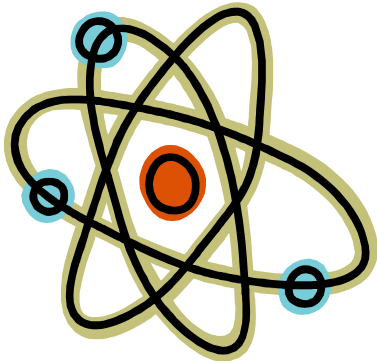


Achilles Test Systems, Inc.



*No one is invincible.
Finding and managing flaws is the secret to survival.*

Four Years of Quantum Simulation



Insights and Lessons Learned
Verifying the QoS Engine of
an Innovative Network Processor

About the Author



- Co-Founder: EDA Software (2006-present)
 - DV Notebook Series, Achilles Test Systems
- Technical Leader: Arch, Micro-Arch, & Verification
 - Packet Scheduler of Quantum Flow Processor, Cisco
- Principal HW Engineer: RTL
 - Packet Scheduler on forwarding ASIC, Hammerhead
- Principal HW Engineer: Architectural modeling & RTL
 - Bandwidth Aware Memory Controller, C-port
- Senior SW Engineer: EDA Software
 - RTL Compilation, ASIC Emulator, Meta Systems
- Senior HW Engineer: RTL & Embedded Software
 - Ethernet and ATM Switching, UB Networks

Lessons Apply to Many SoCs



- Features, complexity, ship date
 - Innovative enough to represent some risk
 - Leap-frog current (next) generation
 - Tested at many layers of abstraction: unit, chip, architectural
 - Diverse IP blocks
- Typical SoC team demographics
 - Multi-site, Multi-discipline, Multi-group
- Challenges
 - How to leverage corporate resources
 - How to track, prioritize, and close issues
 - How to maintain momentum and focus



Take-home Message



- Use more CPUs, not more bodies
 - 100's of CPU years of simulation
 - 1000's of semi-directed test cases
 - 1M's of test executions, stored results
 - Methodical closure of issues and exceptions
- Requires investment in environment
 - Increase productivity by 2 orders of magnitude
 - Find ways to keep the work load manageable
 - This talk presents a few ideas...



Related DV Club Presentations



- **Kersten Eder: University of Bristol**
 - Genetic Programming in Automated Test Code Generation for a Multi-Treaded Microprocessor
 - DV Club: Bristol – Q4 2008
- **Wilson Snyder: SiCortex**
 - Test E.R. Our hope to triage a million tests a day
 - DV Club: Boston – Q4 2007
- **Don Steiss: Cisco**
 - Layered Self-checking Test Generation
 - DV Club: Dallas - Q2 2007
- **Shahram Salamian: Intel**
 - CPU Verification Metrics
 - DV Club: Austin – Q2 2006

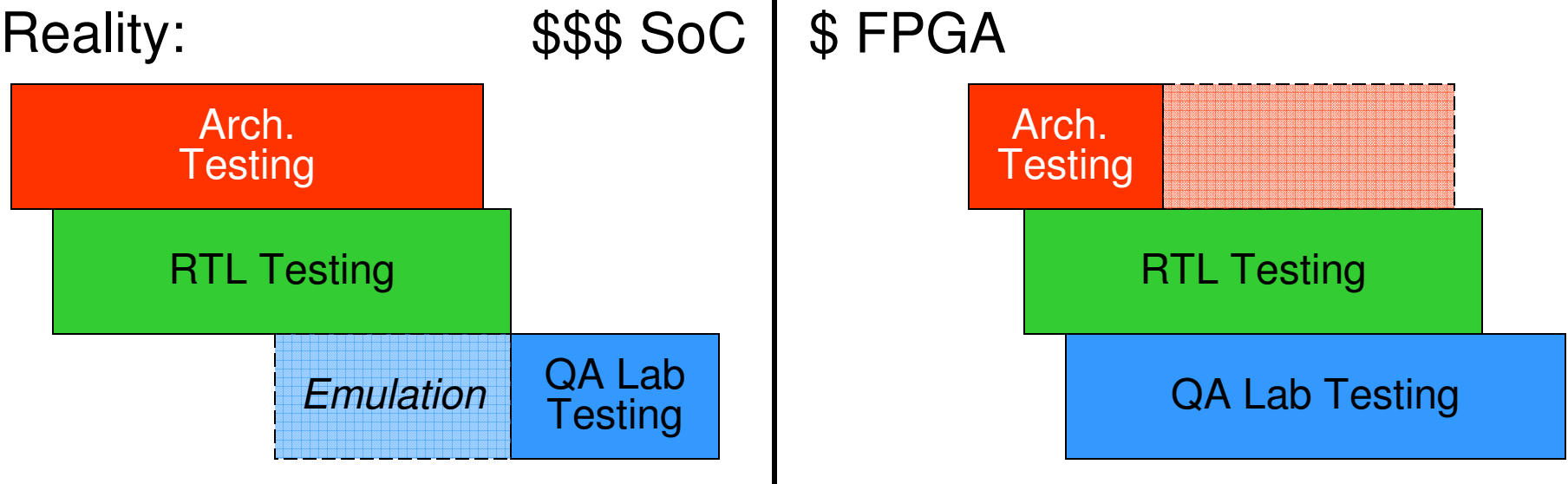


Hardware Testing Schedules



Theory:

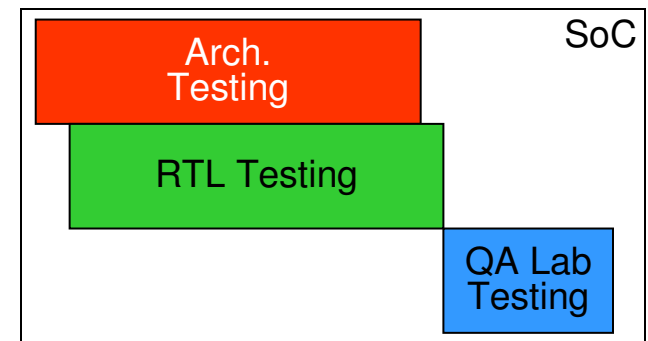
Reality:



Planning for First-Pass Silicon



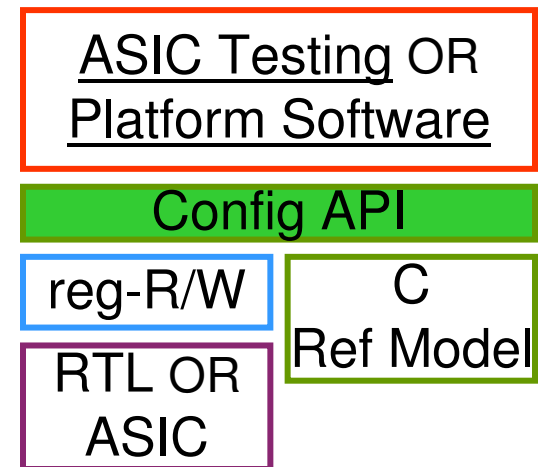
- Assume a typical SoC flow
 - New architecture, next-gen features
 - Mitigate risk, build confidence
- Leverage corporate resources
 - How many CPUs? (5, 50, 500, more)
 - Assume 1-2 hours per test
 - How many test runs prior to tape-out?
 - 10 CPUs, 120 runs/day, 44K runs per year
 - 200 CPUs, 2400 runs/day, 876K runs per year
- Constraints / Obstacles
 - Small team, often new to problem domain
 - No pre-existing tests for next-gen features
 - Millions of cycles needed per test



Getting Help on Test Creation



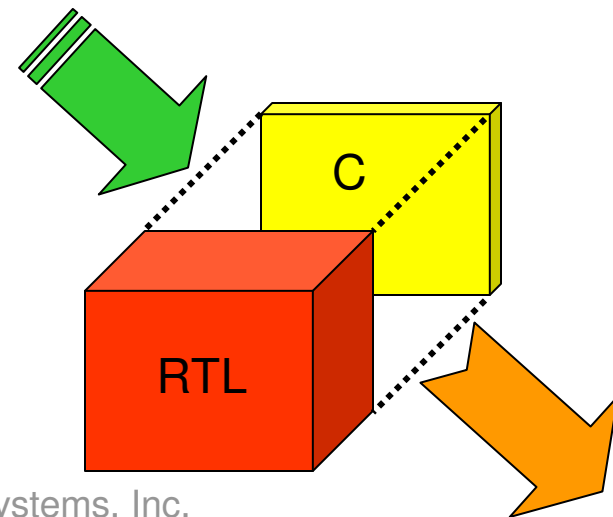
- End-user test descriptions
 - e.g. Simplified Command Line Interface (CLI)
 - Architects, marketing, FAEs, SW, and QA contribute to testing
 - Jump starts testing with 50-100 important configurations
- Have S/W team review testing configurations
 - Architectural simulation pre-dates register definition
 - Create a Config API, higher level than registers
- Lessons learned on past projects...
 - Simple CLI is not good for all tasks
 - Hard to create certain low-level cases
 - Missing some chassis-level details
 - Rigid implementation of test language
 - Lex and Yacc require expert S/W design



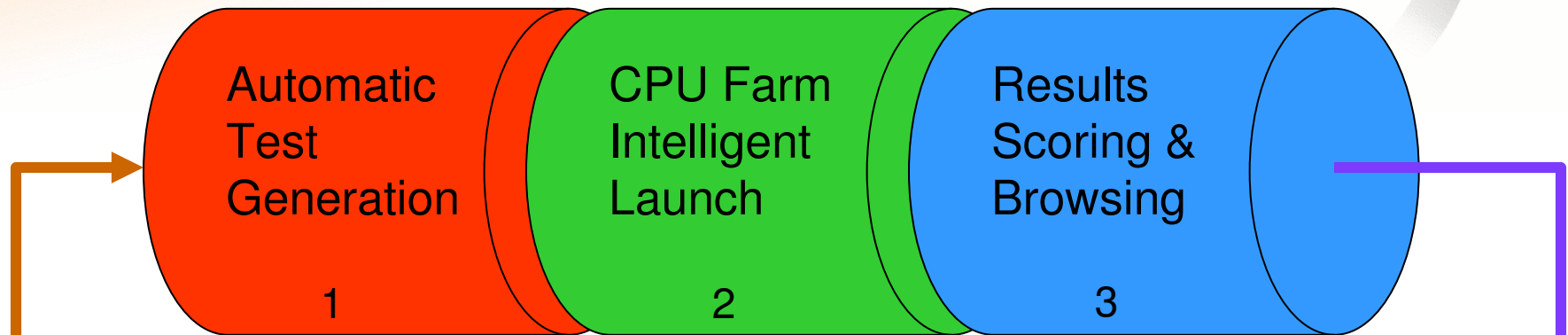
Leveraging a High Speed Model



- Emulation: Ideal
 - Fast and cycle accurate, but expensive
- Otherwise: Fast simulation
 - C model for requirements testing -----> Performance > 1MHz
 - C & RTL cosim for comparison -----> < 1KHz
 - Discrete-event when possible -----> $T=f(\text{pps})$
 - Cycle-accurate when necessary -----> $T=f(\text{Hz})$
- Look for transitive correctness
 - IF (C satisfies requirements)
 - AND (C==RTL)
 - THEN RTL satisfies requirements



Create a Balanced Testing Pipeline



~100 CPUs : ~1000 Tests a day (~ 85% existing, ~ 15% new)

Detect, Debug, Diagnose,
Report, Repair, Rerun

Assume 6-8 tests
debugged per day



Keep team busy
without drowning
them

Need a Game-Changing Technology



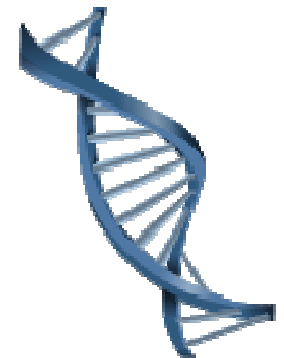
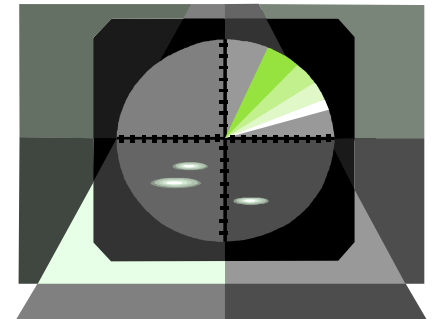
- Consider using an SQL database
 - Store test results in DB
 - Track failures, history, fixes, complement bug DB
 - Store test lists in DB
 - Tune run lists to maximize value
 - Store actual tests in DB
 - Treat a test as a data structure
- But...
 - Hand-coded C \leftrightarrow SQL interface can be rigid
 - Impedes the creation of diverse test types and result formats
 - Need a light-weight way to upload any test description



Automated Test Generation



- Plan the test population
 - A certain number of constrained-random tests
 - e.g. 1,000 - 10,000 directed configs
 - 10 - 100 input scenarios per config
 - Yields 10K-100K unique tests
- Upload directed tests to DB
 - Small programs perform parameter sweeps
 - Heuristic coverage mutations of bin-neighbors
- Lesson learned by analyzing generated tests...
 - Found some unintended common factors
 - Snow-ball effects: too much in-breeding
 - Try to limit the influence of any one test
 - Only possible when functional coverage is persistent



Test Generation w/ Feedback



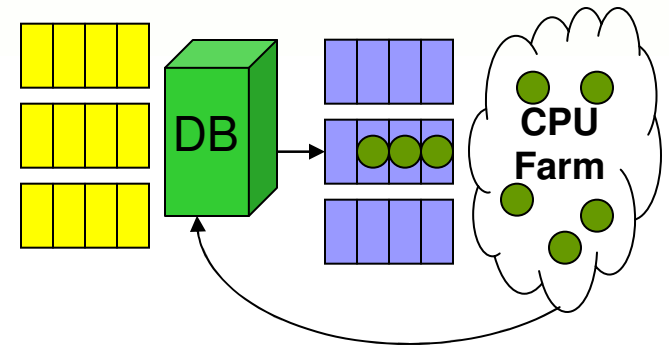
- Testing for numeric error vs. pass/fail
 - Performance (QoS) divergence from a theoretical model
 - Also useful in Analog/Mixed-Signal electrical compliance
- Optimization: Particle Swarms or Genetic Algorithms
 - Apply stimulus
 - Measure error in test result
 - Create modified tests to amplify result
 - Only possible if both test and result are data
- But...
 - Optimizations only amplify problems
 - They do not provide coverage closure
 - Most effective when used with coverage sweeps



Regression Launch Functions



- Linux farm management
 - Simple user-managed queuing
 - Lighten the load on corporate job queuing
 - Custom queuing policies: e.g. time-of-day



- Rerunning <test, seed> pairs
 - Every failing <test, seed> pair is a mini bug report
 - Aggressive re-run of failing <test, seed> pairs
 - Automatically isolate the check-in that causes a new failure
 - n-ary search using the same test and seed

- Interesting challenges
 - Fairness and perception in a linux-farm
 - C-only simulations vs. licensed apps

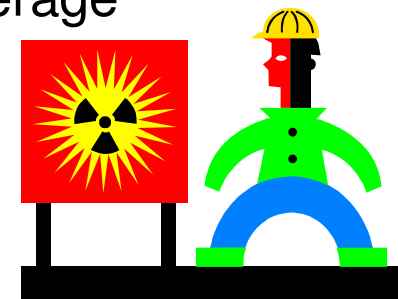
If only I had run a regression!



Coverage Tables and Test Lists



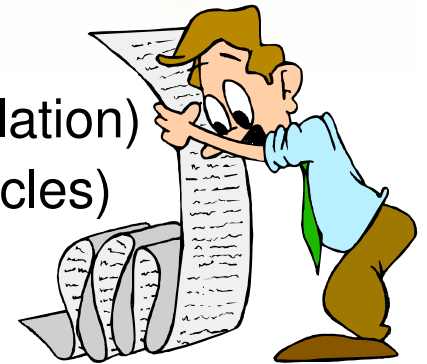
- Functional coverage tables
 - Capture coverage assertions from all model types
 - Assertions in C & RTL, special regs in FPGAs
- When results, coverage, & tests list are data...
 - compute expected time per test list
 - compute expected coverage per test list
 - optimize out tests that don't add coverage
 - swap out one test for a newer tests w/ same coverage
 - select groups tests to run for any reason
 - rotate among test lists w/ similar coverage
- But...
 - Creating test lists should not require SQL knowledge
 - Coverage tables grow large, performance can suffer



Checking and Result Management



- To monitor output from 1000 test a day...
 - Create automated results checkers
 - Flag feature and performance issues (end of simulation)
 - Flag C/RTL mismatch or assertion failure (within 10 cycles)



- But...
 - Limitations of a checker can impose limits on test stimulus
 - Test harness can become too narrowly focused

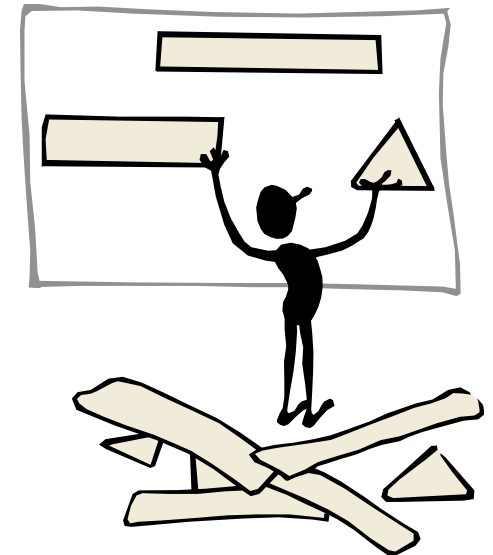
- Lessons learned
 - Need to create diverse drivers and checkers
 - Not every stimulus should pass every checker
 - Emphasis on flexibility, agility



Leverage (the Right) Reusable Pieces



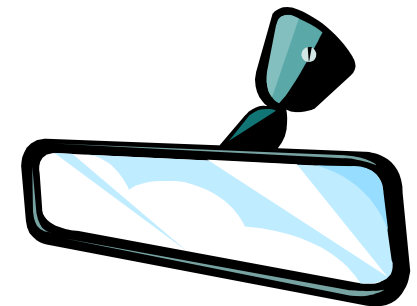
- Open source web packages
 - Sort results by error type and severity
 - Drill down: result, history, coverage, code version
 - Small status pages for each regression
 - Display useful pre-made queries/graphs via web site
- Create web/DB mirrors for each model type
 - Models: C, Cosim, and RTL-only
 - Results and coverage per model type
 - Higher level management views for group status
- Lessons learned...
 - Hand coded CGI can be laborious
 - Other teams could not directly re-use our work
 - Should not require users to know SQL



Lessons Learned



- Drive to SoC tape-out with high confidence
 - Methodical closure of feature issues and exceptions
 - Exceed 10K tests & 100K captured results per engineer
 - Over 1M CPU hours of simulation (125 CPU years)
 - Keep team fed with work and information
 - Small team, Very high productivity
- Would definitely do it again, but...
 - Make it easy enough to be useful on smaller projects
 - 1-5 engineers, 5-10 sims per day
 - Leverage recent Web 2.0 advances
 - Canned database-backed platforms
 - Wikis
 - Social collaboration and tagging



Achilles Test Systems, Inc.



*No one is invincible.
Finding and managing flaws is the secret to survival.*

Thank You

Questions?

chris.kappler@achillestest.com