1

Overview

- The focus will be on CPU cores
- Arm then and now
- How we think about DV
- DV history
- A side note on complexity
- So we just need to boot an OS right?
- What a real project looks like by the numbers
- Current directions and improvements
- Future directions

  arm

2

Arm history

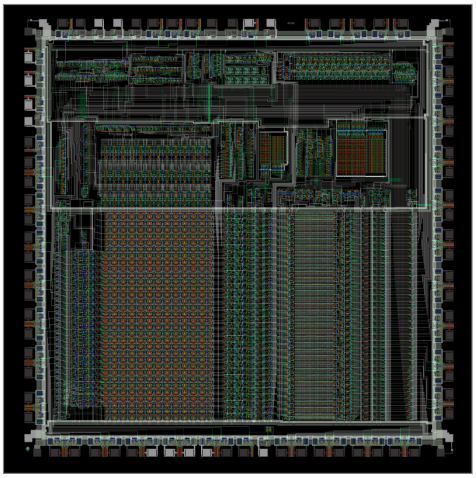
- Founded in November 1990
- With 12 people from Acorn computer…
- And Robin Saxby as CEO
- First office was a barn outside Cambridge, UK

- Internal simulator (asim) + full custom design flow

arm

3

ARM1 Visual Simulator



http://visual6502.org/sim/varm/armgl.html

arm

4

Arm today

- More than 7500 people worldwide many added last year
- 47 main offices worldwide
- Wide range of products from CPUs, GPUs and Video Processors to System IP, Physical IP, IOT devices, IOT infrastructure / data analytics, eSim and various software stacks
- We are part of a very large eco-system
- Industry standard simulators + industry standard ASIC design and implementation flows

arm

5

Some of our partners

arm

6

arm

Design Verification

7

How we think about design verification

- Verification partitioned into different levels
  - System-level – realistic system implementation
  - Kit-level – higher stress sub-system level
  - Top-level – full CPU(s) + Memory system
  - Unit-level – Interesting units in isolation (L2, LS, Core, IF …)
  - Multi-unit – interesting multiple units together (LSL2, MP tb)
- And different techniques
  - Simulation
  - Static methods
  - Emulation/FPGA

arm

8

9

DV history

- It all started with
  - Architectural Validation Suite(s) (AVS) – hand written assembler tests exercising architectural features
  - Device Validation Suite(s) (DVS) – hand written assembler tests exercising micro-architectural and implementation features
- And that was it (for the most part)
- But we realized that top-level testing was insufficient (although it is very much required)

  arm

10

DV history

- So that brought in the era of unit-level testbenches
  - Initially in Vera and ‘e’, but latterly in SystemVerilog
  - Mostly home-grown base class library, but now completely transitioned to UVM
  - Plus functional coverage, portable checkers, assertions, offline checkers, ISS Compare …
- It was also a time of learning what makes a good testbench
  - Trading off testbench performance and maintainability vs. details of checking
  - Portability of checkers and code for unit ➔ multi-unit and unit ➔ top-level

arm

11

DV history

- Emulation and FPGA prototypes required for long single thread execution
  - OS boots and stress testing
  - Custom bare metal stress testing that takes many hours to run
- Rise of formal methods
  - Initially expert users only
  - But designer bring up is rapidly broadening appeal
  - Adding process here allows broader adoption – now a major part of our process

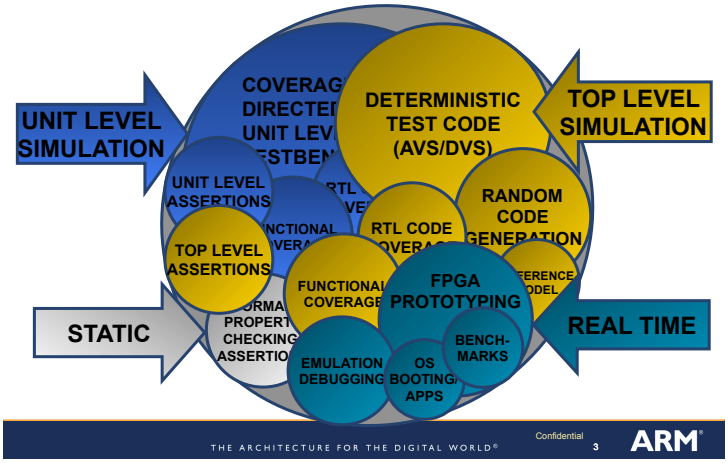
arm

12

DV history

Processor DV Methodologies

- No single methodology covers the entire validation space



arm

13

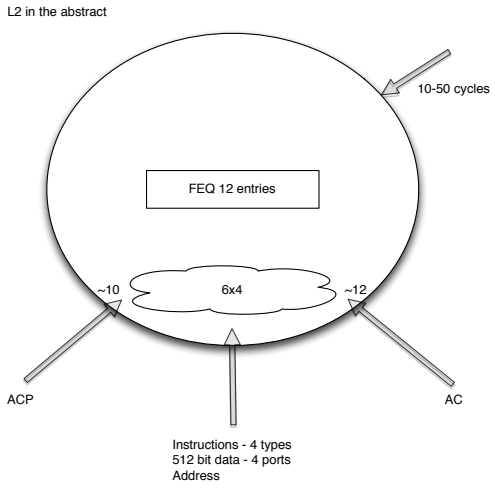
arm

Verification Complexity

14

Diversion on complexity

Cortex-A15 L2



arm

15

By the numbers

- Simplified view of L2:
  - Reachable "state" space – $5x10^{30}$
  - Number of seconds since Big Bang ~$5x10^{17*}$
- Realistic reachable state space is much bigger

* According to http://www.physicsoftheuniverse.com/numbers.html

arm

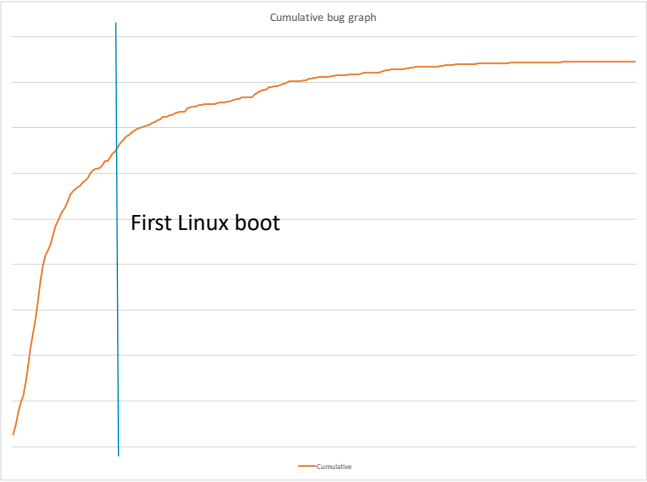
16

arm

OS suitability

17

So booting an OS is stressful right?



arm

18

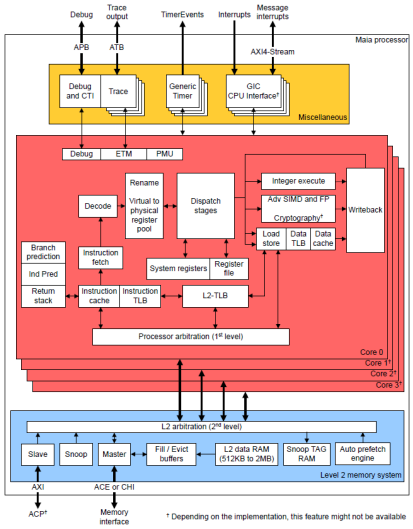
arm

Case study

Neoverse N1

19

Cortex-A72 (Maia) overview

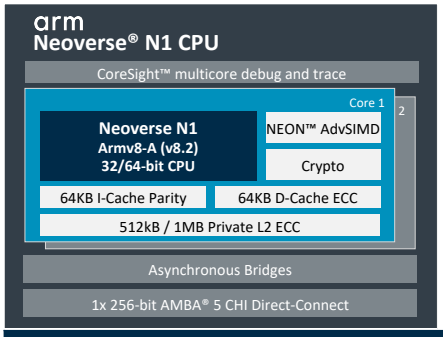


- Full implementation of the base ARMv8 architecture
- AMBA 4 ACE or CHI master interface
- ECC and parity protection for all SRAMs
- Aggressive CPU and L2 power reduction capability
- Support for 4M L2 feature
- Support for ACP port performance improvements

arm

20

Neoverse® N1



Configuration	7nm
Performance	36.4 SPECint2k6 1T @ 2.8GHz 64C system
Freq (Vnom-Vmax)	2.6GHz – 3.1GHz
Power (core)	195 mW/GHz
Ptot (2.6GHz,105C)	650 mW
Area (core, 1MB)	1.47 mm^2

64C System: CMN-600 32xMP2 1MB L2 @ 2.8GHz, 64MB SLC @2GHz, 8xDDR4-3200
Performance: bare-metal SimPts emulation measurement in 64C system
Freq: 16nm Vnom=0.8V, Vmax=1V, 7nm Vnom=0.75V, Vmax=1V
Power: Aarch64 Dhrystone dynamic power, leakage at Tj=105C
Area: post-scaled in mm^2

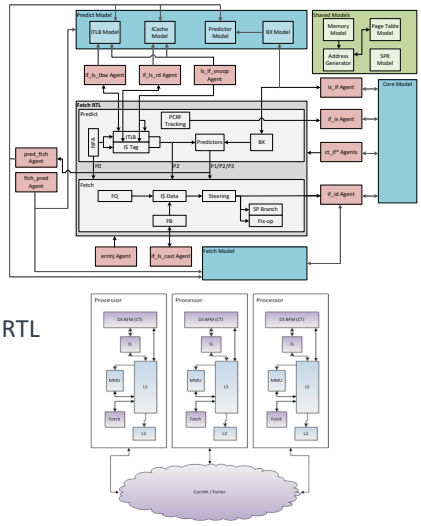
Headline Features

- Full Armv8.2-A A64, A32 and T32 ISA
- Armv8.4-A Dot product support
- AArch32 @ EL0 only, AArch64 EL0-EL3
- TrustZone® technology support
- Full Armv8.2-A MMU support
- Superscalar, variable-length OOO pipeline
- RAS and SPE extension support
- Load acquire (LDAPR) instruction support (v8.3)
- 256-bit CHI-5 direct connect to CMN-600
- Optional strictly inclusive L1/L2 I-Cache

arm

Unit level testbenches

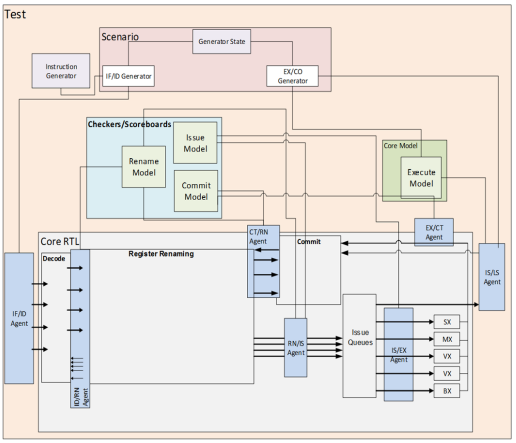
- Instruction Fetch Testbench
  - Clean slate UVM SystemVerilog testbench
  - Cycle agnostic most of the time; cycle accurate where required
  - Internal monitors and checks to detect issues at point of failure
  - Stimuli are randomly generated and controlled by config files
  - Detailed functional coverage model to verify stimulus
- Memory System Testbench
  - Clean slate UVM SystemVerilog testbench
  - Single and multi-cluster setup; individual blocks either model or RTL
  - Majority of stimulus able to run on any config
  - Extensive unit test environment to stop testbench regression
  - Detailed functional coverage and statistical coverage



arm

Unit level testbenches (cont.)

- Core Testbench
  - Clean slate UVM SystemVerilog testbench
  - Covers from pipeline post fetch to completion
  - Decode translates Arm instruction into µops
  - Rename does register renaming
  - Dispatch pushes µops into execution units
  - Issue does µop issue out-of-order to execution units
  - Commit manages reg dealloc, order & exceptions
  - Is primary architectural correctness checking
  - Contains ID, RN, CT, IX & VX
  - Runs with ISSCompare as primary checker
  - Plus local µArch checkers to isolate issues quickly

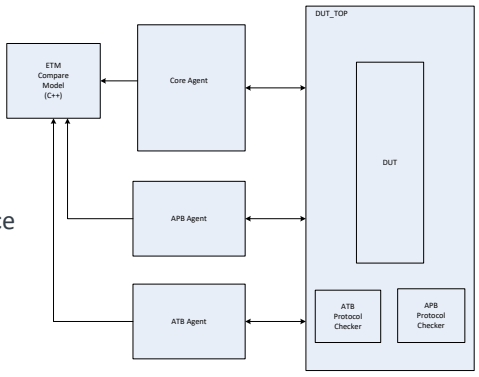


arm

23

Unit level testbenches (cont.)

- Debug & Trace Testbench
  - UVM SystemVerilog testbench with C++ checker
  - Three active agents that drive respective interfaces
  - Each agent has
    - Random stimulus generator
    - BFM
    - Driver attached to DUT
  - C++ checker is called through SystemVerilog DPI interface

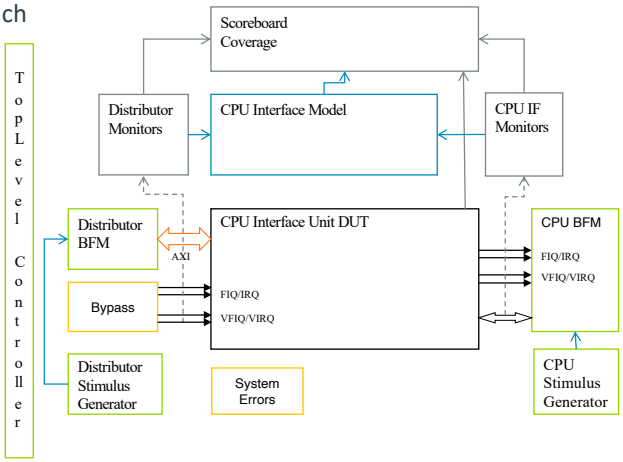


arm

24

Unit level testbenches (cont.)

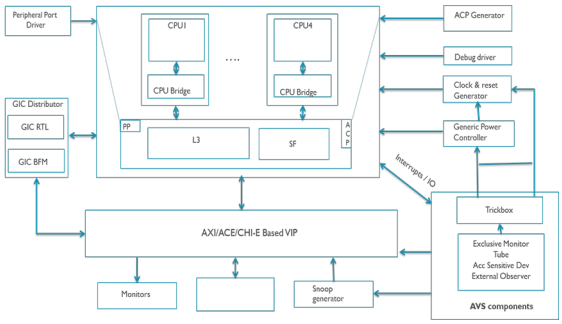
- GIC Testbench
  - Clean slateUVM SystemVerilog testbench
  - Shared Ares UVM infrastructure



arm

Top-level testing

- Vehicle for:
  - Checking architectural compliance
  - Running random instruction sequence generators
  - Power aware simulation with UPF file
  - Broader static configuration space
  - Dynamic configurations
    - Clock ratios, page table attributes, chicken bits
  - Dynamic irritation
    - Random bus traffic, random events (WFI, FIQ, IRQ…)



arm

System-level testing

- Hardware Emulation
  - For OS boots, baremetal testing, Emulator optimized RIS
- FPGA
  - For faster execution, but poorer debug, so lots of long random testing to pull out any potential system issues
  - Memory ordering litmus testing (based on DIY)(very long running) ~2 weeks @ 10MHz

arm

Formal verification

- Formal team worked on all units
  - All units had many embedded properties and several end-to-end checkers
  - Full proof of floating point (VX) instructions using ACL2 theorem prover
  - Sequential equivalence check of abstract C model with implemented RTL
  - Detailed reset abstractions and targeted black boxing to prove complex corner cases around LSU (biggest and most complex unit in the design)
  - First trials of cone of logic analysis and coverage (somewhat inconclusive)

arm

Regressions

Unit level

Stimulus	Cycles Run
Core tb	$448x10^9$ cycles
Fetch tb	$2580x10^9$ cycles
Memsys tb	$890x10^9$ cycles
GIC	$54x10^9$ cycles
DT tb	$170x10^9$ cycles

Top level

Stimulus	Cycles Run /No. Tests
Raven	$1290x10^9$ cycles
Anvil	$410x10^9$ cycles
GenasmMP	$250x10^9$ cycles
AVS	97594 tests
DVS	6653 tests

arm

29

System Validation

Emulation

Stimulus	Cycles
AEGIS	$3x10^{12}$ cycles
PINAKA	$2x10^{12}$ cycles

FPGA

Stimulus	Cycles
HWRIS	$2.2x10^{15}$ cycles

arm

30

Team sizes

- Fairly small team:
  - Around 40 designers working on RTL development
  - Around 50 DV engineers here in Austin
  - Around 10 Top level and System level verification in BLR
- Recent / Current projects in Austin
  - Cortex-A72
  - Cortex-A76
  - Cortex-A77
  - Neoverse N1
  - Cortex-A76AE
  - Zeus + 9 other projects currently

arm

31

arm

Current directions

32

Current directions and improvements

- Completed move to UVM based testbenches
  - Putting TB linting tools in place to help with style and quality of code
- Enhanced system-level verification
- Statistical coverage
- Enhanced formal verification

arm

Completed move to UVM

- New projects have been moving to UVM when they can
  - But it takes a long time to deprecate old testbenches (but we are done now)
- We had a chance to clean slate for some of the current projects
  - So there has been some significant re-writes (not without issues)
  - Some ground up development (using more modern software development ideas (unit test, pairs programming, agile development, etc.))
- Also taking the time to re-evaluate current testbench methodology
  - So we don't lose anything in the switch over
  - So we can build more maintainable testbenches for the future
  - Optimize testbench performance out of the gate, but without optimizing too early
- Starting to look a portable stimulus standard / graph-based stimulus to see if that could aid generation and coverage closure

arm

Enhanced system-level verification

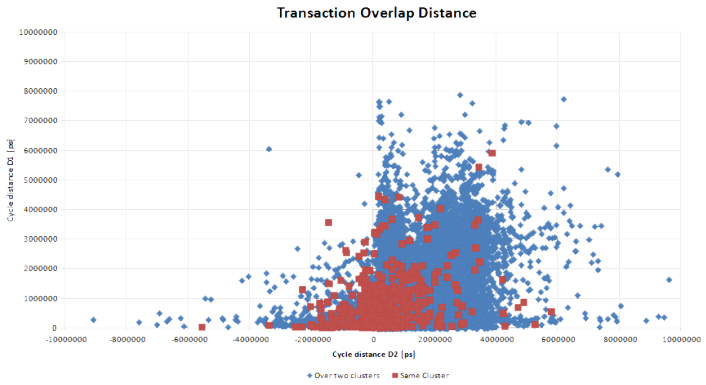
- More configurations of our IP in different system settings
  - Try and find more bugs earlier in the process
- Additional features to aid silicon/FPGA debug
- Always adding stimulus to try and further stress integration testing
  - Tie back to unit-testbenches to further improve stimulus (virtuous cycle)
- Additional bare metal stimulus and checking to enhance hard to hit areas
  - Areas like memory barriers, relaxed memory semantics and SMT

arm

35

Statistical Coverage

- Finding relationships between disparate data
- Correlations may exist beyond single simulation
- Independent of pass/fail status
  - Fairness
  - Arbitration
  - Performance
- Improves stimulus
- Improves checking quality



arm

36

Enhanced formal verification

- Formal property verification
  - Formal testbench development (tracking code, end to end checkers, interface constraints, and embedded assertions)
- Formal coverage collection and fine tuning the coverage model
- Automated deadlock detection flows (this is different from the deadlock detection app)
  - Whatever this flow finds, truly is a deadlock, but it will not necessarily find all possible deadlocks
- Formal X-propagation, and X-checking (RTL tainting)
- Sequential equivalence checking (against RTL, or a C/SystemC model)
- Bug hunting flows as opposed to proofs for all the above
  - Scales well with design size (as opposed to proofs)
  - Is not exhaustive
- Forward progress checking through custom forward progress checks
- Reset lock step analysis for Automotive Enhanced split lock cpus
- Full ACL2 based proof of floating-point hardware (David M. Russinoff; see book - Formal Verification of Floating-Point Hardware Design)

arm

37

arm

Future directions

38

Future directions

- Looking at machine learning techniques for many areas, including
  - Testbench coverage closure
  - Rapid debug analysis
  - Testbench constraint optimization
  - Implementation layout optimization
- All verification results go into a “data lake” for offline analysis
  - Data analysis back end coming online now (EAP) (all Arm based compute)
  - Allows for data mining previous verification projects
- Enabling ISA formal verification
  - Extracting formal ISA properties directly from ARMARM pseudocode
  - Using those properties to bug hunt with formal tools

arm

39

arm

Conclusions

40

Conclusions

- We have a lot of history to deal with (e.g. we have 94 AVS suites – some of which are 20 years old)
- So it can be slower than we like to move to new technologies sometimes
- But there must be a balance of the old and the new
- The goal of any new technique is to bring quality and efficiency to the verification process
- To do that we have to measure everything

arm

41

arm

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
תודה

42