

# ARM formalized in HOL

---

Anthony Fox

Computer Laboratory  
University of Cambridge

# Overview

---

- What is HOL?
- Instruction set architectures.
- Formal verification of ARM6.
- Formal specification of ARMv7.

Interactive theorem proving: HOL

HOL4

---

# HOL4

---

- HOL4 is an interactive theorem prover for Higher Order Logic.

# HOL4

---

- HOL4 is an interactive theorem prover for Higher Order Logic.
- The current HOL4 system has a long history, starting with Robin Milner's Stanford LCF in 1972.

Stanford LCF → Edinburgh LCF → Cambridge LCF →  
HOL88 → HOL90 → HOL98 → HOL4

# HOL4

---

- HOL4 is an interactive theorem prover for Higher Order Logic.
- The current HOL4 system has a long history, starting with Robin Milner's Stanford LCF in 1972.

Stanford LCF → Edinburgh LCF → Cambridge LCF →  
HOL88 → HOL90 → HOL98 → HOL4

- A nice history of the development of HOL can be found from Mike Gordon's web page:

<http://www.cl.cam.ac.uk/~mjc/papers/HolHistory.html>

- Modern relatives include: HOL Light, Isabelle/HOL and ProofPower.

# HOL and the LCF approach

---

# HOL and the LCF approach

---

- The object language is essentially **typed lambda calculus**.
- The metalanguage is Standard ML. (Works with PolyML and Moscow ML.)

# HOL and the LCF approach

---

- The object language is essentially **typed lambda calculus**.
- The metalanguage is Standard ML. (Works with PolyML and Moscow ML.)
- The Edinburgh LCF approach is based on using ML's type system to ensure the soundness of results. Also known as a **fully expansive** approach (makes soundness bugs extremely rare).
- The primitive HOL inferences are encapsulated with an **abstract data type** for terms, types and theorems — this gives us a small “trusted logical kernel”.

# HOL and the LCF approach

---

- The object language is essentially **typed lambda calculus**.
- The metalanguage is Standard ML. (Works with PolyML and Moscow ML.)
- The Edinburgh LCF approach is based on using ML's type system to ensure the soundness of results. Also known as a **fully expansive** approach (makes soundness bugs extremely rare).
- The primitive HOL inferences are encapsulated with an **abstract data type** for terms, types and theorems — this gives us a small “trusted logical kernel”.
- Mike Gordon developed HOL with hardware verification in mind — using predicates to model circuits. It is now widely used in other areas.
- He carried out early processor verification work, e.g. Tamarack and Viper.

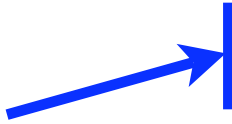
# Interactive proof in HOL4 (extremely short primer)

---

# Interactive proof in HOL4 (extremely short primer)

---

load theories  
and tools



```
> load "blastLib";  
val it = (): unit
```

# Interactive proof in HOL4 (extremely short primer)

---

load theories  
and tools

```
> load "blastLib";  
val it = (): unit
```

```
> val ExtendNot_def = Define`  
  ExtendNot b (w : word8) =  
    let v = ~w in  
      if b then sw2sw v else w2w v : word16`;
```

Definition has been stored under "ExtendNot\_def".

```
val ExtendNot_def =  
  |-  $\forall b w. \text{ExtendNot } b w = (\text{let } v = \neg w \text{ in if } b \text{ then } \text{sw2sw } v \text{ else } \text{w2w } v):$   
  thm
```

make definitions

# Interactive proof in HOL4 (extremely short primer)

---

load theories  
and tools

```
> load "blastLib";  
val it = (): unit
```

make definitions

```
> val ExtendNot_def = Define`  
  ExtendNot b (w : word8) =  
    let v = ~w in  
      if b then sw2sw v else w2w v : word16`;  
Definition has been stored under "ExtendNot_def".
```

```
val ExtendNot_def =  
  |-  $\forall b w. \text{ExtendNot } b w = (\text{let } v = \neg w \text{ in if } b \text{ then } \text{sw2sw } v \text{ else } \text{w2w } v)$ :  
  thm  
> g `!b1 b2 v. (7 >< 0) (ExtendNot b1 v + ExtendNot b2 (-v)) = -2w : word8`;  
val it =  
  Proof manager status: 1 proof.  
1. Incomplete goalstack:  
  Initial goal:  
   $\forall b1 b2 v. (7 >< 0) (\text{ExtendNot } b1 v + \text{ExtendNot } b2 (-v)) = -2w$ 
```

initiate a proof

```
:  
proofs
```

# Interactive proof in HOL4 (extremely short primer)

---

load theories  
and tools

```
> load "blastLib";  
val it = (): unit
```

make definitions

```
> val ExtendNot_def = Define`  
  ExtendNot b (w : word8) =  
    let v = ~w in  
      if b then sw2sw v else w2w v : word16`;  
Definition has been stored under "ExtendNot_def".
```

initiate a proof

```
val ExtendNot_def =  
  |-  $\forall b w. \text{ExtendNot } b w = (\text{let } v = \neg w \text{ in if } b \text{ then } \text{sw2sw } v \text{ else } \text{w2w } v)$ :  
  thm  
> g `!b1 b2 v. (7 >< 0) (ExtendNot b1 v + ExtendNot b2 (-v)) = -2w : word8`;  
val it =  
  Proof manager status: 1 proof.  
1. Incomplete goalstack:  
  Initial goal:  
   $\forall b1 b2 v. (7 >< 0) (\text{ExtendNot } b1 v + \text{ExtendNot } b2 (-v)) = -2w$ 
```

apply tactics

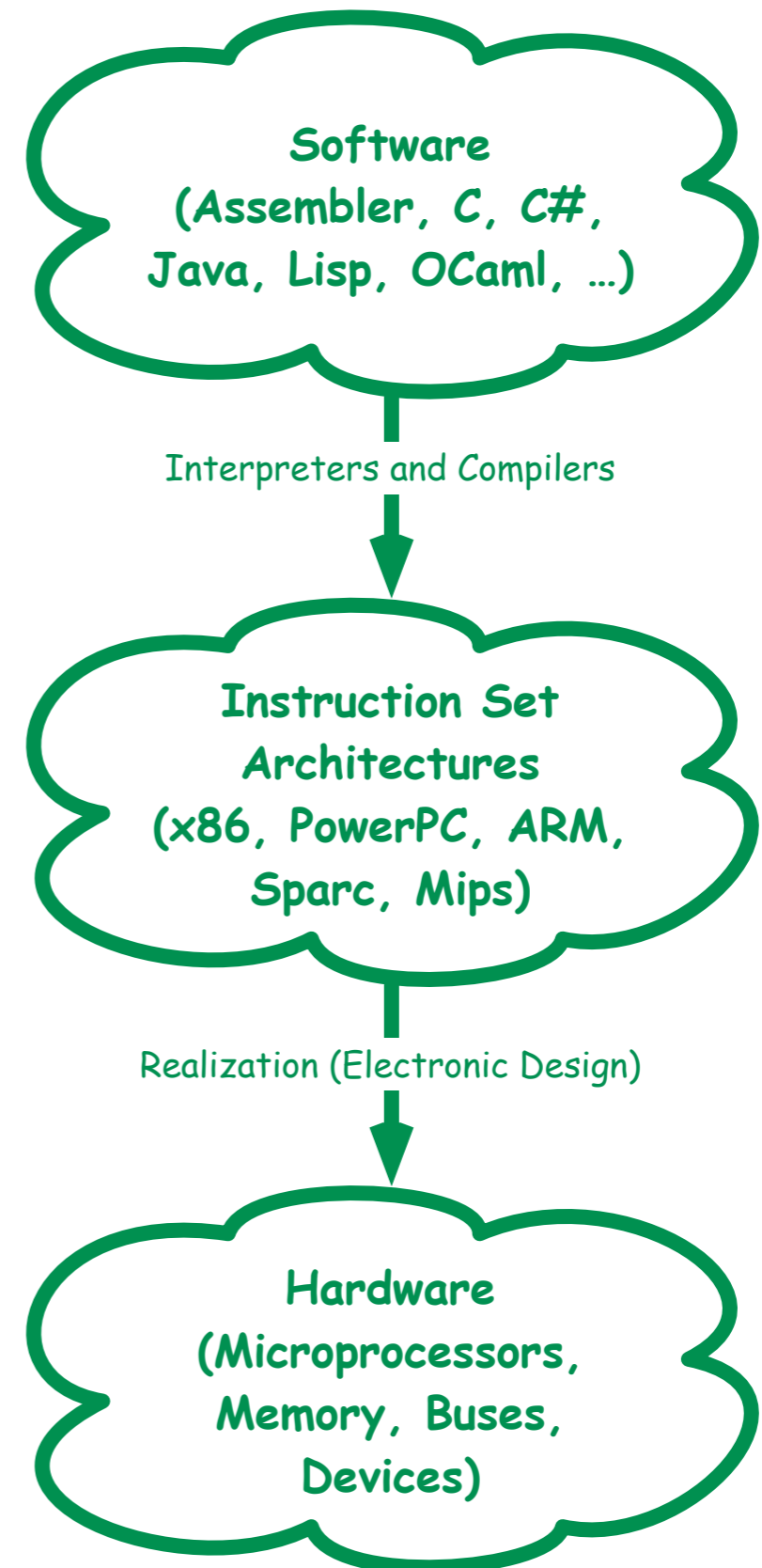
```
:  
  proofs  
> e (SRW_TAC [boolSimps.LET_ss] [ExtendNot_def] THEN blastLib.BBLAST_TAC);  
OK..  
val it =  
  Initial goal proved.  
|-  $\forall b1 b2 v. (7 >< 0) (\text{ExtendNot } b1 v + \text{ExtendNot } b2 (-v)) = -2w$ :  
  proof
```

# Instruction Set Architectures

# ISAs

---

- Instruction set architectures play an important role in computing.
- They provide an **interface** between hardware and software.
- ISA models are needed for reasoning about:
  - interpreters and compilers
  - operating systems (device drivers and I/O)
  - micro-architecture designs.



# ISA specification

---

- There are many instructions — even with RISC architectures.
- Architecture manuals are big, verbose and open to misinterpretation.
- Precise implementation details are often proprietary and protected by IP rights.
- Formal reasoning at the ISA level produces concrete results/artefacts with direct industrial relevance.
- There are not many complete, high-fidelity, formal models for commercial ISAs in the public domain.

# Formal verification of ARM6

# ARM6 verification

---

- In 2000, the EPSRC funded the project: “Formal Specification and Verification of ARM6”.
- ARM6 is 3-stage pipelined processor that implements the ARMv3 ISA.
- The principle investigators were Graham Birtwistle, then based at Leeds, and Mike Gordon at Cambridge.
- Two PhD students at Leeds: Dominic Pajak worked on a Standard ML model of the ISA and Daniel Schostak worked on detailed models of the ARM6 micro-architecture. (Went on to work at ARM full time.)
- Mike recruited me as a research associate to do a formal verification using the HOL4 proof assistant.

# ARM6 verification

---

- Modelled the ARM programmer's model and the ARM6 processor core in HOL. Covers all ARMv3 instructions.
- Defined state spaces and next state functions (the operational semantics) at both levels.
- Defined data and temporal abstraction maps.
- The overall approach came from Neal Harman and John Tucker at Swansea.

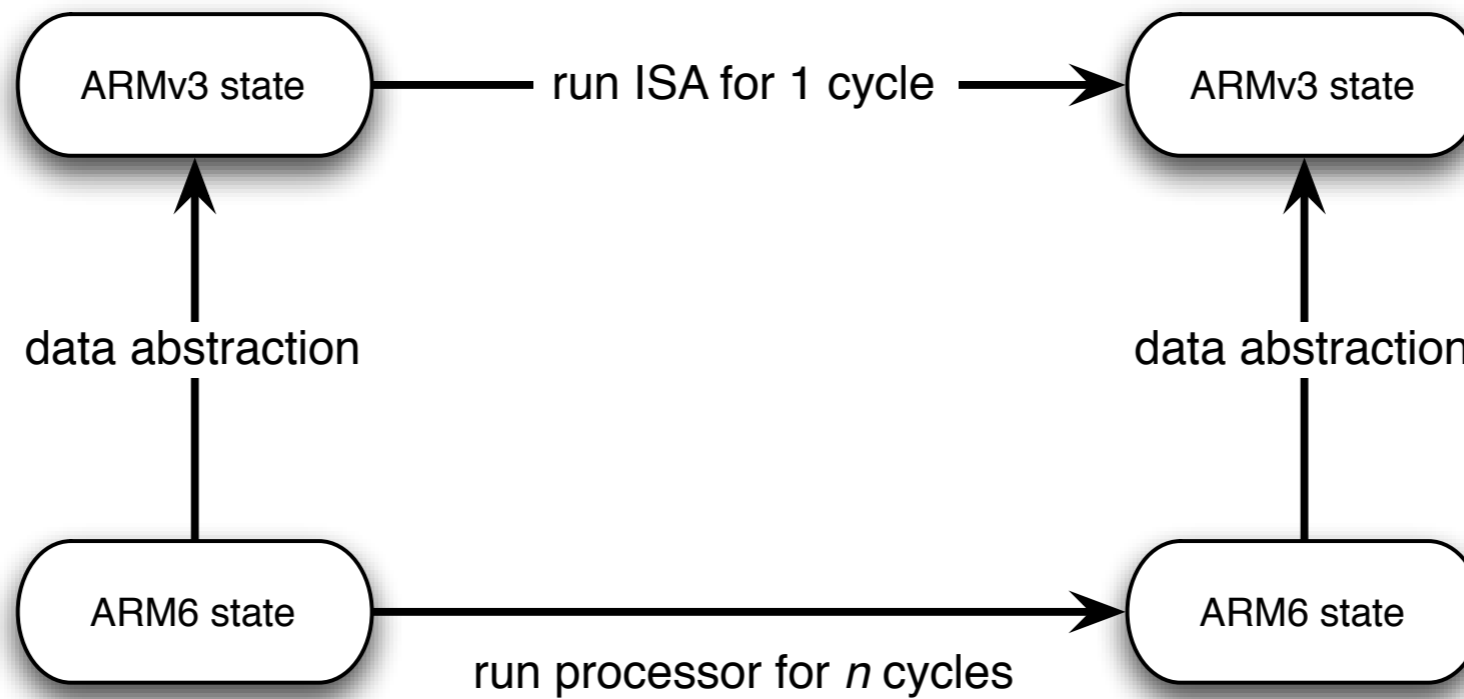
# ARM6 verification

---

# ARM6 verification

---

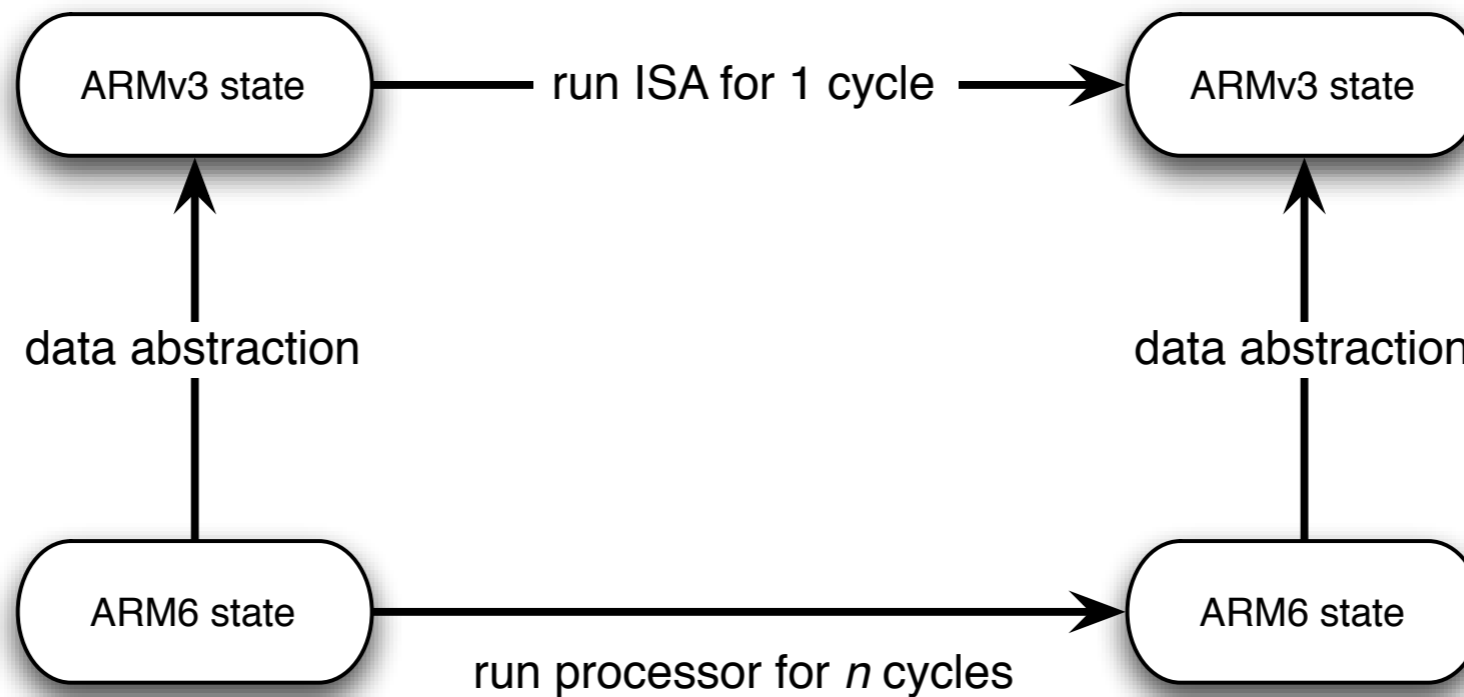
- Classic commutativity correctness requirement:



# ARM6 verification

---

- Classic commutativity correctness requirement:

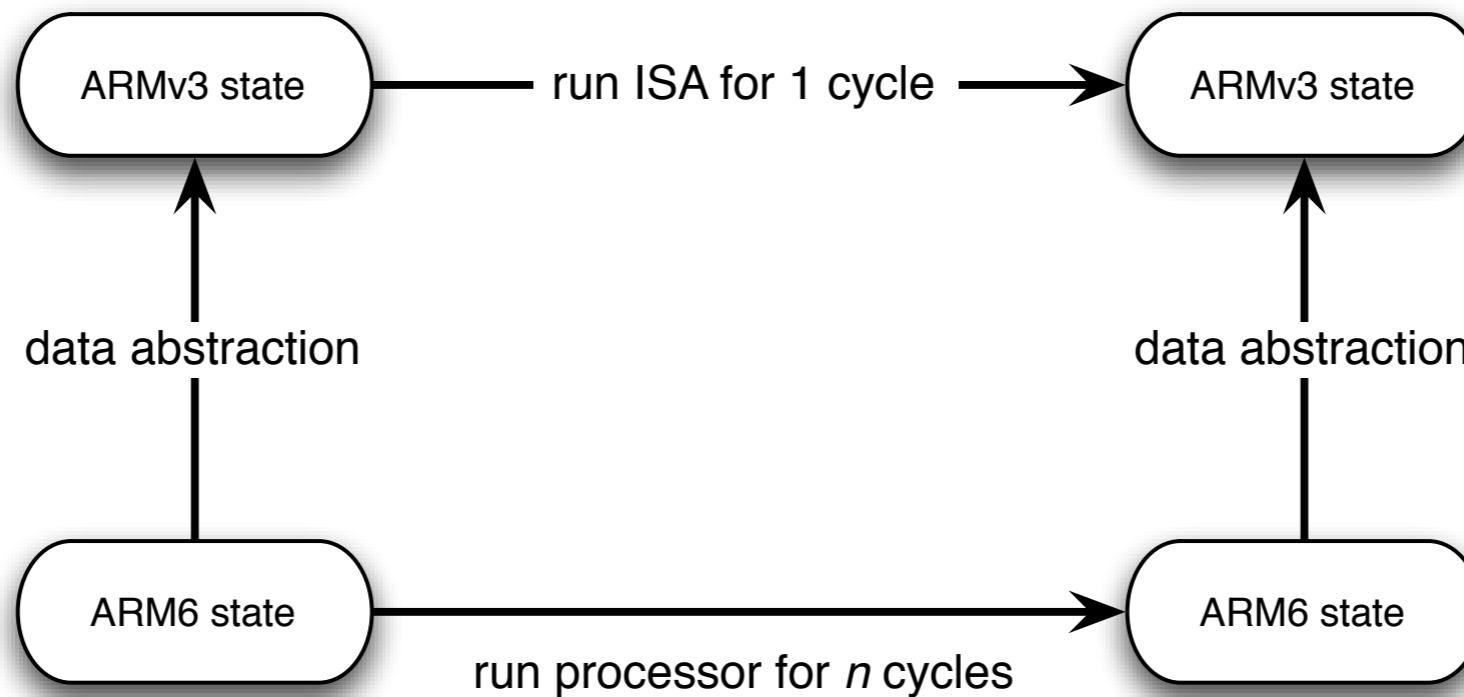


- Verification corresponds with full functional correctness, i.e. not just basic safety and liveness properties

# ARM6 verification

---

- Classic commutativity correctness requirement:



- Verification corresponds with full functional correctness, i.e. not just basic safety and liveness properties
- This provides a very high level of assurance.

Formal specification of ARMv7

# ARM machine code verification

---

# ARM machine code verification

---

- Work on ARM has continued at Cambridge.
- Emphasis has shifted to ISA specification and machine code verification.
- Magnus Myreen has developed an approach for verifying machine code.

# ARM machine code verification

---

- Work on ARM has continued at Cambridge.
- Emphasis has shifted to ISA specification and machine code verification.
- Magnus Myreen has developed an approach for verifying machine code.
- His approach is based on using Hoare triples with a separating conjunction operator (borrowed from separation logic). This avoids having to explicitly reason about the distinctness of resources (registers and memory addresses).
- His most advanced case study is a verified Lisp interpreter.
- He has developed compilation / de-compilation tools in HOL:

**HOL functions ↔ machine code.**

(Works for x86, PowerPC and ARM. All based on the LCF approach.)

# ARM ISA specification

---

# ARM ISA specification

---

- The ARMv3 model was initially extended to ARMv4T (adding a few more instructions).
- The latest Cortex processors implement ARMv7.
- A completely new **monadic** ISA specification was developed.

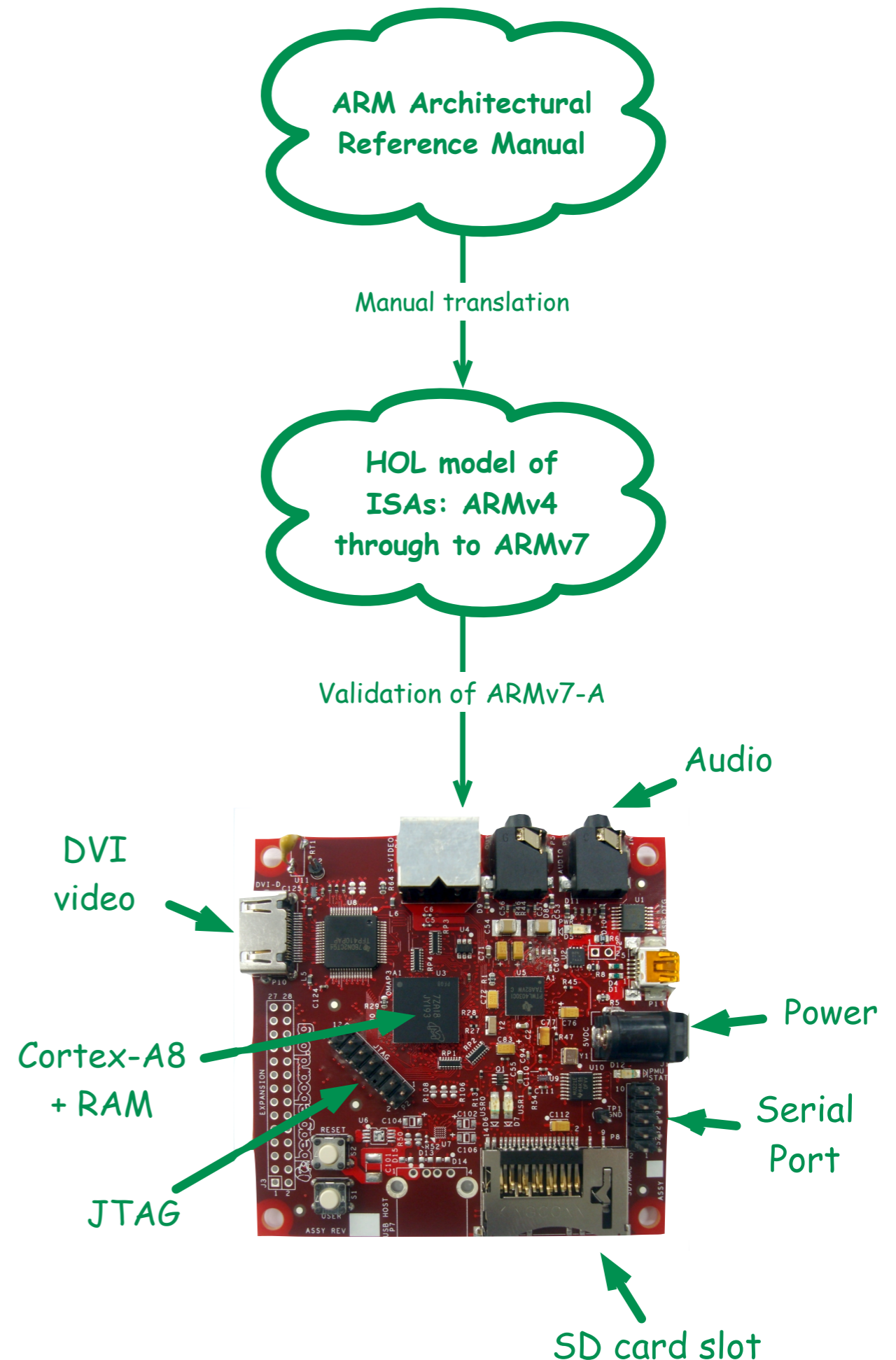
# ARM ISA specification

---

- The ARMv3 model was initially extended to ARMv4T (adding a few more instructions).
- The latest Cortex processors implement ARMv7.
- A completely new **monadic** ISA specification was developed.
- It simultaneously covers all current ISA versions:
  - ✓ Easily specialised to ARMv4, ARMv4T, ARMv5T, ARMv5TE, ARMv6, ARMv6K, ARMv6T2, ARMv7-A and ARMv7-R.
  - Doesn't cover ARMv7-M.
  - ✓ Covers Thumb, Thumb-2 and ThumbEE instructions.

# Latest ISA model

- Specification very closely follows pseudo-code from the ARM Architectural Reference Manual.
- Validated against development boards.
- Extensive tool support:
  - including custom assembler and evaluator.
- Can be “emitted” as SML for fast evaluation.
- **25K lines of development.**



# ARM instruction evaluator

[| Home](#) | [About](#) | [Learn more](#) | [Feedback](#) |

## Enter an instruction to run...

Architecture:

Instruction set:

Processor mode:

Byte order:

If-Then block:

Machine code:

Assembly code:

## Lookup an instruction...

Mnemonic:

## SCM Repositories - hol

Files shown: 28  
 Directory revision: 8700 (of 8706)  
 Sticky Revision:

Ads by Google

Outsourced Mailroom

Fast and secure! Your business dataat your fingertips in an instant.

[www.orsgroup.com](http://www.orsgroup.com)

File ▲	Rev.	Age	Author	Last log entry
Parent Directory				
<a href="#">eval/</a>	8607	4 weeks	acjf3	A few changes: - Make use of the new zDefine - Moved some theorems from arm_ste...
<a href="#">EXAMPLES</a>	8371	3 months	acjf3	Minor change capturing unpredictable case for LDM -- from latest ARM reference e...
<a href="#">Holmakefile</a>	8612	4 weeks	acjf3	Add a selftest program for ARM tools. (NB. takes around 10 mins to run on my ma...
<a href="#">README</a>	8607	4 weeks	acjf3	A few changes: - Make use of the new zDefine - Moved some theorems from arm_ste...
<a href="#">armLib.sig</a>	8607	4 weeks	acjf3	A few changes: - Make use of the new zDefine - Moved some theorems from arm_ste...
<a href="#">armLib.sml</a>	8607	4 weeks	acjf3	A few changes: - Make use of the new zDefine - Moved some theorems from arm_ste...
<a href="#">armScript.sml</a>	8371	3 months	acjf3	Minor change capturing unpredictable case for LDM -- from latest ARM reference e...
<a href="#">armSyntax.sig</a>	8295	4 months	acjf3	Update that adds support for ThumbEE. /* From the ARM reference... ThumbEE Is...
<a href="#">armSyntax.sml</a>	8295	4 months	acjf3	Update that adds support for ThumbEE. /* From the ARM reference... ThumbEE Is...
<a href="#">arm_astScript.sml</a>	8295	4 months	acjf3	Update that adds support for ThumbEE. /* From the ARM reference... ThumbEE Is...
<a href="#">arm_astSyntax.sig</a>	8295	4 months	acjf3	Update that adds support for ThumbEE. /* From the ARM reference... ThumbEE Is...
<a href="#">arm_astSyntax.sml</a>	8295	4 months	acjf3	Update that adds support for ThumbEE. /* From the ARM reference... ThumbEE Is...
<a href="#">arm_coretypesScript.sml</a>	8607	4 weeks	acjf3	A few changes: - Make use of the new zDefine - Moved some theorems from arm_ste...
<a href="#">arm_decoderScript.sml</a>	8700	3 days	acjf3	Minor change to the predictability of the CPS instruction, to keep up with the A...

# Monadic specification

---

# Monadic specification

---

- Nice way to handle “Unpredictable” cases (error states).
- Separates specification from underlying semantics. (Can easily change the underlying monad.)
- Makes specification more readable (closer to the source) and easier to maintain.

# Monadic specification

---

- Nice way to handle “Unpredictable” cases (error states).
- Separates specification from underlying semantics. (Can easily change the underlying monad.)
- Makes specification more readable (closer to the source) and easier to maintain.
- Easy to represent in HOL:

$$f(s, instr) = \text{let } s' = \text{write\_reg}(s, r_1, v_1) \text{ in}$$
$$\text{let } v_2 = \text{read\_reg}(s', r_2) \text{ in}$$
$$\text{write\_mem}(s', m, v_2)$$
$$f(instr) = \text{do}$$
$$\text{write\_reg}(r_1, v_1);$$
$$v_2 \leftarrow \text{read\_reg}(r_2);$$
$$\text{write\_mem}(m, v_2)$$
$$\text{od}$$

# ISA specification wish list

---

# ISA specification wish list

---

Here are a few things that I believe are important when specifying ISAs:

# ISA specification wish list

---

Here are a few things that I believe are important when specifying ISAs:

1. Excellent bit-vector support, i.e. a wide range of operations, unconstrained by word size.

# ISA specification wish list

---

Here are a few things that I believe are important when specifying ISAs:

1. Excellent bit-vector support, i.e. a wide range of operations, unconstrained by word size.
2. Arbitrary precision integers and natural numbers. This provides a mathematically sound grounding and is how things are defined in the ARM reference manual. (Real numbers help when specifying floating-point arithmetic.)

# ISA specification wish list

---

Here are a few things that I believe are important when specifying ISAs:

1. Excellent bit-vector support, i.e. a wide range of operations, unconstrained by word size.
2. Arbitrary precision integers and natural numbers. This provides a mathematically sound grounding and is how things are defined in the ARM reference manual. (Real numbers help when specifying floating-point arithmetic.)
3. Also handy are: tuples, records, algebraic data types (pattern matching) and finite maps (for register banks and memory). Lists have been useful too.

# ISA specification wish list

---

Here are a few things that I believe are important when specifying ISAs:

1. Excellent bit-vector support, i.e. a wide range of operations, unconstrained by word size.
2. Arbitrary precision integers and natural numbers. This provides a mathematically sound grounding and is how things are defined in the ARM reference manual. (Real numbers help when specifying floating-point arithmetic.)
3. Also handy are: tuples, records, algebraic data types (pattern matching) and finite maps (for register banks and memory). Lists have been useful too.
4. The specification language should be formal (have an unambiguous semantics) and support evaluation (preferably not just ground expression evaluation).

# ISA specification wish list

---

# ISA specification wish list

---

5. For full scale program evaluation an efficient representation of main memory is required. (Patricia trees are used in HOL.)

# ISA specification wish list

---

5. For full scale program evaluation an efficient representation of main memory is required. (Patricia trees are used in HOL.)
6. Evaluation is import for validation against hardware. This requires modelling instruction decoding, e.g. maps from machine code to an abstract syntax data type.

# ISA specification wish list

---

5. For full scale program evaluation an efficient representation of main memory is required. (Patricia trees are used in HOL.)
6. Evaluation is important for validation against hardware. This requires modelling instruction decoding, e.g. maps from machine code to an abstract syntax data type.
7. Instruction encoding (AST  $\rightarrow$  machine code) and assembly code parsing/printing are also useful (but these needn't be formalized).

# ISA specification wish list

---

5. For full scale program evaluation an efficient representation of main memory is required. (Patricia trees are used in HOL.)
6. Evaluation is important for validation against hardware. This requires modelling instruction decoding, e.g. maps from machine code to an abstract syntax data type.
7. Instruction encoding (AST  $\rightarrow$  machine code) and assembly code parsing/printing are also useful (but these needn't be formalized).
8. Helps having mechanisms to define instructions families and cleanly support coprocessor instructions, multiple instruction encodings and multiple architecture versions. (Try to avoid unnecessary repetition and support "specialisation".)

# ISA specification wish list

---

# ISA specification wish list

---

9. Need a good story for handling “Unpredictable”, “Implementation Dependent” and “Unknown value” cases.

# ISA specification wish list

---

9. Need a good story for handling “Unpredictable”, “Implementation Dependent” and “Unknown value” cases.
10. Need a way to model hardware interrupts.

# ISA specification wish list

---

9. Need a good story for handling “Unpredictable”, “Implementation Dependent” and “Unknown value” cases.
10. Need a way to model hardware interrupts.
11. Should support multi-processor and relaxed memory model configurations.  
(Ability to handle parallelism.)

# ISA specification wish list

---

9. Need a good story for handling “Unpredictable”, “Implementation Dependent” and “Unknown value” cases.
10. Need a way to model hardware interrupts.
11. Should support multi-processor and relaxed memory model configurations.  
(Ability to handle parallelism.)
12. Test suites and regression testing are important.

# ISA specification wish list

---

9. Need a good story for handling “Unpredictable”, “Implementation Dependent” and “Unknown value” cases.
10. Need a way to model hardware interrupts.
11. Should support multi-processor and relaxed memory model configurations.  
(Ability to handle parallelism.)
12. Test suites and regression testing are important.
13. Should hopefully support export to, and integration with, other tools:  
theorem provers, model checkers, analysis and synthesis tools.

# ISA specification wish list

---

9. Need a good story for handling “Unpredictable”, “Implementation Dependent” and “Unknown value” cases.
10. Need a way to model hardware interrupts.
11. Should support multi-processor and relaxed memory model configurations. (Ability to handle parallelism.)
12. Test suites and regression testing are important.
13. Should hopefully support export to, and integration with, other tools: theorem provers, model checkers, analysis and synthesis tools.
14. Should support multiple levels of abstraction: high-level to circuits.

Questions?