Accelerating the IP Design Cycle with Formal Techniques Beyond Everyday FPV

Bhushan Parikh, Shaman Narayana, Buck Lem, David Cassetti
Intel Corporation
5000 W. Chandler Blvd,
Chandler, AZ 85226

***Abstract* - Traditional shift-left strategies for accelerating IP design convergence aim to push verification and validation activities as early as possible in the IP development life cycle in order to shorten the total development time. In addition, late-breaking design fixes need timely and thorough validation to mitigate the risk to quality and schedule. Formal techniques such as Formal Property Verification (FPV) play an increasingly important role in all phases of design validation. However, there are other formal techniques in addition to FPV that can optimize and shorten the IP development life cycle even further. This paper discusses the application of such powerful formal-based techniques used in a HW based Compression IP development, which improved RTL quality for early design hand-off to Pre-Silicon (Pre-Si) Verification (PSV) team, accelerated key design phases (Tech-readiness, Execution and Engineering Change Order (ECO)) and resulted in delivering tapeout-quality IP 12 weeks ahead of schedule.**

I. Introduction

In this highly competitive era, IP development teams must deliver tapeout-quality designs to aggressive schedules. Traditional constraint-based random verification methodology comprises 50% - 70% of the overall development time [1]. Due to this, PSV activities have become a major challenge and the critical path to tapeout in recent projects. As a result, design and verification engineers are always looking for ideas to efficiently reduce the time required for verification, without compromising the quality of verification [2].

In this regard, developing a HW Compression IP is especially challenging. The HW must be compliant with industry standard SW compressors, and as a data transformer, must be able to handle an exponentially large number of input combinations. To address these problems, we deployed FPV - a Formal Verification (FV) based technique that is widely acknowledged and accepted for improving validation effectiveness [2]. FPV is a prevailing technique to catch corner case bugs, raise confidence of design quality and accelerate IP development life cycle. Additionally, we used other powerful FV techniques such as Area Estimation, Design Exploration and Formal Sequential Equivalence to improve and monitor the IP quality from tech-readiness to tapeout.

The following sections provide details on the FV techniques we used along with their results showing a clear impact on Return On Investment (ROI). We show that by deploying these techniques through the IP development life cycle, we estimated design area relatively quickly with reasonable accuracy, significantly increased design quality of the initial RTL drop to the PSV team and reduced PSV efforts for logic/timing/power bug fixes thus improving IP quality and time to market.

II. Adopted Methodology

A. Utilizing design information from formal tool for

1. Early area analysis

Each IP design has an upper bound on overall Si area and power. One of the key aspects of the Proof of Concept (PoC) phase is to determine whether the HW implementation of the proposed architecture can be realized within the specified upper bound. Additionally, initial floor-planning and placement are influenced by the area estimates from the PoC phase. Even though it is an approximation, the area estimate must not exceed 20% of the final implementation. Otherwise, late RTL changes may cause a slip in IP delivery.

In the traditional methodology for area estimation, a PoC level RTL design is synthesized using various synthesis tools. These tools come with the overhead of setup/enablement and may not yet be available during the PoC phase. In addition, these tools have long run times, which can significantly impact early area estimation activities.

Formal tools have the capability of synthesizing the design and providing various design information, e.g., number of flops/latches, gate count etc., along with design elaboration. Furthermore, its design elaboration takes orders of magnitude less time to complete than the standard synthesis tools. Using area numbers for RTL-only logic (i.e., excluding logic for FPV testbench and properties) from the design elaboration stage of the formal tool and

applying the percentage-based approach described below, we were able to estimate area change for the Compression IP relatively quickly compared to the traditional method of using synthesis tools.

Using the formal tool, we first obtained flop and gate count numbers for the previous generation Compression IP based on its tapeout design (A_{PREV}). With the same tool we obtained flop and gate count numbers for the current generation Compression IP based on its PoC design (A_{NEW}). Using these two numbers, we calculated the percentage area change as follows,

$$\% \text{ Area change for the Compression IP} = \{[(|A_{NEW} - A_{PREV}|) * 100] / A_{PREV}\}$$

2. Maintain RTL quality

During the course of IP development, the RTL design team is heavily focused on implementing various IP features with correct functionality. However, non-synthesizable constructs such as combinational loops or unintentional latches may have crept into the RTL code. These issues may be discovered late in the design flow as synthesis tools and other quality checks come online, thus potentially pushing out an IP delivery milestone and impacting overall schedule.

Using formal tools, issues related to non-synthesizable RTL code can be caught relatively quickly. To ensure RTL quality from the onset of the Compression IP execution phase, we enabled elaboration, combinational loop check and latch count check using the formal tool as a GateKeeper check for the design team. These checks prevented the submission of non-synthesizable code multiple times, thus maintaining high design quality as the program progressed.

B. Find early RTL bugs using design exploration and formal tools

In the constraint-based random verification methodology, the PSV team develops the test-bench concurrently with RTL development. Due to this, often there is no testing done for the first RTL hand-off to PSV team, leaving various trivial bugs (i.e., missing correct state transitions, missing correct level/edge transition of signals at interface level, etc.) in the design, resulting in low quality RTL. In some cases, the quality is so low that it cannot even pass basic tests. This impacts PSV progress as the team must resolve these trivial RTL bugs in addition to test-bench issues.

In the traditional FPV methodology, RTL bugs are not found until the start of FV test-plan execution. However, using formal tools for design exploration can help identify such trivial bugs before execution of the FPV plan. Design exploration is used to examine and experiment with various behaviors of the design and get a feel for how it acts under various conditions without requiring a simulation testbench to be developed. This is carried out by describing known design behavior via cover properties and use formal tools to identify if these conditions can be reached or not.

To improve our confidence of the initial RTL drop, we used cover properties for design exploration for the following logic areas:

- Control signals of interface protocols
- Transactions/Data signals of interface protocol
- State machine transitions

As an example, Figure 1 illustrates an FSM with a missing next state assignment required to transition to STATE_G from STATE_B. A simple cover property as shown below, during design exploration would show that the STATE_G is unreachable and identify the bug,

```
cover_state_g_reached: cover property (fsm_present_state == STATE_G)
```

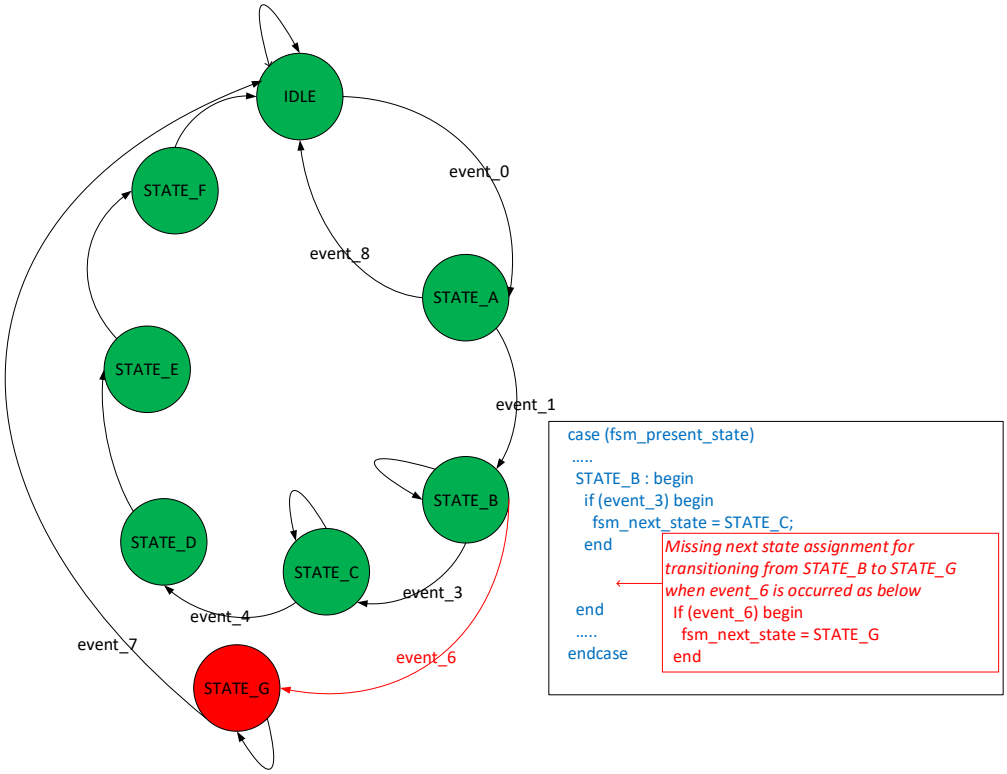

Figure 1. Missing next state assignment in FSM design found using design exploration

C. Leveraging Formal Sequential Equivalence

After the second RTL hand-off to the PSV team, validation of any RTL change due to logic/power/timing fix requires validation of the new functionality as well as re-validation of the existing functionality. This may require having weeks/months of random regression runs depending on the severity of the change adding extra validation work. A traditional LEC (Logic Equivalence Check) method may not be useful here, as it cannot compare non-state-matching designs.

Formal tools have the capability of verifying identical output behavior between two blocks of RTL irrespective of their state-matching, also known as Formal Sequential Equivalence. After the first RTL hand-off to the PSV team, we leveraged Formal Sequential Equivalence for any RTL changes targeting logic/power/timing/lint fixes for early identification of any introduced bugs.

Formal Sequential Equivalence does not require developing any additional checkers/assertions. As illustrated in Figure 2, using formal constraints, the design team can disable the new logic and apply Formal Sequence Equivalence to prove that the original functionality is not impacted [2].

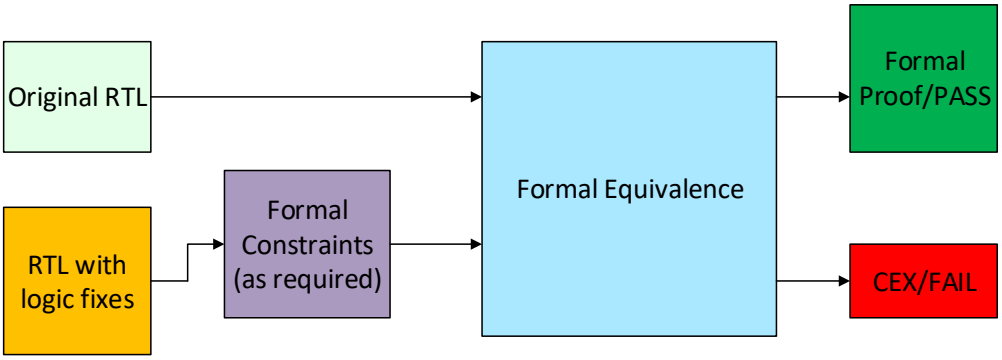

Figure 2. Formal Sequential Equivalence setup to validate “old RTL is same as new RTL without logic fixes”

The formal constraints shown in the Figure 2 are required for logic changes due to functional bug fixes and can be used to accomplish any of the following:

1. To ensure that original design is equivalent to the new design, after enabling the "de-feature" bit (if implemented in HW) or disabling the newly added logic for simple fixes (e.g., adding a term to an equation).
2. To formally prove that the logic change did not introduce any side effects into the design by restricting the formal tool from exercising scenarios that trigger the failure condition. Any failures reported by the formal tool may give insight into the changed behavior.

These constraints are not required for RTL changes due to lint cleanup or code refactoring but may be required for timing and power optimizations.

III. Results and Discussions

In the Compression IP development, use of design information from formal elaboration played a key role during PoC and Execution phases. For PoC, we provided the Back End (BE) team with reasonable area estimates for floor-plan evaluation without relying on them to configure the standard synthesis tools and shortened the PoC phase by 4 weeks. At the end of the design cycle, area estimates were within 5% of the targeted numbers, as shown in TABLE I. In addition, using this information we were able to quickly root-cause issues related to high flop count seen in regular synthesis due to use of incorrect RTL defines. From an execution perspective, we enabled formal elaboration as part of our GateKeeper check and caught submission of non-synthesizable RTL code multiple times thus maintaining quality of the RTL.

TABLE I
COMPARISON OF AREA ESTIMATION USING FORMAL TOOLS AND SYNTHESIS TOOLS FOR SOME KEY BLOCKS OF THE COMPRESSION IP

	Gate count + Flop count using formal tool		% change	Area for previous generation after synthesis	Estimated area for current generation	Actual area for current generation after synthesis	% Difference between Estimated area and Actual Area for current generation
	Previous generation IP1p0	Current generation PoC					
Block_A	669,500	775,305	16	48,041	55,727	56,840	1.99%
Block_B	46,611	52,484	12.6	2,957	3,329	3,439	3.33%

Describing good/expected RTL behavior using cover properties to analyze reachability resulted in identifying 31% of the overall bugs through the development life cycle, as shown in Figure 3. Some of the categories of RTL bugs that were found using design exploration along with example cover properties are shown in TABLE II. More importantly, the use of design exploration uncovered fundamental bugs which enabled the PSV team to shift their activities to corner case testing with the first RTL drop.

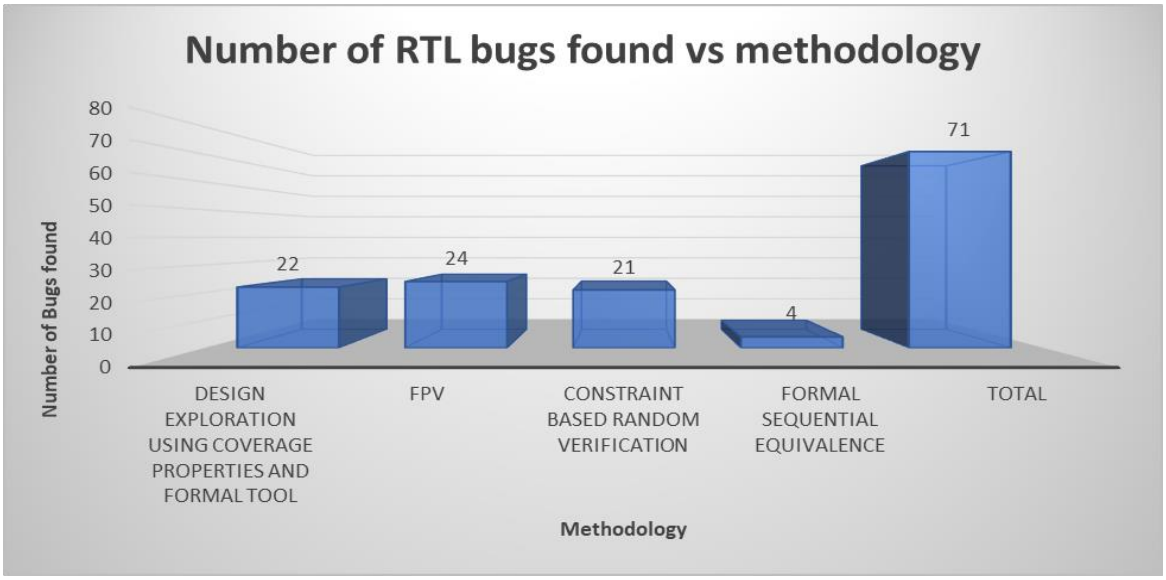

Figure 3. Number of bugs found vs methodology used

TABLE II
RTL BUG CATEGORIES AND EXAMPLE COVER PROPERTIES

RTL bug category	Example cover property
Missing state machine transition	cover_state_g_reached_in_fsm: cover property (fsm_present_state == STATE_G)
Missing port connection	cover_port_a_is_driven_via_signal_a: cover property (port_a == signal_a)
Incorrect signal sensitivity	cover_signal_a_is_pulse: cover property (signal_a ##1 ~signal_a)
Incorrect parameters for memory pointers	cover_mem_pointer_reach_max_value: cover property (mem_pointer == MAX_VALUE)
Counters not incrementing correctly	cover_counter_value_should_match_to_expected_at_the_end: cover property ((end ==1'b1) ##0 (counter_value == expected_value))
Mismatch between error codes and error type	cover_error_code_0_and_its_type: cover property ((error_code == code_0)_&& (error_type == type_0))
Missing assertion of valid signals for interfaces	cover_valid_for_interface_0_is_asserted: cover property (valid_for_interface_0 == 1'b1)

Embracing Formal Sequential Equivalence for validating various RTL changes, reduced PSV efforts for bug fix validation tremendously from 4-6 weeks to a couple of days. This gave the PSV team more time to complete their PSV sign-off activities such as functional/code coverage analysis and closure, test-bench quality review etc. The following is one of the examples where a bug introduced during power optimization activity was found using Formal Sequential Equivalence.

Figure 4 shows simplified RTL code which combines the data from multiple modules into one bus. Power analysis showed that the *data_out* register was not being clock gated. To save power, clock gating and associated logic were added to a standalone combinational block. However, one of the data busses was inadvertently removed in the modification as shown in Figure 5. All PSV tests added as GateKeeper checks passed despite the presence of the bug. Using the Formal Sequential Equivalence methodology, the RTL designer was able to find this bug in less than an hour. Later, we analyzed the time required to identify this bug using random regressions and found that it would have required four days of run time with ~20000 seeds using the tapeout-quality PSV test-bench to catch this bug.

```
parameter   DATA_WIDTH_SIZE                = 64;

input   logic                                       clk ,
input   logic                                       rst ,
input   logic  [DATA_WIDTH_SIZE-1:0]                data_in ,
input   logic                                       data_vld ,
output logic  [DATA_WIDTH_SIZE-1:0]                 data_out ,
output logic                                        data_out_vld

        logic  [DATA_WIDTH_SIZE-1:0]                data_1;
        logic                                       vld_1;
        logic  [DATA_WIDTH_SIZE-1:0]                data_2;
        logic                                       vld_2;
        logic  [DATA_WIDTH_SIZE-1:0]                data_3;
        logic                                       vld_3;

//Power Analysis and code review pointed out that below logic
//identify possible opportunity for power optimization

always_ff @ (posedge clk) begin
  if (rst) begin
      data_out          <= '0 ;
      data_out_vld      <= '0 ;
  end
  else begin
      data_out          <= data_1 | data_2 | data_3 ;
      data_out_vld      <= vld_1 | vld_2 | vld_3 ;
  end
end
```

Figure 4. Simplified RTL code for combining data from various modules

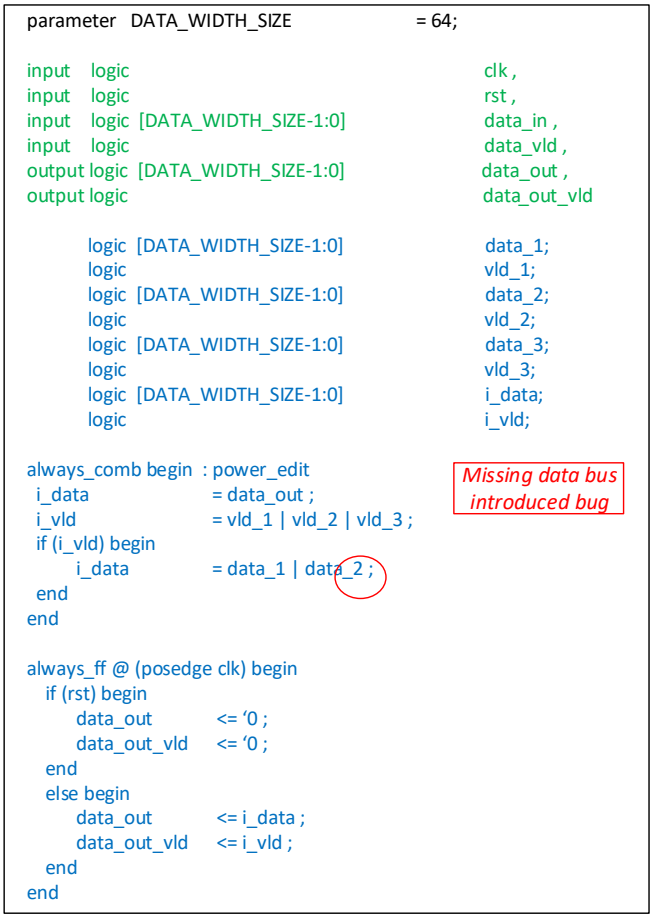

Figure 5. Simplified RTL code for combining data from various modules with RTL bug

In addition, we successfully leveraged the Formal Sequential Equivalence checking for following types of designs:

- Data transformation
- Processor pipeline stages
- Arithmetic blocks
- Flow control logic
- Complex arbiter

By deploying several powerful FV techniques along with FPV, at various stages of IP development life cycle as discussed in this paper, we were able to deliver tapeout-quality RTL for our Compression IP 12 weeks ahead of schedule as shown in Figure 6.

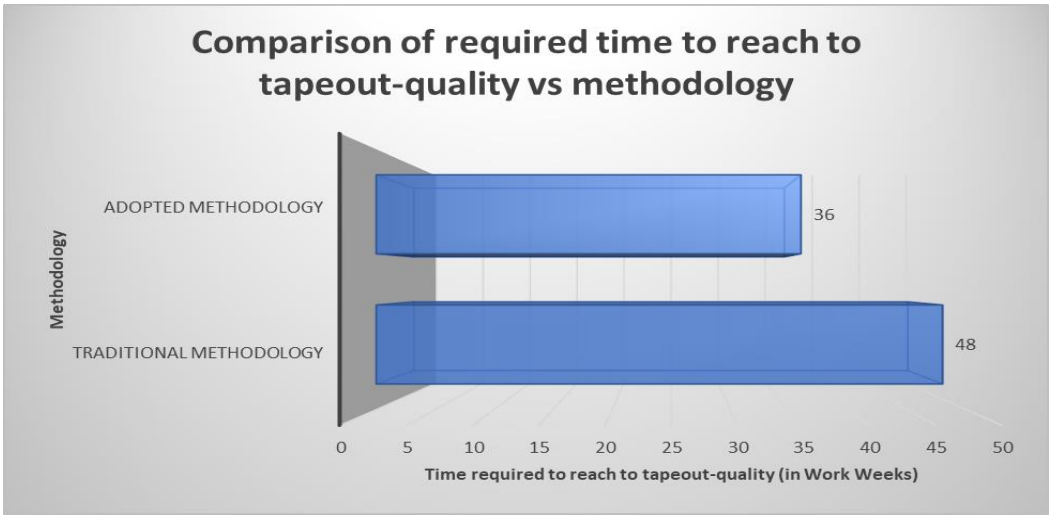

Figure 6. Comparison of required time to reach tapeout-quality using traditional and adopted methodologies

IV. Conclusions and Future work

The paper discussed the use of various FV techniques in addition to standard FPV: Use of design information from formal elaboration, early bug finding via Design Exploration and Formal Sequential Equivalence. We clearly demonstrated the effectiveness of these techniques: saved ~4 weeks from the PoC phase, identified and root-caused ~31% of the total design bugs before the start of PSV activities and delivered tapeout-quality RTL 12 weeks ahead of schedule. We believe that these techniques will also provide similar schedule savings and improvement in quality for a wide variety of designs.

Future work includes deploying more formal checks (e.g., formal linting, formal connectivity check) in the IP development life cycle to further accelerate design convergence. In addition, for our area analysis using the formal tool we relied entirely on data from previous generation. There may be opportunities to leverage the formal tool for area estimation of new designs.

References

[1] Keerthikumara Devarajegowda, Jeroen Vliegen, Goran Petrovity, Kawe Fotouhi, "A Mutually-Exclusive Deployment of Formal and Simulation Techniques Using Proof-Core Analysis" DVCON Europe, October 2017.

[2] Erik Seligman, Carl Dreyer, Ken Haren, Raman Nayyar, "Zero Escape Plans: Tying Together Design, Simulation, and Formal Methods for Bulletproof Stepping Validation" DVCON USA, February 2008.