# SNS COLLEGE OF TECHNOLOGY

*(An Autonomous Institution)*

**COIMBATORE – 641035**

**19MCE304- DESIGN OF EMBEDDED SYSTEMS**

## Challenges in embedded system design

Designing embedded systems presents a unique set of challenges due to the specialized nature of these systems and their diverse applications. Here are some of the key challenges:

### 1. Resource Constraints

- **Memory**: Limited RAM and ROM require efficient use of memory and careful management of resources.
- **Processing Power**: Often limited by the capabilities of the microcontroller or microprocessor, necessitating optimized code.
- **Power Consumption**: Essential for battery-operated devices, requiring efficient power management strategies.

### 2. Real-Time Requirements

- **Deterministic Behavior**: Ensuring the system responds to inputs within a specific timeframe is crucial for real-time applications.
- **Latency and Jitter**: Minimizing delays and variations in response times.

### 3. Reliability and Robustness

- **Fault Tolerance**: The system must handle errors gracefully and continue to operate or fail safely.
- **Long-Term Operation**: Designed for continuous operation, often without human intervention, requiring robust components and software.

### 4. Security

- **Vulnerability to Attacks**: Protecting the system from various types of cyberattacks.
- **Data Integrity**: Ensuring the accuracy and consistency of data.
- **Secure Communication**: Safeguarding data transmitted between devices.

**5. Cost Constraints**

- **Component Costs**: Balancing the need for performance and functionality with the budget.
- **Production Costs**: Considering the cost of manufacturing, especially in large volumes.

**6. Scalability and Flexibility**

- **Scalability**: Designing systems that can be scaled up or down based on changing requirements.
- **Flexibility**: Ensuring the system can be easily updated or modified to accommodate new features or changes in requirements.

**7. Integration**

- **Hardware-Software Co-Design**: Ensuring seamless integration and interaction between hardware and software components.
- **Compatibility**: Integrating with other systems or devices, which may involve diverse communication protocols and standards.

**8. Development and Debugging**

- **Complexity**: Managing the complexity of both hardware and software design.
- **Testing and Verification**: Thoroughly testing the system to ensure it meets all requirements and specifications.
- **Debugging**: Diagnosing and fixing issues in an environment where traditional debugging tools may be limited.

**9. Environmental Conditions**

- **Operating Conditions**: Ensuring the system performs reliably under various environmental conditions, such as extreme temperatures, humidity, and vibrations.
- **Durability**: Designing for physical robustness to withstand harsh conditions.

**10. Regulatory Compliance**

- **Standards and Regulations**: Meeting industry-specific standards and regulations, such as those for medical devices, automotive systems, or industrial equipment.

**11. Time-to-Market**

- **Development Timeline**: Rapid development cycles to bring products to market quickly while maintaining quality and reliability.
- **Prototyping and Testing**: Efficiently creating prototypes and conducting thorough testing to validate designs.

**12. Interfacing with Legacy Systems**

- **Backward Compatibility**: Ensuring new systems can interface with and support older, legacy systems.
- **Migration Paths**: Providing clear and feasible paths for transitioning from old to new systems.

Successfully addressing these challenges requires a combination of careful planning, skilled engineering, and often innovative problem-solving. Advanced tools and methodologies, such as model-based design, simulation, and formal verification, can aid in managing these complexities.

## Design Challenge — Optimizing Design Metrics

The embedded-system designer must of course construct an implementation that fulfills desired functionality, but a difficult challenge is to construct an implementation that simultaneously optimizes numerous design metrics.

## Common Design Metrics

For our purposes, an implementation consists either of a microprocessor with an accompanying program, a connection of digital gates, or some combination thereof. A *design metric* is a measurable feature of a system's implementation. Commonly used metrics include:

---

[1] *JPEG* is short for *Joint Photographic Experts Group*. "Joint" refers to the group's status as a committee working on both ISO and ITU-T standards. Their best-known standard is for still-image compression.

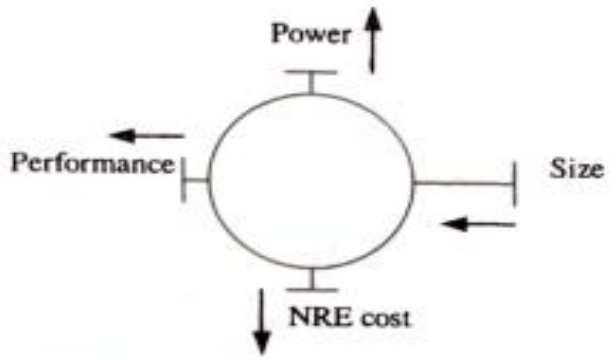


Figure 1.3: Design metric competition — improving one may worsen others.

- *NRE cost* (nonrecurring engineering cost): The one-time monetary cost of designing the system. Once the system is designed, any number of units can be manufactured without incurring any additional design cost; hence the term *nonrecurring*.
- *Unit cost*: The monetary cost of manufacturing each copy of the system, excluding NRE cost.
- *Size*: The physical space required by the system, often measured in bytes for software, and gates or transistors for hardware.
- *Performance*: The execution time of the system.
- *Power*: The amount of power consumed by the system, which may determine the lifetime of a battery, or the cooling requirements of the IC, since more power means more heat.
- *Flexibility*: The ability to change the functionality of the system without incurring heavy NRE cost. Software is typically considered very flexible.
- *Time-to-prototype*: The time needed to build a working version of the system, which may be bigger or more expensive than the final system implementation, but it can be used to verify the system's usefulness and correctness and to refine the system's functionality.
- *Time-to-market*: The time required to develop a system to the point that it can be released and sold to customers. The main contributors are design time, manufacturing time, and testing time.
- *Maintainability*: The ability to modify the system after its initial release, especially by designers who did not originally design the system.
- *Correctness*: Our confidence that we have implemented the system's functionality correctly. We can check the functionality throughout the process of designing the system, and we can insert test circuitry to check that manufacturing was correct.
- *Safety*: The probability that the system will not cause harm.

Metrics typically compete with one another: Improving one often leads to worsening of another. For example, if we reduce an implementation's size, the implementation's performance may suffer. Some observers have compared this phenomenon to a wheel with numerous pins, as illustrated in Figure 1.3. If you push one pin in, such as size, then the other

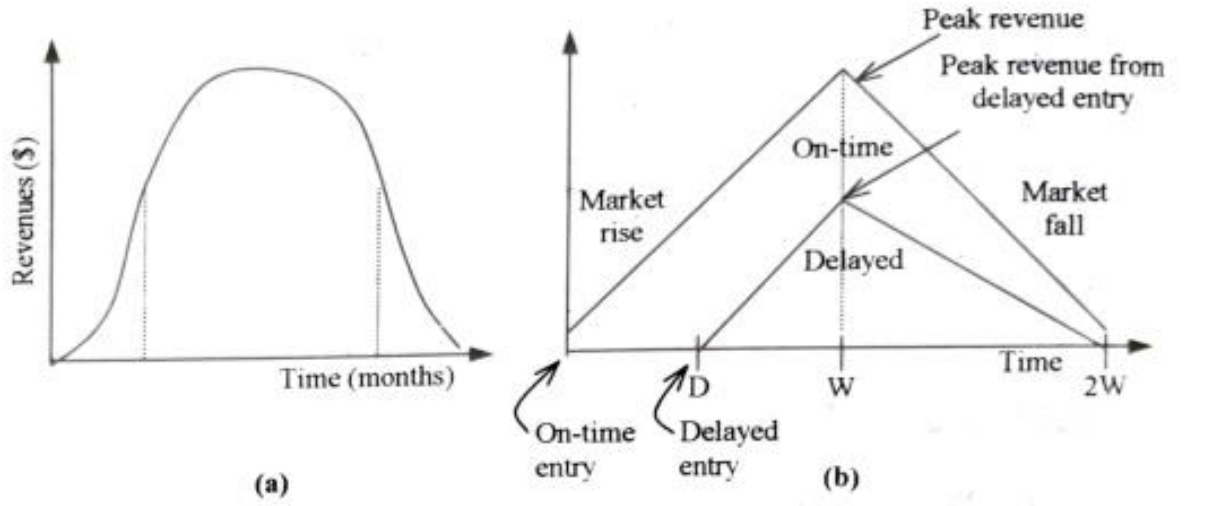


Figure 1.4: Time-to-market: (a) market window, (b) simplified revenue model for computing revenue loss from delayed entry.

pins pop out. To best meet this optimization challenge, the designer must be comfortable with a variety of hardware and software implementation technologies, and must be able to migrate from one technology to another, in order to find the best implementation for a given application and constraints. Thus, a designer cannot simply be a hardware expert or a software expert, as is commonly the case today; the designer must have expertise in both areas.