Introduction to Embedded Systems

CSE 237A

Tajana Simunic Rosing
Department of Computer Science and Engineering
UCSD
What are embedded systems?

- Systems which use computation to perform a specific function
- Embedded within a larger device and environment
- Heterogeneous & reactive to environment

Main reason for buying is **not** information processing
Embedded System Design

Hardware components

Hardware

Software Components

Verification and Validation
Welcome to CSE 237A!

- Instructor: Prof. Tajana Simunic Rosing
  - Email: tajana-at-ucsd.edu; put CSE237a in subject line
  - Office Hours: TTh 9:45-10:45am, CSE 2118

- Admin: Arlene C. Lopez
  - Email: aclopez@eng.ucsd.edu
  - Office: CSE 2274
  - Phone: (858) 822-1516

- TA: Jug Venkatesh
  - Email: jvenkate@eng.ucsd.edu
  - Office hrs: TBD

- Class Website: http://cseweb.ucsd.edu/classes/wi16/cse237A-a/index.html
- Grades: http://ted.ucsd.edu
- Discussion board: https://piazza.com/ucsd/winter2016/cse237a
About This Course

- Part of a four course group
  - CSE 237A: Introduction to Embedded Systems
  - CSE 237B: Software for Embedded Systems
  - CSE 237C: Hardware for Embedded Systems
  - CSE 237D: Embedded Systems Design

- Depth sequence:
  - Embedded Systems and Software
Course Objectives

- Develop an understanding of the technologies behind the embedded computing systems
  - technology capabilities and limitations of the hardware, software components
  - methods to evaluate design tradeoffs between different technology choices.
  - design methodologies

- Overview of a few exciting research topics in embedded systems

- For more details, see the schedule on the webpage
Course Requirements

- No official graduate course as prerequisite

- Knowledge
  - Digital hardware, computer architecture (ISA, organization), programming & systems programming, algorithms

- Skills
  - Advanced ability to program
    - We will be developing a loadable kernel module
  - Ability to look up references and track down pubs (Xplore etc)
  - Ability to communicate your ideas (demos, presentations, reports)

- Initiative
  - Open-ended problems with no single answer requiring thinking and research

- Interest 😊
Textbook &Assigned Reading

- Recommended text:
  - By Peter Marwedel

- A set of papers will be required reading
  - will relate to the core topic of that class
  - you are expected to read it BEFORE the class

- There are additional pointers to papers and web resources on the class website
Course Grading

- Homework (~3): 8%

- Embedded systems projects: 45%
  - Individual project 50%:
    - Energy-efficient real-time sensing app scheduler
      - Part 1: 35%:
        - Compile the kernel, create a loadable kernel module, characterize the workloads running on the hardware
      - Part 2: 65%:
        - Add HW sensors to the board and implement a real-time scheduler, adjust the scheduler so it uses DVFS settings on the board
  - Team project 50%:
    - Topic of your choice, needs to have significant HW & SW components
      - Proposal: 5%
      - Progress Report: 25%
      - Final demo and report: 70%

- Exam: 45%

- Class participation, attendance, engagement: 2%
  - Come prepared to discuss the assigned paper(s)
Previous Years’ Final Projects

- Face recognition application
  - Interfaced with multiple webcams in the environment
- Time and Location Sensitive Messaging Protocol for Automated Message Delivery
  - Used laser interface for message delivery
- Comparison of Ad-hoc Routing Algorithms
  - Algorithms run on multiple sensor nodes
- Context-aware energy management for healthcare applications
  - Used body worn sensors plus phone for managing energy
- User experience-aware non-volatile memory management
  - How to leverage STT-RAM and PCRAM for responsive user applications
- Ambient-aware thermal management
  - Context & sensors around the phone, and thermal sensors within the phone
Project has been designed by:
Yeseong Kim, Jinseok Yang, Jug Venkatesh and Tajana S. Rosing

- Raspbian – Linux variant
- Build and deploy a custom version onto our Raspberry Pi 2 that extends default functionality
- Collect metrics about workloads running on device
- Create a real-time scheduler to allow time-constrained processes to execute successfully
Individual Project Part 1

- Custom Raspbian build:
  - Enable performance counters to be used to capture memory and processor (CPU) metrics

- Familiarization with build processes

- Cross-compile code for Raspberry Pi
  - Default compilation is for x86-64 (amd64 architecture)
  - Goal is to compile programs for ARMv7 architecture
Individual Project Part 1

- Simple program execution vs. **kernel modules**
  - Programs in kernel space that extend underlying functionality

- Build and compile kernel modules
  - Simple module
  - Main goal: Execute assembly code to manipulate PC registers

- Analyze program behavior
  - Classify programs based on performance counter (PC) results
Individual Project Part 2: Sensors

- Raspberry PI 2 has a few onboard sensors
  - Temperature sensors for CPU cores

- External sensors and actuators
  - Can add multiple digital sensors via general-purpose input/output (GPIO)
    - ADC required to use analog sensors
  - WiringPi: Library to easily use GPIO
  - Programmable with Python, C, and C++
  - Many sensors available
  - You will integrate with Raspberry PI 2: button, LEDs, buzzer, shock, tracking, temperature

```c
#include <wiringPi.h>
int main (void) {
  wiringPiSetup ();
  pinMode (0, OUTPUT);
  for (;;) {
    digitalWrite (0, HIGH); delay (500);
    digitalWrite (0, LOW); delay (500);
  }
  return 0;
}
```
Individual Project Part 2: Scheduler

- Turn heterogeneous programs from a collection of sequential processes into a schedule by creating a real-time *earliest-deadline first (EDF) scheduler*
- Make EDF implementation energy-efficient by leveraging different voltage-frequency settings of Raspberry PI 2
Embedded Systems Development Platforms
Why platforms?

- Embedded systems **operate** in, **interact** with, and **react** to an analog, real-time world
  - Interfacing with this world is not easy or monolithic

- **Sensors**: provide measurements of the outside world
- **Actuators**: provide an output, or a means to modify the physical world
- **Platforms**: everything in between – the framework that allows a system to analyze sensors, process their data, and drive actuators
Platforms

- Infrastructure between sensors and actuators
  - Especially important to embedded systems

- Increased emphasis on embedded software structure and functionality
  - Why?
Embedded Platform Requirements

- Operate within hardware/resource constraints
- Real-time requirements
- Safety/Reliability
- Upgradability
- Limited manual interfacing
- Input/Output (I/O) interface
Microcontrollers

- Low-power, low-capacity System-on-Chip (SoC)
  - Processor core (e.g. ARM7)
  - Memory (Flash ROM + RAM)
  - General-purpose I/O (GPIO)
- Intended for *sequential control* rather than general computation

- Interrupts for I/O handling, preemption
- Programmable interval timer
- Tools to implement models of computation (MoCs)
- “Flash” the controller to upgrade
- Software-driven GPIO + Conversion
Microcontrollers: Development Workflow

- Software development
  - Tools for formal models, models of computation (MoCs)...
  - ...or just plain C!

- Libraries:
  - Input/Output (I/O)
  - Analog-to-Digital conversion (ADC) & Digital-to-Analog conversion (DAC)
  - Basic memory management
  - Interrupts, timing, and constraints

- Compile for microcontroller (MCU)
  - “Flash” the device – load the binary
  - Execute!
Microcontrollers: Arduino

- Atmel AVR microcontroller (ARM on higher-end models)
- I²C bus communication with custom modules – “shields”
- General-purpose Input/Output (GPIO)
- Microcontroller pins exposed with established interface for direct connection

- Merge traditional MCU with an established library system and development environment (IDE)
  - Simplify addressing & control
  - Program in C or C++
  - setup() – initialization
  - loop() – main loop
  - bootloader
General-Purpose
Embedded Processing: ARM

- Designed around embedded use
  - Reduced Instruction Set Architecture (RISC) for reduced size, power, cost

- Specialized digital signal processing (DSP) instructions for I/O

- Multiple CPU modes allow for interrupts, real-time operating system (RTOS) task scheduling and device monitoring

- Conditional execution for nearly every CPU instruction

- Single-instruction Multiple Data (SIMD) extensions for streaming data use
Direct ARM Deployment – ARM SDT

- ARM Software Development Toolkit (SDT) – tools for CL and Windows:
  - C, C++, Thumb compilers, assemblers & linkers
  - Project management software, utilities & debuggers
  - ARMulator – ARM core emulator

- Workflow very similar to that of MCUs, but more tools and libraries to work with:
  - Code optimization
  - Object-oriented programming
  - Memory management
  - ARM big.LITTLE
  - GPU processing
ARM Deployment – Raspberry Pi

- Low-tier ARM SoC (700 MHz)
  - Flash-based persistent storage
  - Onboard 256/512MB RAM
  - GPIO for peripheral connectivity
  - I²C open-pin bus
  - Linux
  - USB-connected Ethernet + USB ports
  - HDMI Out

- Application Development:
  - Python, primary
  - C, C++, Ruby, supported
Using OS for ARM Deployment

- Operating System (OS) takes care of system management – threads, memory & caching, I/O
- Dozens of popular ARM operating systems
  - Linux (including RTLinux)
  - VxWorks (Real-Time Operating System)
  - TinyOS (unofficial)
  - iOS
  - Android
  - Windows RT
  - …and more!
OS-based ARM Deployment: Linux

- Draw upon Linux/Windows/iOS kernel
  - Established libraries for common embedded operations

- Application Framework
  - Develop and deploy feature-rich, connected applications
  - Part of a larger system
OS-based ARM Deployment: Linux Development Process

- **Kernel:**
  - Compile source into installable image
  - Kernel modifications (e.g. enable kernel features)

- **Kernel Module:**
  - Compile C code into installable module (<module_name>.ko)
  - Kernel extensions (e.g. kernel-level operations that use/expose kernel functionality)

- **Applications**
  - Written in any language compilable/interpretable on the machine
  - Run in user space
OS-based ARM Deployment: Raspberry Pi

- Primarily use Linux-kernel-based OS
  - **Raspbian**: Debian based Linux
    - Leading OS for Raspberry Pi
    - Port of Debian Wheezy
    - Useful precompiled software from debian packages e.g., LibreOffice, web browsers
    - Optimized for hard float instructions
  - **Pidora**: Fedora based Linux
  - **Arch Linux ARM**: A light-weight OS, full control to competent users
  - **Puppy Linux**: Run from RAM with a minimal memory footprint
  - **OpenELEC**: Optimized for home media streaming

- Available to use other popular OSs
  - Windows 10, Ubuntu, ...
OS-based ARM Deployment: Android

Foundation: Linux Kernel
**OS-based ARM Deployment: Android**

**Android runtime**: unique system alongside system libraries, OS

**Libraries** for compilation, core Java

**Dalvik Virtual Machine** – Android-optimized Java Virtual Machine
Enables full-fledged Java features (threading, memory mgmt)
OS-based ARM Deployment: Android

Libraries for user (Angry Birds) and core apps (Phone, Browser)

Extensible, customizable resources & interfaces for applications

![Diagram of Android application framework](image-url)
OS-based ARM Deployment: Android
OS-based ARM Deployment: Android vs. Linux

- Trade off overhead, low-level efficiency for high-level feature-richness
- Low-level functionality masked by infrastructure
More about Android: Kernel Layer

- At heart, Android is based on Linux (Version 2.6 and higher)
- Provides OS services:
  - Memory Management
  - Process/thread handling
  - Network stack
  - Drivers – system, hardware devices, sensors, power management
- Abstracts hardware away from software stack
- Provides interface to Android Libraries

- **Kernel-level development is performed here**
  - Code is written in C, and entire OS is either compiled from source, or *cross-compiled* for the device and installed as a module
  - Identical to Linux kernel development
Android OS Layer

- System Libraries – abstracts kernel interface for Application Framework’s application programming interfaces (APIs)

- Kernel-level interfaces are abstracted and extended with developer libraries

- Common Libraries:
  - **SystemC**: C library, optimized for embedded
  - **Media**: playback and decoding of common formats
  - **Surface Manager**: display subsystem, composes multiple implementations of 2D, 3D graphics
  - **LibWebCore**: web browser engine
  - **SGL, 3DLib**: 2D, 3D graphics, respectively
  - **SQLite**: compact database engine – within Android, used for app data storage

- **Library development is performed here**
  - Code is written in C/C++, and OS is either compiled from source
  - Platform designers introduce special libs, hooks to custom sensors, etc.
Android Runtime

- Sits alongside system libraries – *unique* to Android and Java-based operating systems

- Libraries for compilation, core Java libraries

- Dalvik Virtual Machine – Android-specific JVM
  - Interface b/w compiled Java apps and underlying Linux kernel
  - Enables full-fledged Java features (threading, memory management)
The heart of the Android front-end
- Sits on top of JVM and system libraries, provides functionality for both user (Angry Birds, etc.) and core applications (Phone, Browser, etc.)

Contains system resources and interfaces needed for applications, extensible with new, custom libraries
- Enables reusability of reliable modules – the power of Android
- Views for display, Content Providers for data access, Resource Manager for system resources, Notification and Activity Managers

Application development is performed on top of this
- Code is written in Java, using libraries found in the App Framework
Android Application Development

- Individual applications are composed of *Application Components*
  
  - **Activities:** each UI screen + interaction elements (applications can have multiple activities)
  - **Services:** background, long-running processes, with no UI components
  - **Content Providers:** application data storage interface, for file system, SQLite, etc.
  - **Broadcast Receivers:** non-UI listener for system events and broadcast messages