Engineering New System-Chips

The changing chip industry and how it is driving the changing research agenda

Rajesh Gupta, UC San Diego

Microelectronic Embedded Systems Laboratory
mesl.ucsd.edu
Outline: System-on-Chips

- The economic opportunity
- The technological challenges
- The research agenda
A Chip Is A Wonderful Thing!

A typical chip, circa: 2006

- 50 square millimeters
- 50 million transistors
- 1-10 GHz, 100-1000 MOP/sq mm, 10-100 MIPS/mW
- 300 mm, 10,000 units/wafer, 20K wafers/month
- $5 per part

Does not matter what you build

- Processor, MEMS, Networking, Wireless, Memory

So there is a strong incentive to port your application, system, box to the "chip"
The Economic Engine

- Faster growing semiconductors claiming increasing share of the electronic systems market
  - From 5% to 19% over the two decade period from 1980 to 2000
  - Another 10% proliferation potential in next 6 years or about $100B in additional output in semiconductor chips

Supply push from technology
+ Moore’s law, decreasing bom

Demand pull from application
+ New sockets for chips
New Silicon Sockets

Semi Content in Electronic Systems ($)

- 20%
- 15%
- 10%
- 5%

Decade

Early 80's  Early 90's  Late 90's  Here

Networking, Wireless


$300,000

$10

5/31/04
R. Gupta,
UC San Diego
Near Term View

PC -> Networking -> Wireless -> Home AV

- Media proliferation from mobile handsets to digital home
  - Of $214B:
    - $31B from mobile phones (+15%)
    - and almost $60B from consumer electronics (+27%)

- From 2003 to 2008, about $181B worth of semis for handsets (majority is in the front end)
  - 73% of these chips will be sold outside of US

- By 2008, 1.2 billion users will spend $55B on data content of over $100B in wireless data services
  - Services moving from ringtones, photo messaging to broadcast media, network gaming, location services

Given the telecom experience, this probably understates the potential.

Source: Strategy Analytics, Goldman Sachs
Disruptive Trends

● (Hyper) integration remains compelling
  ■ Economics is just the beginning
  ■ Technology Enabler:
    - no wires, no battery, no stiffness

● Technology convergence = New systems capabilities
  ■ Computing and radios
  ■ On the horizon: sensing, micromechanical, microfluidics
    - Biofluidic sample preparation, transport, disposal
    - Chemical analysis, biological assays
    - In-situ monitoring, control, communication, adaptation
    - Remotely operated, reconfigurable laboratories for biochemical analysis
Chips in Human Body

Figure 1. A conceptual model of a totally implantable vestibular prosthesis. The implant is based on 3-axes micro-size gyroscopes integrated alongside with signal conditioning electronics on the same silicon chip.

Source: Shkel (MAE), Ikei (Biomed), Zheng (ENT), UC Irvine
Chips Into Fabrics and Buildings

Figure 1. Virginia Tech e-textile prototypes: Shape-sensing glove, an LED array display, and a single-cluster beamforming fabric

Figure 3. Geometry of wearable acoustic beamformer

Ember radios and networks

Source: Ember Networks
Increasing cost of SoC solutions

Not a problem, until you consider this…
“Disaggregation” in Semis

- Of the 72 distinct application markets that rely on value added IC designs (ASIC, ASSP, FPGA, SOC)
  - over 50% are less than $500M
  - 75% are less than $1B
- Outsourced manufacturing is changing the supply dynamics

Source: IBS
Structural changes in industry...

- The rising fabless, fablite
  - As domestic manufacturing declines (to 17%), increasing share of the (systems) design market

- The US
  - has 56% of over 1K design houses...
  - ...and accounts for 76% of industry revenues
  - $24B going onto 50% of semis by 2006
  - Segments: Wireless 27%, networking 25%, consumer 20%
Growing gap between technology capability and use
- Driven by cost, tool considerations
- Only 17% of designs above 500 MHz
  - 67% of ASIC designs are 299 MHz and lower
- Sizes pretty much evenly distributed from 100K to 5M gates

Joe Chip today:
- 200 MHz, CPU+RAM+NVM, 2M gates

On-chip memory is a growth area
- About ½ the area, rising to 70% on 70nm nodes @1-2 Mb/mm²
- In the meantime, DDR DRAM controllers are pretty common

FPGAs and CPLDs making inroads into boxes
- 61% of system cost is in FPGAs
- Some have gone so far as predicting death of the standalone MPU
- Or is it, Makimoto wave in progress?
Swing to standardization

- Expensive first silicon
  - (about 6X at 90nm than at 0.35nm)

- Product life span much shorter
  - 1-2 years compared to 3-5 years earlier

- Result: shrinking ASIC starts, increasing FPGA
The New Semi Characteristics

- Highly application specific (at least, first silicon)
  - domain specific IC design, focus on system level

- Content increasingly determines processing
  - “embedded intelligence” through embedded software

- Connection more important than processing
  - bandwidth delivery more important than computational efficiency

- Architect these to be usable through software
The tall-thin engineer has very little middle-ground left

- Engineering moving up and moving down
  - Silicon Engineering vs. Systems Engineering

Problems of the small

- Leaky transistors, porous oxide, multiple $V_t$, ...
- The electrical engineering in IC design
- A lot of engineering at the interfaces

Problems of the large, diversity, efficiency

- The computer science in IC design
- Millions of transistors, thousands of complex, diverse blocks working together
- Technology, methods must scale at every level
- Widespread heterogeneity
Engineering at the “Interfaces”

- Layout fracturing, device engineering
- Electronic, biochemical signaling
- Architecture, compiler, algorithm
- Operating system, middleware, applications,
- … and so on.
Sub-Wavelength Optical Lithography

- Slide courtesy of Numerical Technologies, Inc.
Circuits and Layout
-- Andrew Kahng, UCSD

Leakage Variability
Gate-length
Leakage
30%
20X

OPC Corrections

Original Layout
Cross Layer Low Power Design

Rajesh Gupta, UCSD
Outline: System-on-Chips

- The economics
- The technology
- The research: systems engineering for SoCs
  - 1, 2, 3
- The research enterprise
Consider platform for wireless

- Commercial “Platforms” for wireless SOC
  - TI OMAP (Open Multimedia Applications Platform)
  - Intel PCA (Personal Internet Client Architecture)
  - Motorola MXC (Mobile Extreme Convergence)

- Different timelines, but all three directed at wireless SOC based systems (cellular, 802.11)

  Tasks = Communication : Networking : Applications

  Generally seek energy efficient processing through division of labor (among multiple processing elements)

  Additional datapath; memory mapped functions; coprocessing

  Common Theme

  Integration in hardware, but specialization in software and sw infrastructure

  Exploitation of full potential of SOC architectural space is a challenge.
Example Problem: How to achieve high throughput in a SOC for wireless applications?

- Can select a modem sub-system that packs more bits/Hz, but it will tolerate less noise and be less robust so that link throughput may not improve.

- Can increase transmit power in RF subsystem to improve robustness but this increases energy cost, reduces network capacity, and requires more expensive analog circuits (power amps).

- Can reduce bits/frame to tolerate higher bit error rates (BER) and provide more robustness, but this may increase overhead and queuing delays.

- Can increase precision in digital modem to reduce noise, but this leads to wider on-chip busses and more power consumption.

The design technology must support right sub-system option and parametric determination.

To do that we need to be able to carry out model integrated simulations and analysis.
Our Research Focus

1. Components and Compositional Correctness
   - \textit{A posteriori} validation is simply not possible

2. Low power systems design
   - Engineer systems (components, interfaces, networks) for low power

3. Software and Software Infrastructure
   - Hardware capabilities and constraints driving need for new software architecture
   - New “awareness” into software infrastructure
     - Energy, Location, Security, Reactivity, Precision
1 Compositional Correctness

- Build “Complete” System Models
  - That include the application and system software
  - Adapt, control and debug applications
  - Explore the full potential of SOC architectural platforms
    - e.g., by exploring applications, networking and communication subsystems together
- High(er) level languages are increasingly explored for system modeling
- These can enable higher abstraction levels and system complexity reduction (e.g., through object orientation)
- But this requires ability to reuse system components
  - Need for ‘adequate’, ‘hierarchical’ composability
  - Need for verifiability of components and composition
- One way to do this would be to build a
  ➤ Component Composition Framework (CCF)
    - Incorporate capabilities from programming language to runtime system to enable composability and verification.
Our Approach to Building CCF

- Define compositional semantics across MOCs
  - Focus on interface ‘behavior’ and its algebraic manipulation through a ‘multi-clock calculus’
  - enable easy system construction and its “formal” validation
  - “adequate”, hierarchical and verifiable composition
  - Create “Virtual” System Architectures

- Can be done through
  - Polymorphic interfaces and mixed compiled and interpreted programming components
  - Incorporating capabilities in the design technology for reflection and introspection

- Ensure Correctness through advances in type theory
  - Capture “behavioral types” and model checking obligations

- Primary obstacle to composability
  - Semantic gap between silicon IP and their software models
A composition environment
- Built upon existing class libraries, to add a software layer for manipulation and configuration of C++ IP models
- Ease software module connectivity
- Run-time environment structure

A SW architecture that enables
- composition of structural and functional information

Current state
- SystemC + NS2 + ISS + OS services

System designer
Component Integration, CIL
Split-Level Interface/BIDL
C++, SystemC
Design Example

Adaptive Memory Reconfiguration & Management

Communication and concurrency refinement

Use a minimum subset of CIL for composing an architecture or a platform.
2 Low power has been a design focus

- Speed power efficiency has indeed gone up
  - 10x / 2.5 years for μPs and DSPs in 1990s
    - between 100 mW/MIP to 0.3 mW/MIP since 1990
  - IC processes have provided 10x / 8 years since 1965
  - rest from power conscious IC design in recent years.

- Another 10x is possible.

- But there is limit to energy efficiency:

Source: ISI/USC, DARPA PACC Program
Basically two ways to save power

- **DPM (Dynamic Power Management)**
  - “shutdown” through choice of right system & device states
    - Multiple sleep states
- **DSS (Dynamic Voltage/Frequency Scaling)**
  - “slowdown” through choice of right system & device states
    - Multiple active states
- **DPM + DSS**
  - Choice between amount of slowdown and shutdown
  - However, the problem changes “qualitatively”.
Slowdown Preferred Over Shutdown

- Example: 50ms task with 100ms deadline
  - 50ms computation, 50ms idle/stopped time
  - Half speed: 100ms computation, 0ms idle
    - ¼ energy than the full speed case if voltage scales

- Use voltage to control the operating point on the power vs. speed curve
  - i.e., power and clock frequency are functions of voltage

- Because of higher savings in slowdown often shutdown becomes a “secondary” strategy
  - First slow down and then look for ways to shutdown.
Drop in energy efficiency

Figure 2. Cycle time and Energy variation for 0.07μm technology
“System Design” for Low Power

- Energy efficiency (has to) cut across all system layers
  - circuit, logic, software, protocols, algorithms, user interface, power supply...
  - Computation versus Communication; Node versus network

- Trade-off between energy consumption & QoS
  - optimize energy metric while meeting “quality” constraint
3 Software and Its Infrastructure

- Changes in structure of system software
  - OS moving towards micro-kernels
    - Services moved to processes (e.g., Nucleus, Symbian)
    - Still legacy remains: memory, file semantics as unifying theme for communications.

- Changes in division of labor among
  - Application, middleware, operating system
  - Compiler, runtime

- Challenges in bringing new capabilities and contract into the system software
Consider Energy “Awareness”

- What does it mean to be aware?
  - Services “know” about energy, power
    - File system, memory management, process scheduling
    - Make each of them energy aware

- How does one make software to be “aware”?
  - Use “reflectivity” in software to build adaptive software
  - Ability to reason about and act upon itself (OS, MW)
Energy Awareness

A. Application

B. OS

C. Middleware
Reflection, Reification, Introspection

● "Reflection Architecture Pattern" in software
  ■ provides mechanism for changing structure and behavior of software dynamically
  ■ Supports modification of fundamental aspects such as type structures, function call mechanisms etc
  ■ Software architecture: meta versus base level
  ■ Information exchange between levels can be done through cooperation of runtime, user defined function calls, or software layering (IDLs).

● Our approach
  ■ Capture application intent through reflective mechanisms
  ■ Adaptivity in middleware, new services that interact with OS level services, application APIs to build user-level dialogues.
  ■ Architect these to minimize overheads.
**Approach**

- Characterize application offline
  - Divide an application into *phases of execution*
    - A group of program intervals executing similar code
  - Each phase has similar demand on resources
    - Similar code, similar resource demands (memory, IPC)

- Annotate source code
  - Phase signatures

- Enable OS (and hardware) to recognize signature
  - Smart hardware and/or online learning techniques

- Dynamically tune the power manager
  - As application moves from one phase to another.
Our approach - dynamic analysis

- Use performance counters:
  - Can be programmed to generate an interrupt on specified counts
- ISR provides matching with the meta data and mode changes
  - Every S*10,000 loop branches try a match
  - Phase matching can also be done in hardware
- Notify power manager to trigger proper action
Example: Memory Bank Shutdowns

Average among bzip, mpeg, ghostscript and ADPCM
Approx. 350K instructions for every 10,000 loop branch instructions

Number of instructions executed by the match algorithm at every 10,000 loop branches to match a partial signature (500 instructions per phase)

<table>
<thead>
<tr>
<th># of phases</th>
<th># instructions</th>
<th>overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2,580</td>
<td>0.7%</td>
</tr>
<tr>
<td>10</td>
<td>4,500</td>
<td>1%</td>
</tr>
<tr>
<td>20</td>
<td>8,280</td>
<td>2%</td>
</tr>
<tr>
<td>30</td>
<td>12,060</td>
<td>3%</td>
</tr>
</tbody>
</table>

Size overhead. 4 bytes per inter arrival estimate per bank / phase. 4 x 16 x 10 = 640 bytes assuming 16 banks and 10 phases.
The signatures take 1280 bytes for 10 phases. Total of 2KB of meta data
Energy Awareness

A Application

B OS

C Middleware
Power Aware Parts in the OS

- **PA-API (Power Aware API)**
  - interfaces applications and OS making the power aware OS services available to the application writer.

- **PA-OSL (Power Aware Operating System Layer)**
  - implements modified OS services and active components such as a DPM manager.

- **PA-HAL (Power Aware Hardware Abstraction Layer)**
  - interfaces OS and Hardware making the power control knobs available to the OS programmer.
OS Services

- **PA-API** - Power aware function calls available to the application writer.
  - Some functions of this layer are specific to certain scheduling techniques.

- **PA-Middleware** - Power aware services
  - Implemented on the top of the OS (power management threads, data handling, etc...).

- **POSIX** - Standard interface for OS system calls.
  - This isolates PA-API and PA-Middleware from OS.

- **PA-OSL** - Power aware OS layer.
  - Calls related to modified OS services should go through this level. Also isolates OS from PA-API and PA-Middleware.

- **PA-HAL** - Power Aware Hardware Abstraction Layer.
  - Isolates OS from underlying power aware hardware.

- **Modified OS services**
  - Implementation / modification of OS services in a power related fashion. Ex: scheduler, memory manager, I/O, etc.
Prototype Implementation

- **Platforms**
  - **eCOS RTOS:**
    - open source, Object oriented and highly configurable RTOS (by means of scripting language)
  - **Hardware platforms we are currently working with:**
    - Linux-synthetic (emulation of eCos over Linux - debugging purposes only)
    - Compaq iPaq Pocket PC, Accelent IDP
    - LRH Intel evaluation board 80200EVB
OS-directed DVS Results

Energy Consumption for each scheme

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Taskset A</th>
<th>Taskset B</th>
<th>Taskset C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shut/Static/Dyn./Adapt. (0.75)</td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
</tr>
<tr>
<td>Shut/Static/Dyn./Adapt. (0.80)</td>
<td>0.85</td>
<td>0.85</td>
<td>0.85</td>
</tr>
<tr>
<td>Shut/Static/Dyn./Adapt. (0.85)</td>
<td>0.90</td>
<td>0.90</td>
<td>0.90</td>
</tr>
<tr>
<td>Shut/Static/Dyn./Adapt.</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>Shut/Static/Dyn.</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Only Shutdown</td>
<td>1.20</td>
<td>1.20</td>
<td>1.20</td>
</tr>
<tr>
<td>Normal</td>
<td>1.20</td>
<td>1.20</td>
<td>1.20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task</th>
<th>Application</th>
<th>WCET (us)</th>
<th>Std Dev (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>MPEG2 (wg_gdo_1.mpg)</td>
<td>30700</td>
<td>3100</td>
</tr>
<tr>
<td>T2</td>
<td>MPEG2 (wg_cs_1.mpg)</td>
<td>26300</td>
<td>2100</td>
</tr>
<tr>
<td>T3</td>
<td>ADPCM</td>
<td>9300</td>
<td>3300</td>
</tr>
<tr>
<td>T4</td>
<td>FFT</td>
<td>15900</td>
<td>0</td>
</tr>
<tr>
<td>T5</td>
<td>FFT (gaussian distribution)</td>
<td>13600</td>
<td>800</td>
</tr>
</tbody>
</table>
Energy Aware Middleware in FORGE

[Jointly with Nalini Venkatsubramaniam, UC Irvine.]

- Use reflective middleware services to continuously monitor and keep track of application needs
  - Use a rule base and director service to carry out its functions.

- Example:
  - multimedia streaming from a server to a set of mobile “nodes”
  - Use a proxy server to adapt video stream to specific nodes

- Node device: sends device info to proxy, connects video stream and network parameters to lower layers
- Proxy: admission control, real-time transcoding, network traffic regulation.
Example

- **T₀**: 3 users (1, 2, 3)
- **T₁**: user 4 joins
  - System readjusts quality levels
- **T₂**: residual power on user 1 decreases
  - Quality level is decreased for 1
- **T₃**: user 5 joins
  - All levels go down to accommodate 5
- **T₄**: user 3 finishes streaming
  - Quality levels rise back
- **T₅**: user 1 finishes streaming
  - Even higher levels
Coordinated Nodal DPM

- **CPU**
  - Dynamic Voltage & Freq. Scaling

- **Sensors**
  - Scalable Sensor Processing

- **Radio**
  - Freq., Power, Modulation, & Code Scaling

**Coordinated Power Management**

**PA-APIs for Communication, Computation, & Sensing**

**Energy-aware RTOS, Protocols, & Middleware**

**Hardware**

Jointly with Mani Srivastava, UCLA
Summary

- The new Silicon comes out of the fab fast and furious
  - Our ability to implement and manufacture vastly exceeds our capability to architect, reason and validate the new generation of silicon systems
- Our challenge is to make sure what goes into manufacturing has tremendous value-add to end application (systems)
  - Software is the defining IP
  - But it is a whole new ballgame: new awareness to support new services (conventional middleware does not apply)
- And that means keeping the focus on systems and their applications
  - Modeling, conceptualization, composition, validation
  - System level strategies for scalability, operational efficiency.
Related Projects:

- **FORGE Project**
  - *Adaptive Middleware Services for Distributed Embedded Systems*
  - Collaborators: Nik Dutt, Alex Nicolau, Nalini Venkatsubramaniam, UCI; Sandeep Shukla, Virginia Tech
  - Students: Cristiano Pereira, Radu Cornea, Shivjit Mohapatra
  - Funded by: NSF

- **PADS Project**
  - *Power Aware Distributed Systems*
  - Collaborator: Mani Srivastava, UCLA
  - Students: Cristiano Pereira, Yuvraj Agrawal, Vijay Raghunathan
  - Funded by: SRC, DARPA

- **OSDPM Project**
  - *Formal Methods in Dynamic Power Management*
  - Collaborators: Sandeep Shukla, Virginia Tech; Sandy Irani, UCI
  - Students: Ravindra Jejurikar, Hiren Patel
  - Funded by: NSF