RETURN

Introduction to embedded and real-time systems

Robert Oshana
Engineering Manager
Texas Instruments
roshana@ti.com

Overview of embedded systems

An embedded system is a specialized computer system that is part of a larger system or machine. Embedded systems can also be thought of as information processing subsystems integrated in a larger system. As part of a larger system it largely determines its functionality. An embedded system usually contains an embedded processor. Many appliances that have a digital interface -- microwaves, VCRs, cars -- utilize embedded systems. Some embedded systems include an operating system. Others are very specialized resulting in the entire logic being implemented as a single program. These systems are embedded into some device for some specific purpose other than to provide general purpose computing . A typical embedded system is shown in Figure 1.

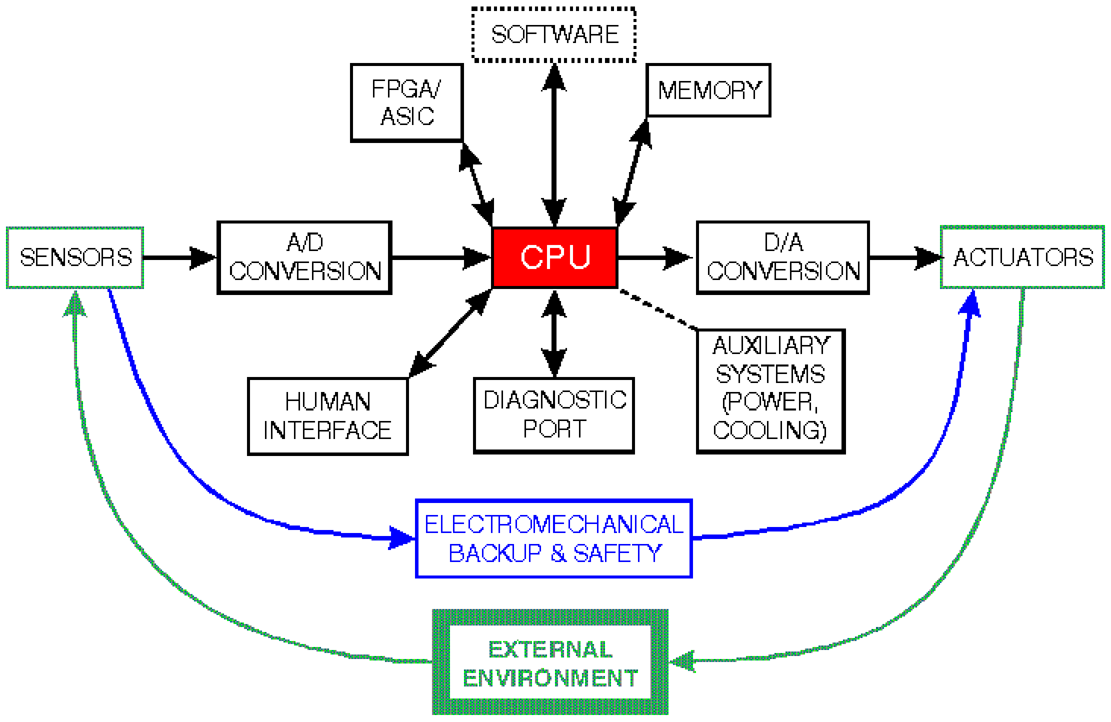

Figure 1 A Typical Embedded System

There are over 3 billion embedded CPUs sold each year. Embedded CPUs are growing at a faster rate than desktop processors (Figure 2). A large part of this growth is in smaller (4-, 8-, and 16-bit) CPUs and DSPs.

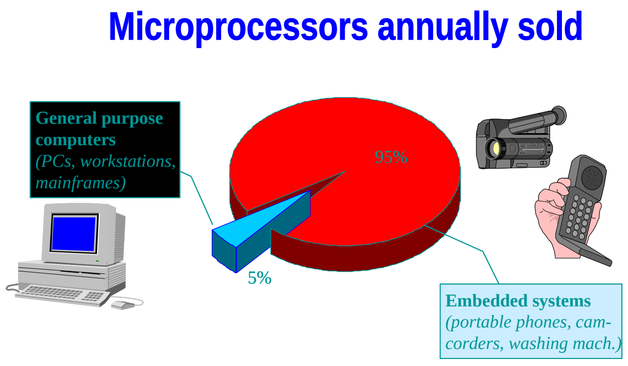

Figure 2 Embedded systems dominate the microprocessor landscape

Embedded systems provide several functions (Figure 3);

- Monitor the environment; embedded systems read data from input sensors. This data is then processed and the results displayed in some format to a user or users
- Control the environment; embedded systems generate and transmit commands for actuators.
- Transform the information; embedded systems transform the data collected in some meaningful way, such as data compression/decompression

Although interaction with the external world via sensors and actuators is an important aspect of embedded systems, these systems also provide functionality specific to their applications. Embedded systems typically execute applications such as control laws, finite state machines, and signal processing algorithms. These systems must also detect and react to faults in both the internal computing environment as well as the surrounding electromechanical systems.

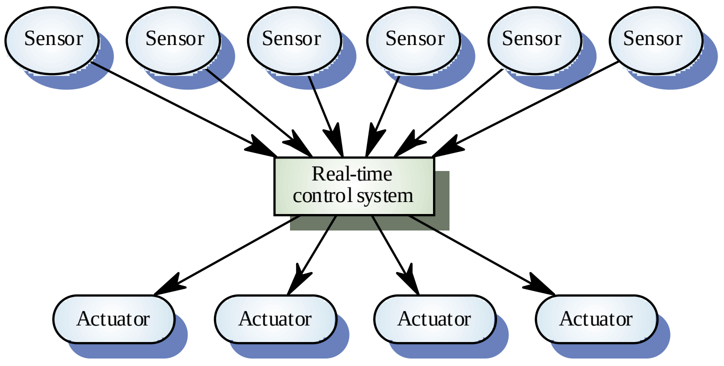

Figure 3 Sensors and Actuators in an Embedded System

There are many categories of embedded systems, from communication devices to home appliances to control systems. Examples include;

- Communication devices

    - modems, cellular phones
- Home Appliances
    - CD player, VCR, microwave oven
- Control Systems
    - Automobile anti-lock braking systems, robotics, satellite control

Characteristics of Embedded Systems

Embedded systems are characterized by a unique set of characteristics. Each of these characteristics imposed a specific set of design constraints on embedded systems designers. The challenge to designing embedded systems is to conform to the specific set of constraints for the application.

Application Specific Systems

Embedded systems are not general-purpose computers. Embedded system designs are optimized for a specific application. Many of the job characteristics are known before the hardware is designed. This allows the designer to focus on the specific design constraints of a well defined application. As such, there is limited user re-programmability. Some embedded systems, however, require the flexibility of re-programmability. Programmable DSPs are common for such applications.

Reactive Systems

As mentioned earlier, a typical embedded systems model responds to the environment via sensors and control the environment using actuators. This requires embedded systems to run at the speed of the environment. This characteristic of embedded system is called "reactive". Reactive computation means that the system (primarily the software component) executes in response to external events. External events can be either periodic or aperiodic. Periodic events make it easier to schedule processing to guarantee performance. Aperiodic events are harder to schedule. The maximum event arrival rate must be estimated in order to accommodate worst case situations. Most embedded systems have a significant reactive component. One of the biggest challenges for embedded system designers is performing an accurate worst case design analysis on systems with statistical performance characteristics (e.g., cache memory on a DSP or other embedded processor). Real time system operation means that the correctness of a computation depends, in part, on the time at which it is delivered. Systems with this requirement must often design to worst case performance. But accurately predicting the worst case may be difficult on complicated architectures. This often leads to overly pessimistic estimates erring on the side of caution. Many embedded systems have a significant requirement for real time operation in order to meet external I/O and control stability requirements. Many real-time systems are also reactive systems.

Distributed Systems

A common characteristic of an embedded system is one that consists of communicating processes executing on several CPUs or ASICs which are connected by communication links. The reason for this is economy. Economical 4 8-bit microcontrollers may be cheaper than a 32-bit processors. Even after adding the cost of the communication links, this approach may be preferable. In this approach, multiple processors are usually required to handle multiple time-critical tasks. Devices under control of embedded systems may also be physically distributed.

Heterogeneous Architectures
Embedded systems often are composed of heterogeneous architectures (Figure 4). They may contain different processors in the same system solution. They may also be mixed signal systems. The combination of I/O interfaces, local and remote memories, and sensors and actuators makes embedded system design truly unique. Embedded systems also have tight design constraints, and heterogeneity provides better design flexibility.

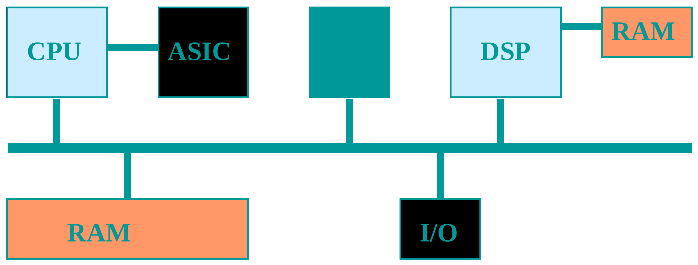

Figure 4. Embedded Systems have Heterogeneous Architectures

Harsh environment
Many embedded systems do not operate in a controlled environment. Excessive heat is often a problem, especially in applications involving combustion (e.g., many transportation applications). Additional problems can be caused for embedded computing by a need for protection from vibration, shock, lightning, power supply fluctuations, water, corrosion, fire, and general physical abuse. For example, in the Mission Critical example application the computer must function for a guaranteed, but brief, period of time even under non-survivable fire conditions. These constraints present a unique set of challenges to the embedded system designer, including accurately modeling the thermal conditions of these systems.

System safety and reliability
As embedded system complexity and computing power continue to grow, they are starting to control more and more of the safety aspects of the overall system. These safety measures may be in the form of software as well as hardware control. Mechanical safety backups are normally activated when the computer system loses control in order to safely shut down system operation. Software safety and reliability is a bigger issue. Software doesn't normally "break" in the sense of hardware. However software may be so complex that a set of unexpected circumstances can cause software failures leading to unsafe situations. Discussion of this topic is outside the scope of this book, but the challenges for embedded designers include designing reliable software and building cheap, available systems using unreliable components. The main challenge for embedded system designers is to obtain low-cost reliability with minimal redundancy.

Control of physical systems
One of the main reasons for embedding a computer is to interact with the environment. This is often done by monitoring and controlling external machinery. Embedded computers transform the analog signals from sensors into digital form for processing. Outputs must be transformed back to analog signal levels. When controlling physical equipment, large current loads may need to be switched in order to operate motors and other actuators. To meet these needs, embedded systems may need large computer circuit boards with many non-digital components. Embedded system designers must carefully balance system tradeoffs among analog components, power, mechanical, network, and digital hardware with corresponding software.

Small and low weight
Many embedded computers are physically located within some larger system. The form factor for the embedded system may be dictated by aesthetics. For example, the form factor for a missile may have to fit inside the nose of the missile. One of the challenges for embedded systems designers is to develop non-rectangular geometries for certain solutions. Weight can also be a critical constraint. Embedded automobile control systems, for example, must be light weight for fuel economy. Portable CD players must be light weight for portability purposes.

Cost sensitivity
Cost is an issue in most systems, but the sensitivity to cost changes can vary dramatically in embedded systems. This is mainly due to the effect of computer costs have on profitability and is more a function of the proportion of cost changes compared to the total system cost.

Power management
Embedded systems have strict constraints on power. Given the portability requirements of many embedded systems, the need to conserve power is important to maintain battery life as long as possible. Minimization of heat production is another obvious concern for embedded systems.

Requirements for Embedded Systems

Embedded systems are unique in several ways, as described above. When designing embedded systems, there are several categories of requirements that should be considered;

- Functional Requirements
- Temporal Requirements (Timeliness)
- Dependability Requirements

Functional Requirements
Functional requirements describe the type of processing the system will perform. This processing varies, based on the application. Functional requirements include the following;

- Data Collection requirements
- Sensoring requirements
- Signal conditioning requirements
- Alarm monitoring requirements
- Direct Digital Control requirements
- Actuator control requirements
- Man-Machine Interaction requirements (Informing the operator of the current state of a controlled object for example. These interfaces can be as simple as a flashing LED or a very complex GUI-based system. They include the ways that embedded systems assist the operator in controlling the object/system.

Temporal Requirement
Embedded systems have many tasks to perform, each having its own deadline. Temporal requirements define the stringency in which these time-based tasks must complete.

Examples include;

- Minimal latency jitter
- Minimal Error-detection latency

Temporal requirements can be very tight (for example control-loops) or less stringent (for example response time in a user interface).

Dependability Requirements
Most embedded systems also have a set of dependability requirements. Examples of dependability requirements include;

- Reliability; this is a complex concept that should always be considered at the system rather than the individual component level. There are three dimensions to consider when specifying system reliability;

    - Hardware reliability; probability of a hardware component failing
    - Software reliability; probability that a software component will produce an incorrect result
    - Operator reliability; how likely that the operator of a system will make an error.

  There are several metrics used to determine system reliability;

    - Probability of failure on demand; likelihood that the system will fail when a service request is made.

- Rate of failure occurrence; frequency of occurrence with which unexpected behavior is likely to occur.
- Mean Time to Failure; the average time between observed system failures.

- Safety; describe the critical failure modes and what types of certification are required for the system

- Maintainability; describes constraints on the system such as type of Mean Time to Repair (MTTR).

- Availability; the probability that the system is available for use at a given time. Availability is measured as;

  Availability = MTTF / (MTTF+MTTR)

- Security; these requirements are often specified as "shall not" requirements that define unacceptable system behavior rather than required system functionality.

Overview of real-time systems

A real-time system is a system that is required to react to stimuli from the environment (including the passage of physical time) within time intervals dictated by the environment. The Oxford dictionary defines a real-time system as "Any system in which the time at which output is produced is significant". This is usually because the input corresponds to some movement in the physical world, and the output has to relate to that same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness. Another way of thinking of real-time systems is any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified period. Generally, real-time systems are systems that maintains a *continuous timely* interaction with its environment (Figure 5).

Correctness of a computation depends not only upon its results but also upon the time at which its outputs are generated A real-time system must satisfy bounded response time constraints or suffer severe consequences. If the consequences consist of a degradation of performance, but not failure, the system is referred to as a soft real-time system (e.g. time adjusting system on computers over the network) If the consequences are system failure, the system is referred to as a hard real-time system. (e.g. emergency patient management system in hospitals).

There are two types of real-time systems: reactive and embedded. Reactive real-time system involves a system that has constant interaction with its environment. (e.g. a pilot controlling an aircraft). An embedded real-time system is used to control specialized hardware that is installed within a larger system. (e.g. a microprocessor that controls the fuel-to-air mixture for automobiles).

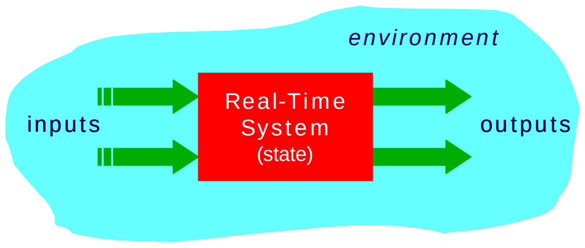

Figure 5 A real-time system interacts with the environment

Real time is a level of computer responsiveness that a user senses as sufficiently immediate or that enables the computer to keep up with some external process (for example, to present visualizations of the weather as it constantly changes). Real-time is an adjective pertaining to computers or processes that operate in real time. Real time describes a human rather than a machine sense of time.

Examples of real-time systems include;

•Software for cruise missile
•Heads-up cockpit display
•Airline reservation system
•Industrial Process Control
•Banking ATM

Real-time systems can also be found in many industries;

•Defense systems
•Telecommunication systems
•Automotive control
•Signal processing systems
•Radar systems
•Automated manufacturing systems
•Air traffic control
•Satellite systems
•Electrical utilities

Real-Time Event Characteristics
Real-time events fall into one of the three categories: asynchronous, synchronous, or isochronous.

- *Asynchronous events* are entirely unpredictable. For example, the event that a user makes a telephone call. As far as the telephone company is concerned, the action of making a phone call cannot be predicted.
- *Synchronous events* are predictable and occur with precise regularity if they are to occur. For example, the audio and video in a movie take place in synchronous fashion.

- *Isochronous events* occur with regularity within a given window of time. For example, audio bytes in a distributed multimedia application must appear within a window of time when the corresponding video stream arrives. Isochronous is a sub-class of asynchronous.

Real-time systems are different from time shared systems in several ways (Table 1)

- *predictably fast response* to urgent events
- *high degree of schedulability*; timing requirements of the system must be satisfied at high degrees of resource usage.
- *stability under transient overload*; when the system is overloaded by events and it is impossible to meet all deadlines, the deadlines of selected critical tasks must still be guaranteed.

Metric	Time-shared systems	Real-time systems
Capacity	**High throughput**	**Schedulability; the ability of system tasks to meet all deadlines**
Responsiveness	**Fast average response**	**Ensured worst-case latency; latency is the worst-case response time to events**
Overload	**Fairness**	**Stability; under overload conditions, the system can meet its important deadlines even if other deadlines cannot be met**

Table 1 Real-time systems are fundamentally different than time shared systems

Characteristics of real-time systems

Real-time systems have many special characteristics which are inherent or imposed. This section will discuss some of these important characteristics.

Large and Complex

Most of the problems associated with developing software are those related to size and complexity. Writing small programs presents no significant problem because they can be designed, coded, maintained and understood by a single person. This largeness is related mostly to variety.The variety is that of needs and activities in the real world and their reflection in a program. The real world is continuously changing. It is evolving. So too are, therefore, the needs and activities of society. Thus large programs, like all complex systems, must continuously evolve.Software programs tend to exhibit the undesirable property of largeness. This is mainly due to continuous change. Real-time systems undergo constant maintenance and enhancements during their lifetimes. They must therefore be extensible.

Manipulation of real numbers

•Many real-time systems involve the control of some engineering activity. For example, consider the model of a plant in Figure 6. In this example, the plant is the controlled entity. The plant produces a vector of output variables that change over time. These outputs are compared to a desired or reference signal to produce an error signal. The controller then uses the error signal to change the input variables.

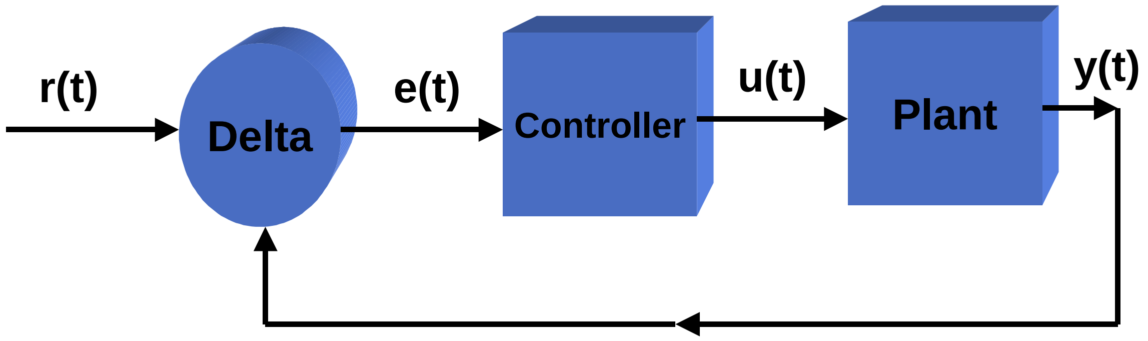

Figure 6 A simple real-time controller

A mathematical model of this system is based on first order differential equations. The output of the system is linked to the internal state of the system and its input variables. A real-time requirement of this system is to move to a new point set within a fixed time period. This adds to the complexity of the computations. This is one reason real-time systems can be so complex.

Reliable and safe

The more society relinquishes control of its vital functions to computers, the more it becomes imperative that those computers do not fail. Failure in ATM machine can result in millions of dollars lost irretrievably. A faulty component in electricity generation could fail a life support system in an intensive care unit. In hostile environments such as the military, systems must be able to fail in a controlled way. For operator interaction, we must minimize the possibility of human error. The size and complexity of real-time systems exacerbates the reliability problem. All expected difficulties inherent in the application must be taken into account (including those introduced by faulty software design!).

Concurrent control of separate system components

A typical real-time embedded system consists of computers and sensors and actuators. There are usually several co-existing external elements which the computer must interact with simultaneously. The very nature of these external elements is that they exist in parallel. Actions performed by the computer must be carried out in sequence but give the allusion of being simultaneous. In some cases this is not possible. An example of this is data that must be collected and processed at various geographical points. In this case, a distributed multiprocessor system must be used. A major problem for systems that must exhibit concurrency is how to express that concurrency in the structure of the program. In the past, it was left up to the programmer to deal with these problems. Systems would be designed to involve the cyclic execution of a program sequence to handle the various concurrent tasks. This was not advisable because is complicated the programmers task and forces consideration of structures that are irrelevant to the control of the tasks at hand. The resulting programs will be more obscure and inelegant. This makes proving a program correctness more difficult. It also makes decomposition of the problem more complex. Also, parallel execution of the program on more than one processor will be much more difficult to achieve, and placement of code to deal with faults becomes more problematic. We will discuss several approaches for handling these problems in the chapter on Real-Time operating systems.

Real-time Facilities
As we have been discussing, response time is crucial to any embedded system. It is very difficult to design and implement systems which will guarantee the appropriate output will be generated at the appropriate times *under all possible conditions.* Doing this and making use of all computing resources at all times is often impossible.
Real-time systems usually constructed using processors with considerable space capacity. This ensures worst case behavior does not produce any unwelcome delays during critical periods of the systems operation. The designer, however, must be cognizant of weight and power issues!
Given adequate processing power, a good real-time programming language, and run-time support is required to enable the programmer;

- to specify times at which actions are to be performed
- to specify times at which actions are to be completed
- to respond to situations where *all* timing requirements cannot be met
- respond to situations where the timing requirements are changed dynamically (mode changes)

Interaction with hardware devices
Nature of embedded real-time systems requires them to interact with the external world. Sensors and actuators are used for a wide variety of real-world devices. Many of the operational requirements for real-time systems are device and computer dependent. These devices may generate interrupts in response to certain events and errors. Interrupts usually handled by assembly language (although more and more is now being done in higher level languages).

Efficient execution and the execution environment
Real-time systems are time critical. Therefore, the efficiency of their implementation is more important than in other systems. One of the main benefits of using a higher level language is to allow the programmer to abstract away the details and concentrate on solving the problem. This is not always true in the embedded system world. Some higher level languages have instruction 10 times slower than assembly language. However, higher level languages can be used in real-time systems effectively.