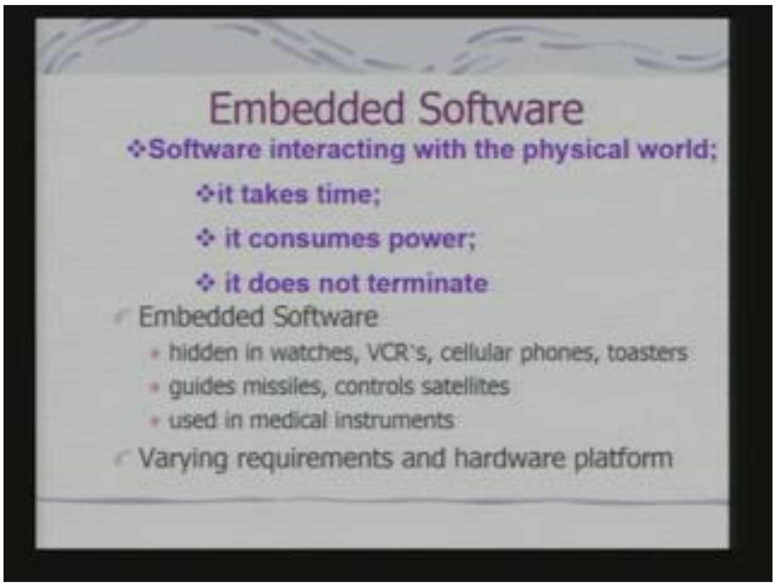
Embedded Systems
Dr. Santanu Chaudhury
Department of Electrical Engineering
Indian Institution of Technology, Delhi

Lecture - 19
Software for Embedded Systems

In today's class, we shall discuss particularly the software for embedded Systems. And in particular, the software environment and the practices to be followed for developing software's for embedded Applications.

(Refer Slide Time: 01:27)



Embedded Software

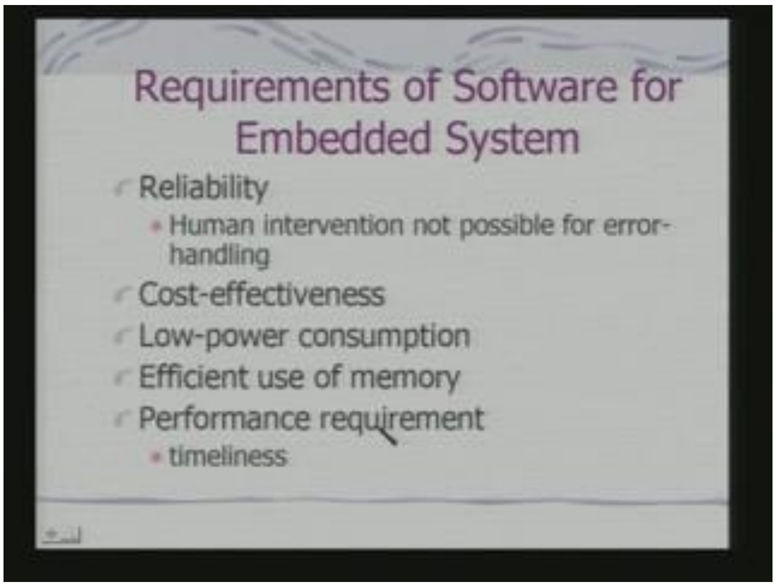
- Software interacting with the physical world;
  - it takes time;
  - it consumes power;
  - it does not terminate
- Embedded Software
  - hidden in watches, VCR's, cellular phones, toasters
  - guides missiles, controls satellites
  - used in medical instruments
- Varying requirements and hardware platform

So embedded software, a typically hidden in watches, VCR's, cellular phones, toasters. And, not like a general purpose software. It guides missals controls satellites also used in medical instruments. So obviously, this application scenario makes embedded software special and different from that of standard desktop software. You have varying requirements and the characteristic of hardware platform is also different. Because, you have already understood the hardware aspects of embedded systems.

In a way, this software for embedded systems interacts with physical world. It therefore, definitely takes time. So, there is an issue of time consume by the software in relation to the timings of the external events it consumes power. So, we have to think in terms of designing software which is power efficient. The last thing is that it does not terminate.

Unlike standard programs, where we expect then to terminate, an infinite number of steps these programs typically do not terminate.

(Refer Slide Time: 02:56)

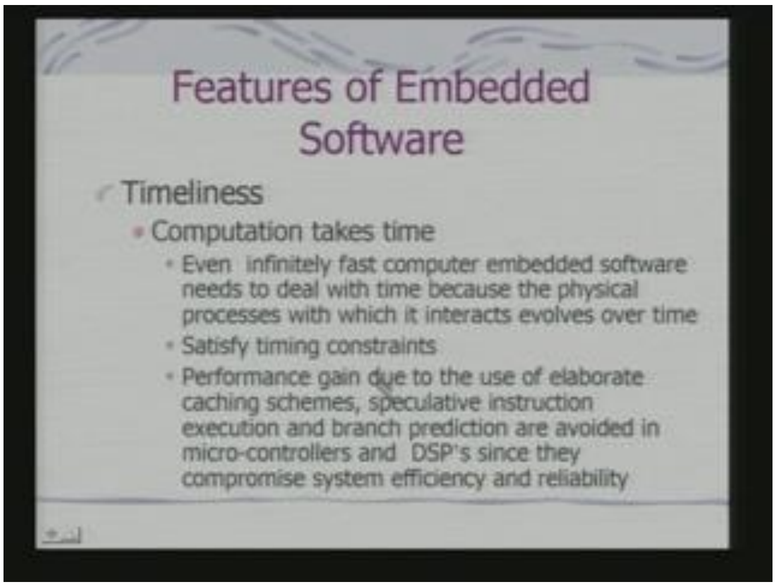


Requirements of Software for Embedded System

- Reliability
  - Human intervention not possible for error-handling
- Cost-effectiveness
- Low-power consumption
- Efficient use of memory
- Performance requirement
  - timeliness

What are the requirements of software for embedded System? Reliability, because human intervention is not possible for error handling. They are excepted to run without human intervention to long period. They should be cost effective. Low power consumption, they should make efficient use of memory, because the memory cannot be unbounded. And, the performance requirement the most important performance requirement is that of timeliness.

(Refer Slide Time: 03:31)



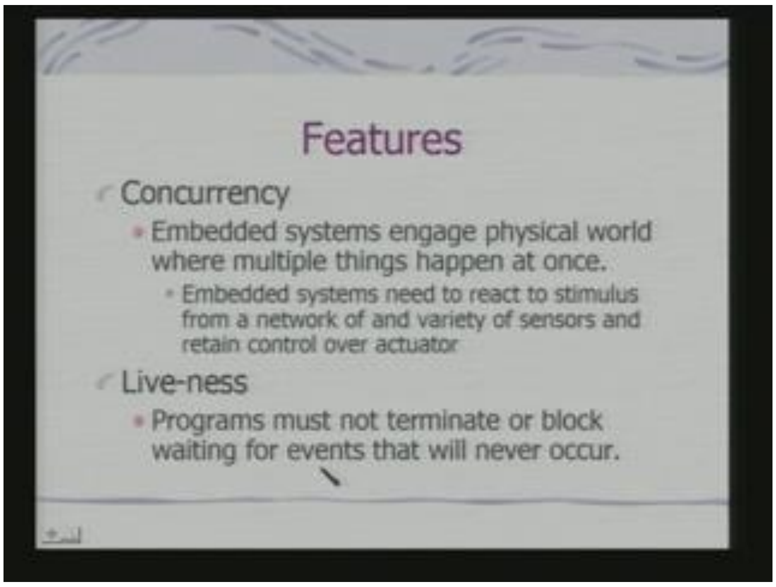
Features of Embedded Software

- Timeliness
  - Computation takes time
    - Even infinitely fast computer embedded software needs to deal with time because the physical processes with which it interacts evolves over time
    - Satisfy timing constraints
    - Performance gain due to the use of elaborate caching schemes, speculative instruction execution and branch prediction are avoided in micro-controllers and DSP's since they compromise system efficiency and reliability

So, the basis of requirements, what kind of features to expect in embedded software, the most importantly timeliness; computation takes time. Even if you have got infinitely first computer embedded Software needs to deal with time because, the physical a processor with, which interacts evolves over time. So, this timing conditions and constraints are to be followed.

And, the other important issue is that use of elaborate caching schemes speculative instruction execution branch predictions. These are avoided in microcontrollers and DSP’s which are targeted for embedded applications, because they may compromise system efficiency. In fact last time, we have seen that these hardware features kindly two additional power consumptions. So, we have to design software such that this timing constraints and met. And, we really do not have this special hardware features for speeding up the software.

(Refer Slide Time: 04:41)



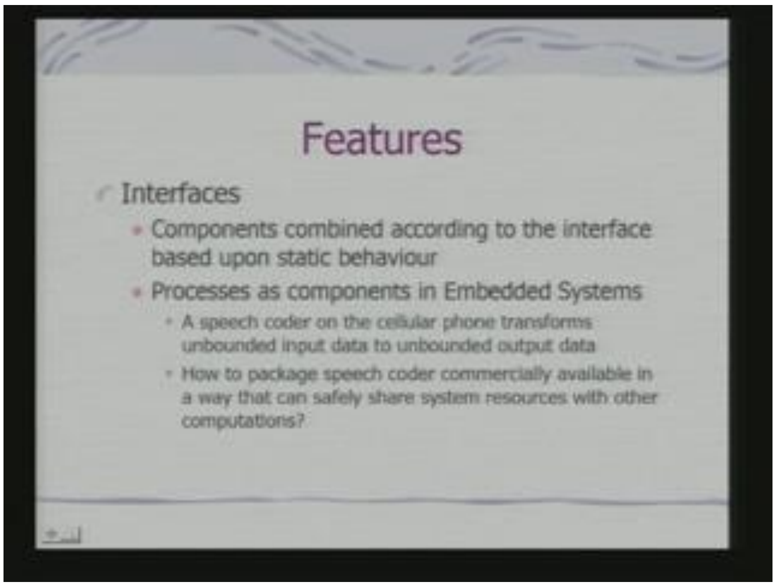
Features

- Concurrency
  - Embedded systems engage physical world where multiple things happen at once.
    - Embedded systems need to react to stimulus from a network of and variety of sensors and retain control over actuator
- Live-ness
  - Programs must not terminate or block waiting for events that will never occur.

The physical processors are concurrent. So, concurrency is a very important issue in relation to embedded Systems. So, embedded Systems make to react stimulus from a network of and variety of sensors and retain control over actuators. And so, the processing should be concurrent.

Related to concurrency is a issue of live-ness. Programs must not terminate are block waiting for events that will never occur. Because, if it gets blocked in such a way. Then, the whole system would basically fail. So in fact, watchdog timer is one of the hardware feature can be used to ensure live-ness.

(Refer Slide Time: 05:30)



Features

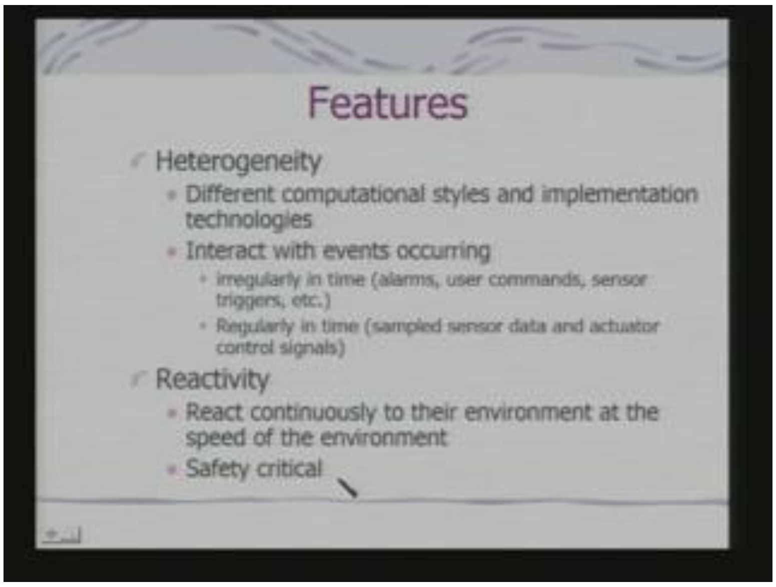
- Interfaces
  - Components combined according to the interface based upon static behaviour
  - Processes as components in Embedded Systems
    - A speech coder on the cellular phone transforms unbounded input data to unbounded output data
    - How to package speech coder commercially available in a way that can safely share system resources with other computations?

Next issue is that of interfaces, because when you build up software. You have to do a kind of a composition, a reuse of the components. So, if you go back to the basic object oriented framework. We can do reuse via inheritance or via composition. And, these compositions are particularly at the level of static behavior. In terms of dynamic behavior, if you are talking about processors.

Then, the issue comes up as how to combine the two different processors, so that they can work in a cooperative fashion. And, the other important issue is that the combination of the software which is coming from various sources may be third party software. And, how you can safely, combine them with the existing software.

The issue of security correctness of operation for this kind of composition is a important feature for embedded Applications. Since, you are getting software’s from different sources and also you need to deal with external live-ness which are of different nature Heterogeneity becomes an important component.

(Refer Slide Time: 06:43)



Features

- Heterogeneity
  - Different computational styles and implementation technologies
  - Interact with events occurring
    - irregularly in time (alarms, user commands, sensor triggers, etc.)
    - Regularly in time (sampled sensor data and actuator control signals)
- Reactivity
  - React continuously to their environment at the speed of the environment
  - Safety critical

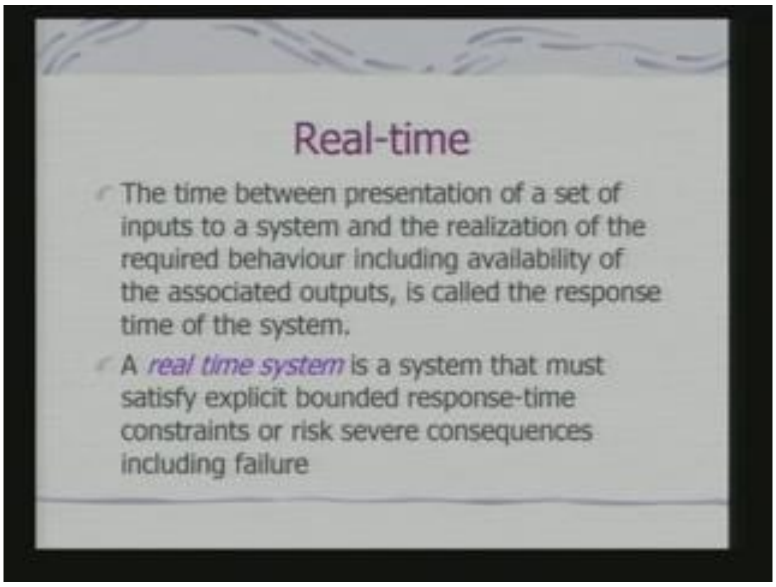
If you consider as simple mobile phone, the job of receiving. The speech data in a compress form decompressing and playing it back. As got the characteristic different from that of accessing your phone book. So, these two different types of computation and these two computation styles have to be handled together. And also the events, which occur in the external world may of different type. Then may be regular, then may be irregular.

And, your software needs to handle both kinds of events without unnecessary delayed. In a way, what we are talking about is reactivity of the software, which is of fundamental importants for embedded Systems. It should react continuously to the environment at the speed of the environment. If it cannot react at the speed of the environment then the whole situated ness of the plans gets diminished. So, the whole importance of dealing with external Live-ness as they occur makes embedded Systems, so powerful.

So, reactivity is an important issue with embedded Systems. They should be also safety critical, because in many cases there handling critical equipment a critical machinery. So, they cannot fail arbitrarily or fail without giving priority signals such that a catastrophe lockers. So, they have to be there have to be built in safety critical features with the software.

We are so far talking about that reactivity. The reactivity means reacting to the external environment. And since, it is reacting to the external environment; obviously, there would be bounds, bounds in the response time.

(Refer Slide Time: 08:46)



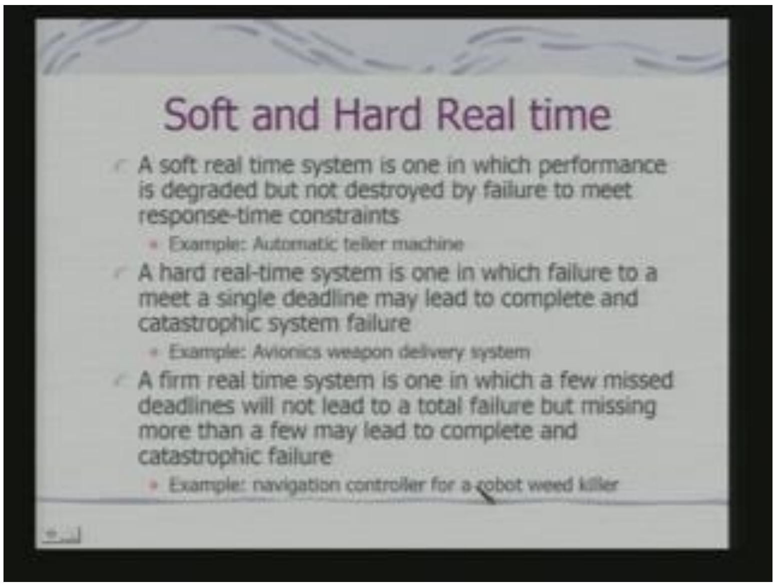
Real-time

- The time between presentation of a set of inputs to a system and the realization of the required behaviour including availability of the associated outputs, is called the response time of the system.
- A *real time system* is a system that must satisfy explicit bounded response-time constraints or risk severe consequences including failure

So formally, there would be a response time bound and what is really of response time, the time between presentations of a set of inputs to the system and realization of the required behavior, including availability of associated outputs. So, this is, what is response time. So, when there is an external signal coming in corresponding to the characteristics of the external signal. There would be bound on the response time and that list as to the formal definition of real time systems.

A real time system is a system that must satisfy explicit bounded response time constraints or risk severe consequences including failure. But, one point also to be noted that all embedded Systems are not necessarily real time systems. There as some systems which are definitely real time and for them this bound is critical.

(Refer Slide Time: 09:52)



Soft and Hard Real time

- A soft real time system is one in which performance is degraded but not destroyed by failure to meet response-time constraints
  - Example: Automatic teller machine
- A hard real-time system is one in which failure to a meet a single deadline may lead to complete and catastrophic system failure
  - Example: Avionics weapon delivery system
- A firm real time system is one in which a few missed deadlines will not lead to a total failure but missing more than a few may lead to complete and catastrophic failure
  - Example: navigation controller for a robot weed killer

This real time systems can be categorized further we talk about soft real time systems and hard real time systems as well as form real time systems. A soft real time system is one in which performance is degraded. But, not destroyed by failure to meet response time constraints. It may be an automatic teller machine. There may be delay in delivery of the cash or delivery of the reseed. But, the delay is not critical. This is true for a typical DVD player as well when it is decompressing a video frame for play back.

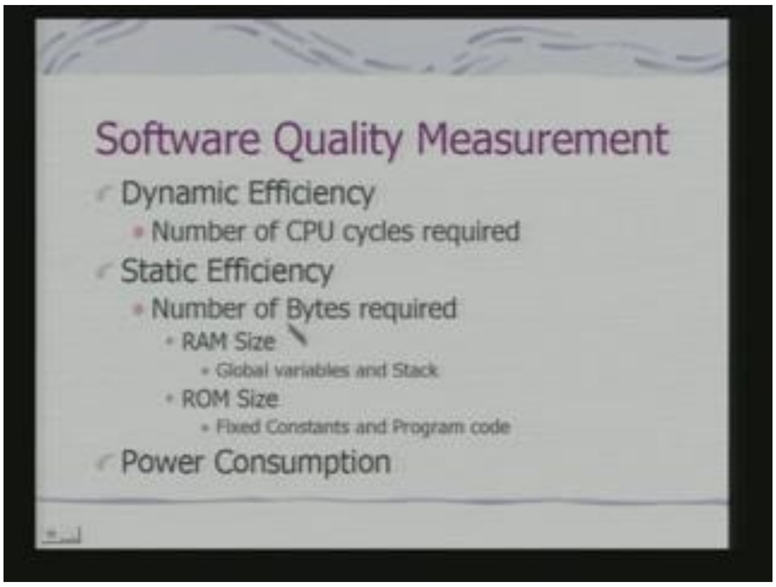
In fact in a way, you can consider that all systems or all software for that matter in some other soft real time. Because, you would not like to use to what processor which would give you response time of a 5 minutes. So, there is also a kind of minimum expectations for the interactive systems. A hard real time system is one in which failure to meet a single deadline may lead to complete and catastrophic system failure. So, example is say Avionics weapon delivery system.

Because, after the decided presses a button to release a weapon from that point to the actual delivery should be bounded. Otherwise the target would be missed. So, the whole purpose of releasing the weapon is lost. So, this is a hard real time system. The other example could be control system in a nuclear power plant evolved has to be closed within a definite bound otherwise there may be an explosion.

A form real time system is one in which a few missed deadlines will not lead to a total failure. But, missing more than a few may lead to complete and catastrophic failure. So, it is somewhere between hard and soft real time systems. An example is a navigation controller for a robot weed killer. This is slightly different kind of an example.

So, if a robot weed killer is moving around there will be deadlines corresponding to its navigation. It can miss few deadlines but, if it continuous to miss that meet miss the deadlines, what it what can happen. It may actually destroy the good scrapes instead of the weed. So, the effect can be really catastrophic.

(Refer Slide Time: 12:28)



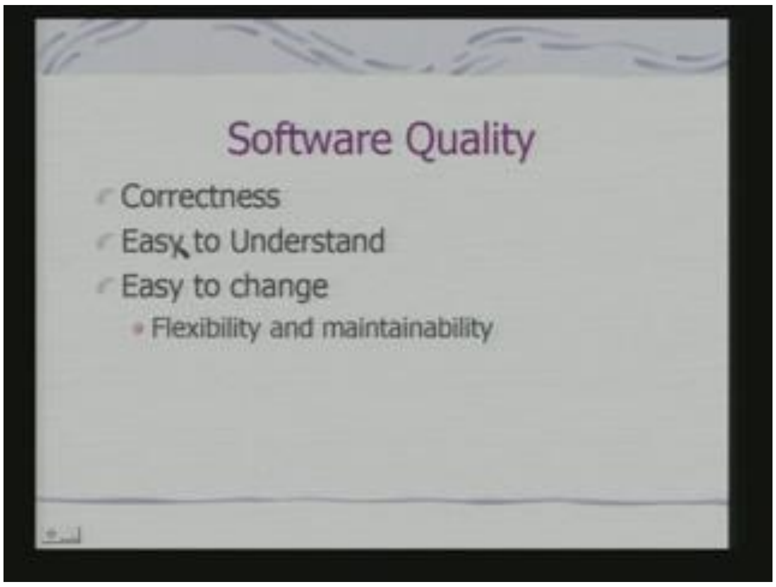
Software Quality Measurement

- Dynamic Efficiency
  - Number of CPU cycles required
- Static Efficiency
  - Number of Bytes required
    - RAM Size
      - Global variables and Stack
    - ROM Size
      - Fixed Constants and Program code
- Power Consumption

Now, how do you measure quality of such software what is dynamic efficiency that is number CPU cycles required for execution of the software. The other part is a static efficiency that is the number of bytes required, the RAM size the ROM size. The RAM would accommodate the variables and the runtime stack as well as the heat. The ROM size would determine the fixed constants on the program code.

ROM we are using in a generic sense. It can be EPROM E square PROM. It can be even flash. And the power consumption is another issue. So, these three are very important parameters to measure quality of a software targeted for embedded Application.

(Refer Slide Time: 13:22)

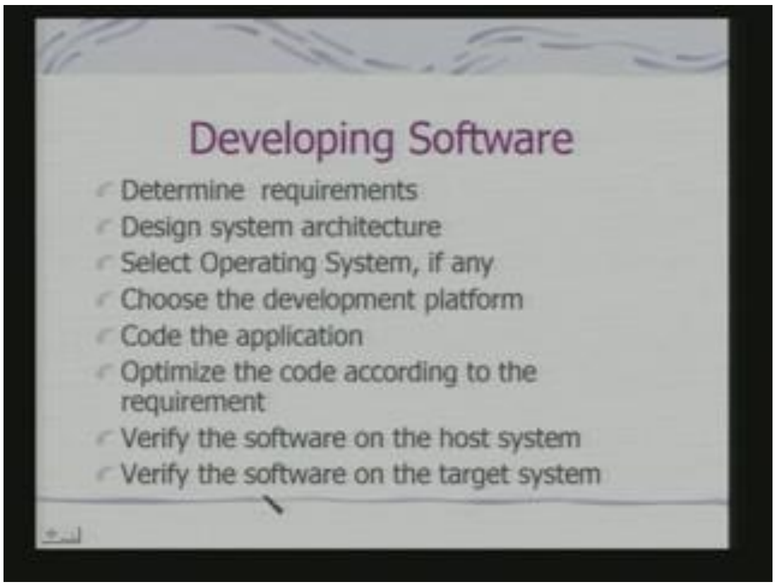


Software Quality

- Correctness
- Easy to Understand
- Easy to change
  - Flexibility and maintainability

The other issues the obvious is correctness logical correctness as well as temporal correctness and these two a primarily for the purpose of maintaining and reusing the software. So, it should be easy to understand it should be easy to change. So, ensure flexibility and maintainability. These are more from the classical software engineering perspective.

(Refer Slide Time: 13:48)



Developing Software

- Determine requirements
- Design system architecture
- Select Operating System, if any
- Choose the development platform
- Code the application
- Optimize the code according to the requirement
- Verify the software on the host system
- Verify the software on the target system

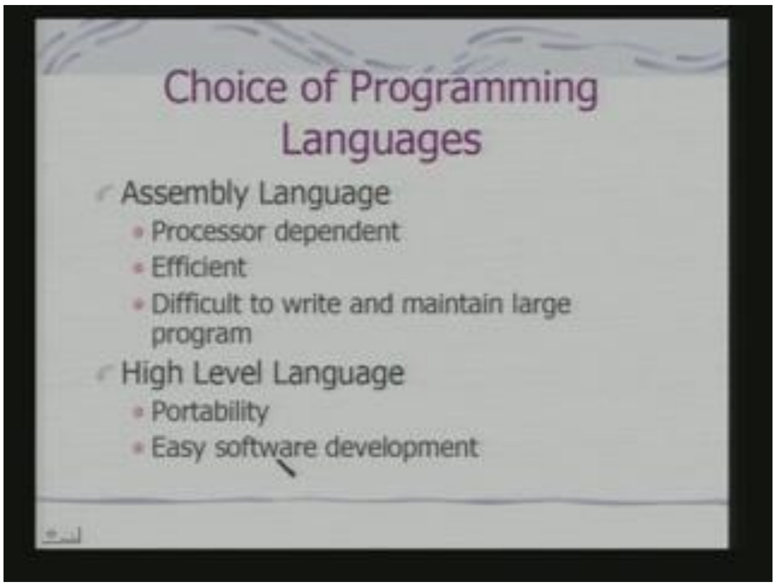
So, what is a process of developing software for an embedded System? To be very similar to that of general software, but there may be and there will be some differences.

First you; obviously, determine the requirements; you design the system architecture, the software system architecture with respect to the requirements of the system.

Architecture, you select Operating System. It is not that for all embedded Systems you really require an OS. If you really require an OS, what kind of characteristics, the way should satisfy. Then, you choose the development platform because, the development platform is in many ways depended on the OS that we choose. Then, you obviously, code the application.

Optimize the code, according to the requirement you should optimize the code. So that, you can meet the quality measures verify the software in the host system. It may be simple simulation and verify the software on the target system.

(Refer Slide Time: 14:51)



Choice of Programming Languages

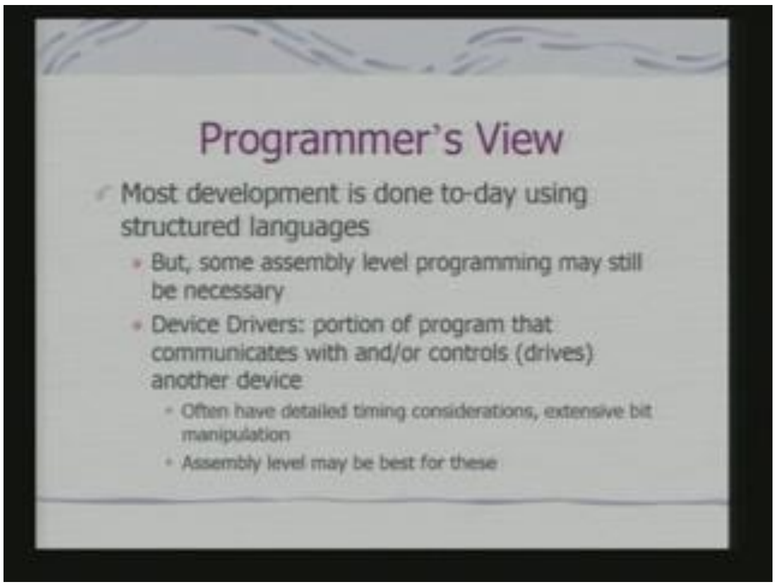
- Assembly Language
  - Processor dependent
  - Efficient
  - Difficult to write and maintain large program
- High Level Language
  - Portability
  - Easy software development

So, what are the different choices of programming languages that you can use for developing the software? It may be assembly language; it may be high level language. If you are using assembly language assembly languages obviously, a processor dependent an assembly language program can be really efficient. Because, you can exploit the architectural features of the target processor directly.

But, it is difficult to maintain and write large programs. Because then, it becomes complex, because the number of instructions that would be required will be much more. As a number of instructions increases the probability of bugs coming into the core also increases. At the same time, the other problem is portability. Because, if you decide to change the processor. Then, your software has to be completely changed.

High level language; obviously, ensures portability and easy software development. Because, the number of instructions the number lines of core that you need to handle for a given task would be much smaller. And the compiler performs the job of translation. But, compiler may not have the capability to optimize the target core with reference to the target architecture.

(Refer Slide Time: 16:24)



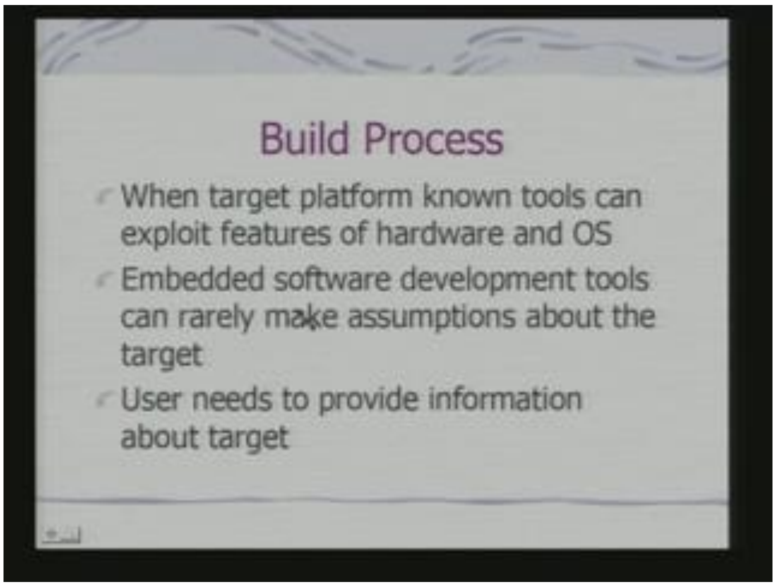
Programmer's View

- Most development is done to-day using structured languages
  - But, some assembly level programming may still be necessary
  - Device Drivers: portion of program that communicates with and/or controls (drives) another device
    - Often have detailed timing considerations, extensive bit manipulation
    - Assembly level may be best for these

So, if you look at the basic programmers view, what we find today? The most development is done today using structured languages that is high level languages. But, some assembly level programming may still be necessary. In particular the device drivers, the portion of program that communicates with or controls another device, which is connected with your basic micro controller for those device interfacing applications.

You still need to do assembly language programming. Because, you have to take care of very strict timing constraints and may involve extensive bit manipulation. So, for that purpose assembly language program is pretty efficient.

(Refer Slide Time: 17:06)

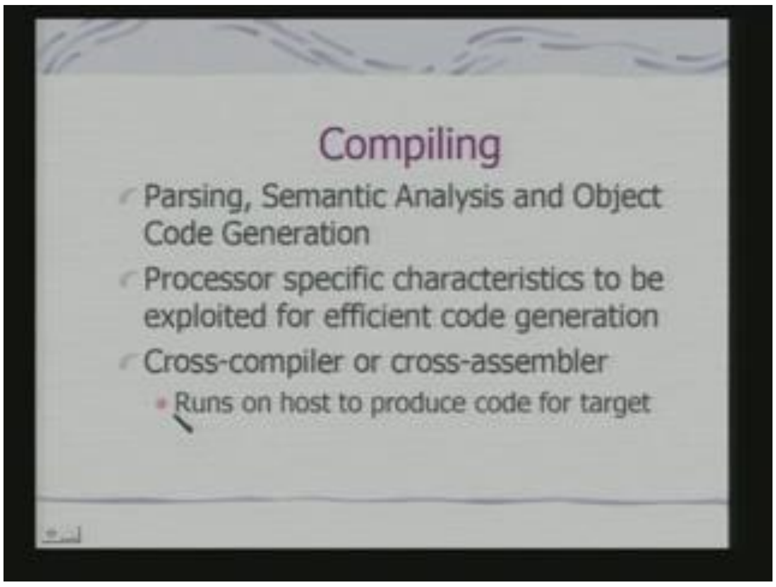


Build Process

- When target platform known tools can exploit features of hardware and OS
- Embedded software development tools can rarely make assumptions about the target
- User needs to provide information about target

So therefore, the software development goes through what is known as a built process. Now, when target platform is known tools can exploit features of the hardware on the OS in the building process. But, it is not always that you can make assumptions about the target. So, in that condition user needs to provide information about the target. So, the build environments which are targeted for these kinds of developments always have provision for putting in these kind of parameters from the uses.

(Refer Slide Time: 17:53)



Compiling

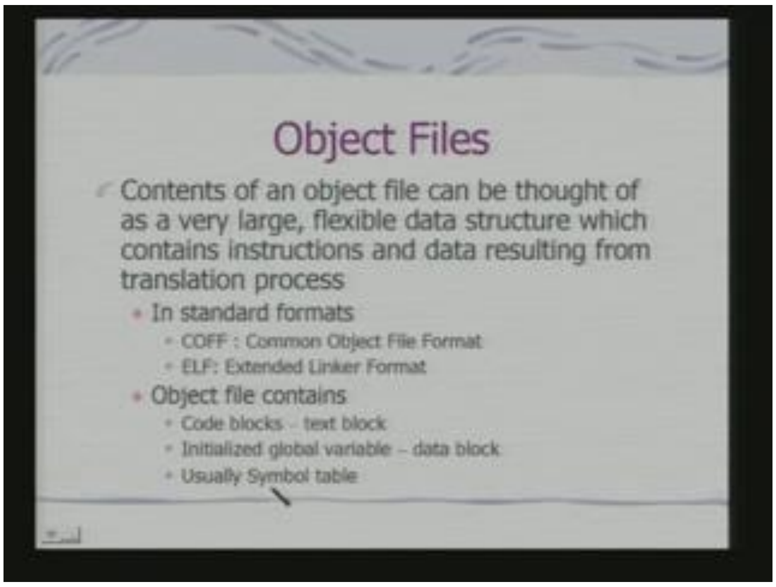
- Parsing, Semantic Analysis and Object Code Generation
- Processor specific characteristics to be exploited for efficient code generation
- Cross-compiler or cross-assembler
  - Runs on host to produce code for target

The most important aspect of the component of the build processor is compile. And the compiling is important here that aspect of compiling is important is the object code generation and what we call code optimization. Because, the object code has to be optimize with respect to the target architecture. And, you need to optimize also the code that you are generating. You cannot really translate straight way a high level language code in to the target machine instructions.

There has to be some kind of the core transformations if we one to optimal exploit. So, the register set of the processors. So that aspect of compiler design and implementation as of tremendous relevance an importance for embedded Systems. Then, the other important issue is that you actually use cross compilers or cross assemblers. This compilers or assemblers actually run on host which is different from the target.

Typically, a standard compiler or C compiler in a Linux machine, you will run and generate the code for the target architecture itself. The target architecture may be the Pentium itself. But, when you are using a across compiler it may run on a Linux machine. But, it would generate the core for say ARM processors.

(Refer Slide Time: 19:26)



Object Files

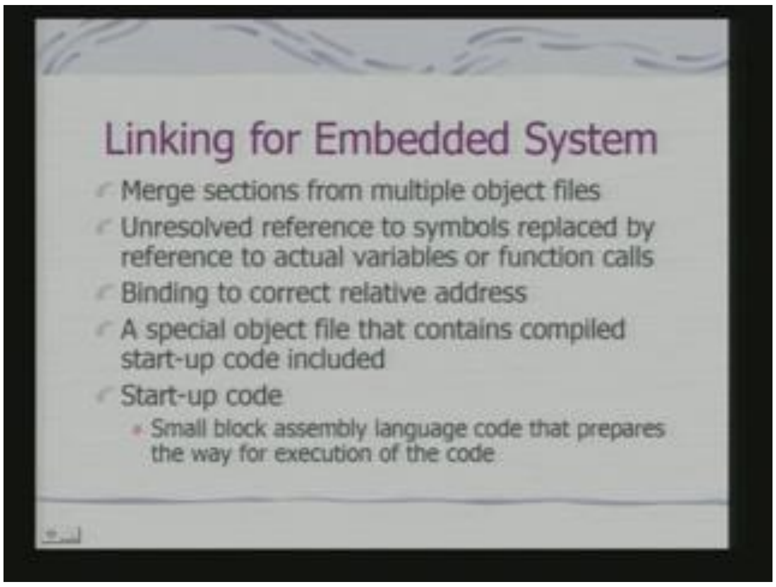
- Contents of an object file can be thought of as a very large, flexible data structure which contains instructions and data resulting from translation process
  - In standard formats
    - COFF : Common Object File Format
    - ELF: Extended Linker Format
  - Object file contains
    - Code blocks – text block
    - Initialized global variable – data block
    - Usually Symbol table

Now, you what does compilation rate? Compilation rate object files. Contents of object file can be thought of as a very large flexible data structure which contains instructions and the data resulting from the translation process. So, the object file is not directly executable image. So, object file which is generated in the host system is not definitely the executable image.

They come in standard forms, these are common object file format COFF ELF extended link a formats. Why they come in standard form? Now, these object files can be used with the linkers are the software tools, which are targeted for the specific platforms for generating the executable image.

So, object files typically contains code blocks which are text block, which we refer to a text block initialized or even un initialized global variables, which is a data block. And usually the symbol table and this symbol table is used in many cases for debugging the software.

(Refer Slide Time: 20:32)



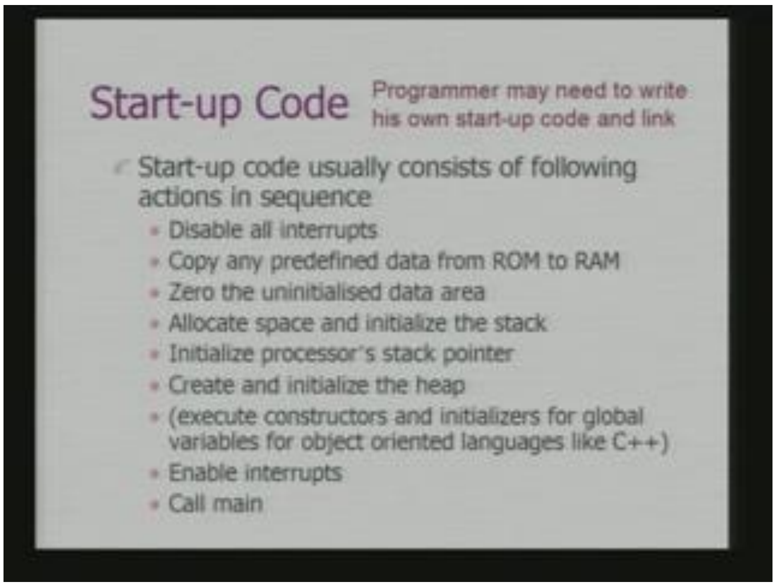
Linking for Embedded System

- Merge sections from multiple object files
- Unresolved reference to symbols replaced by reference to actual variables or function calls
- Binding to correct relative address
- A special object file that contains compiled start-up code included
- Start-up code
  - Small block assembly language code that prepares the way for execution of the code

Next comes a linking, the next important processor linking, because you can have the compiler compiling individual modules and generating the object code. So, linking is basic process of merging sections for multiple object files. The unresolved reference to symbols or replaced by reference to actual variables or function calls because, these references to the variables may be to modules external to the current module. And they are to be bound to correct relative address.

As special object files that contains compiled start up code is included. So, what is the start up code? Start up code is a small block of assembly language code that prepares the way for execution of the code. In fact, this is typical to that of an embedded systems. Because, when you are linking your linking the different modules you have resolve the address references. You have resolved the address references with respect to the relative addresses.

(Refer Slide Time: 21:47)



Start-up Code

Programmer may need to write his own start-up code and link

- Start-up code usually consists of following actions in sequence
  - Disable all interrupts
  - Copy any predefined data from ROM to RAM
  - Zero the uninitialised data area
  - Allocate space and initialize the stack
  - Initialize processor's stack pointer
  - Create and initialize the heap
  - (execute constructors and initializers for global variables for object oriented languages like C++)
  - Enable interrupts
  - Call main

But now, you have to initialize a system to get the code running. And that code is typically is your start up code. So, start up code usually consists of following actions in sequence. You disable all interrupts typically, copy any predefine data from ROM to RAM zero, the uninitialized data Area allocates space and initialize a stack. Initialize processors stack pointer create and initialize the heap.

Execute constructors and initializes for global variables for object oriented languages like C++. Enable interrupts and then call main. So, what does this all this things mean really, with regard to an embedded target platform. So, you basically prefer the environment for the software to run. Many of these jobs typically will be done on a general purpose computing platform via loader.

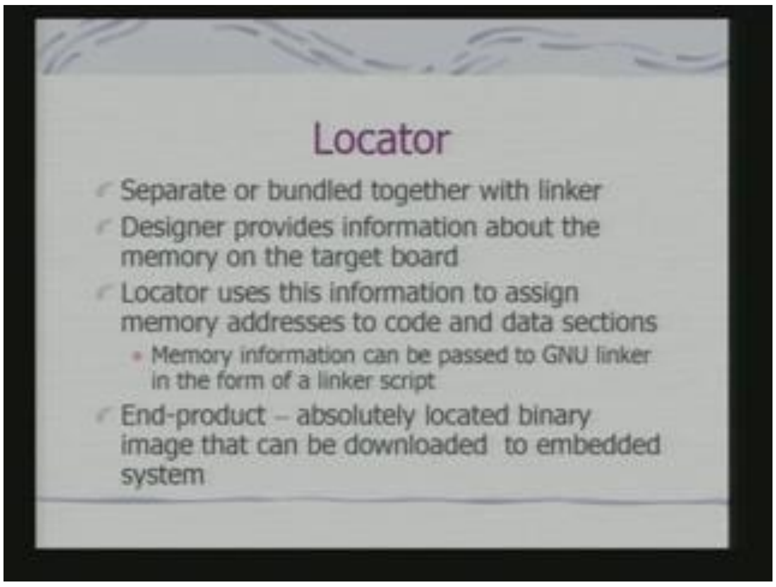
But, you really do not have a loaded in this case. You are directly loading the software on to the processors memory or the target board's memory for the purpose of execution. So, this startup code can include all these features because for execution of a program, what you need? You need stack to accommodate local variables as well as a retain addresses. So, stacks needs to be initialized for dynamic memory allocations you need heap.

So, heap is also to be initialized, what does that mean, it means basically allocating memory areas which are to be used for heap area as well as stack area. And the area in which actually the program would really reside, why you disable interrupts, because until and unless these parameters as set you cannot possibly service the interrupts. Because, interrupt service routine may be part of your code. In fact, this is typically done for an initialization of a system.

In fact for a raw system, a programmer may need to write his own startup code and link if a programmer is using a target OS on the target board. The OS may supply the corresponding startup code and please try to understand one very basic issue is that you will be loading OS with specific functionality and not general purpose functionality.

So, the Startup Code which comes the OS needs to be link with a software’s that you are developing along with the modules, which may come from the OS to develop the complete software environment for the target embedded System. You also have what is called the locator.

(Refer Slide Time: 24:32)



Locator

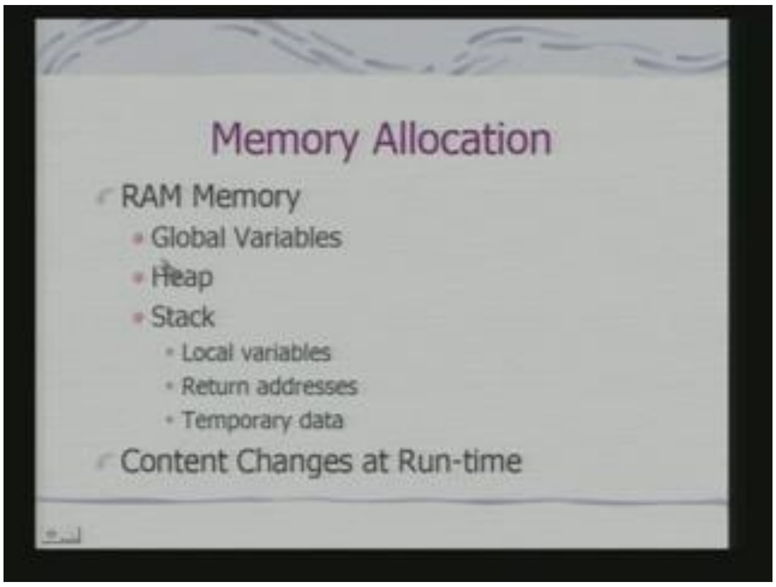
- Separate or bundled together with linker
- Designer provides information about the memory on the target board
- Locator uses this information to assign memory addresses to code and data sections
  - Memory information can be passed to GNU linker in the form of a linker script
- End-product – absolutely located binary image that can be downloaded to embedded system

Locator is, what is bundled together with a linker. Now, designer provides information about the memory on the target board via the Locator. And Locator uses this information to assign memory addresses to code and data sections. In fact, genuine link has this provision that can be pass to genuine linker in the form of a linker script.

So, you need to know the memory map of the target board your host system where you have actually done the linking does not know it. The linker has also done the job that of a relative address resolution. So, this locator gives you the memory map. So, you can actually create the memory image fine and you use along with this the startup code.

Because, now startup code exactly knows how the memory has to be partition between stack heap, as well as a program code. And then you load the complete bundle on the target board for the purpose of execution.

(Refer Slide Time: 25:41)



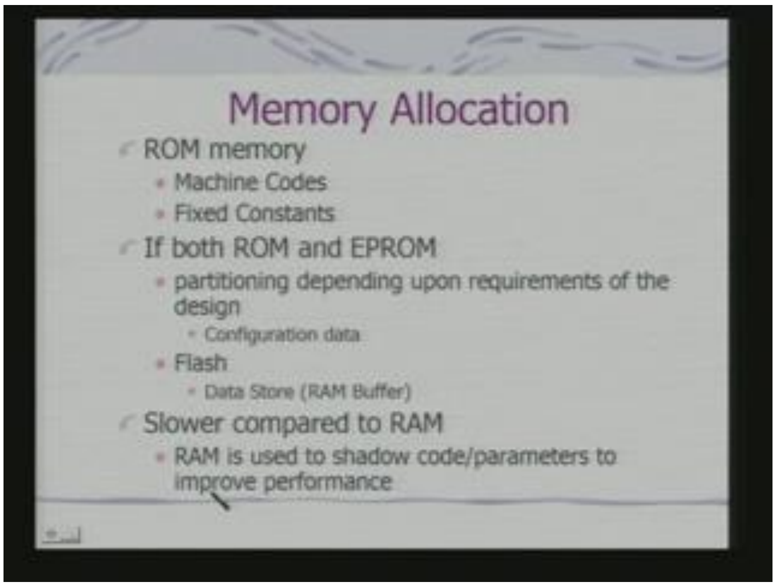
Memory Allocation

- RAM Memory
  - Global Variables
  - Heap
  - Stack
    - Local variables
    - Return addresses
    - Temporary data
- Content Changes at Run-time

So, the one very basic issue is that of memory allocation for any kind of software environment, what is global variables, because a program when it is in execution. It got to have the global variables, got to have the heap for dynamic memory allocations. In fact for an object oriented language, the objects are also allocated in heap.

You have got stack for local variables return addresses and temporary data. And this contains changes in run time. So, the RAM memory is distributed among these components. And if you can estimate the size of these components beforehand that. You determine the size of RAM that you actually require on the target board.

(Refer Slide Time: 26:22)



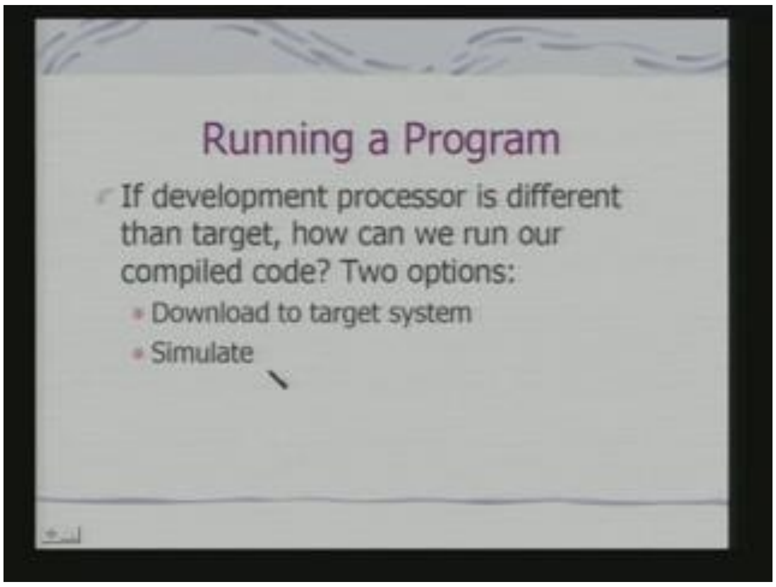
Memory Allocation

- ROM memory
  - Machine Codes
  - Fixed Constants
- If both ROM and EPROM
  - partitioning depending upon requirements of the design
    - Configuration data
  - Flash
    - Data Store (RAM Buffer)
- Slower compared to RAM
  - RAM is used to shadow code/parameters to improve performance

The ROM memory has the machine codes as well as the fixed constants. If you have both ROM and EPROM partitioning depends upon the requirements of the design. Because, the configuration data can be typically put in ROM. And your software can going on EPROM which can be modified or updated.

The Flash can also have your basic software, but it many cases flash can be used for a data store say at typical camera application. You would like to have the images stored in the flash. And flash is slower compare to the RAM and if an ROM you have found to slower compare to RAM. So, you have to see that RAM is used to shadow code or parameters to improve performance. So, typically it is copied from ROM or flash to the RAM if that is containing your program code. So, it is faster in terms of access.

(Refer Slide Time: 27:29)

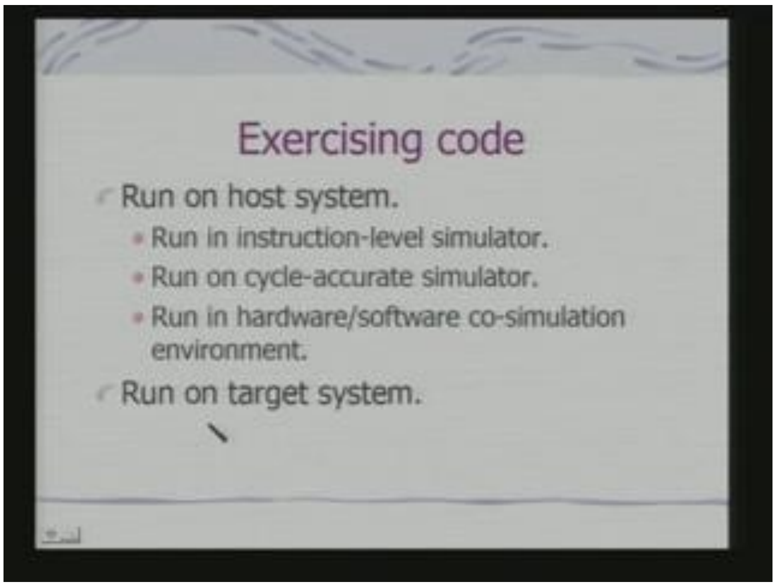


Running a Program

- If development processor is different than target, how can we run our compiled code? Two options:
  - Download to target system
  - Simulate

So, running a program is if development processor is different from the target you download to the target are you simulate.

(Refer Slide Time: 27:37)



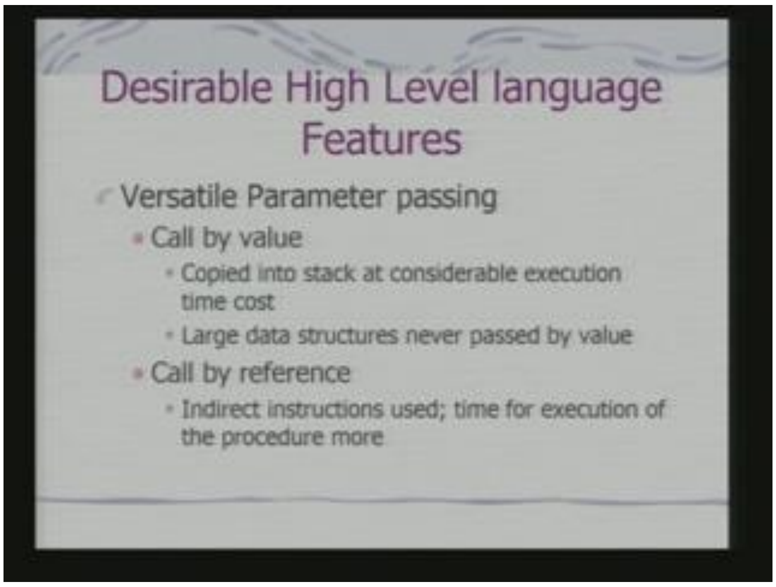
Exercising code

- Run on host system.
  - Run in instruction-level simulator.
  - Run on cycle-accurate simulator.
  - Run in hardware/software co-simulation environment.
- Run on target system.

This is the two basic thing you stimulate this is the two basic things. And the basic process of excising a code is run on the host system. So, run on a instruction level stimulator cycle accurate simulator. Run in hardware software co stimulation environment, where you stimulating hardware component as well. Once you satisfied about your software in this environment.

Then, you go to the target system in fact, when you are actually going to this target system. Then this locator as a system programming component as system software as well as you startup code becomes critically important.

(Refer Slide Time: 28:22)



Desirable High Level language Features

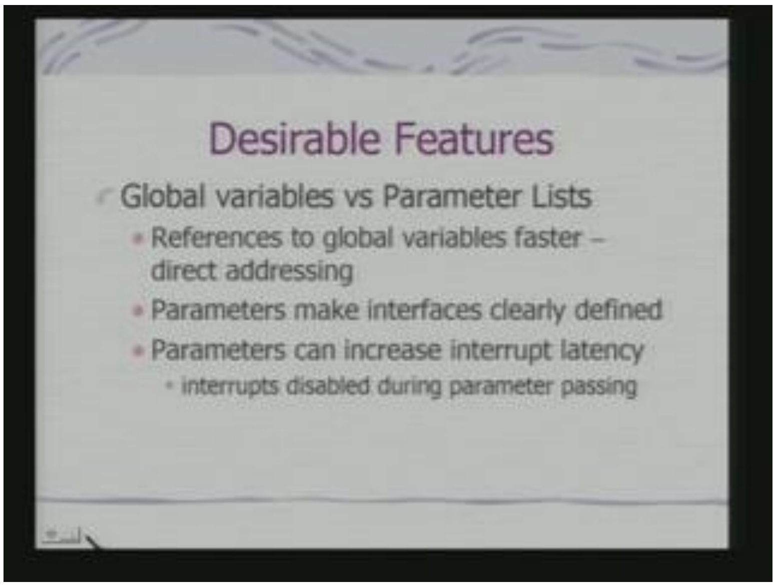
- Versatile Parameter passing
  - Call by value
    - Copied into stack at considerable execution time cost
    - Large data structures never passed by value
  - Call by reference
    - Indirect instructions used; time for execution of the procedure more

Next we look at the issues related to develop and a programming high level language for embedded systems. Now, all of us know the basic programming all of us know the basic programming techniques and tools. What we shall look at here, is that this different mechanisms which are available in a high level language, what are their implications, with reference to embedded Systems.

So, let us look at parameter passing. If you look at call by value call by value means the value of the variables are actually copy to the stack. So, there is always a time cost. But and that is, why you will never find that large data structures and never pass by value. You actually a pass on a pointer a pointer is pass by value, but if you a passing by value that is calling by reference.

The problem is you have to use indirect addressing modes for accessing the variables. Since you are using indirect addressing modes you are paying time for execution of a code. So, if you need to optimize in terms of timing parameters. You may need to decide which mechanism to be followed for parameter passing. So, there are tradeoffs and these tradeoffs are to be balanced for deciding on the mechanisms to be followed.

(Refer Slide Time: 29:49)

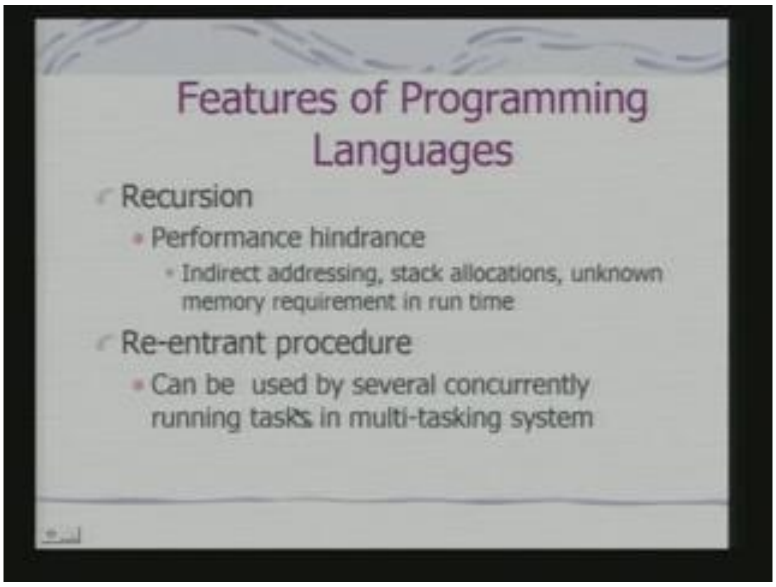


Desirable Features

- Global variables vs Parameter Lists
  - References to global variables faster – direct addressing
  - Parameters make interfaces clearly defined
  - Parameters can increase interrupt latency
    - interrupts disabled during parameter passing

The global variables and parameter list, so global variables references to the global variables will be definitely faster. Because, you can do a direct addressing to global variables, but parameters make interfaces clearly defined. Because, if you are using only global variables then if the global variables gets modified. There can be side effects and that can introduce a bug. So, that is why the global variables from the purpose of software designs are to be avoided. But, they provide you the fastest mechanism for communication. If you are really press for time and time deadline you may need to resort to global variables.

(Refer Slide Time: 30:33)



Features of Programming Languages

- Recursion
  - Performance hindrance
    - Indirect addressing, stack allocations, unknown memory requirement in run time
- Re-entrant procedure
  - Can be used by several concurrently running tasks in multi-tasking system

Recursion is something which should be avoided. Fundamentally, because recursion requires, what indirect addressing, also you really do not know the memory requirement beforehand. The memory requirement in run time you cannot really asses beforehand, because that would depend on the recursion depth.

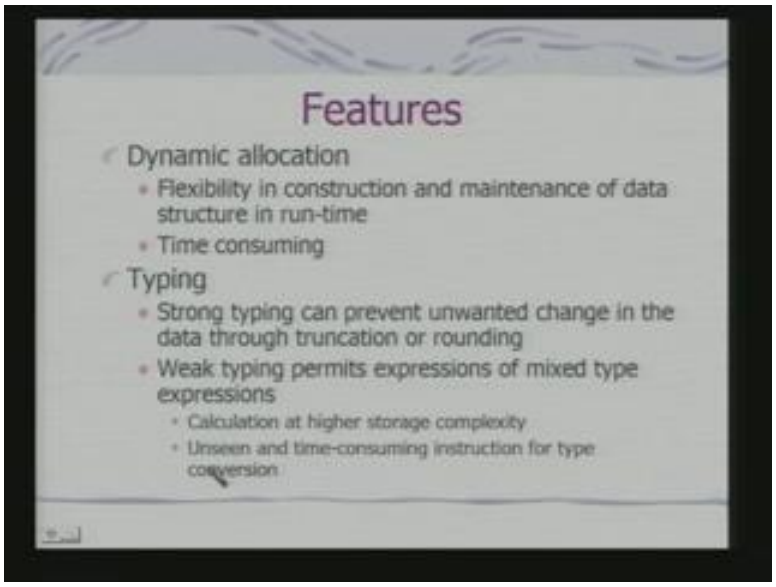
So, it is always desirable that when you are writing software for embedded applications you replace recursion by iterative constants. So you really know that, the number of times that will be executed. So, you have a more deterministic memory requirement assessment. The other issue is that Re-entrant procedure. A re-entrant procedure can be used by several concurrently running tasks in multi tasking systems.

And if you really have a multi tasking system a basic requirement is proceed yours must be re-entrant, what does reentrancy imply, it implies that if you are writing such proceed here. You should take here; you should save the state of the processor; that means all the registers as well as the variables that have been used. This variables cannot be in an arbitrary memory locations or variable must be local variables must be in the stack.

And all registers must be stored in the stack. Before this procedure starts execution. And once you come out of that procedure these state must be restored back. These make sure that if the same proceed here is invoked. By another concurrent process that same procedure can be used by another concurrent process without disturbing the data of this procedures invocation by the original thread.

So, a very simple example could be if you using an editor say VI editor in an UNIX system. There could be only one VI editor running and each user can use different files to edit account. Since the program or the procedure is designed in an re-entrant form there one be any conflict. And there may be multiple VI editors or multiple version of the same procedure running in the context of multiple concurrently running tasks.

(Refer Slide Time: 33:05)



Features

- Dynamic allocation
  - Flexibility in construction and maintenance of data structure in run-time
  - Time consuming
- Typing
  - Strong typing can prevent unwanted change in the data through truncation or rounding
  - Weak typing permits expressions of mixed type expressions
    - Calculation at higher storage complexity
    - Unseen and time-consuming instruction for type conversion

The dynamic allocation that is dynamic memory allocation provides flexibility in construction and maintenance of data structure in run time. And in fact, many cases we need to use this kind of data structures. But, obviously, the allocation is time consuming and the same issue can come back that if you really do not have an estimate on the upper bound. You may run out of the heap area allocated to a process at some point in time.

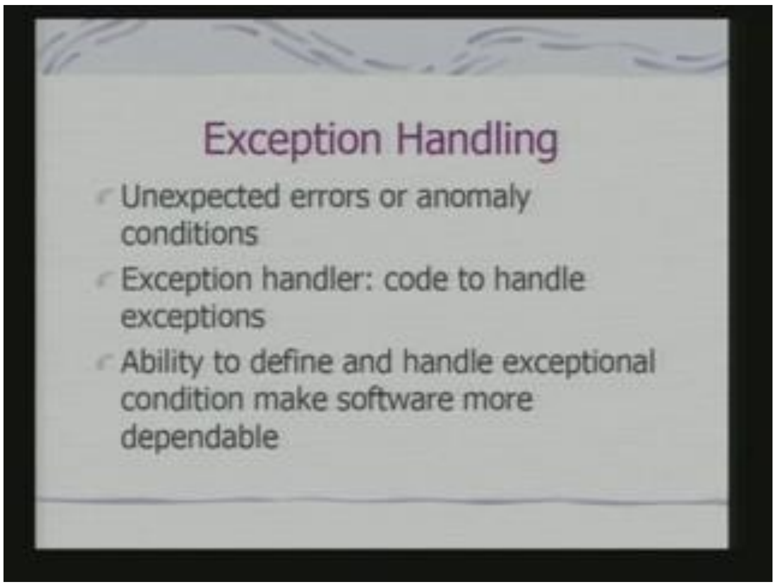
Because, you are expecting your program to do what run continuously? So, this management of the dynamic memory becomes a critical component their typing is an issues. Strong typing can prevent unwanted change in the data through truncation are rounding. But, say a weak typing permits expressions of mixed type ZX for a example as C expression in a C expression you can use integer double float at the same time.

And what happens, if you using the mixed types the whole evaluation takes place in the double form that means, all variables are converted to the double form. And then, depending on the type of the variable to which you are assigning the value the value will be rounded or truncated depending on the policy followed. But these, obviously, gives two issues, one is since you are converting them to a double. You require higher storage requirement during computation.

So, you need to asses that that weather that kind of a memory would be really available or not. And there would be unseen and time consuming instruction for type conversion, which you might not have computed. When you have done the basic assessment source level assessment, only a proper profiling would really give you the kind of timing which may be required.

What is profiling? Profiling gives profiling tools would give you a kind of timing requirements or timing time being consume per proceed here or a block of codes, number of times a proceed you have been called.

(Refer Slide Time: 35:23)



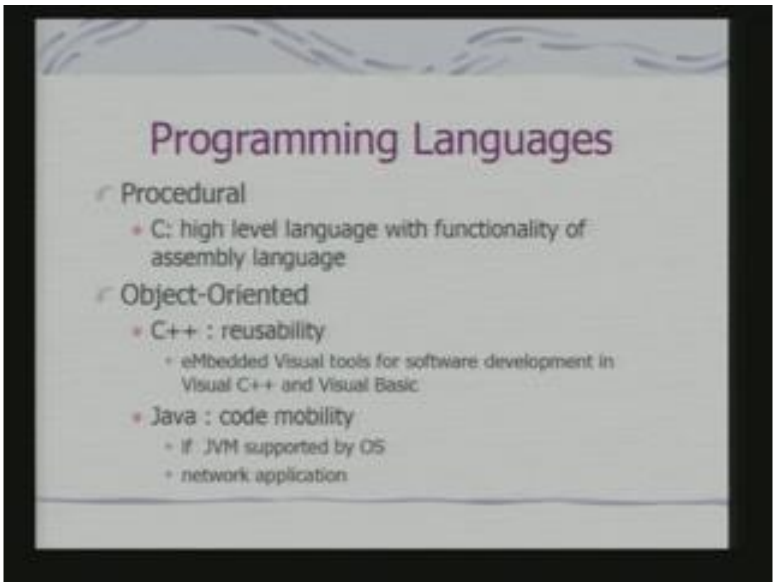
Exception Handling

- Unexpected errors or anomaly conditions
- Exception handler: code to handle exceptions
- Ability to define and handle exceptional condition make software more dependable

A number of times a block code being executed Exception Handling is another important feature in. In fact today, you find that all major languages even ANSI C, since it is targeted for embedded and other applications ANSI C also gives a provision to deal with signals. So, these are exceptions.

There can be variety of exceptions and there should be exception handler codes because, if you cannot handle exceptions the system can go to an undefined state. So, exception handling for embedded System is important and that makes the software what we say dependable.

(Refer Slide Time: 35:54)



Programming Languages

- Procedural
  - C: high level language with functionality of assembly language
- Object-Oriented
  - C++ : reusability
    - eMbedded Visual tools for software development in Visual C++ and Visual Basic
  - Java : code mobility
    - if JVM supported by OS
    - network application

The programming languages are all we familiar with C C++ Java. And now, if you look at adjust picked up three samples. In fact, add a is also being used for embedded Software developments add a 95. C is very widely used with functionality, because it has various constructs which is similar to that of an assembly language concepts.

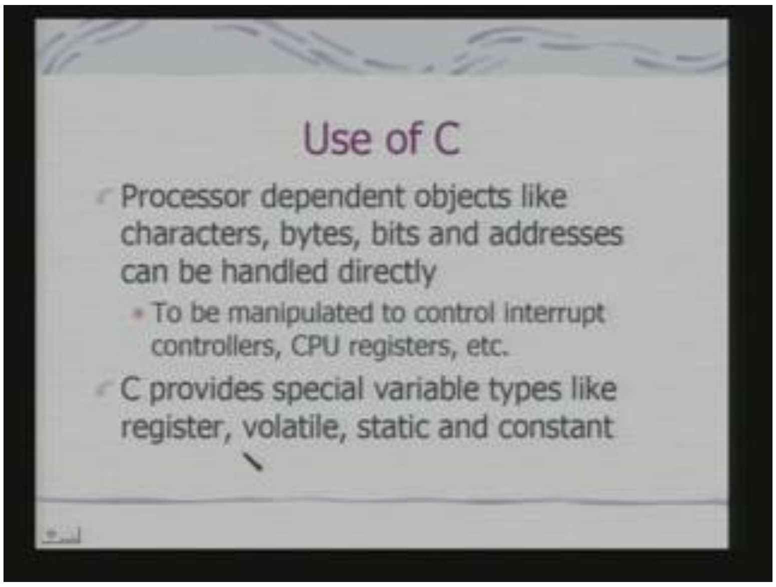
You would like to go for an object oriented language. Because, of its reusability of any of the design components. The reusability comes in terms of composition, because you can build an object by combining the objects, which have been already designed also reusability comes through inheritance.

And java satisfies object orientation, but at the same time the interesting java is because of the code mobility. Because, the java virtual machine is a representation, in fact it is nothing but a kind of a code stimulator an instruction level stimulator of a target machine architecture which can run in a platform independent fashion.

In the sense that you have got for each OS or target architecture, the JVM implementation and when you write a java code and you compile it into a byte code. That byte code is suitable for executing on the java virtual machine. So, if you have with the OS a java virtual machine, which is matched with the hardware requirements as well as that of the OS. You can have same code downloaded on to variety of embedded Systems and executing.

So, if a java gives you a very interesting feature. We typically fine then the software for an embedded System is the close software that. Once you have developed the software and deliver the product software is constant. But, java gives you on the interesting feature is that because of its portability of your executable code. You can get the software from third party sources and enhance the functionality of an embedded System that makes java as a attractive programming vehicle.

(Refer Slide Time: 38:15)



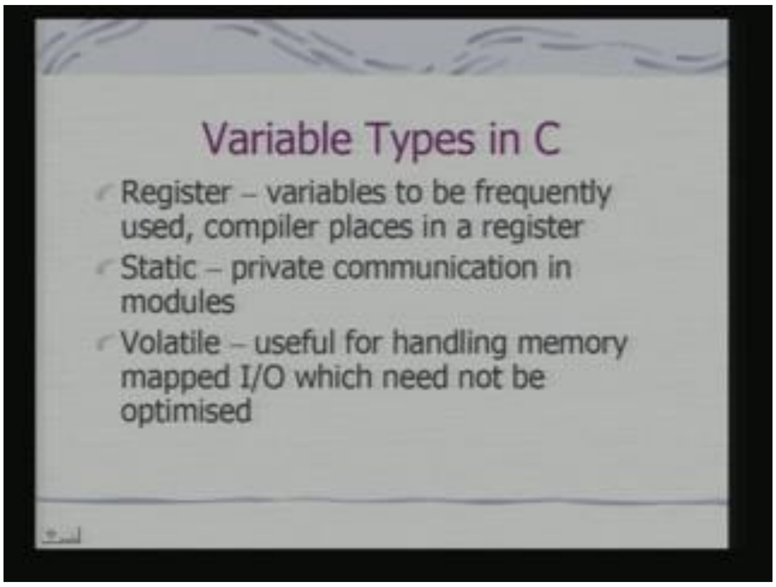
Use of C

- Processor dependent objects like characters, bytes, bits and addresses can be handled directly
  - To be manipulated to control interrupt controllers, CPU registers, etc.
- C provides special variable types like register, volatile, static and constant

So, use of C is; obviously, the reason, what we are already talked about because of various processor dependent definitions of objects like characters, bytes, bits, addresses can be handle directly. You can have a bit vector and using the bit vector. You can do variety of operations even in many of these.

C variance supported by the embedded Processors. That is development environment of embedded Processors; you have got structures defined in terms of bit vectors. So, in a bit vector you can have fills of fixed bit length. And you can do manipulation with the fixed bit length. C also provides special variable types, in fact storage types not really type in terms of the data type. Storage types like register volatile static and constant an in fact these helps in various kind of software developments.

(Refer Slide Time: 39:19)



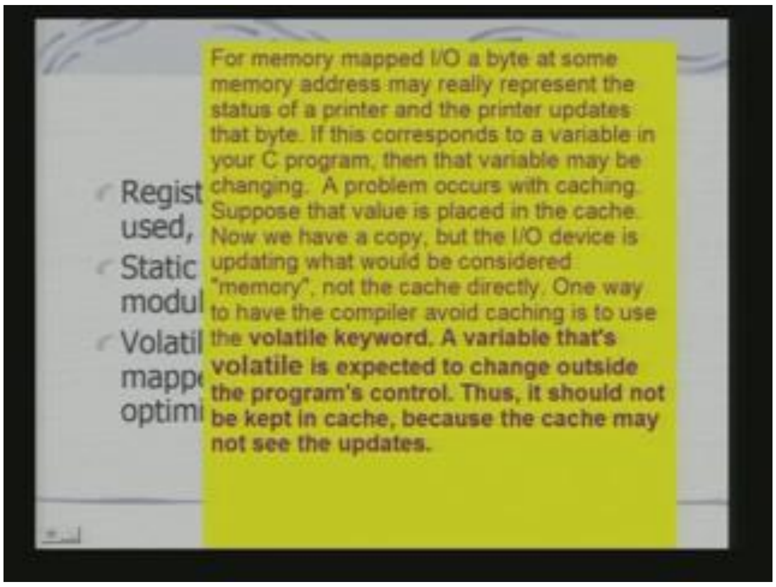
Variable Types in C

- Register – variables to be frequently used, compiler places in a register
- Static – private communication in modules
- Volatile – useful for handling memory mapped I/O which need not be optimised

So, if you look at register, register variables to be frequently used compiler places in a register. So, this is the directive to the compiler to generate optimal code, in fact because compiler is not really able to understand completely your code, the intension of the code usage of the code. So, that way, register if you mark a variable register it is an directive or a suggestion to the compiler to map to the registers of the target processor.

Static is a private communication in modules. In fact, static retains the value beyond in occasions. Because, a local variable in proceed here. We lose the value the moment you are out of proceeding here. But, static retains a value so, it is stored globally and so you can have encapsulation that is in terms of the private value it is not visible outside. But you can carry on the value from invocation to invocation.

(Refer Slide Time: 40:21)



Regist
used,
Static
modul
Volatil
mappe
optimi

For memory mapped I/O a byte at some memory address may really represent the status of a printer and the printer updates that byte. If this corresponds to a variable in your C program, then that variable may be changing. A problem occurs with caching. Suppose that value is placed in the cache. Now we have a copy, but the I/O device is updating what would be considered "memory", not the cache directly. One way to have the compiler avoid caching is to use the **volatile keyword. A variable that's volatile is expected to change outside the program's control. Thus, it should not be kept in cache, because the cache may not see the updates.**

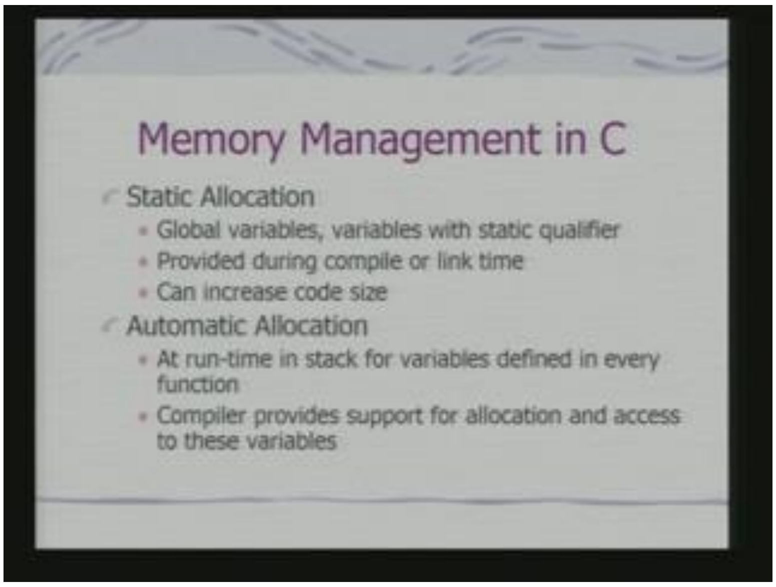
So, volatile is useful for handling memory mapped IO which need not to be optimized. In fact, the basic issue is that if you are using a memory mapped IO, I discussed this point earlier also. Ideally, you should not map those memory locations to cache, because they can be changed by external devices. So, if you look in to this that for memory mapped IO a byte some memory address may really represent the status of a printer and a printer updates that byte.

If this corresponds to a variable in your C program then that variable may be changing. A problem occurs with the cache. Suppose that value place in the cache, now we have a copy, but the IO device is updating what would be consider memory and not the cache directly. So, the value in the cache and that in the memory are not the same.

One way to have the compiler avoid caching is to use volatile keywords. In fact, compiler does not avoid caching. But, compiler generates the code in such a way that the caching would be avoided for those data structures. A variable that is volatile is expected to change outside the programs control does it should not be kept in cache, because the cache may not see the updates.

So, these storage steps specification you can understand its important, because a today if ARM is one of the most popular 32 bit processor ARM supports only memory mapped IO. So, the reference to that kind of an architecture, these consideration becomes important.

(Refer Slide Time: 41:51).



Memory Management in C

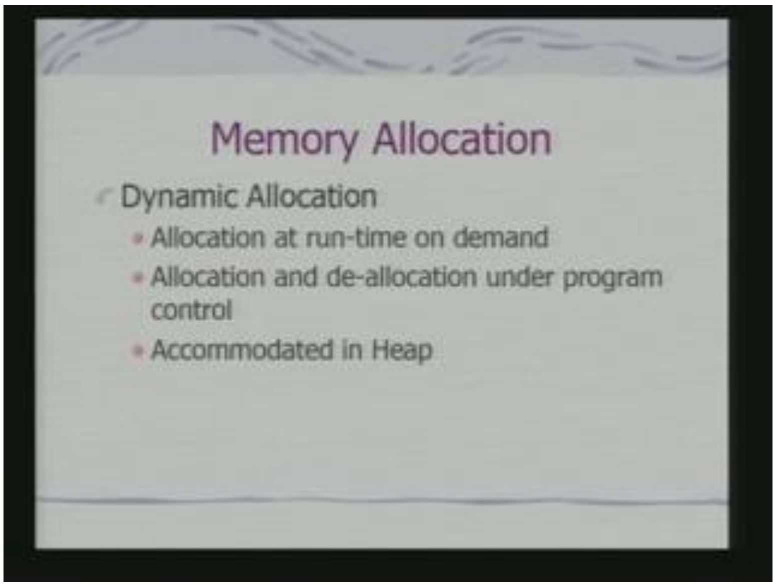
- Static Allocation
  - Global variables, variables with static qualifier
  - Provided during compile or link time
  - Can increase code size
- Automatic Allocation
  - At run-time in stack for variables defined in every function
  - Compiler provides support for allocation and access to these variables

Memory management in C it can; obviously, supports static allocation. The global variables are really allocated during compiler link time an actually can increase the code size, because these memory areas are form part of the code. The other variables the most commonly used variables are automatic allocation. There allocated at run time in stack.

Compiler provides support for allocation and access to these variables. So, these variables really do not form part of the code. So, if you are estimating the code size, if you are trying to estimate the memory requirement by looking at the compile code size that will not be correct.

Because, automatic variable sizes will not be part of that. So, that estimation has to be done separately. And, the dynamic memory allocation, allocation at runtime and demand an allocation and de-allocation and a program control.

(Refer Slide Time: 42:44)

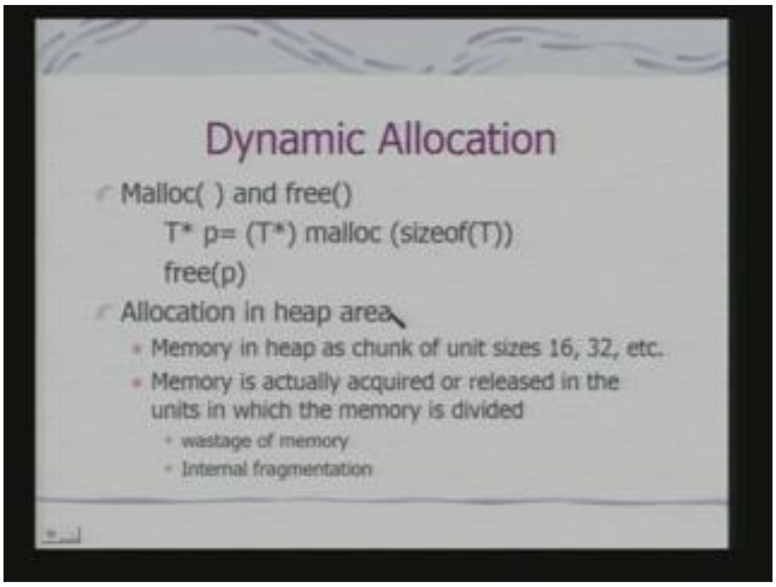


Memory Allocation

- Dynamic Allocation
  - Allocation at run-time on demand
  - Allocation and de-allocation under program control
  - Accommodated in Heap

Typically, you will be using Malloc Calloc for allocation and free for releasing this memory area.

(Refer Slide Time: 42:48)



Dynamic Allocation

- Malloc( ) and free()

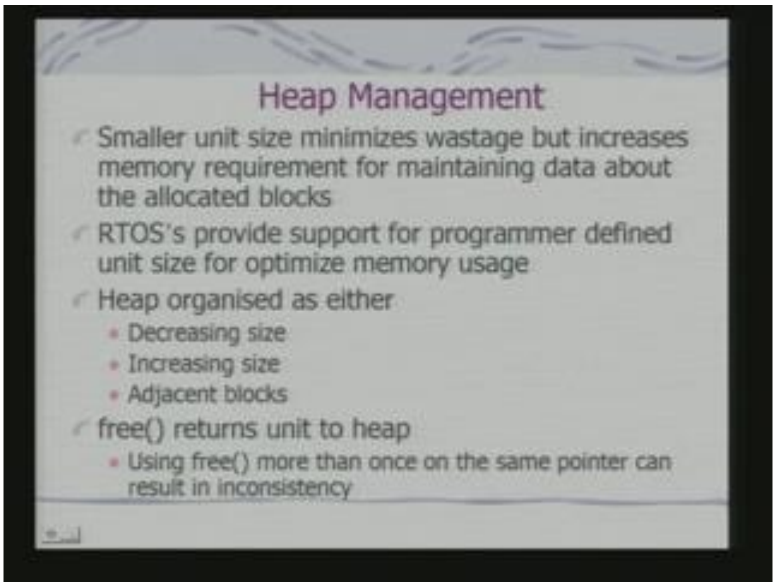
  T* p= (T*) malloc (sizeof(T))

  free(p)
- Allocation in heap area
  - Memory in heap as chunk of unit sizes 16, 32, etc.
  - Memory is actually acquired or released in the units in which the memory is divided
    - wastage of memory
    - Internal fragmentation

This allocated memory is allocation, where in heap area. So, the memory in heap is cons organize typically in chunk of 16 or 32 bits, 32 bytes typically or they may be other sizes as well. So, memory is actually acquired or released in units in which this memory is divided.

Now; obviously, this kindly to and wastage of memory, because if you are allocating data structure or structure of size 4 bytes, then you would be wasting 2 bytes, if you are allocating by units of 16. And that is the wastage because of what you call internal fragmentation, why this a fixed chunk is used, for maintaining the heap as a data structure for keeping track of free memory area in the memory for the processor.

(Refer Slide Time: 43:47)



Heap Management

- Smaller unit size minimizes wastage but increases memory requirement for maintaining data about the allocated blocks
- RTOS's provide support for programmer defined unit size for optimize memory usage
- Heap organised as either
  - Decreasing size
  - Increasing size
  - Adjacent blocks
- free() returns unit to heap
  - Using free() more than once on the same pointer can result in inconsistency

So, the smaller unit size, minimizes; obviously, this wastage because of internal fragmentation. But, increases memory requirement for maintaining data about the allocated blocks. Because, what is really heap? We talk about stack and we talk about heap, what is really heap and stack? You have memory is not getting organized in form of a heap or a stack.

You are keeping a data structure say in case of a stack you are using a stack pointer for accessing and area of a memory following stack discipline. For a heap, you have a memory area corresponding to that memory area you are maintaining a heap to provide access to the free blocks in that memory area. So, what is a heap? Heap is nothing but a tree. And you are using that keeping the data structure.

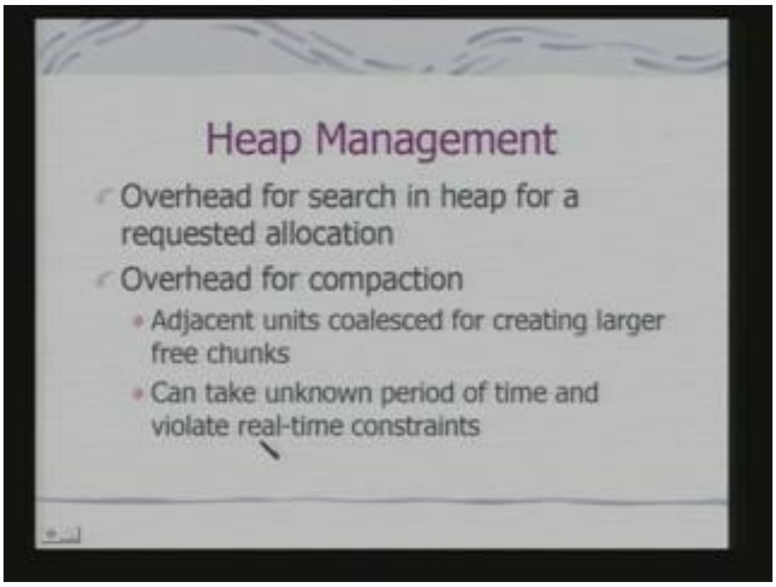
So, if you have smaller unit size, that memory consume by the tree itself become silage. Now, these issues are not very important. If you are looking at a standard computing platform using may be 2 GB of RAM along with the facility for virtual memory. This becomes important, when you are looking at an embedded system which may not, may be using just may be 64 MB of memory or may be just 256 MB of memory. These are all on a higher site.

So, real time OS provides support for programmer defined unit size to optimize memory usage. Because, if you are wasting a memory then that memory. You are wasting a memory at the same time. That is, why you have to use bigger memory? Bigger memory means more energy consumption. So, real time OS is or the OS is which are targeted for embedded Applications.

The gives you a facility and the feature to decide on this unit size for the heap. And heap is organized in terms of decreasing size increasing size or adjacent block. So, typically what it means if it is in terms of a decreasing size the root node of the heap would be pointing to the largest feed lock available in the key. Free returns unit to heap.

Now, when you are retaining the free the type of the pointer, if you are using C becomes important and critical why, because on the basis of the type of the pointer the runtime environment knows, what is the exact size of the memory which is to be freed and put back to heap and if you are using a free on the same variable more than once, what can happen? Your freedom memory area then that memory area might have been allocated for some other data structure. Then again you are freeing it, so that memory area going back to the heap and that be give rise to an inconsistency and a fault which may surface later on.

(Refer Slide Time: 46:48)

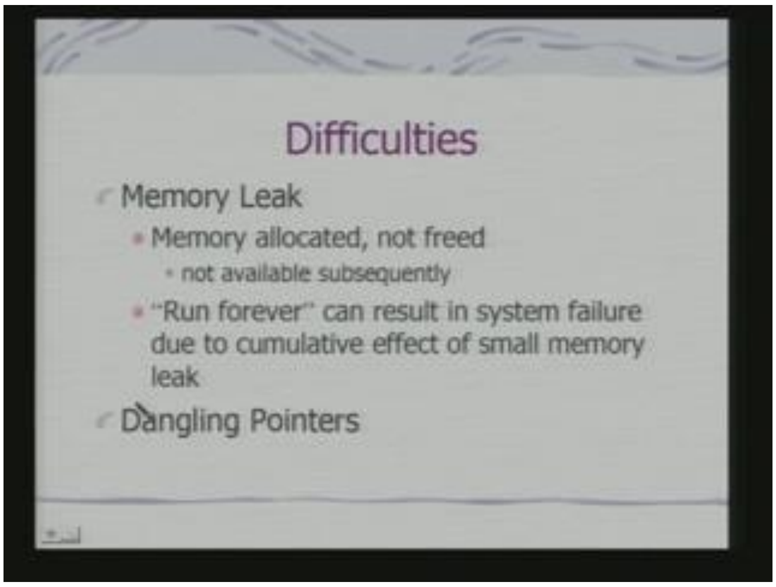


Heap Management

- Overhead for search in heap for a requested allocation
- Overhead for compaction
  - Adjacent units coalesced for creating larger free chunks
  - Can take unknown period of time and violate real-time constraints

So, overhead in a heap management, so what you have you pay for two kinds of overhead? Overhead for search in a heap for requested allocation, Overhead for compaction. Now, adjacent units coalesced for creating larger free chunks and can take unknown period of time and violate real time constraints. Because, when you are compact the different blocks together to get and create larger memory chunks or memory areas it may be really time consuming.

(Refer Slide Time: 47:14)



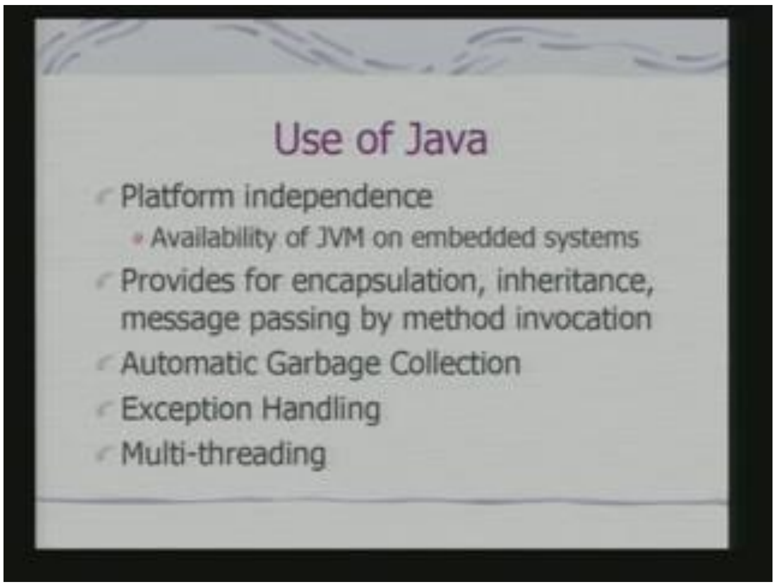
Difficulties

- Memory Leak
  - Memory allocated, not freed
    - not available subsequently
  - "Run forever" can result in system failure due to cumulative effect of small memory leak
- Dangling Pointers

A fundamental problem with dynamic allocation, which is of importance for embedded Systems, is call memory leak. Memory is allocated, but not properly freed. So, this memory is not available subsequently. And since your program is running for ever, then this error can surface subsequent get some point in time. So, what would it been a malloc is returning a null pointer. So, this memory leak is a very important problem in the context of dynamic memory allocations.

The other problem is that of Dangling pointers, what is mean by Dangling Pointers, pointers referring to memory locations which have been freed. So, it may be use in the context of other data structure. So, if you are using Dangling Pointers in may corrupt data structure and may create problems. So, a pretty discipline is to be followed while using dynamic memory.

(Refer Slide Time: 48:30)



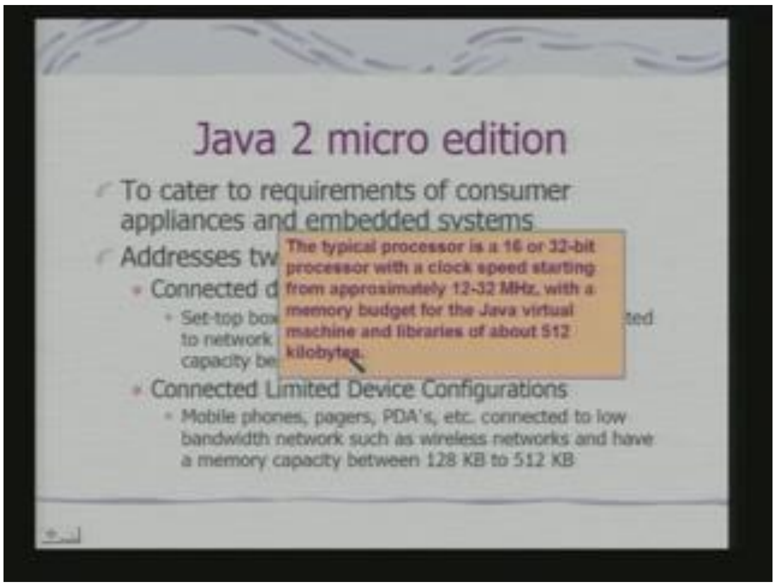
Use of Java

- Platform independence
  - Availability of JVM on embedded systems
- Provides for encapsulation, inheritance, message passing by method invocation
- Automatic Garbage Collection
- Exception Handling
- Multi-threading

The other language C object oriented language is java and I told you the java is interesting, because of that portable feature it has got it supports various features of object orientation. And a very interesting feature is that of automatic garbage collection because, it is an object oriented language. The object gets allocated in the heap area.

And these objects have to be freed if we have last reference to that object. So, it has gotten automatic garbage collection. This garbage collection task gets scheduled automatically for removing this objects and that is freeing this objects. So, what does that imply? It implies that your code gets suspended and garbage collector gets executed depending on the time when the garbage collector is schedule. It has got exception handling mechanism it also supports multithreading.

(Refer Slide Time: 49:09)



Java 2 micro edition

- To cater to requirements of consumer appliances and embedded systems
- Addresses tw
  - Connected d
    - Set-top box ted to network capacity be
  - Connected Limited Device Configurations
    - Mobile phones, pagers, PDA's, etc. connected to low bandwidth network such as wireless networks and have a memory capacity between 128 KB to 512 KB

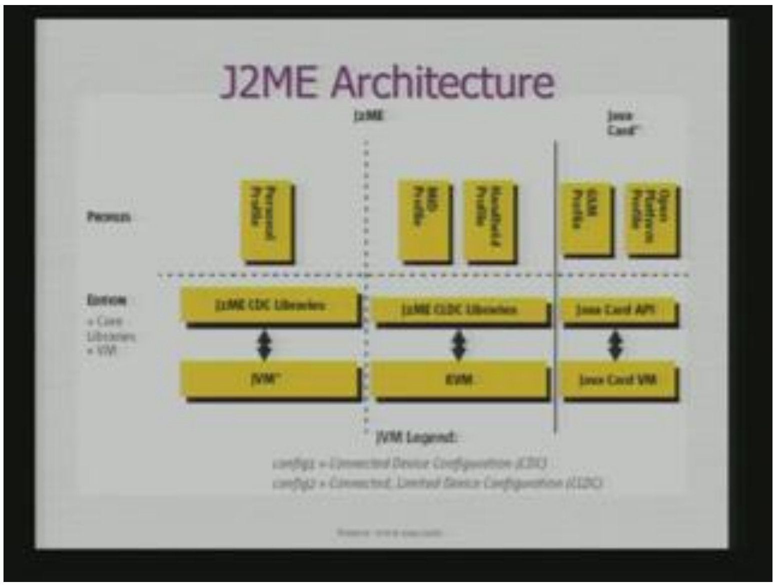
The typical processor is a 16 or 32-bit processor with a clock speed starting from approximately 12-32 MHz, with a memory budget for the Java virtual machine and libraries of about 512 kilobytes.

Now, some version of java that is one version of java which is called Java 2 micro edition which is targeted for consumer appliances and embedded Systems, in fact your java compliant mobile phones PDS. They are all they do support J2ME versions are not really fully fledged java. And they address J2ME two types of devices, what you call connected device configuration, typically set-top box internet TV etc, where you have memory capacity between 2 to 16 MB.

And other disconnected limited device configurations, typically mobile phones pagers PDA's which are connected to low bandwidth network and having capacity between 128 KB to about 512 KB. In fact, this basic assumption is that your processor is a 16 or 32 bit processor with the clock speed starting from approximately 12-32 MHz with the memory budget for java virtual machine and libraries of about 512 kilobytes.

So, this java virtual machine is nothing but software which is involved in execution of your code and that size has to be restricted. So, this is typically 512 kilobytes. In fact, in context of J2ME the java virtual machine is called KVM or kilobyte virtual machine.

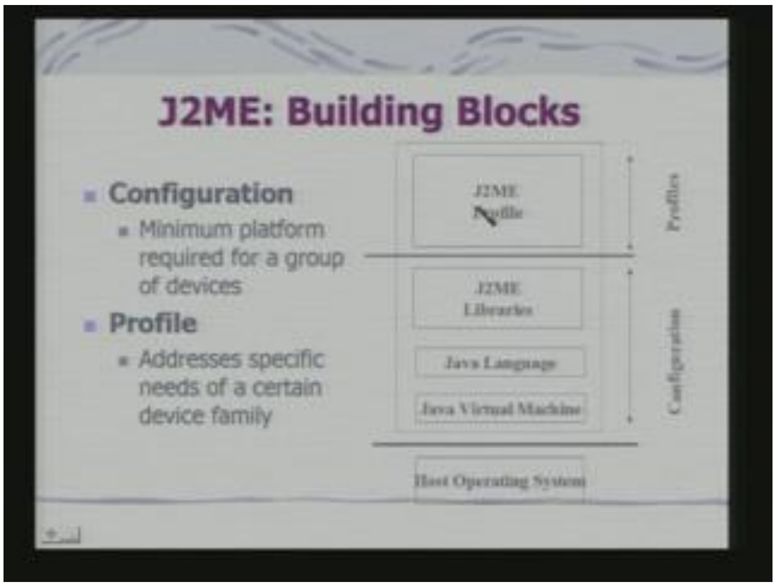
(Refer Slide Time: 50:38)



Architecturally it something like this that you have got in a CLDC. You have got the KVM then on top of that the J2ME CLDC libraries that constitutes, the core runtime environment. Then on top of that you have got profiles. Profiles are device specific in many cases. You may have a handheld profile and these together gives you the runtime environment.

Now, this is for your CDC which is largest. So, you have the JVM java virtual machine which is similar in capability is that of your standard edition. And you have got the profile on top of the smaller than that is java card which is targeted for smart card applications. You can have java code running on smart cards. And smart cards would have still smaller memory and show your JVM needs to be have smaller foot print.

(Refer Slide Time: 51:33)



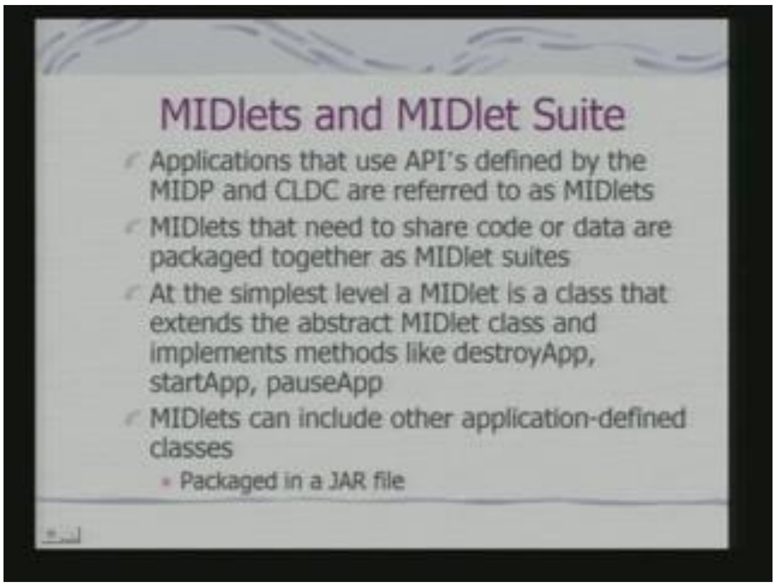
J2ME: Building Blocks

- **Configuration**
  - Minimum platform required for a group of devices
- **Profile**
  - Addresses specific needs of a certain device family

J2ME Profile
J2ME Libraries
Java Language
Java Virtual Machine
Host Operating System
Profiles
Configuration

The J2ME, what I heard already talked about the basic building blocks is your java virtual machine, the libraries and profile. So, configuration part is the minimum platform required for a group of devices and profile addresses specific needs for a certain device family may be a mobile phone. So, it will have an MIDP profile as specific profile sitting on top of this configuration. And this complete thing is your the runtime environment for java on your devices and.

(Refer Slide Time: 52:04).



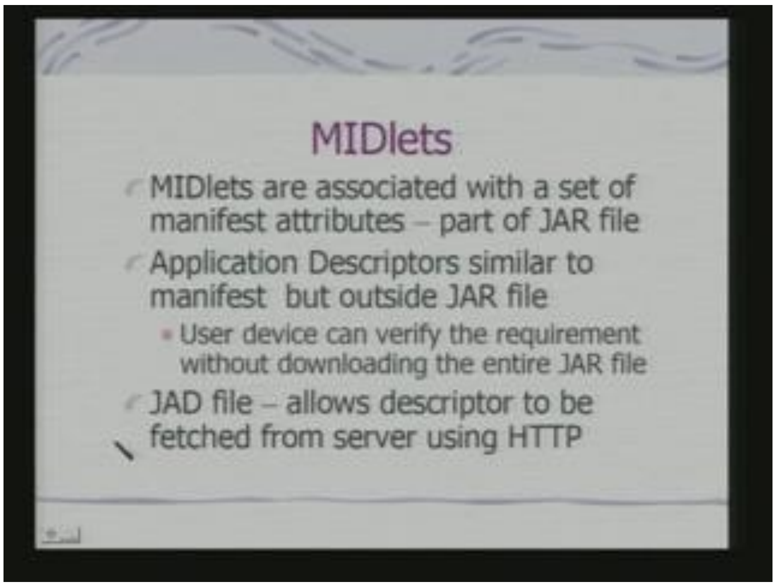
MIDlets and MIDlet Suite

- Applications that use API's defined by the MIDP and CLDC are referred to as MIDlets
- MIDlets that need to share code or data are packaged together as MIDlet suites
- At the simplest level a MIDlet is a class that extends the abstract MIDlet class and implements methods like destroyApp, startApp, pauseApp
- MIDlets can include other application-defined classes
  - Packaged in a JAR file

On these devices, what you run, you do not run your applets or servlets, but what you run, is called MIDlets. Applications show these MIDlets are nothing but a set of java code, compile java code along with the data everything put together. So, what we say at the simplest level a MIDlet is a class. That extends abstract MIDlet class an implement methods like destroy application start application and pause application which is very very similar conceptually that often applet.

But, what is interesting here is, that all the code that is all classes that is involve in an application as well as the data everything is put together and put together into a JAR file. And these JAR file is actually, what is to be loaded, finally onto the target device for execution of the java application.

(Refer Slide Time: 53:05)



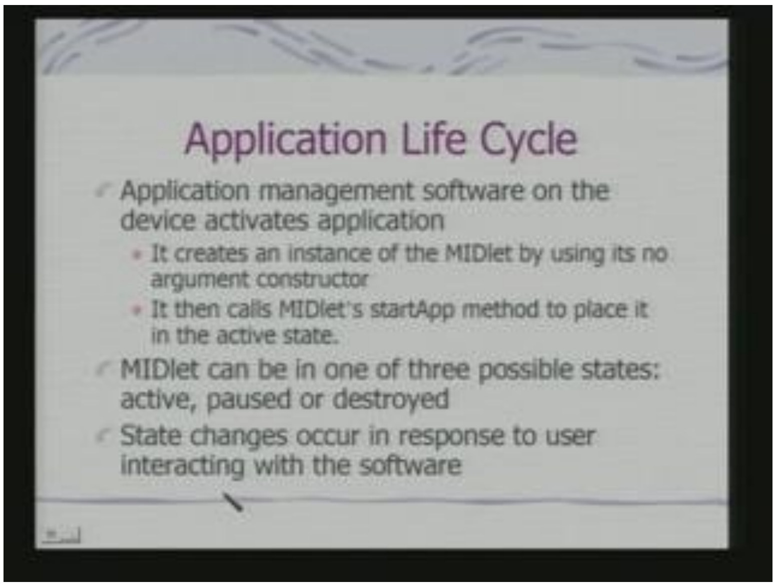
MIDlets

- MIDlets are associated with a set of manifest attributes – part of JAR file
- Application Descriptors similar to manifest but outside JAR file
  - User device can verify the requirement without downloading the entire JAR file
- JAD file – allows descriptor to be fetched from server using HTTP

So, MIDlets are associated with the set of attributes manifest attributes which are part of JAR file. Because, looking at manifest attributes the system can know exactly, what it can do and what it is what is required to be done to support this MIDlet. And application descriptors similar to manifest, but outside JAR file. These are application descriptor that is another set of descriptors which are available.

And user device can verify the requirement without downloading the entire JAR file. So, it can find out, what it does, then only it may decide to download the JAR file. And there is this special files for JAD file allows descriptor to be fetched from server using typically HTTP protocol. So, if you are using a internet connection you can do that.

(Refer Slide Time: 53:59)



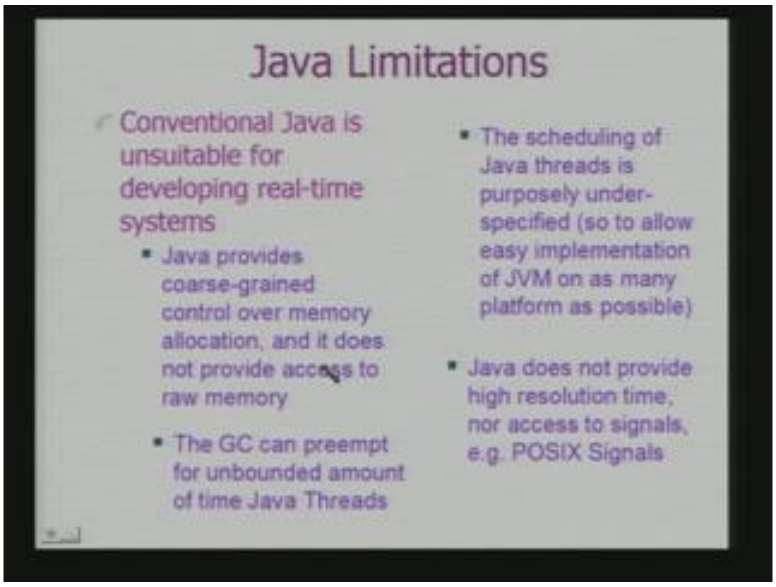
Application Life Cycle

- Application management software on the device activates application
  - It creates an instance of the MIDlet by using its no argument constructor
  - It then calls MIDlet's startApp method to place it in the active state.
- MIDlet can be in one of three possible states: active, paused or destroyed
- State changes occur in response to user interacting with the software

So, a MIDlet typically is invoked by the application level software and instance is created; that means, and then it calls MIDlet start application method to place it in what is called an active state. The MIDlet can be an active state paused or destroyed; that means, when it is doing something it is inactive state. If it is waiting for an input it goes into a power state else it gets destroyed.

And state changes occur in response to user interacting with the software. In fact, today there is variety of these kind of third party. Java code which you can load onto your mobile devices and have added functionality. In fact, this is the basic framework for this kind of third party code to go into your embedded device, but java if you look at from a real time applications pointer view.

(Refer Slide Time: 54:46)



Java Limitations

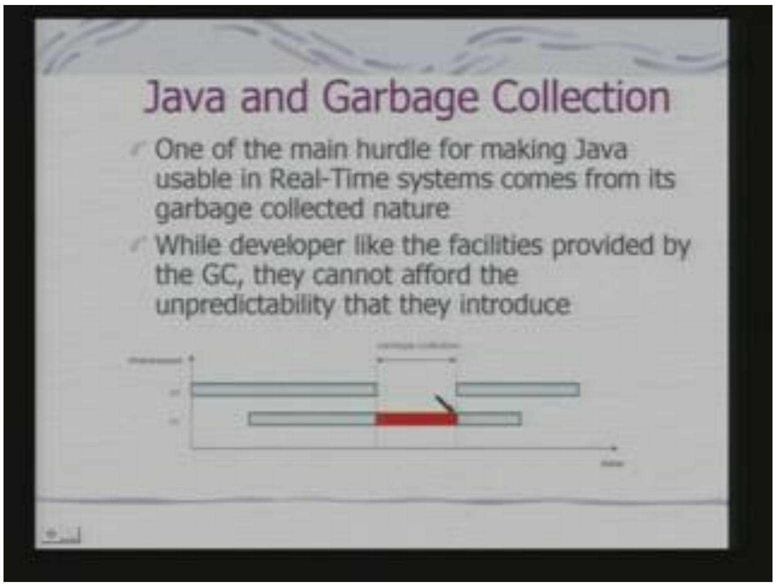
- Conventional Java is unsuitable for developing real-time systems
  - Java provides coarse-grained control over memory allocation, and it does not provide access to raw memory
  - The GC can preempt for unbounded amount of time Java Threads
  - The scheduling of Java threads is purposely under-specified (so to allow easy implementation of JVM on as many platform as possible)
  - Java does not provide high resolution time, nor access to signals, e.g. POSIX Signals

Java is unsuitable for developing real time applications. Because, fundamentally it does not provide you control our low level memory just like C. The GC that this garbage collector can preempt unbounded amount of time Java Threads. If it is preempt, then you deadline can be missed this is the fundamental problem for developing real time applications.

The scheduling of java threats although it can support multi threading the scheduling mechanism is not really completely specified; obviously, to enable implementation across variety of OS platforms. Java does not provide high resolution time, what is that mean, because you were talking about a real time systems. Then, how do you specify your time or timing constraints. You have the mechanism to deal with time.

And POSIX is standards in terms of signals and other interfaces for your real time developments. This java is not strictly confirming to that. So, classically java is not really suitable as a real time programming language.

(Refer Slide Time: 55:52)

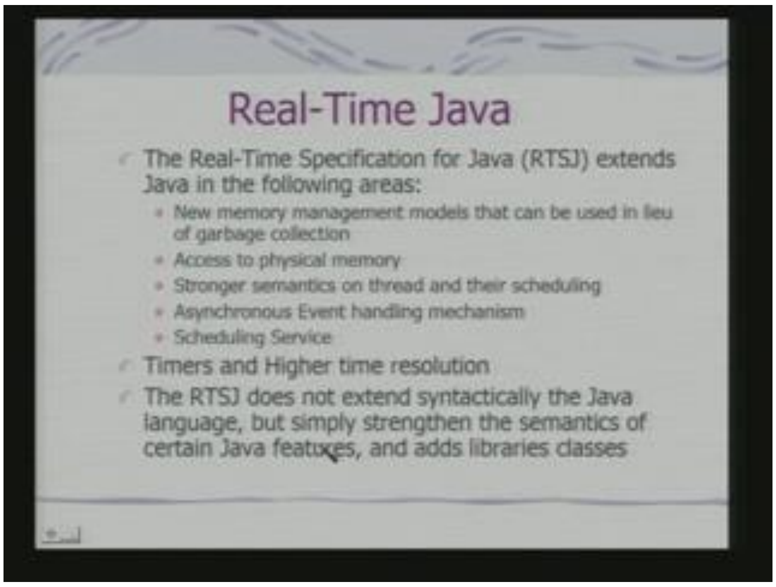


Java and Garbage Collection

- One of the main hurdle for making Java usable in Real-Time systems comes from its garbage collected nature
- While developer like the facilities provided by the GC, they cannot afford the unpredictability that they introduce

Now, there is a if you look in to it this basic problem of GC that I was trying to talk about. This may be your code which is getting executed and garbage collector comes in here and interrupts, so your deadline happy mist.

(Refer Slide Time: 56:02)



Real-Time Java

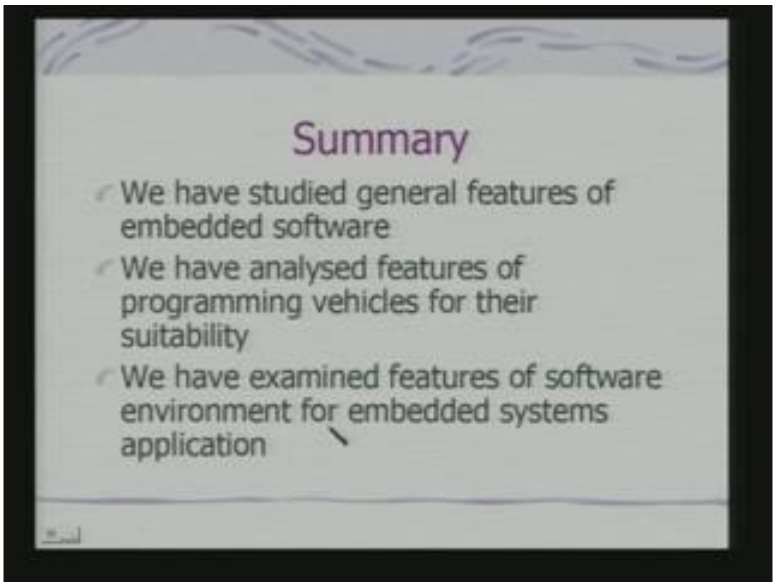
- The Real-Time Specification for Java (RTSJ) extends Java in the following areas:
  - New memory management models that can be used in lieu of garbage collection
  - Access to physical memory
  - Stronger semantics on thread and their scheduling
  - Asynchronous Event handling mechanism
  - Scheduling Service
- Timers and Higher time resolution
- The RTSJ does not extend syntactically the Java language, but simply strengthen the semantics of certain Java features, and adds libraries classes

So, that is a suggestion for real time java. And, why we are looking at real time java, because there are also various real time programming languages. And real time programming languages one example that is, why we are looking at with reference to java, because they are expected to have some special features to facilitate real time implementations.

With reference to java, what you find, that they it has got a new memory management models that can be used in lieu of garbage collection. So, the garbage collector need not preempt a process for a non bounded time. It requires access to physical memory. You have stronger semantics on thread and their scheduling; that means, this threads and how they can be schedule. Because, scheduling scheme actually tell you whether the real time constraint can be real image or not.

Are you should have a mechanism for asynchronous even handling, what is an asynchronous event? If an interrupt its being generated by a device. That is an asynchronous event. And there should be a mechanism to handle that asynchronous events and you have the timer's ability to deal directly with the timers. These are the features which are typically expected of a real time programming language and real time java do support this features and its under basically acceptance at various levels of standardization schemes.

(Refer Slide Time: 57:25)



Summary

- We have studied general features of embedded software
- We have analysed features of programming vehicles for their suitability
- We have examined features of software environment for embedded systems application

So, today therefore, what we have studied? A general features of embedded software. We have analyzed features of a programming vehicle for this suitability assembly language as well as higher level language. An also examine features of software environment for embedded Systems application.