Embedded Systems

A system that has embedded software and computer hardware, which make it a system dedicated for an application or a specific part of an application or product or a part of a larger system.

Any device that includes a programmable computer but is not itself intended to be a general-purpose computer.
Real Time Computing

Embedded computing where responsiveness matters; where inputs and/or outputs have to be processed in real time.
History

- Elevators in the early 1900’s
- Alan Turing’s BOMBE computer
- Navy ships in the 1950’s
- Pinball machines from the 1950’s on
- Arpanet hubs in the 1960’s
- Today: everything from pacemakers to thermostats to toasters
What device did you think of and how might it work
Real-Time & Embedded

- Restrictions => challenges
- Constraints => techniques
Challenge: Restricted Hardware

- CPU power, memory, power
  - Arduino: 32K memory, 16mhz clock

- Hardware costs dominate
  - 10 million units – every penny matters
  - Unlikely to be upgraded
  - Need to get it right the first time
  - Energy is important
Challenge: Reliability

- A system crash can cause injury or real damage
  - X-ray machines, planes
  - GPS driving a car into a river
  - Imlac display amps
  - Mars probe with wrong units

- **Systems must minimize risk**
  - Self-driving cars
  - Fly-by-wire jet with minor pilot errors
  - Computer driven cars
Reliability: Solutions

- Modeling and analysis of SW & HW
  - To determine behavior before implementation
  - Finite state modeling (control-based)
  - Continuous modeling of physical systems
  - Queueing theory, data and control modeling
  - Timing analysis, Fault analysis, Simulation
  - Testing techniques

- Verifying software
Challenge: Maintainability

- Who is going to upgrade a toaster
- Other devices may be upgraded
  - How much work & cost involved
  - Is it feasible
  - Planning for this
- Upgrades might be regulated
  - Nuclear plants, medical devices
Challenge: Limited OS Support

- Often none at all
  - You have to do your own scheduling
- You might build a mini-OS as part of software
  - Threads, scheduling, synchronization, ...
  - Input/output devices and drivers
  - Interrupt processing
  - Debugger?
- Might need test hardware platform
Challenge: Emphasis on I/O

- Need to interact with the world
- Multiple types of input devices
  - And each is different/unique
- Multiple types of output devices
- Reactions need to be quick
- Devices may be analog/continuous
Challenge: Performance

- **Important:**
  - Reacting within the proper time
  - Finding input changes appropriately
  - Power usage

- **Unimportant:**
  - Performance: *fast enough* is all that is needed
  - Performance sacrificed for
    - Memory, cost, battery life, predictability, simplicity, reliability
Challenge: Control Orientation

- Dealing with inputs/outputs
  - Get external stimulus
  - Provide appropriate external actions
  - Event-based (stimulus – response)
- Object-oriented designs don’t work
  - Too bulky
  - Miss the whole point of the system
- Embedded systems are **TASK** oriented
Challenge: Hardware Details

- Common operations might be expensive
  - Floating point operations
  - Integer multiply, divide
  - Trade memory for cpu (lookup tables)
- Limited I/O channels
- Limited interrupt channels
Real Time Distinctions

- **Hard real time**
  - There are physical consequences for not meeting deadlines
  - Planes crash, cars crash, fuses blow, tanks overflow, ...

- **Soft real time**
  - The results might not be as accurate or convenient
  - GPS in the phone
  - Flashing rate on lights
  - Video

- Applications are often a combination of these
  - And non-real time tasks
Real Time Challenges

- Meeting timing requirements
  - Determining maximum run time of code
    - Over all possible executions
    - Including interrupts, external I/O, ...
    - Memory caching, hyperthreading, pipelining
  - Avoiding unpredictable algorithms & structures
    - Hash tables, memory allocation, quicksort

- Done in terms of TASKS
Real Time Challenges

- Scheduling multiple tasks
  - NP-complete problem (optimal)
- Ensuring predictable software
  - Modeling program behavior
  - Modeling timing issues
  - Formal verification of the software
  - Formal verification of the timing
Example: Clapper

- **Input:** microphone
- **Output:** LED lights
- **Computer:** Arduino
  - CPU, memory, I/O ports
  - Digital and analog inputs
  - Serial inputs (UART) to computer
- **Program**
  - Check for input
  - Drive output
  - Clapper logic
Clapper: Input Issues

- What does sound input look like
- Different frequencies (whistler)
- Threshold of loudness (clapper)
  - What is the threshold
  - Continuous loud noises
  - Interim noises
- Sampling rate
Clapper: Other Issues

- Handling output lights
  - Flashing on and off at the right times
- Handling logic
  - Handling multiple claps
- We’ll go over this next week
Homework for Monday

- Project Idea Presentations
  - 2-8 minutes each (depends on group size)
  - Pitch your project
  - Recruit others if appropriate
  - Elevator talks

Lecture 2: Introduction