CSE 438/598 Embedded Systems Programming

Computer Science & Engineering Department
Arizona State University
Tempe, AZ 85287

Dr. Yann-Hang Lee
yhlee@asu.edu
(480) 727-7507
Course Syllabus (1)

Course Goals:
- Understand the design issues of embedded software and gain an in-depth knowledge of development and execution environment.
- Understand the functions and the internal structure of device interfaces, drivers, and real-time operating systems.
- Acquire the skill to develop multi-threaded embedded software in target environment.
- Develop feasible task scheduling and carry out system performance and task schedulability analyses.

Pre-requisites:
- Assembly language and computer organization (CSE230), microprocessor interfaces (CSE 325), and experience of C programming language.
Course Syllabus (2)

- **Major topics covered:**
  - Introduction: characteristics of embedded applications (1 lecture)
  - Intel embedded processor architecture (3 lectures)
  - PCI Express, device controllers, and programming approaches. (4 lectures)
  - Device driver: software structure of device driver, Linux loadable kernel module, blocking and non-blocking IO, mmp and DMA, block drivers, top-half and bottom-half ISR. (5 lectures)
  - Embedded software and thread programming: task model and specification, event loop, coroutine, never-ending tasks, periodic and aperiodic tasks, thread synchronization, inter-task communication. (6 lectures)
  - Scheduling algorithms and analysis. (6 lectures)
Course Syllabus (3)

- **Office hours** – 1:15-3:00pm, Monday and Wednesday

- **Evaluation**
  - Midterm exams (20%) (during the class periods on Oct. 10, 17 or 22)
  - Project assignments (50%)
  - Final exam (30%) (during the final exam period scheduled by the University, i.e. 12:10 - 2:00 PM, Monday, Dec. 17)

- **Exams are open book, open note, and laptop PC is allowed.**
Real-time Embedded Systems

- **Embedded system**
  - the software and hardware component that is an essential part of another system

- **Real-time system**
  - provide well-timed computation
  - deadlines, jitters, periodicity
  - temporal dependency

![Diagram of real-time control system]

A/D

Control-raw computation

D/A

Controller

Reference

input

sensor

Plant

actuator
Embedded Systems -- Examples
Emerging Embedded Systems
Embedded Systems

- They are everywhere
- Number of embedded processors in your home??
- What are they?

**Hardware (chips) + Software (program)**

- CPU processor (ARM, PowerPC, Xscale/SA, 68K)
- Memory (256Mbtye)
- Input/output interfaces (parallel and serial ports)
Hardware Platform

- **Organization**
  - buses to connect components – PCI, ISA, PC104+

- **Package**
  - standard chips on PC
  - processor + ASIC
  - SOC

---

Real-Time Systems Lab, Computer Science and Engineering, ASU
SW Development for RT ES

- **To write the control software for a smart washer**
  - initialize
  - read keypad or control knob
  - read sensors
  - take an action

- **System current state**
  - state transition diagram
  - external triggers via polling or ISR

- **If there are multiple triggers and external conditions – single or multiple control loops**
Periodic Tasks

- **Invoke computation periodically**
  - Adjust pressure valves at a 20 Hz rate

```
Task initialization
(set up periodic timer interrupts)
```

```
wait for the interrupt event
```

```
computation
```

```
start_time = time()
```

```
computation
```

```
Sleep(period - (time() - start_time))
```
SW Development for RT ES

- **In the example of smart washer**
  - Never-ending in a single control loop
  - Single execution threat and one address space
  - Event triggering and state transitions
  - Small memory footprint

- **What are missing:**
  - no concurrency (real-world events occur in parallel)
  - no explicit timing control (let’s add a timer)
  - difficult to develop and maintain large embedded systems – verifiable, reusable, and maintainable
RT ES vs. General Software

- Multi-tasking for concurrent events
- Machine dependence and portability
- Software abstraction, modular design
  - information hiding, OO, separate compilation, reusable
  - a sorting procedure -- function, input, output specification
- Control timing
  - predictable actions in response to external stimuli
  - deadline (absolute or relative), and jitter
- Resource constraints and sharing
  - CPU time, stack, memory, and bandwidth
- Scheduling
Timing Constraints and Multi-threading

- Given input $x_1$ at time $t_1$, produce output $y_1$ at time $t_2$
- Non-deterministic operation, Time-dependent behavior, and race condition
  - difficult to model, analyze, test, and re-produce.
- Example: NASA Pathfinder spacecraft
  - Total system resets in Mars Pathfinder
  - An overrun of data collection task →
    - a priority inversion in mutex semaphore →
    - failure of communication task →
    - a system reset.
  - Took 18 hours to reproduce the failure
    in a lab replica → the problem became obvious and a fix was installed
Trends of RT Embedded Systems Applications

- **Wide-spreading, distributed, connected, and heterogeneous**
  - The average new car has a dozen microprocessors in it. The Mercedes S-class has 65.

- **Mission and safety critical**

- **High-end consumer products**
  - cell phone, HDTV, home network, PDA, GPS-based location system

- **Quality of the products**
  - portable/reusable, reliable/dependable, interoperable, predictable (schedulable), and secured

- **Software extensive**
  - A typical cell phone is made with more than 3 million lines of code
Building RT Embedded Systems

- Advances in general-purpose computers
  - PCs are powerful, cheap, and versatile
  - Information processing is ubiquitous
- Thanks for the increase in productivity

Graph:
- Process technology: +58%
- Hardware design productivity: +21%
- Software productivity: +8%
Embedded Software

- **Characteristics**
  - Concurrent, time dependent, and device/environment dependent

- **Embedded software development**
  - 80% programs in embedded system is with C/C++ and 15% in assembly
  - the same thing that has been done more than 30 years (Ada?)

- **Software complexities**
  - inherent and cannot be eliminated, i.e. algorithm, concurrency, etc.
  - accidental (due to technology or methods used), i.e. memory leaks

- **What can we do?**
  - **abstraction** (e.g. high-level languages, modeling)
  - **automation** (e.g. compiler)
Embedded System Development

- Need a real-time (embedded) operating system?
- Need a development and test environment?
  - Use the host to edit, compile, and build application programs, and configure the target
  - At the target embedded system, use tools to load, execute, debug, and monitor (performance and timing)
From Source to Executable

- Compiler, linker, and loader
- In ELF: executable, relocatable, shared library, and core
  - information for relocation, symbol, debugging
  - linker resolves symbol reference
- Link script or link command file
  - assigns absolute memory addresses (program area, static data, bss, stack, vector table, etc.)
- Startup code to disable interrupts, initialize stack, data, zero uninitialized data area, and call main().
Target Server at Development Host

- The target server (tgtsvr) manages the interactions with the target
  - communication channel
  - symbol table for the target

- Tools use WTX (Wind River Tool Exchange) protocol to communicate with the target server
Debugger

- Observability, real-time analysis, and run control
- At host: GUI, source code, symbol table, type information, line number
- Communication to the target
  - serial port, Ethernet, USB, etc.
  - read/write memory and registers
  - control processor execution (single step, breakpoint, watchpoint, etc.)
- At target:
  - debugging stub (software): at breakpoint, replace the instruction with breakpoint inst. or invalid instruction to trigger exception
  - BDM (JTAG interface): hardware breakpoint, trace buffer, flash programming
Debugging Embedded Systems

- Visibility
- The symptom is the result from several bugs
- The bug and symptom are widely separated in space or time
- Mistaken assumptions or unexpected events
- The bug is from other objects (e.g., a library, hardware, OS, or compiler)
- The system is nondeterministic
- and ....
CSE 325 – Course Syllabus

- **Major topics covered:**
  - Introduction and review of instruction set and assembly language programming (3 lectures)
  - Interfacing between C and assembly languages (2 lectures)
  - Coldfire processor and IO multiplexing (2 lecture)
  - Interrupt and exception (2 lectures)
  - Timers and counters (2 lectures)
  - Serial communication: UART, SPI, and I2C (5 lectures)
  - Parallel I/O interface and signal handshaking (2 lectures)
  - Keyboards and LCD (3 lectures)
  - A/D-D/A converters (2 lectures)
  - Memory devices, SRAM, DRAM, flash memory, and SDRAM controller (4 lectures)
Target Environment

- Freescale Project Board Student Learning Kit and Coldfire 5211SLK

Development Software
- CodeWarrior IDE (editor, compiler, assembler, debugger, etc.)
CSE 325 – Example Assignments (1)

- Configuring the boards, creating a project, compiling and debugging, ELF files
- Board initialization, program execution, LCF and memory map, and utility routines (clock initialization, delay, and etc.), in-line assembly
- Event driven embedded software with interrupt, pad signal configuration, PWM for fan speed control, and pulse width measurement for fan speed monitoring.
CSE 325 – Example Assignments (2)

- SPI bus and dot matrix display
- ISR-driven serial communication using UART
- I2C interfacing with Wii Nunchuk
- Integration and memory optimization