Introduction to
Design Verification COMS31700

Kerstin Eder

Design Automation and Verification

University of Bristol — Department of Computer Science

Welcome to COMS31700

- Lecturer
  - Kerstin EDER
  - Department of Computer Science
  - Room 3.25 MVB
  - Kerstin.Eder@bristol.ac.uk
  - Office hours:
    - Mo morning and Tu morning
- Lecture notes, exercises and assignments are/will be online at: http://www.cs.bris.ac.uk/Teaching/Resources/COMS31700/
- Comments and feedback are always welcome

COMS31700 Unit Details

- Lectures (weeks 1 – 10)
  - Monday 11:10 – 13:00 in MVB 1.11a
  - Tuesday 11:10 in QB 1.69
  - Friday 12:10 in MVB 1.11a
- Laboratory (weeks 1 – 10)
  - Need to sign Modelsim Agreement!
  - 2h of lab per week
  - DV Help Desk on demand
  - Teaching Assistant: Dr Jimson Mathew
- Assessment — Deadlines on COMS31700 web page!
  - 2 assignments (25% due in week 4/5, 25% due in week 9/10)
  - 1 exam (50% in January 2012)

Literature and Study Resources

- **Janick Bergeron** [WTB]
  Writing Testbenches: Functional Verification of HDL Models.
  First Edition, Kluwer, 2000, ISBN: 0-7923-7766-4
  Second Edition, Kluwer, 2003, ISBN: 1-4020-7401-8
- In addition:
  - Lecture slides and on-line tutorials on COMS31700 web page
  - On-line documentation of ModelSim Simulator and SpecMan Elite
  - Watch the unit web page for further supplementary literature.

[**Credits:** Parts of the lecture notes contain material from the book “Comprehensive Functional Verification” by Bruce Wile etal, the book ”Writing Testbenches: Functional Verification of HDL Models” by Janick Bergeron, the book ”The Verilog Hardware Description Language” by Donald Thomas and from lecture slides developed at IBM (by Avi Ziv and Jaron Wolfstal), the University of Pittsburgh, Penn State University, North Carolina State University and Ohio State University. The HDL for the assignments has been developed at IBM.]

What is this unit about?

Aim:
To familiarise you with the routine tasks in **verification management**, and to give you the **technical background** plus some of the **practical skills** expected from a Design Verification Engineer.

- Pre-/Co-requisites: COMS21101 or some programming experience

On successful completion of this unit, you will be able to:
- understand the complexities and limits of verification;
- carry out functional verification and determine its effectiveness;
- set appropriate verification goals, select suitable verification methods and assess the associated risks;
- compile a verification plan that fits into the flow of a design project;
- organise a verification team.

Why is Verification so important?

- **Verification is the single biggest lever to effect the triple**
- **constraints: Timing/Schedule, Cost and Quality**
  - Fewer revs through the fabrication process means lower costs and faster time-to-market, ideally right-first-time designs
  - Re-spinning a chip costs:
    - hundreds of thousands of USD/GBP **and** 6-8 weeks at least

Companies who get it right in fewer ”phases”, will win/survive!
- Getting it right (first time) is more and more difficult with rapidly increasing design complexity!
- International Technology Roadmap for Semiconductors identified Verification as a Crisis Area (http://public.itrs.net/)

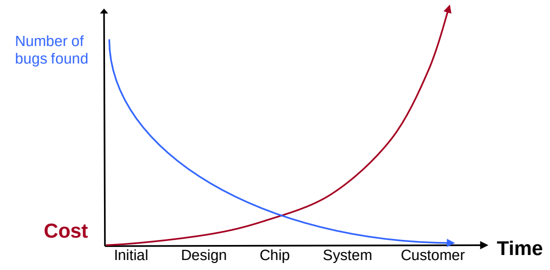
Shortfall in Qualifications, Talent and Skills:
- Verification has a separate career path from logic and circuit design.
- Industry needs more software-trained engineers to do verification!

Cost of Bugs over Time

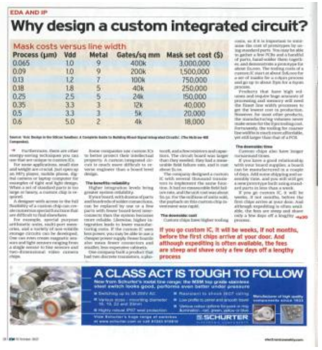
The longer a bug goes undetected over time, the more expensive it is!

- Bug found early (during design simulation) has little cost.
- Finding a bug at chip/system level has moderate cost:
  - Requires more debug time and isolation time.
  - Could require new algorithm, which could affect schedule and cause board rework.
- Finding a bug in System Test (test floor) requires re-spin of a chip.
- Finding a bug in customer's environment can cost hundreds of millions and worst of all – **REPUTATION.**
  - **In the case of safety critical applications potentially even lives!**

Cost of Bugs Over Time



Mask Costs (Electronics Weekly, 10 October 2007)



EDA AND IP

Why design a custom integrated circuit?

Mask costs versus line width

Process (µm)	Vdd	Metal	Gates/sq mm	Mask set cost ($)
0.065	1.0	9	400k	3,000,000
0.09	1.0	9	200k	1,500,000
0.13	1.2	7	100k	750,000
0.18	1.8	5	40k	250,000
0.25	2.5	5	24k	150,000
0.35	3.3	3	12k	40,000
0.5	3.3	3	5k	20,000
0.6	5.0	2	4k	18,000

[illegible]

If you go custom IC, it will be weeks, if not months, before the first chips arrive at your door. And although expediting is often available, the fees are steep and shave only a few days off a lengthy process

A CLASS ACT IS TOUGH TO FOLLOW

SCHURTER

Unit Outline

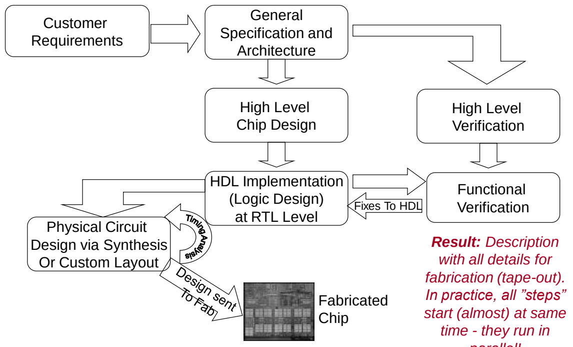
Lecture Topics

- Introduction: What is Verification? What is a Testbench?
- Verification Flow and Tools including basic Verilog HDL coding
- Verification cycle, methodology and plan including coverage
- Simulation-based Verification: Stimulus Generation and Checking
- Assertion-based Verification (ABV)
- Advanced Testbench Design Methodology with SpecMan Elite
- (Functional Formal Verification and Property Checking)

Labs Need to sign ModelSim agreement!

- Exercise 1: Evita Verilog interactive tutorial (do @ home asap)
- Exercise 2: Introduction to the ModelSim Simulator
- A1, weeks 2-5: Verification of calculator design with ModelSim
- Exercise 3: How to collect Code Coverage with ModelSim
- Exercise 4: Introduction to SpecMan Elite
- A2, weeks 6-10: Advanced testbench design with SpecMan Elite

Chip Design Process



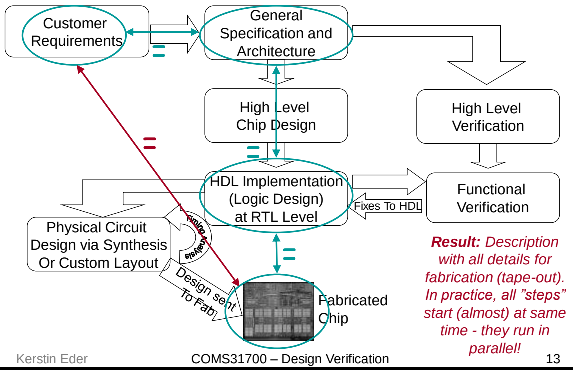
***Result:** Description with all details for fabrication (tape-out). In practice, all "steps" start (almost) at same time - they run in parallel!*

Role of Verification in IC Design

IC design process is complex:

- **Engineers need to balance conflict of interest:**
  - Tight time-to-market constraints vs. increasing design complexity
- **Aim:** "Right-first-time" design, "correct-by-construction"
- More and more time-consuming to obtain acceptable level of confidence in correctness of design!
- **design time << verification time**
  - Remember: Verification does not create value!
    - But it preserves revenue and reputation!
  - Roughly 70% of design effort goes into verification.
  - 80% of all written code is in the verification environment
  - Properly staffed design teams have dedicated verification engineers.
  - In some cases verification engineers outnumber designers 2:1.

Chip Design Process



Result: *Description with all details for fabrication (tape-out). In practice, all "steps" start (almost) at same time - they run in parallel!*

How do Designers know whether a circuit is correct?

Concept
VERIFY: What you specified is what you envisioned
Specification
VERIFY: What you designed is what you specified
HDL Design (RTL)
VERIFY: What you taped out is what the RTL describes
Tape-out
TEST: What was manufactured is what you taped out
Silicon

There is skill, science and methodology behind verification.

What is Verification?

*"Verification is a **process** used to demonstrate the correctness of a design w.r.t. the requirements and specification."*

Types of verification:

- **Functional verification** (Covered in COMS31700)
- Timing verification
- Performance verification
- (what else?)

Functional Verification Techniques

Formal verification

- a.k.a. static verification
- "Mathmatically" prove the correctness of the implementation

Simulation-based methods

- a.k.a. dynamic verification
- Find bugs by executing the implementation and checking its behaviour
  - Use a **Testbench** to execute the implementation!
    - (Focus of skills development in COMS31700.)

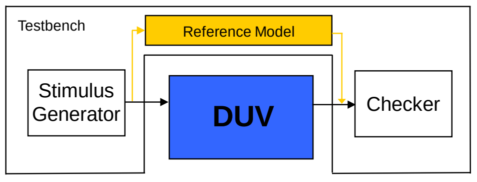
What is a Testbench?

*"**Code** used to create a predetermined **input sequence** to a design, and to then observe the **response**."*

- Generic term used differently across the industry.
- Always refers to a test case/scenario.
- Traditionally, a testbench refers to code **written in a Hardware Description Language (VHDL, Verilog) at the top level of the design hierarchy**.
  - Most HDLs are for design/simulation - not for verification!
- Testbenches are often simple, but may have some elements of randomness. (WHY?)

- **Completely closed system:**
  - No inputs or outputs.
  - Effectively a model of the universe as far as the design is concerned.

Generic Structure: Testbench



Verification Challenges:

- What input patterns to supply to the Design Under Verification (DUV)
- and what is **expected** for the output for a **properly working design**?
- When is verification done?

Increasing Verification Productivity

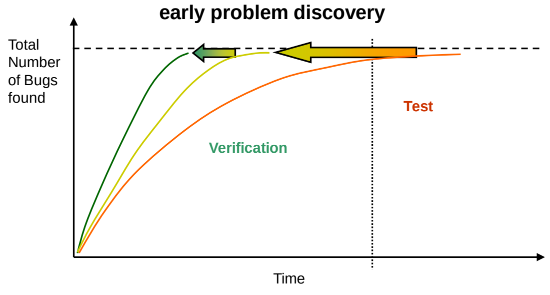
Need to minimise verification time e.g. by using:

- **Parallelism:** Add more resources
- **Abstraction:**
  - Higher level of abstraction (i.e. C vs Assembly)
  - This often means a reduction of control!
- **Automation:**
  - Tools to automate standard processes
  - Requires standard processes/methodology.
  - Usually a variety of functions, interfaces, protocols, and transformations must be verified.
  - Not all (verification) processes can be automated.

Productivity improvements drive early problem discovery!

Increasing Verification Productivity

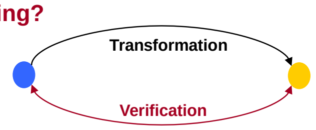
Productivity improvements drive early problem discovery



Reconvergence Model

Conceptual representation of the verification process

- Most important question: **What are you verifying?**

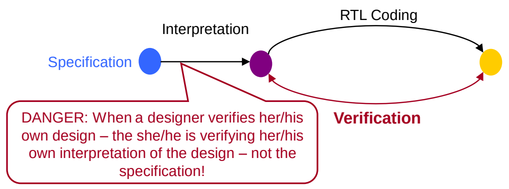


- Purpose of verification is to ensure that the result of some transformation is as intended or as expected.

The Human Factor in Verification

An individual (or group of individuals) must **read and interpret** the **specification** and transform it into the **design**.

- In practice, the **specification** is often a document written in a natural language by individuals of varying degrees of ability to communicate.
- Human introduced errors are introduced by **(mis)interpretation**.



DANGER: When a designer verifies her/his own design – the she/he is verifying her/his own interpretation of the design – not the specification!

Reducing human-introduced Errors

Automation: Take the human out!

- In practice often not feasible:
- Process not well enough defined (not formal enough).
- Transformation requires ingenuity and creativity.

Make human intervention “mistake-proof” (fool-proof):

- Similar to USB stick
- Same pitfalls as automation: *Verification is an art!*

Redundancy: Duplicate every transformation

- Have two individuals (or groups) check each other’s work
- Two completely separate transformations are performed with each outcome compared to verify that both produced the same or equivalent results.
- Most costly, but still cheaper than redesign and product recall.
- DANGER: Designer should NOT be in charge of verification!

What is being verified?

- Choosing a common origin and reconvergence points determines
  - what is being verified and
  - **what type of method** is best to use.
- Investigate the reconvergence models for:
  - Formal Verification
  - Simulation-based Verification

Formal Verification

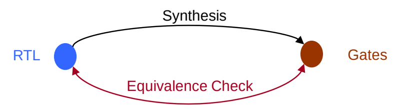
3 types of Formal Verification:
- Equivalence Checking
- Model Checking
- (Theorem Proving - not covered here)

The power of Formal Verification is often misunderstood!
- Engineers unfamiliar with the formal verification process often believe that it is a tool that mathematically determines the correctness of their design, without having to write testbenches.
  - This is not so!

[Nice intro to Formal Verification by Carl-Johan Seger: "An Introduction to Formal Hardware Verification"]

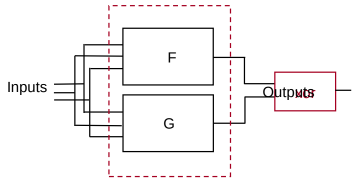
Formal: Equivalence Checking

Compares two models to check for equivalence.
- Proves mathematically that both are *logically equivalent*.
  - Commonly used on **lower levels** of design process.
- Example: RTL to Gates (Post Synthesis)



Why do equivalence checking when EDA tools exist for synthesis?
- See "*HDL Chip Design - A Practical Guide for Designing, Synthesising, and Simulating ASICs and FPGAs using VHDL or Verilog*" book by Douglas Smith page 136 and compare MUX spec with what they claim will be synthesised!

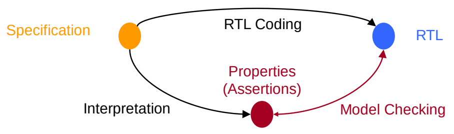
Equivalence Checking



- Is there an input vector such that the output of the XOR gate can be "1"?

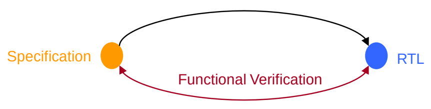
Formal: Model Checking

Properties of a design are formally proven or disproved.
- Used to check for generic problems or violations of user-defined properties of the behaviour of the design.
- Usually employed at **higher levels** of design process.
- **Properties** are derived from specification.
- **Properties** are expressed in some (temporal) logic.
- Checking often involves a (finite) state machine model.
  - This model needs to be derived from the RTL. WHY?



Simulation-based Verification

Functional Verification verifies the design *i n t e n t*.



- NOTE: Unless a specification is written with precise semantics, it is impossible to demonstrate a design meets the intent of its specification.
- Simulation can (only) demonstrate the presence of bugs, but (in practice) cannot demonstrate their absence!

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 (10^{17} kg)
    - each computer has size of electron (10^{-30} kg)
    - one computer simulates 10^{12} cases per second
    - start at time of Big Bang (10^{10} years ago)
  - Just about 0.05% of task completed (now). Hmmm. :(

  - Verify a 100 Million gate SoC design with coverage of 95%.
  - **5 Million** gates **not** verified!

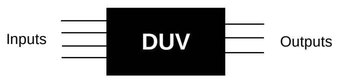
- **Answer: Formal Verification? Design/Verification re-use?** [ITRS]
- In future: **Correct-by-construction** design methodology?

Observability during Verification

There are several levels of observability:

- Black box verification
- White box verification
- Grey box verification

Black Box Verification

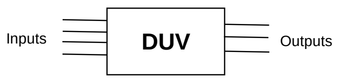


- The black box has inputs, outputs, and performs some (well documented) function.
- To verify a black box, you need to **understand the function** and be able to predict the outputs based on the inputs.
- The verification code utilizes only the external interfaces as defined by the specification. The internal signals and constructs remain in the dark.
- **Pros:**
  - No knowledge of the actual implementation is required.
  - Ability to predict functional results based on inputs alone ensures that the reference model remains independent from the DUV implementation.
  - Verification code is less sensitive to changes inside the DUV.
- **Cons:**
  - Difficult to locate source of problem, only exposes effects.(if at all – not all bugs propagate to the outputs).
  - Lacks controllability and observability.

Note on: Perfect Verification

- To **fully verify a black box**, you must show that the logic works correctly **for all combinations of inputs**.
- This means:
  - Driving **all permutations** on the input lines.
  - Checking for proper results in **all cases**.
- OK for verifying a two input "*and*" gate, BUT:
- **Exhaustive verification is not practical on large designs**.
  - NOTE: Formal verification is (by definition) exhaustive (modulo abstractions) but has limited scope.

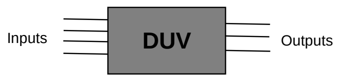
White Box Verification



(Opposite of black-box approach.)

- For white box verification the internal facilities of the DUV are known, visible and utilised for verification.
- **Pros:**
  - Full visibility and controllability of internal signals.
    - Can detect bugs as soon as they occur.
  - Quickly possible to set up interesting conditions, e.g. counter roll-over.
- **Cons:**
  - Danger to follow the implementation/design instead of the specification.
  - Sensitive to changes in the DUV (implementation).
  - Too many details make it hard to create and maintain.

Grey Box Verification



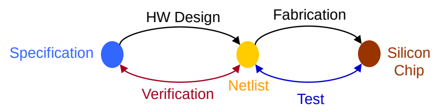
- For grey box verification a limited number of DUV facilities are utilised in a mostly black-box environment.
  - Access important and stable features, the rest is kept in the dark.
- Combines the pros (if done the right way) or the cons (if done the wrong way) of black and white box.
- In practice, most verification environments are grey box!
  - Often start with black box and evolve into grey box.
  - Note: Prediction of correct results on the interface is occasionally impossible without viewing an internal signal.

Notes on Controllability

- In theory, the same levels as for observability also exist for controllability
- In practice, we seldom control the internals of the DUV
  - May lead to inconsistent state
- The main exception – **warm loading**
  - Brings the DUV to a predefined initial state
    - E.g. almost full buffer
  - Reduces the time needed for reaching this state

Verification vs. Test

- **Often confused!**
  - Purpose of test is to show design was **manufactured** properly.
  - Verification is done to ensure that **design meets its functional intent prior to manufacture!**



- One test method is scanning: **"Design for Test"** methodology
  - Link all internal registers together into a chain.
  - Chain accessible from chip pins. ! Control/observe internal state.
  - Restricts design, but keeps cost (due to fab problems) down.
- Why not **"Design for Verification"?** [Hot topic of research!]
- Consider: What is design supposed to do? How will it be verified?

Cost of Verification

Necessary Evil

- Always takes too long and costs too much.
- As number of bugs found decreases, cost and time of finding remaining ones increases.

So when is verification done? **(Will investigate this later!)**

- Remember: Verification does not generate revenue!

Yet indispensable

- To create revenue, design must be functionally correct and provide benefits to customer.
- Proper functional verification demonstrates **trustworthiness** of the design.
- Right-first-time designs demonstrate **professionalism** and "increase" reputation of design team.

Verification is similar to statistical hypothesis testing

Hypothesis "under test" is: Is the design functionally correct?

	Bugs found	No Bugs found
Bad Design (buggy design)		Type II: False Positive
Good Design (no bugs in 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.

Summary

Now we have (Ch1 of WTB)

- Seen what a **testbench** is.
- Understood *how* **important verification** is and why it is so important.
- Seen that parallelism, automation and abstraction are strategies to reduce the time necessary to perform verification.
- Used **reconvergence models** to help us identify what is verified.
- Been introduced to various levels of observation.
- Briefly discussed the **cost of verification.**

- **Next:** Verification flow and tools, especially simulators, waveform viewers and simulation languages (Verilog).
- **Lab:** Exercise 2: Debug simple MUX design. **Then start A1!**
- **Web pages for COMS31700 will be updated this week.**