Energy-Efficient Embedded System Design at 90nm and Below
~ A System-Level Perspective ~

Tohru Ishihara
System LSI Research Center
Kyushu University
Agenda

- Introduction
- Software-Level Energy Characterization
- Process-Variation Aware Compilation
What is Embedded System?

- A combination of computer hardware and software
- A specialized computer system which is dedicated to a specific task
- Performance is not necessarily very important
- Can spent much time for compiler optimization
- The number of chips produced is 1/100 of general purpose processors
Mask Costs at 90nm & Below

- Mask cost doubles every 2 years

<table>
<thead>
<tr>
<th>Year of Production</th>
<th>'05</th>
<th>'06</th>
<th>'07</th>
<th>'08</th>
<th>'09</th>
<th>'10</th>
<th>'11</th>
<th>'12</th>
<th>'13</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPU/ASIC Metal1 (\frac{1}{2}) Pitch (nm)</td>
<td>90</td>
<td>78</td>
<td>68</td>
<td>59</td>
<td>52</td>
<td>45</td>
<td>40</td>
<td>36</td>
<td>32</td>
</tr>
<tr>
<td>Mask Cost ($m)</td>
<td>1.5</td>
<td>2.2</td>
<td>3.0</td>
<td>4.5</td>
<td>6.0</td>
<td>9.0</td>
<td>12.0</td>
<td>18.0</td>
<td>24.0</td>
</tr>
</tbody>
</table>

- Suppose the number of chips per a mask set is 100,000

The mask cost per chip is more than $30

Fujitsu’s embedded processor

Chip price $27
Solutions

- Field Programmable Devices
  - Energy consumption is still high
  - 10x or 100x of ASIC chips

- Software and processors
  - The energy consumption is highly depending on the software running on the target processor
  - Software programmers do not pay attention to energy reduction
Agenda

- Introduction
- Software-Level Energy Characterization
- Process-Variation Aware Compilation
Motivation

- Energy consumption of embedded systems depend on the behavior of the software running on the hardware.
- Most software programmers pay less attention to the energy issue than hardware designers do.
- **Software-level energy analysis is needed for reducing energy consumption of embedded systems.**
Our Goal

- Energy analysis framework which can be used in a software design phase

Work in cooperation with GDB

Bottleneck analysis from a software viewpoint
Existing Tools

- **SimplePower [Irwin@PSU]**
  - Based on RTL simulation
  - Energy is calculated from activities of RTL blocks and predetermined energy values of the blocks

- **Wattch [Brooks@Princeton]**
  - Based on modified SimpleScalar
  - SimpleScalar is not popular among embedded system designers
Our Tool

- Technology Independent Framework

Target CPU → Cell Library → CPU Characterization (Our tool) → Energy Consumption Model

Target Application → C Compiler (GCC) → ISS (GDB) → Interface

GDB GUI → GUI
Gate-level Power Estimation (e.g. NC-Verilog, PowerCompiler)

Overview

Netlist of CPU | Cell Library

Characterization Benches

ISS (GNU tool)

Golden Model

Gate-level Power Estimation

Find coefficients using regression analysis

Approximation Model

Golden Model

\[ E_{GL}(m) \leftarrow E_{estimated}(m) = k_1P_1(m) + k_2P_2(m) + \ldots + k_nP_n(m) \]

\[ \Sigma |E_{estimated}(i) - E_{GL}(i)| \]

\[ E_{estimated}(1) = k_1P_1(1) + k_2P_2(1) + \ldots + k_nP_n(1) \]

\[ E_{estimated}(2) = k_1P_1(2) + k_2P_2(2) + \ldots + k_nP_n(2) \]

\[ E_{estimated}(m) = k_1P_1(m) + k_2P_2(m) + \ldots + k_nP_n(m) \]

\[ P_1: \# \text{ instruction/data cache misses} \]

\[ P_2: \# \text{ taken branches executed} \]

\[ P_3: \# \text{ load/store instructions executed} \]

\[ \ldots \]

\[ \ldots \]
Experimental Setup

- M32R-II or SH3-DSP and SDRAM (Micron)
  - 5-stage pipeline
  - 8KB 2-way I-Cache and D-Cache
  - 32KB SRAM
- GNU CC (e.g., m32r-linux-gcc)
- NC-Verilog from Cadence and PowerCompiler from SYNOPSYS for the Gate-Level Energy Estimation
- 0.18um Standard Cell Library
- System Power Calculator for the energy model of SDRAM
- GNU based ISS (e.g, m32r-linux-run)
- CPLEX from ILOG for solving Linear Programming
Detailed Characterization Flow

Netlist of CPU → Gate-level Simulator

Cell Library → Instruction-Set Simulator

Test Benches

# instructions simulated is 500,000

# parameters is 20

Gate-level Simulator

Instruction-Set Simulator

SAIF₁
Test Bench 1

SAIFₙ
Test Bench n

Trace₁
Test Bench 1

Traceₙ
Test Bench n

3.5 hours

Energy Calculation

Counting Parameter Values

Linear Programming for finding optimal coefficients

\[ E_1 \leftarrow E'_1 = c_1 P_{11} + c_2 P_{12} + c_3 P_{13} \ldots \]

\[ \vdots \]

\[ E_n \leftarrow E'_n = c_1 P_{n1} + c_2 P_{n2} + c_3 P_{n3} \ldots \]

Linear Equation

Test Bench 1

Test Bench n

300,000 ins./sec
## Experimental Results (M32R-II)

<table>
<thead>
<tr>
<th>Benchmark Program</th>
<th>Error (%)</th>
<th>Standard Deviation of error Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ave.</td>
<td>Max.</td>
</tr>
<tr>
<td>JPEG</td>
<td>2.70</td>
<td>10.32</td>
</tr>
<tr>
<td>JPEG_opt</td>
<td>6.09</td>
<td>16.46</td>
</tr>
<tr>
<td>MPEG2</td>
<td>1.54</td>
<td>3.97</td>
</tr>
<tr>
<td>MPEG2_opt</td>
<td>1.78</td>
<td>5.15</td>
</tr>
<tr>
<td>compress</td>
<td>5.00</td>
<td>6.41</td>
</tr>
<tr>
<td>compress_opt</td>
<td>4.35</td>
<td>7.18</td>
</tr>
<tr>
<td>FFT</td>
<td>1.55</td>
<td>6.87</td>
</tr>
<tr>
<td>FFT_opt</td>
<td>1.45</td>
<td>5.59</td>
</tr>
<tr>
<td>DCT</td>
<td>1.42</td>
<td>8.58</td>
</tr>
<tr>
<td>DCT_opt</td>
<td>1.47</td>
<td>8.07</td>
</tr>
<tr>
<td>Total</td>
<td>2.74</td>
<td>16.46</td>
</tr>
</tbody>
</table>

* _opt corresponds programs compiled with -O3 option*
## Experimental Results (SH3-DSP)

<table>
<thead>
<tr>
<th>Benchmark Program</th>
<th>Error (%)</th>
<th>Standard Deviation of Error Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ave.</td>
<td>Max.</td>
</tr>
<tr>
<td>JPEG</td>
<td>3.61</td>
<td>11.23</td>
</tr>
<tr>
<td>JPEG_opt</td>
<td>5.09</td>
<td>15.70</td>
</tr>
<tr>
<td>MPEG2</td>
<td>3.43</td>
<td>5.93</td>
</tr>
<tr>
<td>MPEG2_opt</td>
<td>3.33</td>
<td>5.59</td>
</tr>
<tr>
<td>compress</td>
<td>8.50</td>
<td>10.67</td>
</tr>
<tr>
<td>compress_opt</td>
<td>1.48</td>
<td>15.23</td>
</tr>
<tr>
<td>FFT</td>
<td>2.89</td>
<td>5.87</td>
</tr>
<tr>
<td>FFT_opt</td>
<td>3.25</td>
<td>6.30</td>
</tr>
<tr>
<td>DCT</td>
<td>0.72</td>
<td>1.91</td>
</tr>
<tr>
<td>DCT_opt</td>
<td>0.99</td>
<td>2.38</td>
</tr>
<tr>
<td>Total</td>
<td>3.33</td>
<td>15.70</td>
</tr>
</tbody>
</table>

* _opt corresponds programs compiled with -O3 option*
JPEG Encoder (M32R-II)

**Gate Level**

Sample Number (5000 instructions/Sample)

- JPEG Encoder w/o opt Ave. 2.70%, Max. 10.32%
- JPEG Encoder w/ opt Ave. 6.09%, Max. 16.46%
JPEG Encoder (SH3-DSP)

**JPEG Encoder w/o opt**
- Ave. 3.61%, Max. 11.23%

**JPEG Encoder w/ opt**
- Ave. 5.09%, Max. 15.70%
Summary

- Proposed an energy characterization framework
- The error of our approach is 3% on an average and 16% at the maximum case.

Future work

- Extending the current approach for targeting multi-core processors
Agenda

- Introduction
- Software-Level Energy Characterization
- Process-Variation Aware Compilation
Leakage exponentially increases as the transistor size shrinks.


Leakage exponentially increases along with the chip temperature.

Process Variation

S. Borkar, Parameter variations and impact on circuits and microarchitecture, DAC 2003.
Motivation

Large Intra-Die Variation

Current 3-sigma = 13%
Vth 3-sigma = 67mV

Variation is huge in small transistors

\[ \sigma_{Vth} = \frac{q}{C_{ox}} \sqrt{\frac{N_a \cdot W_{dm}}{3 \cdot L \cdot W}} \]

\(L, W\): Effective channel length and width
\(q\): electron charge
\(C_{ox}\): oxide capacitance
\(N_a\): substrate doping concentration
\(W_{dm}\): maximum depletion width

Process Variation at 90nm

\[ I_{\text{Subthreshold}} \propto \frac{W \cdot V_T^2}{T_{ox} \cdot L} \cdot \exp\left(\frac{-V_{th}}{\alpha \cdot V_T}\right) \]

- \( V_T \): Thermal voltage (25mV@room temperature)
- \( \alpha \): Sub-threshold factor (1.40~1.65)
- \( T_{ox} \): Oxide thickness

More than 2 transistors out of 64K-bit SRAM

\[ 5\sigma_{V_{th}} = 0.3V \]

Leakage is 1,500x higher than average!

<table>
<thead>
<tr>
<th>Year</th>
<th>min. L [nm]</th>
<th>( ^1V_{TH} ) [V]</th>
<th>( ^2V_{TH} ) [V]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2004</td>
<td>37 (90)</td>
<td>0.32</td>
<td>0.12</td>
</tr>
<tr>
<td>2005</td>
<td>32 (80)</td>
<td>0.33</td>
<td>0.09</td>
</tr>
<tr>
<td>2006</td>
<td>28 (70)</td>
<td>0.34</td>
<td>0.06</td>
</tr>
</tbody>
</table>

1: Low Operating Power Process   2: MPU process

Delay is 2x of the average

\[ 5\sigma_v = 0.3V \]

\[ \text{Large Leak} \]

\[ \text{Threshold Voltage} \]

\[ \pm \sigma: 68.3\% \]
\[ \pm 2\sigma: 95.4\% \]
\[ \pm 3\sigma: 99.7\% \]
\[ \pm 4\sigma: 99.9936\% \]
\[ \pm 5\sigma: 99.99994\% \]
Marking Bad Cache-Lines

- Use an unused combination of existing flag bits to indicate a slow SRAM cell in a specific cache-line.
- Invalidate and skip a cache-line if it is marked.

4-way set-associative cache memory

Cache replacement policy
Cache Miss Reduction

- Using a smaller cache memory does not affect the correct operation of a processor.
- The idea is to mark extremely slow cache lines and to use fast cache lines only.
- The problem is an increase of cache miss rate due to a reduced cache size.

![Diagram of processor and main memory with cache]

Mark sections which contain a leaky or slow bit.
Use fault-free sections only.
Our Approach

- Modify the order of functions in the address space such that ultra-slow cache-lines are not accessed frequently.
Process Variation at 90nm

\[ I_{\text{Subthreshold}} \propto \frac{W \cdot V_T^2}{T_{ox} \cdot L} \cdot \exp\left(\frac{-V_{th}}{\alpha \cdot V_T}\right) \]

\( V_T \): Thermal voltage (25mV@room temperature)
\( \alpha \): Sub-threshold factor (1.40~1.65)
\( T_{ox} \): Oxide thickness

More than 2 transistors out of 64K-bit SRAM

5\( \sigma_{V_{th}} \)=0.3V

Leakage is 1,500x higher than average!

\[ \begin{array}{|c|c|c|c|}
\hline
\text{Year} & \text{min. } L [\text{nm}] & V_{TH}^{1} [\text{V}] & V_{TH}^{2} [\text{V}] \\
\hline
2004 & 37 (90) & 0.32 & 0.12 \\
2005 & 32 (80) & 0.33 & 0.09 \\
2006 & 28 (70) & 0.34 & 0.06 \\
\hline
\end{array} \]

1: Low Operating Power Process  2: MPU process

\( \frac{\text{V}_{th}=0.3V}{\text{Delay is 2x of the average}} \)

\( \frac{\text{Large Leak}}{\text{Large Delay}} \)

\( \pm \sigma: 68.3\% \\
\pm 2\sigma: 95.4\% \\
\pm 3\sigma: 99.7\% \\
\pm 4\sigma: 99.9936\% \\
\pm 5\sigma: 99.99994\% \)
Masking Leaky Cells

Leakage current of a SRAM cell depends on the logic value stored

If M2, M3, or M5 is leaky, the SRAM cell is 1-leaky
If M1, M4, or M6 is leaky, the SRAM cell is 0-leaky

Charged to Vdd at inactive mode

BL

WL

M1

M2

M3

M4

M5

M6

Q

Q

BL

1

1

10 1110
Masking Leaky Cache-Lines

- Modify the order of instruction codes considering binary expressions of the codes and locations of 0/1-leaky bits in a cache so that the total leakage current is minimized.
The Flow

Fabricated Chip

Target Application
Detect Locations of Bad Cells
Generate a new object code or reuse an object code previously generated
Mark Slow Cache-Lines
Execute Object Code

Testing Phase (BIST)
Compiling Phase
Booting Phase
Running Phase (Flash Microcontroller)
Compiler Optimization Flow

- Target Application
- Original Object Code
- Sample Data
- Address Trace (Profiling Information)
- Find instruction scheduling and code placement, for which the total energy consumption can be minimized
- Locations of Bad Cells
- Modified Object Code
New Paradigm

- Use different object codes for different chips

- Future work: Reducing the test cost
Previous Work (1/2)

- Vergos et al. proposed a technique using spare cache.
- Sohi proposed a technique using error correcting code.
Shiravani et al. proposed \textit{PADded} cache

- Customize an address decoder so that faulty blocks will not be accessed.

\[ f_0 a_0 + f_1 \]

\[ f_1 a_0 + f_0 \]

\[ f_2 a_0 + f_3 \]

\[ f_3 a_0 + f_2 \]

\[ \text{Cache-Line0 (a}_0\text{a}_1\text{)=00} \]

\[ \text{TAG} \quad \text{DATA} \]

\[ \text{Cache-Line1 (a}_0\text{a}_1\text{)=01} \]

\[ \text{TAG} \quad \text{DATA} \]

\[ \text{Cache-Line2 (a}_0\text{a}_1\text{)=10} \]

\[ \text{TAG} \quad \text{DATA} \]

\[ \text{Cache-Line3 (a}_0\text{a}_1\text{)=11} \]

\[ \text{TAG} \quad \text{DATA} \]
Experimental Setup

- Applied our technique to ARMv4T architecture
- Used three programs, *Compress*, *JPEG*, and *MPEG2*
- Considered three scenarios: best, typical, and worst
Experimental Results

Cache-1 (1-way, 32Kb)

Cache-2 (2-way, 32Kb)

Cache-3 (4-way, 32Kb)

Cache-4 (2-way, 16Kb)
Summary

- Cancel the degradation of cache hit-rate even in presence of 25% slow cache-lines.
- Worst case (5-sigma) delay can be reduced by 40%.
- No major HW modification is required.

Future work

- Implement an instruction scheduling algorithm for reducing leakage current
Conclusion

- Flexibility and customizability become much more important in future technologies.
- Process-variation-aware design at system-level is essential for saving energy.
- Hardware and software cooperation is very important.


Input Data Dependency

• Compared cache miss rates for 6 different input values.
• The optimized code for Data0 achieves very good results for other input values too.