CSE 466 – Software for Embedded Systems

- Instructors:
  - Gaetano Borriello
    - CSE 572, Hours: T 10:00-11:00, W 1:30-2:30
    - cse466-instructor@cs.washington.edu
  - Joshua Smith – Intel Research Seattle
    - TBD
    - cse466-instructor@cs.washington.edu
- Teaching Assistants:
  - Fabian Kidarsa and Brian Mayton
    - CSE 003, Hours TTh 2:30-4:30
    - cse466-tas@cs.washington.edu

CSE 466 – Software for Embedded Systems

- Class Meeting Times and Location:
  - Lectures: EEB 003, MWF 12:30-1:20
  - Lab: CSE 003, T – Section A, 2:30-5:20
    Th – Section B, 2:30-5:20
- Exams
  - Final demo: Friday, 14 March, CSE Atrium, 12:30-1:20
  - Exam-I: Friday, 11 February, EEB 003, 12:30-1:20
  - Exam-II: take home during finals week (but only 1 hr)
- Course Evaluation
  - Wednesday, 12 March, EEB 003, 12:30-1:20

Embedded system – from the web

- Definitions
  - A device not independently programmable by the user.
  - Specialized computing devices that are not deployed as general purpose computers.
  - A specialized computer system which is dedicated to a specific task.
  - An embedded system is preprogrammed to perform a narrow range of functions with minimal end user or operator intervention.
- What it is made of
  - Embedded systems range in size from a single processing board to systems with operating systems.
  - A combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function.
  - In some cases, embedded systems are part of a larger system or product, as is the case of an anti-lock braking system in a car.
  - A specialized computer system that is part of a larger system or machine.
  - Typically, an embedded system is housed on a single microprocessor board with the programs stored in ROM.
  - Some embedded systems include an operating system, but many are so small and specialized that the entire logic can be implemented as a single program.
- Examples
  - Virtually all appliances that have a digital interface -- watches, microwaves, VCRs, cars -- utilize embedded systems.
  - A computer system dedicated to controlling some non-computing hardware, like a washing machine, a car engine or a missile.
  - Examples of embedded systems are medical equipment and manufacturing equipment.
  - While most consumers aren't aware that they exist, they are extremely common, ranging from industrial systems to VCRs and many net devices.

What is an embedded system?

- Different than a desktop system
  - Fixed or semi-fixed functionality (not user programmable)
  - Different human interfaces than screen, keyboard, mouse, audio
  - Usually has sensors and actuators for interface to physical world
  - May have stringent real-time requirements
- It may:
  - Replace discrete logic circuits
  - Replace analog circuits
  - Provide feature implementation path
  - Make maintenance easier
  - Protect intellectual property
  - Improve mechanical performance

What do these differences imply?

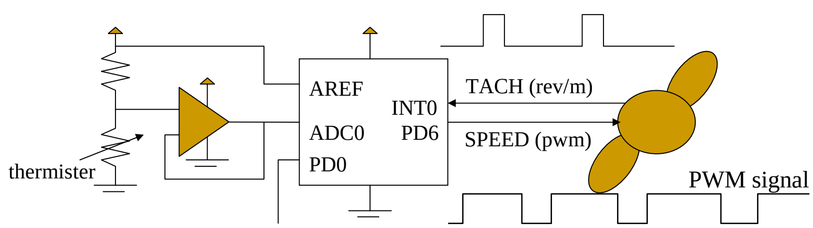
- Less emphasis on
  - Graphical user interface
  - Dynamic linking and loading
  - Virtual memory, protection modes
  - Disks and file systems
  - Processes
- More emphasis on
  - Real-time support, interrupts (very small OS, if we’re lucky)
  - Tasks (threads)
  - Task communication primitives
  - General-purpose input/output
  - Analog-digital/digital-analog converters
  - Timers
  - Event capture
  - Pulse-width modulation
  - Inter-device communication methods and protocols

Examples of embedded systems

What is an embedded system? (cont'd)

- Figures of merit for embedded systems
  - Reliability – it should never crash, or crash "safely"
  - Safety – controls things that move and can harm/kill a person
  - Power consumption – may run on limited power supply
  - Cost – design cost, manufacturing cost, service cost
  - Product life cycle – maintainability, upgradeability, serviceability
  - Performance – real-time requirements, power budget

Example: a temperature controller



Task: Tachometer (external interrupt)
```
now = getTime();
period = then - now; //overflow?
then = now;
return;
```

Task: FanPWM (periodic, hard constraint)
```
count++;
if (count == 0) PD6 = 1;
if (count > duty_cycle) PD6 = 0;
if (count == 255) count = 0;
return;
```

Task: TempControl (periodic, soft constraint)
```
temp = getTemp();
if  (temp > setpoint) duty_cycle++;
if  (temp < setpoint) duty_cycle--;
if  (period<min || period>max) PD0 = 1;
```

Task: Main
```
Thi = 0;
setup timer for 1ms interrupt; // fan
setup timer for 100ms interrupt; // temp
while (1) ;
```

Capacity

- Assume:
  - 8 MHz processor @ one instruction/cycle
  - Assume fan runs between 30Hz and 60Hz
  - Assume 256ms period on speed control PWM, with 1ms resolution.
- What percent of the the available cycles are used for the temperature controller?
  - [total instructions in one second] / (8MInstr/sec)
- How much RAM do you need?
- How much ROM?

Resource analysis of temp controller

Task: Tachometer (external interrupt)

```
now = getTime();
period = then - now; //overflow?
then = now;
return;
```

Task: FanPWM (periodic, hard constraint)

```
count++;
if (count == 0) PD6 = 1;
if (count > duty_cycle) PD6 = 0;
if (count == 255) count = 0;
return;
```

Task: TempControl (periodic, soft constraint)

```
temp = getTemp();
if  (temp > setpoint) duty_cycle++;
if  (temp < setpoint) duty_cycle--;
if  (period<min || period>max) PD0 = 1;
```

Task: Main

```
duty_cycle = 0;
setup timer for 1ms interrupt; // fan
setup timer for 100ms interrupt; // temp
while (1) ;
```

Task	ROM	RAM	Instructions/Sec
Tach	~4	3 (period, then, now)	4 * 60 = 240
FanPWM	~10	1 (count)	10 * 1000 = 10000
TempControl	~10	2 (temp, duty_cycle)	10 * 10 = 100

Total Instructions/Sec = 10340, at 8MIPS, that's only 0.13% utilization!
Other resources? global and static variables, stack

What Are You Going to Learn?

- Hardware
  - I/O, memory, busses, devices, control logic, interfacing hw to sw
- Software
  - Lots of C and assembly, device drivers, low level OS issues
  - Concurrency
- Software/Hardware interactions
  - Where to put functionality
    - Hardware or software
    - Software functions: threads, interrupt handlers
  - What are the costs
    - performance
    - memory requirements (RAM and/or ROM)
  - How to communicate
    - shared memory
    - encoding of information

What are you going to learn? (cont'd)

- Understanding of basic microcontroller architecture
- Understanding of interfacing techniques
- Appreciation of power management methods
- Understanding of basic communication methods and protocols
- Facility with a complete set of tools for design/debug
- Experience implementing some real systems

Class logistics – see course web

- http://www.cs.washington.edu/466
  - expands to the current quarter http://www.cs.washington.edu/education/courses/cse466/08wi/
- Class structure
- Grading
- Syllabus
- What we'll be doing

Class structure

- Lecture (26)
  - Closely linked to laboratory assignments
  - Cover main concepts, introduced laboratory problems
- Lab (8)
  - Implementation of two projects
  - Lab demos/reports due within 30 minutes of start of next lab section
- Exams (2 – one in-class, one take-home)
  - Based on lectures, labs, and reading assignments
- Final demo (1 – in atrium)
  - During last scheduled meeting – participation required
- Reading and source material (medium amount)
  - Some assigned, most you'll find on your own

Other Matters

- Lecture slides will be on line after class (links in several places)
- Random lab partner assignments, changed mid-quarter
- Sign up for CSE466 mailing list

Grading

- Lab reports:
  - Demonstration(s) required
  - Brief answers to questions embedded in assignment
  - Sometimes hand-in code
  - Do with your partner
- Distribution:
  - Labs: 40%
  - Exams: 30% (Monday, 11 Feb and take-home during finals week)
  - Demo: 10%
  - Class and Lab Participation: 20%

CSE466 Lab Projects

- Two multi-week projects
  - Four lab assignments each
  - Different lab partners
- First project
  - Familiarize with microcontroller
  - Learn how to interface devices to it
  - Build your own sensor
  - Test and debug
  - Basic communication between devices as well as devices and PC
- Second project
  - Wireless communication (of two flavors: RF and electric field)
  - Embedded operating system
  - Real-time issues
  - Test and debug
  - Coordinated behavior among multiple devices

CSE466 Lab Projects (cont'd)

- Project 1 – a basic USB device
  - Platform: ATmega16 AVR microcontroller
  - Detect varying capacitance from proximity to hand
  - RGB tri-color LED to display value
  - Connects sensor to PC (over USB) where calculation/averaging is performed
  - Result is communicated back (over USB) to microcontroller and displayed on LED
  - Summary: LED pulse rate and color changes to match sensed capacitance
    - Sensor detects changes in capacitance due to presence/position of hand
    - Microcontroller interfaces to sensor converting readings to values
    - Communicates to PC to get smoothed value
    - Pulse-width modulation of multi-color LED to produce appropriate color
- Past: heart-rate sensor using LED and light sensor

CSE466 Lab Projects (cont'd)

- Project 2 – controller for multi-player soccer
  - Platform: Intel iMote2 wireless sensor nodes ("motes")
  - XScale processor + 802.15.4 radio + Linux
  - Custom UW board w/ LCD, camera, USB, mic, speaker, jog dial, capacitance sensor, etc.
  - "Cell phone" functionality (except GSM/GPRS)
  - Air joystick (using capacitance sensor) to control player movement
- Past: accelerometer to control via tilting

Our platform (iMote2 + sensors + SuperBird)