1 Introduction

1.1 Embedded Systems

1. Embedded Systems are virtually everywhere
2. Here are some examples
3. Embedded systems are characterizes by interacting directly with hardware. But, the biggest difference is.....:

1.2 Timing

1. Timing!
2. For embedded systems, how the system performs within timing constraints also determines its correctness.
3. For that reason, embedded systems are often a subset of ”real-time” software systems.

1.3 Definition

1. Point out whatever you think is interesting about the definitions.

1.4 Differences

1. Again, point out any of the interesting points.

2 Design

2.1 Bottom Up

1. Often, sometimes the hardware is being developed in concert with the software.
2. Since timing is everything, low-level details must be considered early in the design process.

2.2 Stimulus-Response

1. General approach is stimulus/response model. Since embedded systems have to react to events in environment.
2. Behavior defined by Stimulus (events in environment which causes system to act)
3. Response (signal sent by SW to environment - say to actuators in response to a signal)
4. real-time-systems must react to stimuli at any given time (both periodic and aperiodic)
5. Aperiodic stimuli are often handled by interrupts.

2.3 Design Activities

1. Platform selection depends on timing, power, costs, developer experience

2.4 Design Validation

1. Need to check design (just like any other program)
2. Have to ensure timing constraints are met as well. Can do:
3. Static analysis - prior knowledge or well defined system behavior required.
4. Real-time system modelling (state models often used)
5. Can now go into a bunch of stuff about state models

3 Real Time Programming

3.1 Tools

1. Programming real time systems is very difficult.
2. Assembly is sometimes used, but is slow development, and few developers with enough expertise.
3. C is good, but does not have native support for concurrancy. System calls need to be well defined in order to rely on timing.
4. OO languages have very high overhead. Especially languages which have automatic garbage collection. The real-time developer cannot have processes he is unaware of taking over timing.

3.2 Real-Time OS

1. Bare metal systems require a development team to have a large array of system calls, can be cumbersome to larger the system.
2. Real-time operating systems have been developed, RTOS for linux, windows has one, a number of other commercial systems are also in place.

3.3 Scheduling

1. The times by which stimuli must be processed and some response produced by the systems.
2. If the system does not meet a deadline, then that would be a failure for a hard RTOS.
3. Frequency: The number of times per second that a process must execute so that you are confident that it can always meet its deadlines.
4. Execution time. Time required to process a stimulus and produce a response. Need to keep track of average and worst execution times.
5. Now must develop a scheduling system that will ensure that a process will always be scheduled to meet its deadlines. RTOS needs to support the scheduling algorithm developed.