Timing Analysis for Sensor Network Nodes of the Atmega Processor Family

By: Sibin Mohan, Frank Mueller, David Whalley, and Christopher Healy
Introduction

- Networks of Embedded systems (EmNets)
- Atmel Atmega family CPUs
- Limited Research into timing constraints on these architectures

Goal: Provide a timing framework of tools
- Tool should give worst-case execution time (WCET)
- WCET should be a tight bound of actual timing
- WCET should be safe (never underestimate actual)
Verification of result correctness

Use a 3 step system

- Compile and run code on actual hardware
- Run same code on cycle-accurate simulator (provided by hardware manufacturer)
- Run same code through developed timing analysis framework
Types of Timing Analysis:

- **Dynamic**
  - Simulates execution on worst case input
  - WCET Safety can not be guaranteed
  - Can be difficult to determine worst input set
  - Hardware/Software interactions can hide worst-case
    - Architectural complexities: pipelines and cache

- **Static**
  - Examines code
  - Analyzes all possible execution paths
  - Combines paths to construct worst case execution
  - Does not trace code execution
  - Does not take variable state into consideration
  - Guarantees WCET
Timing Analysis Framework

- Performs Static Timing Analysis
- Originally designed for SPARC I
- Modified to support Atmel architecture
- Enhanced to provide tighter WCET bounds
- Uses Fixed-point algorithm to determine WCET of loops and functions
- WCET tree constructed:
  - Paths -> loops -> functions -> program
Timing Analysis Framework

- **Program information:**
  - Compiler produces Control Flow
  - Loop bounds through analysis or programmer

- **Hardware information**
  - Cache description and behavior
  - Pipeline description
Timing Analyzer: Pipeline

- Pipeline Simulator handles for each path
  - Structural Hazards
  - Data Hazards
  - Branch Prediction
  - Cache Misses
Timing Analyzer: Path Analysis

- Takes path info from pipeline simulator
- Longest execution path selected
- Fixed-Point algorithm for loops
  - Uses longest path of loop body
  - Faster each iteration (benefit from cache)
  - Stop when cache stops improving execution of body
  - Can now bound WCET of loop
Modifications to Architecture

Variable Cycle Instructions

Example:

- Branch: 1 cycle if not taken
  - Fall through to next instruction
- Branch: 2 cycle if taken
  - Memory lookup of target instruction required

Past approaches would assume always max

- Overly pessimistic, especially when in loops
- Unnecessarily bloats WCET estimates
Variable Cycle Instructions

- During fixed-point algorithm, instructions that modify control flow are analyzed
- Solution: modify length of path chosen by this instruction
- For branch example: assume instruction takes 2 cycles and reduce not-taken path by one cycle to compensate.
- Fixed-point algo will operate as before and produce tighter WCET bound
Variable Cycle Instructions

Red cycles by 1 on this path

branch instruction with variable execution cycles

Not Taken Path (A, B, D)

Taken Path (A, C, D)
Modifications to Architecture

Pipeline Modeling across Loops Iterations

- Simulators tend to estimate WCET of loops by \((\text{loop count}) \times (\text{body WCET})\)
- Ignores pipeline state between iterations
- Common errors when compensating
  - Place blocks end to end producing noop (ci)
  - Place IF phase end to end overlapping EX phase of pipeline (cii)
Instructions IF EX

(a) Single Iteration of Loop

(b) Correct Handling of Loop Iterations

OR

IF and EX NOT Overlapped Extra Cycle Introduced

(c) Incorrect Handling of Loop Iterations

(i) Over-estimation of WCET

(ii) Under-estimation of WCET

IF of instr1 overlaps with EX of instr3 of previous iter
Atmega Architecture

- Atmega128 / Atmega103 processors
- CMOS 8-bit RISC controller
- Separate memory for program and data
- Separate bus for program and data
- Two stage pipeline IF & EX
- No cache
Instruction Set

- 16 bit or 32 bit wide instructions
- Integer based, floating point in emulation only
- Almost all instructions are 1 or 2 cycles
  - organized into 2 categories for analysis
- Some variable cycle instructions (loads, compares, branches...)
  - Handled through modifications described
Experimentation

- Benchmarks
  - C-Lab embedded WCET suite
  - NesC benchmarks
- Worst-case measurements in terms of processor cycle count
- Hardware timing obtained using interrupt-driven routines and hardware counters
  - Two hardware counters initialized to 0
  - Increment counter 1 at each cycle
  - At overflow of counter 1, counter 2 incremented
- Worst-case input sets manually constructed
- Same assembly output from compiler used for all 3 levels of experiment
Hardware overhead compensation

\[ O_1 = \text{overhead for starting and stopping timers [cycles]} \]
\[ O_2 = \text{overhead per invocation of overflow handler [cycles]} \]

\[ x = \text{value of cycle counter} \]
\[ y = \text{value of overflow counter} \]

We note that an overflow occurs once every 65,536 cycles, because we use a 16 bit counter as the cycle counter.

Then, the total execution time for the benchmark is obtained as follows:

\[ \text{total\_time} = y \times 2^{16} + x \]

Accounting for the start and stop overhead, we get:

\[ wcet' = \text{total\_time} - O_1 \]

Now, accounting for the overflow handling overhead, we obtain our final WCET estimate:

\[ wcet = wcet' - (y \times O_2) \]
Hardware/Simulator vs. Timing Analyzer Mismatch

- Analyzer provides WCET for loops and functions which can leave out initialization blocks
- Hardware and simulator can provide arbitrary block WCET
- Causes result discrepancies
- Must be compensated for when comparing WCET
Timing Analysis for NesC

- Programming Language for applications running on the TinyOS platform
- Defined especially for distributed embedded wireless sensor networks
- Built on C
- NesC compiler converts to intermediate C code
- Timing analysis can be performed on intermediate code
- Loop bounds can be determined manually from C code
## Experiment Results

### Initialization block compensation

### Simulator underestimated!

### Loop stalls removed

### Variable Cycle Instructions

<table>
<thead>
<tr>
<th>C Benchmark</th>
<th>Mica Motes Before Adjustment</th>
<th>Mica Motes After Adjustment</th>
<th>Simulator Before Adjustment</th>
<th>Simulator After Adjustment</th>
<th>Ratio</th>
<th>Initial Results</th>
<th>Ratio</th>
<th>After Pipeline Fix</th>
<th>Ratio</th>
<th>After Var. Instr. Fix</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum array</td>
<td>141,524</td>
<td>141,500</td>
<td>141,521</td>
<td>141,497</td>
<td>0.99</td>
<td>161,498</td>
<td>1.14</td>
<td>141,500</td>
<td>1.00</td>
<td>141,600</td>
<td>1.00</td>
</tr>
<tr>
<td>fibcall</td>
<td>151</td>
<td>145</td>
<td>146</td>
<td>140</td>
<td>0.96</td>
<td>258</td>
<td>1.78</td>
<td>202</td>
<td>1.39</td>
<td>146</td>
<td>1.01</td>
</tr>
<tr>
<td>insertsort</td>
<td>1,629</td>
<td>1,613</td>
<td>1,622</td>
<td>1,606</td>
<td>0.99</td>
<td>1,978</td>
<td>1.23</td>
<td>1,880</td>
<td>1.17</td>
<td>1,861</td>
<td>1.15</td>
</tr>
<tr>
<td>matrix mult</td>
<td>1,851</td>
<td>1,845</td>
<td>1,848</td>
<td>1,842</td>
<td>0.99</td>
<td>2,318</td>
<td>1.26</td>
<td>2,070</td>
<td>1.12</td>
<td>1,878</td>
<td>1.01</td>
</tr>
<tr>
<td>bubble sort</td>
<td>3,628,249</td>
<td>3,628,239</td>
<td>3,628,249</td>
<td>3,628,239</td>
<td>1.00</td>
<td>3,900,998</td>
<td>1.08</td>
<td>3,650,000</td>
<td>1.01</td>
<td>3,776,518</td>
<td>1.04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NcsC Benchmark</th>
<th>Mica Motes Before Adjustment</th>
<th>Mica Motes After Adjustment</th>
<th>Simulator Before Adjustment</th>
<th>Simulator After Adjustment</th>
<th>Ratio</th>
<th>Initial Results</th>
<th>Ratio</th>
<th>After Pipeline Fix</th>
<th>Ratio</th>
<th>After Var. Instr. Fix</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArraySum</td>
<td>86</td>
<td>81</td>
<td>97</td>
<td>92</td>
<td>1.14</td>
<td>105</td>
<td>1.30</td>
<td>87</td>
<td>1.07</td>
<td>88</td>
<td>1.09</td>
</tr>
<tr>
<td>RC5:encrypt</td>
<td>15,956</td>
<td>15,951</td>
<td>15,951</td>
<td>15,946</td>
<td>1.00</td>
<td>17,958</td>
<td>1.13</td>
<td>16,088</td>
<td>1.00</td>
<td>16,088</td>
<td>1.00</td>
</tr>
<tr>
<td>RC5:decrypt</td>
<td>15,860</td>
<td>15,855</td>
<td>15,855</td>
<td>15,850</td>
<td>1.00</td>
<td>17,982</td>
<td>1.13</td>
<td>16,112</td>
<td>1.01</td>
<td>16,122</td>
<td>1.01</td>
</tr>
</tbody>
</table>

Note: All ratios are with respect to Mica Motes "After Adjustment"
Scalability of Timing Analyzer

Input size scaled for fibcall benchmark