CHARACTERISTICS OF EMBEDDED SYSTEMS.

Embedded computing is in many ways much more demanding than the sort of programs that you may have written for PCs or workstations. Functionality is important in both general-purpose computing and embedded computing, but embedded applications must meet many other constraints as well.

On the one hand, embedded computing systems have to provide sophisticated functionality:

 Complex algorithms: The operations performed by the microprocessor may be very sophisticated. For example, the 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 to control complex user interfaces that may include multiple menus and many options. The moving maps in Global Positioning System (GPS) navigation are good examples of sophisticated user interfaces.

To make things more difficult, embedded computing operations must often be performed to meet deadlines:

 Real time: Many embedded computing systems have to perform in real time—if the data isn't ready by a certain deadline, the system breaks. In some cases, failure to meet a deadline is unsafe and can even endanger lives. In other cases, missing a deadline doesn't create safety problems but does create unhappy customers—missed deadlines in printers, for example, can result in scrambled pages.

 Multirate: Not only must operations be completed by 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 that run at high rates. Multimedia applications are prime examples of multirate behavior. The 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 the audio or video portions spoils the perception of the entire presentation.

Costs of various sorts are also very important:

 Manufacturing cost: The total cost of building the system is very important in many cases. Manufacturing cost is determined by many factors, including the type of microprocessor used, the amount of memory required, and the types of I/O devices.

 Power and energy: Power consumption directly affects the cost of the hardware, 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.

Finally, most embedded computing systems are designed by small teams on tight deadlines. The use of small design teams for microprocessor-based systems is a self-fulfilling prophecy—the fact that systems can be built with microprocessors by only a few people invariably encourages management to assume that all microprocessor-based systems can be built by small

teams. Tight deadlines are facts of life in today's internationally competitive environment. However, building a product using embedded software makes a lot of sense: Hardware and software can be debugged somewhat independently and design revisions can be made much more quickly.

CHALLENGES IN EMBEDDED COMPUTING SYSTEM DESIGN

External constraints are one important source of difficulty in embedded system design. Let's consider some important problems that must be taken into account in embedded system design.

 How much hardware do we need? We have a great deal of control over the amount of computing power we apply to our problem. Not only can we select the type of microprocessor used, we can select the amount of memory, the peripheral devices, and more. Because we often must meet both performance deadlines and manufacturing cost constraints, the choice of hardware is important—too little

hardware and the system fails to meet its deadlines, too much hardware and it becomes too expensive.

 How do we meet deadlines? The brute force way of meeting a deadline is to speed up the hardware so that the program runs faster. Of course, that makes the system more expensive. It is also entirely possible that 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 nonbattery applications, excessive power consumption can increase heat dissipation. One way to make a digital system consume less power is to make it run more slowly, but naively slowing down the system can obviously lead to missed deadlines. Careful design is required to slow down the noncritical parts of the machine for power consumption while still meeting necessary performance goals.

 How do we design for upgradeability? The hardware 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 we design a machine that will provide the required performance for software that we haven't yet written?

 Does it really work? Reliability is always important when selling products—customers rightly expect that products they buy will work. Reliability is especially important in some applications, such as safety-critical systems. If we wait until we have a running system and try to eliminate the bugs, we will be too late—we won't find enough bugs, it will be too expensive to fix them, and it will take too long as well. Another set of challenges comes from the characteristics of the components and systems themselves. If workstation programming is like assembling a machine on a bench, then embedded system design is often more like working on a car—cramped, delicate, and difficult.

Let's consider some ways in which the nature of embedded computing machines makes their design more difficult.

 Complex testing: Exercising an embedded system is generally more difficult than typing in some data. We may have to run a real machine in order to generate the proper data. The 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 computing systems usually do not come with keyboards and screens. This makes it more difficult to see what is going on and to affect the system's operation. We may be forced to watch the values of electrical signals on the microprocessor bus, for example, to know what is going on inside the system. Moreover, in real-time applications we may not be able to easily stop the system to see what is going on inside.

 Restricted development environments: The development environments for embedded systems (the tools used to develop software and hardware) are often much more limited than those available for PCs and workstations. 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.