life.augmented

Faculty Development Program’23: Verification

Mansi Chadha

Abhinav Gaur

About myself

I am currently managing the SoC Verification team for General Purpose Micro (GPM) products in ST, Greater Noida.

I have extensively worked on MCUs and MPUs across Industrial & Automotive applications and contributed to multiple successful tape-outs.

I have driven key initiatives in testbench standardization, regression automation, coverage, GLS, etc. In formal verification, I have helped in establishing many areas of its application. I have a total of 22 publications/presentations in various forums and 2 patents in the area of functional safety.

life.augmented

Agenda

#1 Introduction

#2 Purpose of Verification

#3 Verification Platforms

#4 Levels of Verification

#5 Popular Verification Methodologies SV and UVM

#6 Verification Sign-Off Requirements and Criteria

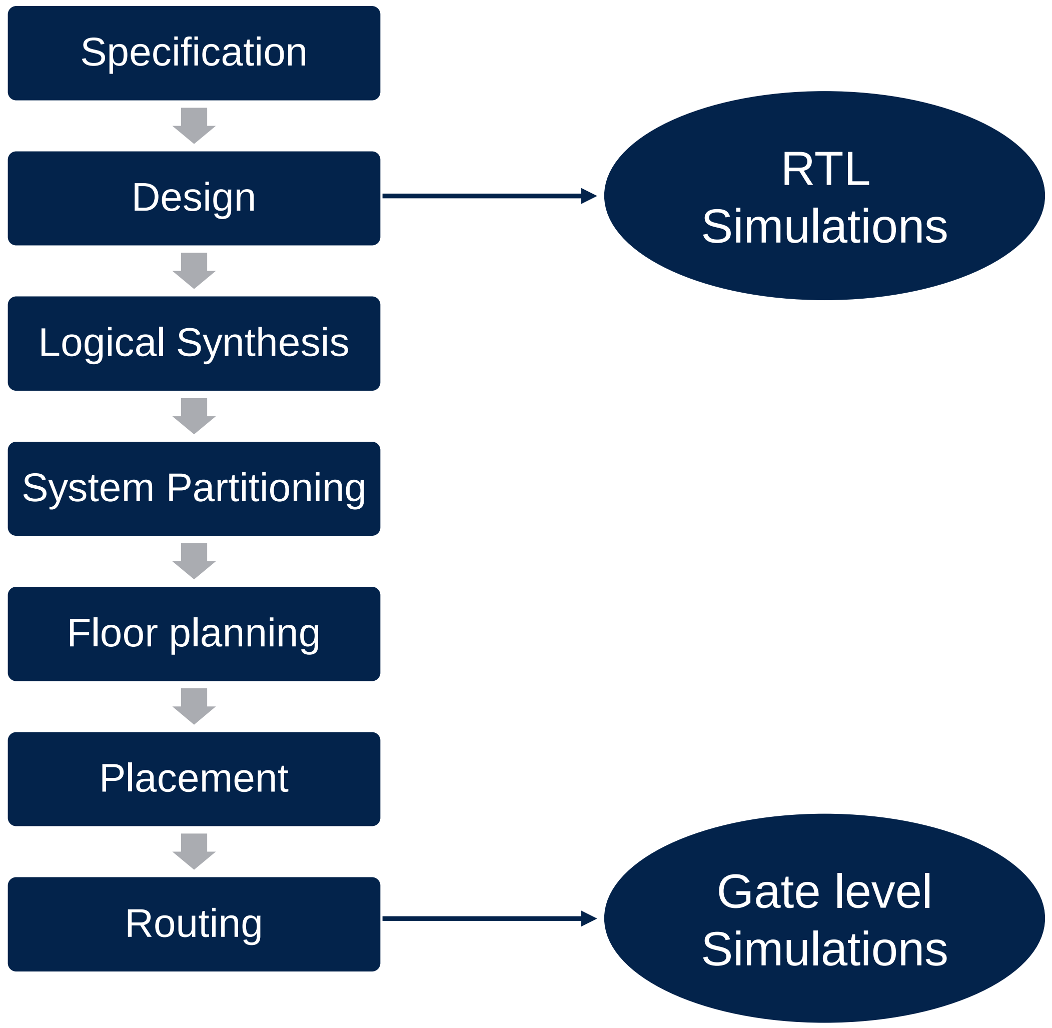
#7 Research areas in verification

life.augmented

Introduction

life.augmented

ASIC Design Flow



ASIC Design Flow

- From Specifications to RTL
  - Human Translation
  - Designer understands the spec and writes the RTL in Verilog, VHDL, etc.

- From RTL to Netlist to Layout to Physical
  - Mostly automated translation
  - Constraints from designer are understood by timing team while doing static timing analysis (STA)

- The main source of bugs is
  - Specs to RTL translation
  - Gate level simulations on final netlist with timing information to validate the correctness of constraints

life.augmented

ASIC Implementation Flow

1. Write RTL language
2. Verify RTL and find bugs
3. Fix RTL
4. Verify RTL and find bugs
5. Repeat from 3 until finished
6. Synthesize to netlist and move further on implementation
7. Run Gate level simulations on netlist along with timing information in SDF (Standard Delay Format)

Purpose of Verification

life.augmented

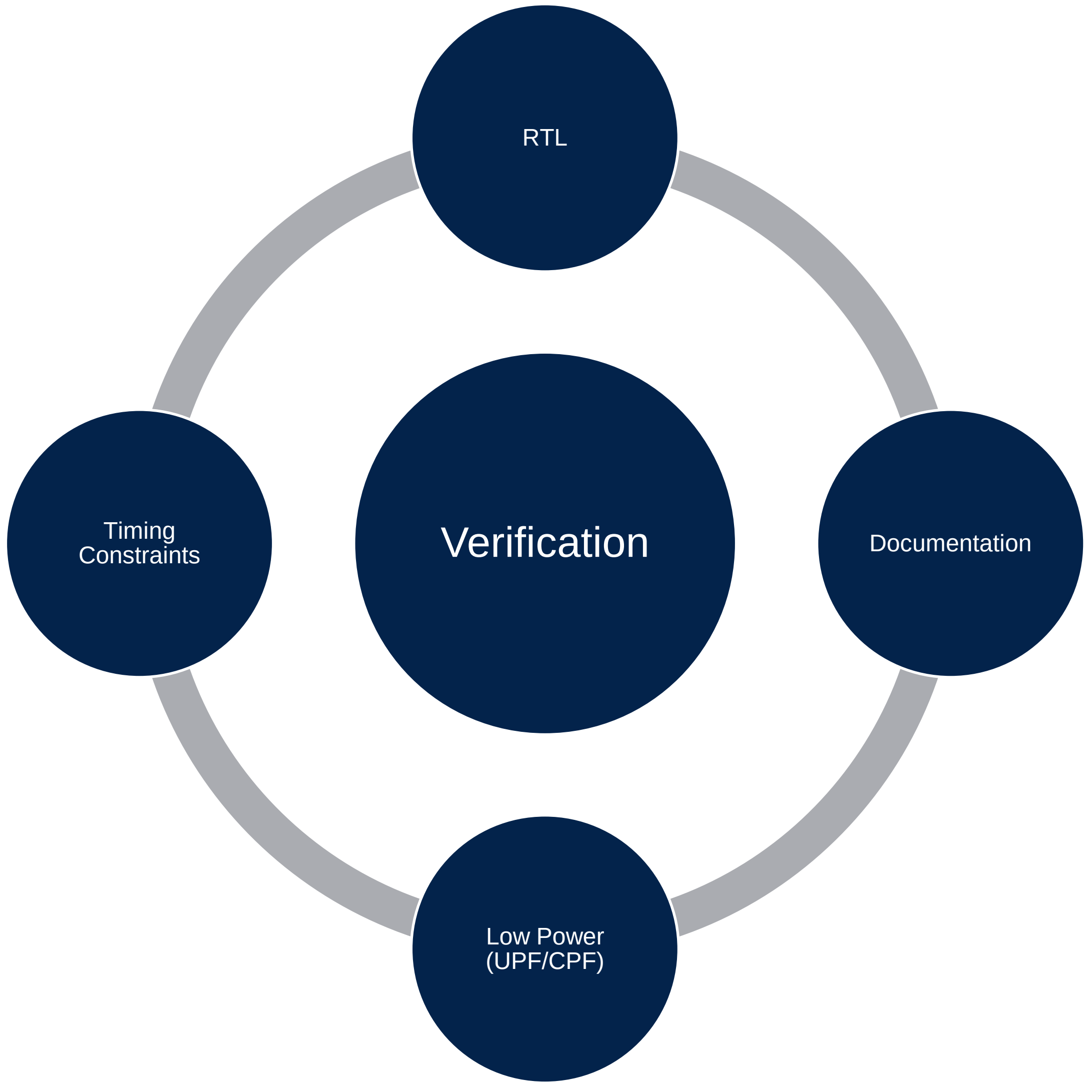
What is meant by Verification?

- Verification:
  - *The process or an instance of establishing the truth or validity of something.*
  - *The process of demonstrating the intent of a design is preserved in its implementation.*
  - *Does not ensure that specification is correct…*

Why is it needed?

- To prove that the design behavior matches its specifications/description in the documentation
- To use the design in a same way as end user before it gets taped-out
- **Cost of fixing a bug once chip has been manufactured is huge!**

What all is verified?



Verification Platforms

life.augmented

Verification platforms

- Dynamic: Stimulus generated by user
  - Simulation
  - Accelerated Verification (Emulation, FPGA)
- Static: No stimulus needed, proves like a theorem
  - Formal Verification

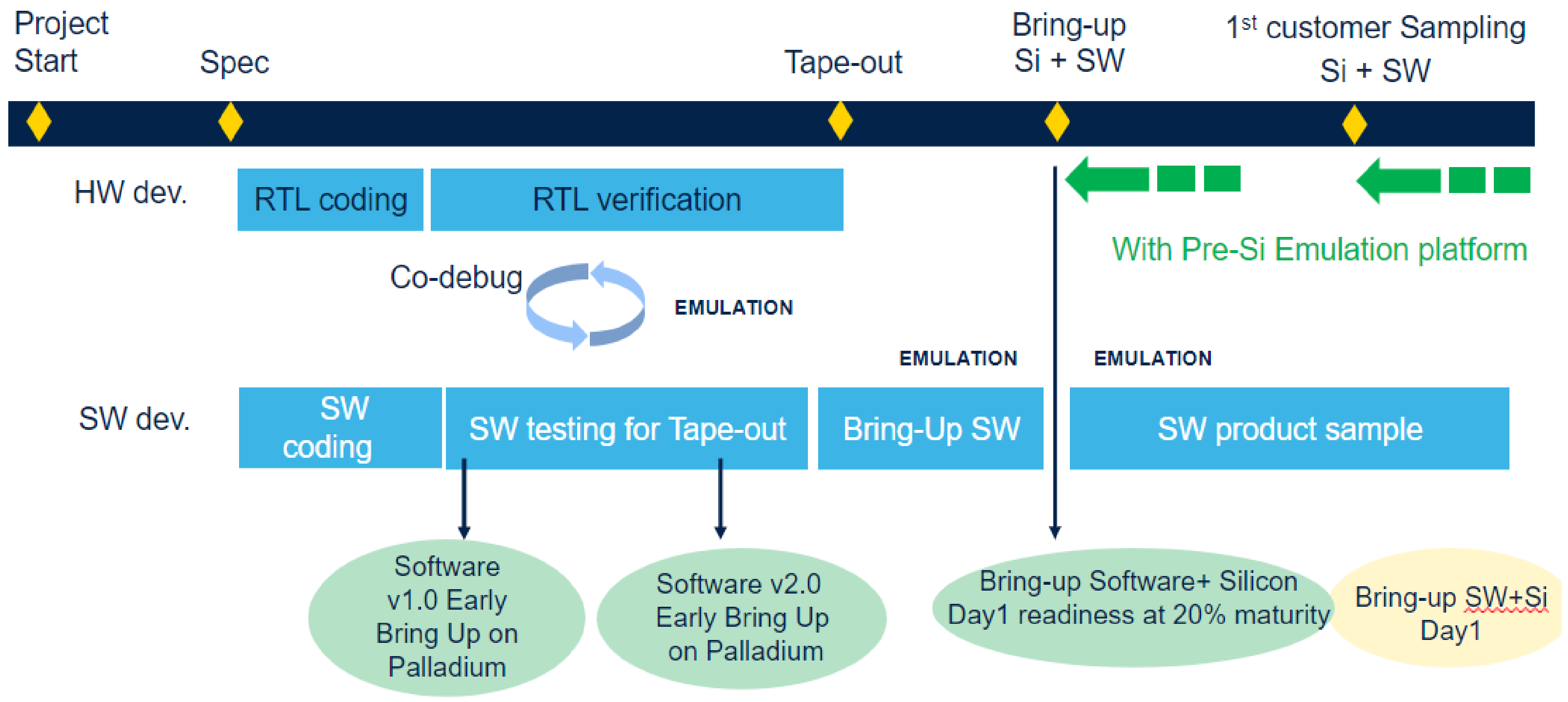
life.augmented

Simulation

- System Verilog/UVM based testbench at IP/SS levels to generate stimulus to program the IPs as well drive/monitor the interfaces
- At SoC level, typically CPUs exist to program the IPs, hence the need of C-based tests. The external interfaces are driven/monitored through system Verilog/UVM based components
- Randomization to generate stimulus more efficiently
- Coverage to measure generation of all possible stimulus and find gaps in the environment

life.augmented

HW accelerated verification


Project Start
Spec
Tape-out
Bring-up Si + SW
1st customer Sampling Si + SW
HW dev.
RTL coding
RTL verification
With Pre-Si Emulation platform
Co-debug
EMULATION
EMULATION
EMULATION
SW dev.
SW coding
SW testing for Tape-out
Bring-Up SW
SW product sample
Software v1.0 Early Bring Up on Palladium
Software v2.0 Early Bring Up on Palladium
Bring-up Software+ Silicon Day1 readiness at 20% maturity
Bring-up SW+Si Day1

Flow showing Early SW Development and H/W Verification using Pre-silicon Emulation Platform during SI Development

life.augmented

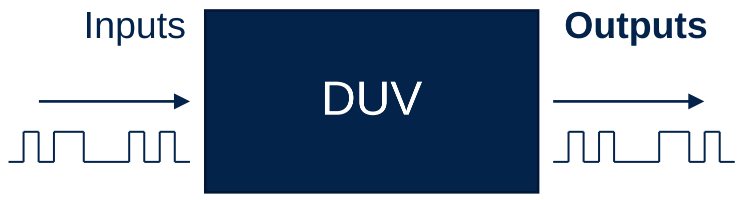
Gate level simulations (GLS)

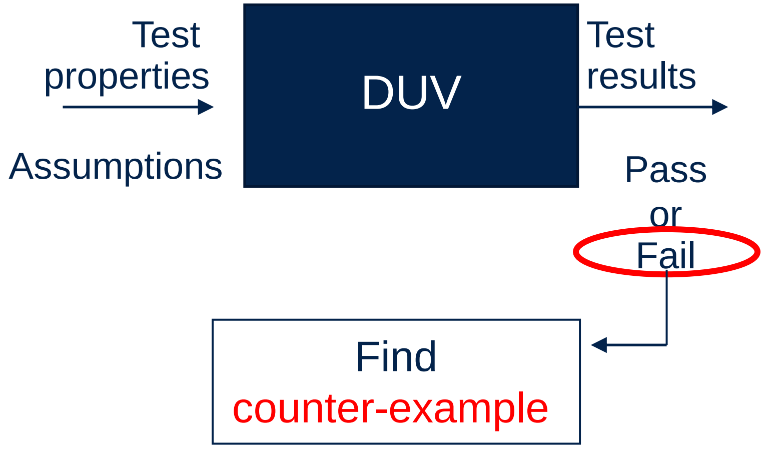
- Instead of RTL, simulations are run on the actual netlist
- Can be run without delay (as part of bringup) and then with timing information in the form of SDF (Standard Delay Format) in different timing corners
- Confidence building mechanism, not a way of signing-off timing or netlist equivalence to RTL
- Aims at verifying constraints applied correctly by physical design team, glitches during power-up/boot/shut-down sequence
- To avoid any surprises on running tests on actual netlist with delays in places (RTL is zero delay)

Formal

- Mathematical approach of proving/disproving the effectiveness of a statement
- Assumptions added to avoid false failures
- Design specifications are converted to assertions which act as checkers
- Stimulus comes for free!
- Due to its exhaustive nature, cannot be applied for complex designs. Needs to be smartly applied in the right areas of the design.
- A combination of formal and dynamic techniques is recommended for faster sign-off

Simulation-based versus Formal Verfication





- ❑ Dynamic
- ❑ Cannot cover all possible cases
- ❑ Possibility of surviving (corner case) bugs

- ❑ Static
- ❑ **Exhaustive** : equivalent to simulating all cases in simulation but fully static
- ❑ Requires modeling of the context (assumptions)
- ❑ Faces complexity limitations

Levels of Verification

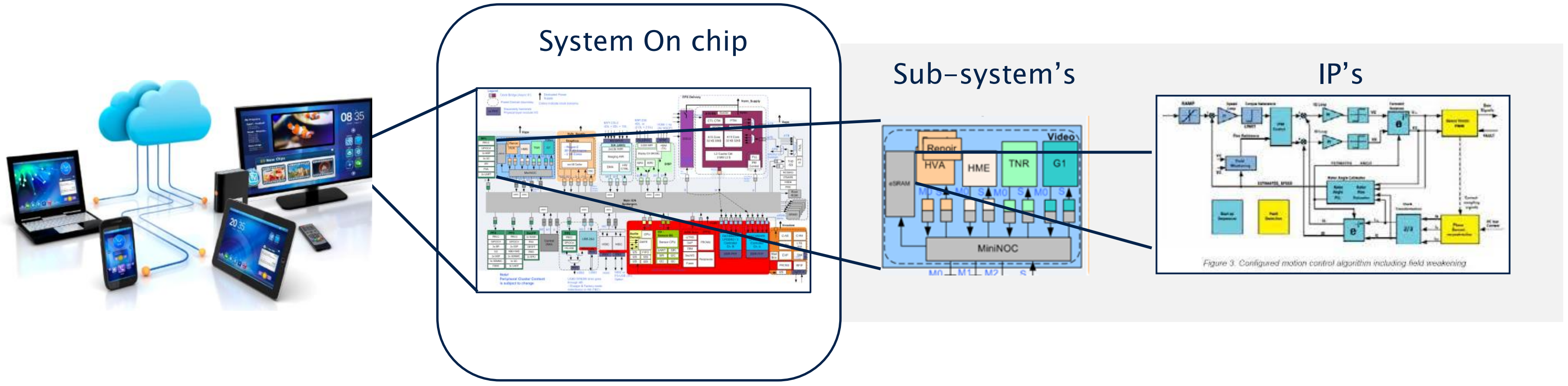
life.augmented

Stages of Verification

- IP Verification
- SS Verification
- SoC Verification

life.augmented

Functional Verification at different levels


System On chip
Sub-system's
IP's
Repair
HVA
HME
TNR
G1
Video
eSRAM
MiniNOC
Figure 3. Configured motion control algorithm including field weakening

SoC Verification

- Connectivity (Formal)
- Interoperability (Simulation)
- Performance - Application Scenarios (Emulation/Simulation)

SS Verification

- Connectivity (formal)
- Interoperability (simulation)
- Performance - Application Scenarios (Emulation/Simulation)

IP Verification

- Functionality (UVM)
- Protocol Compliance (Formal)
- Standard Compliance (Formal/Simulation)
- Performance - Application Scenarios (Emulation/Simulation)

life.augmented

Mansi Chadha : About myself

I am currently managing the Digital IP Verification team for General Purpose Micro (GPM) products in ST, Greater Noida.

I bring in a demonstrated history of IP/SubSystem/SoC Level Verification and Verification IP portfolio.

I have extensive experience in UVM/SystemVerilog/ C/C++/SystemC/Transaction Level Modeling (TLM), Object-Oriented concepts and Formal Verification.

I have expertise in project management comprising of defining Verification Methodologies, Verification Milestones and Roadmaps, Execution, and mitigation plans.

I am a result oriented and self-driven professional with leadership qualities of managing multiple Metric Driven Verification projects across different geographies and having multiple technical publications//presentations/articles, and patents.

life.augmented

Agenda

life.augmented

life.augmented

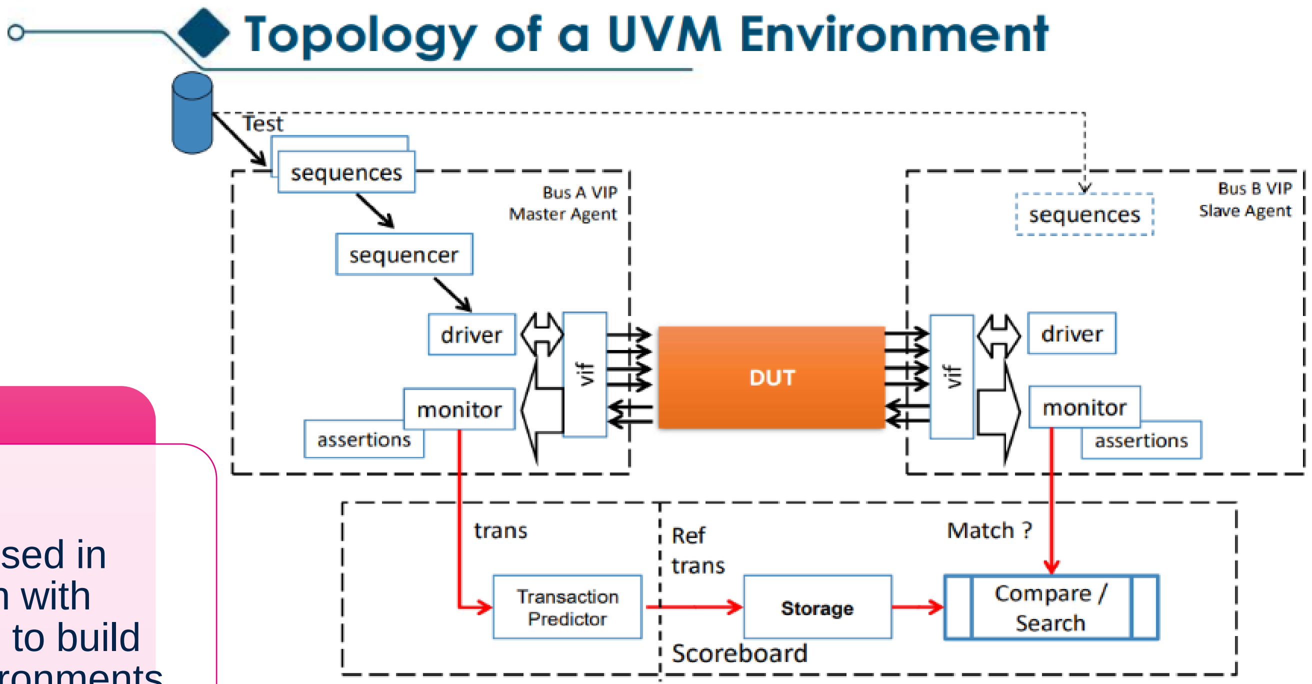
Popular Verification Methodologies SV and UVM

Overview

UVM (Universal Verification Methodology) and SystemVerilog (SV) are two popular methodologies used in the verification of digital circuits and systems

While both UVM and SystemVerilog can be used to build verification environments

UVM often used in combination with SystemVerilog to build verification environments



SV : System Verilog

- Is a hardware description and verification language that provides a rich set of features for design and verification.
- It includes support for data types, classes, interfaces, and other programming constructs that make it easier to write more complex and sophisticated verification environments.
- Provides the programming constructs needed to implement a comprehensive and sophisticated verification environment.
- The SystemVerilog testbench, on the other hand, is a more general-purpose verification environment that can be designed and implemented using SystemVerilog constructs.
- It is also used to implement assertions to check a particular design functionality : called as SVA (System Verilog Assertions)

life.augmented

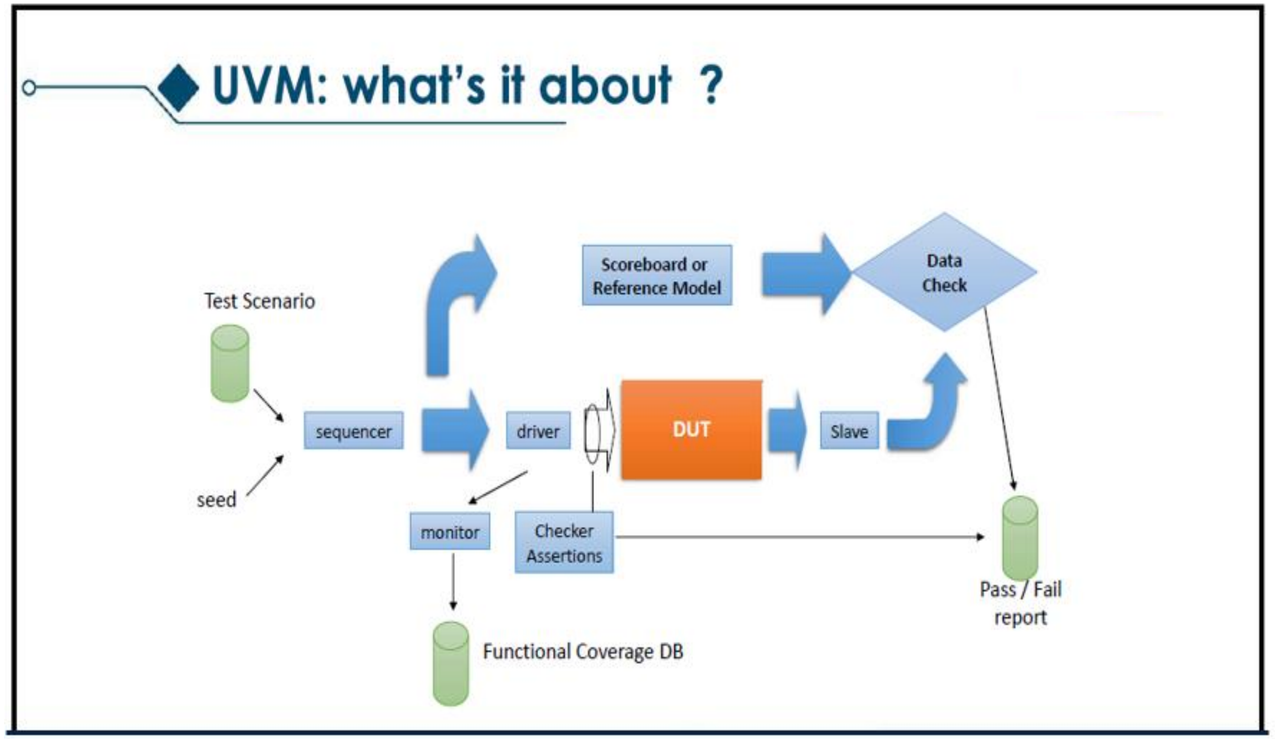
UVM (Universal Verification Methodology)

Is a standard developed by Accelera

It is both a methodology and a class library for building advanced reusable verification component

Provides the best framework to achieve coverage-driven verification (CDV) or Metric Driven Verification (MDV)

The verification components based on UVM are called UVCs (Universal Verification Components)

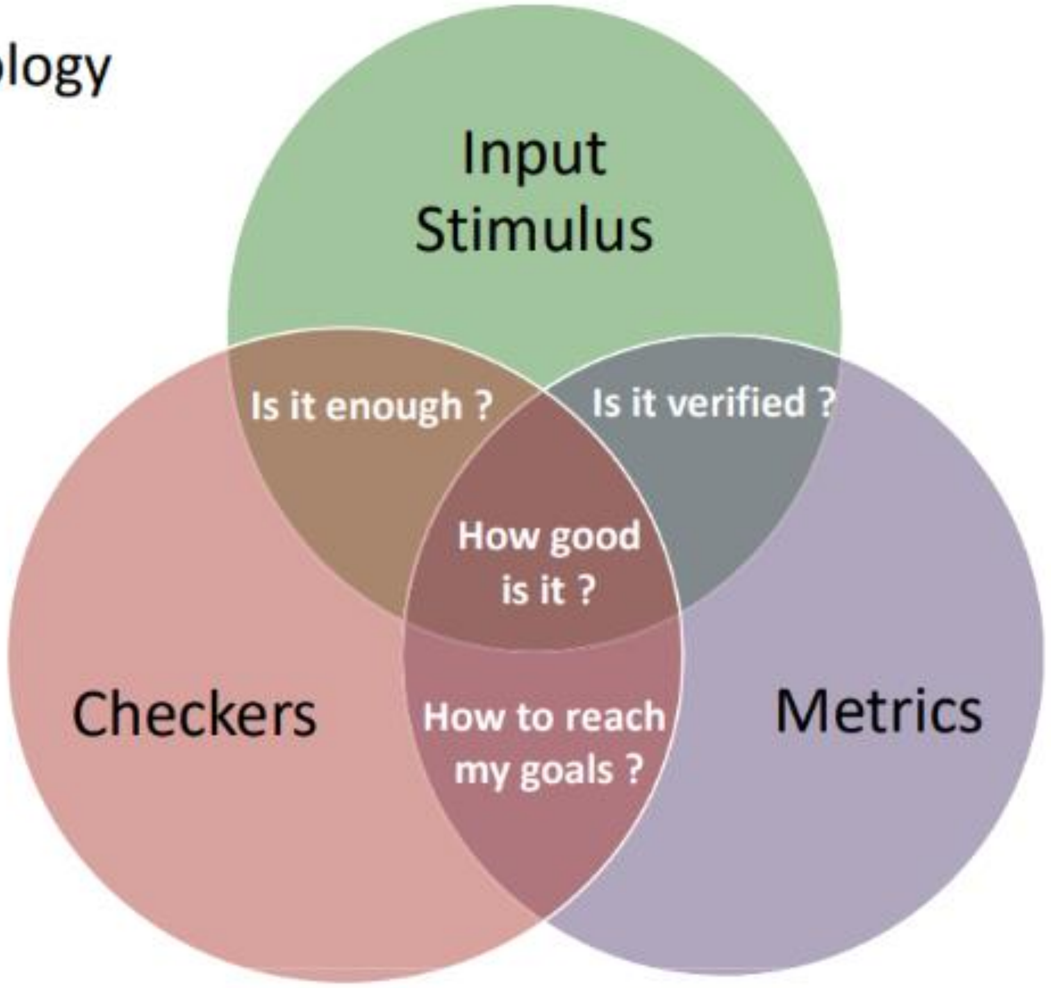


UVM and coverage driven verification methodology

Coverage Driven Verification Methodology
- All system behaviors should be:
  - Exercised (input generation)
  - Checked (verification)
  - Monitored (to lead us to our goal)

SystemVerilog
- Constrained Random Generation
- Functional Coverage
- Assertions

UVM provides
- test scenarios automation
- support for transaction checking
- framework to connect monitors



UVM (Universal Verification Methodology)

UVM is a verification methodology that provides a standard and reusable architecture

-> For verifying digital circuits and systems.

-> It provides a library of components and classes that can be used to build a verification environment, including testbenches, agents, and scoreboards

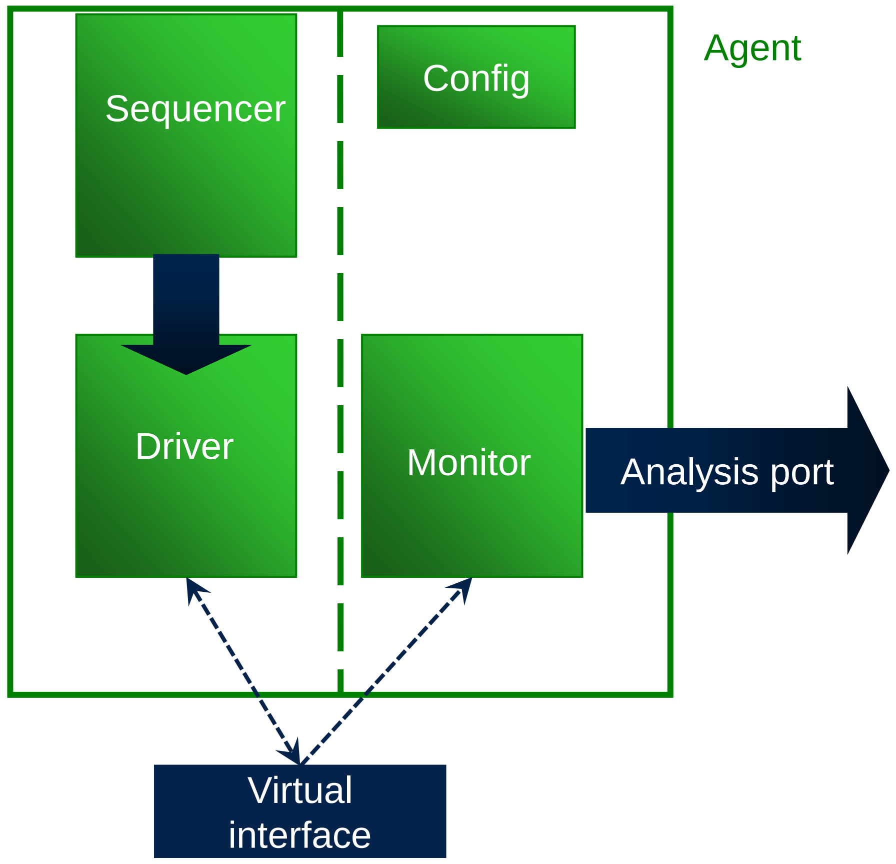
UVM also provides a standard way of defining, configuring, and communicating between these components, making it easier to reuse verification components and ensure consistency across different verification projects

Useful Definitions

• Virtual Interface	Connected to the signals of the design
• Driver	Drives actual signals from a higher level transaction Get transaction items from the sequencer
• Sequencer	Executes the sequences Generates transactions and sending them to the driver
• Monitor	Snoops actual signals Checks the protocol and report coverage
• Configuration	Deal with configuration fields, settings, variabilities

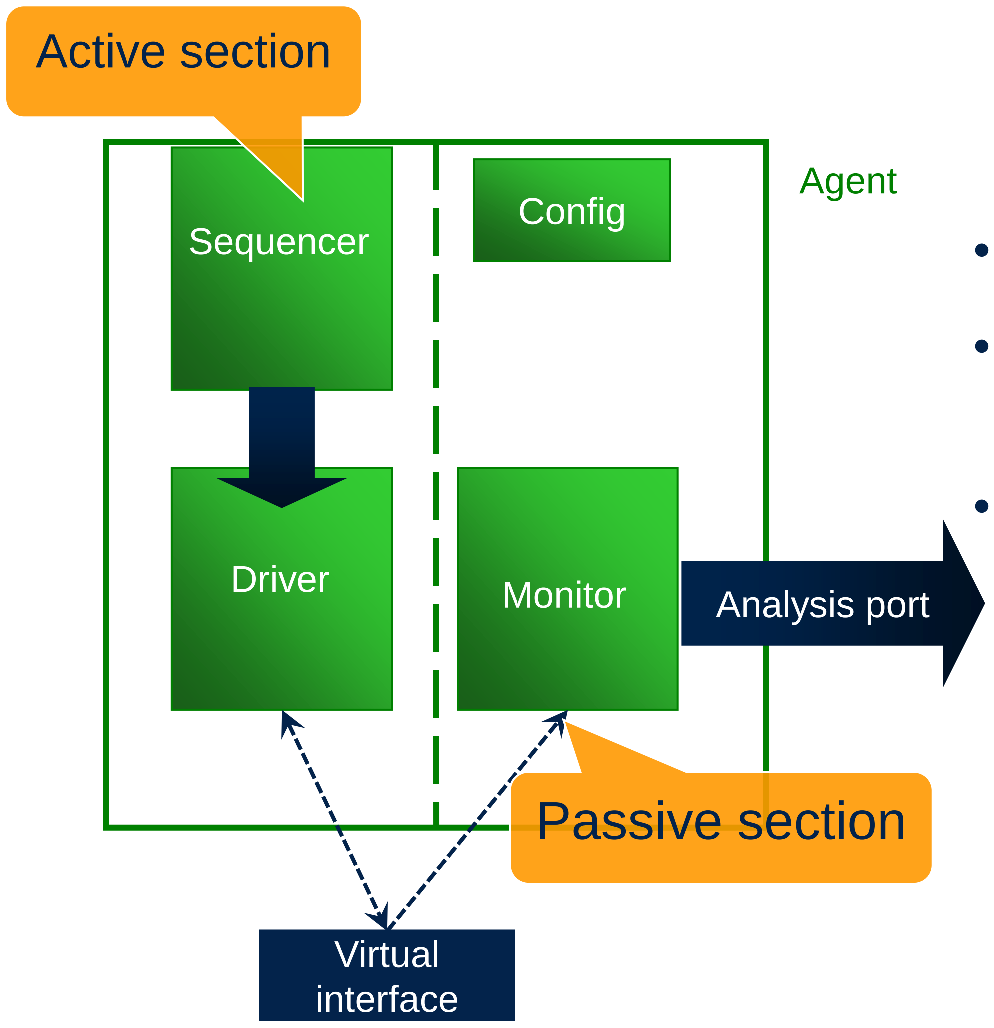
life.augmented

UVM architecture



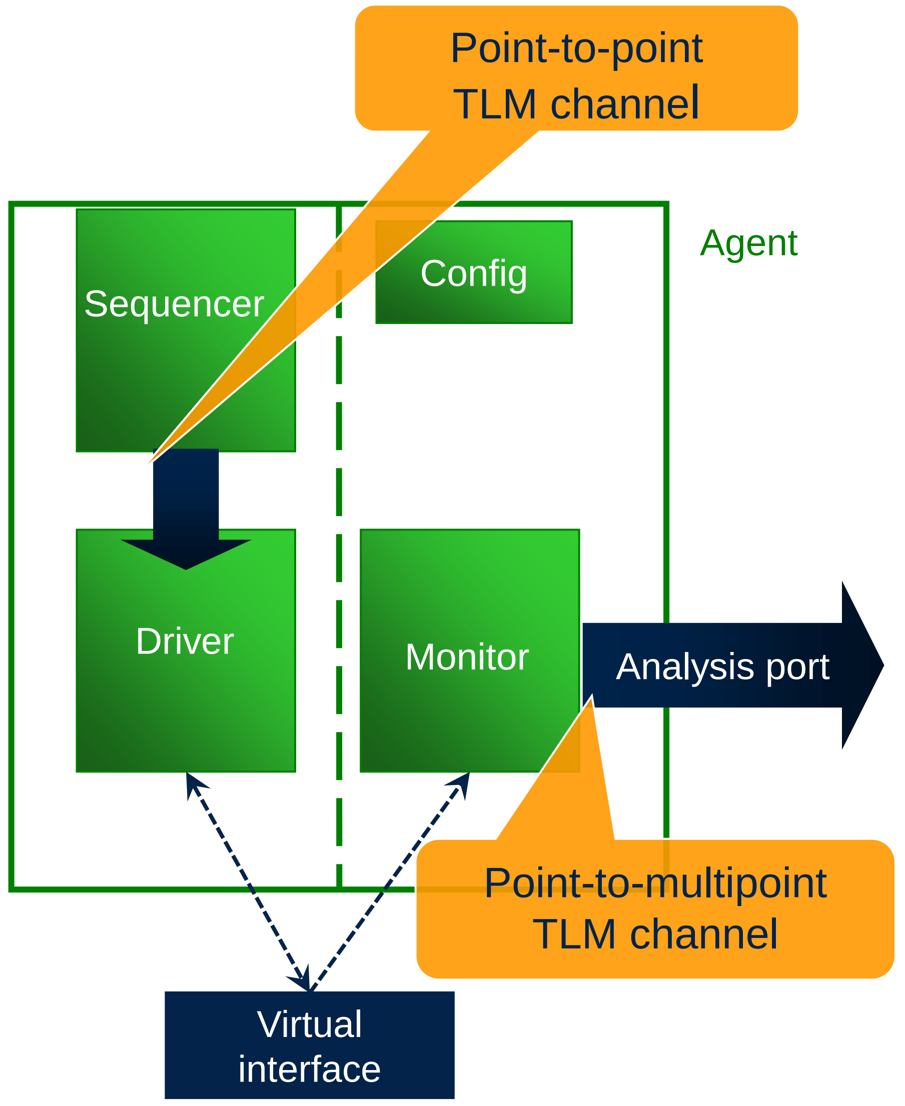
- A UVM agent collects together a group of verification components focused on a specific pin-level interface
- A generic agent has:
  - A sequencer for generating traffic
  - Driver to drive the DUT
  - Monitor
- The monitor is independent of the driving logic
- Agent has standard configuration parameters

UVM architecture



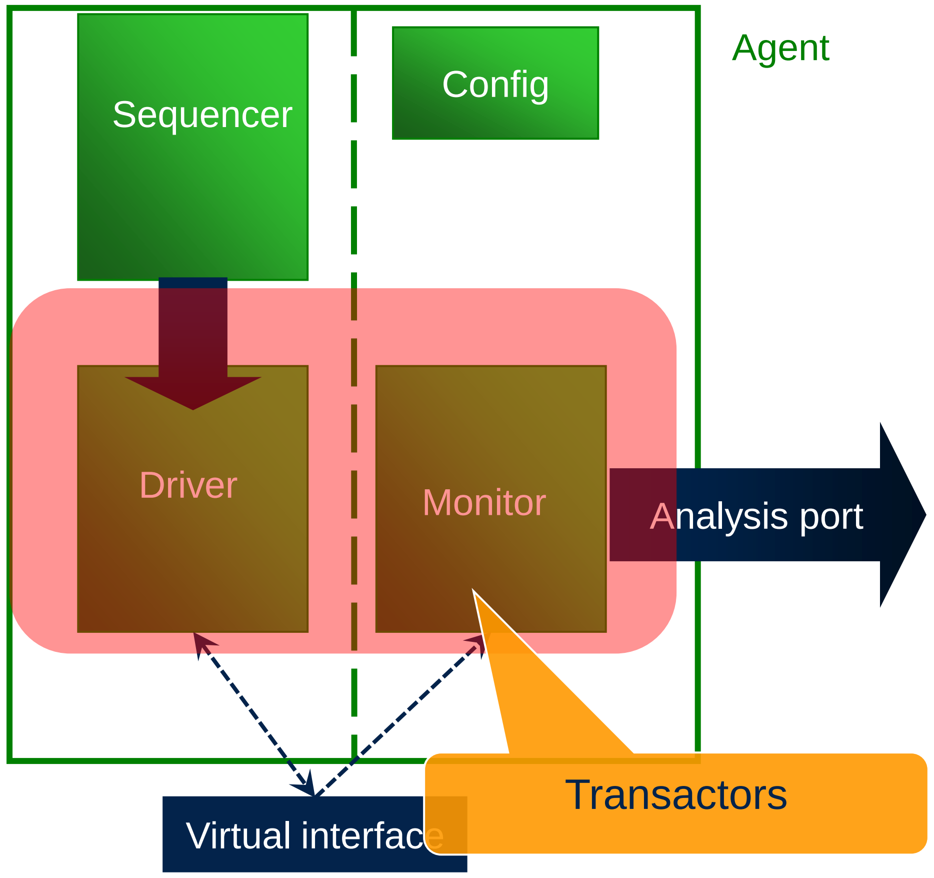
- Each Agent can be configured either as active or passive
- An active UVC is composed by an active section and a passive section
- A passive UVC is composed by a passive section only

UVM architecture



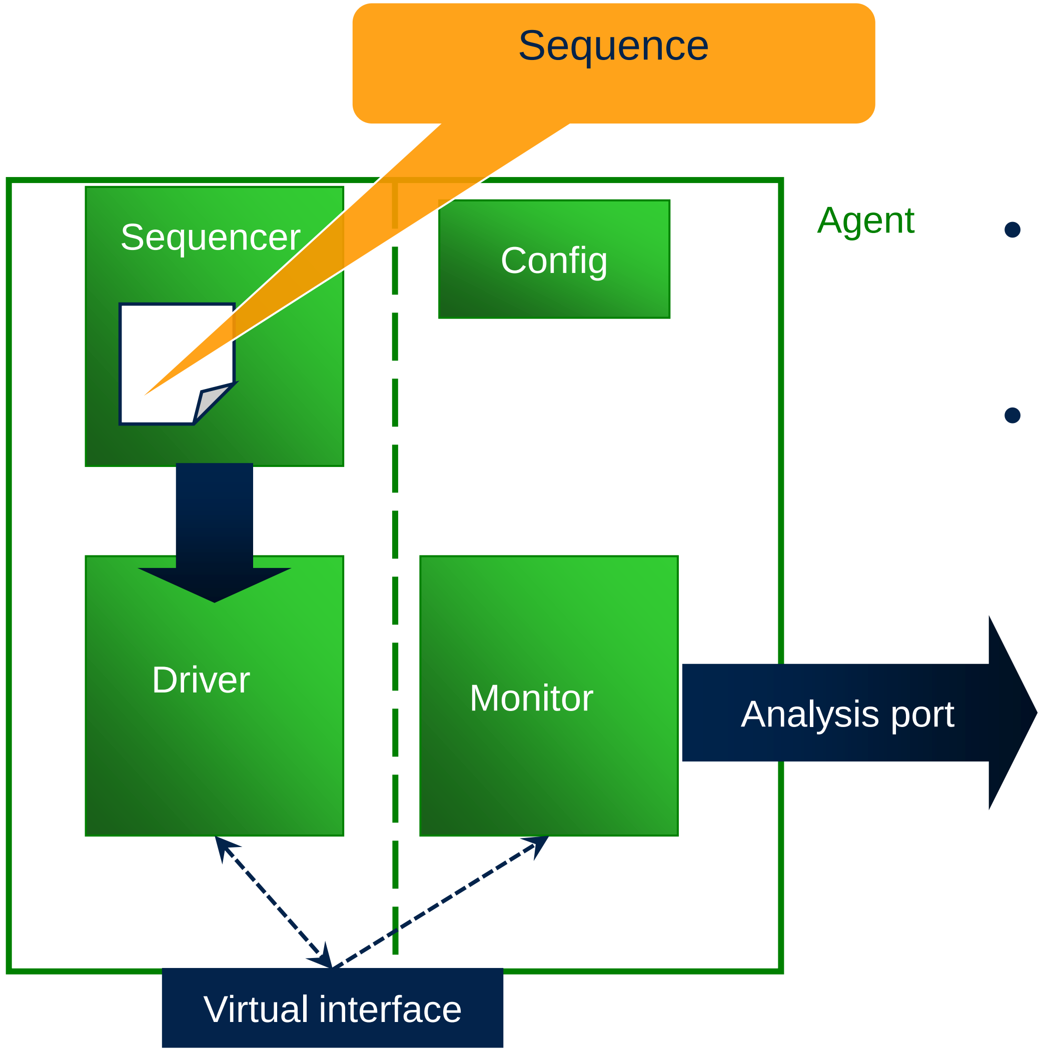
- Sequencer and driver are connected with a point-to-point TLM connection
- Monitor can be connected to external verification components through a point-to-multipoint TLM channel called analysis port
  - The analysis port can be left unconnected without worries if there are no components interested to the data from the monitor

UVM architecture



- UVM Drivers and UVM monitors act as transactors
- The driver converts TLM packets from sequencer into pin-level signals
- The monitor do the opposite converting pin-level signals into TLM packets flowing through an analysis port

UVM Sequences

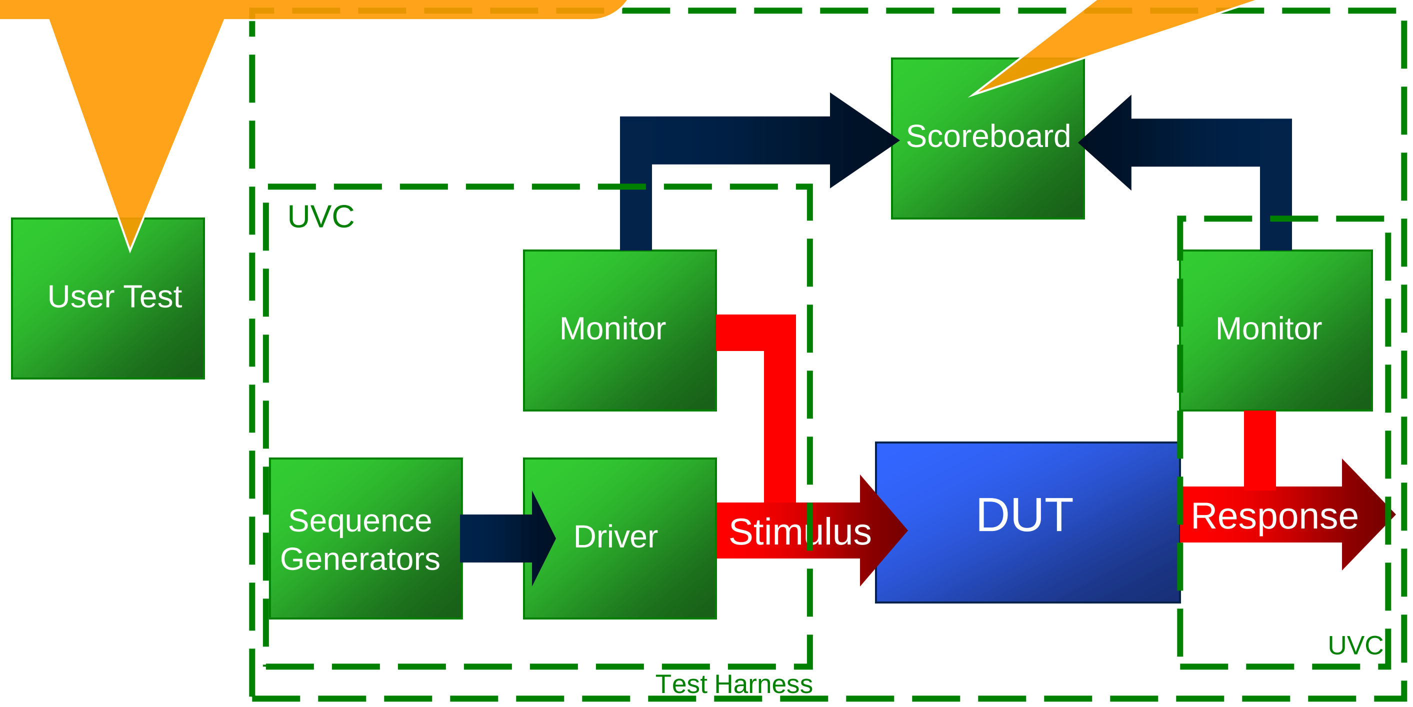


- A sequencer control the generation of random stimulus by executing sequences
- A sequence capture a meaningful stream of transactions
  - The simplest sequence is the atomic packet exchanged between sequencer and driver (UVM sequence item)

UVM Verification Environment

Used to configure:
- Layout of the environment
- Topology of the environment
- Sequence of transactions

Analysis component that checks the DUT behave correctly. Usually compares transactions from at least 2 agents



Verification Sign-Off Criteria

Verification Sign-Off Requirements

Verification signoff

1. To start with a comprehensive verification plan

2. Covering every functional requirement defined in design specifications, applicative use-cases, the architectural definition, and any other relevant documents

4. Tests are then developed to cover every feature of the verification plan

   -> Those tests are run and debugged (Running Regression)

5. To identified issues/bugs within the design

   -> This process iterates until the agreed level of coverage is met

life.augmented

Verification Sign-Off Criteria

Verification sign-off is granted if the following requirements are met:

•Code Coverage is 100% or Justified

•Functional Coverage is 100%

•All checkers written in the form assertions should get passed completely.

•Formal checks (Connectivity, Registers, unreachability etc..) passed or justified.

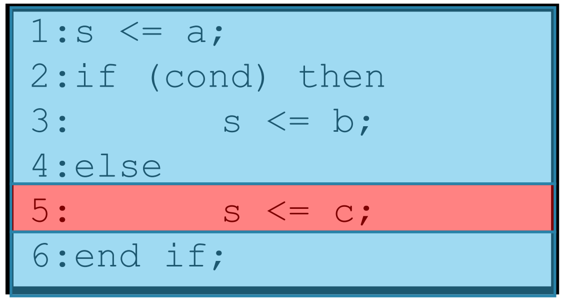
•Bus protocol checking pass.

life.augmented

Types of Coverage

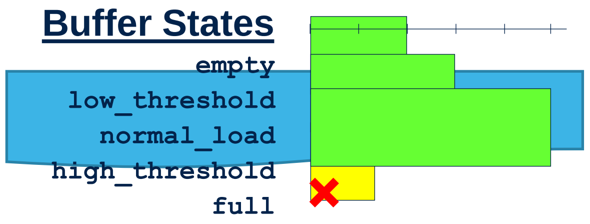
- Associated to verification closure
- Provide a measure of the verification completeness
- Should answer the questions
  - “Have I exercised all expected design states?”
  - Did we do all the tests that we wanted to do based on verification plan?
  - Did we try all combination of inputs/outputs?

- **Code coverage**
  - Automatically built coverage space
  - Typically: code coverage: did I exercise all the lines of code of my design ?



```
1:s <= a;
2:if (cond) then
3:        s <= b;
4:else
5:        s <= c;
6:end if;
```

- **Functional Coverage**
  - Requires engineering time to model coverage space
  - Typically: did I cover all the functionality, states, scenarios
  - Difficult to model, but easy to analyze
  - Mandatory with constrained random



Quality Metrics : Code Coverage

Code coverage -> Have all lines of RTL been exercised ?

Metrics used

- Block / Line / Statement Coverage
  - how many times each line is executed
  - Target 100% explained
- FSM arc / state / transition Coverage
  - All states are reached & all the possible state transition have happened
  - Easier to analyze than block + condition
  - Target 100%
- Expression Coverage
  - To exercise all expression combination
- Toggle Coverage
  - Which bits in the RTL has toggled
  - Used for connectivity correctness at integration level
  - Target 100%

life.augmented

Quality Metrics : Functional/Checker/Test Coverage

Functional coverage

- Define in verification plan, it list the functionality / state of design to be covered
- Both specification and implementation are handwritten (reuse is a must)
  - Specification : verification plan
  - Implementation : cover point in testbench

Checker coverage

- Assertion coverage → Does all my assertion has been activated?
- Checkers → Does all my checkers has been activated?

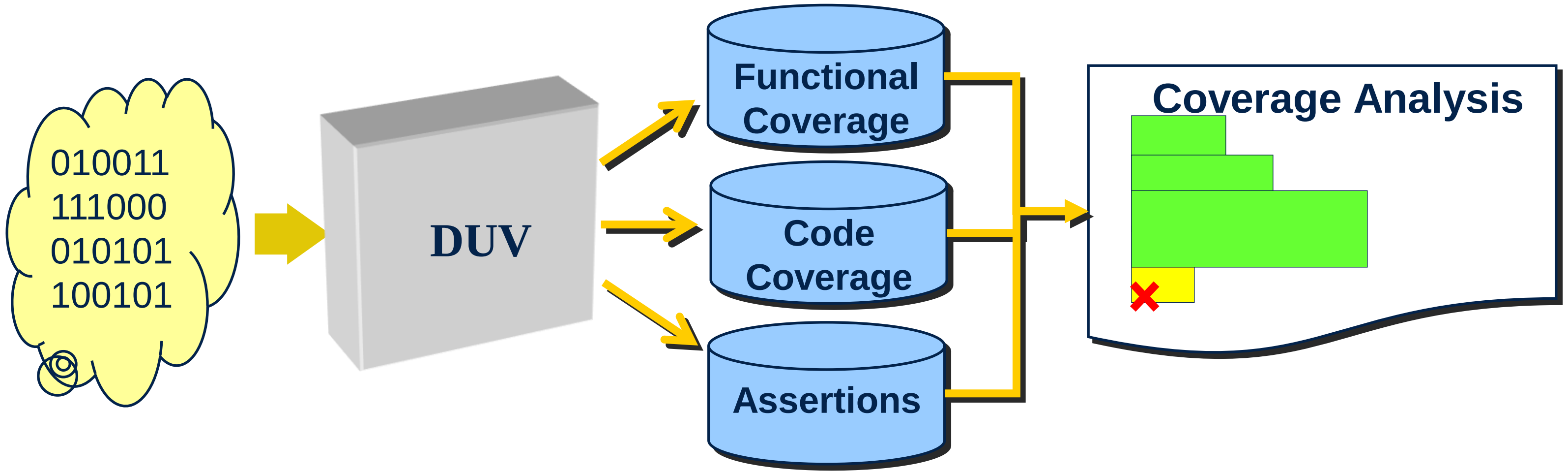
Test coverage

- Have all my tests been run?
- What is the status of each (Pass/Fail)?

life.augmented

Coverage Driven Approach

- Why write directed tests if you can have randomization do most of the work for you?
- But with Random – how do you know what you are testing?
- Coverage (various kinds) is the key metric
  - Verification plan defines coverage goals
  - Simulations produce coverage information
  - Cumulative “**total coverage**” analysis done iteratively to steer further work



Ideally, we are done when…

- **100% complete on test-bench work.**
  - No pending checkers, monitors, stimulus generation.
- **100% tests are run in regression on final build, no known failures.**
- **100% functional coverage:**
  - We have implemented all functional coverage points and we covered them, here is the report.
- **100% code coverage:**
  - We have done code coverage, no known coverage holes.
  - We can't do much about over-design in the library we are reusing.

- **100% coverage on corner cases**
  - We tried to break the design and we couldn't.
  - We tried known error scenarios and design recovers gracefully
- **100% coverage on real life scenarios (Applicative use cases):**
  - We have run "realistic" scenarios, no known failures.
  - We have run simulations with real FW, no known failures.
- **We are trending downwards and flat on bug finding cycle.**
- ***Let's ship it.***

Research areas in Verification

Research areas in Verification

Main research areas are :

- To shorten the whole Verification Completion Cycle
- Improve the Quality of Verification

Why there is a need for research:

- Since many years now, biggest challenge in verification techniques is to provide scalable solutions to find problem/bugs in the design
- This worsen with increasing complexity of the designs and thus results into larger number of bug escaping into silicon.
- That's why there is an urge and immediate need to research for better and faster techniques, like Automation, Mutation, or through Artificial intelligence/Machine learning

What can be done in research:

- Since hardware verification is very costly in terms of time and resources, verification engineers make huge efforts to reduce the whole verification project cycle time by increasing automation of this process
  - Artificial intelligence is an ideal candidate in these cases
- Machine learning in verification can also be used for multiple cases: generating complete and random range of stimuli needed to bring verified design into all its functional states; analyzing test results; filling coverage gaps and so on...
- To improve the quality of verification through Verification Qualification or Mutation

In conclusion:

AI/ML is often introduced into these systems to deal with such complexity and uncertainty!

Challenge:

Is to choose which parts of verification environment can be best suitable for automation and which verification techniques are most appropriate to provide to automatic engines.

life.augmented

Thank you

life.augmented