**Embedded Systems Introduction**

**What is an Embedded System?**

- Definition of an embedded computer system:
  - is a digital system.
  - uses a microprocessor (usually).
  - runs software for some or all of its functions.
  - frequently used as a controller.

**What an embedded system is NOT.**

- Not a computer system that is used primarily for processing.
- Not a software system on a PC or Unix box.
- Not a traditional business or scientific application.

**Examples of Embedded Systems**

- Medical instrument's controls: CAT scanners, implanted heart monitors, etc.
- Automotive systems: electronic dashboards, ABS brakes, transmission controls.
- Controls for digital equipment: CD players, TV remote, programmable sprinklers, household appliances, etc.

**Why ‘embedded’?**

- Because the processor is ‘inside’ some other system.
- A microprocessor is ‘embedded’ into your TV, car, or appliance.
- The consumer does not think about performing processing.
- Considers running a machine or ‘making something work’.
- Considered “part of” the thing rather than the thing

**Special Characteristics**

- **Hardware and software** (in one system)
- concurrency (several processes working at same time)
- timing (often real time)
- sensors and actuators (for inputs and outputs)
- synchronization (this process must complete before this process begins)
Timing and Concurrency

- Engine shaft angle
- Fire Spark
- Fire Injection
- Watch Emissions

How are embedded systems different than traditional software?

- Responding to sensors (Was this button pushed?)
- Turning on actuators (Turn on power to the boiler.)
- Real-time (Respond to temperature change within 3 seconds.)

Differences between ES and traditional software development

- Not dealing with only sequential code.
- Routine can stop at completion or in response to an external event.
- Many parts of system might be running concurrently.
- Safety-critical component of many systems.

Small and Many!

- Most embedded systems use 4-, 8-, or 16-bit processors. 32-bit used for intensive applications like printer controls.
- 8-bit processors have about 64K of memory, that limits amount of code.
- "By 1990 a total of about 45 million recognizable computers (i.e., PCs, Macintosh, even CP/M systems) were in place. Yet over 1 billion microprocessors and microcontrollers were shipped in that year alone!"


hardware or software?

- Where to place functionality?
  - ex: A Sort algorithm
    - Faster in hardware, but more expensive.
    - More flexible in software but slower.
    - Other examples?
- Must be able to explore these various trade-offs:
  - Cost.
  - Speed.
  - Reliability.
  - Form (size, weight, and power constraints).

hardware/software Codesign or ‘Codesign’

- Model the hardware and the software system in a unified approach.
- Use similar design models.
- Need for ‘model continuity’ spanning levels of the design process.

**Traditional Embedded System Development Approach**

- Decide on the hardware
- Give the chip to the software people.
- Software programmer must make software ‘fit’ on the chip and only use that hardware’s capabilities.

**Increased Complexity**

- Systems are becoming more and more complex.
- Harder to think about total design.
- Harder to fix ‘bugs.’
- Harder to maintain systems over time.
- Therefore, the traditional development process has to change.

**Less Time to Develop**

- In embedded electronics, the total design cycle must decrease.
- Historically, design for automotive electronic systems takes 3-5 years to develop.
- Must become a 1-3 year development cycle.
- Must still be reliable and safe.

**Solutions to Complexity:**

- Need to keep design process abstract for a longer period of time.
- Decomposable hierarchy (object-oriented).
- Reuse previous designs:
  - When a design changes, reuse similar sections.
  - Don’t throw away last year’s design and start from scratch!
- Automated verification systems.

**Example: Fly-by-Wire Airplane**

- How would you start to think about developing this complex/large system?
- What are potential problems with deciding on the hardware right away?
- What are possible concurrent systems needs?
- What type of timing constraints might be needed?

**Fly-by-Wire Airplane Continued**

- What would be the sensors and actuators of this system?
- How concerned should developers be about the safety of the system?
- Would testing be enough?