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

Verification and Validation

Software Components

Concept

Specification

HW/SW Partitioning

Software

Hardware

Estimation - Exploration

Design (Synthesis, Layout, ...)

Design (Compilation, ...)

see
Welcome to CSE 237A!

- Instructor: Prof. Tajana Simunic Rosing
  - Email: tajana.teach@eng.ucsd.edu; put CSE237a in subject line
  - Office Hours: TTh 3:30-4:30 PM @ CSE 2118

- Admin: James Martinez
  - Office: CSE 2225
  - Email: jtmartinez@eng.ucsd.edu

- TAs:
  - Yeseong Kim yeko48@eng.ucsd.edu

- Class Website: [http://cseweb.ucsd.edu/classes/wi18/cse237A-a](http://cseweb.ucsd.edu/classes/wi18/cse237A-a)
- Grades: [http://ted.ucsd.edu](http://ted.ucsd.edu)
- Discussion board: [https://piazza.com/ucsd/winter2018/cse237a](https://piazza.com/ucsd/winter2018/cse237a)
About This Course

- Part of a five 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
  - CSE 237E: Principles of Physical Computing

- 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

- Homeworks: 8%
- Embedded systems projects: 45%
  - Individual project 50%:
    - Energy-efficient sensing application 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 task 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% (assigned today)
      - Progress Report: 25%
      - Final demo and report: 70%
- Exam: 45%
- Class participation, attendance, engagement: 2%
  - Come prepared to discuss the assigned paper(s)
Individual Project Overview
Individual Project with Raspberry Pi

- Raspbian – Linux variant

- Build and deploy a custom version onto our Raspberry Pi that extends default functionality

- Collect metrics about workloads running on device

- Create an energy-efficient 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 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++

- You are going to make a simplified LED based music visualizer similar to this: https://www.youtube.com/watch?v=LS_oEj7hTtw
  - Raspberry PI 3 will be integrated with a button, LEDs, sound sensors

```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 task dependency-aware scheduler.

- Make scheduler implementation energy-efficient by leveraging different voltage-frequency settings of Raspberry PI.
CSE237a Final Project Examples & Ideas
Previous Years’ Final Projects

- Driving Dashboard Simulator
- Engine Control Unit
- 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
A comprehensive management framework for power, temperature, reliability, & variability management in individual devices

User Experience Requirements

Comprehensive Management Framework

- Variability
- Reliability
- Temperature
- App perf
- Battery
- Power

System feedback

Online Variability Emulation

Online Reliability Emulation

Control decisions

Target system

LTC: Long Term Controller
STC: Short Term Controller

Average target
Unexploited margins

App perf & Battery

Power

Reliability

Unexploited margins

Temperature

Average target

LTC

STC
HPWREN wireless connectivity covers 20k sq. mile area with numerous sensors

Motion detect cameras

Wildfire tracking cams
Cal Fire Dept.

Environmental sensors & cams

Acoustic sensors
Wolf howls at the CA Wolf Center

HW Braun, F. Vernon, T. S. Rosing, UCSD
Goal: seamless analysis and reuse of context

- Applications process raw input data \(\rightarrow\) output actions
  - Avoids processing redundancy & monolithic applications
- Provide ontological information
  - Ontology provides range, discretization, type of input data

**Our approach:** Replace black-box applications with multi-input single-output functional units – *context engines*

- Composable units allow data reuse & parallelization
- General purpose statistical learning to generate output based on ontological metadata:
  - Classification: Decision Trees, SVM
  - Regression: KNN, SVR, Linear Regression, TESLA
  - Clustering: K-means, D-Stream II
  - Anomaly Detection
- Simple API facilitates development; includes full RSA
Machine Learning over IoT Network Hierarchy

Motivation:

- Leverage computation capabilities throughout the IoT hierarchy
  - Reduces communication costs -> increases battery lifetime
  - Cloud only has access to derived features -> accuracy

Neural Networks fit perfectly!

- Powerful modeling technique
- Can solve many ML problems
- Model structure fits well into a hierarchy

Towards a Solution:

- Optimal compression through local NNs
- Trading width for depth of NN
  - Formulate bounds on accuracy-communication trade off
- On-line adaptation
CitiSense: Wearables & Environmental Sensors

- Sensors and phones given to commuters using various transportation means throughout the greater San Diego area
  - Results found “urban valleys” where buildings trapped pollution
  - Major routes reported contrastingly high/low AQI for the same location
  - Ease of deployment and in-network adaptation is key: learn from context!
Context Engine Example:
Proactive Fire Tracking and SDR-based Communication with Drones

Current state of the art:
Stationary sensors & cameras are placed in high risk areas.

Our goal: Develop a proactive infrastructure that leverages both stationary & drone based sensors to sniff out fires & provide adaptive wireless SDR connectivity to the troops on the ground.
Tracking great apes in the jungle

The largest chimpanzee sanctuary in the world (Tchimpounga) and the only bonobo sanctuary in the world (Lola Ya) need ways to track groups of animals in enclosures/islands while in the jungle to learn information about them for future release in the wild.
Goals for tracking the great apes

The main things that we would like to track: group dynamics with special attention to the proximity between individuals during the day (dominant male, etc). Group distribution in the release area in agreement with the availability of resources (trees that they use for feeding). Visual Health status

3 islands: 100, 60 and 30 hectares
Release site: 20,000 hectares
Individuals per island: 50, 30 and 20
For the first time in history, we will longitudinally study the impact that daily habits, the environment, genetics, and the microbiome have on older adult cognition.

Goal: We will model the subtle changes of aging, and deploy personalized health interventions via cognitive robots to support independent living.
Projects in collaboration with MHtech

- Monitor subtle changing in behavior of animals
- Measure forces exerted on head that may result in traumatic brain injury
- Measure and quantify changes in human behavior by using passive, non-invasive sensing
- System to monitor location of a dementia patient and notify when a person wanders away
- System to help support patients with eating disorders, alcoholism
- Measure child’s quality of sleep
- For more details see:
  - https://healthsciences.ucsd.edu/som/psychiatry/research/MHTechCenter/Pages/Collaborations.aspx
Acceleration of Large Scale Workloads: CPUs, GPUs, DSPs, PIM

Approximate Computing HW Supports

Enhanced CPU
- Associative MEM

Enhanced GPU
- ReCAM
- MASC

Enhanced DSP
- Enhanced DSP
- Enhanced DSP

Ap Storage

NDC* Accelerator

Relaxed MEM

Streaming App

Application Framework

System Libraries

Original Computing Units

Main Memory

New NVM-based Components
Accelerating Large Scale Workloads via Processing-In-Memory

Supported Operations
- Bitwise: OR, AND, XOR

Example of Operations
- OR, AND, XOR: Multiple row

Applications
- HD computing
- Graph processing
- Query processing

Search Operation
- Search: Nearest Search

Addition/Multiplication
- Matrix Multiplication

Classifications
- Clustering
- Database

Deep learning
- Security
- Multimedia
Introduction to Embedded Systems

CSE 237A

Tajana Simunic Rosing
Department of Computer Science and Engineering
UCSD
Embedded Systems Development Platforms
Why platforms?

- Embedded systems **operate** in, **interact** with, and **react** to an analog, real-time world.

- **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
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 (eg. 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
- “Flash” the controller to upgrade
- Software-driven GPIO + Conversion
Microcontrollers: Development Workflow

- Software development
  - Tools for formal models, models of computation...
  - ...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 microcontrollers with an established library system and development environment
  - Simplify addressing & control
  - Program in C or C++
  - setup() – initialization
  - loop() – main loop
  - bootloader

```c
#define LED_PIN 13

void setup () {
  pinMode (LED_PIN, OUTPUT); // Enable pin 13 for digital output
}

void loop () {
  digitalWrite (LED_PIN, HIGH); // Turn on the LED
delay (1000); // Wait one second (1000 milliseconds)
digitalWrite (LED_PIN, LOW); // Turn off the LED
delay (1000); // Wait one second
}
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 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):
  - 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 3

- ARM SoC (Cortex A53)
  - 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
    - 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
  - **FreeRTOS**: Real-time OS for microcontrollers

- 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
OS-based ARM Deployment: Android
OS-based ARM Deployment: Android vs. Linux

Primary Android Development

- 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)
Application Framework

▪ 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
Summary

- Class overview:
  - We’ll cover the highlights of embedded HW, control systems, signals & systems, embedded SW and embedded system modeling

- Platforms for embedded system development
  - HW
    - Small and low-power (e.g. Arduino)
    - Mid range e.g Raspberry PI
    - Larger size e.g. larger FPGA boards
  - SW
    - Development environments mostly based on C (e.g. Arduino)
    - OS and development environment (e.g. Linux for RPi3, Andriod)
  - Next: we start on overview of embedded HW components
UCSD Note #1: Basic Needs Resources

- Are you eating properly? Do you have adequate access to nutritious food? Do you have stable housing? Are you homeless or couch surfing?
- **If you or someone you know is suffering from food and/or housing insecurities, please note:**
  - The **Triton Food Pantry** (in the old Student Center), [https://www.facebook.com/tritonfoodpantry/](https://www.facebook.com/tritonfoodpantry/), is free and anonymous, and includes produce.
  - Financial aid resources, the possibility of emergency grant funding, and off-campus housing referral resources are available.
  - CAPS and college deans can connect students to the above resources, as well as other community resources and support.
UCSD Note #2:
Sexual Violence and Harassment

The Office for the Prevention of Harassment & Discrimination (OPHD) provides assistance to students, faculty, and staff regarding reports of bias, harassment, and discrimination. OPHD is the UC San Diego Title IX office. Title IX of the Education Amendments of 1972 is the federal law that prohibits sex discrimination in educational institutions that are recipients of federal funds. Jacobs School students have the right to an educational environment that is free from harassment and discrimination.

Students have options for reporting incidents of sexual violence and sexual harassment. Sexual violence includes sexual assault, dating violence, domestic violence, and stalking. Information about reporting options may be obtained at OPHD at (858) 534-8298, ophd@ucsd.edu or http://ophd.ucsd.edu. Students may receive confidential assistance at CARE at the Sexual Assault Resource Center at (858) 534-5793, sarc@ucsd.edu or http://care.ucsd.edu or Counseling and Psychological Services (CAPS) at (858) 534-3755 or http://caps.ucsd.edu.

Students may feel more comfortable discussing their particular concern with a trusted employee. This may be a Jacobs School student affairs staff member, a department Chair, a faculty member or other University official. These individuals have an obligation to report incidents of sexual violence and sexual harassment to OPHD. This does not necessarily mean that a formal complaint will be filed.

If you find yourself in an uncomfortable situation, ask for help. The Jacobs School is committed to upholding University policies regarding nondiscrimination, sexual violence and sexual harassment.