Industrial Strength Refinement Checking

Jesse Bingham, John Erickson, Gaurav Singh, and Flemming Andersen
Intel IAG
FMCAD 2009
Introduction

- Standard approach to FV of HW protocols
  - Develop high level model (HLM) in guarded-command-like language (e.g. Murphi, TLA, Spin etc)
  - Write invariants, e.g. cache coherence
  - Model check as big as you can

- So the HLM is golden, but what about the implementation (RTL)?
  - Ideal: prove that RTL implements HLM… hard!
  - Our solution: test that RTL implements HLM during dynamic simulation
  - check == test in this talk/paper
Key point #1

The ingredients needed for equivalence testing are also needed to prove implementation.

⇒ might as well start with testing
What should Implements Mean?

- What does it mean for RTL to implement HLM? They have different
  - execution semantics
  - state variables/representations
  - rule atomicity (HLM has more)
  - rule concurrency (RTL has more)
- Not always clear [Vardi FMCAD09]
- For our domain, we found a notion we call behavioral refinement appropriate…
  - Similar to notion of Bluespec and also super-scalar processor verification literature
Behavioral Refinement

RTL Behavior (i.e. simulation)...
reset state
one RTL clock cycle
Behavioral Refinement

Murphi Behavior (witness)

- initial state
- a rule fires

RTL Behavior (i.e. simulation)

- reset state
- one RTL clock cycle
Behavioral Refinement

Murphi Behavior (witness)

initial state

RTL Behavior (i.e. simulation)

reset state

a rule fires

one RTL clock cycle

Refinement map

...
Each RTL clock cycle corresponds to zero or more rules firing.
Idea: at each RTL cycle, select what sequence of rules are about to fire
How Refinement Checker Works

**Idea**: at each RTL cycle, select what sequence of rules are about to fire
How Refinement Checker Works

**Idea**: at each RTL cycle, select what sequence of rules are about to fire

Diagram:
- Murphi \( \rightarrow RM(r) \rightarrow \ldots \rightarrow Next \)
- RTL simulation \( \rightarrow r \)

Rule selection
**How Refinement Checker Works**

**Idea:** at each RTL cycle, *select* what sequence of rules are about to fire.

![Diagram showing rule selection process](image)
Example: Toy Cache Controller

- **CPU**
- **Cache Controller**
- **Main Memory**
## Toy Cache in Murphi

A toy cache in Murphi is shown with a state transition diagram. The diagram involves the following transition:

- **Cpu2Cache**
- **CacheArray**
- **Cache2Mem**

### CacheArray

<table>
<thead>
<tr>
<th>State</th>
<th>Addr</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dirty</td>
<td>0xC54</td>
<td>0x823E</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Clean</td>
<td>0x6D7</td>
<td>0x01</td>
</tr>
</tbody>
</table>

This table illustrates the state transitions in the cache system.
Ruleset i : CacheIndex "Evict"
    CacheArray[i].State != Invalid
==> if (CacheArray[i].State == Dirty) begin
    Cache2Mem.opcode := WriteBack;
    Cache2Mem.Addr = CacheArray[i].Addr;
    Cache2Mem.Data = CacheArray[i].Data;
end;
    CacheArray[i].State := Invalid;
end
Receiving a Store Request

Ruleset i : CacheIndex "Recv_Store"

Cpu2Cache.opcode = Store &

( ( CacheArray[i].State != Invalid &
    CacheArray[i].Addr = Cpu2Cache.Addr) |
  ( addr_misses_in_cache(Cpu2Cache.Addr) &
    CacheArray[i].State = Invalid ) )

==> CacheArray[i].Data := Cpu2Cache.Data;
CacheArray[i].State := Dirty;
Absorb(Cpu2Cache);
end
Cache Controller RTL

Cpu2Cache

Pipe stage 1

Eviction Logic

Hit?

Cache State & Addr Array

Cache Data Array

Pipe stage 2

Cpu2Mem
Example RTL Behavior

Pipe stage 1:
- Store(A0,D0)

Eviction Logic:
- Hit?

Pipe stage 2:
- WriteBack(A1,D1)
- Store(A0,D0)

Cache State & Addr Array:
- Dirty,A0

Cache Data Array:
- D0

Store
Evict

Example RTL Behavior

Pipe stage 1:
- Store(A0,D0)

Eviction Logic:
- Hit?

Pipe stage 2:
- WriteBack(A1,D1)
- Store(A0,D0)

Cache State & Addr Array:
- Dirty,A0

Cache Data Array:
- D0

Store
Evict
Key point #2

Pipelining causes rules that are atomic in Murphi to be non-atomic in the RTL...

This non-atomicity is resolved by the refinement map & history variables
Murphi semantics fire one rule at a time, while RTL has true rule concurrency.

This is resolved by \textit{rule selection}, which picks a sequence of Murphi rules to fire @ each RTL clock cycle.
Example with Refinement Checker

Pipe stage 1

Pipe stage 2

Store(A0,D0)

WriteBack(A1,D1)

HLM

RTL

Evict

RecvStore

Cache State & Addr Array

Cache Data Array

Dirty,A0

D0
BTW: Everything’s System Verilog

- RTL design under verification
- Test stimulus
- Refinement Map
- Rule Selection
- High Level Model
  - in consultation with Architects
  - compiled into SV by a tool *mu2sv*

⇒ any off-the-shelf SV simulator works

Paper gives disciplined approach to writing SV code for these buggers

HW designers

HW validators

Us (FV team)
 mu2sv

• Translates a Murphi model into SV
• Typedefs, procedures, functions, procedures, invariants
• State variables get wrapped in a record type called MURPHI_STATE
• Murphi rule $R$ becomes SV function

    function MURPHI_STATE R_sv(MURPHI_STATE ms,...);

• Errors if invoked when $R$’s guard is false in $ms$
• Rule coverage logging
• Valuable feedback for test-writers
Inspiration

  - Used TLA+ & linked TLC model checker to simulation engine
  - Done as research *after* the project was complete
  - Showed that subtle bug would have been caught
Application: Hierarchical Cache Protocol

- 3 person months to develop
- Caught 8 bugs during just 1 month of deployment!

- Was not deployed due to chip cancellation ;-(
- Could allow up to 8 murphi rules to fire per RTL clock