CS 211
Embedded Systems
Reconfigurable Architectures

Introduction to Embedded Systems
- What?
- Challenges in design
  - What do we need to do differently from what we have studied thus far
- Future Trends: Reconfigurable Architectures?
  - We have looked at fixed ISA and fixed datapath/logic processors...
  - Can we design architectures that can be reconfigured to meet the application requirements?
    - What new problems/challenges?
  - Field Programmable Gate Arrays (FPGA) as an example of reconfigurable architectures
    - Guest lecture: Brian Crilly, Thales Communications Inc.

Definition: Embedded System
- Embedded system: any device that includes a programmable computer but is not itself a general-purpose computer.
- Take advantage of application characteristics to optimize the design:
  - don’t need all the general-purpose bells and whistles.

Defining Embedded Systems
- An embedded system may take on several definitions. Similarities amongst these definitions, however, may include:
  - Constrained system resources
  - Minimized human-machine interface (if at all)
  - Singly-focused application that runs when power is applied and terminates when power source is turned off or depleted.
Modern Embedded Systems

- Personal digital assistant (PDA).
- Consumer electronics (e.g. digital cameras and household appliances)
- Cell phone
- Automobile engines, fuel control, etc.
- Global Positioning System (GPS) units
- Printers
- Home automation systems
- Manufacturing plants – process control

Embedding a computer

Recognize This System?

Cruise Missile Guidance
Examples: Minimized Human-Machine Interface

- ATM
- Vending machine

Singly-Focused Application

- This Global Positioning System (GPS) module receives a signal from a satellite and calculates its geographic position.
- It may be used as a component in a larger embedded system application.
- Notice...no on/off switches!!

A little history

- Automobiles used microprocessor-based engine controllers starting in 1970’s.
  - Control fuel/air mixture, engine timing, etc.
  - Multiple modes of operation: warm-up, cruise, hill climbing, etc.
  - Provides lower emissions, better fuel efficiency.
- First microprocessor was Intel 4004 in early 1970’s
- What types of processors account for over 95% of all processors sold today?

Automotive embedded systems

- Today’s high-end automobile may have 100 microprocessors:
  - 4-bit microcontroller checks seat belt;
  - Microcontrollers run dashboard devices;
  - 16/32-bit microprocessor controls engine.
BMW 850i brake and stability control system

- Anti-lock brake system (ABS): pumps brakes to reduce skidding.
- Automatic stability control (ASC+T): controls engine to improve stability.
- ABS and ASC+T communicate.
  - ABS was introduced first—needed to interface to existing ABS module.

Characteristics of embedded systems

- Sophisticated functionality
  - Often run complex algorithms; eg. cell phones
- Real-time operation.
  - Hard real-time constraints = system fails if time deadlines are not met…(missing an exam?)
- Low manufacturing cost—mass market items
  - Simple processors; small memory, microcontrollers, DSPs
- Low power.
  - Battery powered devices; battery size
- Designed to tight deadlines by small teams.

Challenges in embedded system design

- How much hardware do we need?
  - How big is the CPU? Memory?
- How do we meet our deadlines?
  - Faster hardware or cleverer software?
- How do we minimize power?
  - Turn off unnecessary logic? Reduce memory accesses?
Sample Embedded Systems requirements

<table>
<thead>
<tr>
<th>name</th>
<th>purpose</th>
<th>inputs</th>
<th>outputs</th>
<th>functions</th>
<th>performance</th>
<th>manufacturing cost</th>
<th>power</th>
<th>physical size/weight</th>
</tr>
</thead>
</table>

Example: GPS moving map requirements

- Moving map obtains position from GPS, paints map from local database.

GPS moving map needs

- **Functionality:** For automotive use. Show major roads and landmarks.
- **User interface:** At least 400 x 600 pixel screen. Three buttons max. Pop-up menu.
- **Performance:** Map should scroll smoothly. No more than 1 sec power-up. Lock onto GPS within 15 seconds.
- **Cost:** $500 street price = approx. $100 cost of goods sold.

GPS moving map needs, cont’d.

- **Physical size/weight:** Should fit in hand.
- **Power consumption:** Should run for 8 hours on four AA batteries.
What is Pervasive and Ubiquitous Computing

- "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."
- Creation of environments saturated with computing and communication capability, yet gracefully integrated with human users

Embedded Systems and Pervasive Computing?

- Embedded system
  - Driving technology
  - Individual components
- Pervasive Computing
  - Using embedded devices/systems to provide a pervasive computing environment
    - Networked, ubiquitous applications using embedded systems

Pervasive Computing: Examples

- Handheld
- Wearable computers
- Wireless LANs
- Devices to sense and control appliances
  - Smart homes
  - Security
  - Health care

SYSTEM COMPONENTS

- Network of Sensors: monitor and sense state of the house and occupants
- User Recognition: RFID, Biometrics
- Remote Management: an "operating system" for the SmartHome– the HomeOS
- Personalization: set user preferences
- Context following applications
**SAMPLE APPLICATIONS**

- Smart Phones – Running Java
- HVAC: Automatic Climate Control  
  - tailored to user needs and energy conservation
- Entertainment  
  - Recognize user and set music preferences automatically
- Home Security

**Software Development Process**

- There are many ways to develop applications for embedded systems
- Unlike general-purpose platform application development, you may not (generally) develop applications on the target embedded system platform
- Typically, a general-purpose host computer is used in conjunction with a cross compiler/assembler to create the binaries for the target system.
- The resultant code is downloaded to the target system and tested.

**Program design and analysis for Embedded Systems**

- Optimizing for execution time.
- Optimizing for energy/power.
- Optimizing for program size.

**Energy/power optimization**

- **Energy:** ability to do work.  
  - Most important in battery-powered systems.
- **Power:** energy per unit time.  
  - Important even in wall-plug systems—power becomes heat.
Power Equation

\[ P_{AVG} = \frac{1}{2} N_G f_{clk} C_L V^2_{DD} \]

- \( P_{AVG} \): the average power consumed by the gates
- \( N_G \): the number of gates that transition
- \( f_{clk} \): the frequency of the system clock
- \( C_L \): the average capacitive load per gate
- \( V_{DD} \): the supply voltage

Energy Equation

\[ \text{Energy} \end{cycle} = \frac{1}{2} N_G f_{clk} C_L V^2_{DD} \]

\[ \text{Energy} \end{operation} = \frac{1}{2} N_G f_{clk} C_L V^2_{DD} \]

- \( T_{MAX} \): the system throughput in operations / second

Energy Efficiency Metric

\[ ETR = \frac{\text{Energy} \end{operation}}{T_{MAX}} = \frac{P_{AVG}}{T_{MAX}} \]

- ETR is the Energy Throughput Ratio
- Balance energy per operation against throughput

Capacitive Load

- Load fixed in hardware
- External devices present higher loads
- Minimize accesses to external memory
  - Use code profiling to identify frequently executed traces and move to internal PROM
  - Maximize utilization of on-chip registers
  - Avoid cache misses
    - Penalty of filling an entire cache line
  - Beware of write policy
    - Write-through accesses external memory on every write
Number of Gates Switching

- Use optimized algorithms and compilers
  - Shorter dynamic run length results in fewer gates switching
- Use clock gating to disable functional units that are not needed for the current instruction
- Bit correlation
  - Number of gates switching is a function of the previous state and the current inputs
- Asynchronous design
  - Independently clock each functional unit

Clock Frequency

- Unless idle time dominates, clock frequency is detrimental to energy efficiency
- Clock frequency reduction is beneficial during idle periods
  - Many processors have a sleep mode available to reduce or disable the clock during idle periods
  - Beware of the transition penalty

Voltage

\[ \tau_{AVG} = \frac{C_i}{\beta V_{DD}} + \frac{C_i}{\beta' V_{DD}} \]

- Beware of the critical path delay
  - Frequency reduction may be necessary
- Attractive solution since energy is proportional to the square of the voltage

Considerations

- Which techniques are appropriate for incorporation into a compiler?
  - Code Optimization
  - Clock Gating
  - Voltage Scaling
  - Bit Correlation
  - Memory power consumption
- What other benefits can be derived from these techniques?
  - Heat and component reliability
  - EMI noise reduction
Cache behavior is important

- Energy consumption has a sweet spot as cache size changes:
  - cache too small: program thrashes, burning energy on external memory accesses;
  - cache too large: cache itself burns too much power.

Optimizing for energy

- First-order optimization:
  - high performance = low energy.
  - Not many instructions trade speed for energy.

Optimizing for energy, cont’d.

- Use registers efficiently.
- Identify and eliminate cache conflicts.
- Moderate loop unrolling eliminates some loop overhead instructions.
- Eliminate pipeline stalls.
- Inlining procedures may help: reduces linkage, but may increase cache thrashing.

Key Insights: Literature

- Does reduction in switching activity reduce power?
  - Compare code (a) minimal switching and (b) maximal switching
  - Measure Hamming distance (num. Of bit flips between two values) to estimate measure of switching
  - Was seen that there was no noticeable improvement in energy or power savings
Key Insights: Literature

- Do all type of ALU instructions consume same amount of power?
  - Run different type of instructions and collect power numbers
  - Observed that add/sub take much less than multiply
  - Try to replace multiply instructions with series of additions and shifts (strength reduction)
  - However, care must be taken since time penalty incurred can offset savings in energy!

Key Insights: Registers

- Does number of accesses to register file play a role in power consumption?
  - Compare code (a) accessing values from registers and (b) immediate operands
  - Measurements showed no significant difference

- Does value accessed affect power?
  - The more the switching of the value in the register file, the more power consumed
  - So try to minimize switching in register file accesses

Key Insights: Cache

- Do number of accesses to cache matter?
  - More power consumed when full access to data cache as opposed to no access
- Does value accessed from cache matter?
  - Implementation details of cache play role
  - Savings due to less switching are perceptible but not significant

Key Insights: Memory

- Do number of accesses to memory matter
  - Major impact on timing and therefore energy

- Does location in memory play a role?
  - Effect on number of memory modules that need to be “active”
  - Placing data into memory can be used to switch memory modules on or off and have big savings in power -- Levy, Crilly, Narahari, Simha 2000
Optimizing for program size

- Goal:
  - reduce hardware cost of memory;
  - reduce power consumption of memory units.
- Two opportunities:
  - data;
  - instructions.

Data size minimization

- Reuse constants, variables, data buffers in different parts of code.
  - Requires careful verification of correctness.
- Generate data using instructions.
- May need to limit amount of optimization
  - Example of loop unrolling

Reducing code size

- Avoid function inlining.
- Choose CPU with compact instructions.
- Use specialized instructions where possible.

Code compression

- Use statistical compression to reduce code size, decompress on-the-fly:

  ![Code compression diagram]
Issues in Compiling for Embedded Systems

- Code Generation for Specialized Architecture
  - Code size
    - Using techniques for performance optimizations, such as loop unrolling etc., can increase code size
- Timing requirements
  - To get precise timing may need to use assembly language
  - Not always "fast as you can" – may need minimum time, and duration
- Energy/Power optimization
  - Can we control power through software??
  - Instruction power – use of low power instructions
  - Dynamic voltage scaling
  - Memory power optimization
    - Placement of data
    - Dynamic power control of memory modules
    - For i=1 to N A[i]++4; / For j=1 to M B[j]++; / For k=1 to N C[k]++10;

Hardware Design Process

- In addition to software aspects of embedded systems, there is usually some amount of hardware design or interfacing that is required to support the intended application.
- These hardware features must be taken into account early in the system design process.
- In many cases, the hardware and software subsystems are developed concurrently. This is commonly referred to as hw/sw co-design.

Adding logic to a board

- Programmable logic devices (PLDs) provide low/medium density logic.
- Field-programmable gate arrays (FPGAs) provide more logic and multi-level logic.
- Application-specific integrated circuits (ASICs) are manufactured for a single purpose.

Designing Low-Level Logic

- Sometimes it is necessary to design circuitry to support a specific function for the system.
- We may, for instance, wish to design a sequence detector to cause our system to perform a particular function.
- We may use standard logic design practices to realize the desired function.
- The following steps outline the process for such a design. This is developed using standard logic gates.
HDLs facilitate design specification through language semantics.
Focus is on the high level system functionality.
Logical partitioning of design components may be represented in HDL semantics.
Designers think about how the design will work rather than its low level implementation.
An electronic circuit may be derived from the HDL model description through design synthesis.

A System on a Chip is a collection of components that are designed as a system and implemented in a fashion similar to that of an Application Specific Integrated Circuit (ASIC).
The goal is to pack as much functionality into a single die as possible.

Embedded computers are all around us. Many systems have complex embedded hardware and software.
Embedded systems pose many design challenges: design time, deadlines, power, etc.
Design methodologies help us manage the design process.
Relevance to Software designers?
Trend is towards software development environments.
May no longer need sophisticated hardware designers to build embedded systems.