Multi-Variant Coverage: Effective Planning and Modelling

Vikas Sharma – Mentor, A Siemens Business

Manoj Manu – Mentor, A Siemens Business

Mentor®
A Siemens Business

accellera
SYSTEMS INITIATIVE

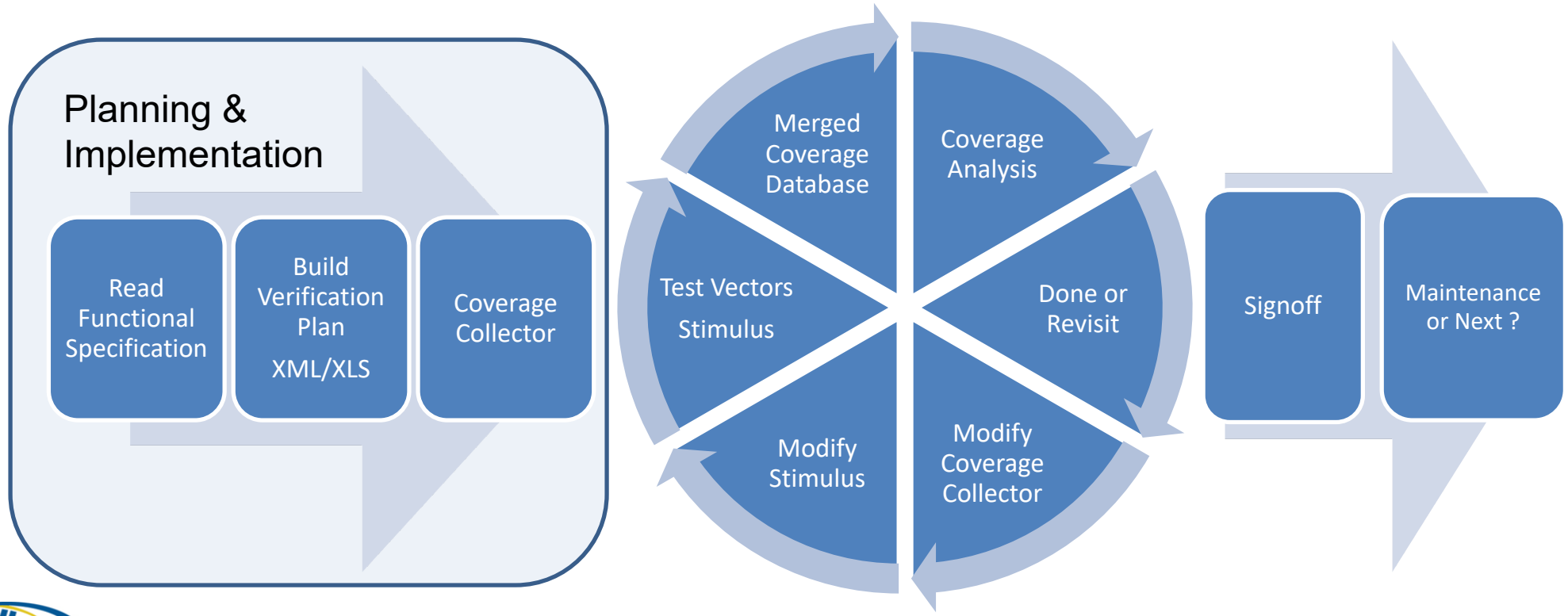
2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

- Introduction
- Motivation
- Application
- Challenges
- Recommendation
- Planning
- Implementation
- Maintenance
- Summary

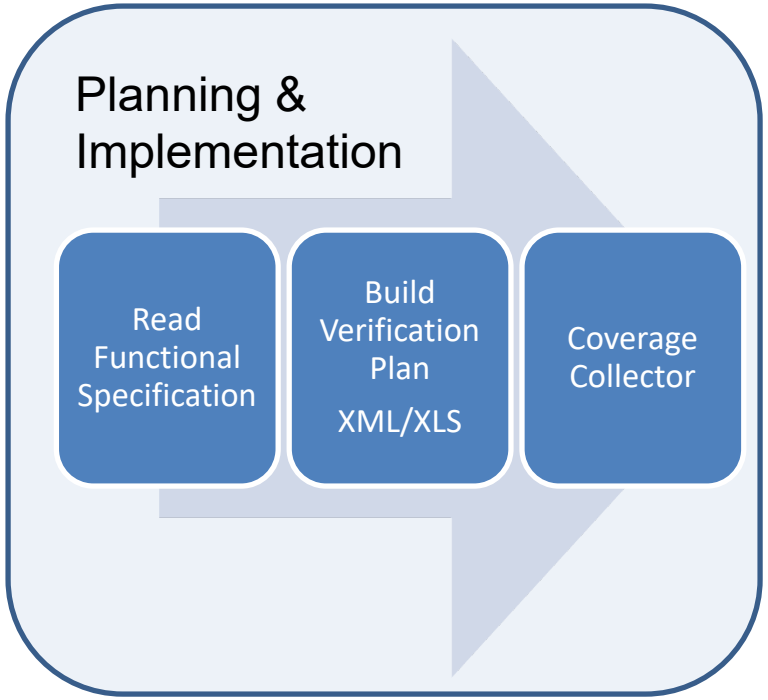
Introduction

- Traditional Coverage Driven Verification (CDV)
  - Involves verification planning and management



Motivation

- Simple Coverage Model
  - Tightly coupled with verification Intents
  - Rewrite happens when intent changes



- What happens when intended approach changes from compliance to exhaustive ?
- What happen when intended IP-version changes from version-1 to version-2 ?
- What happens when intended hierarchical context changes from block to SoC ?

Application

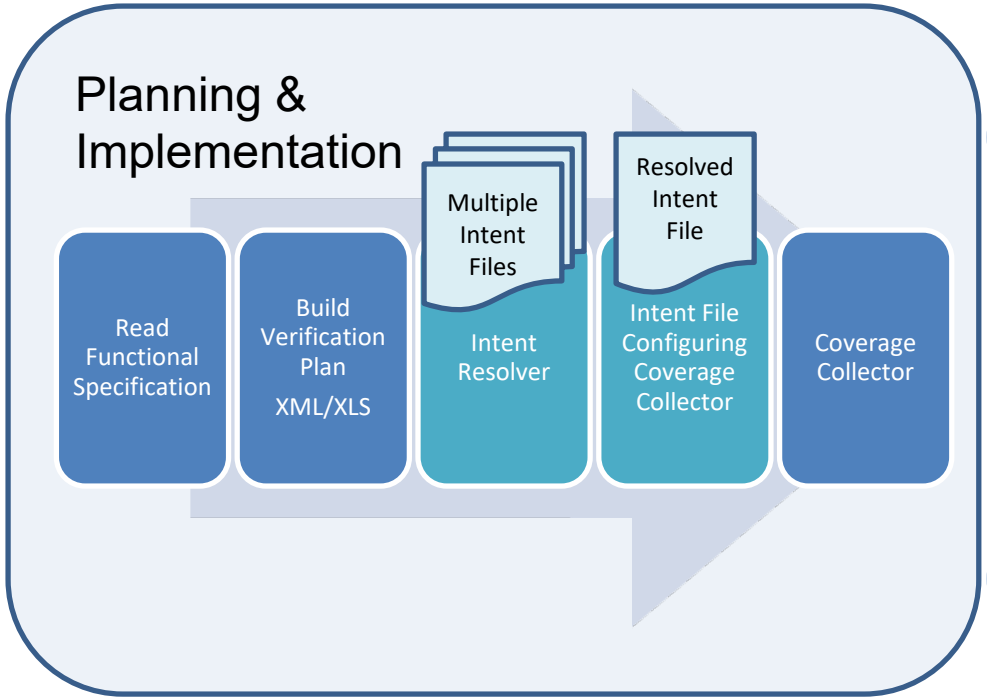
- Where DUT has multiple different verification goals targeting different verification intents
- The verification goals and intents may change due to variation in
  - Exhaustiveness of the verification (conformance, exhaustive, and application)
  - Design specification version (PHY-v1, PHY-v2, STACK-v1, and STACK-v2)
  - Application specific features (configuration read write, data streaming, DDR mode)
  - Hierarchical instantiation context (block-level and SoC-level)
- MIPI ecosystem of camera, display and PHY's
- Memory and Flash models with multiple vendors and part numbers

Challenges

- To configure desired intents seamlessly
- To add coverage and intents for new versions
  - Introduction of completely new functionality (Additional application layers)
  - To accommodate unforeseeable future extensions (Additional traffic classes)
- To provide coverage support for previous versions as well
- Traditional approaches doesn’t help
  - Multiple coverage models with one for each intent
  - Single coverage model with multiple compiler directives controlling intents
  - Verification planning tools with exclude options of undesired intents

Recommendation (Efficient Model)

- Multi-Variant Coverage Model
  - Intermediate layers are introduced to process and resolve intents
  - Intents may have logical relationships



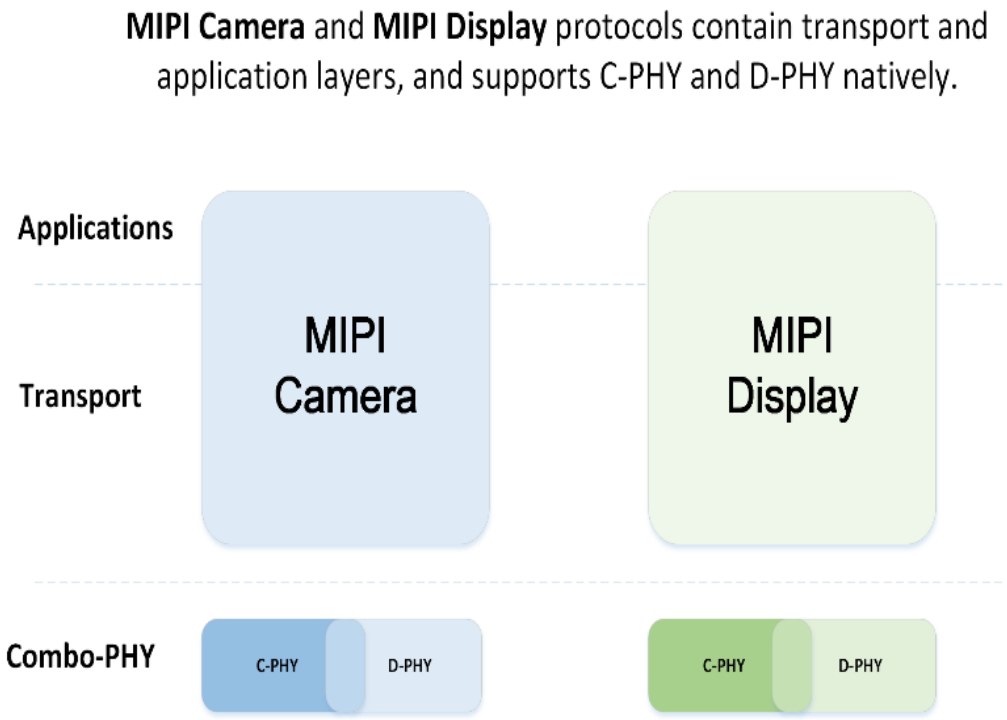
Multiple Intent files can be maintained for capturing different approaches

Multiple Intent files can be maintained for different IP-versions

Multiple Intent files can be maintained for block-level and SoC-level

Planning (Example MIPI)

- MIPI ecosystem of camera, display and PHY's
  - Includes a combination of protocols whose dynamics change quickly
  - Every layer has different versions allowing mix and match
  - Involves multiple coverage variants targeting different verification intents
  - Has more combinations of PHYs, protocols and their applications with varying versions
  - There can be many more verification intents of interest, depending on requirements

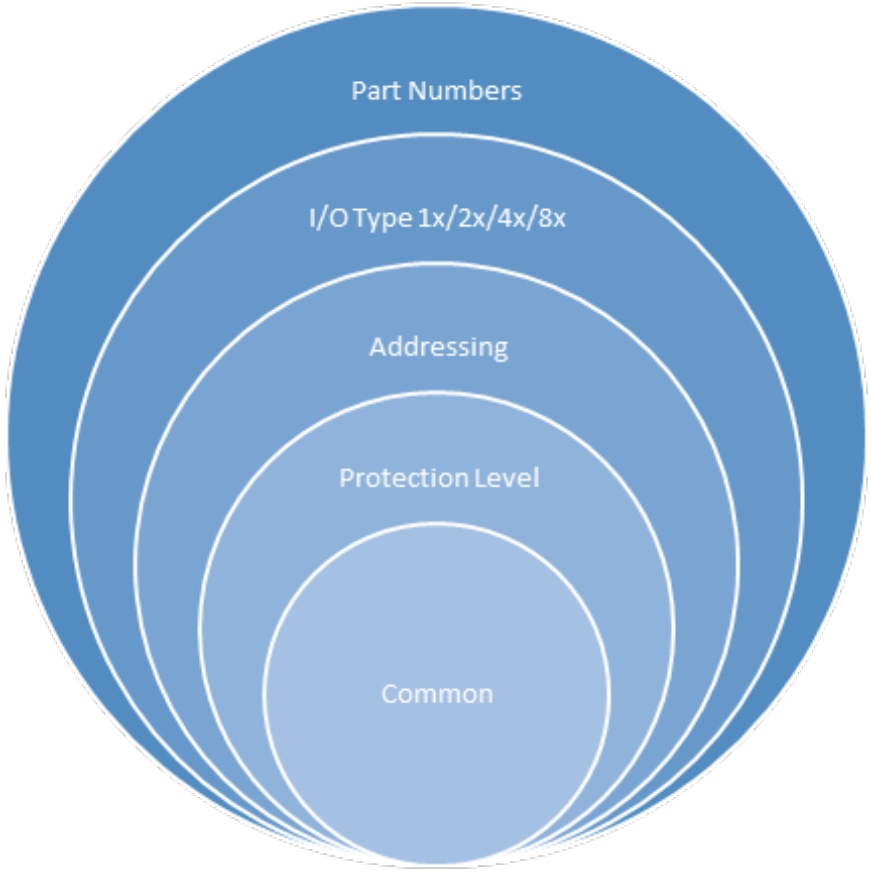


Planning (Report MIPI)

	Combo D-PHY	Combo C-PHY	Camera D-PHY	Camera C-PHY	Display D-PHY	Display C-PHY
Possible Intent 1	Version 1.3	Version 1.3	Exhaustive	Exhaustive	Exhaustive	Exhaustive
Possible Intent 2	Version 2.0	Version 2.0	Conformance	Conformance	Conformance	Conformance
Possible Intent 3	Low Power	Low Power	High Resolutions	High Resolutions	High Resolutions	High Resolutions
Possible Intent 4	Data Rates	Data Rates	SoC Level	SoC Level	SoC Level	SoC Level

Planning (Example FLASH)

- Memory and Flash models with multiple vendors and part numbers
  - Involves multiple vendors and part-numbers
    - Common in terms of erase/write/read procedures, data strobe, latency, timing information, refresh techniques, OTP techniques, and status and configuration registers
    - Differs in terms of memory density, data width, addressing mechanisms, protection levels, operating speed, I/O type, and applications
  - There can be many more verification intents of interest, depending on requirements

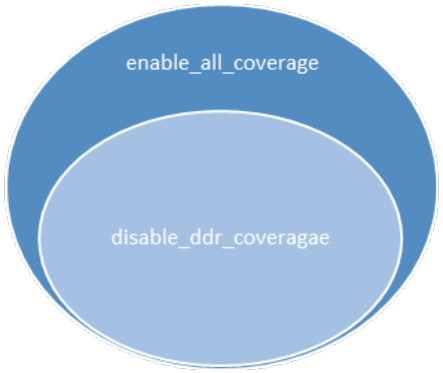


Planning (Report FLASH)

	Winbond	Micron	Macronix	Cypress
Possible Intent 1	Part W25Q64FW	Part MT25QU512BB	Part MX66L1G45G	Part S25FL512S
Possible Intent 2	Part W25Q128FW	Part MT25QU01GBBB	Part MX66L51245G	Part S70FL01GS
Possible Intent 3	Exhaustive	Exhaustive	Exhaustive	Exhaustive
Possible Intent 4	Basic Read/Write/Erase	Basic Read/Write/Erase	Basic Read/Write/Erase	Basic Read/Write/Erase
Possible Intent 5	Single/Dual/Quad/Octal	Single/Dual/Quad/Octal	Single/Dual/Quad/Octal	Single/Dual/Quad/Octal

Implementation

- Specify the intents carefully
  - Should be specified at higher abstraction
    - **<intent-file>** contains intent names as string type
    - **<intent-file>** file is parsed with system Verilog or scripting file handling mechanisms whichever fits perfectly
    - Parsed intents are saved in associative arrays of string type **<bit enable_cp[string]>**
  - Logical relations and expressions can also be specified by adding multiple intents in **<intent-file>** as Venn-diagrams
    - enable_all_coverage
    - disable_ddr_coverage




```
class multi_variant_coverage_model extends uvm_component;
 `define cvg_options(value) option.weight = (value)?1:0; option.goal = (value)?100:0;

 // Configuration handle
 m_config cfg;

 // Associative arrays for enabling and disabling intents
 protected bit enable_cp[string];
 protected bit disable_cp[string];
 . . .

 // Covergroup declaration with an optional list of arguments
 covergroup multi_variant_cvg (m_config cfg, int at_least, bit per_instance, string instance_name);

  // Coverage option settings for controlling the behavior of the covergroup, coverpoint, and cross.
  option.at_least = at_least;
  option.per_instance = per_instance;
  option.name = instance_name;

  // Coverpoint declarations
  coverpoint a1 {
    `cvg_options(enable_cp["enable_all_coverage"] || enable_cp["enable_all_a**"] || enable_cp["a1"])
     . . .
  }
  coverpoint a2 {
    `cvg_options(enable_cp["enable_all_coverage"] || enable_cp["enable_all_a**"] || enable_cp["a2"])
     . . .
  }
  coverpoint b1 {
    `cvg_options(enable_cp["enable_all_coverage"] || enable_cp["enable_all_b**"] || enable_cp["b1"])
     . . .
  }

  // Cross declarations
  a1_X_b1 : cross a1, b1 {
    `cvg_options(enable_cp["enable_all_coverage"] || enable_cp["enable_all_crosses"] || enable_cp["a1_X_b1"])
     . . .
  }
 endgroup
 . . .
endclass
```

Implementation

- Analyze and process intents
  - A simple **<intent-file>** contains
    - List of intents with simple logical relationship
    - List of coverpoint and cross names to be enabled
    - List of coverpoint and cross names to be disabled
  - A complex **<intent-file>** contains
    - List of intents with complex logical relationship
    - List of versions, feature list, and much more which might need processing prior parsing via. System Verilog
  - Use scripting languages for processing and converting them to a simple **<intent-file>**
  - configures goals and weights of enabled or disabled coverpoints and crosses using **`cvg_options**

```
covergroup multi_variant_cvg (m_config cfg, int at_least, bit per_instance, string instance_name);
  . . .

  // Coverpoint declarations
  coverpoint sdr1 {
    `cvg_options((disable_cp["disable_sdr_coverage"]) ? 0 :
                 (enable_cp["enable_all_coverage"] || enable_cp["enable_sdr_coverage"] || enable_cp["sdr1"]))
    . . .
  }
  coverpoint sdr2 {
    `cvg_options((disable_cp["disable_sdr_coverage"]) ? 0 :
                 (enable_cp["enable_all_coverage"] || enable_cp["enable_sdr_coverage"] || enable_cp["sdr2"]))
    . . .
  }
  coverpoint timer1 {
    `cvg_options((disable_cp["disable_timer_coverage"]) ? 0 :
                 (enable_cp["enable_all_coverage"] || enable_cp["enable_timer_coverage"] || enable_cp["timer1"]))
    . . .
  }
  coverpoint timer2 {
    `cvg_options((disable_cp["disable_timer_coverage"]) ? 0 :
                 (enable_cp["enable_all_coverage"] || enable_cp["enable_timer_coverage"] || enable_cp["timer2"]))
    . . .
  }
  coverpoint ddr1 {
    `cvg_options((disable_cp["disable_ddr_coverage"]) ? 0 :
                 (enable_cp["enable_all_coverage"] || enable_cp["enable_ddr_coverage"] || enable_cp["ddr1"]))
    . . .
  }
  coverpoint ddr2 {
    `cvg_options((disable_cp["disable_ddr_coverage"]) ? 0 :
                 (enable_cp["enable_all_coverage"] || enable_cp["enable_ddr_coverage"] || enable_cp["ddr2"]))
    . . .
  }
endgroup
```

Implementation

- Link **<intent-file>** with Verification Planner tools
  - All major EDA vendors provides verification planner tools for building test-plans (xml or spreadsheet)
  - User can change their verification goals which are not of their interests by passing **<intent-file>** in exclude options
  - Explore user manuals because different tool provides a different way to exclude by section and tag lists
  - Final verification test-plan shall consist of only intended coverage and merged to universal coverage database.
- IP-level
  - Coverage should be captured in one super covergroup
  - Covergroup may be controlled via. multiple **<intent-file>**
  - Must have a SoC **<intent-file>** targeting their SoC verification intentions only
- SoC-level
  - SoC should set global intent as SoC, and may contain a version number, and list of IP's to be included or excluded
  - When an included IP sees the global intent as SoC, then that IP should parse their corresponding SoC **<intent-file>**

Maintenance

- Key maintenance aspects
  - Maintenance of multi-variant coverage model lies in the ability to configure or change intents quickly and reliably
    - User-specified intents can be added very easily by making multiple intent-files, saved for future references
  - These arguments can be distinguished from other simulator arguments using the plus (+) character.
    - +ENABLE_INTENT_FILENAME=<~path/filename.enable>
    - +DISABLE_INTENT_FILENAME=<~path/filename.disable>

```
function new(m_config cfg = null);
  string  enable_intent_file_name;

  // The configuration handle should be checked against null
  . . .
  // enable_intent_file_name shall be provided either from configuration class or from command line option
  if ( ! ($value$plusargs("ENABLE_INTENT_FILENAME=%s", enable_intent_file_name))) begin
    enable_intent_file_name = cfg.enable_intent_file_name;
  end

  // Scanning disable-intent-file for filling up "enable_cp"
  if(enable_intent_file_name != "" ) begin
  . . .
  end

  // Scanning disable-intent-file for filling up "disable_cp"
  if(enable_intent_file_name != "" ) begin
  . . .
  end
endfunction
```

Summary

- Discussed how a simple coverage model can become a problem in an environment that involves varying verification intents
- Demonstrated how a well-captured verification plan can be converted to a working multi-variant coverage model that targets varying verification intents
- Multi-variant coverage models that employ these techniques are much easier to reuse and maintain than their counterparts.

Questions