Complete Formal Verification of a Family of Automotive DSPs

Rafal Baranowski, Marco Trunzer
Robert Bosch GmbH, Reutlingen, Germany
Challenge

• Verification of an in-house family of Digital Signal Processors (DSP) with safety requirements
  – three-stage SIMD pipeline with in-order execution

• DSP family members vary in:
  – instruction set (ASIP-like features)
  – number and width of data paths and registers
  – safety features, peripheral functionality, etc.

• Family is open
  – future variants are yet to be specified
Goals

• Good maintainability of the verification framework
  – little effort to verify new DSPs
• Early verification start
• 100% functional coverage at affordable cost
Agenda

• Introduction
• Completeness Concepts
• Verification Process
• Semi-Formal Specification
• Implementation & Results
• Conclusion
Complete Formal Specification

• Formal specification is complete when any two implementations adhering to it are equivalent
  [Bormann & Busch 2005, Claessen 2007]

• Complete formal specification is derived:
  – manually from informal specifications
    [Bormann et al. 2007, OneSpin GapFree Methodology]
  – automatically from intermediate formal models
    [Kühne et al. 2010, Loitz et al. 2010]
Operational Property

• Basic component of a complete formal specification
• Defined as **implication** over time
  – e.g.

```
  A1   A2
  C1   C2   C3
  n    n+1  n+2  n+3  n+4
  start    end
```

• Has a defined **start** and **end** cycle for sequencing
  – end cycle = start cycle of the next operational property
  – sequencing defined with a “property graph”
Completeness Requirements

- Every **execution trace** must seamlessly match to a **sequence of properties**, such that:
  - around every start cycle, the antecedent of exactly one property is satisfied
  - in every clock cycle, the state of all relevant signals is unambiguously **determined** (specified) by the consequents of consecutive properties

[Bormann et al. 2007]

```
reset
0 1 1 1 1 0 1 0 1 0 0 1 0 1 0 1 1 0 1 0 1 ...
```

```
determined
0 1 1 1 1 0 1 0 1 0 0 1 0 1 0 1 1 0 1 0 1 ...
```

```
execution 0 0 0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 0 1 0 1 0 ...
```

```
property P1 P2 P3 ...
```

```
signal
0 1 1 1 1 0 1 0 1 0 0 1 0 1 0 1 1 0 1 0 1 ...
```
Agenda

• Introduction
• Completeness Concepts
• Verification Process
• Semi-Formal Specification
• Implementation & Results
• Conclusion
Traditional Formal Verification Process

- Informal specification
- Spec. gaps found?
- white-/gray-box approach

- RTL implementation
- Property development
- Late verification start
- Specification/RTL changes are costly

- Formal RTL verification
- Completeness check
Verification Process with Semi-Formal Specification

- **Formal spec.**
  - Automatic translation to properties
  - Early completeness check
  - Formal RTL verification
  - Early completeness check

- **Informal spec.**
  - Manual property development

- Offload what does not fit

- Early verification start
- Early completeness check
- Low effort spec. changes
Agenda

• Introduction
• Completeness Concepts
• Verification Process
• Semi-Formal Specification
• Implementation & Results
• Conclusion
Semi-Formal Specification

Core functionality
(instr. decode, execution, ...)

covered by
formal (tabular) specification

Address decoders

covered by
informal specification

Safety monitors

Stacks

“auxiliary functionality”
Tabular (Formal) Specification

- Built upon instruction set description
- Defines **triggers** and **commitments** of instructions
- Each row is a cycle: start cycle = n, end cycle = n+1
- Example:

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction [n]</th>
<th>PC [n+1]</th>
<th>ACC [n+2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>00000000</td>
<td>PC[n]+1</td>
<td>stable</td>
</tr>
</tbody>
</table>
Tabular (Formal) Specification (2)

- Features:
  - bit patterns
  - expressions in SystemVerilog
  - overlay rows for conditional execution, interrupts, etc.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>00000000</td>
<td>PC[n]+1</td>
<td>stable</td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>ADD D ACC</td>
<td>01aadddd</td>
<td>PC[n]+1</td>
<td>ACC(a)[n+1] + D(d)[n+1]</td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>JSR addr</td>
<td>10bbbbbb</td>
<td>PC[n]+1</td>
<td>stable</td>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>cycle 2</td>
<td>cccccccc</td>
<td>cb</td>
<td>stable</td>
<td>push</td>
<td>PC[n]</td>
</tr>
</tbody>
</table>

stack functionality in informal spec.
Informal Specification

• Functionality specified **informally** if:
  – complex and used by many instructions
  – would bloat tabular representation
  – e.g. stacks, queues, interrupt controllers, etc.

• Off-loading to informal specification allows to:
  – improve clarity and reduce redundancy of tabular representation
  – improve verification efficiency
Agenda

• Introduction
• Completeness Concepts
• Verification Process
• Semi-Formal Specification
• Implementation and Results
• Conclusion
Automatic Property Generation

• Straightforward translation:
  – input: tabular specification (standard spreadsheet)
  – output: properties and constraints (SystemVerilog/TiDAL)

• Implementation: Java Emitter Templates (JET)

• Followed rules:
  – keep the translator simple (3400 LOC)
  – minimum design knowledge in the translator
  – one property per multi-cycle instruction (simplifies completeness check)
## Property Example

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction [n]</th>
<th>INT_REQ [n]</th>
<th>PC [n+1]</th>
<th>ACC [n+2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD D ACC</td>
<td>01aaddd</td>
<td>0</td>
<td>PC[n]+1</td>
<td>ACC(a)[n+1] + D(d)[n+1]</td>
</tr>
<tr>
<td><strong>interrupt</strong></td>
<td><strong>overlay row</strong></td>
<td>1</td>
<td>*<em>INT_NO[n]<em>2</em></em></td>
<td></td>
</tr>
</tbody>
</table>

**property** prop_ADD_D_ACC;

```verilog
reg [p_instr_width-1:0] opcode;

// variable for the opcode

// TRIGGERS
...;
```

```verilog
// store opcode
implies

// COMMITMENTS
...;
```

```verilog
// ACC[0] modified or stable
```

```verilog
// ACC[1] modified or stable
...;
```

```verilog
endproperty
```
Verification Summary

• Specification includes:
  – **tabular**: about 260 rows & 60 columns
  – **informal**: address generation, I/O and interrupt control, stacks, queues, MAC

• Implementation effort:
  – 7 person months in total (incl. debug of 5 DSPs)
  – easy management

• Achieved **100% functional coverage** for tabular spec.
  – found about 50 spec. issues (mostly gaps) and 10 corner case bugs
  – typical compute time: 5 h

• Just a **few hour’s work** to apply to a new DSP!
  – with added / removed / modified instructions
  – with changed data path width, amount of data paths and registers, etc.
Agenda

- Introduction
- Completeness Concepts
- Verification Process
- Semi-Formal Specification
- Implementation and Results
- Conclusion
Conclusion

• Good trade-off between spec. precision and readability
  – 100% functional coverage for tabular spec.
  – spec. development requires little/no formal background

• Improved maintainability and reusability
  – late specification/implementation changes not critical

• Weaker dependency on RTL
  – early verification start
  – early completeness check
Questions
Literature

• Bormann, J., Beyer, S., Maggiore, A., Siegel, M., Skalberg, S., Blackmore, T., and Bruno, F. 2007. Complete formal verification of TriCore2 and other processors. In Design and Verification Conference (DVCon’07).


