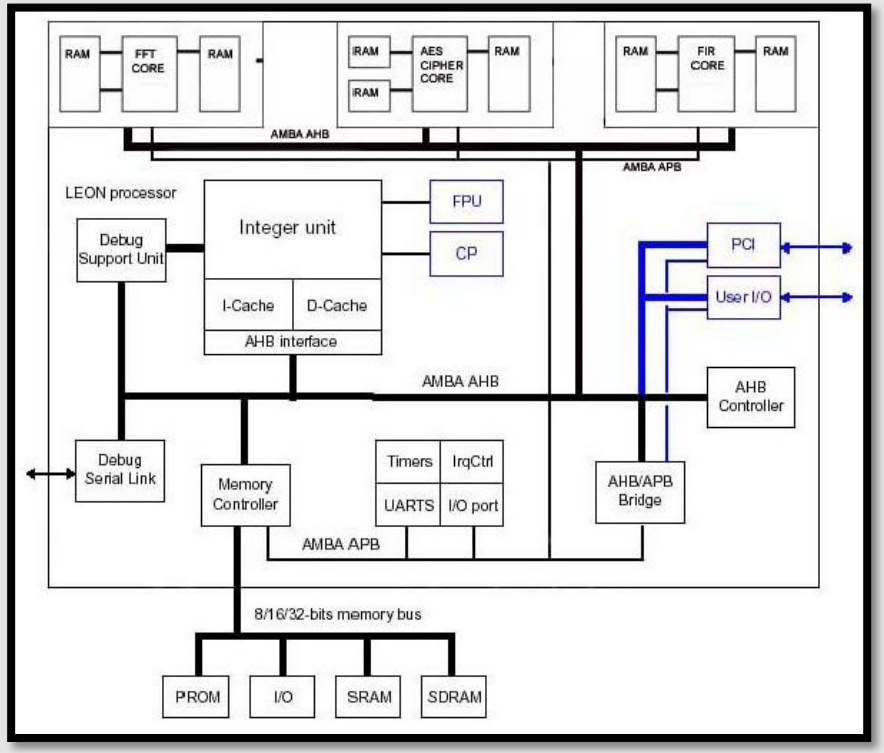
Design for Verification - Case Reopened

An EDA/IP perspective

The 70% Challenges in SoC Verification



Ever increasing SoC design complexity –
design size, clock speed, number of clock and power domains,
leads to increasing challenges for Soc Verification

70% Project cycle is spent in verification

70% Verification time is spent in creating env and tests, running simulation, and debugging failure

70% Projects are behind schedule

70% ASIC requires more than one respin

Design for Verification (DFV) today

- What is DFV? Specific design techniques and practices that make the verification process more manageable, spanning:
  - RTL development
  - RTL and IP Integration
  - Testbench and test practices for shortening and improving the quality of verification
  - Debug and observability improvement and best practices
- Benefits:
  - Reduced and efficient integration and verification efforts
  - Early error detection (“shift-left”)
  - Easier debug and root cause
  - More efficient verification cycles
  - Higher quality and confidence
- A sample of current techniques:
  - Integration:
    - Customized Auto-generation and stitching of code
    - Leverage existing language features such as config
    - Conventions macros / symbol / identifier naming
    - Acceptance criteria for IP integration
  - Verification:
    - Ad-hoc approaches for initialization shortening
    - X-injection, random init for “higher” quality verification
    - Custom controllability, backdoors, init shortening techniques
  - Debug: Building custom controllability and observability
    - Assertions, Monitors & Tracers, Coverage Points
    - Backdoors into memory, CSRs, trickboxes etc.,

Gaps and Pain points

Ad-hoc approaches

Code & design patterns

Verification techniques

Debug techniques

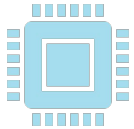
Checklists

Tribal knowledge

Corner cases and Edge conditions can slip through

Additional Production Verification?

Worst nightmare: bugs slip through due to the DFV ‘shortcuts’



Complexity Management

High # configurations

Multiple sources of IP of varying quality and complexity

Chip Integration

Multiple verification platforms

Time and Resource constraints

Do what works in the tight integration timeline

Internal and third-party IPs

Whose responsibility is it?

Design team

Verification team

Integration team

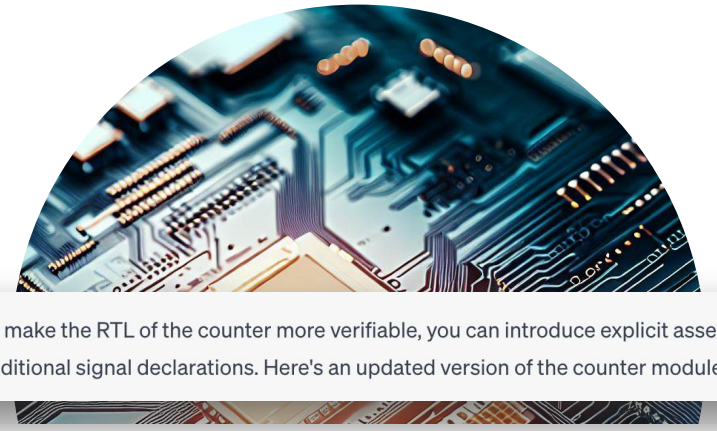
EDA, IP

Opportunities

Starting point: Evolve best practices into a set of Coding Styles, Methodology, cookbooks

Enablement Opportunities in EDA Tools and IP:

- Lint Checks and Hints
- Automated “DFV” Instrumentation
- Initialization state capture and restoration methodology and flow
- Debug Intent Capture
- Configuration Management best practices codified
- “Equivalence” between DFV shortcut and regular operation
- Impact assessment on PPA
- Design Intent Verification Capabilities (one example: X-propagation)
- Requirements on newly developed IPs and VIPs
- Higher Abstraction of Design (HGL? Domain Specific?)
  - Potential lowering phases / decorating / flavoring for ‘verifiability’
- Generative AI for DfV



To make the RTL of the counter more verifiable, you can introduce explicit assertions and additional signal declarations. Here's an updated version of the counter module:

In this updated version:

1. The counter's maximum value is 31'hFFFFFFFF, and the overflow behavior is modified to wrap around to 0 when the maximum value is reached. This is achieved using a ternary operator in the assignment to `next_out`. If the current value of `out` is equal to the maximum value, `next_out` is set to 0; otherwise, it is incremented by 1.
2. The shortened reset sequence remains the same as in the previous version, lasting for 2 clock cycles.

By shortening the counter overflow sequence, you reduce the number of clock cycles required to verify the counter's behavior.

JULY 9–13, 2023

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA