Embedded Systems

By
Chandra Sekhar G.
Assistant Professor
Electronics and Communication Engineering Dept.
Bharati Vidyapeeth's College of Engineering, New Delhi

UNIT - III

- Embedded Software
- Concept of Real Time Systems
- Software Quality Measurement
- Compilers for Embedded System.

Note: Prerequisites: Clear Conceptual knowledge in Microprocessors, Microcontrollers and their Functional Units.
This Notes is prepared mostly using NPTEL Videos of Embedded Systems, IIT Delhi. Some concepts are taken from Internet.
This notes is prepared to complete the syllabus in the given number of lectures and in the exam point of view. For more and detailed theory go through the text book Computers as components: Principles of Embedded Computing System Design, Wayne Wolf, Morgan Kaufman Publication, 2000.

Software for Embedded Systems

Now, we will study the software for embedded Systems. And in particular, the software environment and the practices to be followed for developing software's for Embedded Applications.

Embedded software****

- Program that controls an embedded system. Program is a piece of code written to perform a task.
- Written specifically for the particular hardware that it runs on.
- Loaded in the microcontroller which then takes care of all the operations that are running.
- Tools need for developing an embedded software include editor, compiler, assembler and debugger.
- Software interacts with physical world
  - Therefore takes time.
  - Consumes power.
  - It does not terminate
- Embedded software examples
  - Hidden in watches, VCR's, cellular phones, toasters etc.
  - Guides missiles, Controls satellites
  - Used in medical instruments.

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 and also used in medical instruments. So obviously, this application scenario makes embedded software special and different from that of standard desktop software.

We have varying requirements and different characteristics of hardware platform for embedded systems.
In a way, the software for embedded systems interacts with physical world with sensors and actuators. 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 also 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 them to terminate, with an infinite number of steps these programs typically do not terminate.

Software Complexity

- Embedded software can be very simple, run on an 8-bit microcontroller with just a few kilobytes of memory, Ex: Controlling lighting in homes.
- Complex embedded software - in aircraft avionics systems, in missile guidance systems, in navigation systems
- Depends on varying requirements and hardware platforms

Requirements of software for embedded System

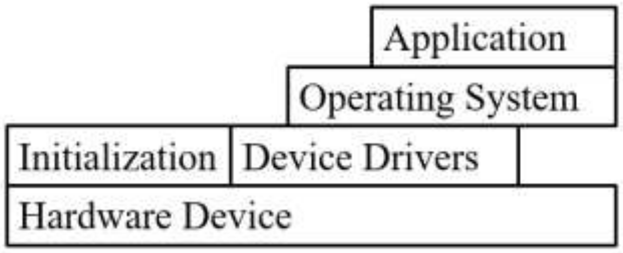
- **Reliability:** Should be Reliable, because human intervention is not possible for error handling. They are expected to run without human intervention to long period.
- **Cost:** They should be cost effective.
- Power : Low power consumption,
- **Memory**: They should make efficient use of memory, because the memory cannot be unbounded.
- **Performance requirement:** The most important performance requirement is that of timeliness.

Embedded software is similar to firmware, as they usually serve the same function. The latter, however, is a special type of embedded software that is written in non-volatile memory (such as ROM or EPROM), which cannot easily be modified — hence the name "firm" — and is used primarily for running or booting up the device. In contrast, embedded software is used for the overall operation of the device.

The main difference between embedded software and application software is that the former is usually tied to a specific device, serving as the OS itself, with restrictions tied to that device's specifications, so updates and additions are strictly controlled, whereas application software provides the functionality in a computer and runs on top of an actual full OS, so it has fewer restrictions in terms of resources.

Embedded System Software

An embedded system needs software to drive it. Figure below shows four typical software components required to control an embedded device. Each software component in the stack uses a higher level of abstraction to separate the code from the hardware device.



The initialization code is the first code executed on the board and is specific to a particular target or group of targets. It sets up the minimum parts of the board before handing control over to the operating system. The operating system provides an infrastructure to control applications and manage hardware system resources. Many embedded systems do not require a full operating system but merely a simple task scheduler that is either event or poll driven. The device drivers are the third component shown in Figure 1.4. They provide a consistent software interface to the peripherals on the hardware device. Finally, an application performs one of the tasks required for a device. For example, a mobile phone might have a diary application. There may be multiple applications running on the same device, controlled by the operating system. The software components can run from ROM or RAM. ROM code that is fixed on the device (for example, the initialization code) is called *firmware*.

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. So, this timing conditions and constraints are to be followed.
  - 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.

Concurrency

- Embedded systems interact with the physical world where multiple things happen at once.
- Embedded Systems need to react stimulus from a network of and variety of sensors and retain control over actuators. And so, the processing should be concurrent.

Liveliness – Program must not terminate or block waiting for events that will never occur

- The whole system would basically fail.
- Watchdog timer - ensure liveliness.

Heterogeneity – Software must be able to hand the Different computational styles (access phone book, handle call) and implement technologies.

- Interact with events occurring in the external world may be of different types.
- They may be regular, they may be irregular
- Irregularly in time (alarms, user commands, sensor triggers, etc.)
- Regularly in time (sampled sensors data and actuator control signals)

Reactivity –

- The reactivity means reacting to the external environment.
- React continuously to the environment at the speed of the environment. If it cannot react at the speed of the environment then the whole sequence of the plans gets diminished.

- And since, it is reacting to the external environment; obviously, there would be bounds, bounds in the response time.
- 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.

Interfaces –

- Components combined according to the interface based upon static behavior
- 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?

Real – Time Systems****

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, is called the response time of the system. 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 leads 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.

Classification of Real time systems:
This real time systems can be categorized further into soft real time systems and hard real time systems as well as firm 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 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 **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. 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 can happen. It may actually destroy the good scrapes instead of the weed. So, the effect can be really catastrophic.

Software Quality Measurement*****

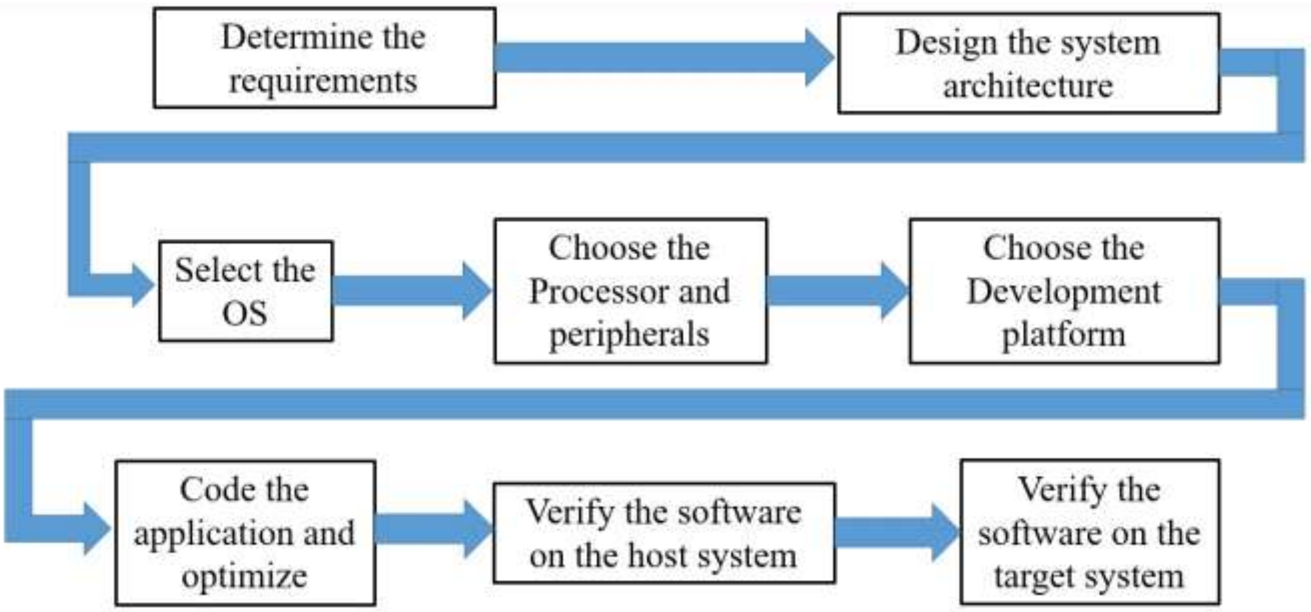
Now, how do you measure quality of a software?

- **Dynamic efficiency:** Number of CPU cycles required for execution of the software.
- **Static efficiency**: Number of bytes required i.e. the RAM size the ROM size.
  - The RAM would accommodate the Global 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.
- **Correctness:** Logical correctness as well as temporal correctness and these two a primarily for the purpose of maintaining and reusing the software.
- **Easy to Understand**
- **Easy to change:** Flexibly and Maintainability

So, Dynamic efficiency, Static efficiency and power consumption – these three are very important parameters to measure quality of a software targeted for embedded Application.

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.

Embedded Software Design Process


Determine the requirements
Design the system architecture
Select the OS
Choose the Processor and peripherals
Choose the Development platform
Code the application and optimize
Verify the software on the host system
Verify the software on the target system

The different steps in an embedded software design process include the following.

Determine the Requirements

Functional and nonfunctional

- Multimode or multifunctional system
- Size, cost, weight, etc.

Choosing the hardware components

- Application specific hardware
- External interfaces
- Input and output devices

Design the System Architecture

The architecture of an embedded system depends on,

- Whether the system is real time
- Whether operating system needs to be embedded

- Cost, size, power consumption, etc.

Select the OS

If operating system we can select,

- Real-time operating systems like RTLinux, VX works, pSOS, QNX, VRTX, etc.
- Nonreal operating systems like Windows CE, embedded Windows XP, etc.

Choose the Processor

The following processors can be used in the development of an embedded system

- Microprocessors-8085, 8086, Pentium
- Microcontrollers-PIC, MCS-51, MSP-430, AVR
- Digital signal processor- dsPIC, Sharp, Blackfin,Tigersharc

Choose the Development Platform

The development platforms of an embedded systems include the following

- The hardware platform
- The programming language
- The operating system
- The development tools

Code the Applications and Optimize

The coding of an embedded system can be done by using the following programming languages.

- Assembly language
- C language
- Object oriented languages like C++, Java, etc.
- Optimizing the code

Verify the Software on the Host System

- Compile and assemble the source code into object file
- Use a simulator to simulate the working of the system

Verify the Software on the Target System

- Download the program using a programmer device
- Use an Emulator or on chip debugging tools to verify the software

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.

Choice of Programming languages

- **Assembly language**
  - Processor dependent
  - Efficient
  - Difficult to maintain and write large programs
- **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 and 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.

Programmer's view:

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 and/or controls (drives) 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.

Build Process

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 i.e. Embedded software development tools can rarely 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.

Compiling: Parsing, Semantic Analysis and Object Code Generation

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 for 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 away a high level language code in to the target machine instructions.
There has to be some kind of the core transformations if we want optimal exploit. So, the register set of the processors. So the aspect of compiler design and implementation has 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.

Object Files

Now, 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, object file which is generated in the host system is not definitely the executable image.

- **In standard formats**
  - COFF - Common Object File Format
  - ELF - Extended Linker Formats.

Why they come in standard form? Now, these object files can be used with the linkers as the software tools, which are targeted for the specific platforms for generating the executable image.

- **Object files typically contains**
  - Code blocks which are text block, which we refer to a text block
  - And 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.

Linking for Embedded System

Next comes an important processes 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 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.

Start up Code:**** A special object file 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 the different modules, you have to resolve the address references with respect to the relative addresses.

Compilers for Embedded Systems****

Compilers:

- Translates high level language program to machine instructions of target processors
- Compilers need to exploit characteristics of the target processor

Now, we will look at the compilers, which are used for generation of the machine code targeted for embedded systems and the processors which have been used in embedded systems. In fact, the compiler technology in a way is part of the design tools that we used for designing embedded systems. Because, if we have to exploit the characteristics of the processor you have to generate optimize code.

And, compilers provide you the tool to generate optimized code, exploiting features of the target architecture. We shall look at some of the issues, that compilers are concerned with in the context of embedded systems and processors, which are used for embedded systems. So, what are really compilers just recapitulate. Compilers translates high level language program to machine instructions of target processors. And compilers; obviously, would need to exploit characteristics of the target processor, if it has to generate optimized code.

Compilation:

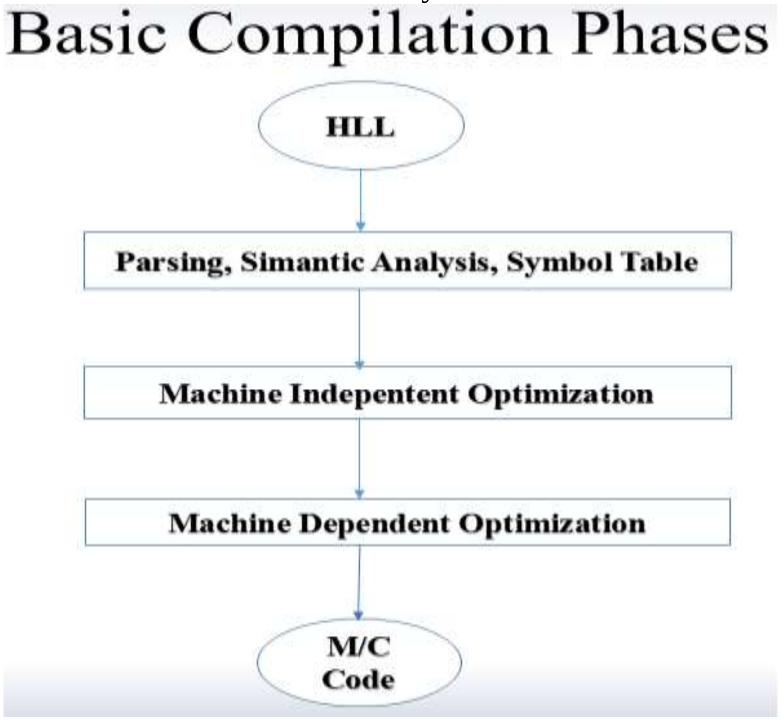
- Compilation strategy(wirth): Compilation = translation + optimization
- Compiler determines quality of code:
  - Use of CPU resources;
  - Memory access scheduling;
  - Code size;

Compilation, in fact in a way can be considered as combination of two phases. One is your translation, the other part is optimization. In fact, we are actually more concerned with optimization. Because, the techniques which go into the translation phase in a way or generic and not processor specific there may be language specific. But, they are not really processors specific. But, when we look at optimization and the optimization part also includes code generation and using it includes optimized code generation.

Then, this part is strictly processor specific. In fact, compiler determines the quality of code optimal usage of CPU resources memory access scheduling as well as that of code size. You can understand that all these parameters are determinant for various functional features of the system. Because, if I can use CPU resources optimally I am not accessing memory. Because, one of the most important CPU resources or registers if i am using registers optimally, I can minimize on memory access.

The moment I minimize on memory access my execution time improves at the same time energy consumption decreases. If my CPU consists of multiple functional units, if I can use this multiple functional units optimally then; obviously, the time taken would reduce. See if I consider a very large instruction set computer which has got multiple functional units. Then, the compiler would figure out the appropriate sequence of instructions, so

that these functional units can be optimally used. It should not be that one of the functional units is being used and others are not being exploited. The similar issues related to the memory access scheduling. So, if I can access memory in an optimal fashion both my time and energy is optimized. This is related to the code size. If you remember the example we have considered that if we are using arm processor and for a particular computation if the 16 bit instruction is good enough.
Then, the compiler can detect that and generate may be automatically 16 bit instructions and you have seen that if it is generating 16 bit instructions what has it been the memory required will be less. And; obviously, I can have the code put into a smaller size memory in the embedded system. So therefore, compiler becomes a key resource for designing efficient software for embedded systems.


Basic Compilation Phases
HLL
Parsing, Simantic Analysis, Symbol Table
Machine Indepentent Optimization
Machine Dependent Optimization
M/C Code

Basic compilation phases, this is again brief recapitulation, our input is the high level language program. Then, we have these phases, the first phase we have termed as parsing, which will also include lexical analysis that should be followed by the semantic analysis, which is language specific. And in the process, we shall also generate the symbol table that is definition of variables constants functions as well as their type information. Because, that type information is also used during semantic analysis. Then, we can do machine independent optimizations which are not specific to the target architecture. In fact, in a way this part of the compiler can be built independent of the target architecture. Next part is machine dependent optimization; that means, it will exploit the target architecture features. Primarily, it will exploit the instruction sets, the nature of the instruction sets.

And accordingly, it will optimize the generated code. Optimizing generated code would actually mean selection of appropriate instructions. So, that it may be, your time required for a computation would be less. Because, same computation can be done through combination of different instructions and this combinations may require different number of cycles to execute. So, when I am optimizing I may like to choose a set of instructions which minimizes the number of cycles. And this would lead to time optimization.

The other kind of optimizations could be in terms of memory accesses and register usages depending on the number of registers available. As well as, when you have multiple functional units, the scheduling of instructions also becomes important. It is not necessary that you need to execute instructions in the order in which they are generated from the high level code. If there are no dependencies, in fact, dependencies can be model using data flow graph if such dependencies can be detected and if we can re adding instructions. So, that these dependencies are removed then we can have also a better utilization of CPU resources like registers as well as functional units. So, these are the different kinds of machine dependent optimizations and you produce the machine code.

Now, the question could be that if I am writing program in assembly language. Knowing the instruction set of the processors, you can possibly yourself generate an optimized code. But; obviously, you can realize then what you are sacrificing is the portability of the code. If the architecture is changed from one version to another version then again you need to recode the program using the features of the modified architecture. But, if you are doing it in a high level language then your entire set of applications can remain unchanged. If you can really design and implement an efficient compiler say for example, if I am switching form arm seven to arm nine architecturally which are different. If I already have an application coded in C and if I can use an appropriate compiler to translate your application in an optimize fashion for target arm 9 processor. Then, the course that goes into recoding of the application disappears. Because, compilers in a way can be used for many other applications as well the basic principle of bringing down the cost of the design by reusing tools and modules is equally valid in this context as well.

Why Compilers are an issue?

This point we have already touched upon. The processor architecture for embedded systems exhibit special features. And, a high level of optimization is more important than high compilation speed, because your code is expected to be compiled once and executed 100 of times. So, you are ready to pay for more complex compiler and more time in compilation.
Compilers potentially help to meet and prove real time constraints. Because, if we can generate optimize code then the possibility of meeting real time constraints becomes higher. And in fact, you can actually prove from the number of cycles required for a computation and depending on the clock speed of the processor whether this computation would meet certain timing constraints or not.

And, optimization of instruction sets of processor using retargetable compilers. This is another way of design space exploration, when we are dealing with parameterized architectures. In fact, this is one point we shall discussed. If you remember in the last class, we had talked about soft core based platforms. So; that means, your architecture is parameterizable. If your architecture is parameterizable you need to now search for an appropriate instruction set as well to exploit the modified architecture. There also compiler becomes a useful tool.

Now, we shall look at one important issue for optimization which is typical to embedded systems that is energy aware compilation. Because, energy is always resource particularly for battery operated embedded systems a number of appliances and at the same time providing you with sophisticated functionalities. See if I have to realize this sophisticated functionalities with minimum consumption of battery power then compiler needs to play a critical role. So, energy saving is essential for battery powered devices.
So, compiler can do an optimization of the machine code that reduces energy consumption. And, this optimization is not that obvious. In fact, I had already talked about I can choose a set of instructions for doing a computation which consumes least number of cycles that gives me efficiency in terms of time. But, it may not give you efficiency in terms of energy.
So, the first kind of optimization in a way is universally valid. I would like my code to run faster even on a general purpose computer. But, in the context of embedded systems is not that I would like my code to run faster consuming less power, so the optimization problem next to look at power as well. So energy of compilation is an important feature for embedded appliances. And in fact, these interest and concern has grown of late in recent years, because of availability of the number of battery power devices.

or such compilation the power models form essential ingredient, what is the power model? Power model actually represents a kind of model which encodes the energy consumption pattern of instructions of the processor.

So here, we are looking at the tradeoff between optimization for low energy and optimization for high performance. High performance is available if memory bandwidth is fully used. And, low energy consumption, if memories are primarily outstanding by mode. And reduced energy, if more values are kept in registers. Then, if we are keeping the variables in registers, what happens, the instructions involving registers may consume less energy. If it is consuming less energy, then it is becoming power efficient. And, in a way is will consume less energy because there are no activities on the bus. So, there will be power efficiency at the same time you are avoiding the memory access machine cycles. So, that would speed up the execution as well. So, if I can come up with these kinds of strategies that become the best for embedded systems.

Now, whether you can come up to these kinds of strategies, would depend on whether you have got an appropriate power model to characterized execution sequence for a processor. So, power model actually provides you with the pattern in which energy will be consumed while execution of instructions.

So, let us take an example of the power model. It would provide, what are called base costs and inter instruction costs, what is the base cost? Base costs of an instruction correspond to energy consumed per instruction execution, if an infinite sequence of that instruction is. If we considered an instruction of the processor and if I want to execute that instructions in definitely then there will be certain power consumption. So, that is the base power consumption for execution of an instruction then there will be inter instruction costs, what is inter instruction costs? The inter instruction costs model the additional energy consumed by the processor if instruction changes. Because, your program will not typically consists of a single instruction. There will be change from one instruction to the other instruction and there will be a cost involving to it.

So, additional cost can come because of switching functional units on and off. Because, there may be you have switching on your floating point functional units. So, that over-ride initiation of the floating point functional unit will have an override and that should be model by inter instruction costs. So, typically a power model of a processor would consist of a base cost for instructions and inter instruction cost.