Compilers for DSP Processors and Low-Power

Jenq-Kuen Lee
Ting-Ting Hwang
Department of Computer Science
National Tsing-Hua Univ.
Hsinchu, Taiwan
Agenda

- DSP Compilers
- Compilers for Low-Power
NSC 3C DSP Compiler Infrastructures

- **Target Machine**
  - DSP Processor
  - Low power instruction support.

- **Compiler Infrastructure**
  - Cross-Compiler
    - GNU Compiler Collection v1.37.1.
    - Support low power instructions.
  - Cross-Assembler
    - Re-write new assembler.
    - Support low power instructions.
3C DSP Compiler Infrastructure (cont’d)

Source Program

GNU Compiler Collection

Front-end

Machine description

Optimization

Low power instructions / support

Instruction scheduling for low power

Assembly Code

Front-End

Assembler

Optimization

Executable Code
ORISAL Architecture Description

Language

- ORISAL is based on Java-like syntax and styles.
- Object oriented styles will reduce specification writing efforts from scratch and also give the designers a more natural view of coding.
- Object oriented styles will reduce mistakes compared to other imperative language based ADL.
- ORISAL will incorporate power model descriptions to deliver more adaptable power simulations and optimizations for low-power.
ORISAL and Simulator Generator

- **Benefits in ORISAL:**
  - Java natively has good thread and exception handling support, could behave better than other language (C/C++) in *synchronization* mechanism.
  - Simulator could be easily extended with distributed network environments and accelerate *large-scale System-On-a-Chip simulation*. (RMI and JavaBeans)

- **Status:**
  - Simulator Generator implementation is in progress and we will have an example simulator in several months.
  - Power model is designed in progress.
ORISAL and DSP Library Porting

- We have designed an preliminary pseudo assembly language:
  - A *speed-critical* or *size-critical* program written by pseudo assembly could retarget to another platform more easily than compiler.
  - Pseudo assembly with machine description annotations provides a different layer for code optimizations in compiler toolkits, especially for *library optimizations*.

- Status:
  - We are going on with implementing an pseudo assembler.
  - We are starting to write an pseudo assembly based DSP library and keep on enhancing the features of our pseudo assembly design.
Compiler Optimization for Low Power on Power Gating

Yi-Ping You
Chingren Lee
Jenq Kuen Lee

Programming Language Lab.
National Tsing Hua University
Motivation

- Power dissipated while components are idling
- **Static/Leakage power** accounts for the majority of power dissipated when the circuit is inactive
- Clock gating doesn’t help reduce leakage power
Significance of Leakage Power

- As transistors become smaller and faster, static/leakage power becomes an important factor
  - Deep submicron CMOS circuits
- Leakage power soon becomes comparable to dynamic power
Trends in Dynamic and Static Power Dissipation (From Intel)

![Graph showing trends in dynamic and static power dissipation](image-url)
Leakage Power Trend in Temperature (0.13um) (From Intel)

0.13um, 15mm die, 1V
Leakage Power Trend in Temperature (0.1um) (From Intel)

![Leakage Power Trend Graph]

- Dynamic
- Static

0.1um, 15mm die, 0.7V

- Power (Watts)
- Temperature (°C)
- 30 40 50 60 70 80 90 100 110
- 6% 9% 14% 19% 26% 33% 41% 49% 56%
Static/Leakage Power Reduction

- $P_{static} = V_{cc} \ast N \ast K_{design} \ast I_{leak}$

- Partition circuits into several domains operating at different supply voltages
- Reduce number of devices
- Use more efficient circuits
  - Technology parameter
  - Subthreshold leakage
Power Gating

- Sleep transistor to power on or power off the circuit
- Used to turn off useless components in processors

DAC’97 Kao et al.
Machine Architecture

Program Counter

Instruction Bus (32bits)

Instruction Decoder

Micro Codes

Integer ALU/Normal Operation

Integer Multiplier

Floating Point Adder

Floating Point Multiplier

Floating Point Divider

Constant Supplying Voltage

Input/Output (64bits)

Input/Output (64bits)

Input/Output (64bits)

Power Gating Control Register (64bits)

Integer Registers (64bits x 32)

Floating Point Registers (64bits x 32)
Objective

- Use compiler analysis techniques to analyze program behaviors
  - Data-flow analysis
- Insert power gating instructions into proper points in programs
  - Find the maximum inactive intervals
  - Employing power gating if necessary
Component-Activity Data-Flow Analysis

\[ \text{comp\_in}[B] = \bigcup \text{comp\_out}[P] \]

P a predecessor of B

\[ \text{comp\_out}[B] = \text{comp\_gen}[B] \cup (\text{comp\_in}[B] \setminus \text{comp\_kill}[B]) \]
A component-activity is

- generated at a point $p$ if a component is required for this executing
- killed if the component is released by the last request
Data-Flow Analysis Algorithm for Component Activities

Begin
for each block $B$ do begin
  /* computation of $\text{comp}_\text{gen}$ */
  for each component $C$ that will be used by $B$ do begin
    RemainingCycle[$B$][$C$] := $N$, where $N$ is the number of cycles needed for $C$ by $B$;
    $\text{comp}_\text{gen}[B]$ := $\text{comp}_\text{gen}[B] \cup C$;
  end
end
for each block $B$ do begin
  $\text{comp}_\text{in}[B]$ := $\text{comp}_\text{kill}[B]$ := $\emptyset$;
  $\text{comp}_\text{out}[B]$ := $\text{comp}_\text{gen}[B]$;
end
while changes to any $\text{comp}_\text{out}$ occur do begin
  /* iterative analysis */
  for each block $B$ do begin
    for each component $C$ do begin
      /* computation of $\text{comp}_\text{kill}$ */
      RemainingCycle[$B$][$C$] := $\text{MAX}(\text{RemainingCycle}[P][C])$ -1), where $P$ is a predecessor of $B$;
      if RemainingCycle[$B$][$C$] = 0 then $\text{comp}_\text{kill}[B]$ := $\text{comp}_\text{kill}[B] \cup C$;
    end
    $\text{comp}_\text{in}[B]$ := $\cup \text{comp}_\text{out}[P]$, where $P$ is a predecessor of $B$; /* computation of $\text{comp}_\text{in}$ */
    $\text{comp}_\text{out}[B]$ := $\text{comp}_\text{gen}[B] \cup (\text{comp}_\text{in}[B] - \text{comp}_\text{kill}[B])$; /* computation of $\text{comp}_\text{out}$ */
  end
End
Example for comp_gen_set Computation

Instruction sequence

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$</td>
<td>ALU</td>
</tr>
<tr>
<td>$I_2$</td>
<td>Multiplier</td>
</tr>
<tr>
<td>$I_3$</td>
<td>Divider</td>
</tr>
<tr>
<td>$I_4$</td>
<td>Data Bus</td>
</tr>
<tr>
<td>$I_5$</td>
<td>ALU</td>
</tr>
<tr>
<td></td>
<td>Data Bus</td>
</tr>
<tr>
<td>$I_6$</td>
<td>others</td>
</tr>
</tbody>
</table>

Mapping table

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Component</th>
<th>Execution Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$</td>
<td>ALU</td>
<td>3</td>
</tr>
<tr>
<td>$I_2$</td>
<td>Multiplier</td>
<td>4</td>
</tr>
<tr>
<td>$I_3$</td>
<td>Divider</td>
<td>2</td>
</tr>
<tr>
<td>$I_4$</td>
<td>Data Bus</td>
<td>1</td>
</tr>
<tr>
<td>$I_5$</td>
<td>ALU</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Data Bus</td>
<td>2</td>
</tr>
<tr>
<td>$I_6$</td>
<td>others</td>
<td>-</td>
</tr>
</tbody>
</table>
### Example for comp_gen_set Computation (Cont.)

<table>
<thead>
<tr>
<th>Block</th>
<th>CycleCount</th>
<th>comp_gen_set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ALU</td>
<td>Mul</td>
</tr>
<tr>
<td>B₁</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B₂</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>B₃</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>B₄</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B₅</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B₆</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B₇</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>B₈</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B₉</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B₁₀</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Example for Component-Activity Data
Flow Analysis (1/4)

<table>
<thead>
<tr>
<th>Block</th>
<th>comp_gen_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>B_1</td>
<td>0000</td>
</tr>
<tr>
<td>B_2</td>
<td>1000</td>
</tr>
<tr>
<td>B_3</td>
<td>0001</td>
</tr>
<tr>
<td>B_4</td>
<td>0100</td>
</tr>
<tr>
<td>B_5</td>
<td>0000</td>
</tr>
<tr>
<td>B_6</td>
<td>0000</td>
</tr>
<tr>
<td>B_7</td>
<td>0001</td>
</tr>
<tr>
<td>B_8</td>
<td>0010</td>
</tr>
<tr>
<td>B_9</td>
<td>0000</td>
</tr>
<tr>
<td>B_{10}</td>
<td>1000</td>
</tr>
<tr>
<td>B_{11}</td>
<td>0000</td>
</tr>
<tr>
<td>B_{12}</td>
<td>0010</td>
</tr>
<tr>
<td>B_{13}</td>
<td>0001</td>
</tr>
<tr>
<td>B_{14}</td>
<td>0000</td>
</tr>
</tbody>
</table>

B_1: I_6  
B_2: I_1  
B_3: I_4  
B_4: I_2  
B_5: I_6  
B_6: I_6  
B_7: I_4  
B_8: I_3  
B_9: I_6  
B_{10}: I_1  
B_{11}: I_6  
B_{12}: I_3  
B_{13}: I_4  
B_{14}: I_6
Example for Component-Activity Data Flow Analysis (2/4)

<table>
<thead>
<tr>
<th>Block</th>
<th>Initial</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>comp_in_set</td>
</tr>
<tr>
<td>$B_1$</td>
<td>0000</td>
</tr>
<tr>
<td>$B_2$</td>
<td>0000</td>
</tr>
<tr>
<td>$B_3$</td>
<td>0000</td>
</tr>
<tr>
<td>$B_4$</td>
<td>0000</td>
</tr>
<tr>
<td>$B_5$</td>
<td>0000</td>
</tr>
<tr>
<td>$B_6$</td>
<td>0000</td>
</tr>
<tr>
<td>$B_7$</td>
<td>0000</td>
</tr>
<tr>
<td>$B_8$</td>
<td>0000</td>
</tr>
<tr>
<td>$B_9$</td>
<td>0000</td>
</tr>
<tr>
<td>$B_{10}$</td>
<td>0000</td>
</tr>
<tr>
<td>$B_{11}$</td>
<td>0000</td>
</tr>
<tr>
<td>$B_{12}$</td>
<td>0000</td>
</tr>
<tr>
<td>$B_{13}$</td>
<td>0000</td>
</tr>
<tr>
<td>$B_{14}$</td>
<td>0000</td>
</tr>
</tbody>
</table>
### Example for Component-Activity Data Flow Analysis (3/4)

<table>
<thead>
<tr>
<th>Block</th>
<th>Pass 1</th>
<th>Pass 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>in</td>
<td>kill</td>
</tr>
<tr>
<td>B₁</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>B₂</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>B₃</td>
<td>1000</td>
<td>0000</td>
</tr>
<tr>
<td>B₄</td>
<td>1001</td>
<td>0001</td>
</tr>
<tr>
<td>B₅</td>
<td>1100</td>
<td>1000</td>
</tr>
<tr>
<td>B₆</td>
<td>0100</td>
<td>0000</td>
</tr>
<tr>
<td>B₇</td>
<td>0100</td>
<td>0000</td>
</tr>
<tr>
<td>B₈</td>
<td>0101</td>
<td>0101</td>
</tr>
<tr>
<td>B₉</td>
<td>0010</td>
<td>0000</td>
</tr>
<tr>
<td>B₁₀</td>
<td>0010</td>
<td>0010</td>
</tr>
<tr>
<td>B₁₁</td>
<td>1000</td>
<td>0000</td>
</tr>
<tr>
<td>B₁₂</td>
<td>0010</td>
<td>0010</td>
</tr>
<tr>
<td>B₁₃</td>
<td>0010</td>
<td>0000</td>
</tr>
<tr>
<td>B₁₄</td>
<td>0011</td>
<td>0011</td>
</tr>
</tbody>
</table>
## Example for Component-Activity Data

### Flow Analysis (4/4)

<table>
<thead>
<tr>
<th>Block</th>
<th>Component-Activity</th>
<th>Data Bus</th>
<th>comp_out_set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ALU</td>
<td>Multiplier</td>
<td>Divider</td>
</tr>
<tr>
<td>B₁</td>
<td>INACTIVE</td>
<td>INACTIVE</td>
<td>INACTIVE</td>
</tr>
<tr>
<td>B₂</td>
<td>ACTIVE</td>
<td>INACTIVE</td>
<td>INACTIVE</td>
</tr>
<tr>
<td>B₃</td>
<td>ACTIVE</td>
<td>INACTIVE</td>
<td>INACTIVE</td>
</tr>
<tr>
<td>B₄</td>
<td>ACTIVE</td>
<td>ACTIVE</td>
<td>INACTIVE</td>
</tr>
<tr>
<td>B₅</td>
<td>INACTIVE</td>
<td>ACTIVE</td>
<td>INACTIVE</td>
</tr>
<tr>
<td>B₆</td>
<td>INACTIVE</td>
<td>ACTIVE</td>
<td>INACTIVE</td>
</tr>
<tr>
<td>B₇</td>
<td>INACTIVE</td>
<td>ACTIVE</td>
<td>INACTIVE</td>
</tr>
<tr>
<td>B₈</td>
<td>INACTIVE</td>
<td>INACTIVE</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>B₉</td>
<td>INACTIVE</td>
<td>INACTIVE</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>B₁₀</td>
<td>ACTIVE</td>
<td>INACTIVE</td>
<td>INACTIVE</td>
</tr>
<tr>
<td>B₁₁</td>
<td>ACTIVE</td>
<td>INACTIVE</td>
<td>INACTIVE</td>
</tr>
<tr>
<td>B₁₂</td>
<td>INACTIVE</td>
<td>INACTIVE</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>B₁₃</td>
<td>INACTIVE</td>
<td>INACTIVE</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>B₁₄</td>
<td>INACTIVE</td>
<td>INACTIVE</td>
<td>INACTIVE</td>
</tr>
</tbody>
</table>
Cost Model

- \( E_{\text{turn-off}}(\text{Component}) + E_{\text{turn-on}}(\text{Component}) \)
  \[ \text{BreakEven}_{\text{Component}} \times P_{\text{static}}(\text{Component}) \]

- Left hand side:
  - Energy consumed when power gating employed

- Right hand side:
  - Normal energy consumption
Scheduling Policies for Power Gating

- **Basic_Blk_Sched**
  - Schedule power gating instructions in a given basic block

- **MIN_Path_Sched**
  - Schedule power gating instructions by assuming the minimum length among plausible program paths

- **AVG_Path_Sched**
  - Schedule power gating instructions by assuming the average length among plausible program paths
MIN\_Path\_Sched (Based on Depth-First-Traveling)

MIN\_Path\_Sched(\(C, B, Branched, Count\)) {
    if block \(B\) is the end of CFG then return \(Count\);
    if block \(B\) has two children then do {
        if \(C \notin \text{comp\_out}[B]\) then do { // conditional branch, inactive
            \(Count := Count + 1;\)
            \(l\_Count := r\_Count := Count;\)
            if left edge is a forward edge then
                \(l\_Count := \text{MIN\_Path\_Sched}(C, \text{left child of } B, \text{TRUE}, Count);\)
            if right edge is a forward edge then
                \(r\_Count := \text{MIN\_Path\_Sched}(C, \text{right child of } B, \text{TRUE}, Count);\)
            if \(\text{MIN}(l\_Count, r\_Count) > \text{BreakEven}_C\) and !\(Branched\) then
                schedule power gating instructions at the head and tail of inactive blocks;
            return \(\text{MIN}(l\_Count, r\_Count);\)
        } else { // conditional branch, active
            if \(Count > \text{BreakEven}_C\) and !\(Branched\) then
                schedule power gating instructions at the head and tail of inactive blocks;
            if left edge is a forward edge then
                \(l\_Count := \text{MIN\_Path\_Sched}(C, \text{left child of } B, \text{FALSE}, Count);\)
            if right edge is a forward edge then
                \(r\_Count := \text{MIN\_Path\_Sched}(C, \text{right child of } B, \text{FALSE}, Count);\)
            return \(Count;\)
        }
    }
}
else {
    if \( C \not\in \text{comp_out}[B] \) then do { // statements except conditional branch, inactive
        \( \text{Count} := \text{Count} + 1; \)
        if edge is a forward edge then
            \text{return MIN\_Path\_Sched}(C, \text{child of } B, \text{Branched}, \text{Count});
        else
            \text{return Count;}
    } else {
        // statements except conditional branch, active
        if \( \text{Count} > \text{BreakEven}_C \) and !\text{Branched} then
            schedule power gating instructions at the head and tail of inactive blocks;
        if the edge pointing to \text{child of } B \text{ is a forward edge then}
            \text{MIN\_Path\_Sched}(C, \text{left child of } B, \text{FALSE}, \text{Count});
        \text{return Count;}
    }
}
}
Experimental Environment

- Alpha-compatible architecture
- Incorporated into the compiler tool
- SUIF and MachSUIF
- Evaluated by Wattch power estimator, which is based on SimpleScalar architectural simulator
# Alpha 21264 Power Components

<table>
<thead>
<tr>
<th>Component</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Clock Network</td>
<td>32%</td>
</tr>
<tr>
<td>Instruction Issue Units</td>
<td>18%</td>
</tr>
<tr>
<td>Caches</td>
<td>15%</td>
</tr>
<tr>
<td>Floating Execution Units</td>
<td>10%</td>
</tr>
<tr>
<td>Integer Execution Units</td>
<td>10%</td>
</tr>
<tr>
<td>Memory Management Unit</td>
<td>8%</td>
</tr>
<tr>
<td>I/O</td>
<td>5%</td>
</tr>
<tr>
<td>Miscellaneous Logic</td>
<td>2%</td>
</tr>
</tbody>
</table>

DAC’98
Digital Equipment Corp.
Benchmarks

- Collections of common benchmarks of FAQ of comp.benchmarks USENET newsgroup,
  - hanoi, heapsort, nsieve, queens, tfftdp, shuffle, eqntott, …
Power Gating on FPAdder for nsieve

![Bar chart showing Power Gating on FPAdder for nsieve](image-url)

- **Legend:**
  - BASIC_BLK_Sched
  - MIN_Path_Sched
  - AVG_Path_Sched

**Y-axis:**
- Power (Watts)

**X-axis:**
- Clock Gating
- BreakEven Cycle
Power Gating on FPMultiplier for nsieve
Power Gating on FPAdder
(BreakEven=32)

![Power Gating on FPAdder Graph](image)

- **Clock_Gating**
- **Power_Gating, BASIC_BLK_Sched**
- **Power_Gating, MIN_Path_Sched**
- **Power_Gating, AVG_Path_Sched**
Power Gating on FPMultiplier (BreakEven=32)
Summary

- We investigated the compiler analysis techniques related to reducing leakage power
  - Component-activity data-flow analysis
  - Power gating scheduling
- Our approach reduces power consumption against clock gating
  - Average 82% for FPUnits
  - Average 1.5% for IntUnits
  - 0.78%~14.58% (avg. 9.9%) for total power consumption
Conclusion

- Architecture design and system software arrangements are playing an increasingly important role for energy reductions.
- We present research results on compiler supports for low-power.
- Reference projects: NSF/NSC Project (3C), ITRI CCL Project, MOEA Project 2002-2005 (Pending).