AMS Verification
Agenda

- ASIC world
- ASIC Industrial Facts
- Why Verification?
- Verification Overview
- Functional Verification
- Formal Verification
- Analog Verification
- Mixed-Signal Verification
- DFT Verification
- Verification Metric
“Application Specific Integrated Circuits (ASICs) are, as the name indicates, non-standard integrated circuits that have been designed for a specific use or application.”

- **Start**
- **Specification**
- **Design**
- **Verification**
- **Layout**
- **Validation**
- **Finish**
“Application Specific Integrated Circuits (ASICs) are, as the name indicates, non-standard integrated circuits that have been designed for a specific use or application.”

✓ Several trends are impacting ASIC design:
  ✓ SoC integration drives more features per chip
    ✓ Chip size is getting larger and designs are getting more complex.
  ✓ Analog functionality is moving into the digital domain and digital functionality is moving into software
  ✓ Software bring-up now takes more time than hardware bring-up. Traditional AMS simulation techniques do not scale to large ICs.
ASIC Industrial Facts

✓ Roughly, **60% – 80%** of effort in ASIC development time is dedicated to **verification**

✓ Only **20%-40%** of efforts is reserved for the **design**

✓ Main industrial goal is **product time-to-market**
  ✓ Being first in the market equals to more money
  ✓ “**Bugs**” prolong product time-to-market
  ✓ Delays ruining reputation

✓ **Developments costs**
  ✓ Goal is spent more time in development to avoid mistakes
  ✓ “**Bugs**” excaudate costs
“**Verification** is an logical answer to design **bugs** existence, and theirs consequences.”

- **Cost of bugs over time** - Longer a bug goes undetected, the more expensive it is
  - Bug found early (designer sim) has little cost
  - Finding a bug at chip/system has moderate cost
    - Requires more debug time and isolation time
    - Could require new algorithm, which could effect schedule and cause board rework
  - Finding a bug in System Test (test floor) requires new ‘spin’ of a chip
  - Finding bug in customer’s environment can cost hundreds of millions and worst of all - **Reputation**
“Verification is defined as the process of verifying that a design meets its specification.”
“Functional Verification is proving design intentions from the point of functional perspective. It proves presence of bugs, but cannot prove their absence”

- Functional Verification grows fast in last 20 years
  - The reason of rapid growth is developing of the methodology
  - Methodology brings metric driven verification, reusability, and universality
    - **Metric driven verification** = measure of quality of verification
    - **Reusability** = shorter product-time-to-market
    - **Universality** = readable code and faster ramp-up for verification engineers
- Two approaches of functional verification:
  - **Directed** verification
  - **Random Constrained** verification
“**Functional Verification** is proving design intentions from the point of functional perspective. It proves presence of bugs, but cannot prove their absence”

- Most relevant vendors:
  - Cadence (earlier Verisity)
  - Synopsis
  - Mentor Graphics

- Languages & Methodologies:
  - Specman eRM
  - Vera RVM, VMM
  - SystemC AVM
  - Specman oVM
  - System Verilog uvM
Functional Verification Example
Metric Benefits

Test-Driven Verification
- Test Coverage
- Code Coverage
- Assertion/Checks

Advanced Verification
- Constrained Random
- Coverage Driven
- Plan Driven

Productivity Benefits
- Relative Automation Effort
  - X
  - 2X
  - 3X

Timeline:
- Days
- Weeks
- Months

Proceses:
- Plan
- Measure/Analyze
- Construct
- Execute
Metric Example

results collection in unified environment

Environment Hierarchy

Code Coverage

Functional Coverage

Assertion Coverage

Test Case Coverage
“Formal verification is the process of proving or disproving properties using formal methods (i.e., mathematically precise, algorithmic methods). A formal proof of a property provides a guarantee that no simulation of the system considered will violate the property. This eliminates the need for writing additional test cases to check the property.”

☑ Equivalence Checking

☑ Compares two models to see if equivalence
☑ Proves mathematically that the origin and output are logically equivalent
☑ Examples:
  ☑ RTL to Gates (Post Synthesis)
  ☑ Post Synthesis Gates to Post PD Gates
“**Formal verification** is the process of proving or disproving properties using formal methods (i.e., mathematically precise, algorithmic methods). A formal proof of a property provides a guarantee that no simulation of the system considered will violate the property. This eliminates the need for writing additional test cases to check the property.”

- **Model Checking**
  - Form of formal verification
  - Characteristics of a design are formally proven or disproved
  - Looks for generic problems or violations of user defined rules about the behavior of the design
“The **analog verification** methodology is traditionally ad-hoc by nature, lacking the formalized methodology that is available on the digital side.”

- Analog verification is driven by directed tests run over **sweeps, corners**, and **Monte Carlo analysis**
- Analog verification
  - Analog top-level only runs DC and transient simulations
  - Directed simulation of “important” modes
  - Simulation time and convergence difficulty limits further use
  - MC, PVT, and extracted simulations run at “block” level (e.g., LNA)
- **AMS Verification**
  - Varies a lot; for us, use is generally limited to specific subblocks (ADC/DAC)
- **Digital Verification**
  - Digital teams are running a lot of mixed signal verification
  - Use behavioral models to make analog useful in the digital environment
“The **analog verification** methodology is traditionally ad-hoc by nature, lacking the formalized methodology that is available on the digital side.”
"The analog verification methodology is traditionally ad-hoc by nature, lacking the formalized methodology that is available on the digital side."
“Verification of a mixed-signal SoC involves many different levels of abstraction and modeling tradeoffs.”

- **Functional Verification**
  - The task of verifying that digital logic and analog input-output requirements are met

- **Parametric Verification**
  - The task of verifying that numerical requirements are met

- **Implementation Verification**
  - The task of verifying that functional and parametric requirements are met considering all the ways that circuits can “go wrong” in “analog way”

- **Reliability Verification**
  - The task of verifying that requirements continue to be met as prescribed by the reliability requirements.
“Verification of a mixed-signal SoC involves many different levels of abstraction and modeling tradeoffs.”

- **Simulation** is the problem
- **Simulators** are the answer

- **AMS Verification** is the problem
- A common verification strategy for Digital, Analog and Mixed Blocks
- **Predictable, repeatable, and observable** process
- **Advanced** debugging, diagnosis, analysis, requirements tracing
“Verification of a mixed-signal SoC involves many different levels of abstraction and modeling tradeoffs.”

- Event-Driven and Real Number Modeling
  - Fastest, least accurate, big digital best choice.

- Mixed-Signal (aka AMS) Modeling
  - Fast, can be accurate, big analog best choice.

- Digital/SPICE joined simulation
  - Slowest, most accurate analog, easiest.
“Verification of a mixed-signal SoC involves many different levels of abstraction and modeling tradeoffs.”

- Apart and together
  - Verifying Digital and Analog when apart: building blocks
  - Verifying Digital with Analog together: AMS IP
  - Verifying Digital and Analog integration: SoC
  - Adding power with UPF
- Testbenches
  - SystemVerilog with UVM (-MS)
  - SPICE or HDL-AMS
- Planning and management
  - Expanding Verification Run Management to AMS
  - Expanding UCDB to AMS
  - Expanding application of Req Tracer to AMS
"Verification of a mixed-signal SoC involves many different levels of abstraction and modeling tradeoffs."
“Verification of a mixed-signal SoC involves many different levels of abstraction and modeling tradeoffs.”
"Verification of a mixed-signal SoC involves many different levels of abstraction and modeling tradeoffs."

Mixed-Signal Verification

Analog Domain

Transistor-Level Schematic

Generate RNM Model

Generate Verilog-AMS Model

Validate Models to Circuit specs

Mixed-Signal Verification

Digital Domain

Real

D

D

Specs

D

D

D

D

D

e or SV Testbench
> DFT Verification

✓ DFT = Design For Testability
✓ DFT <-> Discredit Fourier Transformation
“**DFT verification** is a well-defined set of goals, supported by a methodology developed to provide integration-oriented test methods for chip-level DFT, to enable compatibility across different embedded cores and to incorporate high levels of reuse.”

- Design for testability (DFT) makes it possible to:
  - Assure the detection of all faults in a circuit.
  - Reduce the cost and time associated with test development.
  - Reduce the execution time of performing test on fabricated chips.

- DFT focus is on techniques for digital logic, although it is relevant for memory and analog/mixed-signal components as well.
“DFT verification is a well-defined set of goals, supported by a methodology developed to provide integration-oriented test methods for chip-level DFT, to enable compatibility across different embedded cores and to incorporate high levels of reuse.”

✓ The result is a fully automated path from ensuring:
  ✓ **Completeness** - The verification plan includes a section on all DFT features and their specifics
  ✓ **Intent** – The verification scope was defined early in the process by experts and with complete visibility
  ✓ **Uniformity** – Disparate test strategies can now be driven by a single process
**Verification Metric: Bug Tracking**

- **Bug Reporting** is the activity of posting a bug to the development team to fix it.
- Bug report should be effectively documented for increasing chances to be fixed.

- Qualities of a Good Bug Report:
  - **Reproducible** – Incorporate reproducible steps.
  - **Be Specific** – there should not be any ambiguity in bug summary and description.
  - Add required **screenshots** and error logs wherever required.

- Common Tools: Jira, UTP, PDM.
Thank you for listening!
Any questions?