraints concurrency model scheduling issues and availability of resources and use of resources all this things are basically determine by the application. In fact, these aspects of the design clearly distinguishes you are the problem of designing your embedded system from that of the general purpose computer. So, what will do the development process?

(Refer Slide Time: 05:06)

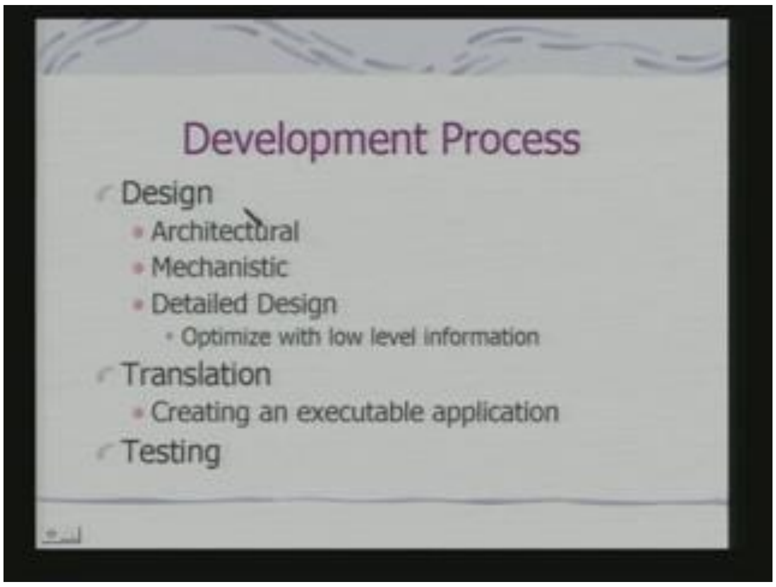


Development Process

- Analysis
  - Requirement Analysis
    - Functional and Performance
  - Systems Analysis
    - Break-down between Hardware and Software
    - High Level Architecture
    - Control Scheme
  - Object Analysis
    - Object Structural Analysis
    - Object Behavioral Application

Since we have found the application as the binding source it has to start with requirement analysis. So, the requirement analysis would covered 2 aspects both functional as well as performance related requirements. They should be followed by system analysis where we break down the requirements in to hardware components as well as the software components. And based on that decide about the high level architecture and the overall control scheme for the embedded systems. And followed by that you can have an object level analysis where your each components identified at the system level that is architecture level should translate to individual objects. These objects can be purely software objects or these objects can be physical objects and in physical objects we can have rape around software's as well. So, basically when you are referring to this objects what we are referring to as are entities which comprise our architecture. This entities may have realization in terms of hardware or in terms of software.

(Refer Slide Time: 06:30)



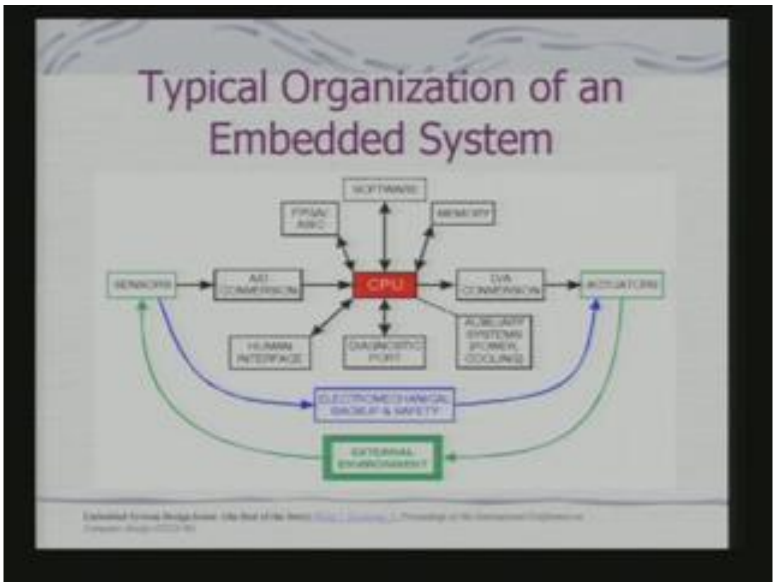
Development Process

- Design
  - Architectural
  - Mechanistic
  - Detailed Design
    - Optimize with low level information
- Translation
  - Creating an executable application
- Testing

So; obviously, this leads to architectural design, but architectural design is not the end of it, because it has to be packaged to deliver. So, mechanist design also places an important role, because how the product is to be packaged whether the packaging will take care of the cooling requirements or not? Whether the packaging take care of this space requirements for battery or not. So, all this mechanist design aspects becomes important. Also we are looking at embedding this device in to an appliance you have to look at the constraints that appliances imposes.

So, putting all this things together you have the detail design and where you can optimize the detail design by making use of low level information. And finally, once the designs aspects are ready design constraints are specified. And you have made the basic design decisions in terms of your architecture, in terms of your objects that actually will form the architecture that has to be translated to an implementation. And once you have implementation that has to be subjected to testing and validation to verify the fact that you have meeting the requirements with which use started. So, let us to understand this overall design process and the design challenges with reference to a conceptual model of the embedded system.

(Refer Slide Time: 08:14)

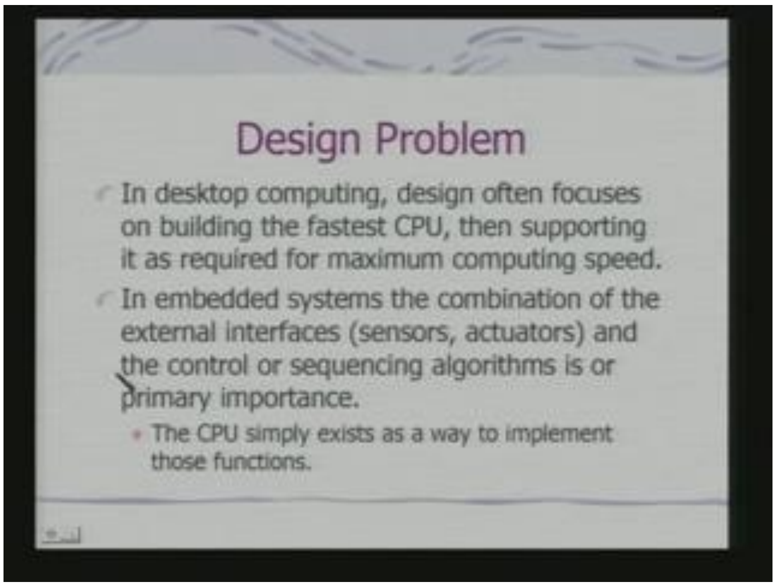


Typical Organization of an Embedded System

In fact, this model we had already looked at earlier I am just looking at it again to have references to various parameters which may turn out to be important. The key component is obviously, CPU along in the CPU you have the memory. So, these part of the CPU and the memory design is a very key component for the processing task. And your software has to satisfy the constraints of the system and be consistent with the CPU and the memory architecture that you have chosen. And you may require times to map some of the elements to an ASIC some of the processing tasks to an ASIC or an FPGA in order to meet may be timing requirements.

These are basically the interfacing issues or interfacing components with the external world to sends and to act. The actuators would modify the environment. This diagnostic port from the design point of view will provide you the facility of repairing as well as maintainability of the embedded system. And this electrical and mechanical backup and safety are important issues when really you have a device which controls may be mechanical components. So, everything is everything has to be considered when you are looking at a system level design of an embedded system. It is just not the issue of CPU design is just not an issue of ASIC design it is just not an issue of mapping a hardware on to an FPGA. These holistic view is what is very important for us to understand in order to design an embedded appliances.

(Refer Slide Time: 10:32)

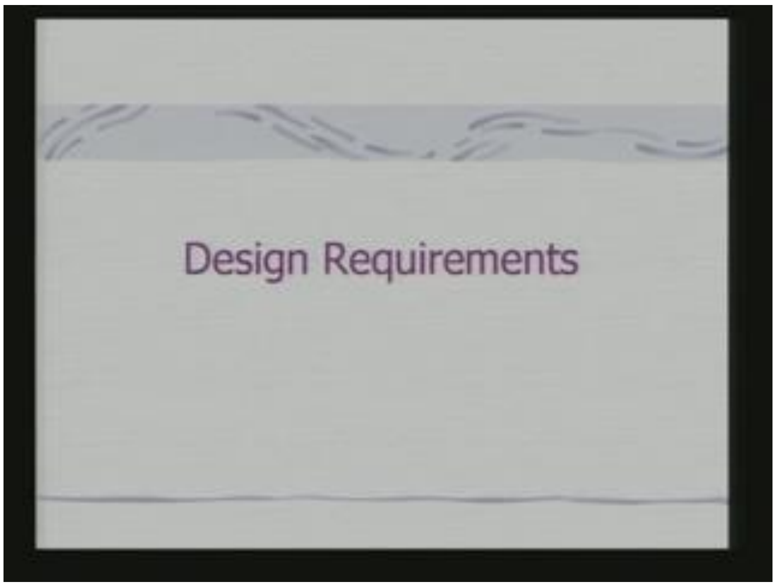


Design Problem

- In desktop computing, design often focuses on building the fastest CPU, then supporting it as required for maximum computing speed.
- In embedded systems the combination of the external interfaces (sensors, actuators) and the control or sequencing algorithms is or primary importance.
  - The CPU simply exists as a way to implement those functions.

So, what we say that in a desktop computing the design focuses on building the fastest CPU and then supporting it as required for maximum computing speed. So, we might like to design just a hardware circuit which may do the job fastest. But in embedded system the interfaces play a very crucial role. So, external interface the control are sequencing algorithms is of primary importance. And the CPU simply exists as a way to implement those functions. So, in that sense your CPU design although it is at the central point is that the central point and also focal them. But it is not the only thing this is the very important issue it is equal important to make choice of an appropriate sensors, make choice of a appropriate actuators. And look at the control loss and the processing tasks the signal processing tasks with the embedded system is suppose to execute. And in fact, your choice of CPU should be guided by these factors only. So, now, let us look at this design requirements of the design constraints.

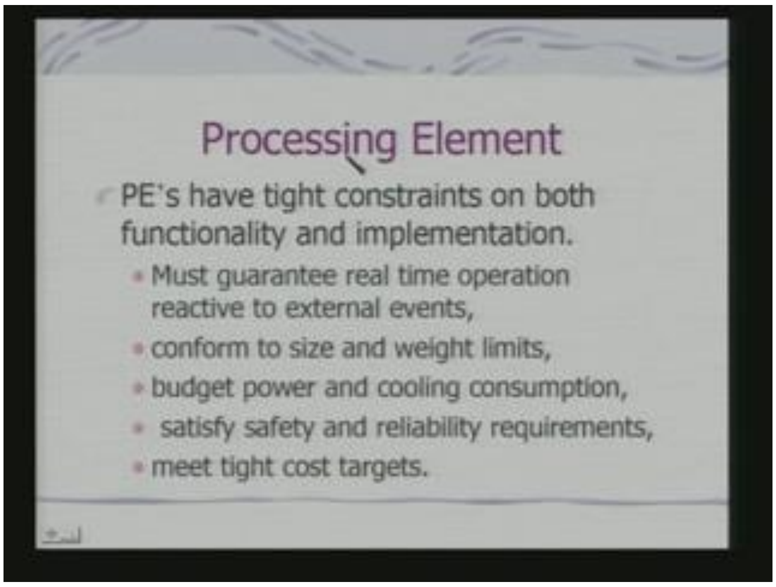
(Refer Slide Time: 11:56)



Design Requirements

These requirements not strictly requirement of an embedded appliances, but requirements that design should ideally satisfied. So, first let us look at the processing element which has said as the focal point.

(Refer Slide Time: 12:10)



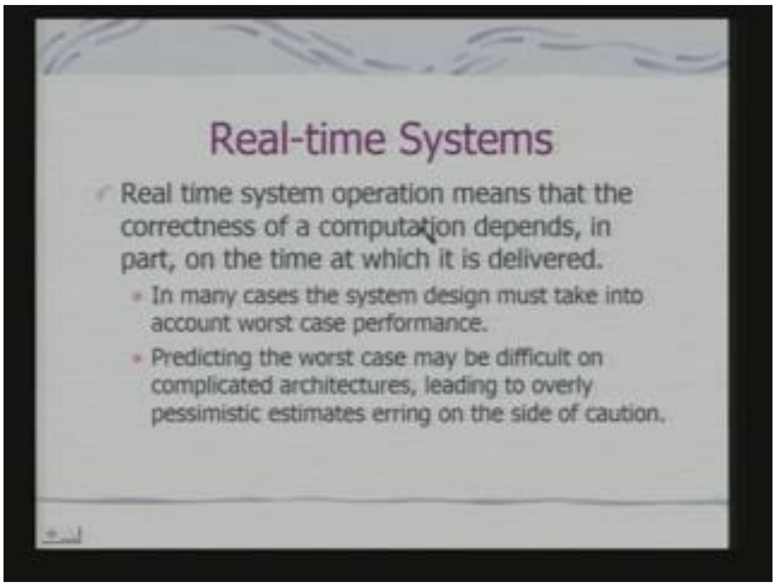
Processing Element

- PE's have tight constraints on both functionality and implementation.
  - Must guarantee real time operation reactive to external events,
  - conform to size and weight limits,
  - budget power and cooling consumption,
  - satisfy safety and reliability requirements,
  - meet tight cost targets.

So, this processing element of type constraints on both functionality as well as implementation. Because the processing element must guarantee real time operation reactive to external events. It should to conform the size and weight limit. It should have proper budgeting of power and cooling consumption. That is its cooling becomes also important issue should satisfy safety and reliability requirements and meet tight cost targets. So, choice of a processing element is therefore, not a straight forward task. In many cases he will not be designing a processing element, but you will be choosing a processing element. So, when you make a choice of the processing elements fundamentally if a real time operation. And if you meet to react to external events the processing element should have the enough computational power to meet this constraints.

But at the same time you’re other factors also important. It should not consume too much energy and should not be possibly a costly processor. So, may you may like to balance out this factors. So, that is why the issue of a designing additional logic on an FPGA or using a special purpose IC for doing some task becomes important in the context of embedded system. Because if you are choosing a faster processor to meet all your timing constraints you may be paying more for cost paying more for energy instead of that you may choose a low in processor and for a special functionality with respect to which you processor is not being able to meet the timing constraint use dedicated hardware. So, these balancing out at are the key issue for an embedded system design. The real time systems when we are designing there the key issues that of deadlines.

(Refer Slide Time: 14:28)

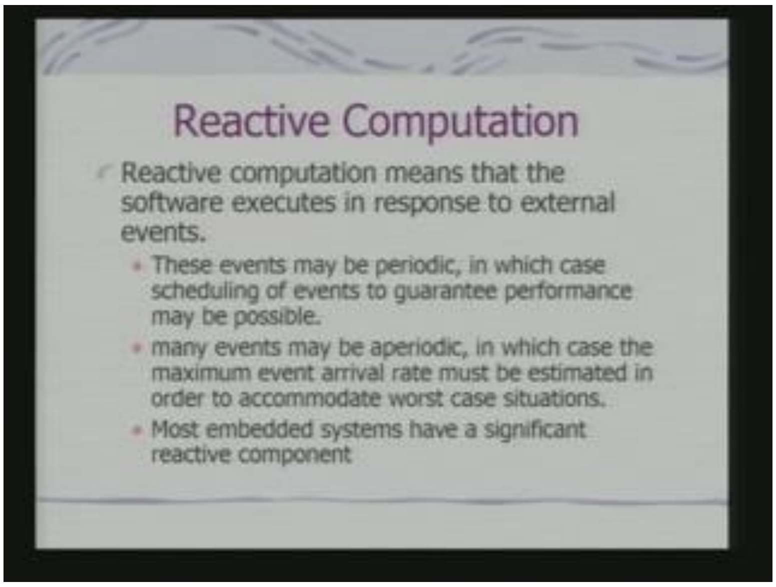


Real-time Systems

- Real time system operation means that the correctness of a computation depends, in part, on the time at which it is delivered.
  - In many cases the system design must take into account worst case performance.
  - Predicting the worst case may be difficult on complicated architectures, leading to overly pessimistic estimates erring on the side of caution.

So; obviously, the definition is that correctness of the computation depends in part of the time at which it is delivered. Now, how do you ensure that when you designing the system? So, system design must take in to account worst case performance. Typically the design should be ideally with reference to worst case performance. Predicting worst case may be difficult on complicated architectures and leading to overly pessimistic estimates erring on the side of caution. Now, if you are a very experience designer then you can minimize this error otherwise work is design may actually increase the cost. But the safest bed for a real time systems is to do a worst case design.

(Refer Slide Time: 15:26)



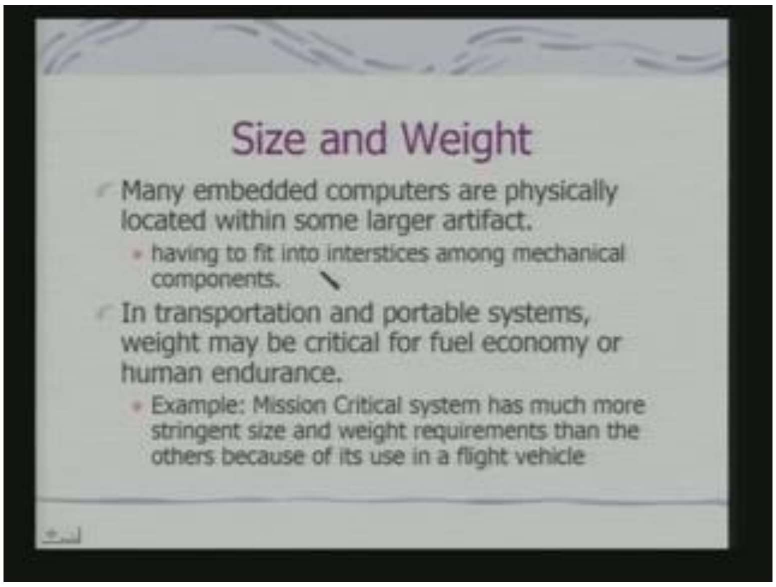
Reactive Computation

- Reactive computation means that the software executes in response to external events.
  - These events may be periodic, in which case scheduling of events to guarantee performance may be possible.
  - many events may be aperiodic, in which case the maximum event arrival rate must be estimated in order to accommodate worst case situations.
  - Most embedded systems have a significant reactive component

Reactive computation means that the software executes in response to external events. Now, this event may be periodic in which case scheduling of events to guarantee performance is possible we have already that scheduling policies. But events may be a periodic as well in which case the maximum event arrival rate must be estimated in order to accommodate worst case situations. So, this becomes an estimation problem you have to have a prior estimation of the worst case event arrival rate. Then only you can have provision for your time such that a periodic task constraints of a periodic tasks can be meeting.

And that may also lead to choice of an appropriate processor because you cannot choose any processor arbitrarily try to understand this. Because if you know the worst case arrival time for a periodic processor, as such that the time budget which will be available for a periodic processor. After servicing your periodic processors will not be sufficient to meets the demand of a periodic process. That essentially means that your processor taking more time for executing periodic tasks. Possibly you need to go for a faster processor. So, that you time budget for a periodic tasks can be increase taking in to a account the worst case arrival rate. So, if you are doing that what you are trying to ensure? You are trying to ensure that your system will not fail even under worst case computational load.

(Refer Slide Time: 17:16)

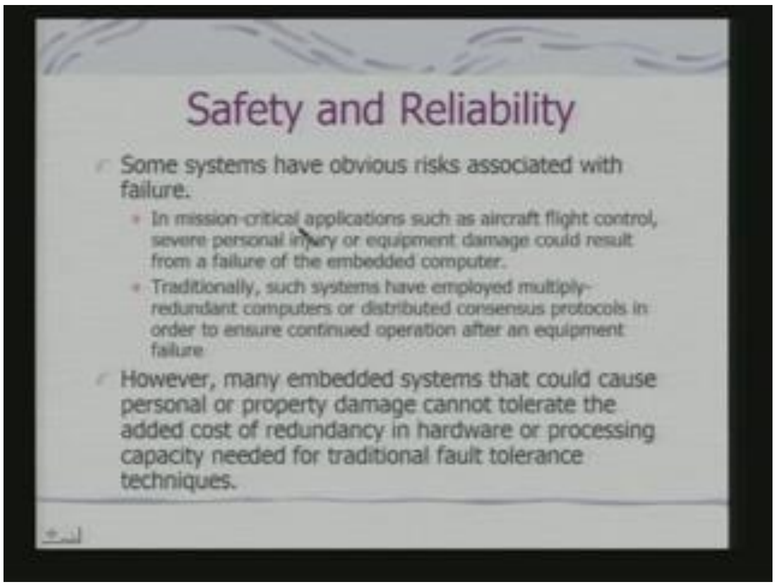


Size and Weight

- Many embedded computers are physically located within some larger artifact.
  - having to fit into interstices among mechanical components.
- In transportation and portable systems, weight may be critical for fuel economy or human endurance.
  - Example: Mission Critical system has much more stringent size and weight requirements than the others because of its use in a flight vehicle

Size and weight are also initial, because if I am using say a single board computer then what should be its weight what should be its size. So, many embedded computer physically located within larger artifact having to fit into what we called interstices among mechanical components. So, the sizes that mechanical components allow you have to meet that requirements. Therefore, it is not always the functional requirements, but it is a non functional physical requirements. So, you might me to design PCB putting together your digital circuits, analog circuits and power circuits in such a way that its food print is minimum. And so, that it can be put in to the place which your mechanical design permits. In transportation and portable systems weight may critical for fuel economy or human endurance. So, an example is a mission critical system has much more stringent size and weight requirements than the others, because of its use in a flight vehicle. Because if I am using a heavy computer weight is more for flyer system than the fuel consumption also increases. So, weight as well as a size they are although non functional requirements but play a major role for design of a embedded system. And they are not of relevance at all when I am designing a desktop computer or a super computer for that ((refer time; 18:43)) then become safety and reliability.

(Refer Slide Time: 18:47)

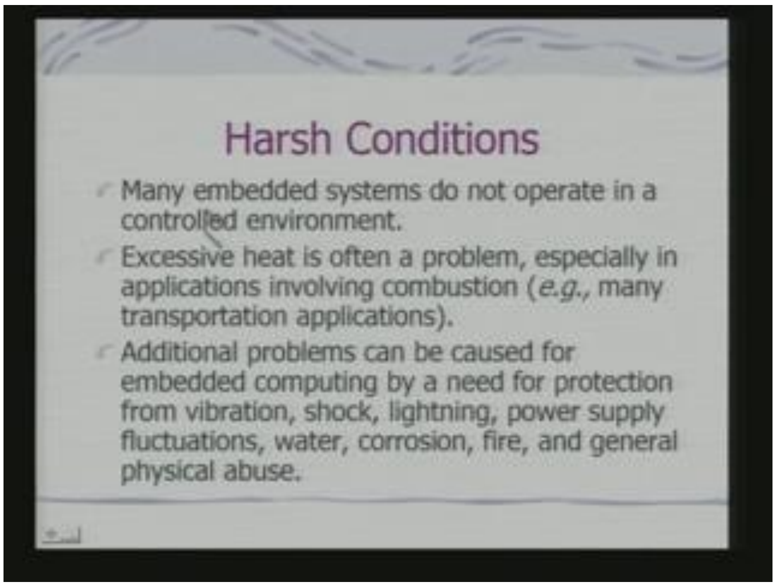


Safety and Reliability

- Some systems have obvious risks associated with failure.
  - In mission-critical applications such as aircraft flight control, severe personal injury or equipment damage could result from a failure of the embedded computer.
  - Traditionally, such systems have employed multiply-redundant computers or distributed consensus protocols in order to ensure continued operation after an equipment failure
- However, many embedded systems that could cause personal or property damage cannot tolerate the added cost of redundancy in hardware or processing capacity needed for traditional fault tolerance techniques.

Some systems have obvious risk associated with failure. So, for a example for a flight if there is a failure of the electronics then there is a huge loss of life and property . So, there has to be a decisions so, let us look at it design decision is for minimizing such risks . So, how do you do that? You can use redundancy typically you gives redundancy or distributed consensus protocols; that means, there are multiple computers located may be a different sights doing the same computations and then you take a consensus among to take a decision. Typically your space flight controls implement this kind of a redundancy with the consensus scheme. In fact, if there is a there is failure then may be a another computer taking over you the function of one of them are if there a 2 then one may fail another can continue and both of them can be synchronized with respect to the computation. But the whole problem is that the moment you bring in redundancy the cost will increase. So, this is an example of a trade of you can create a file save system, but file save system would be a costly system. So, you have to look at the business model or the cost model for deciding on whether you make the system truly file save or not.

(Refer Slide Time: 20:30)

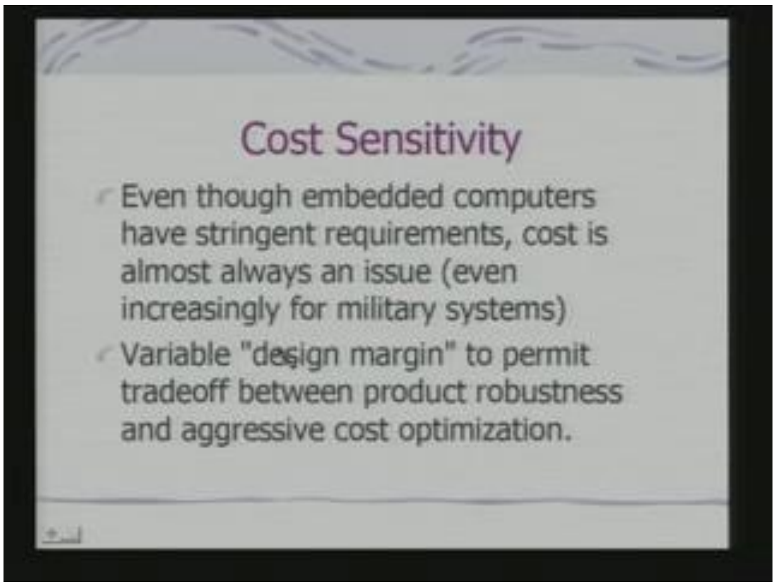


Harsh Conditions

- Many embedded systems do not operate in a controlled environment.
- Excessive heat is often a problem, especially in applications involving combustion (*e.g.*, many transportation applications).
- Additional problems can be caused for embedded computing by a need for protection from vibration, shock, lightning, power supply fluctuations, water, corrosion, fire, and general physical abuse.

And there are many embedded system which are expected to work under harsh conditions. So, excessive heats they can be vibrations, shock, lighting, power supply fluctuations, water, corrosion, fire and general physical abuse. So, how does this affects the design? This would affect the design in terms of choice of components you would like to make chose of components whose operating temperature range would be higher. There are various semi conductor components which are condition for operating over higher temperature ranges. So, the choice of component is one another very important thing is of packaging the components. Because your package should be made in such a way that internally components becomes shock resistance, vibration proof as well as may be water proof. If you look at the problem of yours sensor networks and your sensor nodes make it drop a various places in a forest to track animal it has to be with the resistance. It cannot be effected by your range can not be effected by really sunlight. So, there has to be mechanism that a packaging should be built in such a way that you make your embedded system operative under harsh conditions.

(Refer Slide Time: 22:02)

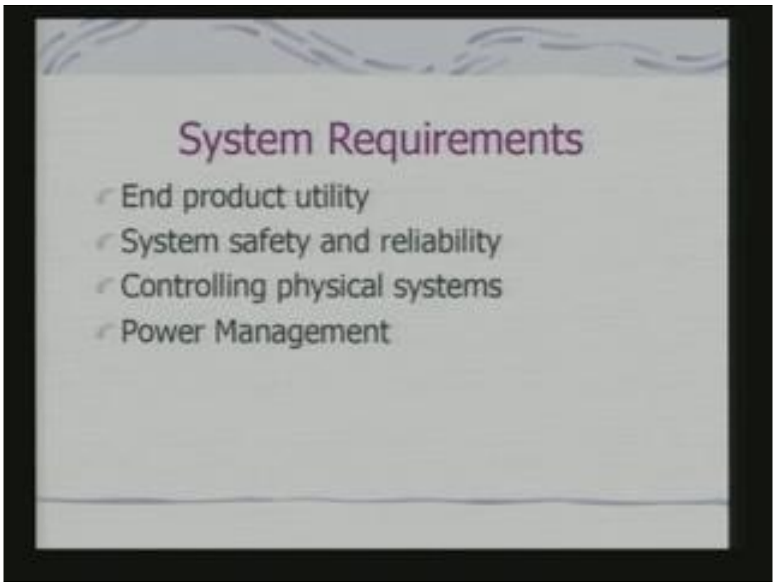


Cost Sensitivity

- Even though embedded computers have stringent requirements, cost is almost always an issue (even increasingly for military systems)
- Variable "design margin" to permit tradeoff between product robustness and aggressive cost optimization.

And these actually what leads to what it called a cost sensitivity? There are stringent requirements, but cost is almost always an issue, because I cannot built something which is highly reliable robust to meet all the requirements and at the same time highly costly. So, what we say? The variable design margin to you has to use to permit tradeoff between product robustness and aggressive cost optimization. If you look in to it this is this leads as to a very interesting issue the issue is what? That if you have the product you; obviously, you will not accept a product if does not satisfy the functional requirements. But satisfying functional requirements does not give you a guarantee about the quality of the product. A product can become robust if can taken care of all this conditions. The moment you have taking care of all this conditions the cost of the product can go up. So, the point is the tradeoff between product robustness and aggressive cost optimization. Somewhere these 2 thing have to meet depending on the perception of the market.

(Refer Slide Time: 23:17)

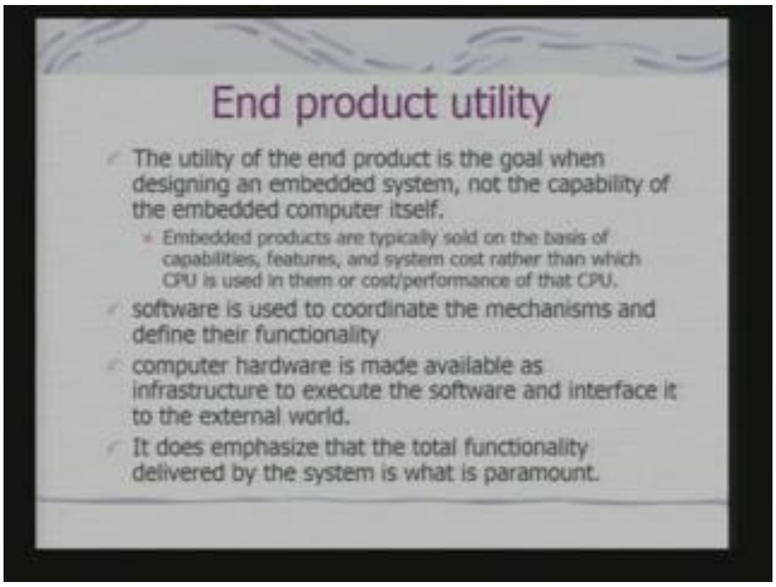


System Requirements

- End product utility
- System safety and reliability
- Controlling physical systems
- Power Management

So, over all from the system point of view. So, the key issue comes up end product utility system safety and reliability. And the aspects involved in controlling physical systems as well as power management, because so, far we are looking at primarily the issues related to processing elements. Now, we are extending these considerations for the complete systems as a whole.

(Refer Slide Time: 23:46)



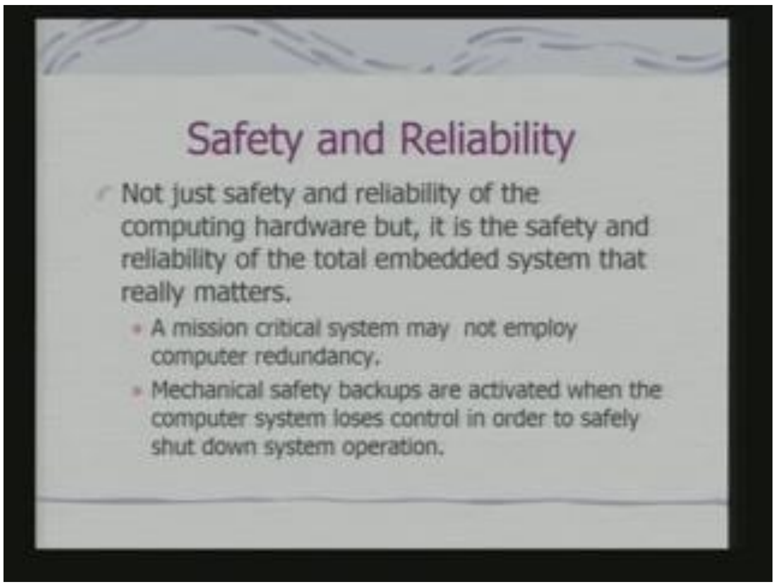
End product utility

- The utility of the end product is the goal when designing an embedded system, not the capability of the embedded computer itself.
  - Embedded products are typically sold on the basis of capabilities, features, and system cost rather than which CPU is used in them or cost/performance of that CPU.
- software is used to coordinate the mechanisms and define their functionality
- computer hardware is made available as infrastructure to execute the software and interface it to the external world.
- It does emphasize that the total functionality delivered by the system is what is paramount.

So, utility of the end product is the goal when designing an embedded system and not the capability of the embedded computer. This is something very important So, what we say that it really does not matter in fact, whether your car is using a 16 bit processor or a 8 bit processor the issue is whether the processor meets the demands of the car electronics or not. Using the 16 bit processor can be simply an advertisement for the company and not really in terms of utility of the product. So, these issues are important not only for designing, but also for appreciating and choosing the product and products specifications. So, software is used to coordinate the mechanisms and define the functionality obvious. And computer hardware is made available as infrastructure to execute the software and interface to the external world. In fact, why it is important? Because your functional requirements is translated through the software.

So, your software specifications will be such that it will actual inside the hardware, because your software and the timing constraints actually it is the hardware's. So, the utility comes from there itself. So, whether it is a 8 bit processor or a 16 bit processor depends on what is the functionality that has been supported in the software for the particular embedded appliances? And total functionality is; obviously, therefore, the paramount utility. So, when we are looking at the overall system and looking at the design of over all the system. So, what is important is that we are critically defining the functionalities. Functionalities are getting translated to specification of the software, constraints on the software. And that constraints would lead to actually the hardware platform on which the software has to be supported and that would dictate the choice of the processor. And on the basis of the choice of the processor will also have other non functional requirements which also become important your power budget, your cooling requirements, cost etcetera.

(Refer Slide Time: 26:08)



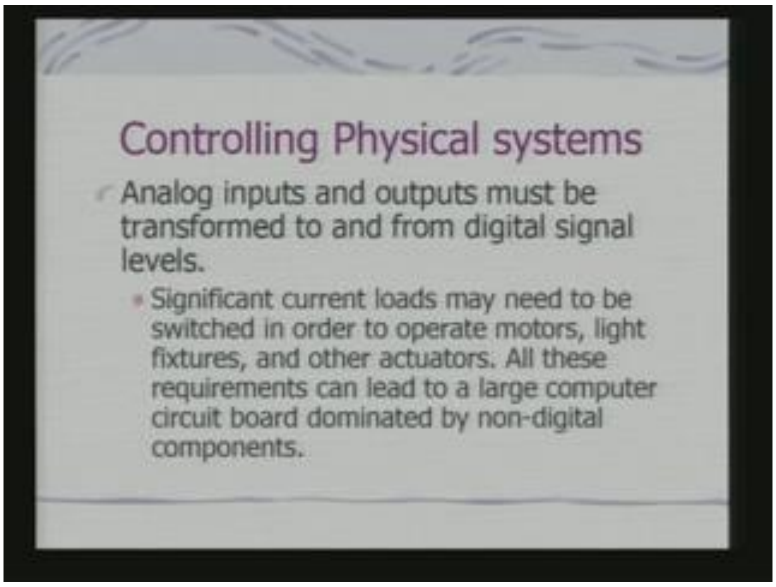
Safety and Reliability

- Not just safety and reliability of the computing hardware but, it is the safety and reliability of the total embedded system that really matters.
  - A mission critical system may not employ computer redundancy.
  - Mechanical safety backups are activated when the computer system loses control in order to safely shut down system operation.

Safety and reliability is not just the reliability of the processor we are talked about the reliability of the processor. So, one way of increasing the reliability of the processing element to redundant number of processing element. So, this may duplicate and use a duplication to make the system fault tolerant. But when you are looking as a system as a whole if you have to optimize the cost we may like to use other mechanisms to take care of the faults and the reliability. So, a mechanical safety backups are activated when the computer systems loses control in order to safety shutdown the system operation. So, basically you can consider a simple example that when all the controls may be operated by the computer.

So, it is a automated control. So, if the computer detects the failure if it can flag that failure. So, some part of the control can be taken over by the human operators you can have a manual control along with the partial computer control. So, that way your building in to the redundant system without increasing the cost. Typically in your car electronics in the maturity of the cases when it is an auto clear; that means, your fuel injection is automatically decided, your gear changes automatically decided the all of this calls also have the manual mode. So, manual mode can take over when there is a failure for this electronic part of the embedded part.

(Refer Slide Time: 27:58)



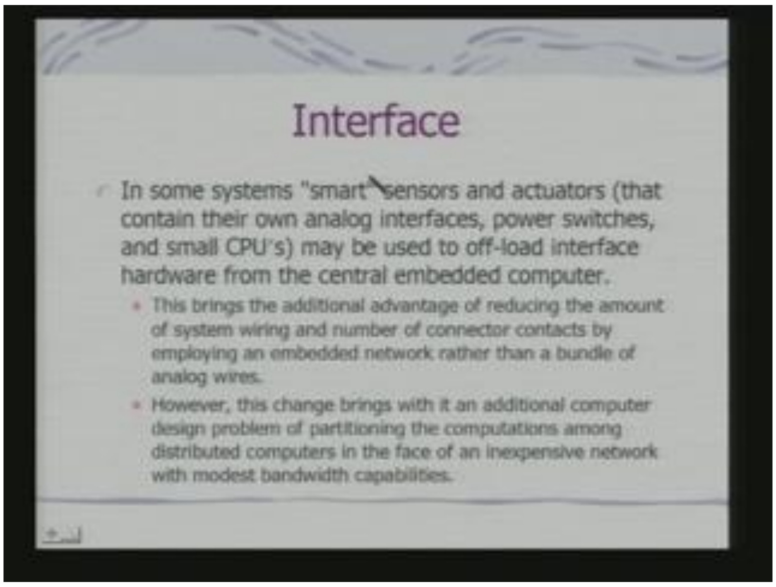
Controlling Physical systems

- Analog inputs and outputs must be transformed to and from digital signal levels.
  - Significant current loads may need to be switched in order to operate motors, light fixtures, and other actuators. All these requirements can lead to a large computer circuit board dominated by non-digital components.

And then if you are controlling the physical systems the analog inputs and outputs must be transformed to and from the digital signal levels. So, significant current load may need to be switched in order to operate motors light fixtures and other actuators. All this requirements can lead to a large computer circuit board dominated by non digital components. So, then designing and then putting this, because switching is also introduce what noise as well. So, there has to be mechanisms built in to take care of these issues, because you are actually controlling the actuators.

So, you may like to separate out this 2 electronics. Whether you put them in to a single box or a separate box plug it in how the system engineering would be done with respect to this requirements becomes an issue. Because if you look in to it there are 2 competing constraints you would like everything can be put on to a single PCB so, that your size comes minimum. But at the same time you moment a put everything on to the same PCB I may actually affect the reliability of the operations. Because noise and the heat generated in the processes in may be the non digital part of the circuit can affect the other part.

(Refer Slide Time: 29:29)

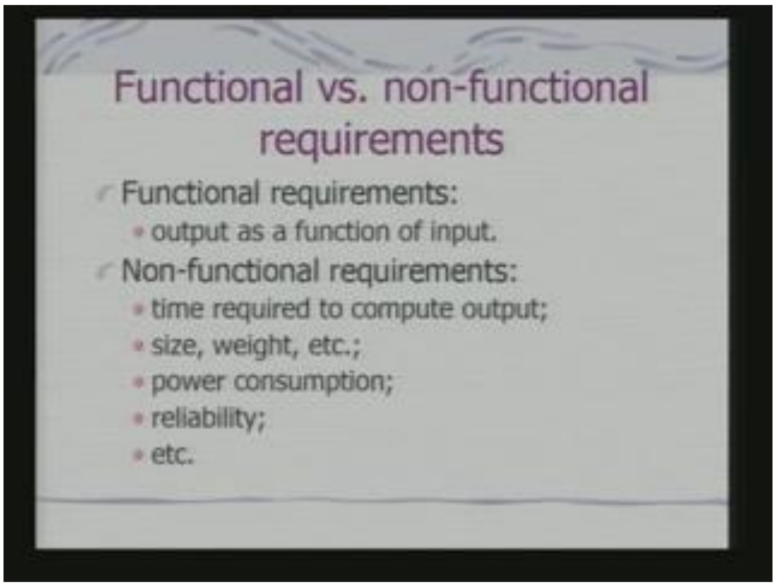

Interface

- In some systems "smart" sensors and actuators (that contain their own analog interfaces, power switches, and small CPU's) may be used to off-load interface hardware from the central embedded computer.
  - This brings the additional advantage of reducing the amount of system wiring and number of connector contacts by employing an embedded network rather than a bundle of analog wires.
  - However, this change brings with it an additional computer design problem of partitioning the computations among distributed computers in the face of an inexpensive network with modest bandwidth capabilities.

So, these issues have to be delete with and when you have the actuators say this is an example where you have got what we call say this not sensors and actuators where the sensors and actuators contain the own analog interface power switches and small CPU's that may be used to offload interface hardware from the central embedded computer. So, that means you are having the small system where your complete actuators electronics separated out your sensor electronics is separated out. And you are using a standard communication link may be a USB port to communicate between the sensor and the actual embedded computer. So, that is exactly the issue of packaging which comes up. And so, this says is brings as that additional advantages of reducing the amount of system varying. And number of connectors contact by employing an embedded network than rather than the bundle of analog wires.

So, the whole PCB design and packaging becomes much more streamlining and its more modular. So, if you imagine your concept of building up a system by putting together different modules. These strategy makes the systems are more modular what is also the advantage of this modular and design? Advantage is that if you have design this kind of smart sensors they become reusable; that means, you using is in product 1, but this sensors or actuators becomes reusable in product 2 as well. So, this is the basic design tradeoff which we have to keep in mind. Obviously, when you are using multiple modules your space requirement increases size requirement increase. But can you do an really engineering design such as size requirements becomes same almost same, but still you have got a modular components to play with.

(Refer Slide Time: 31:27)

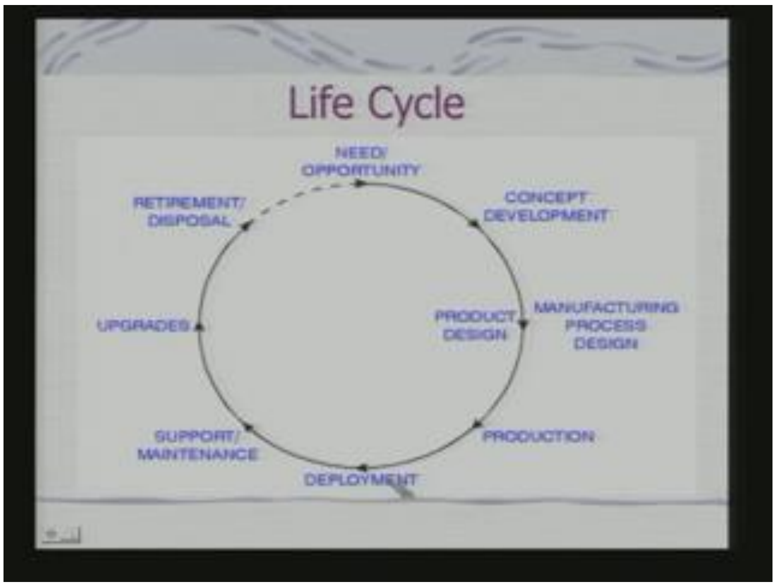


Functional vs. non-functional requirements

- Functional requirements:
  - output as a function of input.
- Non-functional requirements:
  - time required to compute output;
  - size, weight, etc.;
  - power consumption;
  - reliability;
  - etc.

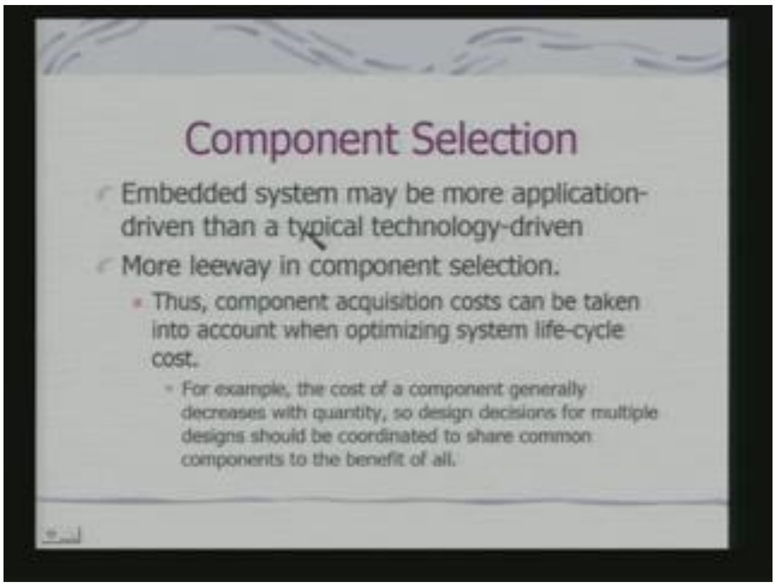
So, obviously, if we come to these a point what we have understood is that for an embedded system we have got functional as well as non functional requirements. Functional requirement is a output as a function of input, but when the design philosophy which we have look that that non functional requirements also play a very critical role. So, time require to compute output is the performance requirement then size weight extra power consumption reliability so, all this are non functional requirements. But these non functional requirements play a major and crucial role for designing an embedded system, because the function can be very easily implemented. So, the challenge is not that of just implementing a function, but that of implementing the function satisfying the constraints of non functional requirements as well as optimizing many of the non functional parameters. This is preciously the design challenge of an embedded system. So, what is the life cycle of development? So, this is a very standard product life cycle that we can look at.

(Refer Slide Time: 32:44)



So, development so, this takes place in a concurrent processes what we talk about a product design and manufacturing process production deployment support upgrades and return it. So, this is how it goes through a product life cycle. So, when you are doing the design there is a requirement that whether you have keeping in mind the different aspects of product life cycle as well or not. Try to thing in this term in a very simple way when you are doing a concept development for a product whether you would like to use a flash memory as part of the embedded system. Flash use a flash instead of a mast rom may lead to additional cost. But that would give you an opportunity to provide upgrades subsequently. So, you have to understand whether such an upgrade would be required or not what is the life time of the product? So, that motivates your decision about using whether a flash or that of around not just the functional requirements. So, product life cycle or conceptualization of a product life cycle also is an important aspect. So, that really leads to what we called a component selection.

(Refer Slide Time: 34:07)

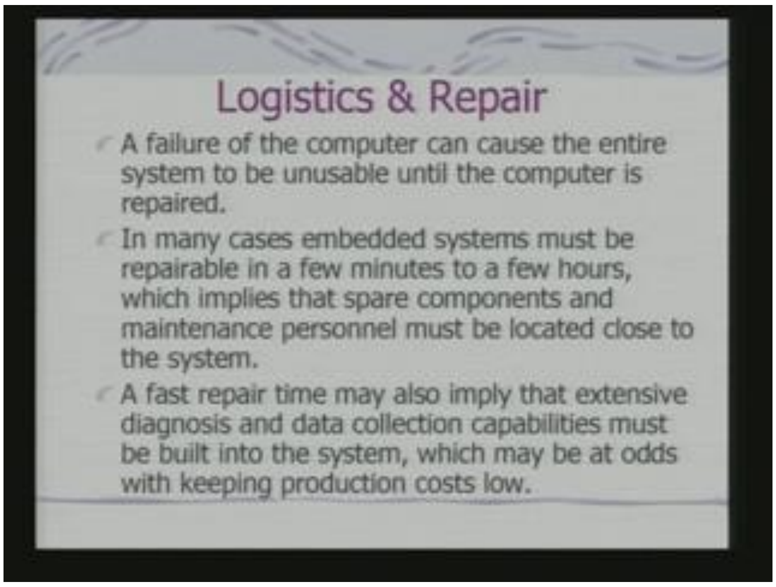


Component Selection

- Embedded system may be more application-driven than a typical technology-driven
- More leeway in component selection.
  - Thus, component acquisition costs can be taken into account when optimizing system life-cycle cost.
    - For example, the cost of a component generally decreases with quantity, so design decisions for multiple designs should be coordinated to share common components to the benefit of all.

So, embedded system may be more application driven than a typical technology driven. So, you have more leeway in component selection that component acquisition costs can be taken in to account when optimizing system life cycle cost. For example the cost of a component generally decreases with quantity. So, design decision for multiple designs should be coordinated share common components the benefit of all. That means, you can look at multiple products using similar components so, if the cost will go down and that would benefit process.

(Refer Slide Time: 34:39)



Logistics & Repair

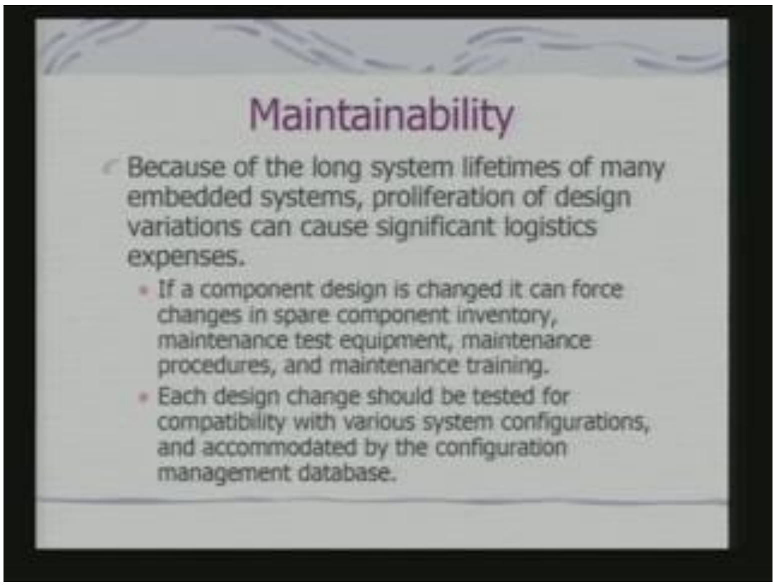
- A failure of the computer can cause the entire system to be unusable until the computer is repaired.
- In many cases embedded systems must be repairable in a few minutes to a few hours, which implies that spare components and maintenance personnel must be located close to the system.
- A fast repair time may also imply that extensive diagnosis and data collection capabilities must be built into the system, which may be at odds with keeping production costs low.

The other thing is that when you putting the product on to the market the issue of maintain the logistics and repair. So, if you want it to be repairable then you have to look at whether the space should be available, whether the man power to repair them should be available. Can we make a design which can be required if repaired by the user himself. Say for example, if you look in to a design of your cell phone the sim card if it goes bad it can be easily replace, but the user himself. You do not need a technical man power or technical set up to replace a sim card.

So; that means, the sim card has got what? Sim card has got some part of the functionality as well as the memory that your cell phone offers. So, you have seen that how it has been patrician? The actual processing elements and the memory part is in the cell phone which cannot really deal with, but there are other parts which is vendor or the service provide specific has been separated out. And separated out in such a packet which can be easily replaced and modified. So, that is basically a design aspect guided by the logistics requirement. Fast repair time may also imply that extensive diagnosis and data collections capabilities must be built into the system.

Now, these if it is built in you can have a technical manpower to repair and designs such a system. There may be on the other hand very low cost systems where you would not like any repair any takes place you would like it to be used and thorn away. So, then they are those systems may not have any support for diagnostic port. If you remember the organization that I showed you one important aspect was diagnostic port if you have design the system which is to be used and thrown away then diagnostic port really has no role. So, that electronics can you straight away eliminating saving cost as well as space.

(Refer Slide Time: 36:50)

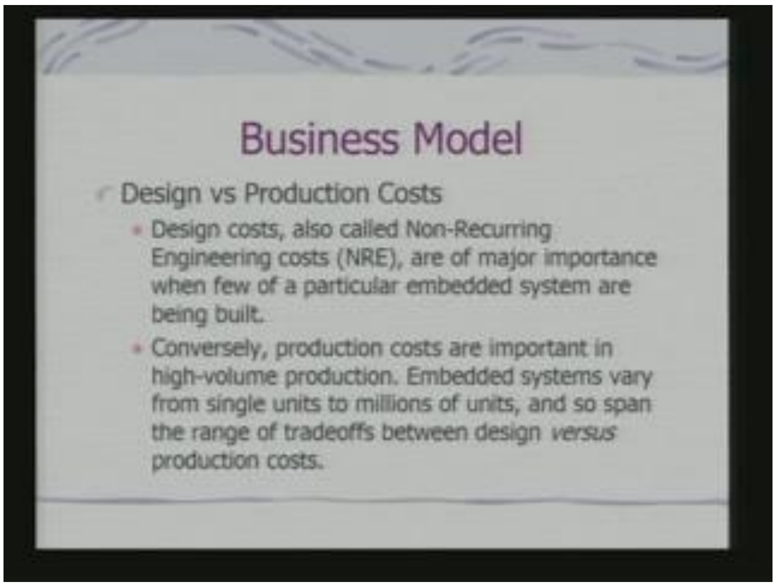


Maintainability

- Because of the long system lifetimes of many embedded systems, proliferation of design variations can cause significant logistics expenses.
  - If a component design is changed it can force changes in spare component inventory, maintenance test equipment, maintenance procedures, and maintenance training.
  - Each design change should be tested for compatibility with various system configurations, and accommodated by the configuration management database.

So, maintainability is if we are looking at long systems life time proliferations of design variations can cause significant logistics expenses. That is if a component design is changed it can force changes in spare component inventory, maintenance test equipments, maintenance procedures and maintenance training. And each design change to be tested for compatibility with various other system configuration and accommodated by the configuration management database typically you should have. So, let us look at a car. Now, as a different models of the car are coming in your electronics also changes. But change of the electronics cannot be searched that it makes the older models in compactable. Because if there is a failure other you have to support the old space the old electronics or else you have to make the current electronics compactable with the old model. So, that they can be plugged in if it is required. So, whenever there is an improvement of the modifications this maintainability of the product becomes an important issue.

(Refer Slide Time: 38:06)

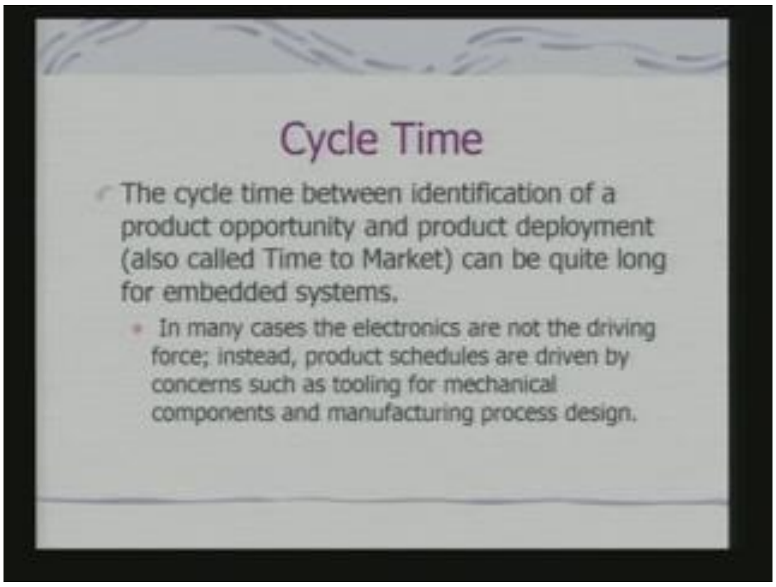


Business Model

- Design vs Production Costs
  - Design costs, also called Non-Recurring Engineering costs (NRE), are of major importance when few of a particular embedded system are being built.
  - Conversely, production costs are important in high-volume production. Embedded systems vary from single units to millions of units, and so span the range of tradeoffs between design *versus* production costs.

So, obviously, the life cycle is related to the business model. So, you have got design as well as the production costs, design cost is non recurring cost production cost is actual production cost. And this point is that this non design cost that of major importance when few of a particular embedded system are being built when it is a very special purpose when it is a must market. And so, a production cost are important in high volume production, because the design cost can be very easily off set over the number of products that have been meant. So, the production of the individual unit becomes a critical component. It is bill of material then it becomes important as well as the cost of assembling together. So, this NRE cost and the production cost at the 2 determining factors. In fact, when we say that we are trying to talk about choosing a processor, choosing a processor which is meeting all constraints, but becoming costly what it means? It means that your deal of material for the embedded system is increasing that gets added in to a production cost. If are using 1 or 2 versions or 1 or 2 specialize embedded systems their increase of a cost of a individual processor may not play a very critical role.

(Refer Slide Time: 39:41)

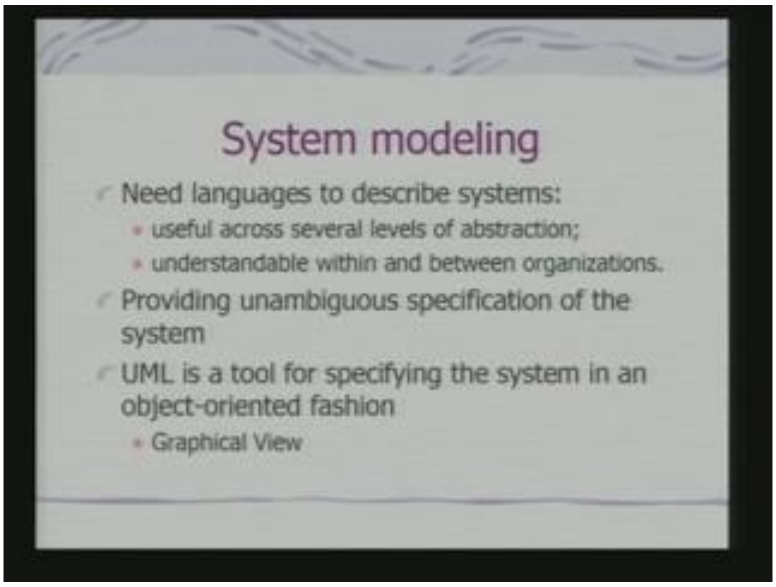


Cycle Time

- The cycle time between identification of a product opportunity and product deployment (also called Time to Market) can be quite long for embedded systems.
  - In many cases the electronics are not the driving force; instead, product schedules are driven by concerns such as tooling for mechanical components and manufacturing process design.

And the cycle time is the time between identification of a product opportunity and product deployment. And this is what is known as time to market and can be quit long for embedded systems. But ideally today we would not like it to be long if your long if you taking it long then will be actually out of the market. So, but the key issue here is this more of a implementation aspect that electronic design must be most of the cases straight forward not it is straight forward its durable. But mechanical components can manufacturing process design consumes the maximum time. Now, what it comes to therefore, is that you cannot really have a complete conceptualization development and production of an embedded system without ignoring the mechanical aspect of the event. This is a very key observation that you should make after the discussions. Now, with this background of design requirements and the design needs we shall now, start looking at actual design process.

(Refer Slide Time: 40:50)

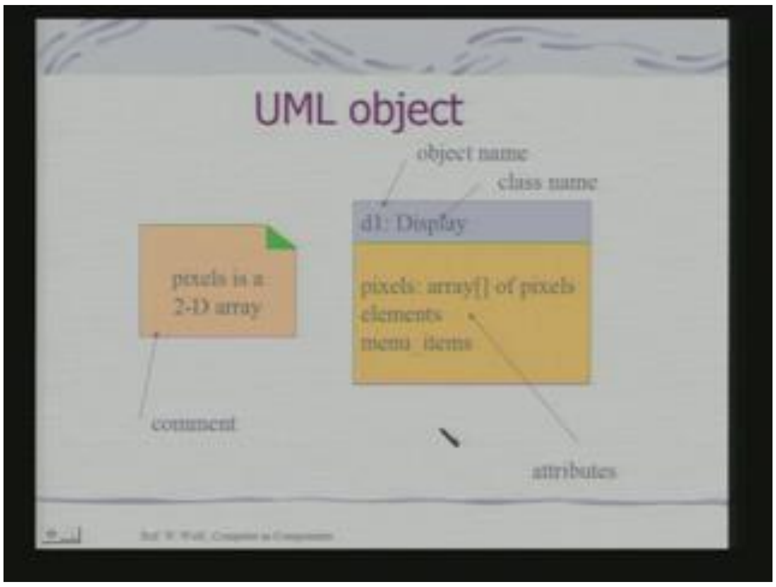


System modeling

- Need languages to describe systems:
  - useful across several levels of abstraction;
  - understandable within and between organizations.
- Providing unambiguous specification of the system
- UML is a tool for specifying the system in an object-oriented fashion
  - Graphical View

So, for what we have discussed? You have discussed the steps involved design we have discussed what kind of requirements to be satisfied by the design. Now, we shall start looking at the design process. The first step in the design process is that of the ability to express the design if you cannot express the design you can do the design. So, you need what is called design languages? So, you use design languages to model your systems . So, if you are using this kind of design languages the useful across several level of abstraction and understandable within and between organizations. And another very important advantage of design languages is they provide unambiguous specification of the system, because you get the ambiguity in the design then you can not translate the design in a executable component. So, there are many such modeling tools and languages which have been used today, for embedded systems we shall first look at UML. UML is a tool for specifying the system in an object oriented fashion and it provides primarily a graphical view of the system.

(Refer Slide Time: 42:15)

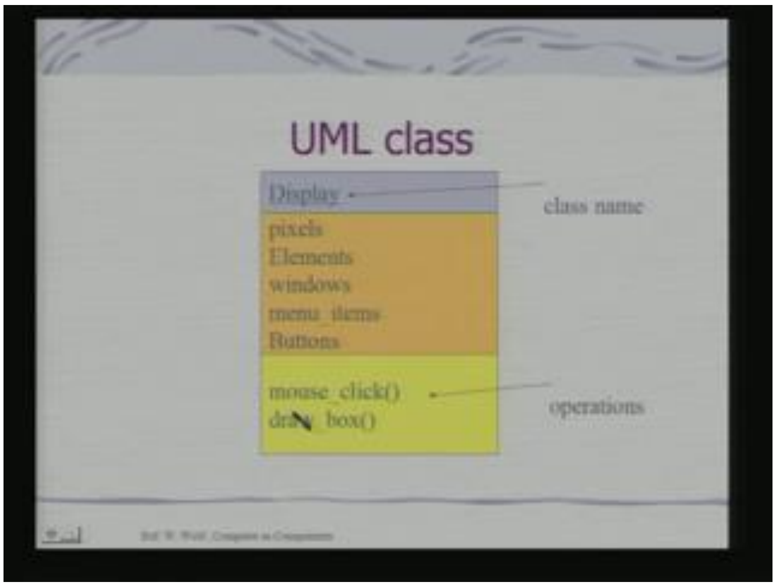


So, this is a typical representation of a UML object. Now, please keep in mind that UML was primarily developed for designing software intensive systems. And in UML the objects corresponded to the physical objects which can have a software implementation or realization. But since the objects can be map to a physical object then when you are talking about an embedded system design, there is really no conceptual difference between a hardware object or software object. You can actually refer to hardware components that object, because all such objects will have a hardware realization along with a software rapper around it. Take an simple example of this UML object which is a display. Now, if you look at a display display has to be a physical display a physical LCD display which is an array of 2 D pixels.

So, an 2 D array effectively. And array 2 D array means it will have the memory locations you have to write data on to the memory locations and so, logical representation of pixels is a array of pixels and what are the different elements which can be displayed on the pixel menu items? So, this is what? This is a basically fields of the object and this is a display and this is the specific instance of the display that is why we have referring it to by a name D 1. So, the basic concept is that I can group all kind of set displays in to an class called displayed . So, all this displays will have the support for this array of pixels that will map to the memory that is actually realized in hardware. Your software when it is writing anything on to a display is writing on to that memory.

So, that writing is whatever you do will be done through the methods which actually would form part of this class itself. So, these class is actually representing the class corresponding to the object is actually representing what? The display as a physical element along with the software which is wrapped around it. So, that you can use that hardware element in your design. This is the basic motivation of using UML as a design tool for embedded system. So, this is what we are showing object name d 1, because it is a instance of the class called display. Display is the class name and this are the attributes of that class.

(Refer Slide Time: 45:07)



UML class

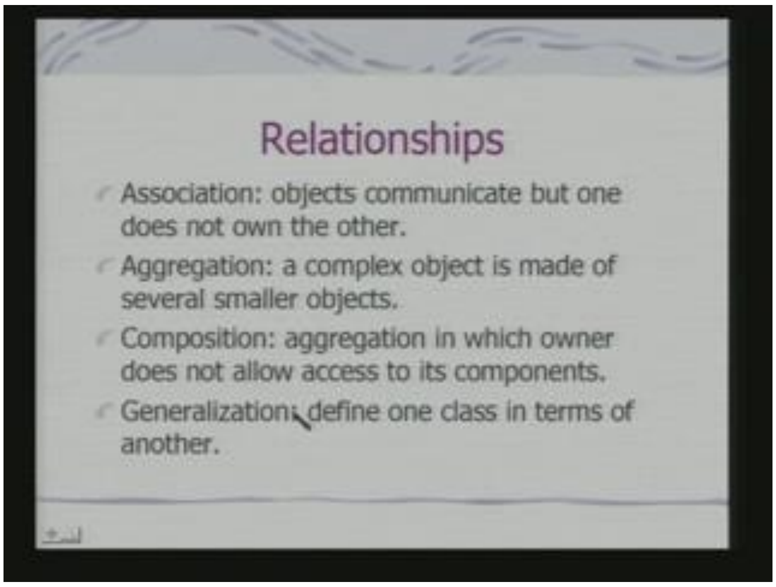
Display

pixels
Elements
windows
menu_items
Buttons

mouse_click()
draw_box()

So, if you go to the complete class the class would be represented these. This is basically the methods so, mouse click or draw box. Now, these would translate to what how actually the data has to be written on to memory? So, that these patterns can be displayed or this mouse click can be recorded. So, you are again looking at the basic concept of modularization we are modularizing this components both hardware and software modularized in to a class and class instances. So, you have got the complete encapsulation. Encapsulation and hiding of details not only of the software, but as well of hardware features. Because you may have a particular display at a display instance can be an LCD display instance another display can be a CRT display. The actual scheme being used for writing the data on to them may be different. But that details is being hidden to a user of the display class. So, these are basically the things.

(Refer Slide Time: 46:16)

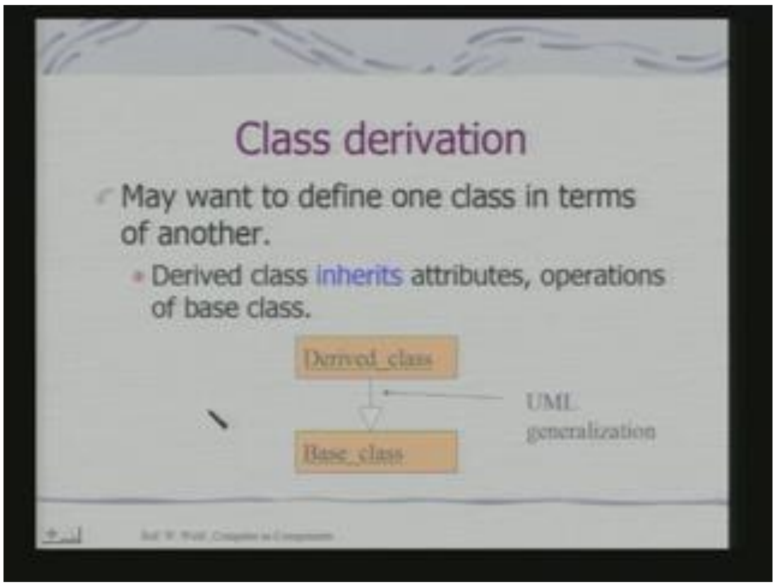


Relationships

- Association: objects communicate but one does not own the other.
- Aggregation: a complex object is made of several smaller objects.
- Composition: aggregation in which owner does not allow access to its components.
- Generalization: define one class in terms of another.

Now, there would be relationships between the objects association, aggregation composition and generalization. Association means objects communicate, but one does not own the other . Aggregation means it is a complex object which is made of several smaller objects. Composition is aggregation is which owner does not allow access to its components. And generalization defines 1 class in terms of the other. Please try to keep in mind these relationships, because they will be used actually for describing variety of embedded architecture.

(Refer Slide Time: 46:56)

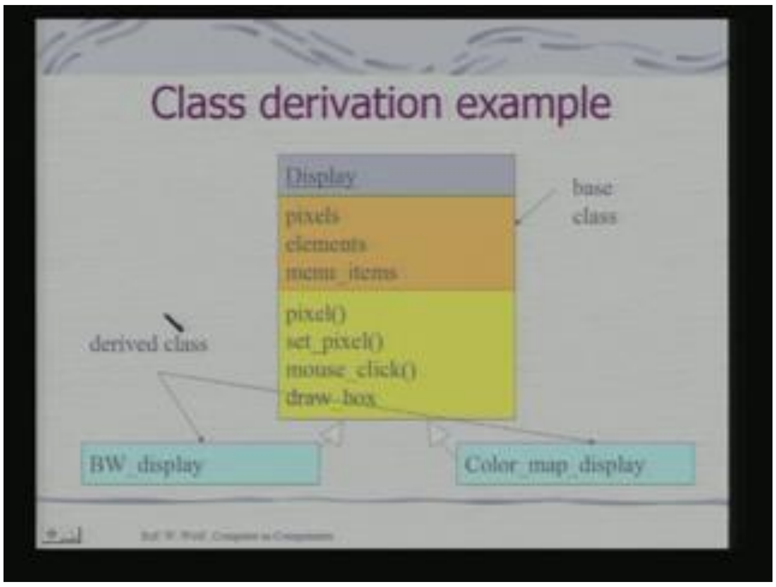


Class derivation

- May want to define one class in terms of another.
  - Derived class inherits attributes, operations of base class.

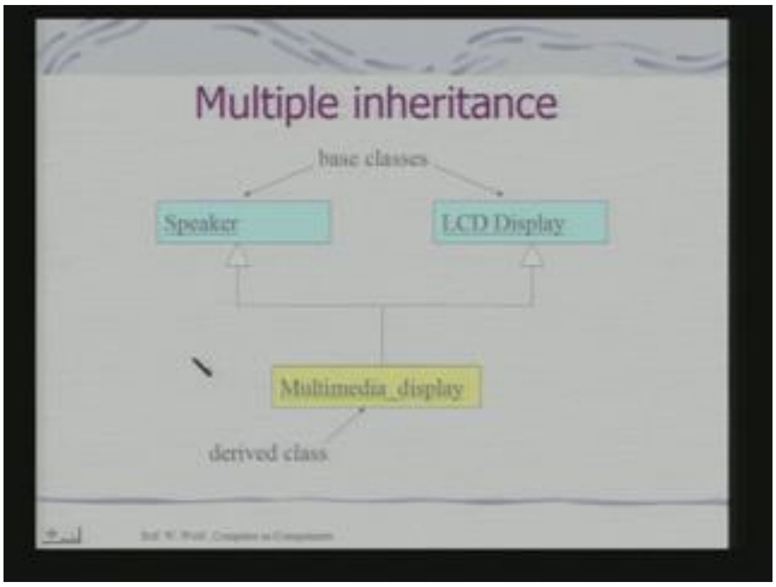
So, this is an example of an derived class simple class inheritance. So, you have a base class and you have got a derived class. Derived class inherits attributes operations of base class. So, this way you can built on the specializations.

(Refer Slide Time: 47:14)



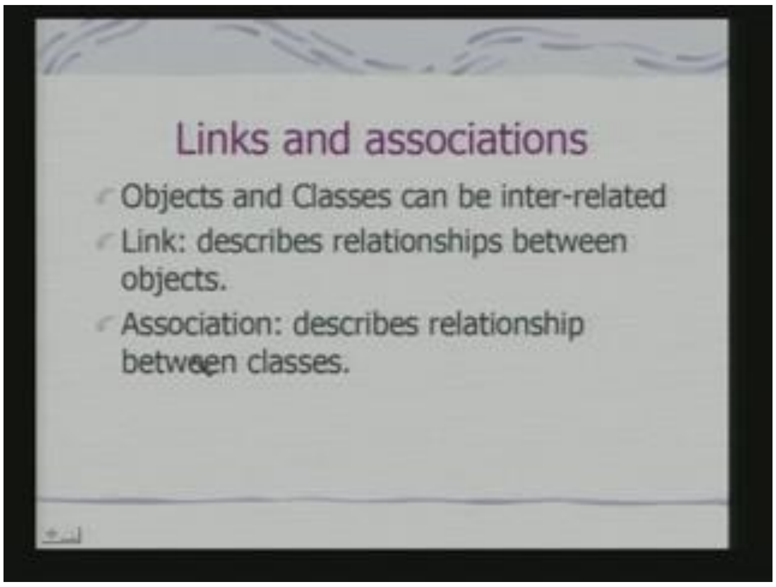
So, if you look in to here I had a display. This was the base class and on the basis of which I can have a specialize black and white display I can have a color map display. This basic functionality is can be defined here and then I can inherit and do what in this case? Overwrite may be some of this functions add on attributes. So, this is the base class and you have the basic derived classes black and white display and the color map display.

(Refer Slide Time: 47:48)



You can have multiple inheritance you can have a LCD display and the speaker both inherited together to form your multimedia display.

(Refer Slide Time: 48:03)

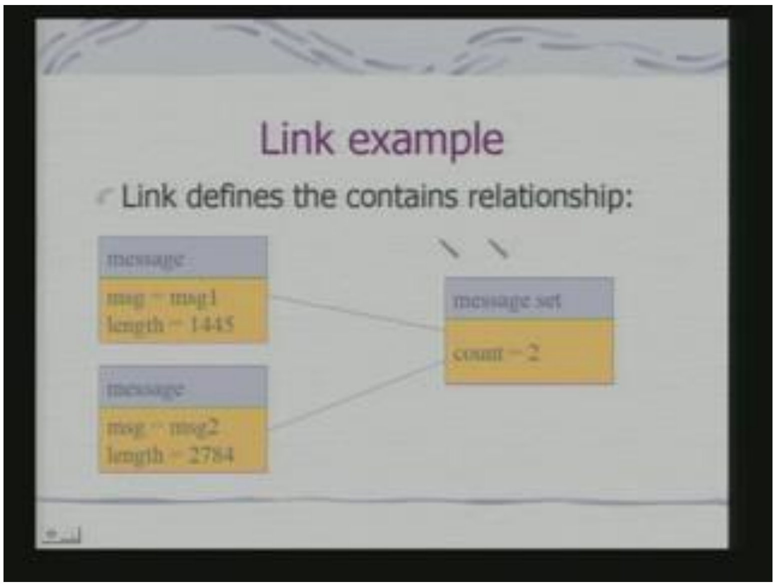


Links and associations

- Objects and Classes can be inter-related
- Link: describes relationships between objects.
- Association: describes relationship between classes.

So, links and associations actually show how objects and classes are inter related? Link describes relationships between objects and association describes relationship between classes. These are the 2 different things objects are what instance of the classes. So, when we talk about the link we talk about relationship between instances when we talk about association we talk about relationship between the classes themselves.

(Refer Slide Time: 48:34)

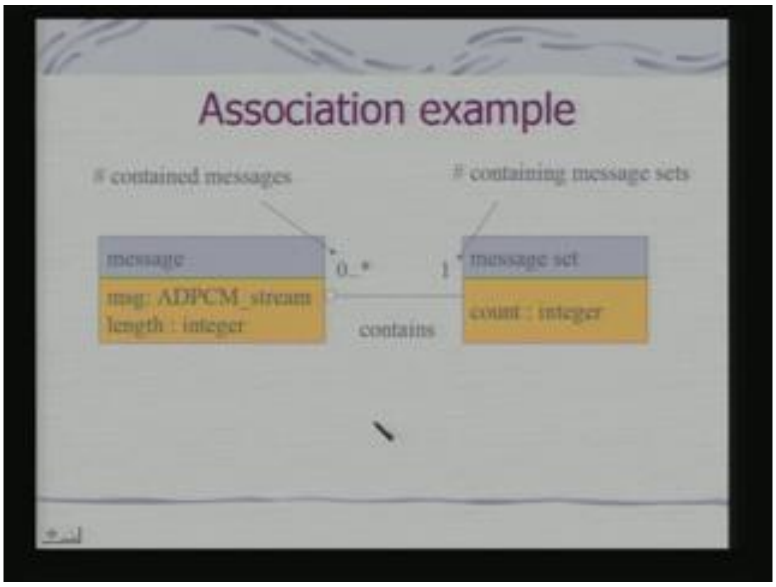


Link example

- Link defines the contains relationship:

So, if we look it to here you are talking about an object called message set which consisting of 2 messages. And these are the 2 instances of the messages the 2 messages of 2 different message length.

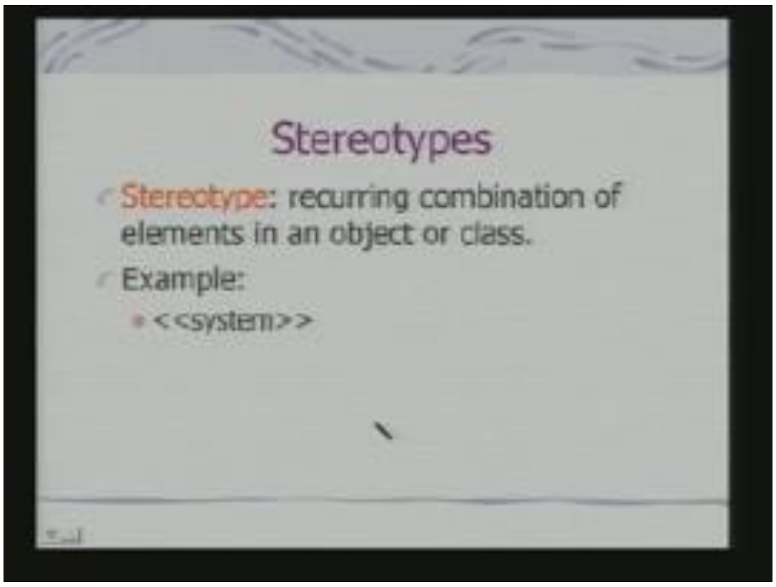
(Refer Slide Time: 48:48)



Association example

And this is an example of an association and what does that mean? The message set is a class message is another class and the message set can contain 1 message set can contain zero to many messages for an instances. So, zero star means actually 1 too many. So, this contains many such message streams. So, this is basically containing the message sets.

(Refer Slide Time: 49:27)

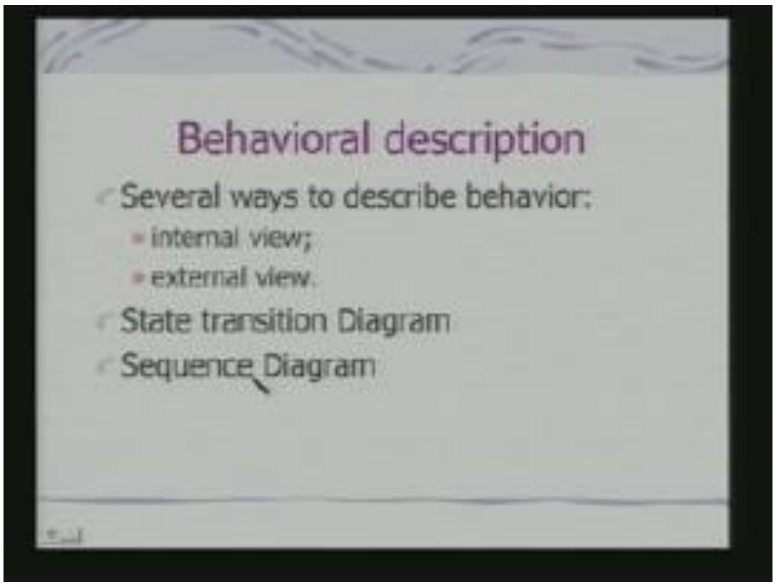


Stereotypes

- Stereotype: recurring combination of elements in an object or class.
- Example:
  - <<system>>

Now, there is also concept of what is called stereotypes. So, if you have a particular kind of a class and so, if you have class and you want to indicate an object that it belongs to a particular stereotype class you would indicate by this symbol. So, an object belong in to a system class can be indicated by this symbol.

(Refer Slide Time: 49:51)

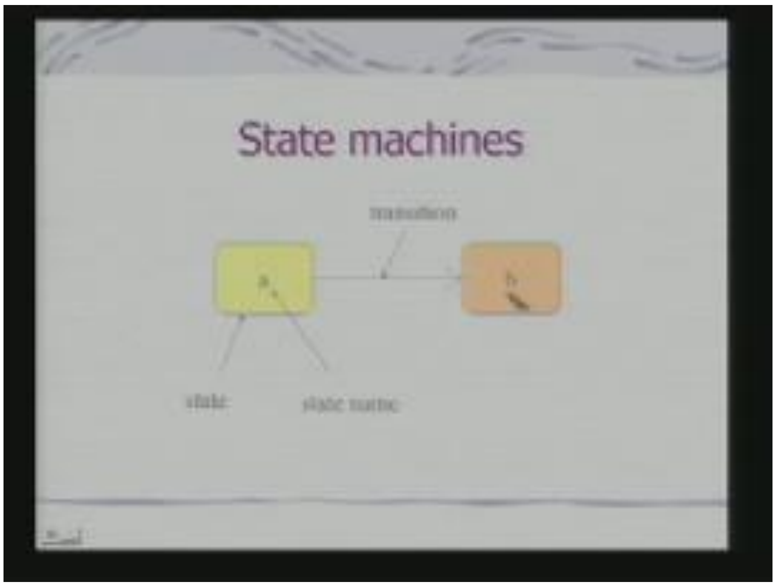


Behavioral description

- Several ways to describe behavior:
  - internal view;
  - external view.
- State transition Diagram
- Sequence Diagram

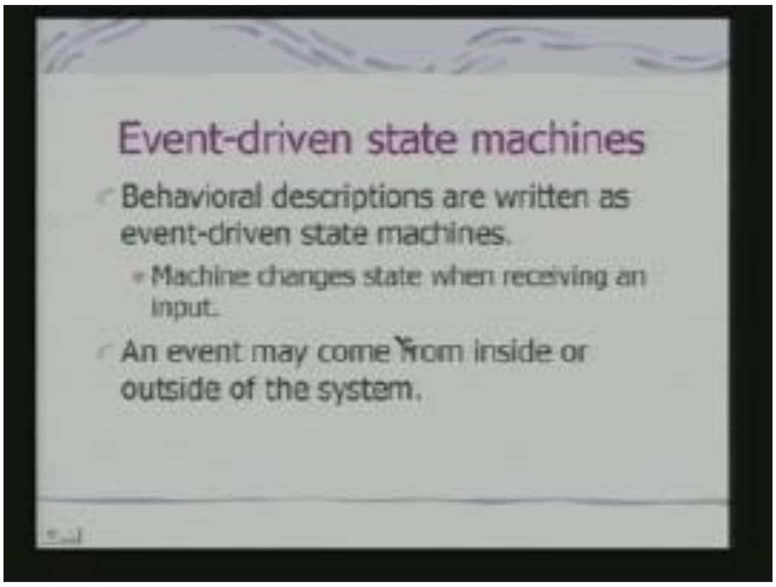
Next. So, what we have got so, for? We have got therefore, a mechanisms to specify the individual components of a design, because your design I said consisting of objects. Because any embedded system design will be built around components. Each component what I am telling is getting encapsulated into an class and an class instance. So, we have got a language for basically describing the components which goes in to building and a embedded system. So, primarily that is a static part of the embedded system. The other part would be the dynamics. So, dynamic is captured by behavioral description. So, primarily the 2 basic 2 tools which are used a state transition diagram and a sequence diagram.

(Refer Slide Time: 50:51)



So, straight machines it is nothing but your finite state machines which we have used for any digital circuit design. So, here you have got a state. A state will have a unique state name and there would be a transitions from a state to another state. And that is described by these kind of a diagram and that specifies behavior of the system.

(Refer Slide Time: 51:14)

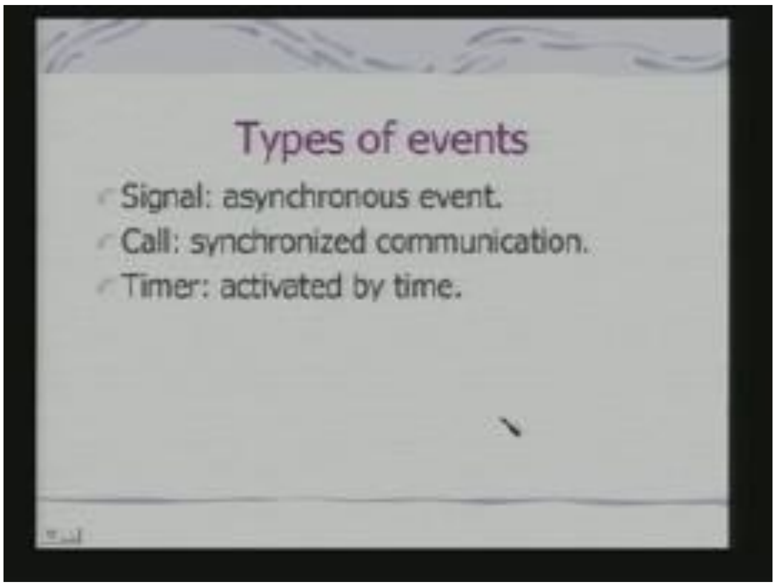


Event-driven state machines

- Behavioral descriptions are written as event-driven state machines.
  - Machine changes state when receiving an input.
- An event may come from inside or outside of the system.

So, behavioral descriptions are written as event driven state machines, because if you look in to it control algorithm which is running in an embedded system is nothing but a state machine. If we remember when we discuss tiny OS we said tiny OS is organize as a collection of concurrent FSM. So, its behavior is nothing but a collection of finite state machines. So, if you are designing the behavior how to express the design? We can express the design by drawing a finite state machine we should become an ambiguous specification of the behavior of the system. So, 1 state machine may not capture all aspects of the behavior. So, actually my system is described in terms of a collection of such finite state machine. An event may come from inside or outside of the system.

(Refer Slide Time: 52:08)

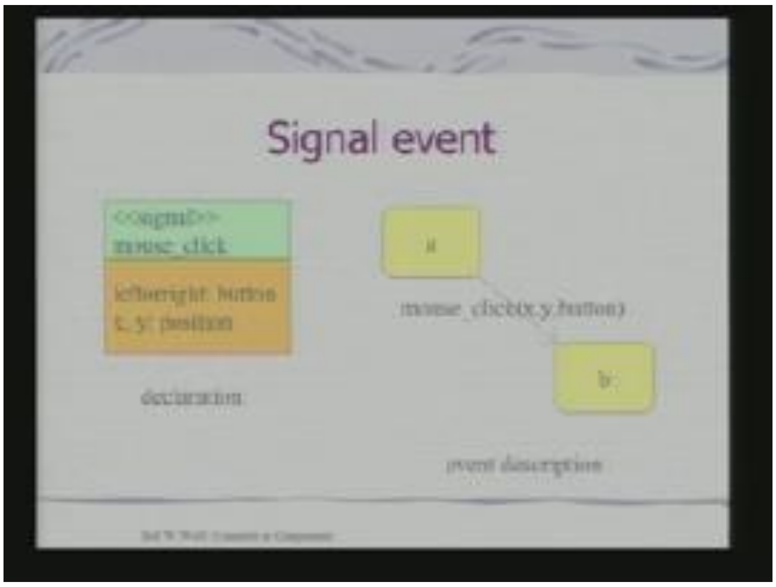


Types of events

- Signal: asynchronous event.
- Call: synchronized communication.
- Timer: activated by time.

So, events can be signal these are actually using your UML terminology signal call timer. So, signal is an asynchronous event. So, it can be an interrupt call is a synchronous communication just like a procedure call timer is in a sense asynchronous. But why it is not purely asynchronous? Because it will occur only at a expire of a time so, it is synchronous with the clock.

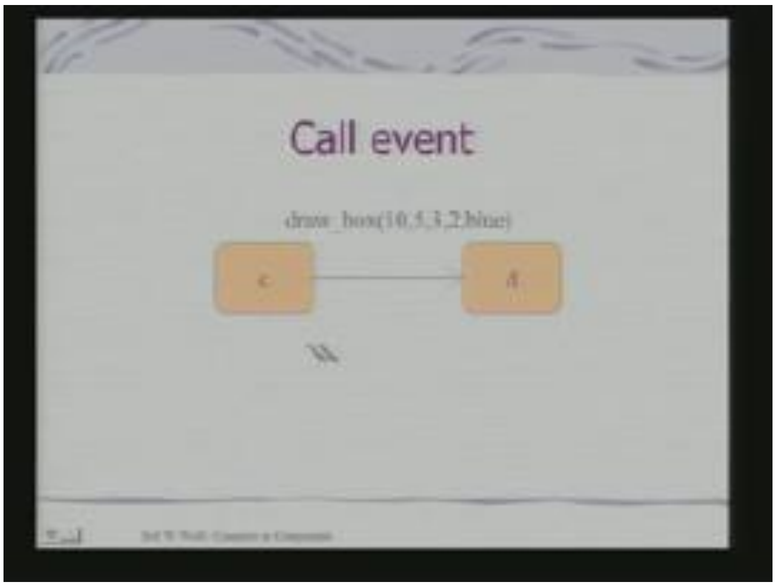
(Refer Slide Time: 52:37)



Signal event

So, signal event is you can declare a signal event. So, if you see here where using this concept of prototype. So, I am considering this as a example of a signal event mouse click. So, mouse click event has go attributes which are left or right button and x y position which button being pressed at what position being pressed. And if this is an event this event can actually can cause transition. So, behavior of a system if the mouse is clicked you make a transition from state a to state b its being described through this state transition diagram.

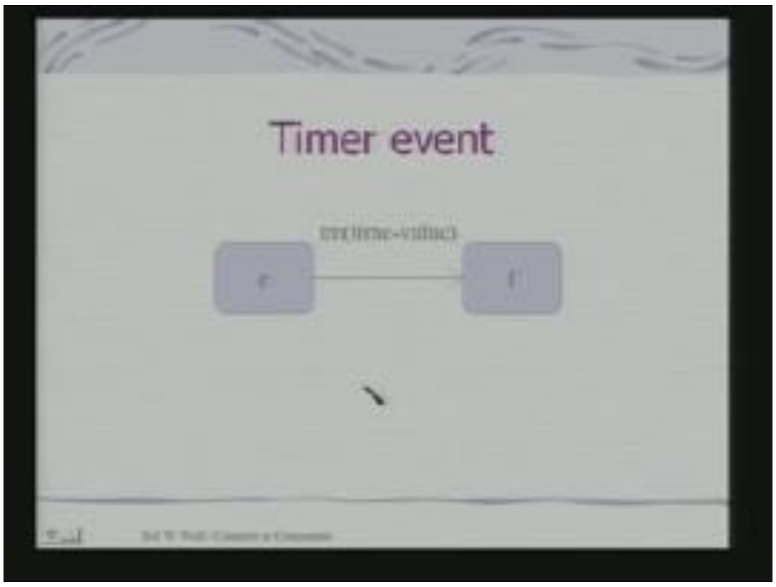
(Refer Slide Time: 53:19)



Call event

Similarly, if you look in to here it is a draw box. Draw box is a call event. It is a method call so, draw box is a call events. So, from c to d transition is talking place can this call event is actually activated.

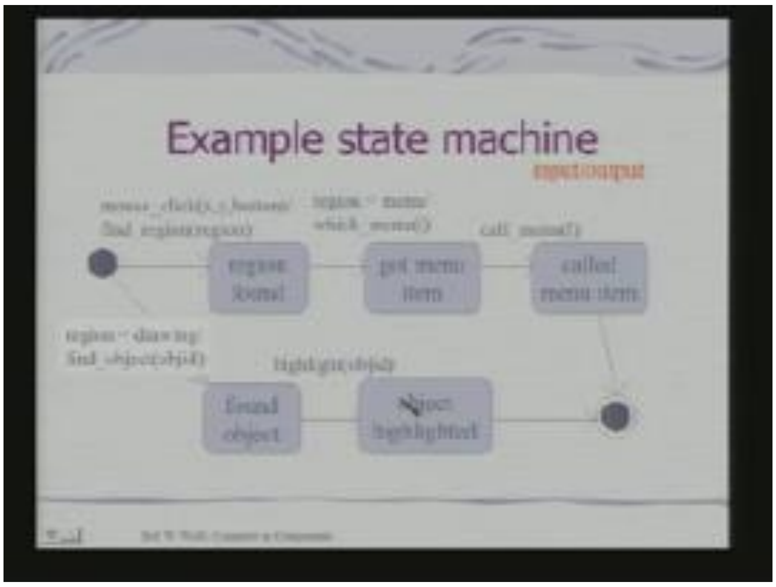
(Refer Slide Time: 53:38)



Timer event

Timer event is when the timer generates a signal at expiry of a time value. So, that is being represented by this value.

(Refer Slide Time: 53:48)

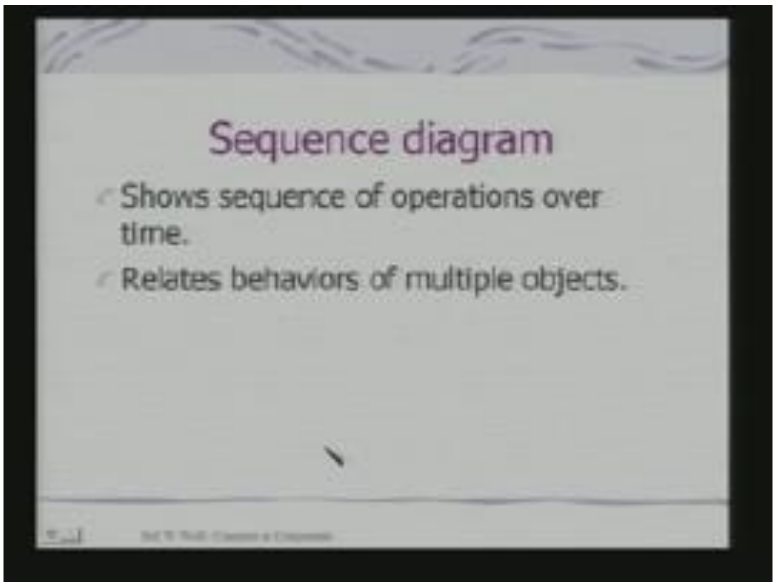


Example state machine

So, let us look at a complete state machine. So, what we have shown here is this input and output norm. So, this is an input and what will be the output at the particular state. This a start state typically in UML indicate this as a start state and what we say that? If it is a mouse click if they include input is the mouse click what is the output? Output is a call event is a find region. So, you find a region where the mouse clicks as actually taken place. So, with respect to that you know the region correspond to a menu. So, you actually generate the menu items.

So, you say that which menu this is the output at this point and when it is clicked that call menu when the menu is selected that is basically the function which goes to the called menu item. And that is that can lead to the end state this is the termination state, because I am showing a a small part of a finite state machine. So, is goes to the termination state. So, they can be another way that region drawing and find object. So, if I found an object so, highlight the object and object gets highlighted, because if I am processing a drawing. So, I am talking a mouse to highlight the drawing. So, you can see that system behavior is getting described through this kind of a state machine.

(Refer Slide Time: 55:15)

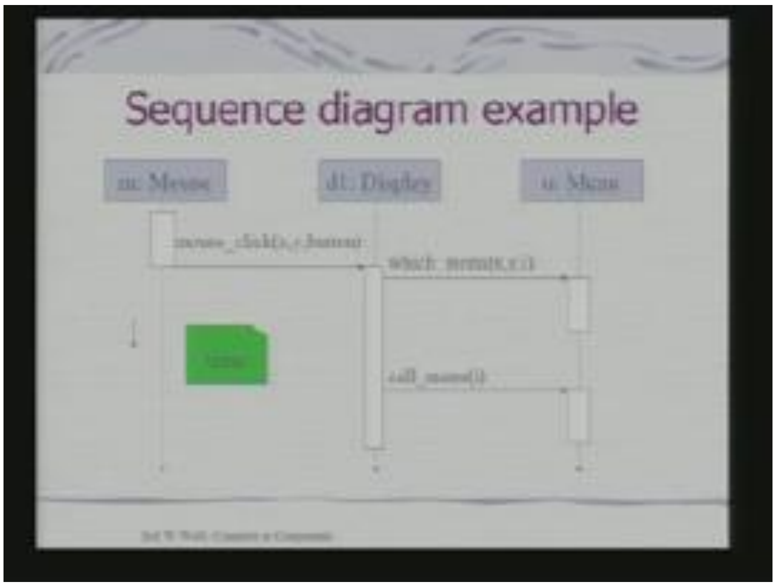


Sequence diagram

- Shows sequence of operations over time.
- Relates behaviors of multiple objects.

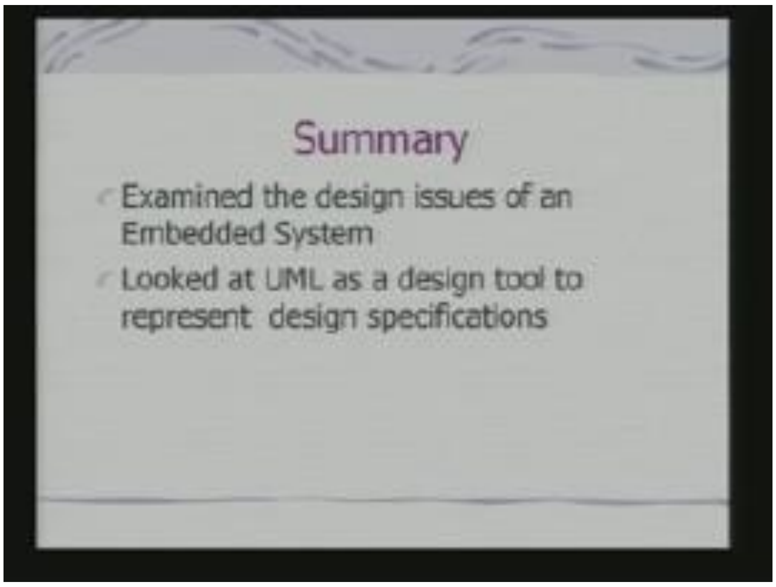
The other of way of doing that it or specifying the behavior of the system is where a sequence diagrams. So, sequence of operations of a time relate behaviors of multiple objects.

(Refer Slide Time: 55:31)


Sequence diagram example

So, if you look in to here the basic sequence diagram syntax is that these are the objects with each object I associate a time line. And from an object effectively what is happening? This is actually you can look as a event even generated at the mouse object and these event has occurred at a time instance which is given by these length along the time axis and that takes you to the display. So, if you remember the mouse click was a method in the display object. Because these event from mouse if you have this object these event actually has to be process by the method here. So, the methods are nothing but what? The message has being pass for the object. So, these object will actually invoke the mouse click method on the display object to process the mouse click. So, that is why it is shown in this fashion following these are the 2 action which takes place. So, which menus its finds out which menu is and then accordingly you means a call menu. So, this is the sequence diagram where explicitly temporal ordering of the activity has been shown. In a state transition diagram in the dependent of the states and the state transition on your input events as well as the output actions are being shown.

(Refer Slide Time: 57:16)



Summary

- Examined the design issues of an Embedded System
- Looked at UML as a design tool to represent design specifications

So, what we have seen so far? Therefore, the design issues of an embedded system. And looked at UML as a design tool to represent design specification. In fact, we have not looked at those aspects of UML by which we can express the architecture of the system we have just seen the components of the modules. We have just seen how do specify the behavior, how to specify the temporal ordering of the actions through a sequence diagram. But you have not seen how to model an architecture, how to model concurrency how to specify the concurrency related constraints for the system that we shall look in the next class. And once see a UML as a tool what it enables us? It enables us to just to describe the design specification. So, design constraints once you have that then actually the translation process would start. There are other design tools as well for the purpose of system modeling we shall also look at them in the next class.