Workload-basierte Verlässlichkeitsanalyse für eingebettete Prozessoren

Oliver Bringmann, Stefan Stattelmann, Björn Sander

Universität Tübingen / FZI Forschungszentrum Informatik
Outline

- Embedded Software: Power & Temperature
- System-Level Simulation of Non-Functional Properties by Host-Compiled Execution
- Application-Dependent Power and Temperature Simulation Framework
- Areas of Application
Challenges: Automotive Electronics

Yesterday …

… and today

<table>
<thead>
<tr>
<th></th>
<th>A8</th>
<th>A4</th>
<th>A8 new</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of ECUs</td>
<td>68</td>
<td>65</td>
<td>85</td>
</tr>
<tr>
<td>Number of Busses</td>
<td>6</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Software (Mbyte)</td>
<td>60</td>
<td>90</td>
<td>&gt;230</td>
</tr>
</tbody>
</table>

Source: Bosch

Source: Audi
Automotive Electronics – Reducing Comfort Area

- **Feature size [µm]**
  - 0.35 µm
  - 0.18 µm
  - 0.09 µm
  - 0.025 µm

- **Year**
  - 1995
  - 2000
  - 2005
  - 2010
  - 2015

- **Comfort area**

- **Maturity Time**

- **Automotive semiconductor**

- **Semiconductor (cutting-edge)**
Temperatures: Dominated by Local Power Effects

- “…thermal modeling at finer granularity level i.e., transistor level or logic gate level is required for more accurate estimation of local hot spots.” Bansal et al., ASP-DAC 2006

- “…hot spots tens of micrometers in diameter with heat fluxes in excess of 1000 W/cm²…” Shakouri, Proceedings of the IEEE, 2006

- Typical ESL approach: Model devices by Power State Machines
  - Example: ARM7TDMI
    - Power RUN state: 1.3 mW =>
      Power density: 3.9 W/cm²

**Conclusion:** Non-Functional Properties

- Performance
- Energy and Power Consumption
- Temperature and Power Densities

have to be considered and are strongly application-dependent

Example: ARM7TDMI

- Power RUN state: 1.3 mW =>
  Power density: 3.9 W/cm²

**Assuming an equal distribution is probably not accurate**
Example: Power Simulation at Different Levels of Abstraction

- ARM7TDMI, 90 nm, 236 MHz, 0.18 mm²

<table>
<thead>
<tr>
<th>Functional Representation</th>
<th>Power Model</th>
<th>gcd</th>
<th>ellip</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Simulation Factor (Simulation Time/ Simulated Time)</td>
<td>Average Power - Error (comp. to MS+PC)</td>
</tr>
<tr>
<td>Task Graph, CDG, CFG</td>
<td>Average</td>
<td>&lt;&lt; 1</td>
<td>4.17 uW - 81%</td>
</tr>
<tr>
<td>Instrum. Source-Code</td>
<td>Instruction-Dependent</td>
<td>0.5</td>
<td>5.8 uW - 31%</td>
</tr>
<tr>
<td>Static BinaryTransl., Dynamic Binary Transl.</td>
<td>Data-Dependent</td>
<td>75</td>
<td>9.69 uW - 28%</td>
</tr>
<tr>
<td>Netlist: ModelSim (MS) + Power Compiler (PC)</td>
<td>PC Internal</td>
<td>&gt; 10^10</td>
<td>7.57 uW</td>
</tr>
</tbody>
</table>

Data-dependent power model leads to a small error. But usually some kind of RTL simulation is necessary to calculate input stimuli of the components.

Simulation speed and appearing error are acceptable. Efficient link with information from binary level could be very promising.

Sander, Bringmann [CODES-ISSS 2009]
SYSTEM-LEVEL SIMULATION OF NON-FUNCTIONAL PROPERTIES BY HOST-COMPiled EXECUTION
Simulation of Functional and Non-Functional Behavior – Basic Idea

- Processor Characteristics (Timing, Power, Power Densities)
- NFP Instrumentation
  - SW w/o timing & power
  - Software with timing & power
- Platform Integration
  - SW w/o timing & power
- Platform Simulation
  - HW platform
  - HW/SW platform
Basic Idea: Source-Level Timing Instrumentation

**Proposed Hybrid Approach**

- **Compilation into binary code**
- **Static execution time analysis** w.r.t. architectural details
  - Back-annotation of analyzed timing information into the source code
- **Simulation** by host-compiled execution

```c
int f( int a, int b, int c, int d )
{
    int res;
    res = (a + b) << c - d;
    delay( 3 ms );
}
```

**Important**

- Requires accurate relation between source code and binary code
- Run-time models for **branch prediction** and **caching** have to be incorporated
Compiler Optimizations

- Structure of source code and binary code can be completely different
  - Function Inlining adds basic blocks
  - Loop Unrolling modifies execution count of basic blocks
  - ...

- Compilers don’t generate accurate debug information for optimized code
  - No 1:1 relation between source-level and binary-level basic blocks
  - Simply annotating delay attributes for source-level timing simulation does not work

How to match structure of source code and machine instructions?
C/C++ Source Code

main.c
...
23    int i = 1;
24    for (int c = 0; c <= pow; c++) {
25       i = i * 2;
26    }
27    pot = i;
...

Binary Executable

a.out
...
0x8000 ADD r1, r0, #1
0x8004 ADD r2, r0, #0
0x8008 CMP r2, r3
0x800C BGT 0x801C
0x8010 MUL r1, r1, #2
0x8014 ADD r2, r2, #1
0x8018 B 0x8008
0x801C MOV r5, r3
...

Structural Analysis

Extract Debug Information

Source-Level Dominator Relation

Binary-to-Source Mapping

Line Information:
0x8000 → main.c:23
0x800C → main.c:24
0x8010 → main.c:25
0x8014 → main.c:24
0x801C → main.c:27

Binary-Level Dominator Relation

Extract Control Flow

Binary-Level Control Flow Graph

Reconstruct Debug Information

Binary-Level Dominator Relation

Low-Level Timing Analysis

Corrected Mapping

Timing-Annotated Control Flow Graph

Instrumentation

Low-Level Path Analysis

void bb (int address)
{
    ...
    if (lastBlock == 0x8000
        && nextBlock == 0x8010)
        delay (0x8000, 0x8010);
    lastBlock = nextBlock;
    ...

Annotated C/C++ Source Code

Path Simulation Code

Model for Source-Level Simulation
Structural Analysis and Code Matching

```
main.c
...
23  int i = 1;
24  for (int c = 0; c <= pow; c++) {
25    i = i * 2;
26  }
27  pot = i;
...
```

C/C++ Source Code

Target Compiler

Binary Executable

Structural Analysis

Extract Debug Information

Extract Control Flow

Binary-to-Source Mapping

Source-Level Dominator Relation

Reconstruct Debug Information

Binary-Level Dominator Relation

Binary-Level Control Flow Graph

Line Information:

<table>
<thead>
<tr>
<th>Address</th>
<th>Source Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8000</td>
<td>main.c:23</td>
</tr>
<tr>
<td>0x800C</td>
<td>main.c:24</td>
</tr>
<tr>
<td>0x8010</td>
<td>main.c:25</td>
</tr>
<tr>
<td>0x8014</td>
<td>main.c:24</td>
</tr>
<tr>
<td>0x801C</td>
<td>main.c:27</td>
</tr>
</tbody>
</table>

Example code snippet:

```
a.out
...
0x8000 ADD r1, r0, #1
0x8004 ADD r2, r0, #0
0x8008 CMP r2, r3
0x800C BGT 0x801C
0x8010 MUL r1, r1, #2
0x8014 ADD r2, r2, #1
0x8018 B 0x8008
0x801C MOV r5 r3
...```
Annotation and Path Simulation Code Generation

Reconstruct Debug Information

Corrected Mapping

Line Information:
- 0x8000 → main.c:23
- 0x8010 → main.c:25
- 0x801C → main.c:27

Instrumentation

Low-Level Path Analysis

Low-Level Timing Analysis

Timing-Annotated Control Flow Graph

void delay (int lastBlock, int nextBlock) {
    int cycles = 0;
    switch (nextBlock) {
        case 0x8010:
            cycles += pipeline_delay (0x8010);
            cycles += cache_delay (0x8010);
            cycles += branch_delay (lastBlock, 0x8010);
            break;
        ...
    }
    wait (cycles * cycle_time);
}

void bb (int address) {
    ...
    if (lastBlock == 0x8000
        && nextBlock == 0x8010)
        delay (0x8000, 0x8010);
    ...
    lastBlock = nextBlock;
}
Simulation of Functional and Non-Functional Behavior – Basic Idea

- Processor Characteristics (Timing, Power, Power Densities)
- NFP Instrumentation
  - SW w/o timing & power
  - Software with timing & power
- Platform Integration
  - HW platform
  - HW/SW platform
- Platform Simulation
Platform Model Integration

Instrumented Software Module

void bb (int address)
{
  ...
  if (previous == 0x800C && address == 0x801C)
    delay (0x800C, 0x801C);
  ...
  previous = address;
}

Annotated C/C++ Source Code
Path Simulation Code

Architectural Model

Cache Model

Branch Prediction Model

Update
Adjust
Sync
Adjust

Virtual Platform

CPU
Bus
CPU
I/O

TLM-2.0 Loosely Timed Simulation

S. Stattelmann, O. Bringmann et al. [DATE 2011]
Experimental Results

- Experiments conducted for an ARM processor
- Improved simulation performance compared to high-performance ISS based on just-in-time compilation
APPLICATION-DEPENDENT POWER AND TEMPERATURE SIMULATION FRAMEWORK
Power and Power Density Characterization

Commercial Flow
- Processor Designer
- VHDL RTL
- Design Compiler
- VHDL Gate-Level Netlist
- ModelSim
- Switching Activities
- Power Compiler

Component Characterization
- Power Models
- Distribution Models

Processor Characteristics (Timing, Power, Power Densities)

NFP Compiler

Software with timing & power

Platform Integration

Platform Simulation

SW w/o timing & power

HW platform

HW/SW platform
Starting Point:
- Commercial Tool Chain
- ARM-like processor design

⇒ Used to create accurate power measurements for processor components
Gate Average Power Consumption

Significant variations due to data dependencies
Characterization: Power Distribution Models

Linear slope = Equal power density

Equal distribution is not a good option!
Characterization: Power Distribution Models

Register Set

Memory

FE

FE/DC

DC

DC/EX

EX

Manageable power distribution models applicable at ESL

Sander, Bringmann [CODES-ISSS 2009]
Power Density Models for Temperature Analysis

Outside Thermal Flow
Uniform power density models

Self-heating
Power distribution

"Reality"
Power at gate level incl. layout
## Low Power Design: What can be done…

<table>
<thead>
<tr>
<th>Low Power Technique</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Multi-Voltage Threshold (MVTH)</strong></td>
<td>Individual logic gates use transistors with low threshold voltages</td>
</tr>
<tr>
<td>MVTH</td>
<td></td>
</tr>
<tr>
<td><strong>Clock Gating (CG)</strong></td>
<td>Disable registers</td>
</tr>
<tr>
<td>CG</td>
<td></td>
</tr>
<tr>
<td><strong>Operand Isolation (OI)</strong></td>
<td>Datapath blocks are prevented from switching (keep inputs constant)</td>
</tr>
<tr>
<td>OI</td>
<td></td>
</tr>
<tr>
<td><strong>Multiple Supply Voltages (MSV)</strong></td>
<td>Blocks operate with different supply voltages</td>
</tr>
<tr>
<td>MSV</td>
<td></td>
</tr>
<tr>
<td><strong>Dynamic Voltage and Frequency Scaling (DVFS)</strong></td>
<td>Adjust voltage and frequency on the fly depending on the workload</td>
</tr>
<tr>
<td>DVFS</td>
<td></td>
</tr>
<tr>
<td><strong>Power Gating (PG)</strong></td>
<td>Turn off supply voltage when not in use</td>
</tr>
<tr>
<td>PG</td>
<td></td>
</tr>
<tr>
<td><strong>Multi-Core (MC)</strong></td>
<td>Distribute functionality to more than a single processor</td>
</tr>
<tr>
<td>MC</td>
<td></td>
</tr>
</tbody>
</table>

For system model integration, 3 different classes of low power techniques can be identified:

- **"Original circuit" is changed**
- **Additional hardware is used**
- **Architectural measures**

Functionality on 2 processors:
- \( f / 2, 0.7 \cdot V \)
- Power cut in half
Incorporation of Low Power Techniques

- Platform Description
  - Apps
  - Allocation
  - Mapping
  - OS
  - Low Power

- Commercial Flow
  - Processor Designer
  - VHDL RTL
  - Design Compiler
  - VHDL Gate-Level Netlist
  - ModelSim
  - Switching Activities
  - Power Compiler

- Exploration
  - Enhance model generation process
  - Temperature/Reliability

- Model Generation
  - Adaptation of component characterization for original circuit, i.e. component power calculation

- Component Characterization
  - Power Models
  - Distribution Models

- ESL Platform Model
  - Functional Representation
  - Component Characterizations

- Implementation
  - Application-specific power values

- Derivation of new component characterizations

- PG, MC, MSV, DVFS
Parameterization of the System Model

```plaintext
1.118*interval_instructions[3]+ // STR
0.595*interval_instructions[7]+ // STM
1.201*interval_instructions[0]+ // ORR
...)*1e-12*pow(voltage/0.9, 2.0)
/power_interval.to_seconds()
+8.0E-9*area[6]*(voltage/0.9);
```
AREAS OF APPLICATION
Thermal Simulation

Scheduling Core 0

Core 0 Core 1

Trigger temperature-dependent robustness and reliability analyses
Triggering of RT / Gate Level Aging Models

Operating Conditions

Modeled Effects
- NBTI
- HCI

Aged Component Model
- gate delay
- output slope
  \(\Rightarrow\) aged critical path

Workload

Annotated C/C++ Source Code

... 23 int i = 0; bb (0x8000);
24 for (int c = 0; c <= pow; c++) {
25    i = i * 2; bb (0x8010);
26 }
27 pot = i; bb (0x801C);
...
Platform for Fast Holistic Vehicle Analysis

- Fast co-simulation to cover application and environment context
- Virtual prototype (VP) reflects timing/power/temperature behavior
- Foundation for application/environment-driven robustness analysis
Conclusion

- Non-functional properties important issue in embedded SW design
- Real-time simulation possible using source code instrumentation
  - Timing and power characteristics are annotated
  - Impact of data-dependencies and access to shared resources are solved by dynamic execution
  - Highly scalable in terms of the number of processors/processor cores
- Applicable to other non-functional properties like determination of thermal distributions and application-dependent reliability analysis
- Foundation for efficient simulation of heterogeneous HW/SW systems including sensors and actors
Thank you very much for your attention!

Questions?

Oliver Bringmann
FZI Forschungszentrum Informatik
Intelligent Systems and Production Engineering (ISPE)
Email: bringmann@fzi.de