Introduction to Real-Time Systems

Embedded Real-Time Software
Lecture 1
• **Instructors:**
  – Dongsoo S. Kim
  – Office: Room 83345 (031-299-4642)
  – E-mail: dskim@iupui.edu
  – Office Hours: by appointment
Syllabus (cont’d)

• Primary textbook
  – Real-Time Systems, Jane W. S. Liu

• Supplementary
  – MicroC/OS-II The Real-Time Kernel 2nd Ed., Jean J. Labrosse

• Grading policy (subject to change)
  – Attendance: 5% (I’ll call the roll at random)
  – Midterm exam: 25%
  – Final exam: 40%
  – Project & Assignment: 30%
Project & Assignments

- uC/OS-II Porting on MRP-S3C2440 Board
- To attach a sensor to the board
- To develop an application using the board and sensor
- Cheating on tests and other assignments will not be tolerated and will result in the automatic zero point for the test!
Project & Assignments

- To port uC/OS-II on DEMO9S12NE64E
- To attach a sensor to the board
- To develop an application using the board and sensor
Aims of This Module

• To introduce and explore programming language and operating systems facilities essential to the implementation of real-time, reactive, embedded and networked systems.

• To provide students with understanding of the practical engineering issues raised by the design and programming of real-time, reactive, embedded and networked systems.
Intended Learning Outcomes

• By the end of this module participants should be able to:
  – Clearly differentiate the different issues that arise in designing soft and hard real-time, concurrent, reactive, safety-critical and embedded systems.
  – Explain the various concepts of time that arise in real-time systems.
  – Analyze and apply a variety of static and dynamic scheduling mechanisms suitable for soft and hard real-time systems. Conduct simple performance and schedulability analysis to demonstrate that a system can successfully meet real-time constraints.
  – Explain the additional problems that arise in developing distributed and networked real-time systems.
  – Describe the design and implementation of systems that support real-time applications. Justify and critique facilities provided by real-time operating systems and networks.
  – Design, construct and analyze a small, concurrent, reactive, real-time system. Select and use appropriate engineering techniques, and explain the effect of your design decisions on the behavior of the system.
Prerequisites

• Students are expected to have done degree-level studies in, and be familiar with operating systems design and implementation, concurrency and threaded programming, and software analysis and design.
Module Outline

• **Introduction to Real-Time and Embedded Systems**
  – Reference Model
  – Hard versus soft real-time

• **Job Scheduling**
  – Clock driven scheduling algorithms
  – Priority driven scheduling algorithms
  – Schedulers in commodity and real-time operating systems

• **Resource access control**
  – Algorithms
  – Implementation

• **Real-time communication**
  – On best-effort networks
  – Enhanced quality of service

• **Other implementation considerations**
Real-Time and Embedded Systems

• A real-time system must deliver services in a timely manner
  – Not necessarily fast, but must meet some timing deadline
• An embedded system is hidden from view within a larger system
• Many real-time and embedded systems exist, often without the awareness of their users
  – Washing machine, photocopier, mobile phone, car, aircraft, industrial plant, microwave oven, toothbrush, CD player, medical devices, etc.
• Must be able to validate real-time systems for correctness
  – Some embedded real-time systems are safety critical – i.e. if they do not complete on a timely basis, serious consequences result
  – Bugs in embedded real-time systems are often difficult or expensive to fix
This module will discuss several representative classes of realtime and embedded system:

- Digital process control
- Higher-level command and control
- Tracking and signal processing
- Real-time databases
- Telephony and multimedia

Algorithms for scheduling tasks such that those systems complete in a reliable and timely fashion

Implementation techniques, operating systems and languages for building such systems
Digital Process Control

- Controlling some device (the “plant”) using an actuator, based on sampled sensor data
  - $y(t)$ is the measured state of the plant
  - $r(t)$ is the desired state of the plant
  - Calculate control output $u(t)$ as a function of $y(t)$ and $r(t)$, or $e(t) = r(t) - y(t)$. 

![Diagram of digital process control system](image)
Digital Process Control

• **Pseudo-code for the controller as an infinite timed loop:**

```plaintext
set timer to interrupt periodically with period T;
at each timer interrupt, do
  do analogue-to-digital conversion of \( y(t) \) to get \( y_k \);
  compute control output \( u_k \) based on reference \( r_k \) and \( y_k \);
  do digital-to-analogue conversion of \( u_k \) to get \( u(t) \);
end do;
```

• **Effective control of the plant depends on:**
  – The correct control law computation and reference input
  – The accuracy of the sensor measurements:
    • Resolution of the sampled data (i.e. bits per sample)
    • Timing of the clock interrupts (i.e. samples per second, \( 1/T \))
Digital Process Control

• The time $T$ between any two consecutive measurement of $y(t)$ and $r(t)$ is the sampling period
  – Small $T$ better approximates the analogue behavior
  – Large $T$ means less processor-time demands
  – Must achieve a compromise

• If $T$ is too large, oscillation will result as the system tries to adapt
• **How to choose sampling period?**
  – *Rise time* – the amount of time that the plant takes to reach some small neighborhood around the final state in response to a step change in the reference input
  – If $R$ is the rise time, and $T$ is the period, a good rule of thumb is that the ratio $10 \leq \frac{R}{T} \leq 20$

• **Must be chosen correctly, and accurately implemented to ensure stability**

• **Multi-rate systems** – system is composed of multiple sensors and actuators, each of which require different sampling periods
  – Need to run multiple control loops at once, accurately
  – Usually best to have the sampling periods for the different degrees of freedom related in a harmonic way
Example: Helicopter Flight Control

Do the following in each 1/180-second cycle:

- Validate sensor data and select data source; on failure reconfigure the system

- Do the following 30-Hz avionics tasks, each once every 6 cycles:
  - Keyboard input and mode selection
  - Data normalization and coordinate transformation
  - Tracking reference update

- Do the following 30-Hz computations, each once every 6 cycles
  - Control laws of the outer pitch-control loop
  - Control laws of the outer roll-control loop
  - Control laws of the outer yaw- and collective-control loop
Example: Helicopter Flight Control

• Do each of the following 90-Hz computations once every 2 cycles, using outputs produced by the 30-Hz computations
  – Control laws of the inner pitch-control loop
  – Control laws of the inner roll- and collective-control loop

• Compute the control laws of the inner yaw-control loop, using outputs from the 90-Hz computations

• Output commands to control surfaces

• Carry out built-in-test
Higher-Level Control

- **Controllers often organized in a hierarchy**
  - Multiple control loops, higher level controllers monitoring the behavior of low-level controllers
  - Time-scale, complexity of decision making, increases as go up hierarchy; Move from control to planning
  - Higher level planning must still be done in real-time, although deadlines are less tight
Real-Time Communications

- Real-time systems are increasingly distributed, including communication networks
  - Control loop may include a communication step
  - System may depend on network stimuli

- Not only does a system need to run a control law with time constraints, it must also schedule communications, sending and receiving messages according to deadlines
Example: Drive by Wire

- **All data must be delivered reliably**
  - Bad if you turn the steering wheel, and nothing happens
- **Commands from control system have highest priority, then sensors and actuators, then control inputs**
  - Anti-lock brakes have a faster response time than the driver, so prioritise to ensure the car doesn’t skid
- **Network must schedule and prioritise communications**
Example: Packet Voice

- **Voice is digitized and sent as a sequence of packets**
  - Constant spacing, every 10-30ms depending on codec
- **Strict timeliness requirement**
  - Mouth to ear delay needs to be less than approximately 150ms
  - Packets must be played out with equal spacing
- **Relaxed reliability requirement**
  - Some small fraction of packets can be lost, and just sound like crackles on the wire; most need to arrive
- **Emergency calls may have priority**
Types of Real-Time Application

- **Purely cyclic**
  - Every task executes periodically
  - Demands in (computing, communication, and storage) resources do not vary significantly from period to period
  - Example: most digital controllers and real-time monitors

- **Mostly cyclic**
  - Most tasks execute periodically
  - The system must also respond to some external events (fault recovery and external commands) asynchronously
  - Example: modern avionics and process control systems

- **Asynchronous: mostly predictable**
  - Most tasks are not periodic
  - The time between consecutive executions of a task may vary considerably, or the variations in resource utilization in different periods may be large
  - These variations have either bounded ranges or known statistics

- **Asynchronous: unpredictable**
  - Applications that react to asynchronous events and have tasks with high run-time complexity
  - Example: intelligent real-time control systems
Types of Real-Time Application

• As we will see later, the type of application affects how we schedule tasks, prove correctness.

• It is easier to reason about applications that are more cyclic, synchronous and predictable:
  – Many real-time systems designed in this manner
  – Safe, conservative, design approach, if it works for your application
Implementation Considerations

- Some real-time embedded systems are complex, implemented on high-performance hardware
  - Industrial plant control
  - Civilian flight control
- Many must be implemented on hardware chosen to be low cost, low power, light-weight and robust; with performance a distant concern
  - Military flight control, space craft control
  - Consumer goods
- Often-times implemented in C or assembler, fitting within a few kilobytes of memory
  - Correctness a primary concern, efficiency a close second
Summary

• Outline of the module structure, assessment, etc.
• Introduction to real-time and embedded systems
  – Examples of digital control, higher-level control, communication
• Types of real-time system
  – Cyclic synchronous vs. asynchronous and unpredictable
• Implementation considerations