16 November 2022

Introduction to Hardware Verification.

Lars Viklund, ASIC Verification Manager, Axis Communications

ASIC Development at Axis Communications

Axis Communications

- Leader in network video surveillance
- Swedish company
- Part of Canon group
- About 4000 employees worldwide

Axis ASIC Development

- Long history of chip development
  - TGA/CGA – Printer connectivity
  - ETRAX – SoC for network connected devices
  - ARTPEC – Network video
- Why in house chip development?
  - Differentiation
  - Optimization for application
  - Access to technology
  - Unit cost

Axis ARTPEC Overview

- Each generation offers more:
  - features
  - processing power
  - memory bandwidth
- ARTPEC-1 (1999) – First chip for network video
- ARTPEC-2 (2003) – MPEG encoding
- ARTPEC-3 (2007) – One chip network camera
- ARTPEC-4 (2011) – Wide Dynamic Range solution
- ARTPEC-5 (2013) – Forensic Capture
- ARTPEC-6 (2017) – Security
- ARTPEC-7 (2019) – LightFinder 2.0
- ARTPEC-8 (2021) – Deep learning on the edge
- ARTPEC-9 (2023) – …

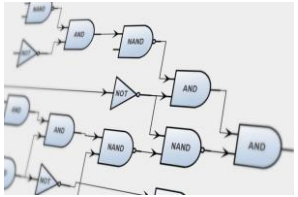
Overview of Hardware Verification

Digital ASIC Development Flow

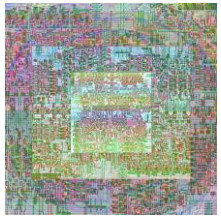
Specification

RTL Code

Netlist



Masks



Chip

Hardware Verification

- Ensuring that the hardware design fulfills the requirements
- Predominantly performed before tape-out
  - RTL
  - Netlists
  - UPF (power intent)
- Verification vs. test in the context of ASIC

Hardware Verification vs. Software Testing

- Same end objective
- RTL is similar to software
- However, hardware differs in that:
  - Very expensive to fix bugs after tape-out
  - Low abstraction level
  - Inherently parallel
  - Large state space

Requirements Specifications

- Ideal world:
  - Complete
  - Unambiguous
  - Consistent
  - Appropriate level of detail
- Reality:
  - Napkin sketches, gossip at the coffee machine, …
  - Natural Language
    - Formal or informal
  - Models
    - Executable or not
    - Bit exact or not

Types of Requirements

- Functional Requirements
  - Specifies the behavior
- Non-functional requirements
  - Performance
  - Reliability
  - Security
  - Safety
  - …

Verification Complexity is Increasing

- Larger and more complicated designs
- More elaborate non-functional requirements
  - Clock gating
  - Power gating
  - Security
  - Safety
  - …
- Stricter project deadlines
- Increased costs for tape-outs

Some Data

Source:
Wilson Research Group and Siemens EDA, 2022 Functional Verification Study

- 50%-60% median ASIC project time spent in verification
- Mean peak number of ASIC engineers:
  - 50% increase in design engineers since 2007
  - 145% increase in verification engineers since 2007
- 76% ASICs require 2 or more respins
- Mask costs (vs 28nm):
  - 12nm 2.5X
  - 5nm 15X
  - 3nm 25X

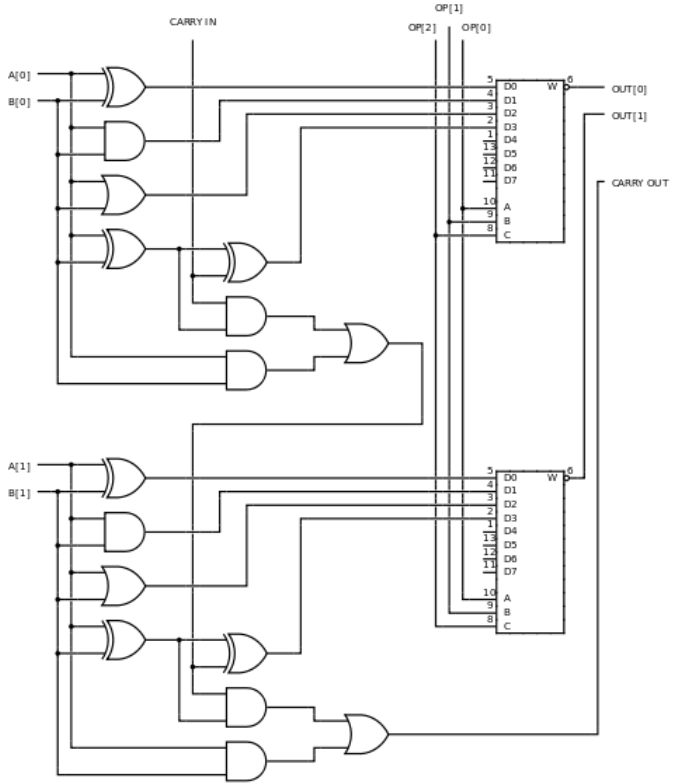
Dealing with Verification Complexity

- Work hierarchically
- Minimize overlap in effort
  - Horizontally
  - Vertically
- Use the best technique for each problem
  - Simulation
  - Formal
  - Specialized tools
- Re-use whenever possible

When is Verification Done?

Verification Techniques

- Dynamic Techniques
  - Simulation
  - Emulation and FPGA Prototyping
- Static Techniques
  - Structural checks
  - Formal verification



SystemVerilog

- Design Features
  - Synthesizable
  - Extends on Verilog
- Verification Features
  - Classes
  - Constraints
  - Coverage
  - Assertions
- Developed by the Accellera System Initiative
- Std IEEE 1800-2017

SystemVerilog Simulation

- Execute in a software environment
- Discrete event simulation
- Four valued logic
  - 0, 1, X, Z
- Performed on RTL or Netlist
  - RTL – Cycle behavior
  - Netlist – With or without timing
- Simulator can integrate other languages
  - Natively (VHDL, SystemC)
  - PLI or DPI

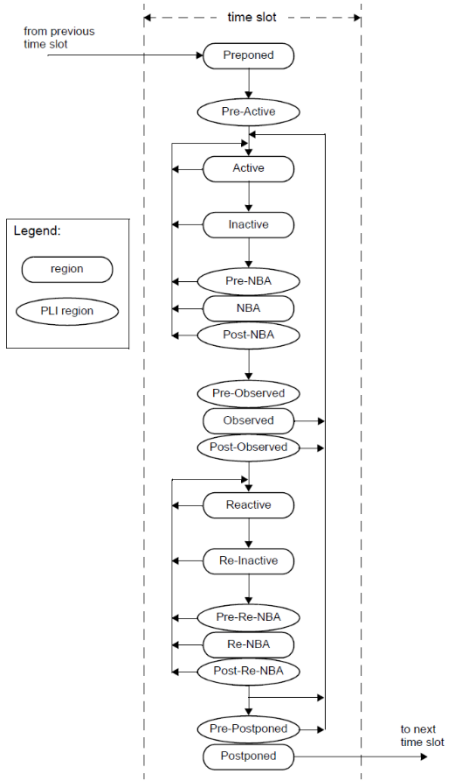


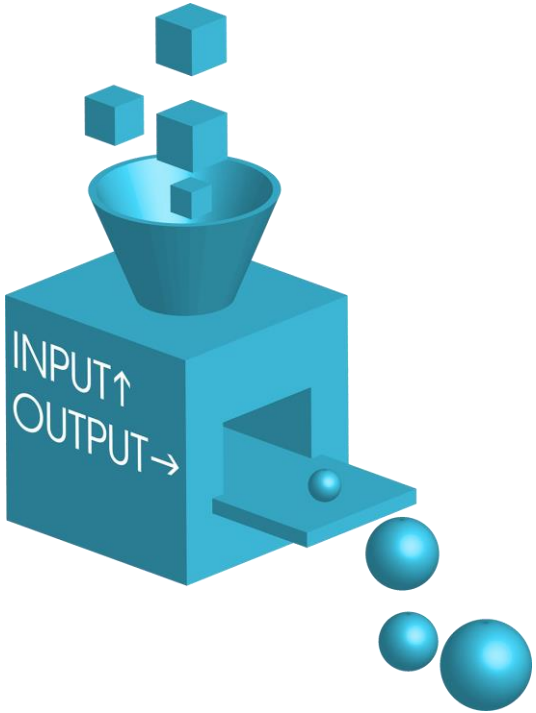
Figure 4-1—Event scheduling regions

Test Benches

- Instantiates the design under test (DUT)
- Facilitates simulating the DUT
- Models the environment of the DUT
  - Drives inputs
  - Monitors outputs
- Does not have to be synthesizable

Modeling the Environment

- Model what is outside the DUT
  - Other blocks on the chip
  - External devices
- Model all legal behavior according to the specification of the interfaces
- Model in layers
  - Handle complexity
  - Facilitate reuse



Coverage Driven Constrained Random Verification

Coverage Driven Constrained Random Verification

Stimuli

1101001001
0101000100
1100010010
0010101010
1110011001
0010010010
1001010101
0010101011

Checkers

DUT

Coverage

Result

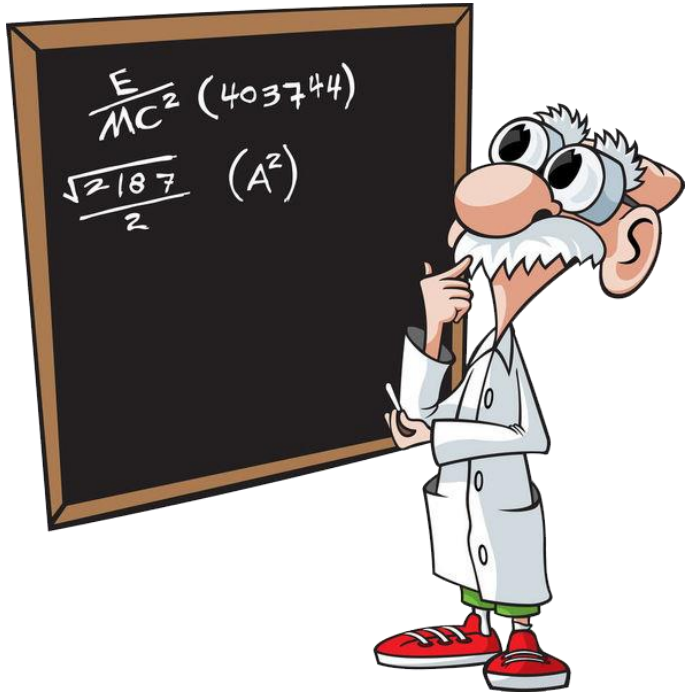
0111001001
0100100010
0100010010
0010101011
1000110100
1101010010
1001010101
0010101011

op ∈ [ld, st, mv]
x > 0 ∧ x < 4096
a = 2 ⨯ b

Constraints

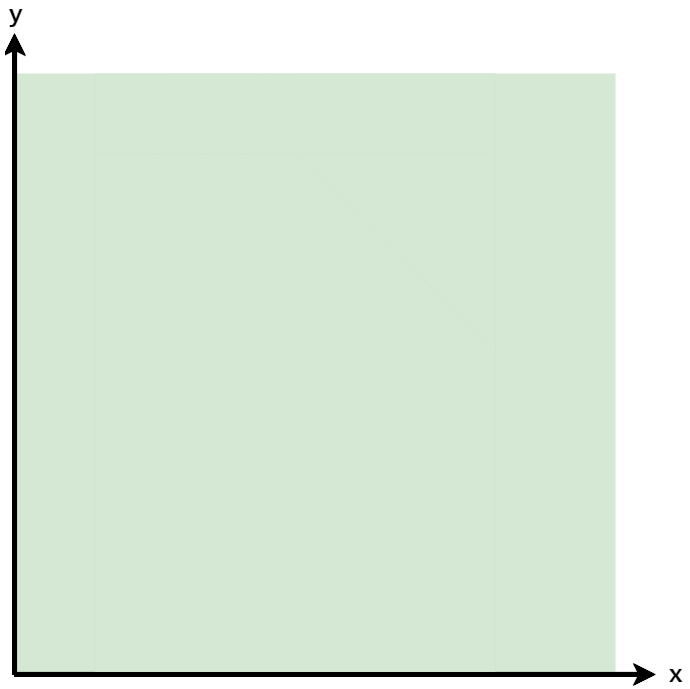
SystemVerilog Constraints

- Defines legal values
- Declarative
- Compact
  - loops, conditionals, set operators, *etc*.
- Constraint blocks are class members
- Solver find random solutions satisfying the constraints



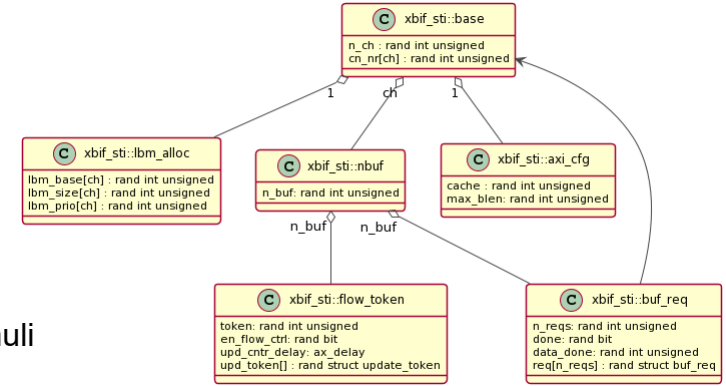
Constraints Example

```
rand logic [3:0] x;
rand logic [3:0] y;

constraint c_x { x inside {[2:12]}; }

constraint c_y { y < 14; }

constraint c_sum { x + y < 21; }

constraint c_x_mod { x % 8 != 0; }
```



Modeling Stimuli

- Model the input to the DUT as random variables
  - Configuration and data
  - Structure using object-orientation
  - Consider reuse of parts
- Express legal stimuli using constraints
- Abstract low-level stimuli when appropriate
  - To simplify constraints
  - To enable us to randomize interesting cases
  - Randomize high-level “knobs” and map to concrete stimuli
- Appropriate granularity of randomization
  - Randomize at once or sequence the randomizations
  - Tradeoff between expressiveness and performance

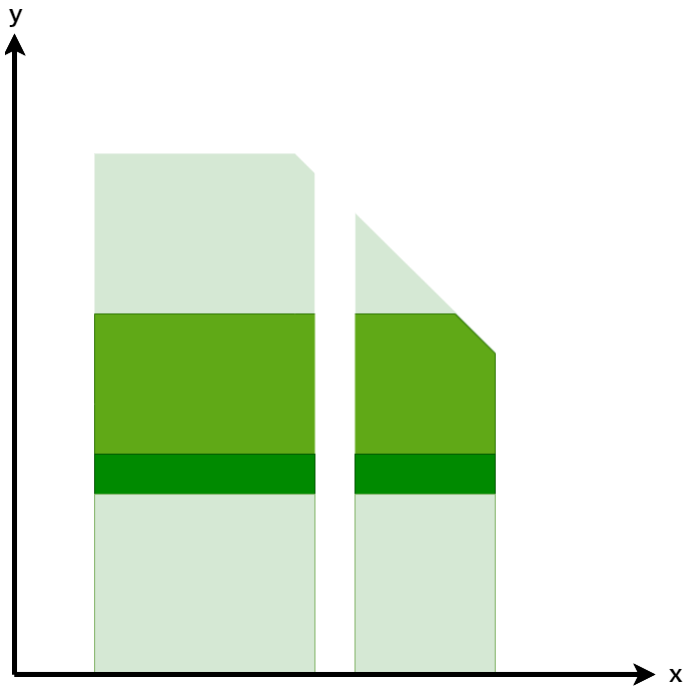


Controlling Random Distribution

- Random distribution is rectangular by default
- In many cases, we need other random distributions to hit all interesting cases
- Define alternative sets of constraints
  - Use subclasses
- Add constraints to limit the solution space
- Add **dist** constraints to control random distribution of a variable

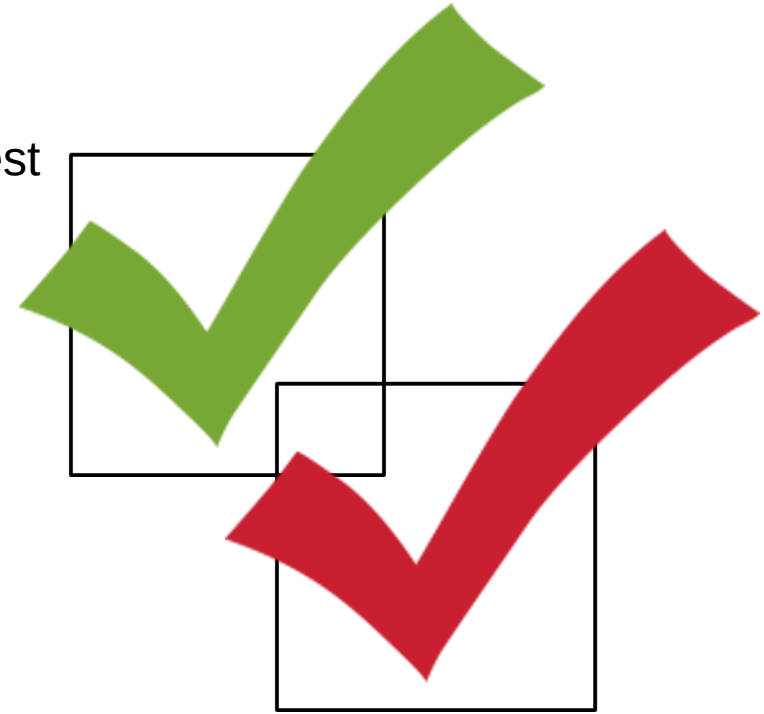
Distribution Constraint Example

```
constraint c_d {
  y dist { [0:4] := 1,
             5   := 5,
           [6:9] := 2 }
}
```



Checkers

- Check that the DUT behaves correctly
- Passive and independent of how the test drives the stimuli
- Handle any legal stimuli
- Can be split into multiple parts
  - Check different aspects
  - Check at different abstraction levels



Reference Models

- Check by *score boarding* the output from the DUT against a reference model
- Abstraction level may vary
  - Cycle exact or transaction level
- Accuracy may vary
  - Bit exact or approximative

Coverage

- Primary metric for measuring completeness
- Two main types:
  - Code Coverage
  - Functional Coverage
- Collected during simulation by the simulator
- Analyzed to determine which aspects have not been fully verified yet

Code Coverage

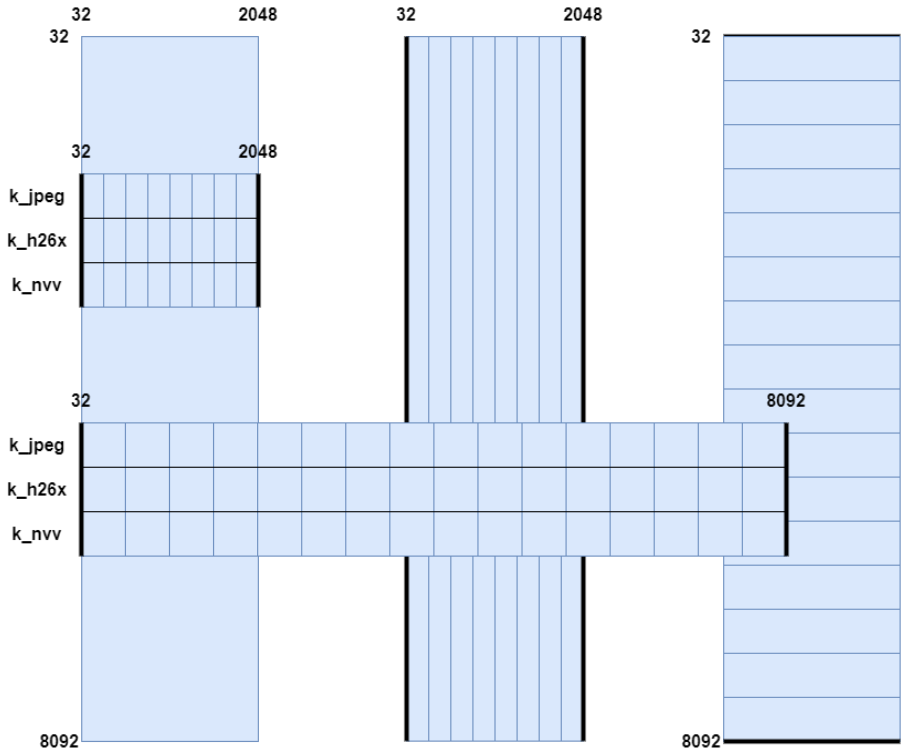
- Implicit
- Based on the code
- Different types:
  - Line Coverage
  - Branch Coverage
  - Toggle Coverage
  - Condition Coverage
  - FSM Coverage

Functional Coverage

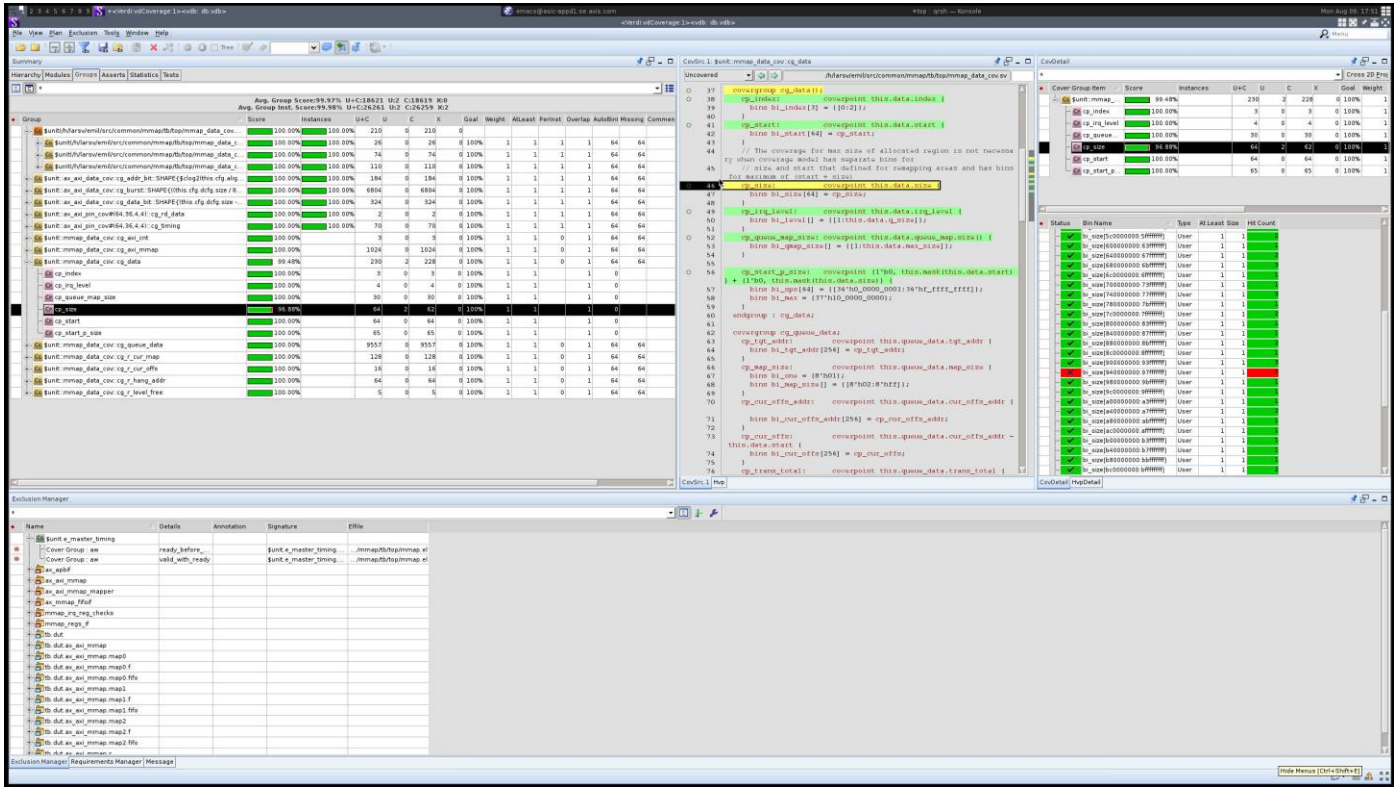
- Explicit
- Based on specification
- Defines interesting cases
- Sample values during simulation:
  - Configuration
  - Input data
  - Output data
  - Internal state
  - Combinations of the above
- Use knowledge about the design
- Consider observability

Functional Coverage Example

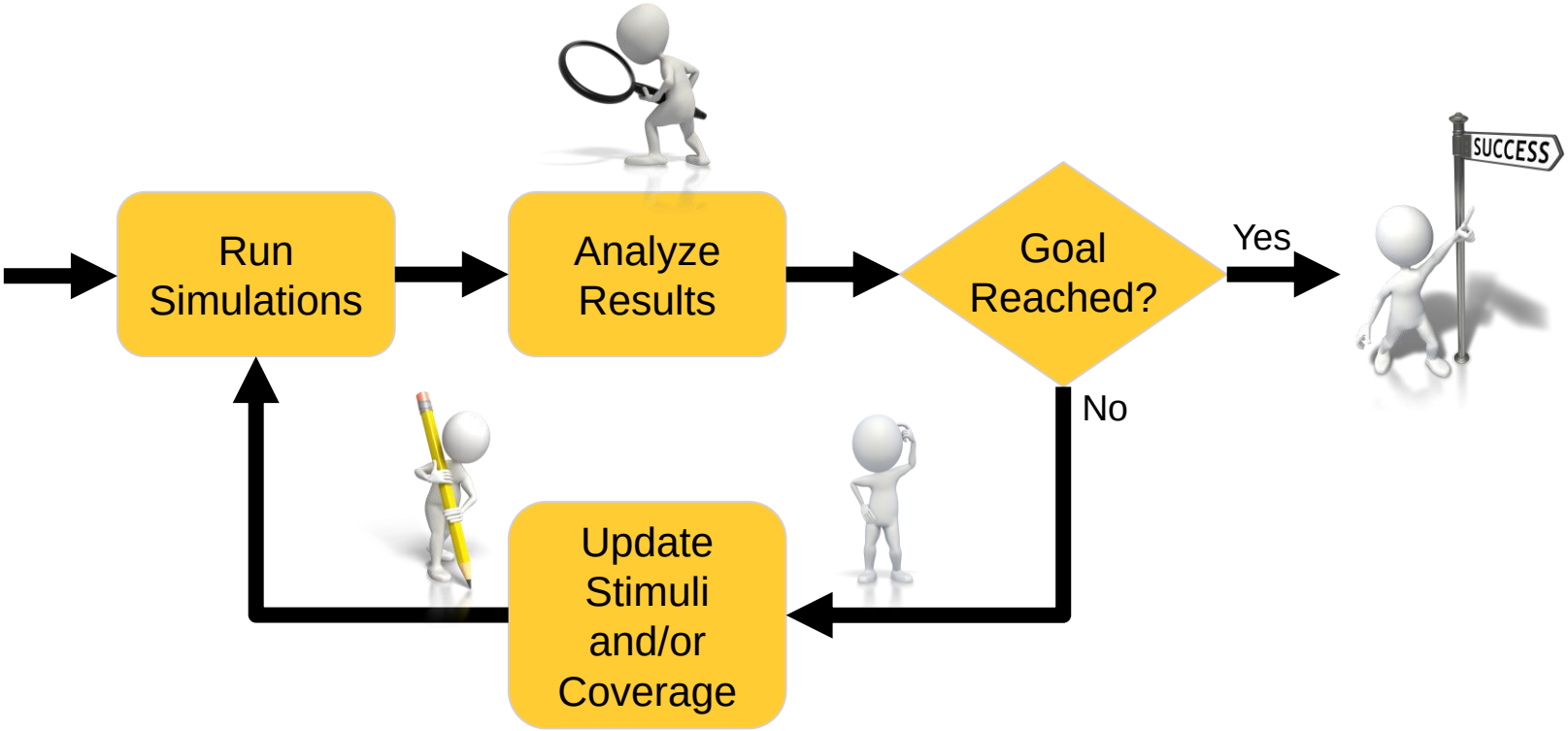
```
covergroup cg_upsamp_cfg;

  cp_subsamp: coverpoint subsamp {
    bins jpeg       = {k_jpeg};
    bins h26x       = {k_h26x};
    bins nnv        = {k_nnv};
    illegal_bins ib = default;
  }

  cp_xs: coverpoint xs {
    bins min          = {       32};
    bins max          = {     2048};
    bins xsize[ 8]    = {[32:2048]};
    illegal_bins bad = default;
  }

  cp_ys: coverpoint ys {
    bins min          = {       32};
    bins max          = {     8192};
    bins ysize[16]    = {[32:8192]};
    illegal_bins bad = default;
  }

  cc_ss_xs : cross cp_subsamp, cp_xs;

  cc_ss_ys : cross cp_subsamp, cp_ys;

endgroup: cg_upsamp_cfg;
```



Coverage Reports



Closing Coverage

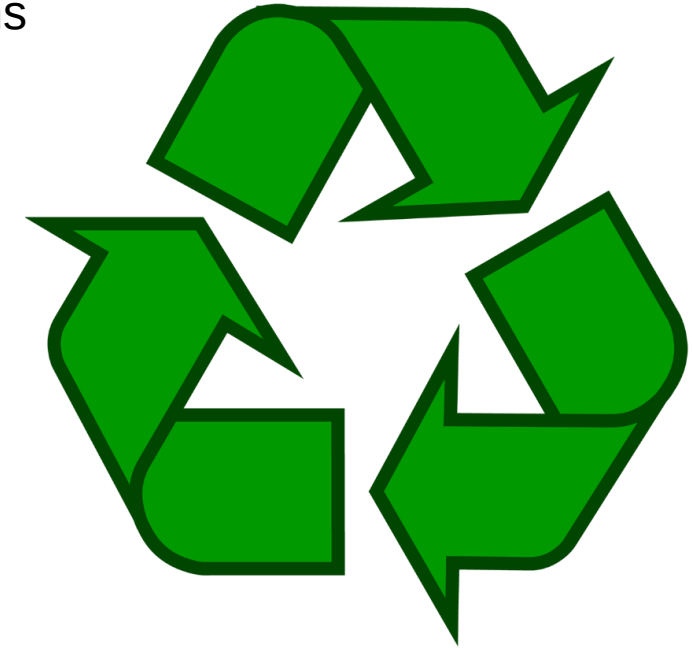


Universal Verification Methodology

- Framework for building coverage driven constrained random test benches
- Consists of a class library
- Std IEEE 1800.2
- Amalgamation of several previous competing frameworks

Code Reuse

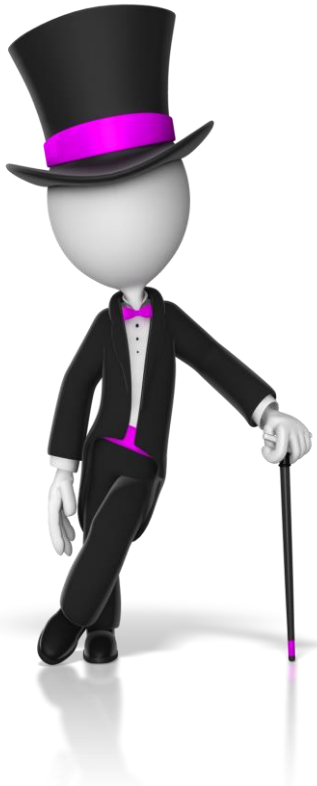
- Test benches are complex software systems
- Large amount of code
- Many test benches for a chip
- Code reuse is essential
  - Across similar test benches – Horizontal
  - Across hierarchy levels – Vertical
  - Across iterations of the chip



Formal Verification

Formal Verification

- Use of tools that analyze the space of all possible behaviors of a design
- Excellent complement to simulation for selected problems
- Complexity often an issue for real world designs



Summary

- Verification is an essential part of developing integrated circuits
- Active area
  - New techniques
  - New tools
- Requires both hardware and software knowledge

THANK YOU

We have more than 50 thesis proposals within:

AI	Software
Machine Learning	Algorithms
Image Quality	Radar
Audio Quality	Lidar
Supply Chain Management	Audio
UX Design	Firmware
Electronics	Deep Learning
ASIC	Image Processing
Testing	Analytics
Mechanics	Computer Vision