Formal for Everyone
Challenges in Achievable Multicore Design and Verification

FMCAD 25 Oct 2012
Daryl Stewart
ARM is an IP company

- ARM licenses technology to a network of more than 1000 partner companies within the ARM® Connected Community®, spanning the semiconductor supply chain.

- ARM provides developers with intellectual property (IP) solutions in the form of:
  - CPUs/GPUs
  - Physical IP
  - Cache and SoC designs
  - Application-specific standard products (ASSPs)
  - Related software and development tools
Our Partners Supply the Silicon

- ARM silicon partners supply chips into 90% of smart phones, 80% of digital cameras, and 28% of all electronic devices – over 20 billion chips to date.

- ARM technology is used in a wide variety of applications ranging from mobile handsets and digital set top boxes to car braking systems and network routers.
ARM11™ MPCore™ processor

- 800MHz to 1 GHz+ in 65G at under 2 mm²
- 1 to 4 cores in an SMP cluster
- 32-bit SIMD for media processing
- Physically tagged caches
- Tightly coupled memories
- ARM TrustZone™ security
ARM Cortex™-A Series processors

- Applications processors for mobile computing
- Single to Quad core clusters
  - Fully coherent L1 cache via Snoop Control Unit
  - Accelerator Coherence Port shares cache with peripherals
- Multi cluster coherency with AMBA Coherency Extension
- Heterogeneous system with Cortex-A15/Cortex-A7 processor clusters: “ARM big.LITTLE™ processing”
  - AMBA®4 ACE™ interconnect
  - Shared interrupt controller
FORMAL IN ARM
## Avoidance, Hunting, Absence, Analysis

<table>
<thead>
<tr>
<th>Technique</th>
<th>Advantages</th>
<th>Avoiding Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug Avoidance</td>
<td>• Improve quality during design</td>
<td>Usually at block level</td>
</tr>
<tr>
<td></td>
<td>• Biggest ROI</td>
<td>– E.g. visualisation by designer</td>
</tr>
<tr>
<td></td>
<td>• Improve quality before property checks are run</td>
<td>May not involve tooling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– E.g. formal modeling, proofs</td>
</tr>
<tr>
<td>Bug Hunting</td>
<td>• Ease of set-up</td>
<td>False failures</td>
</tr>
<tr>
<td></td>
<td>• Corner cases</td>
<td>– Run at higher structural level</td>
</tr>
<tr>
<td></td>
<td>• Low cost, starts early in design process</td>
<td>– Only leads to wasted debug</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-exhaustive checks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– full proofs are welcome, but not required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-uniform run times</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– checks are run just for the time available.</td>
</tr>
<tr>
<td>Bug Absence</td>
<td>• Only way to get 100% assurance</td>
<td>Non-uniform run times</td>
</tr>
<tr>
<td></td>
<td>• Cover corner cases</td>
<td>– Use different proof engines with the tool</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Use “invariants” (helper properties) (this adds non-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uniform/non-predictable engineering time)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Use safe abstractions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Prove under certain condition (Add extra constraints)</td>
</tr>
<tr>
<td>Bug Analysis</td>
<td>• Ease of setup if constraints exist</td>
<td>Interactive generation of constraints to generate legitimate failure scenario</td>
</tr>
<tr>
<td></td>
<td>• Can investigate silicon bugs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Can confirm fix</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Bug Avoidance**
  - Improve quality before property checks are run
  - Biggest ROI

- **Bug Hunting**
  - Looking for bugs
  - Do not worry if proofs do not complete
  - Aim for “No failures”

- **Bug Absence**
  - Aim to get a “complete” set of properties
  - Aim to prove properties
    - under certain constraints

- **Bug Analysis**
  - For bugs in FPGA prototypes or in Silicon
    - write symptom of bug as a property, generate waveform
Formal in the Design Flow

- Formal used at
  - Low-level by designers: design bring-up & embedded properties
  - Medium-level by validation engineers: end-to-end properties
  - High-level by architects: architectural formal specification and validation
RTL Bugs Found by Method

<table>
<thead>
<tr>
<th></th>
<th>ARM1</th>
<th>Cortex-M0+</th>
</tr>
</thead>
<tbody>
<tr>
<td>3µ</td>
<td>24K T</td>
<td>32K T</td>
</tr>
<tr>
<td>6 My</td>
<td>11 My</td>
<td></td>
</tr>
<tr>
<td>2K Hours</td>
<td>1,439K Hours</td>
<td></td>
</tr>
</tbody>
</table>

- **Autochecks**
- **DAPTB**
- **flycatcher_dvs**
- **Formal**
- **Integration Kit**
- **Lint**
- **MBTB**
- **OS / Debug Tools**
- **Other**
- **Partner raised**
- **Review**
- **SBTB**
- **Speculation**
- **Synthesis**
- **Toplevel s/w Config**
- **v6m avs**
- **Seq-X**
- **Power Intent Checks**

**AVS**

**DVS**

**Formal**

**Integration**

**Constrained Random**

**Reviews**
Bottom Up Formal

- **Software Tools**
  - Each level relies on levels around it AND the Architectural behaviour
  - In return the Architecture expects certain behaviour

- **Architectural behaviour**
  - E.g. Deadlock freedom, power modes, coherency

- **Combine techniques to give chain of verification from RTL to Apps**
RTL verification

- Microarchitectural specification for designers is in natural language
- RTL level assertions as standard
  - Written by designers
- Difficult to write end to end properties in terms of RTL state
  - Architectural state is smeared across time and space, or implicit
- Use of abstract models written in SystemVerilog with refinement to RTL level
  - Describing lifecycle of transactions rather than block functionality
Formal for Designers

- Early bug discovery
- Higher quality sooner

The diagram shows a comparison between Testbench and Properties over time, with Testbench leading in early bug discovery and achieving higher quality sooner.
Proof Progress and Scaling

- Historically: hard to track progress of formal proof coverage
  - ARM developed progress metrics for proofs and methodology and deployed during a Bug Analysis project
  - Technique for partial proof allowing identification of bug free code
    - Enables focussed review and simulation for weakest blocks
- Historically: architectural properties involve too much RTL detail for tools to handle
  - Developed micro architectural model of SCU
  - SCU Transaction Ordering proven on this specification model
  - RTL shown to meet specification, hence RTL preserves transaction ordering
- These demonstrate proof is now measurable and scalable
Partial Proof

Unproven lemma D focuses Simulation and Review
Micro Architectural Models

- **Formal Model**
  - An abstraction expressed as transactors, FSMs, assumptions…
  - Provides vocabulary of abstract events

- **Desired Model Properties**
  - Properties which should arise from a correct implementation
    - Safety or liveness assertions

- **High Level Behaviour**
  - What implementation is sufficient?
    - assume to prove formal model exhibits desired properties
    - assert on RTL to deduce that it satisfies specification

- **Covers**
  - sanity check the formal specification
  - RTL bring up
Micro Architectural Models

High Level Behaviours imply Desired Properties
The architecture defines several envelopes of reliable behaviour:

- ISA – programmer’s view of instruction
- Weak Memory – implementation freedom, unintuitive behaviour
- Coherent interconnect – AMBA4 ACE transactions
- Power modes – domains, required functionality
- Security – Trustzone
- Debug and trace behaviour

How to verify individually and interdependently?

How to specify non-determinism?
Architecture Validation

- SystemVerilog model of AMBA4 ACE
- Deadlock discovered in draft specification using JasperGold
  - 4 master system, unlikely to find by hand

- Murphi model of AMBA4 ACE master with bridge to alternative interface for
  - Protocol deadlock
  - System coherency

- PReach Murphi
  - 25 threads, 1Tb
  - Smallest case completed
  - Several bugs found during development

<table>
<thead>
<tr>
<th>Master nodes</th>
<th>IDs</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>3 hours</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>-</td>
</tr>
</tbody>
</table>
Systems and Software

- System level testing
  - Requires accurate models of expected behaviour
  - Relate testing to coverage of specification

- What useful IP can we supply to our partners for software development?
Sequentially Consistent execution

ARM SB
"PodWR Fre PodWR Fre"

{R2=x; R3=y;}

P0
MOV R0, #1
STR R0, [R2]
LDR R1, [R3]

P1
MOV R0, #1
STR R0, [R3]
LDR R1, [R2]

Observe P0 end with R1=0 and P1 end with R1=1

{R3=y; R2=x;}

Ry0

PodWR

Rx1

Wx1

Py0

x=0

y=0

Wx1

Ty0

PodWR

Rf

Wx1

PodWR

Wx1

PodWR

Wy1

Rfe

Rfe

Rx1

LDR R1, [R3]

Program order candidate relations
PodWR = program order different address Write then Read

Coherency ordering (Communication) relations
Rfe = Target Reads its value from a source on an external processor
Fre = Source reads From a write that precedes target (on an external processor) in coherence order
Relaxing candidate relations

ARM SB
"PodWR Fre PodWR Fre"

{R2=x; R3=y;}

P0

MOV R0, #1
STR R0, [R2]

PodWR

PodWR

LDR R1, [R3]

Observe both threads ending with R1=0

Relaxing PodWR breaks the cycle

{x=0; y=0}

P1

MOV R0, #1
STR R0, [R3]

PodWR

PodWR

LDR R1, [R2]

Program order candidate relations

PodWR = Program order different address Write then Read

Coherency ordering (Communication) relations

Rfe = Target Reads its value from a source on an external processor

Fre = Source reads From a write that precedes target (on an external processor) in coherence order
The ARM Architecture Reference Manual

Second Edition

Edited by David Seal
### ARMv7 specification

#### Encoding A1

<table>
<thead>
<tr>
<th>cond</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>S</th>
<th>Rn</th>
<th>Rd</th>
<th>imm5</th>
<th>type</th>
<th>0</th>
<th>Rm</th>
</tr>
</thead>
</table>

if Rd == '1111' && S == '1' then SEE SUBS PC, LR and related instructions;
d = Uint(Rd); n = Uint(Rn); m = Uint(Rm); setflags = (S == '1');
(shift_t, shift_n) = DecodeImmShift(type, imm5);

#### Assembler syntax

```
ADC(S)<cp> {<Rd>,} <Rn>, <Rm> {,<shift>}
```

#### Operation

if ConditionPassed() then
  EncodingSpecificOperations();
  shifted = Shift(R[m], shift_t, shift_n, APSR.C);
  (result, carry, overflow) = AddWithCarry(R[n], shifted, APSR.C);
  if d == 15 then    // Can only occur for ARM encoding
    ALUWritePC(result); // setflags is always FALSE here
  else
    R[d] = result;
    if setflags then 
      APSR.N = result<31>
      APSR.Z = IsZeroBit(result);
      APSR.C = carry;
      APSR.V = overflow;
```
ARMv7 support functions

```c
(SRTypE, integer) DecodeImmShift(bits(2) type, bits(5) imm5)

case type of
when '00'
    shift_t = SRTypE_LSL; shift_n = UInt(imm5);
when '01'
    shift_t = SRTypE_LSR; shift_n = if imm5 == '00000' then 32 else UInt(imm5);
when '10'
    shift_t = SRTypE_ASR; shift_n = if imm5 == '00000' then 32 else UInt(imm5);
when '11'
    if imm5 == '00000'
        shift_t = SRTypE_RRX; shift_n = 1;
    else
        shift_t = SRTypE_ROR; shift_n = UInt(imm5);
return (shift_t, shift_n);

(bits(N), bit) Shift_C(bits(N) value, SRTypE type, integer amount, bit carry_in)
assert !(type -- SRTypE_RRX || amount == 1);
if amount -- 0 then
    (result, carry_out) = (value, carry_in);
else
    case type of
        when SRTypE_LSL
            (result, carry_out) = LSL_C(value, amount);
        when SRTypE_LSR
            (result, carry_out) = LSR_C(value, amount);
        when SRTypE_ASR
            (result, carry_out) = ASR_C(value, amount);
        when SRTypE_ROR
            (result, carry_out) = ROR_C(value, amount);
        when SRTypE_RRX
            (result, carry_out) = RRX_C(value, carry_in);
return (result, carry_out);
```
What ARM uses ISA spec for

CPU
- Design
- Licensing
- Validation
  - Test suites
  - Test tools

Models
- Design
- Validation

Dev Tools
- Asm/dasm/ld
- Compiler
- Debugger
- Validation
Summary

- “Systems design today is on the same level of development as mechanics in the middle ages - based on experiences with no formal theory of design.” J. Sifakis FMCAD 2010
- We have made good progress on pieces of the puzzle, designers are turning to formal to relieve the pain
- A combination of tools and techniques
  - Use those best suited to each problem domain
  - Must be able to relate to each other, and simulation
  - The system design does not end with us – enable partners
THE END