2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Multi-Variant Coverage: Effective Planning and Modelling

Vikas Sharma, Mentor, A Siemens Business (*vikas_sharma@mentor.com*)

Manoj Manu, Mentor, A Siemens Business (*manoj_manu@mentor.com*)

***Abstract* — Effective functional coverage is not a trivial job, and the involvement of multiple coverage variants targeting different verification intents (multi-variant coverage) makes it even more challenging. Though multi-variant coverage modelling is a well-known problem that is often overlooked, it is critical in verification planning. To improve verification planning and modelling, the authors of this paper propose an efficient multi-variant coverage model that enables reuse and improves the upkeep quotients of coverage-driven verification. The proposed solution targets different verification requirements of the same DUT or IP with the following: varying intents (conformance, exhaustive, and application), varying hierarchical context (block-level and system-level), and varying design versions (PHY-1, PHY-2, STACK-1, and STACK-2).**

Keywords—SystemVerilog; Functional Coverage; Muli-Variant Coverage; Varyimg Verification

I. INTRODUCTION

Functional coverage is the metric for determining to what extent the functionality of a design specification, captured as features in a test plan, has been exercised. Usually, the verification team follows a coverage-driven verification (CDV) methodology that determines whether all the intended features of a design under test (DUT) are working. This traditional coverage modelling approach is good enough for cases where the DUT verification requirements and intents do not change for a long time ([1], [2], and [3]). However, it can require substantial development and maintenance efforts for cases where the DUT keeps getting frequent upgrades for supporting leading-edge technology. The coverage goals also change when verification requirements changes from ‘block-level’ to ‘SoC-level’, or from ‘sanity testing’ to ‘exhaustive testing’.

There are studies that talks about verification environment adaptation accommodating multiple versions of design IP ([4]). There are studies that talks about the accuracy and efficiency of various functional coverage modelling techniques of highly configurable design IP ([5]). However, there are other aspects of coverage that are usually overlooked in verification planning, such as controlling coverage modelling with multiple variants. It is important to understand the DUT requirements and scope out the possibilities of *“multi-variant coverage model”*.

This paper aims to address the most important aspects of a multi-variant coverage model and its applications. It describes how to plan, implement and maintain a multi-variant coverage model in SystemVerilog with examples that illustrate how the proposed model can solve multi-variant coverage problems using a singleton coverage model. This paper also demonstrates a configurable and scalable coverage model containing coverpoints and crosses, which can adapt to multiple design versions and varying verification intents that would otherwise require substantial effort.

II. MULTI-VARIANT COVERAGE APPLICATIONS

The multi-variant coverage modelling is critical for achieving coverage of a DUT that has different verification goals targeting different verification intents. Verification intents may change due to (1) variation in the exhaustiveness of the verification (conformance, exhaustive, and application), (2) variation in the design or DUT specification version (PHY-v1, PHY-v2, STACK-v1, and STACK-v2), (3) variation in the hierarchical context (block-level, system-level), or (4) variation in the DUT features as used by multiple applications (configuration read write, data streaming, DDR mode). To fully validate each and every version or combinations, such applications need a configurable coverage model with a combination of intents. At this stage, it is also important to plan multi-variant coverage model carefully because any mistake can cause a redo, which may further increase the time-to-market and the verification cycle cost of a DUT.

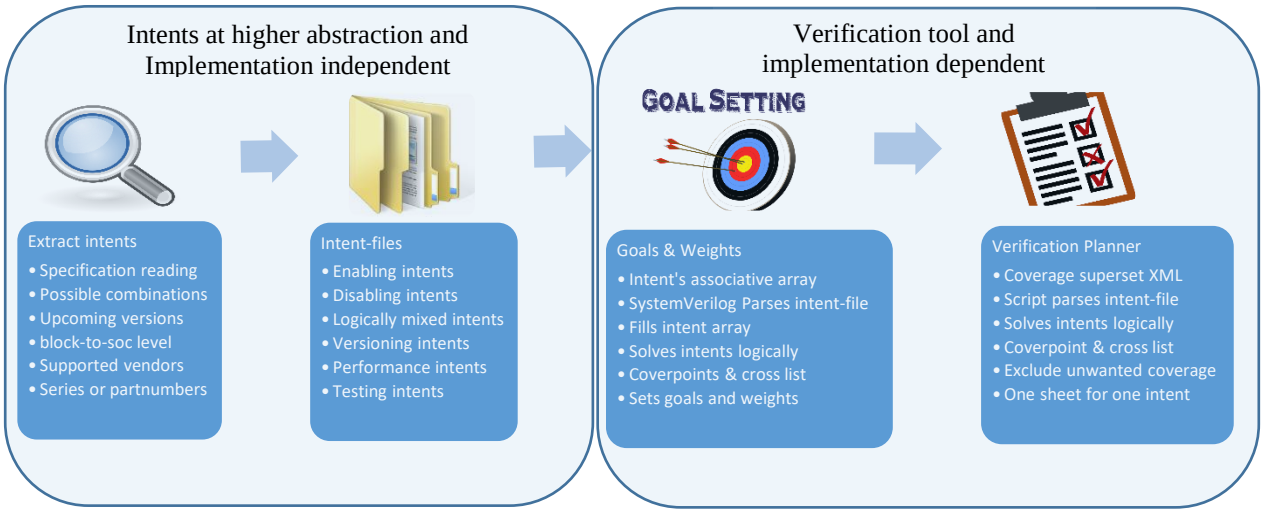
These applications may have multiple verification intents, where some intents are more intuitive and some intents are to be inferred. This paper helps IP and Verification-IP developers in scoping out the possibilities of multi-variant coverage model, identifying the underlying challenges and how to deal with multi-variant coverage modelling problems in order to successfully offer leading-edge products to their customers.

III. MULTI-VARIANT COVERAGE CHALLENGES

The multi-variant coverage modelling is not trivial as it is user-specified and more inclined towards different DUT requirements and verification intents. Additional challenges are introduced with the need to accommodate more upgrades with least redundancy, and to create robust APIs for selecting or enabling specific verification intents. The problem is often solved by different approaches as mentioned below. However, they don't solve the problem because of excessive compile options, increased redundancy and increased maintenance requirements.

- ***Multiple coverage models*** *: In this user develops multiple covergroups for targeting each verification intents. Although it seems like an easy option to begin with, but if model has bug fixes or improvements to make, user will end up making the changes multiple times, once for each model. This approach is also not recommended for cases where cross-combinations of features and intents exist.*
- ***Multiple conditional compilation compiler directives*** *: In this conditional compilation compiler directives are used to include specific lines or blocks of covergroups during compilation such as `define, `ifdef, `undef, and `include. This technique helps a bit in modelling seaparate coverage intents, but can be messy at times, and is not capable of scaling for unforeseeable future intents.*
- ***Verification planning tools with exclude options*** *: All three major EDA vendors (Mentor, Cadence and Synopsys) ([7],[8],[9]) provides verification planner tools for building testplans (xml or spreadsheet) which facilitates the verification process management. In this user can change their verification goals which are not of their interests by using exclude options. This technique may help in cases where user only wants to exclude by section and tag lists. It fails when user wants to exclude coverpoints and crosses by feature lists, or wants to enable only specific intent using logical relationships solved by using venn-diagram.*

The above approaches tends to crumble while developing multi-variant coverage model. So, instead of the traditional approaches, the choices must be influenced by combinations of IPs and intents, scalability, and ease of configuration and manageability. If above challenges are addressed systematically then it could contribute to achieving a more comprehensive approach as explained below. The remainder of this paper presents planning, implementation and maintenance concepts required to build a multi-variant coverage model, and finally the conclusion.



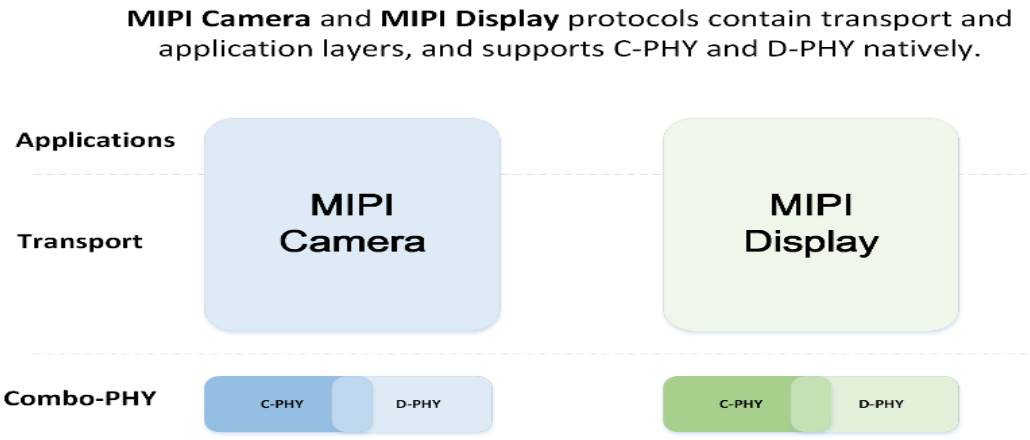
IV. MULTI-VARIANT COVERAGE PLANNING

When planning a multi-variant coverage model, it is important to understand the DUT requirements, keeping its historic plans in mind. It is important to scope out the multi-variant coverage model (MVC model) with different verification intents that a DUT can have and allow for foreseeable future enhancements. This paper includes some case studies that showcase how verification engineers can leverage the multi-variant coverage modelling techniques to accommodate varying intents (conformance, exhaustive and application), varying context (block-level and system-level), and varying part-number or versions (V-1, V-1.1, V-1.3, V-2).

A. *MIPI ecosystem of camera, display and PHY's*

A MIPI ecosystem includes a combination of protocols whose dynamics change quickly. For example, the MIPI D-PHY is the lowest layer of the high-speed source-synchronous interfaces for the connections of MIPI compliant camera (CSI) and display (DSI) applications to a host processor. Nevertheless, it can be applied to many other applications. With applications, it can also be a part of a complex Sub-System or SoC. It uses a 2-wires per-data lane. To further improve the throughput over bandwidth limited channel, a new C-PHY is introduced and is based on a 3-phase symbol encoding technology used in high-speed mode delivering higher data rate over 3-wire trios. C-PHY has many characteristics that are common to D-PHY, and many parts of C-PHY were adapted from D-PHY. For example, it reuses the existing 2-wires from D-PHY to make 3-wires and backward compatible.

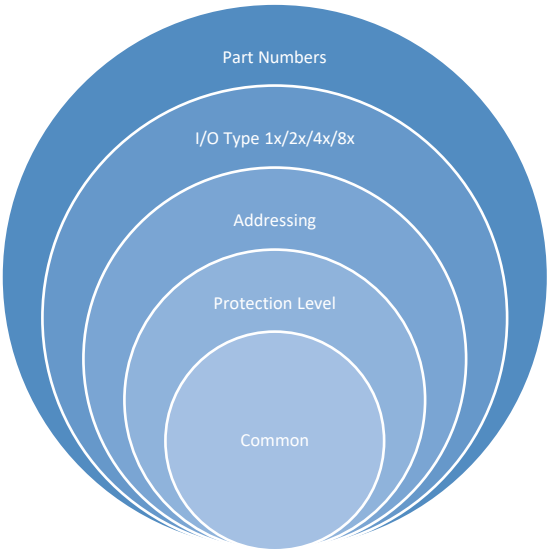
C-PHY is also designed to be able to coexist on the same IC pins as D-PHY, and this flexibility allows IP and VIP vendors to provide customers a low-cost and small-size C-PHY/ D-PHY combo entity in a single package. Combo-PHY involves multiple coverage variants targeting different verification intents because this has more combinations of PHYs, protocols and their applications with varying verions. To illustrate this, some verification intents are mentioned below. There can be many more verification intents of interest, depending on reqirements.



Combo D-PHY	Combo C-PHY	Camera D-PHY	Camera C-PHY	Display D-PHY	Display C-PHY
Version 1.3	*Version 1.3*	*Exhaustive*	*Exhaustive*	*Exhaustive*	*Exhaustive*
Version 2.0	*Version 2.0*	*Conformance*	*Conformance*	*Conformance*	*Conformance*
Low Power	*Low Power*	*High Resolutions*	*High Resolutions*	*High Resolutions*	*High Resolutions*
Data Rates	*Data Rates*	*SoC Level*	*SoC Level*	*SoC Level*	*SoC Level*

B. ***Memory and Flash models with multiple vendors and part numbers***

For Memory and Flash ecosystems, many vendors offer products with different part numbers that differ in terms of memory density, data width, addressing mechanisms, protection levels, operating speed, I/O type, and applications. They also share many common characteristics, such as erase/write/read procedures, data strobe, latency, timing information, refresh techniques, OTP techniques, and status and configuration registers.



For example, multiple Serial NOR Flash and Serial NAND Flash data sheets are offered by multiple vendors with multiple part numbers. However, they all uses the SPI protocol and shares many common characteristics. The common characteristics allows IP and VIP vendors to provide customers a low-cost singleton-entity which involves multiple coverage variants targeting different verification intents because this has more vendors with varying part numbers. To illustrate this, some verification intents are mentioned below. To illustrate this futher, some verification intents are mentioned below. There can be many more verification intents of interest, depending on reqirements.

Winbond	**Micron**	**Macronix**	**Cypress**
Part W25Q64FW	*Part MT25QU512BB*	*Part MX66L1G45G*	*Part S25FL512S*
Part W25Q128FW	*Part MT25QU01GBBB*	*Part MX66L51245G*	*Part S70FL01GS*
Exhaustive	*Exhaustive*	*Exhaustive*	*Exhaustive*
Basic Read/Write/Erase	*Basic Read/Write/Erase*	*Basic Read/Write/Erase*	*Basic Read/Write/Erase*
Single/Dual/Quad/Octal	*Single/Dual/Quad/Octal*	*Single/Dual/Quad/Octal*	*Single/Dual/Quad/Octal*

V. MULTI-VARIANT COVERAGE IMPLEMENTATION

The planning phase involves scoping out all the intents and creating a comprehensive plan. In order to implement all the intents, consider certain aspects, such as (1) what all APIs are required and at what abstraction level should these APIs be provided for implementing the intents , and (2) how to build a multi-variant coverage model that can easily be scaled up to accommodate any upcoming or impending verification intents. For example, in Memory and Flash models, there is always a scope for new vendors and part-numbers.

The coverage model must be implemented in a way that can accommodate such future enhancements gracefully. Otherwise, retrofitting or redoing efforts can further increase the time-to-market and verification cost of a DUT. A scalable singleton covergroup coverage model, where the intents of interest can be enabled from a higher abstraction level, and where the goals and weight for every coverpoint and cross can be set easily, assures correct results. The other important questions which would be explained below in details are “how and where to specify the intents” and “how to set goals and weights via intents”.

A. ***How and where to specify the intents***

APIs are needed for specifying high-level intents at relatively higher abstraction level from where the behavior is communicated and the implementation is encapsulated. To specify them in an intent-file is a better option because all the intents of interest can be captured in an intent-file and passed to coverage model through command-line arguments, such as $test$plusargs and $value$plusargs. The coverage model can easily parse through the intent-file by using *"file-handling mechanisms"*.

SystemVerilog provides system tasks and functions for file-based operations, such as $fopen and $fclose [6]. The intent-file can be passed to a covergroup for parsing using the $fgets and $sscanf operations, which further enables or disbles coverpoints and crosses.

```
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(enable_intent_file_name != "" ) begin
    integer fd;
    string coverpoint_name, fileline, cp_first_char;

    $display("Loading coverage Intent from File (%0s)", enable_intent_file_name);
    fd = $fopen(filename, "r");
    if(fd) begin
      while($fgets(fileline, fd)) begin
        // $sscanf is used to format the data as read from intent-file
        if($sscanf(fileline,"%s",coverpoint_name)) begin
          cp_first_char = coverpoint_name.substr(0,0);
          // Checking out that the first character read isn't '#' if found then considered as commented
          if(cp_first_char != "#")
            enable_cp[coverpoint_name] = 1'b1;
        end
      end
      $fclose(fd);
    end
    else
      $display("Failed to load coverage options from File (%0s)", enable_intent_file_name);
  end
endfunction
```

B. ***How to set goals and weights via intents***

The mechanism of setting coverage options of goals and weights can be used to manage large cluster of intents. They must be specified in a way that captures the verification intent accurately. Furthermore, from a coverage model reuse point of view, it is important that the coverage model is written with all the possible verification intents and configurations in an estimative approach to help during migration from one intent to another when needed. The value for the same option cannot be specified more than once within the same covergroup. Therefore, the values for options can be available before the instantiation of a covergroup.

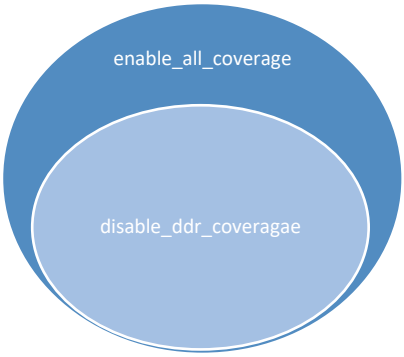
- ***option.weight :*** *This option specifies the weight of a coverpoint or cross for computing the instance coverage of the enclosing covergroup. The specified weight is a non-negative integral value.*
- ***option.goal*** *: This option specifies the target goal for a covergroup instance, coverpoint, or a cross of an instance.*

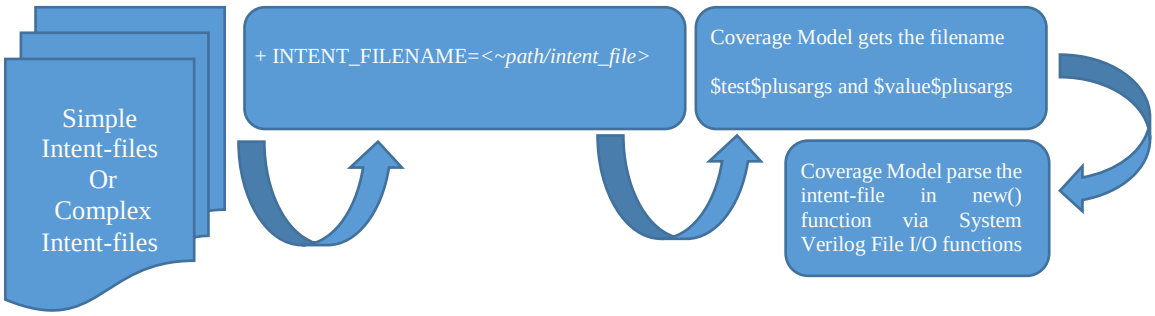
A lookup table can be implmented to map the goals and weights of all the intents, which can be scaled up when needed. This is accomplished using *"associative arrays"*,especially because such arrays can scale up and do not have any allocated storage until they are used. Moreover, the index expression can also be of string type, which helps in indexing intents as names.

```
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
```

The multi-variant coverage model is not restricted to simple high-level intents. It can also be a bit more logical where all possible logical relations between the collection of different intents are specified, such as Venn diagrams.

- ***enable_all_coverage*** *: This intent specifies that all supported features are of interest and the coverage of all coverpoints and crosses must be enabled.*
- ***disable_ddr_coverage*** *: This intents specifies that all DDR features are not of interest and the coverage of all DDR-related coverpoints and crosses must be disabled.*





In the above example, there are two contradictory logical intent in which all coverpoints are enabled by the first intent, but coverpoints related to DDR are disabled by the second intent. If logical relationships are much more complex, then solve them using scripting languages, which then generates an intent-file containg list of coverpoints to be enabled or disabled. The enabling and disabling mechanism can be implemented by using two associative arrays, one for enabling and the other for disabling intents. In order to keep it simple and more predictive, the list of disabled intents must be evaluated first. If not disabled, then it must be evaluated as prescribed by the list of the enabled intents because that is more realistic.

```
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
```

VI. MULTI-VARIANT COVERAGE LINKING WITH VERIFICATION PLANNER

The verification planning tool offers comprehensive approach to verification management, which allows all the relevant verification data to be stored in an extremely efficient format with open access. Typically, spreadsheet (XML/XLS/CSV) formats are used to build verification test-plan ([7], [8], and [9]). Every verification management tool provides several options (i.e. include and exclude options) to process the existing plan before merging with the universal coverage database. A super verification test-plan consisting superset of functional coverage shall be build and a subset of coverage which is not of interest can then be excluded.

- The intent-files that has been extracted from the planning phase can be scanned through scripts to extract out the list of invalid coverpoints and crosses to an exclude coverage file.
- This *"exclude coverage file"* can then be passed to verification planning tool with exclude options so that the exclude coverage can be excluded from the test-plan.
- The processed verification test-plan shall consist of only intended coverage and merged to universal coverage database.

VII. MULTI-VARIANT COVERAGE MAINTENANCE

The key maintenance issues are both managerial and technical. The multi-variant coverage models must be developed for making modification and updates to keep the coverage model up-to-date and usable over long period of time. The maintenance of multi-variant coverage model lies in the ability to configure or change intents quickly and reliably, and the proposed model certainly delivers the expected results. This approach allows verification engineers to reuse a coverage model to a scale as large as possible (including backward compatibility).

```
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
```

- User-specified intents can be added very easily by making multiple intent-files, saved for future references.
- The intent-file repository makes it easy to migrate from one entity to another.
- The user can build many more intent-files and save them at an appropriate path.
- The intent-file paths can be passed to coverage model in an optional argument provided to the simulation.
- 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*>

VIII. CONCLUSION

In this paper, the authors discussed how a simple coverage model can become a problem in an environment that involves varying verification intents. The task of scoping out and planning a multi-variant coverage model is often underestimated. By detailed planning, the authors have demonstrated how a well-captured verification plan can be converted to a working multi-variant coverage model that targets varying verification intents. All of the models presented in this paper have been inspired from real-world projects. The authors firmly believe that coverage models that employ these techniques are much easier to reuse and maintain than their counterparts. In addition, the proposed approach requires less verification resources and enables to focus more on value-added tasks.

IX. REFERENCES

[1] Verilab, *Navigating the Functional Coverage Black Hole: Be More Effective At Functional Coverage Modeling, DVCon 2015*
[2] Verilab, *Effective SystemVerilog Functional Coverage: design and coding recommendations, SNUG 2016*
[3] Paul Marriott and Steven Bailey, *Functional Coverage Using SystemVerilog, DVCon 2006*
[4] Verilab, *My Test bench Used to Break! Now it bends: Adapting to Changing Design Configurations, DVCon 2015*
[5] J. Ridgeway, K. Chaturvedula and K. Dhruv, Molding Functional Coverage for Highly Configurable IP, *DVCon 2016*
[6]ANSI / IEEE 1800-2012, *SystemVerilog–Unified Hardware Design, Specification, and Verification Language*
[7] Cadence Design Systems, Inc., "Incisive© Enterprise Manager Verification Planning," 2014.
[8] Mentor Graphics Corp., "Questa© SIM Verification Management User's Manual," 2014.
[9] Synopsys, Inc., "Verification Planner User's Guide," 2014