Unit - 1 Embedded Computing

Introduction :-

In order to understand embedded computing system design process, we need to know how and why microprocessors are used for control, user interface, signal processing and many other tasks.

Complex systems and microprocessors :- A computer is a stored program machine that fetches and executes instructions from a memory. We can attach different types of devices to the computer, load it with different types of software and build many different types of systems.

Embedded Computing System is any device that includes a programmable computer but is not itself intended to be a general purpose computer. Thus a PC is not itself an embedded computing system. But a fax machine or a clock built from a microprocessor is an embedded computing system.

Embedded computing system design is a useful skill for many types of product design.

eg:- Automobiles
Cell phones
Household Appliances

These above devices make extensive use of microprocessors.

Embedding Computers :- Computers have been embedded into applications since many years.

eg:- Whirlwind – A computer designed at M.IT in late 1940s and early 1950s.
– First computer designed to support real-time operation.

What is a Microprocessor?

It is a single chip CPU. VLSI technology has allowed us to put a complete a CPU on a single chip since 1970s. The first microprocessor, Intel 4004 was designed for an embedded application ie a calculator.

Applications :-

* Automobile designers started making use of microprocessors soon after single chip cpus became available. Microprocessors were used to control the engine, ~~de~~ to determine when spark plug fire, to control the fuel/air mixture and so on. A high-end automobile may use 100 microprocessors.
* House-hold appliances :- (1) The typical microwave oven has one microprocessor to control the oven operations.
(2) Thermostats are used in many houses which change the temperature level at various times during the day.
(3) Modern cameras with different features.
(4) Digital television makes use of embedded processors.
* An inexpensive cars may use 40 microprocessors.

Why to use microprocessors? There are different ways to build digital systems;

* Using custom logic
* " Field Programmable Gate Arrays [FPGAs]

But why use Microprocessors?

* Microprocessors are a very efficient way to implement digital systems.
* They make it easier to design families of products that can be built, to provide various feature sets at different price points and can be extended to provide new features to keep up with rapidly changing markets
* CPUs are flexible :- Use of a predesigned instruction set processor may result in faster implementation of your application than designing the custom logic.
* CPUs are efficient :- Microprocessors execute programs very efficiently. Modern RISC processors can execute one instruction per clock cycle and high-performance processors can execute several instructions per cycle.
* CPUs are highly optimized :- Microprocessors are very efficient utilizers of logic. Even though it is true that microprocessor-based designs are inherently much larger than custom logic design, but the size of the microprocessor is small compared to the size of the logic gates. Also just by changing the programs, it can be used for various applications. If we use custom logic, we have to build separate modules, which will remain idle for most of the times.

* Programmability:- Microprocessors provide substantial flexibility. This advantages makes them the best choice in a wide variety of systems. Use of microprocessors is a great benefit during the design process. It allows the program design to be separated from the design of the h/w on which programs will be run. While one team is designing the board, others can be writing the programs, at the same time. In many cases, high-end products can be created simply by adding code without changing the h/w.

* Real-time :- Real-time performance is often best achieved by microprocessors.

* Low-power and low-cost :- Low power and low cost also drive us away from PC architectures and towards multiprocessors. PCs are designed to satisfy a broad mixture of computing requirements and to be very flexible. Those features increase the complexity and price of the components. Those also increase the power utilization.

Challenges in embedded computing system design :-

External constraints are one important source of difficulty in embedded system design. Some of them are,

(1) How much h/w do we need? We have to often meet both performance deadlines and manufacturing cost constraints. So the choice of h/w is important — too little h/w, makes the system fail to meet its deadlines and too

Characteristics of embedded computing applications: Embedded computing systems have to provide sophisticated functionalities like,

* Complex algorithms: Operations performed by the microprocessor may be very sophisticated.
eg:- Microprocessor that controls an automobile engine must perform complicated filtering functions to optimize the performance of the car while minimizing pollution and fuel utilization.

* User interface:- Microprocessors are frequently used ~~for~~ to control complex user interfaces that may include multiple menus and many options.
eg:- Moving maps in GPS navigation.
Embedded computing operations must be performed to meet deadlines.

* Real time:- Many embedded computing systems have to perform in real time. If the data is not ready by a certain deadline, system breaks. In some situations failure to meet a deadline is unsafe and can be dangerous to lives. In some applications, it can create unhappy customers. Missed deadlines in printers can result in scrambled pages.

* Multirate:- Not only operations be completed by the deadlines, but many embedded computing systems have several real-time activities going on at the same time. They may simultaneously control some operations that run at slow rates and others run at

high rates.
Eg:- Multimedia applications, where audio and video portions of a multimedia stream run at very different rates but they must remain closely synchronized. Failure to meet a deadline on either audio or video portion spoils the perception of the entire presentation.

* Costs :- These can be,
  * Manufacturing cost :- The total cost of building the system is very important in many cases. This includes the types of microprocessors used, the amount of memory required, and the types of I/O devices.
  * Power and energy :- Power consumption directly affects the cost of the h/w because a larger power supply may be necessary. Energy consumption affects battery life which is important in many applications, as well as heat consumption, which can be important even in desktop applications.
* Lastly most embedded computing systems are designed by small teams on tight deadlines. Tight deadlines are facts of life in today's environment

* How do we meet deadlines? The brute force way of meeting a deadline is to speed up the h/w so that the program runs faster. But it makes the system more expensive Increasing the CPU clock rate may not make enough difference to execution time because the program's speed may be limited by the memory system.

* How do we minimize power consumption? In battery powered applications, power consumption is extremely important. Even in non-battery applications excessive power consumption can increase heat dissipation. To reduce the power consumption, system should run more slowly but slowing down the system can obviously lead to missed deadlines. Careful design is required to slow down the non-critical parts of the machine for low power consumption while still meeting necessary performance goals.

* How do we design for upgradability? The h/w platform may be used over several product generations or for several different versions of a product in the same generation, with few or no changes. However we want to be able to add features by changing software. How can be design a machine that will provide the required performance for software that we have not yet written?

* Does it really work? Reliability is always important when selling products → customers rightly expect that products they buy will work. Reliability is important in some applications such as safety-critical systems

Because of the following nature of embedded computing machine, the design also become more difficult.

* Complex Testing :- We may have to run a real machine in order to generate the proper data. Timing of data is often important meaning that we cannot separate the testing of an embedded computer from the machine in which it is embedded.

* Limited observability and controllability :- Embedded systems do not come with keyboards and screens. It makes more difficult to see what is going on and to affect the system's operation. In real-time applications, we may not be able to easily stop the system to see what is going inside.

* Restricted Development Environment :- Development environment for embedded systems are much more limited than those available for PCs and work-stations. We generally compile code on one type of machine, such as a PC and download it onto the embedded system. To debug the code, we must usually rely on programs that run on the PC or workstation and then look inside the embedded system.

Requirements:- Before we design a system, we should know what we are designing. We generally proceed in 2 phases.

① We gather an informal description from the customers known as requirements and we refine requirements into a specification that contains enough information to begin designing the system architecture.

Separating our requirements analysis and specification is often necessary because of the large gap between what the customers can describe about the system and what the architects need to design the system. Sometimes they may have some unrealistic expectations as to what can be done within their budgets.

Requirements may be functional or nonfunctional. Typical non-functional requirements include,

① Performance:- The speed of the system is often a major consideration both for the usability and its ultimate cost. Performance may be a combination of soft performance metrics such as approximate time to perform a user-level function and hard deadlines by which a particular operation must be completed.

② Cost:- The target cost or purchase price for the system is always a consideration. Cost has always 2 major components. manufacturing cost includes the cost of the components and assembly, non-recurring engg (NRE) costs include the personnel and other costs of designing the system.

③ Physical size and weight :- It can vary greatly depending upon the application. An industrial control system for an assembly line may be designed to fit into a std-size rack with no strict limitation on weight whereas a hand held device typically has tight requirements on both size and weight.

④ Power Consumption :- Power is very important in battery-powered systems and even in non-battery powered as well. It can be specified in the requirements stage in terms of battery life - the customer is unlikely to be able to describe the allowable wattage.

Validating Requirements :- Validating a set of requirements is ultimately a psychological task because it requires understanding both what people want and how they communicate those needs. One way of to refine atleast the user interface portion of a system's requirements is to build a mock-up. It will give the customer a good idea of how the system will be used and how the user can react to it.

Requirements analysis for big systems can be complex and time consuming. However capturing a relatively small amount of information in a clear, simple format is a good start toward understanding system requirements. To introduce the discipline of requirents analysis as part of system design, we will use a simple requirements methodology. Fig. below shows a sample requirement form, that can be filled out at the start of the project. Let us consider the entries in the form;

* Performance :- Many embedded computing systems spend at least some time controlling physical devices or processing data coming from the physical word. In most of these cases, computations must be performed within a certain time frame. Performance requirements should be identified early because they must be carefully measured during implementation to ensure that the system works properly.

* Manufacturing cost :- It includes primarily the cost of the hlw. components. Even if you don't know exactly, you should have an approx. idea.

* Power :- You may have only rough idea of how much power the system can consume. The most important decision is whether the machine will be battery powered or plugged into the wall.

* Physical size and weight :- You should give some indication of the physical size of the system to help guide certain architectural decisions.

Sample Requirement form :

Name	GPS Moving Map
Purpose	
Inputs	
Outputs	
Functions	
Performance	
Manuf cost	
Power	
Physical size and weight	

* Name: It is simple but helpful.
* Purpose: It is a brief 1 or 2 lines description of what the system is supposed to do.
* Inputs and outputs :- These two entries are more complex than they seem. It also include,
  Types of data: Analog electric signal ? Digital data ? or Mechanical inputs ?
  Data characteristics :- Periodically arriving data such as digital audio samples ? Occasional user inputs ? How many bits per data element ?
  Types of I/o device :- Buttons ? Analog or digital converters ? Video displays ?
  Functions :- It is a more detailed description of what the system does. When the system receives an input what does it do ? How do user interface I/ps affect the these functions ? How do different functions interact ?

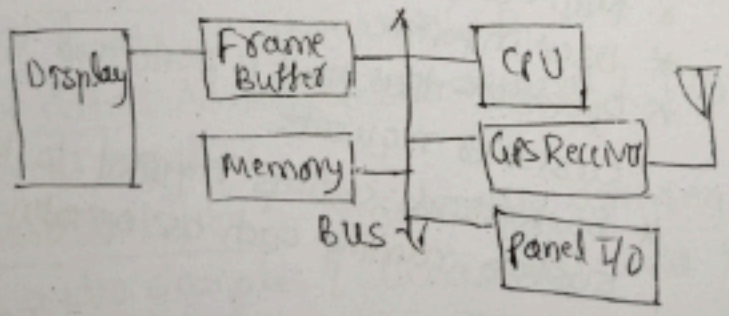
Specification :- It serves as the contract betn the customer and the architect. It must be written carefully so that it accurately reflects the customer's requirements. It should be understandable enough so that some one can verify that it meets system requirements and overall expectations of the customer. Designers can run into several different types of problems caused by unclear specifications If the behavior of some feature in a particular situation is unclear, then the designer may implement the wrong functionality. Specification of the GPU system would include several components.

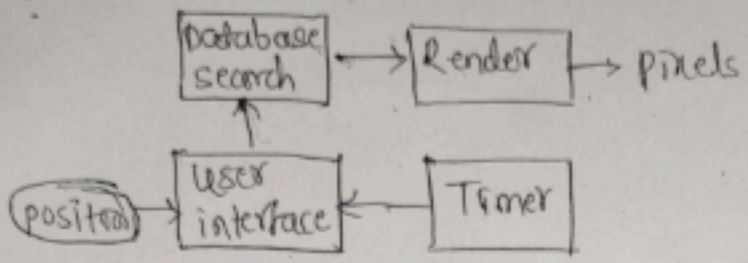
* Data received from the GPS
* Map data
* User interface
* Operations that must be performed to satisfy customer requests.
* Background actions required to keep the system running such as operating the GPS receiver.

Language is used to represent specification.

Architecture Design:- Specification does not say how the system does things, only what the system does. How the system implements the functionalities is the purpose of the arch. Arch is a plan for the overall structure of the system that will be used later to design the components that make up the arch. Below block dgm that shows major operation and data flows among them. This block dgm is still abstract because it is not yet specified which operations will be performed by s/w running on a CPU, what will be done by special-purpose h/w and so on.

H/w and s/w arch. for the moving map.





The h/w blk dgm clearly shows that we have one central CPU surrounded by memory and I/O devices. In particular, we have chosen to use 2 memories; a frame buffer for the pixels to be displayed and a separate prog/data memory for general use by the CPU. The s/w blk dgm fairly closely follows the system blk dgm, but a timer is added to control when we read the buttons on the user interface and render data on to the screen.

Arch. description must be designed to satisfy both functional as well as non-functional requirements. Not only must all the required functions be present but we also need to meet cost, speed, power and other non-functional constraints.

starting out with a system arch. and refining that to h/w and s/w arch. is one good way to ensure that we meet all specifications.

Designing h/w and s/w components:- Components in general include both hardware - FPGAs, boards etc and also software modules.

Some of the components will be ready-made eg: CPU will be a std. component in almost all cases. and memory chips, many other components.

In the GPS moving map, GPS receiver is a good example of a specialised component that will never be predesignd, std. component. We can also make use of std. s/w modules. eg: topographic database. std. databases exist but we want to use " routines to access the database. It is not only in the predetermined format, but it is highly compressed to save storage.