SNS COLLEGE OF TECHNOLOGY

An Autonomous Institution

Coimbatore-35

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

19ITT204 - MICROCONTROLLER AND EMBEDDED SYSTEMS

II YEAR/ IV SEMESTER

UNIT V EMBEDDED SYSTEM DEVELOPMENT

TOPIC – Design Issues and Techniques

1.Design Requirements

In general, an embedded system design must consider the following :

1. It should handle real time operations and react to external events
2. It should conform to size and weight limits
3. It should be within the power budget and cooling requirements.
4. It should satisfy safety and reliability requirements
5. It must meet tight cost targets.

1.1 Real time/Reactive operation

In real time system, it needs to work quickly but correctly. Real time system operation means that the correctness of a computation depends, in part, on the time at which it is delivered. System design must consider worst case performance also. On complicated architectures, predicting worst case performance is difficult. Examples of the Real time operations are Signal Processing and Mission Critical Systems.

In Reactive operation, software executes in response to an external event . These events can either be periodic or aperiodic. Periodic events results in guaranteed performance whereas aperiodic events may end up with the worst case situations.

1.2 Small size and Low weight

Embedded computers are physically located within artifacts . The form factors are dictated by aesthetics, existing in pre electronic versions and fit into interstices. Weight is considered as a major factor in transportation and portable systems. For example the Mission Critical system has much more stringent size and weight requirements than the others because of its use in a flight vehicle.

If the shape is non-rectangular and non-planar geometries, then it is difficult to reduce the size as well as its weight. Packaging and integration of digital, analog and power circuits will lead to reduction in size.

1.3 Safe and reliable

Embedded system failures may result in severe damages. For example in mission-critical applications such as aircraft flight control, embedded system failures may result in severe personal injury and equipment damage. Embedded system that fails cannot tolerate the added cost of redundancy in hardware. At the same time, it cannot have processing capacity needed for traditional fault tolerance techniques. Low-cost reliability with minimal redundancy will be a difficult task to achieve.

1.4 Harsh environment

Embedded systems may not operate in a controlled environment. They may be subject to excessive heat, 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. It is difficult to make accurate thermal modelling. Also depending on operating environment, components has to be changed for each design.

1.5 Cost Sensitivity

Cost is one of the issues in developing a system . A little change in cost affects the manufacturing quantity of the system. Although designers of systems large and small may talk about the importance of cost with equal urgency, their sensitivity to cost changes can vary dramatically. A reason for this may be that the effect of computer costs on profitability is more a function of the proportion of cost changes compared to the total system cost, rather than compared to the digital electronics cost alone. Variable “design margin” to permit trade-off between product robustness and aggressive cost optimization is a challenging task.

2. System-level requirements

Designer should consider the End-product utility, System safety & reliability, Controlling physical systems and Power management .

2.1 End-product utility

The utility of the end product is the goal when designing an embedded system, not the capability of the embedded computer itself. Embedded products are typically sold on the basis of capabilities, features, and system cost. Embedded system mechanisms and their associated I/O are largely defined by the application. Software is used to coordinate the mechanisms and define their functionality. Finally, computer hardware is made available as infrastructure to execute the software and interface it to the external world. The design challenge in end product utility is software- and I/O-driven hardware synthesis.

2.2 System safety & reliability

The Distributed system example is mission critical. But does not employ computer redundancy. Instead, mechanical safety backups are activated when the computer system loses control. A bigger and more difficult issue at the system level is software safety and reliability. A set of unexpected circumstances can cause software failures leading to unsafe situations. The system safety and reliability depends on reliable software and it should be cheap and available using unreliable components. Electronic *vs.* non-electronic design tradeoffs will affect system safety.

2.3 Controlling physical systems

Embedding a computer to interact with the environment by monitoring and controlling external machinery. For this, analog inputs and outputs must be transformed to and from the digital signal levels. Significant current loads may need to be switched in order to operate motors, light fixtures, and other actuators. These lead to a large computer circuit board dominated by non-digital components. “smart” sensors and actuators (that contain their own analog interfaces, power switches, and small CPUS) may be used to off-load interface hardware from the central embedded computer. With an embedded network the amount of system wiring and number of connector contacts are reduced. This change brings with it an additional computer design problem of partitioning the computations among distributed computers in the face of an inexpensive network with modest bandwidth capabilities. Distributed system tradeoffs among analog, power, mechanical, network, and digital hardware plus software is a challenge to control physical system.

2.4 Power management

There is need for power management to either minimize heat production or conserve battery power. With evolution of laptops, significantly lower power is needed in order to run from inexpensive batteries for 30 days in some applications and up to 5 years in others. Ultra-low power design for long-term battery operation is preferred in power management.

3. Life-cycle support

Figure 1. Embedded system life cycle

3.1 Component acquisition

Embedded systems are more application driven than a typical technology-driven desktop computer design. There may be more leeway in component selection. Component acquisition costs can be taken into account when optimizing system life-cycle cost. For example, the cost of a component generally decreases with quantity, so design decisions for multiple designs should be coordinated to share common components to the benefit of all. Challenges here are life-cycle, cross-design component cost models and optimization rather than simple per-unit cost.

3.2 System certification

Embedded computers can affect the safety as well as the performance of the system. Rigorous qualification procedures are necessary after any design change to assess and reduce the risk of malfunction or unanticipated system failure. This additional cost can negate any savings. This hinders use of new technology by re-synthesizing hardware components. Redesigned components cannot be used without incurring the cost of system recertification. One strategy to minimize the cost of system recertification is to delay all design changes until major system upgrades occur. As distributed embedded systems come into more widespread use, another likely strategy is to partition the system in such a way as to minimize the number of subsystems that needs to be recertified when changes occur. This is a partitioning problem affected by potential design changes, technology insertion strategies, and regulatory requirements. Recertification cost has to be minimized with partitioning/synthesis.

3.3 Logistics and repair

Whenever an embedded computer design is created or changed, it affects the downstream maintenance of the product. A failure of the computer can cause the entire system to be unusable until the computer is repaired. Embedded systems must be repairable in a few minutes to a few hours, so spare components and maintenance personnel must be located close to the system. A fast repair time may also imply that extensive diagnosis and data collection capabilities must be built into the system, which may be at odds with keeping production costs low. Designs have to be optimized to minimize spares inventory. High-coverage diagnosis and self-test at system level, not just digital component level is needed.

3.4 Upgrades

Upgrades may be subject to recertification requirements. Legacy software may not be executable on upgraded replacement hardware, and may not be readily cross-compiled to the new target CPU. Timing behavior is likely to be different on newer hardware, but may be both undocumented and critical to system operation. When upgrading designs we should ensure complete interface, timing, and functionality compatibility.

3.5 Long-term component availability

When embedded systems are more than a few years old, some electronic components may no longer be available for production of new equipment or replacements. This problem can be especially troublesome with obsolete processors and small-sized dynamic memory chips. When a product does reach a point at which spare components are no longer economically available, the entire embedded computer must sometimes be redesigned or upgraded. Updating old designs to incorporate new components should be done cost effectively.

4. Business model

The business models under which embedded systems are developed can vary as widely as the applications themselves. Costs, cycle time, and the role of product families are all crucial business issues that affect design decisions.

4.1 Design vs. production costs

Design costs, also called Non-Recurring Engineering costs (NRE), are of major importance when few of a particular embedded system are being built. Production costs are important in high-volume production. Embedded systems vary from single units to millions of units, and so span the range of tradeoffs between design versus production costs.

*C*AD tools can help designers complete their work with a minimum of effort. The designs may be simple enough and engineering cost such a small fraction of total system cost that extensive hand-optimization is performed in order to reduce production costs.

CAD tools may be able to outperform an average engineer at all times, and a superior engineer on very large designs. In small designs some embedded computer designers believe that a superior human engineer can outperform CAD tools. In the Small system example a programmer squeezed software into a few hundred bytes of memory by hand when the compiler produced overly large output that needed more memory than was available. It can readily be debated whether CAD tools or humans are “better” designers, but CAD tools face skepticism in areas that require extraordinary optimization for size, performance, or cost. we should intelligently trade off design time versus production cost.

4.2 Cycle time

The cycle time between identification of a product opportunity and product deployment (also called Time to Market) can be quite long for embedded systems. In many cases the electronics are not the driving force; instead, product schedules are driven by concerns such as tooling for mechanical components and manufacturing process design. Superficially, this would seem to imply that design time for the electronics is not an overriding concern, but this is only partially true. Because the computer system may have the most malleable design, it may absorb the brunt of changes.

4.3 Product families

In many cases embedded system designs are not unique, and there are a variety of systems of various prices and capabilities forming a product family. To the extent that system designers can reuse components, they lower the total cost of all systems in the product family. There is a dynamic tension between overly general solutions that satisfy a large number of niche requirements, and specifically optimized designs for each point in a product family space. There may be cases in which contradictory requirements between similar systems prevent the use of a single subsystem design. In the Mission Critical and Small examples different customers require different interfaces between the embedded system and their equipment. In the Distributed example regulatory agencies impose different safety-critical behavior requirements depending on the geographic area in which the system is deployed.

5. Design culture

Design is a social activity as well as a technical activity. The design of desktop computers and CPUs in particular, has matured in terms of becoming more quantitative in recent years. With this new maturity has come an emphasis on simulation and CAD tools to provide engineering tradeoffs based on accurate performance and cost predictions. Computer designers venturing into the embedded arena must realize that their culture (and the underlying tool infrastructure) are unlike what is commonly practiced in some other engineering disciplines.

Embedded system design requires a confluence of engineering skills, successful computer designers and design methodologies must find a harmonious compromise with the techniques and methodologies of other disciplines as well as company management. In many cases the engineers building embedded computer systems are not actually trained in computer engineering (or, perhaps not even electrical engineering), and so are not attuned to the culture and methodologies of desktop computer design.

In the Mission Critical and Small examples different customers require different interfaces between the embedded system and their equipment. In the Distributed example regulatory agencies impose different safety-critical behavior requirements depending on the geographic area in which the system is deployed.

5.1 Computer culture vs. other cultures

A specific problem is that computer design tools have progressed to the point that many believe it is more cost-effective to do extensive simulation than build successive prototypes. In the mechanical arena much existing practice strongly favors prototyping with less exhaustive up-front analysis.

It may be difficult to convince project managers to spend limited capital budgets on CAD tools and defer the gratification of early prototype development in favour of simulation. It is advisable to make simulation-based computer design accessible to non-specialists.

5.2 Accounting for cost of engineering design

One area of common concern is the effectiveness of using engineers in any design discipline. Some computer design CAD tools are very expensive, and in general organizations have difficulty trading off capital and tool costs against engineering time. This means that computer designers may be deprived of CAD tools that would reduce the total cost of designing a system.

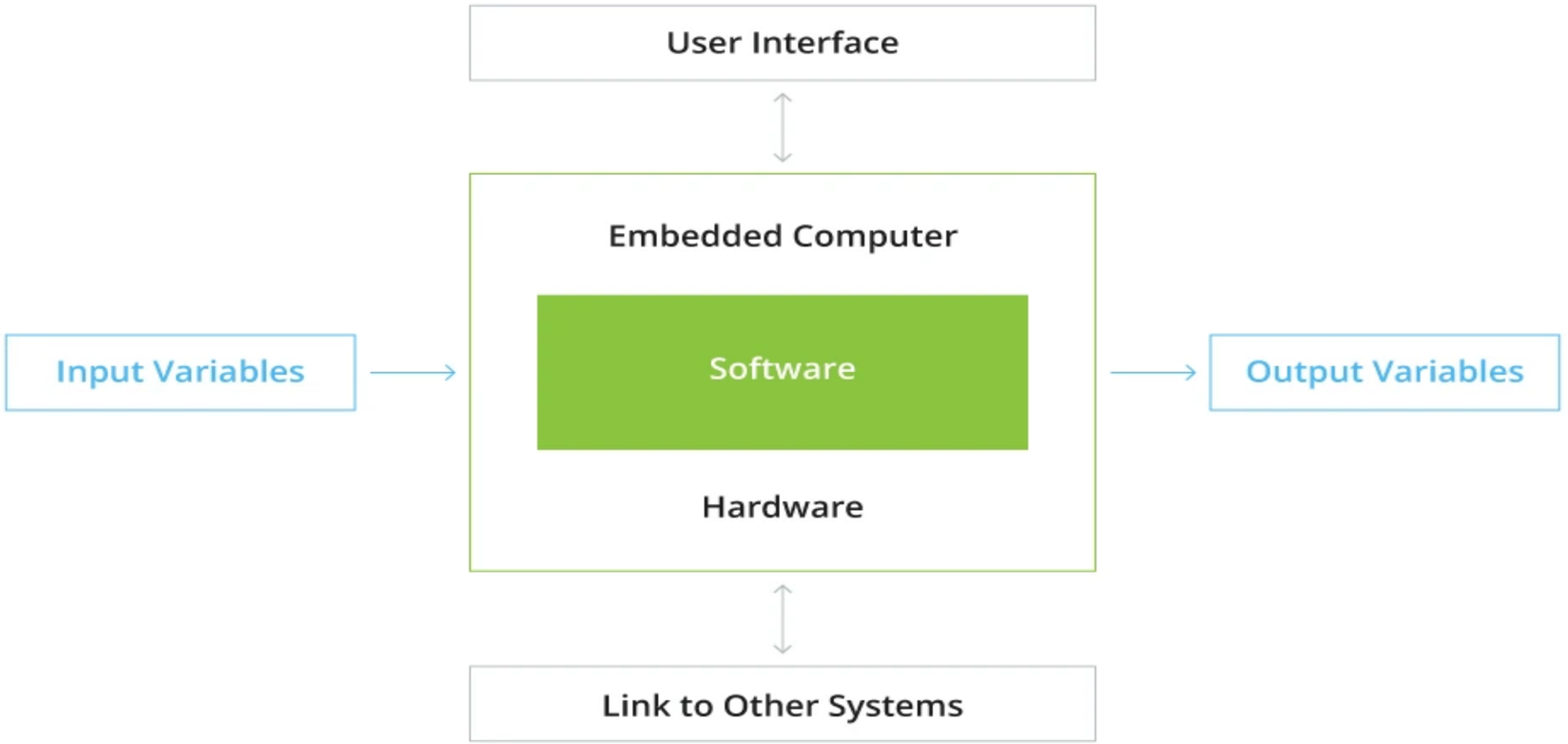
In high-volume applications engineering costs can be relatively small when compared to production costs. The number of engineers is fixed, and book-kept as a constant expense that is decoupled from the profitability of any particular system design, as is the case in all four example systems. This can be referred to as the "Engineers Are Free" syndrome. While the cost of engineering time may have a small impact on product costs, the unavailability of enough engineers to do work on all the products being designed can have a significant opportunity cost (which is, in general, unmeasured). Improved productivity via using tools and methodologies may be better received by managers if it is perceived to increase the number of products that can be designed, rather than merely the efficiency of engineers on any given product design effort.

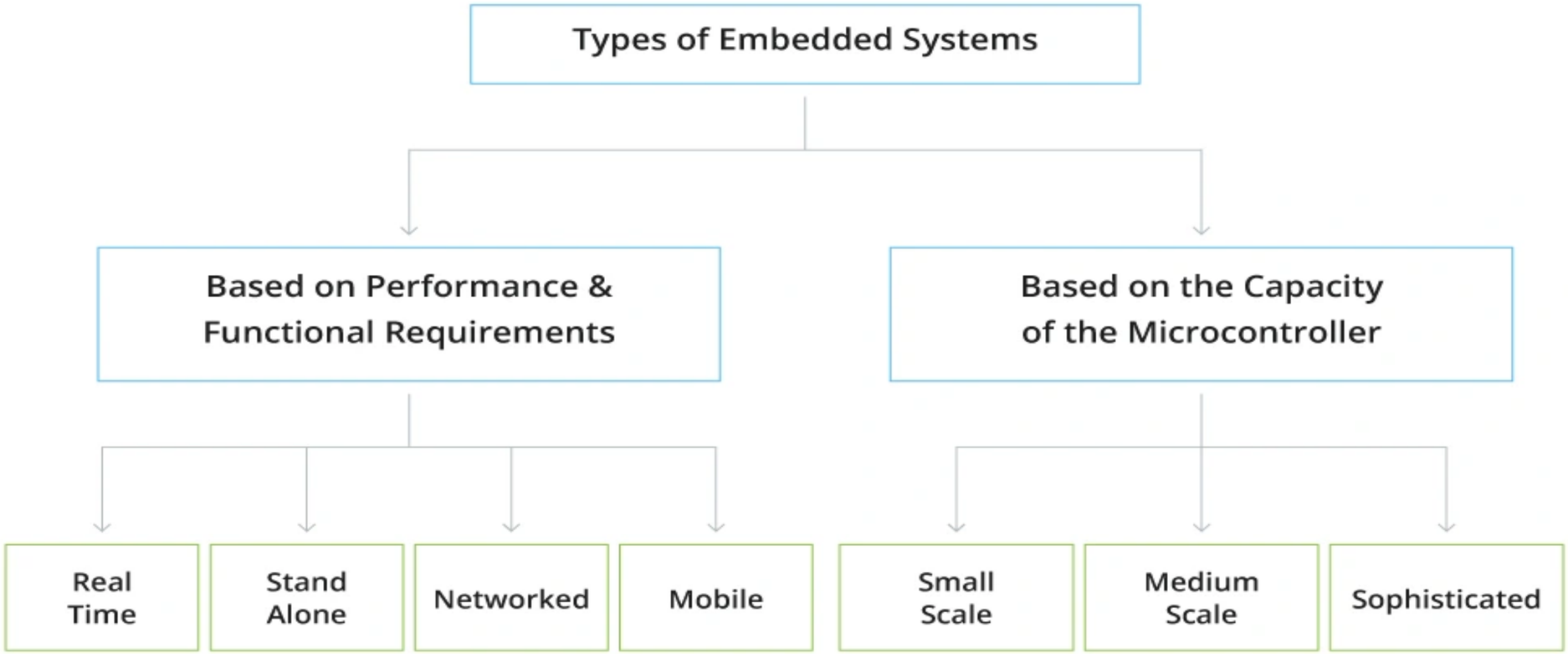
5.3 Inertia

In general, the cost of change in an organization is high both in terms of money and organizational disruption. The computer industry can be thought of as being forced to change by inexorable exponential growth in hardware capabilities. However, the impact of this growth seems to have been delayed in embedded system development. In part this is because of the long time that elapses between new technology introduction and wide-scale use in inexpensive systems.

Thus, it may simply be that complex designs will force updated CAD tools and design methodologies to be adopted for embedded systems in the near future. On the other hand, the latest computer design technologies may not have been adopted by many embedded system makers because they aren't necessary. Tool development that concentrates on the ability to handle millions of transistors may simply not be relevant to designers of systems using 4- and 8-bit microprocessors that constitute the bulk of the embedded CPU market.

And, even if they are useful, the need for them may not be compelling enough to justify the pain and up-front expense of change so long as older techniques work. That is not to say that new tools aren't needed, but rather that the force of cultural inertia will only permit adoption of low-cost tools with significant advantages *to* the problem at hand. Find/create design tools and methodologies that provide unique, compelling advantages for embedded design.


User Interface
Embedded Computer
Input Variables
Software
Output Variables
Hardware
Link to Other Systems


Types of Embedded Systems
Based on Performance & Functional Requirements
Based on the Capacity of the Microcontroller
Real Time
Stand Alone
Networked
Mobile
Small Scale
Medium Scale
Sophisticated

Challenges of Embedded Software Development

- Embedded software is always a constituent of a larger system, for instance, a digital watch, a smartphone, a vehicle or automated industrial equipment.

- Such embedded solutions must have real-time response under all circumstances within the time specified by design and operate under the condition of limited memory, processing power and energy supply.

- Moreover, embedded software must be immune to changes in its operating environment – processors, sensors, and hardware components may change over time. Other challenging requirements to embedded software are portability and autonomy.

Challenge #1: Stability

Stability is of paramount importance. Unexpected behavior from an embedded software is inadmissible and poses serious risks. End users demand that embedded software must have uniform behavior under all circumstances and be able to operate durably without service.

Challenge #2: Safety

Safety is a special feature of embedded software due to their primary application associated with lifesaving functionality in critical environments. Software Development Life Cycle (SDLC) for embedded software is characterized by more strict requirements and limitations in terms of quality, testing, and engineering expertise.

Challenge #3: Security

Security became a burning issue in the digital world. The related risks grow exponentially, especially so for IoT devices gaining popularity worldwide and becoming more interconnected to each other. Because modern home appliances like electric cookers, refrigerators and washing machines have connectivity feature integrated by default, Internet of Things now is exposed to a serious risk of hacking attacks. For instance, the most pressing security concerns of 2020 were malware or ransome protection, financial losses and data theft as well as intellectual property infringement along with confidential informal leaks.

Challenge #4: Launch Phase

Time-to-market and time-to-revenue have always been tough indicators in embedded software development, especially in the IoT segment. That is why the apps and platforms supposed to support numerous IoT devices expected to reach 13.8 billion units in 2021 and surpass a 30-billion mark by 2025. Fabrication of hardware components housing embedded software requires extreme integration and flexibility due to very fast development of IoT industry. In addition, taking into account longer IoT device lifespan, future updates and releases become an issue for component designers.

Challenge #5: Design Limitations

The challenges in design of embedded software have always been in the same limiting requirements for decades:

- Small form factor;
- Low energy;
- Long-term stable performance without maintenance.

The market demands from designers to pack more processing power and longer battery life into smaller spaces, which is often a tradeoff. Finally, depending on applications in IoT, there is a growing demand for manufacture of very scalable processor families ranging from cheap and ultra-low-power to maximum performance and highly configurable processors with forward-compatible instruction set. There is similar demand for increased performance of system buses and internal/external memory caches.

Challenge #6: Compatibility and Integrity

Gartner Group estimation shows that, presently, most of the apps in the market are launched by businesses younger than 3 years old. With all their probable expertise in software development, many of them lack hands-on experience in implementing and updating their applications in IoT environment specifically, especially with regard to security. In its 2020 Unit 42 IoT Threat Report, Palo Alto Networks revealed that 98% of exisiting IoT device traffic is not secured, which constitutes a threat attack of confidential information and exposing personal data.

Further expansion of IoT devices on the background of their connectivity puts more pressure on their adaptability. Users must be capable of administering the app through a simple user interface via all available channels including over-the-air firmware updates, which needs extreme compatibility across the entire ecosystem.

Integrity becomes a function of security. To protect the IoT from malicious attacks or compromising, security must be implemented within each device at every level: the end node, gateway, cloud, etc.

Key Issues of Embedded Software Development

Connectivity

Over-the-air Updates

Debugging

Pace of Change

THANK YOU