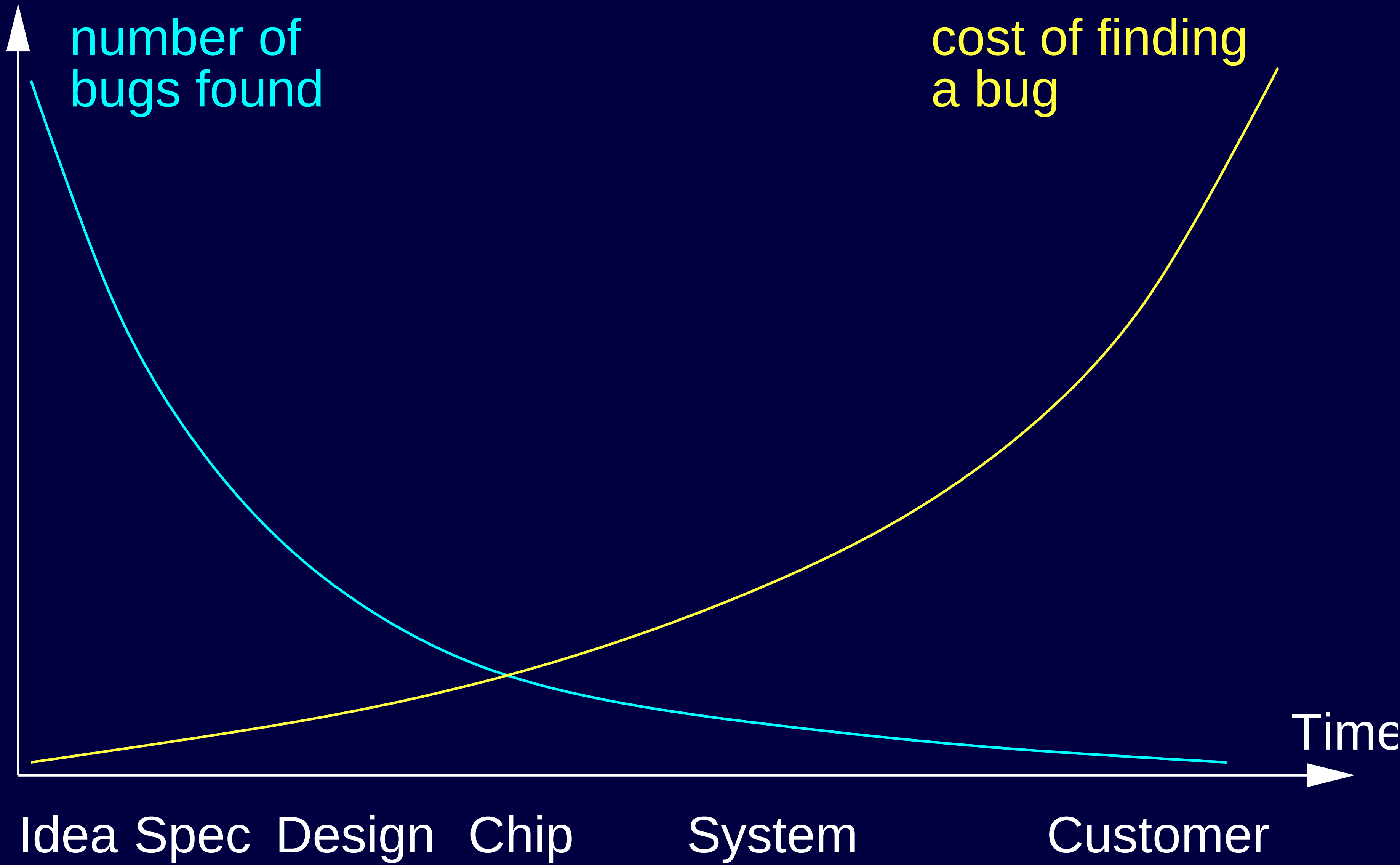
Remember: Verify early!



The later you find a bug, the more it will cost you!

When is Verification DONE?

- **Never truly done on complex designs.**
- Remember: Verification can only show presence of errors, not their absence.
- Skill and experience are needed to:
  - Determine WHAT needs to be verified.
    - Be selective/exhaustive.
    - Identify corner cases.
- Given enough time, errors **will** be uncovered.
- Critical question:
  - **Is the error likely to be severe enough to warrant the effort spent to find/correct it?**

Verification is similar to statistical hypothesis testing.

- Hypothesis ”under test” is: Is the design functionally correct?

Hypothesis Matrix

	Errors	No Errors
Bad Design		Type II (False Positive)
Good Design	Type I (False Negative)	

Type I mistakes: Easy to identify - found error where none exists.

Type II mistakes: Most serious - verification *failed to* identify an error!

- Can result in a bad design being shipped unknowingly!

Knowing where you are in the verification process is much easier to estimate than how long it will take to complete the job.

- It is hard to answer: **How much is enough?**
- ..., but even harder: **When will it be done?**

Verification Tools

Task of Verification Engineer:

- Ensure product is not a Type II mistake (false positive) - as fast and as cost-effective as possible.

(... and as Verification Team Leader):

- Select/Provide appropriate tools for team.
- Decide when cost of finding next bug violates **law of diminishing returns.**

Parallelism, Abstraction and **Automation** can reduce the duration of verification. (Automation is currently the least applicable!)

Automation reduces human factor, improves efficiency and reliability.

Verification TOOLS are used to achieve automation.

- Tool providers: Electronic Design Automation (EDA) industry

Tools to support Functional Design Verification

- Linting Tools
- **Simulators**
- Third Party Models
- **Waveform Viewers**
- Coverage (later in more detail)
- For programming: Verification Languages (Non-RTL!)
- Version Control(!)
- Issue Tracking(!)
- Metrics

These tools are used for traditional simulation-based verification.

Linting Tools

- Assist in finding common programming mistakes
  - For C programs: Try **`man lint`**! (Did you know this existed?)

- Only identify certain class of problems
  - Linting is a **static** checker.

- Many false negatives are reported.
  - Use a filter program to reduce false negatives.
  - Careful - don't filter true negatives though!

- Does assist in enforcing **coding guidelines!**
  - Rules for coding guidelines can be added to linter.

Simulators

Simulate the design (not the implementation) **before** fabrication.

- **Simplifications:** Functional correctness/accuracy is a problem!

Simulation requires **stimulus!** Not just static!

- **Verification Challenge:** "What input patterns to supply to the Design Under Verification (DUV) ..."

Requires to reproduce environment in which design will be used

- **Testbench** (Remember: Verification vs Testing!)

Simulation outputs are validated externally against design intent (specification) Errors cannot be proven not to exist!

- **Verification Challenge:** "... and knowing what is expected for the output for a properly working design."

Two types of simulators: **event-based** and **cycle-based**

Event-based Simulators

- Event-based simulators are driven based on **events.** :)

- Outputs are a function of inputs:
  - The outputs change only when the inputs do.
  - **The event is the input changing.**
  - An event causes the simulator to re-evaluate and calculate new output.
  - Outputs (of one block) may be used as inputs (of another) ...

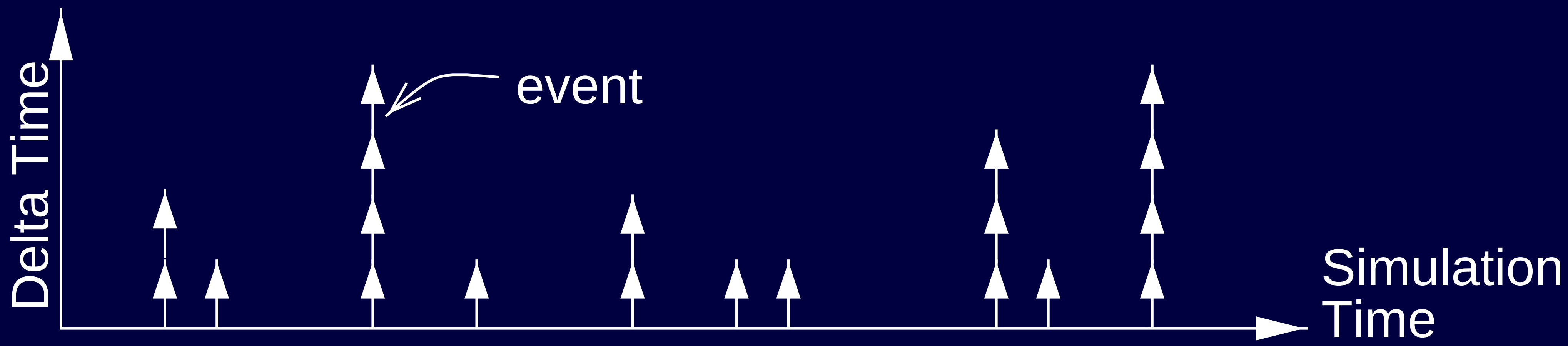
Re-evaluation happens until no more changes propagate through the design.

- Zero delay cycles are called **delta cycles!**

Delta Cycles

Event propagation may cause new values being assigned after **zero** delays. (Remember, this is only a **model** of the physical circuit.)

- Although **simulation time** does **not advance**, the **simulation makes progress.**

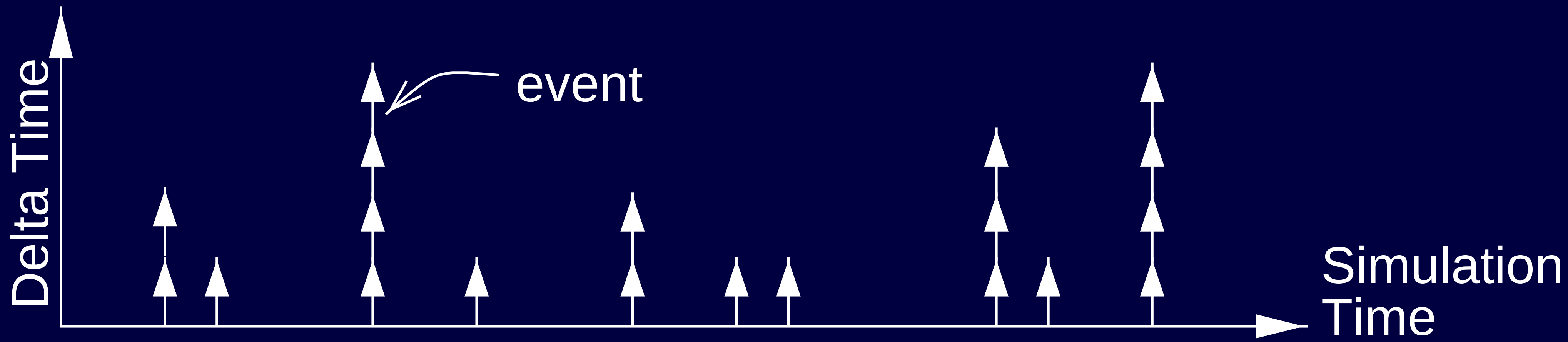


NOTE: Simulation progress is first along the zero/delta-time axis and then along the simulation time axis.

Delta Cycles

Event propagation may cause new values being assigned after **zero** delays. (Remember, this is only a **model** of the physical circuit.)

- Although **simulation time** does **not advance**, the **simulation makes progress.**



NOTE: Simulation progress is first along the zero/delta-time axis and then along the simulation time axis.

- It is possible to write models that **unintentionally get stuck in delta cycles!**

Cycle-based Simulators

- Simulation is based on **clock cycles**, not events.
- Cycle-based simulators **contain no timing information!**
- Can handle **only synchronous circuits:**
  - Only ”event” is *active* edge of clock.
  - All other inputs are aligned with clock.

++ **Much faster than event-based simulation.** (Why?)

–– BUT: Can’t simulate asynchronous boundaries.

Co-Simulation

- Co-simulators are a combination of event, cycle and other simulators (acceleration, emulation)

- **Performance is decreased** due to inter-tool communication overhead.

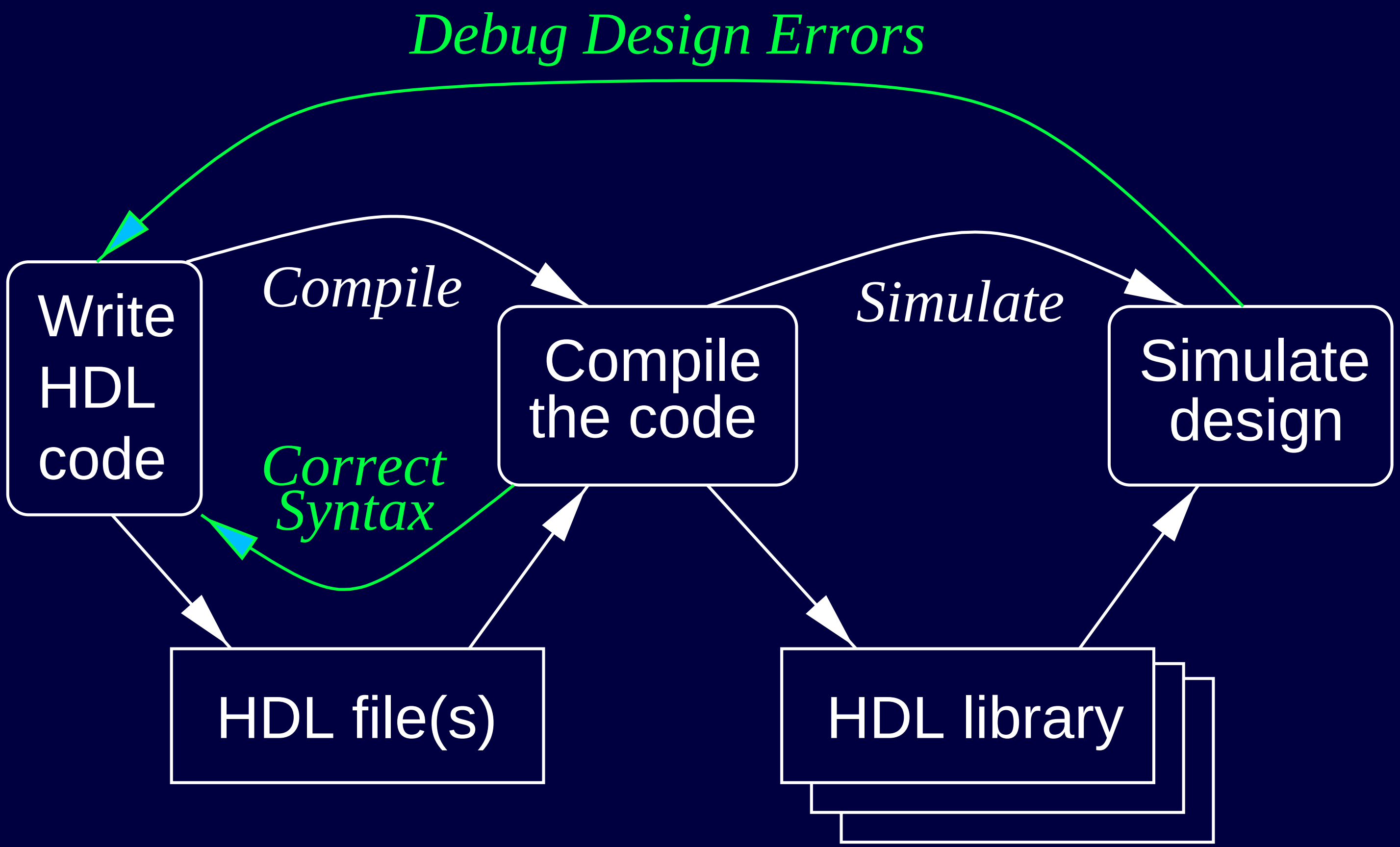
- **Ambiguities arise** during translation from one simulator to the other.
- Verilog's 128 possible states to VHDL's 9!
- Analog current and voltage into digital logic value and strength.

- Few real-world circuits are perfectly synchronous, thus cannot use cycle-based simulators.

Simulation based on COMPILED code

To simulate with **ModelSim:**

- Compile HDL source code into a library.
- Compiled design can be simulated.



Simulation: The bad news

- **Confidence in simulation-approved designs is diminishing!**
- Designs are increasingly complex.
- Too complex to simulate all possible combinations of inputs/states.

■ Verify all (10^{80}) possible state transitions in a 256 bit RAM.[Seger]

build computers from all earth matter ($6 * 10^{24}$ kg)

each computer has size of electron (10^{-30} kg)

one computer simulates 10^{6} cases per second ($3.15 * 10^{13}$ cases per year)

start at time of Big Bang (10^{10} years ago)

$\Rightarrow$ Just about 2% of task completed (now). Hmmm. :(

■ Verify a 100 Million gate SoC design with coverage of 95%.

$\Rightarrow$ **5 Million** gates **not** verified!

Answer: Formal Verification? Design/Verification re-use? [ITRS]

In future: **Correct-by-construction** design methodology?

Third Party Models

Chip needs to be verified in its **target environment.**

- Board/SoC Verification

Do you develop or buy behavioural models (specs) for board parts?

- Buying them may seem expensive!

Ask yourself: **If it was not worth designing on your own to begin with, why is writing your own model now justified?**

- The model you develop is not as reliable as the one you buy.
- The one you buy is used by many others - not just yourself.

Remember: In practice, it is always more expensive to develop your own model to the **same degree of confidence** than licensing it.

Hardware Modellers - often used with emulators/accelerators

- Allow to connect design to a ”real” chip using special interfaces.

Waveform Viewers

Often considered as part of a simulator.

Most common verification tools used...

Used to **visually inspect** design/testbench/verification environment.

- Recording waves decreases performance of simulator.
- Don't use viewer to determine if design passes or fails a test.
- **Use waveform viewer for debugging.** (Assignment 1)

Advanced waveform viewers can **compare** waveforms.

- Problem is how to determine what is the **"golden wave".**
- Most applicable to redesign, where design must maintain cycle by cycle compatibility. (Assumes previous design was correct!)
- **Pitfall when doing comparisons:** Comparison fails when...

Waveform Viewers

Often considered as part of a simulator.

Most common verification tools used...

Used to **visually inspect** design/testbench/verification environment.

- Recording waves decreases performance of simulator.
- Don't use viewer to determine if design passes or fails a test.
- **Use waveform viewer for debugging.** (Assignment 1)

Advanced waveform viewers can **compare** waveforms.

- Problem is how to determine what is the **"golden wave".**
- Most applicable to redesign, where design must maintain cycle by cycle compatibility. (Assumes previous design was correct!)
- **Pitfall when doing comparisons:** Comparison fails when...

Absolute value of signals is not so important - **relative relationship between events** is (often) more important.

Verification Languages

Need to be designed to address **verification** principles.

Deficiencies in RTL languages (HDLs such as Verilog and VHDL):

- **Verilog** was designed with focus on describing low-level hardware structures.
  - No support for **data structures** (■ records, linked lists, etc).
  - Not object-oriented. Useful when several team members develop testbenches.

(• VHDL was designed for large design teams.)

Limitations inhibit **efficient** implementation of verification strategy.

High-level verification languages are (currently):

- **e-language used for Specman Elite** [IEEE P1647]
- Synopsys' Vera, System C

Any other *Verification* Languages?

Tommy Kelly, CEO of Verilab:

"Above all else, the Ideal Verification Engineer will know how to construct software."

Toolkit contains not only **Verilog,** VHDL, Vera and e, but also: Python, Lisp, mySQL, Java, ...

Version Control

Concepts for version control:

- Files must be centrally managed.
- It must be easy to get the files. (Simplify access.)
  - Files are owned by team - not by one individual!

(• Advanced: Automatic notification of changes via e-mail.)

- History is kept for each file.

HDL design/verification is similar to managing a software project!

Software Engineers can bring valuable skills to design/verification.

■ Examples: Subversion (see http://subversion.tigris.org/) or CVS.

Why is version control considered a verification tool?

Version Control

Concepts for version control:

- Files must be centrally managed.
- It must be easy to get the files. (Simplify access.)
  – Files are owned by team - not by one individual!
- (• Advanced: Automatic notification of changes via e-mail.)
- History is kept for each file.

HDL design/verification is similar to managing a software project!

Software Engineers can bring valuable skills to design/verification.

■ Examples: Subversion (see http://subversion.tigris.org/) or CVS.

Why is version control considered a verification tool?

To ensure that what is being verified is actually what is being implemented. Wouldn't you hate to spend a month verifying something that is out of date...?

Issue Tracking

- Another tool often not considered a verification tool.
- Verification engineer's job is to **find bugs.** :)
- Issue tracking is used to deal with the bugs that are found - bugs must be fixed!

Two things to consider:

- **What is an issue?**
- **How to track it?**

What is an issue?

The cost of tracking an issue should not be greater than the cost of the issue itself!

An **issue** is anything that can effect the functionality of the design!

- Bugs found during execution of testbench.
- Ambiguities or incompleteness of a specification.
- Architectural decisions/trade-offs.
- Errors found at all stages of the design (in the design or the verification environment).
- New but relevant stimuli (corner cases) that are not part of the **verification plan.**

Basic principle: **When in doubt - track it!**

Some bugs might not be worth fixing in the current product, but we want to capture the issue so that they are fixed in next revision of chip.

How to track an issue?

Different methods - more or less successful...

- Grapevine system
- Post-It system
- Procedural system
- **Computerised system**
  - Example: track (see http://trac.edgewall.org/)

Computerised Issue Tracking

Use a computer system (usually database) to track issues!

- Issues are seen through to resolution.
- Issues can be assigned and/or reassigned to individuals or small teams.
- Automatic e-mail notification:
  - of new issues;
  - of status of top issues, etc.

++ Contains a history.

++ Provides **lessons-learned database** for others.

Some computerised issue tracking systems fail - why?

Computerised Issue Tracking

Use a computer system (usually database) to track issues!

- Issues are seen through to resolution.
- Issues can be assigned and/or reassigned to individuals or small teams.
- Automatic e-mail notification:
  - of new issues;
  - of status of top issues, etc.

++ **Contains a history.**

++ Provides **lessons-learned database** for others.

Some computerised issue tracking systems fail - why?

- Biggest problem: **Ease of use!!!**
- **It should not take longer to submit an issue than to fix it!**

Metrics

Not really a verification tool - but managers love metrics and measurements!

- Managers often have little time to personally assess progress.
- They want something measurable.

Coverage is one metric - will be introduced later.

Others metrics include:

- Number of lines of code;
- Ratio of lines of code (between design and verifier);
- Drop of source code changes;
- Number of outstanding issues.

Summary: Now we have (Ch2 of WTB)

Investigated **verification tools**

- **Linting tools** are static code checkers.
- **Simulators** exercise design to find functional errors.
- **Waveform viewers** display simulation results graphically.
- **Version control** and **issue tracking** help manage design data.
- **Metrics** help management to understand/monitor progress of design project and to measure productivity gains.

TODO: **Coverage** - e.g. code coverage or functional coverage.

Next:

- Intro to Calculator Calc1 design for Assignment A1.
- **Verilog coding styles** (structural, dataflow, **behavioural**).