2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Towards a Hybrid Verification Environment for Signal Processing SoCs

Jan Hahlbeck, Steffen Löbel, Chandana G P,

NXP Semiconductors Germany GmbH,

Hamburg, Germany

(jan.hahlbeck@nxp.com, steffen.loebel@nxp.com, chandana.guddenahllipalaksha@nxp.com)

***Abstract*—This paper describes a hybrid verification approach for radio signal processing SoCs in the automotive domain. By providing tools and work flows for different methodologies in combination with an integrated regression setup we enable and encourage verification teams to mix functional and formal verification based on the stated verification task. By means of an example design the hybrid verification approach gets demonstrated.**

Keywords—Functional verification, Formal verification, Hybrid verification, UVM, MATLAB Simulink

I. Introduction

Verification in the field of automotive System-on-Chips (SoCs) is a challenging task in terms of different design aspects. We incorporate signal processing functionality, paired with high data rates and automotive compliant requirements. To cope with these verification challenges we are mixing different verification approaches and methodologies. Beside well-known and established Universal Verification Methodology (UVM) testbenches we are working with MATLAB Simulink reference models to verify signal processing algorithms. These models are transformed into C/C++ code which can be linked via Direct Programming Interface (DPI) into the testbenches [1]. Beside using UVM and MATLAB Simulink reference models we have extended our verification environment in the past years with formal verification based tools to benefit from the power of formal property checking and automated formal verification applications like structural connectivity checks, where the tool takes over the property generation. The recent Wilson Functional Verification Study [2] has shown that the adoption of formal techniques and automated formal verification significantly increased over the last years and gets more and more attention in the semiconductor industry. We can acknowledge this trend. Our goal is to enable verification engineers to decide based on the nature of a verification task to accomplish it either by formal techniques or functional simulation with a seamless transition between both worlds. Furthermore we want to convince and motivate verification engineers to try out formal techniques and build up trust for further methodology shift. Formal-friendly blocks are getting identified and selected based on the design characteristics and complexity like shown in [3]. We apply formal structural connectivity checks (CONN), formal property verification (FPV), and formal unreachability analysis (UNR) based on Jasper RTL apps [4]. This paper demonstrates how such a hybrid verification environment looks like for a signal processing subsystem which is part of a radio SoC in the automotive domain. The environment contains so called toolkits which provide code generation tools, common command scripts, quality checkers and a framework for regression runs including an interface to the requirement database. The underlying regression setup allows a coverage database merge of all functional and formal test runs.

II. Hybrid Verification Environment

A. Overview

Central point of our hybrid verification environment are so called toolkits which provide access to tools, code generators and command scripts for different methodologies with respect to the current phase of a verification cycle. A verification team can take advantage of a ready-to-use setup for the selected methodology. Regardless of the chosen approach, all verification artifacts are getting combined and merged in a common regression setup. Figure 1 shows which phases and activities are part of a full verification cycle starting from a Register Transfer Level (RTL) design delivery towards a fully verified verification release.

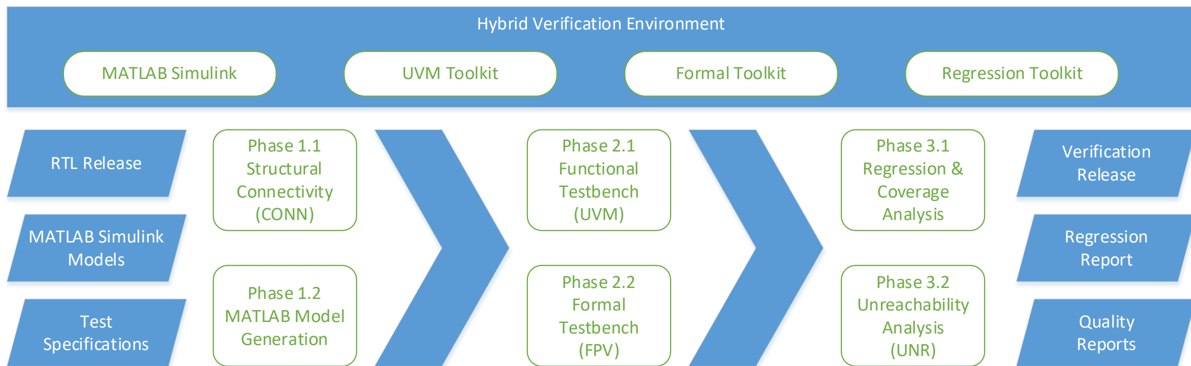

Figure 1: Overview of the hybrid verification environment and verification phases

There are three inputs into a verification cycle:
1. RTL release
2. MATLAB Simulink models for signal processing modules of the RTL
3. Test specifications with respect to the design requirements including verification goals

Based on these inputs a verification team goes through six different phases to achieve a verification sign-off which includes the following deliveries:
1. Verification release including all functional and formal testbenches
2. Regression report including mapped test specifications and coverage metrics
3. Quality reports e.g. a testbench linter report based on [5]

In the following sections we will briefly explain each phase and provide an exemplary design example to demonstrate how we apply the hybrid verification approach on a given RTL top level.

B. Phase 1.1 – Structural Connectivity

Structural connectivity checks belong to the family of automated formal verification tools and we apply them to ensure that we have a fully connected top level RTL. Figure 2 shows our connectivity flow. We apply two flavors: Reverse connectivity to create a golden reference connectivity map and regular connectivity to check whether a RTL update still contains all connections. New design deliveries can be checked within a short period of time and missing or modified connections can be added by the help of the reverse connectivity feature. The Formal Toolkit provides a code generator to create an initial connectivity environment, command scripts, regression hooks and reporting mechanisms. Connectivity targets are provided as part of the test specifications. Usually we apply CONN on the first hierarchy level of the top level RTL to ensure that all integrated blocks are assembled correctly. Furthermore this guarantees the data integrity when multiple UVM block level scoreboards are concatenated or MATLAB reference models are used to verify signal processing functionality.

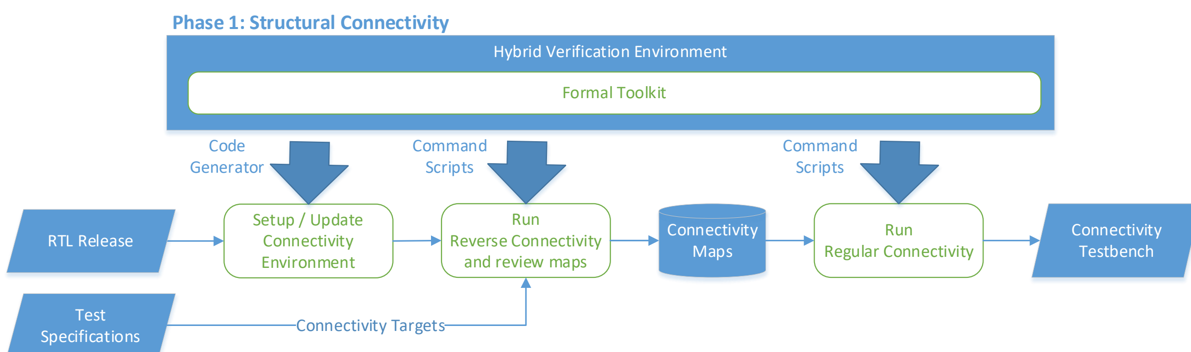

Figure 2: Structural connectivity flow

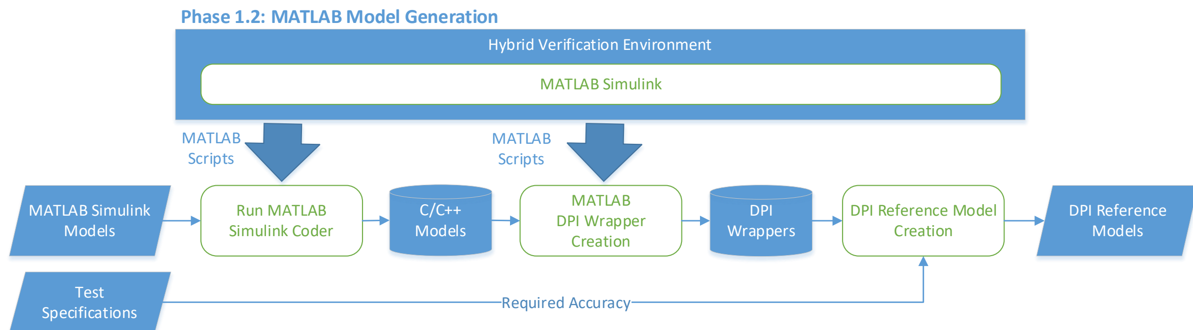

Figure 3: MATLAB Simulink model generation flow

C. Phase 1.2 – MATLAB Model Generation

Signal processing modules are developed with the help of MATLAB and MATLAB Simulink. To ensure that the RTL matches the expected behaviour we generate System Verilog DPI verification modules in three steps like depicted in Figure 3. Firstly we generate C/C++ reference models using the MATLAB Simulink Coder. In a second step these functions are integrated into DPI wrappers. And finally these wrappers are instantiated in a reference model which might contain one or more DPI wrappers plus additional data checks of observed and expected data. The resulting reference models are instantiated in the functional testbench in parallel to the UVM environment. The final pass or fail criteria of a functional UVM test considers both, the UVM status as well as internal error counters of all existing reference models.

D. Phase 2.1 – Functional Verification

The functional verification takes place with a well-known UVM testbench and a mix of constrained random tests and directed tests. Scoreboards ensure the data integrity and coverage collectors enable metric driven verification. Standard protocol interfaces like AMBA AHB, APB or AXI are verified by using Verification IPs (VIPs). Beside the UVM testbench we implement and maintain System Verilog Assertions (SVAs) for dedicated verification tasks like clock period or duty cycle checks. Properties which are identified to be re-usable for FPV can be shared with the formal verification team. Figure 4 shows the functional verification flow where a verification team creates a UVM testbench which gets handed over to the regression setup. The UVM toolkit provides testbench generators and a testbench linting tool including linter rules to ensure that all UVM components, sequences and tests are aligned with our internal coding guidelines.

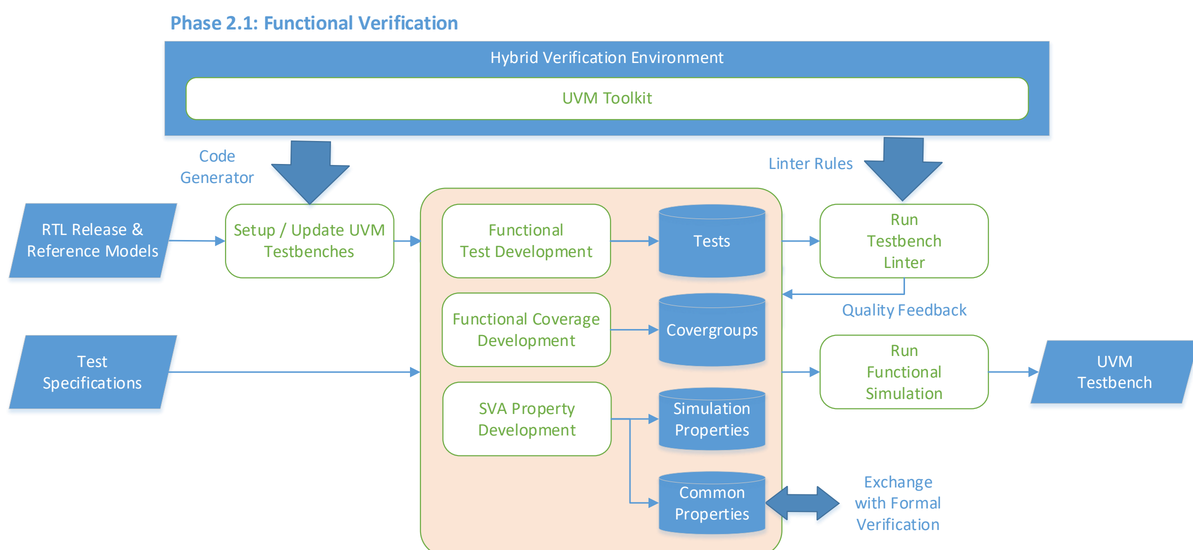

Figure 4: Functional verification flow for UVM testbenches

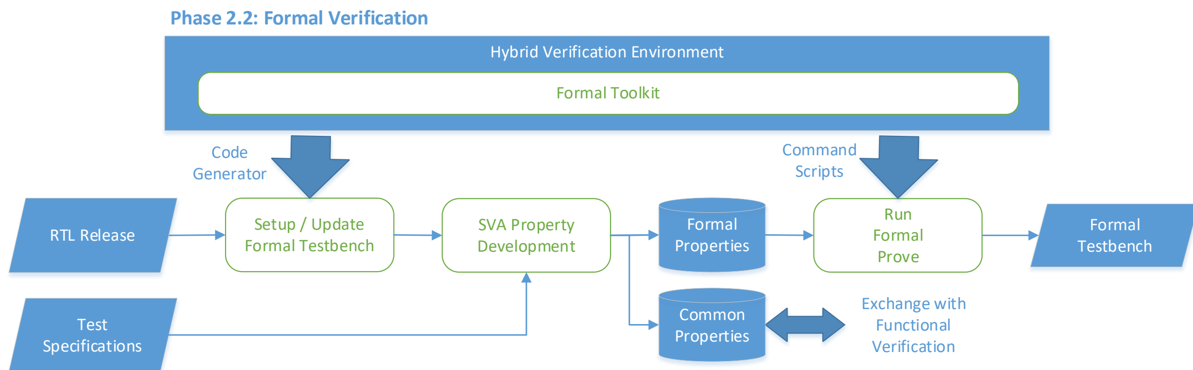

Figure 5: Formal property verification flow

E. Phase 2.2 – Formal Property Verification

Beside automated formal verification we enable verification engineers to use formal property verification (FPV) for verification tasks where functional verification cannot be done exhaustively at a manageable effort. Additionally there is a high demand to use FPV for bug hunting, isolation and exploration of already discovered bugs, especially when there are doubts and findings in simulation based tests. Formal testbenches may include Assertion-based VIPs in case there are standard protocol interfaces. Properties are getting implemented by using SVAs. Figure 5 depicts the formal verification flow for formal testbenches. The Formal Toolkit provides command scripts, regression hooks, reporting mechanisms and code generators to generate initial skeletons of formal testbenches. We provide a possibility the exchange properties with the functional verification team by deploying them into separate interfaces. For properties which are marked for dual use – in functional simulation and FPV - we avoid liveness properties, usage of free variables and extended use of cover properties to keep them simulation friendly like recommended by [6].

F. Phase 3.1 – Regression Setup and Coverage Analysis

The regression setup is directly linked to the requirements tooling and contains a full list of test specifications for the given design release. The setup is capable to execute both functional and formal test runs and merge the results into a single coverage database. Figure 6 shows that formal and functional tests are executed in two separated regression steps as different testbench hierarchies require a post-processing to map the FPV results into the UVM testbench hierarchy. A so called verification plan is used to map the regression results on the test specifications. Test specifications can be fulfilled either by directed tests, constrained random tests, structural coverage, functional coverage, assertions, or formal test runs like CONN or FPV.

G. Phase 3.2 – Unreachability Analysis

To support the structural coverage analysis and closure after a regression run we use automated formal unreachability (UNR) analysis to detect unreachable parts of the design. This allows verification teams to focus on the remaining reachable coverage and provides valuable feedback to design engineers in case a finding is caused by a faulty design description rather than by design parameterization. After a manual code review all unreachable parts of the design are getting excluded by using a refinement file which gets passed to the coverage analysis like shown in Figure 6. To mitigate the effort of UNR in terms of RTL review and extensive runtimes we follow three basic rules:

1. First UNR run only when at least 70% coverage gets reached by simulation based regression
2. Re-run of UNR only for major design updates to reduce the review effort
3. UNR is allowed to start at uninitialized state to optimize state space exploration, with an accepted risk of having false negatives

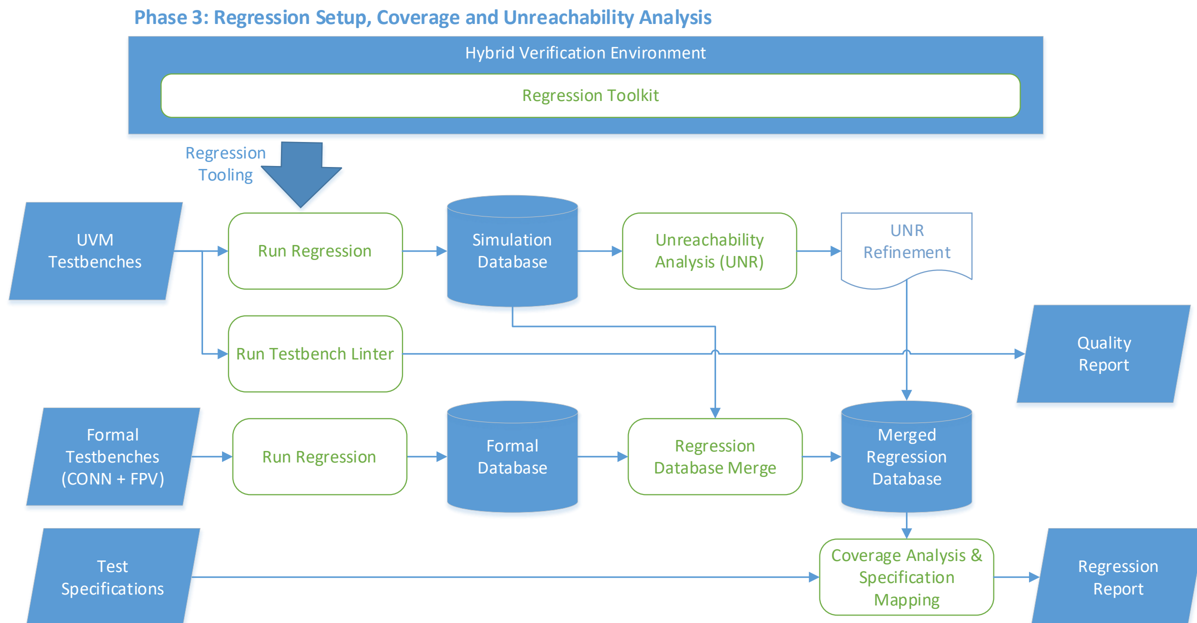

Figure 6: Regression setup, coverage and unreachability analysis flow

III. APPLYING HYBRID VERIFICATION

In this section we will give an example how we apply the hybrid verification flow for a common design in the radio frequency domain. The given design under test (DUT) contains the following modules:

- AXI2APB bridge to access control registers of every module
- Three wideband filters with corresponding MATLAB Simulink models
- Three smallband filters with corresponding MATLAB Simulink models
- Cross-bar to allow dynamic switching from any wideband filter to any smallband filter
- Data Packetizer which encapsulates the filter output into a dedicated streaming protocol
- Write Direct Memory Access (WDMA) engine which allows configurable memory access via standard protocols
- Synchronization unit to trigger filter operations and DMA operations
- Interrupt distribution unit to collect and filter all subsystem interrupt lines

Based on the design characteristics, test specifications, estimated effort and team experience we select suitable verification approaches for each module. Figure 7 shows the architecture overview plus the partitioning of the subsystem verification into different methodologies. The signal processing functionality represented by the wideband and smallband filters gets verified by using bit-accurate MATLAB reference models. Additionally a small portion of critical logic in the wideband filters gets checked by a FPV testbench to proof the absence of possible deadlocks. Modules like the AXI2APB bridge and the Write DMA are getting verified purely by functional verification using directed and constrained randomized UVM tests in combination with scoreboards and functional coverage collectors which are attached to passive UVM agents. The Filter Crossbar and the Interrupt Distribution module are getting fully verified using FPV to handle the number of possible combinations and data range variety without the need to create all possible stimuli in a functional testbench. For the Data Packetizer and the Synchronization Unit we have a hybrid verification approach where critical parts of the design are verified by using FPV and the remaining parts by using UVM. In total we end up with a single UVM testbench, one Connectivity testbench and five FPV testbenches. In addition we generate two MATLAB reference models which are instantiated three times each.

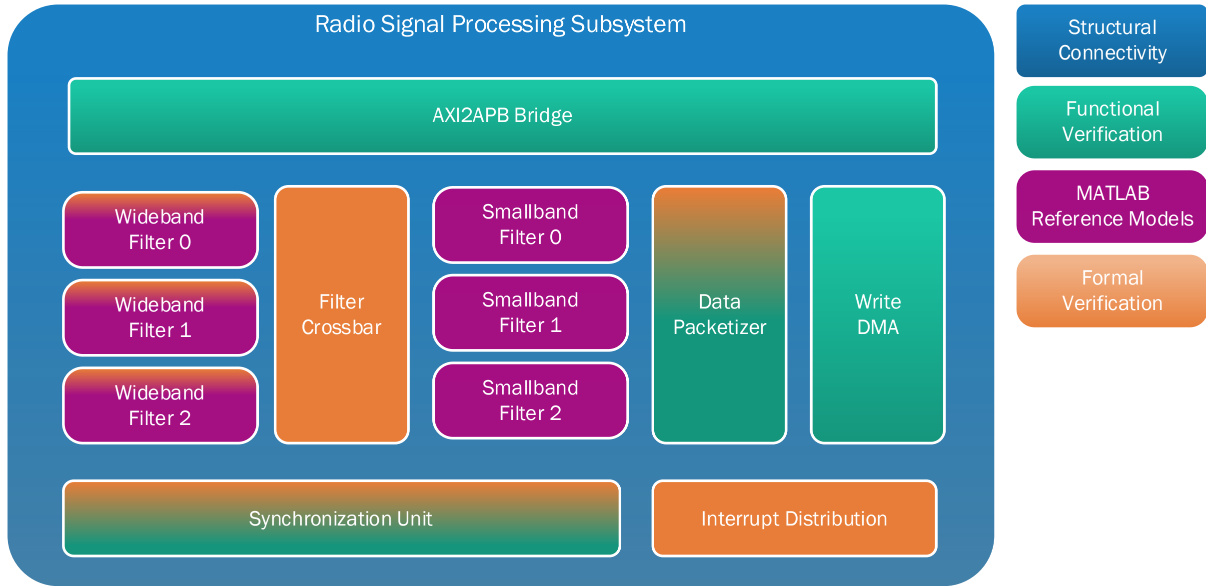

Figure 7: Exemplary radio signal processing subsystem with data flow from left to right

A. *Coherent Data Checks*

Zooming into one possible data path from the design inputs to the design outputs demonstrates the strength of the hybrid verification approach. Figure 8 shows the path from Wideband Filter 0 through the Filter Crossbar to Smallband Filter 0. The filtered data get picked up by the Data Packetizer and finally transferred to the Write DMA module which drives the design outputs towards a memory interfaces. The complete data path is either covered using MATLAB DPI models, UVM scoreboards or a full FPV proof to ensure the data integrity throughout. Formal connectivity checks ensure that all output probes are directly connected with the input probes of the next checker e.g. the MATLAB model of Smallband Filter 0 is connected to the filter data output which gets used as data input of the Data Packetizer UVM scoreboard. By using this approach we can re-use already existing block level scoreboards without any modifications or the need to create additional scoreboards.

B. *Shared Properties*

In case there are common properties which shall be shared between functional simulation and FPV run we encapsulate them in System Verilog interfaces, which can be instantiated in the UVM testbench top level as well as in the FPV testbench like depicted in Figure 9. To avoid runtime penalties of SVAs on the simulation we keep this number limited.

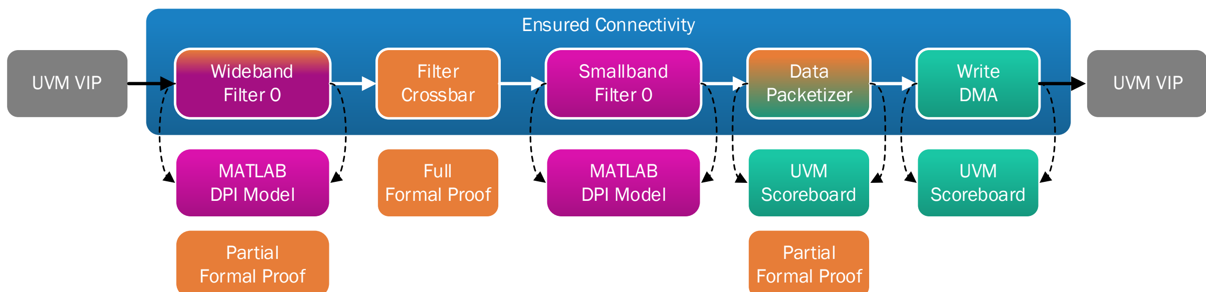

Figure 8: Coherent data check flow

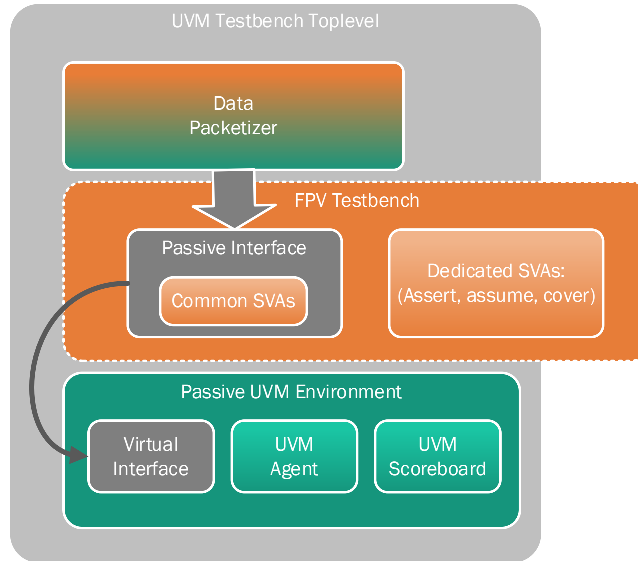

Figure 9: Property sharing between functional and formal verification

C. MATLAB DPI Wrapper

The MATLAB Simulink Coder provides the same API for every module which requires a reference model. The API consists of five functions: initialize, terminate, reset, output and update. Figure 10 shows how these five function calls are integrated in a DPI wrapper module for a single Wideband Filter. An additional delay stage gets inserted to allow cycle and bit accurate comparison with the DUT. In case a signal processing module contains control registers the number of inputs will increase according to the number of controllable registers. Data checkers and error counters are added in another wrapper level on top of the shown DPI wrapper.

```
module WidebandFilter_dpi_wrapper(
  input bit clk, reset,
  input logic signed [31:0] In_re, In_im,
  output logic signed [31:0] Out_re, Out_im
);
  chandle objhandle=null;
  logic signed [31:0] Out_re_temp, Out_im_temp;

  initial objhandle = DPI_WidebandFilter_initialize(objhandle);
  final DPI_WidebandFilter_terminate(objhandle);

  always @(posedge clk or posedge reset) begin
    if(reset == 1'b1) begin
      objhandle=DPI_WidebandFilter_reset(objhandle, In_re, In_im, Out_re_temp, Out_im_temp);
      Out_re <= Out_re_temp;
      Out_im <= Out_im_temp;
    end else begin
      DPI_WidebandFilter_output(objhandle, In_re, In_im, Out_re_temp, Out_im_temp);
      DPI_WidebandFilter_update(objhandle, In_re, In_im);
      Out_re <= Out_re_temp; // Delay stage to be in sync with DUT
      Out_im <= Out_im_temp; // Delay stage to be in sync with DUT
     end
  end
endmodule
```
Figure 10: MATLAB DPI wrapper example for a Wideband Filter

D. Regression Coverage Merge and Unreachability Analysis Results

Table 1 shows exemplary coverage measurements for some of the given modules to show how a formal proof gets used to boost the coverage closure based on block, statement, expression and toggle coverage. The values are based on a status in the late project phase. The Filter Crossbar for instance reaches 94% coverage in functional simulation whereas the FPV testbench easily reaches 100%. By merging the coverage the total coverage indicates that we do not need to implement more functional tests. Even for partial proves we can observe an increase of 23% for the Data Packetizer and 22 % for the Synchronization Unit.

Table 1: Regression coverage merge results of functional and formal simulation (pre-UNR)

Module	Properties	Sim Coverage [%]	Formal Coverage [%]	Merged Coverage [%]
Wideband Filter	2	75	1.1	75
Filter Crossbar	106	94	100	100
Data Packetizer	150	65	75	88
Interrupt Distribution	66	95	100	100
Synchronization Unit	50	73	90	95

The unreachable analysis for the given example design takes around 15 minutes and marks 6% of the entire RTL as unreachable. This fits the expectation when using highly parameterized signals processing modules. Table 2 shows exemplary UNR numbers for selected modules. A highly generic filter module contains up to 7% unreachable code parts whereas the findings for a highly optimized and dedicated module like the Data Packetizer is zero. A partly generic module like the Write DMA fits in between with 2% unreachable code parts.

Table 2: Unreachability analysis results

Module	Coverage pre-UNR [%]	Coverage post-UNR [%]	UNR [%]
DUT Top level	76	82	6
Wideband Filter	75	82	7
Smallband Filter	84	87	3
Data Packetizer	88	88	0
Write DMA	93	95	2

IV. SUMMARY

The presented hybrid verification environment enables our verification teams to take advantage of different verification methodologies and approaches in a single verification cycle. The new degree of freedom encourages verification engineers to choose the best possible approach for a given tasks without being limited in the way of thinking. In six phases we generate MATLAB reference models, apply functional and formal verification tools, and combine everything in a regression setup. An additional UNR run greatly supports the coverage closure. In the future we plan to evaluate more automated formal verification tools and strengthen our FPV knowledge to be ready for a further methodology shift.

V. REFERENCES

[1] Neal Okumura, Glenn Richards, "Co-Simulating Matlab/Simulink Models in a UVM Environment," in *DVCon US*, 2015.

[2] H. Foster, "Functional Verification Study," Wilson Research Group, 2022.

[3] Keerthikumara Devarajegowda, Jeroen Vliegen, Goran Petrovity, Kawe Fotouh, "A Mutually-Exclusive Deployment of Formal and Simulation Techniques Using Proof-Core Analysis," in *DVCon Europe*, 2017.

[4] Cadence, "Jasper RTL Apps," [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform.html. [Accessed 20 August 2023].

[5] AMIQ EDA, "Verissimo SystemVerilog Linter," [Online]. Available: https://dvteclipse.com/products/verissimo-linter. [Accessed 20 August 2023].

[6] Erik Seligman, Tom Schubert, M.V. Achutha Kiran Kumar, Formal Verification: An Essential Toolkit for Modern VLSI Design, Morgan Kaufmann, 2023.