Overcoming AXI Asynchronous Bridge Verification Challenges with AXI Assertion-Based Verification IP (ABVIP) and Formal Datapath Scoreboards

Bochra ELMERAY, ST-Ericsson, Rabat, Morocco
Phone: +212 5 37 67 85 39
Email: bochra.elmeray@stericsson.com

Joerg MUELLER, Cadence, Munich, Germany
Phone: +49 89 4563 1724
Email: jmueller@cadence.com

1 Abstract

In this paper we will present our experience in asynchronous AXI bridge verification using a new methodology based on the use of assertions and formal analysis for both datapath and protocol compliance verification. We also show how new formal datapath scoreboarding methodologies and enhanced tools used to perform this verification unearthed a serious, show-stopper bug that was not detected by prior methods.

2 Introduction

Globally-Asynchronous, Locally-Synchronous (GALS) design techniques are used to solve problems arising at physical implementation (mainly timing convergence, power dissipation, etc.) In the system on a chip (SoC) design context, asynchronous bridges are used to realize GALS, where such bridges allow a master module and a slave module to communicate together through the same protocol, but with asynchronous frequencies. In this case study, an asynchronous AXI bridge is instantiated several times in the communication between the main interconnect and sub-systems and it is also used to perform frequency conversion. The architecture also includes large asynchronous FIFOs which have two ports: one for writing data into the FIFO from one clock domain, and the other for reading data from the FIFO into the other clock domain.

In order to guarantee the functionality of the entire system, the bridge needs to be exhaustively verified. For this reason formal analysis was chosen to implement our verification methodology.

This paper will show how we successfully performed datapath verification with formal analysis – which is typically only used for control logic – to find a serious, show-stopper bug that was missed by other methods (and this issue inspired a significant redesign of our DUT). Additionally, the more exhaustive nature of this technique – including both protocol compliance and end-to-end functional datapath checking – gave us high confidence in the efficacy of our verification.

3 GALS bridges Verification

3.1 "AsyncAXI" Bridge Architecture

The "axi2axi" asynchronous bridge is a component that mainly performs frequency conversion. As shown in Figure 1 below, this block is comprised of three major units:

Write unit: The part responsible for re-synchronizing all signals relative to write transactions: write address channel, write data channel and write response channel

Read unit: The part responsible for re-synchronizing all signals relative to read transactions: read address channel and read data channel

Clocks and Resets unit: The part responsible for reset synchronization and clock gating for the write unit and read unit, respectively.

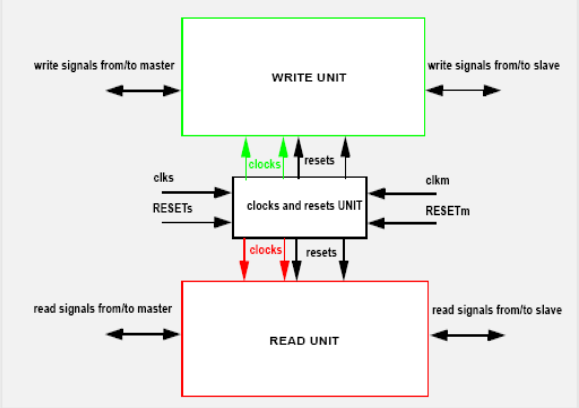

Figure 1: Axi2Axi functional block diagram

3.2 Verification Objectives and Challenges

The 2 main objectives in GALS verification are:

1. Protocol Compliance: Ensure that the Master and Slave interfaces are compliant to AXI protocol
2. Data Integrity: Guarantee that the data transport function of the design is correctly implemented across the clock domain boundary by verifying all datapath channels of this bridge using scoreboards.

Verification of such designs faces several challenges:

1. The complexity of the design itself: in addition to a mix of both control logic and computation on datapath, there can be many internal states created by the number of synchronization FIFOs in such designs.
2. Another challenge comes from the asynchronous nature of these designs that complicates the modeling of the clocking behavior vs. synchronous designs. In short, it is practically impossible to build a dynamic simulation environment that would cover all possible clock frequency ranges and combinations.
3. The AXI protocol itself supports a considerable number of signals and rules that must be completely verified.

3.3 Traditional Verification Methodology

In the past, a constrained random simulation environment was created to

verify protocol compliance and the data integrity for such designs. The effort to create such environments was significant and often failed to find bugs that ideally should have been discovered earlier "if only we had written just one more test". In particular, verifying the full configuration space for protocol and clock frequencies proved to be an unsolvable challenge. In response formal verification was added to augment the protocol compliance analysis. This immediately filled some gaps left open by the traditional verification approach, eliminated additional corner case bugs, and led to earlier verification results.

Although a commercial AXI Verification IP (VIP) component (1) and formal tool (2) were used, the then extant "off the shelf" solution did not go far enough for the following reasons:

1. Due to the complexity of this design (which had many internal states, and a lot of control logic and computation) some assertions were proven only up to a certain depth (the tool calls this result "explored") and never passed or failed for the complete state space.
2. The debugging was extremely difficult and slow in this particular environment because the signal level representation of the counterexamples required significant protocol and design expertise to perform any meaningful analysis.
3. The environment only performed partial checking. Specifically, it only checked for protocol compliance and did not consider any end-to-end functionality like the data transport over a clock domain crossing.
4. No verification plan existed for guiding the verification and measuring progress (which is included in a metric-driven verification flow).

These deficiencies, and the desire to completely verify the design in one environment, caused us to rethink our strategy and adopt a new approach for the verification of this bridge. This new approach is completely based on formal verification without the need of a dynamic simulation environment, and brings to bear a lot of innovative techniques to address known formal analysis challenges.

4 AXI Bridge Formal Verification Experience

4.1 New Verification Strategy

The new verification strategy is mainly based on 2 innovations:

a) A new version of the AXI3 Assertion-Based Verification IP (ABVIP) which enabled a full formal verification of protocol compliance against the ARM AXI specification.
b) A new methodology for verifying asynchronous datapaths, based on

concepts inspired by academia, which uses sequences of symbols to verify data integrity more efficiently with formal verification.

These components are now embedded in a formal-aware metric driven verification and regression environment, taking advantage of debugging capabilities supported by the formal tool.

4.2 Datapath Verification

The methodology employed for formally verifying datapaths is based on the "sequences of symbols" method introduced in research first by Wolper (3) in 1986, then applied by Stangier (4) in 2001, and converted to be usable in industrial settings by Mueller (5) in 2011.

4.2.1 Symbol

A symbol is a non-deterministic constant. It is implemented as a signal that is stable but unassigned in HDL. For example, in Verilog this could be written as:

```
wire [31:0] symbol;
assert property($stable(symbol));
```

4.2.2 Sequences

The symbol is used to form input/output streams of certain shapes (sequences) and reduces the view of the entire value space down to 2 distinct value sets:

a) Value equal to symbol (S)
b) All other values (.)

Examples of sequences used:

```
first_symbol:       S......
always_symbol:      SSSSSSS
never_smbol:        .......
one_symbol:         ...S...
consecutive_symbol: ...SS...
```

For every such sequence there is an associated set of a constraint and an assertion. Separating the sequence checking into these 2 components simplifies the checking and removes the dependency on multiple clock domains in the properties.

Example code fragments of the "one symbol only" sequence checking components:

```
assume property (@(posedge in_clk)
  in_symbol_seen && in_dvalid
  |-> in_data != symbol);

assert property (@(posedge out_clk)
  out_symbol_seen && out_dvalid
  |-> out_data != symbol);
```

This code enforces that the symbol, once it was seen (as indicated by the flag "…seen"), will not appear a second time.

4.2.3 Checks

The check itself drives these sequences into the input and then compares against the output of the datapath (Figure 2).

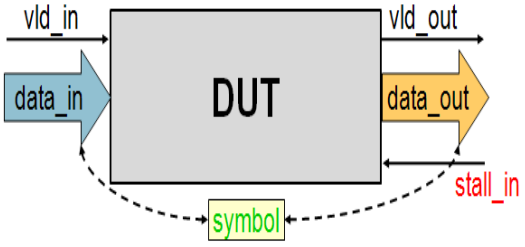

Figure 2: Datapath verification based on symbols

If the sequences don't match, then this points to an error in the DUT transport function. The real power of this approach

comes from the fact that formal verification checks all possible values at any point in time with just one formal proof. This is possible because the formal engine can initialize the symbol to any value and place it at any position in the input stream as required to find a violation of the check.

4.2.4 Formal Scoreboard

Several of the datapath checks with differing characters are assembled in a formal scoreboard package (provided by the tool vendor), that is instantiated for each bus as shown in Figure 3.

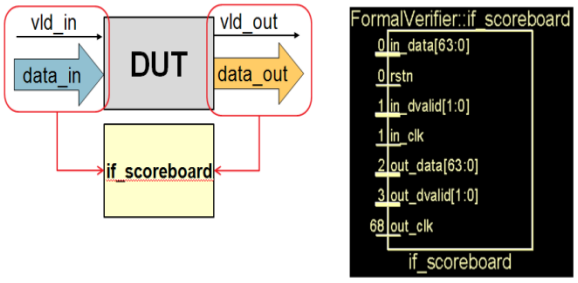

Figure 3: Cadence's Incisive Formal Scoreboard package

For the AXI bridge discussed here there are a total of 7 datapaths and, thus, 7 scoreboard instances:

1. Write Address ID
2. Write Data ID
3. Write Response ID
4. Write Data
5. Read Address ID
6. Read Response ID
7. Read Data

This new methodology enabled more complete verification by fully covering and concluding the previously missing functional end-to-end checks.

4.3 Protocol Checking

Checking protocol compliance of standard bus protocols is much easier today than 8 years ago because of the existence of pre-validated verification components – Assertion-Based Verification IP (ABVIP) – that implement the compliance rules in an executable form (1). Our current environment includes 2 instances of a newer AXI3 ABVIP attached to the two interfaces of the bridge and provides both checking and constraining for legal AXI traffic.

4.3.1 Divide and Conquer

For protocol checking our verification approach was to separate multiple functional cases in a divide and conquer manner. This helps to simplify debugging and reduce the wall clock run time of the formal proofs. It was determined that the available parameters of the ABVIP could be utilized for these separations.

4.3.2 ABVIP Parameters

The parameters used for partitioning our verification included the following:

- **Pipeline Depth**: This parameter was used to control the maximum number of pending transactions. While the design requires 8 transactions to fill the internal pipeline buffers, the parameter was reduced to 2 for many cases that did

not focus on stressing pipeline management capabilities.

- **Byte Strobes**: This parameter allows turning on byte strobe driving and checking. Since this is an "expensive" feature for formal tools to compute, a separate test was created to focus on this functionality.
- **Exclusive Accesses**: Similar to byte strobes, this functionality received a dedicated test because of the complexity it introduces even for unrelated functionality.
- **Data before Control and Write/Read Interleaving**: The AXI protocol allows data before control and out-of-order responses. The separation of these modes was required for compares because the previous version of the ABVIP used did not provide the full set of combinations (see Table 1).

4.3.3 Environment Constraints

In addition to ABVIP parameters reset and clock constraints were also used to create simplified and dedicated environments to focus on specific features.

- **Reset**: In all functional modes, the reset pin was tied off. However, to complete the verification, a dedicated reset test was also added that allowed the reset pin to toggle and evaluate all scenarios under reset.
- **Clock**: The formal tool offered complete freedom on specifying master and slave clock waveforms, or leaving them as completely independent and asynchronous[1] pins. That was a mandatory requirement to stress the clock domain crossing functionality inside the bridge. Following Steffenhagen's "Clocking Strategies" (6) a simple "sync" mode where both clocks were equal and an "async" mode where both clocks were unconstrained (but fair) was created.

4.4 Verification environment

4.4.1 Partitioning

The number of possible combinations of these checks, parameters and constraints was huge. Since there is so much redundancy in many of the combinations, there was an effort to define a reasonable subset of setups.

[1] "asynchronous" in that context means that any ordering of events is allowed. It does not refer to metastability, glitches or similar effects.

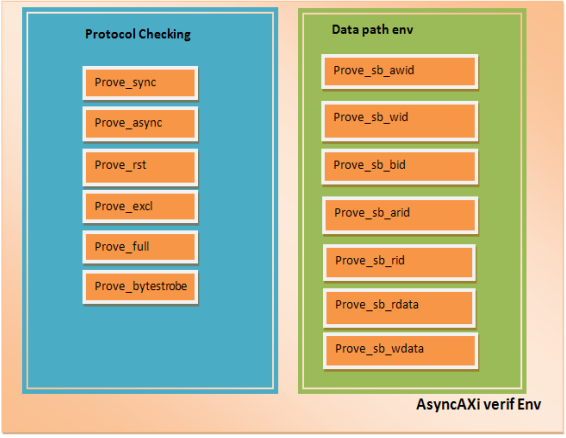

Figure 4: Async AXI verification

As shown in Figure 4 the resulting verification environment is separated into two big partitions: One for protocol checking and one for datapath verification.

4.4.2 Protocol Partition

For protocol checking the following setups were created:

Prove_sync: In this test, all assertions assuming master and slave clocks are synchronous.

Prove_async: In this test, all assertions assuming clocks are fully asynchronous.

Prove_rst: In this test, the reset signal is unconstrained and the reset checks are enabled in the ABVIP.

Prove_excl: In this test, exclusive accesses for generation and checking were enabled.

Prove_full: In this test, the goal was to stress the pipeline in the bridge. The maximum pending transaction parameter of the ABVIP was changed to 8.

Prove_bytsrobe: In this test, bytestrobe calculation and checking is enabled.

4.4.3 Datapath Partition

For datapath checks, a separate setup for each data transport path identified was created; referring to the AXI signals awid, wid, bid, arid, rid, rdata, and wdata on either side of the bridge accordingly.

4.4.4 Conclusion

The conclusion upon review and analysis was that the overlay of results from all these setups is sufficient for the verification needs, while still remaining in a manageable range of tests.

4.4.5 Regression Management

All of the tests were organized in a formal regression suite. Tests were distributed on a server farm and the results were brought back together in one unified view using a regression and analysis tool called “Incisive Enterprise Manager”. The screenshot below shows an intermediate state of our verification. At that stage the total number of runs in this regression summed up to 192 individual formal and assertion-driven simulations (ADS) (as per Section 4.5 below) runs, with 168 passed and 24 failed (see Figure 5).

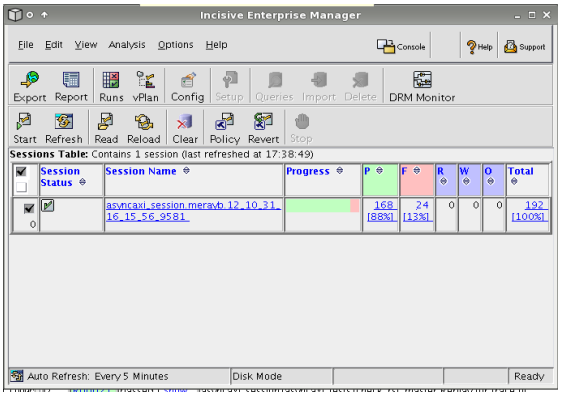

Figure 5: Example Regression Results Table

Analyzing such a high number of tests could easily frustrate and/or tire out the verification engineer, risking the accidental overlooking of important results. The unified results displays provided by this tool allowed us to get an overview of the current state of the verification that's also back annotated into the original verification plan. This clear view of the overall project status and ongoing positive results vs. the plan gave us a high level of confidence in our formal environment.

4.5 Hybrid Use Models

To address the lingering doubt about the formal analyses that yielded the inconclusive "explored" results, a mixed simulation and formal (a/k/a "hybrid") use model for further bug hunting was adopted. As a result, we were able to solve assertions that were previously explored and find more bugs in the process.

4.5.1 Assertion-Driven Simulation (ADS)

Assertion-driven simulation (ADS) uses a seed based property constraint solver to generate a trace for a simulation from the same environment that's used for formal verification (PSL/SVA constraints). Unlike formal proofs, it only computes solutions for these constraints and not for the design, so it is independent of the actual design size and complexity. Similar to regular simulation, this technology is capable of detecting deep assertion failures and passes for covers. (However, it cannot provide passes for assertions and failures for covers.)

In this case study, the use of assertion-driven simulation technology was very useful during the initial environment creation phase because it provided instant feedback on the constraints. The waveform trace provided by the tool enabled a user to confirm visually the environment (i.e. constraints) was actually modeling what it was supposed to model. This helped saving cycles that otherwise would have been wasted with invalid proofs. This efficiency enabled to reduce setup time and consequently the overall time to reach to conclusive results.

4.5.2 Constraint Minimization

Underconstraining is a common method to improve inconclusive formal results and turn them into passes. In the environment created here many of the ABVIP constraints that were active during a proof were actually not needed, i.e. when we disabled them, the assertion was still passing, and runtime was actually faster due to the reduced complexity. But manually creating

these minimal constraint sets is tedious and error prone since it can lead to invalid counter examples if a required constraint was omitted.

To automate this task, the formal tool supports several constraint minimization technologies required to verify the difficult assertions. Internally it uses a sophisticated iteration algorithm utilizing formal and simulation engines that calculates this minimal set of constraints automatically (Iterative Constraint Minimization – Applied for US patent). It is particularly interesting that this algorithm can produce also valid failures, not only passes, unlike our manual minimization approach.

This new constraint minimization feature helped to conclude on some properties that were previously exploring.

4.5.3 Replay

Another feature that utilizes both formal and simulation capabilities is called "Replay". It takes the information obtained by the formal engine during a proof of one specific assertion and uses it as a guide for a constrained random simulation (ADS). While running this re-simulation it enables all the other assertions, which now become subject to additional failures. In the later stages of the verification this enabled us to find additional failures for previously inconclusive properties.

4.6 Debugging Enhancements

4.6.1 Signal Level Challenge

Although the performance of the tool is very important, many hours were spent analyzing and understanding waveforms. At the start of the project the tool produced signal level waveform layouts without any high level information like in a transaction-based simulation environment. Looking at these signals and composing the overlapping AXI transactions from the waveform was extremely difficult, and figuring out the state of the system with respect to ongoing and unfinished transactions was almost impossible.

The tool offered the capability to manually create waveform configuration files that would format the data in a better way, but it would be significant effort and hardly reusable if the topology or a parameter changes.

4.6.2 Transaction Level Analysis and Protocol Aware Debug

During the project we collaborated with the tool and ABVIP provider to implement automatic formatting of the waveforms, meaning the tool would automatically identify the given protocol being used from the instantiated ABVIP, and thus provide protocol specific grouping, formatting, coloring and a transaction level view of the signal level activity.

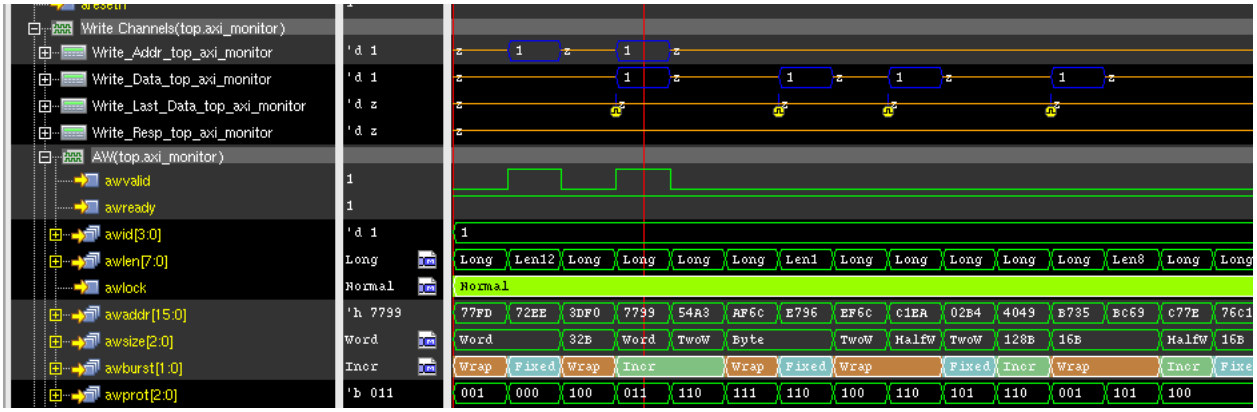

Figure 6: Counter-example waveform formatted per AXI specification

Hence, for every ABVIP instantiated in the environment, the debug waveform now automatically applies groups and transaction-like fibers for the AXI Read and Write channels. Sub channels details underneath the fibers are enriched with mnemonic maps for protocol fields like of burst, lock, etc. for easier readability (See **Error! Reference source not found.**).

A second, complementary register window provides a view of the state of the bus with respect to ongoing and outstanding transactions in form of a table. The position of the cursor in the waveform window (red line in **Error! Reference source not found.**) determines the time at which the status is reflected in the corresponding table (see Figure 7).

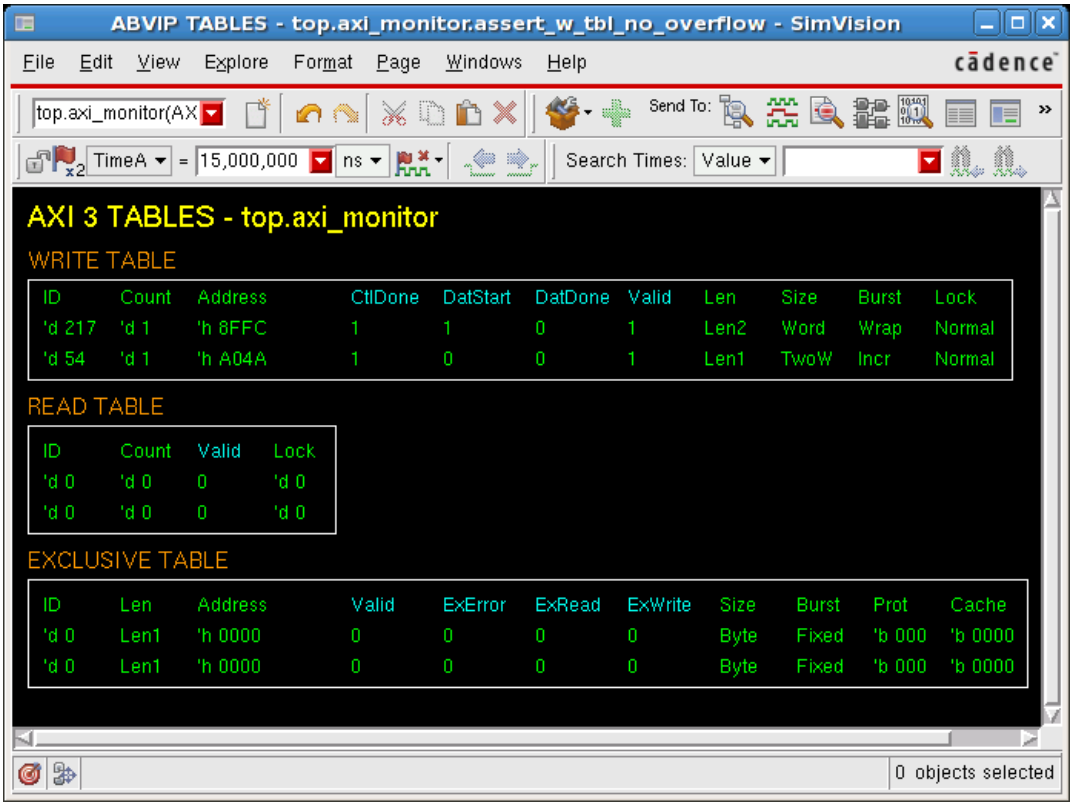

Figure 7: AXI3 formatted data tables

All of these capabilities provided a huge productivity improvement during debugging because the "time-to-understand" the scenario and protocol violation is significantly reduced.

Time saved during debugging freed up time for other verification activities; so this is also to be understood as a contribution to the overall improvement of our results.

5 The Show-Stopper Bug

The new environment described above allowed us to detect a critical corner case bug – a fatal limitation in locked access of the bridge.

5.1 Description

The problem was detected in the exclusive accesses setup: Such exclusive accesses utilize a dedicated signal to lock a destination (slave) to a specific origin (master) exclusively for the duration of a sequence of accesses. There were several assertions that caught this bug, many of which failed with an average runtime of 1 minute, demonstrating that the bridge interfaces did not obey the AXI protocol in an extreme corner case.

To explain the violation in more detail, we present the counter example of one of those failing assertions. This particular assertion checks that transactions driven by

an AXI master within a locked sequence have the same ID value across their read and write transfers, referring to AXI specification (6) section 6.3, pg 6-7.

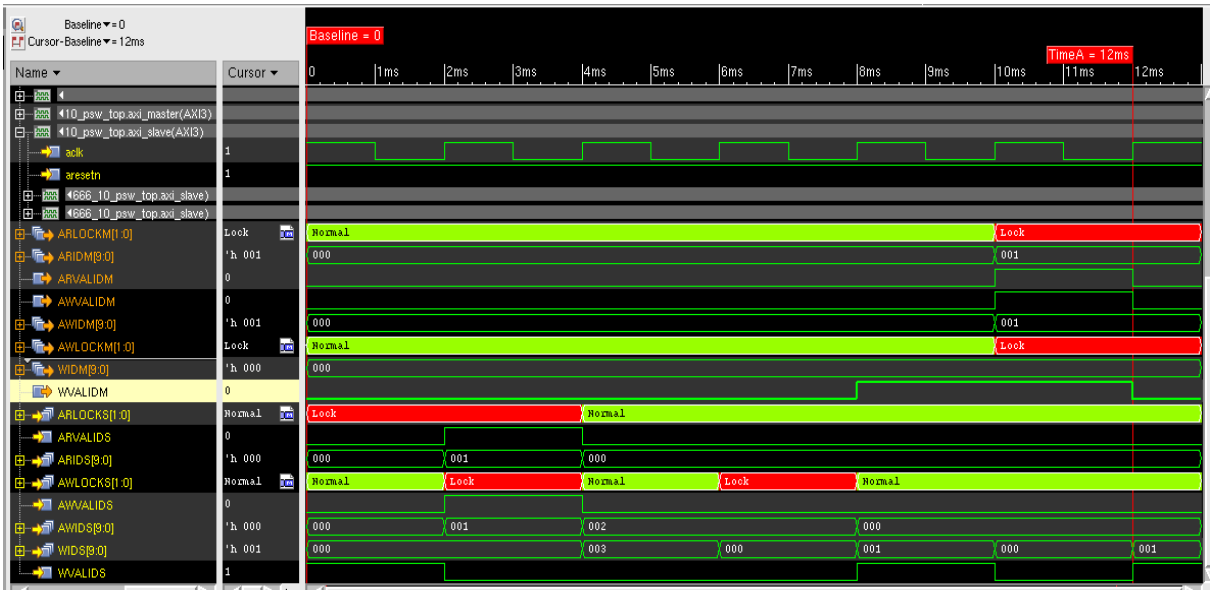

Figure 8: Counter example of bug found

The exact counter example of the failing assertion is captured in Figure 8. It shows such a locked sequence with a locked read and write transfer with ID 1 entering the bridge between 4ms and 10ms (signals with suffix S), and corresponding transfers leaving the bridge through the master interface at 12ms (signals with suffix M). The IDs are expected to be identical within the output sequence and carry a 1, but the actual WIDM signal belonging to this sequence carries a 0, which is a protocol violation originating inside the bridge.

Analysis showed that this was caused by a very specific coincidence: There is an additional write data with ID 0 entering the bridge at 2ms without a corresponding write request, before the locked sequence begins. This will trigger the error in the bridge and subsequently manifest itself as a protocol violation on the other side.

Similar counterexamples also appeared for the other assertion failures.

5.2 Conclusion

This violation could occur because of a designer's assumption that the bridge would be mainly used to transfer control information from a slave to a master interface, rather than data packets utilizing locked accesses. In the worst case this bug could lead to loss of data, and even cause the system to hang. Unfortunately the previous simulation-based environment was not designed to stimulate such a scenario. This is the reason why the bug escaped so far, and a great example of how the more exhaustive nature of formal can save the day.

6 Comparing Formal Results

The new ABVIP together with the tool enhancements improved the number of concluded assertions and increased the reported depth for almost all remaining explored assertions significantly – some even doubled.

6.1 Old vs. New Verification Environment

This new verification environment is much stronger, as it introduces several verification enhancements that do not exist in the old one, starting by protocol checking, asynchronous clock setup, datapath verification, etc. Hence, it was difficult to make a real apples-to-apples comparison between results of the two environments. In order to compare something sufficiently correlated we decided to focus on 2 configurations in both environments that are somewhat similar with respect to the scope of their checks:

1. Interleaving disabled and Data before Control enabled.
2. Interleaving enabled and Data before Control disabled.

6.2 Results

Table 1 presents comparison results by number of assertions for the old and new environment:

	Config 1		Config2	
	old	new	old	new
Total	115	144	108	141
Pass	75 (65%)	108 (75%)	74 (68%)	109 (77%)
Fail	8 (7%)	9 (6%)	3 (3%)	9 (6%)
Explored	32 (28%)	27 (19%)	31 (29%)	23 (16%)

Table 1: Comparison old vs. new Environment

(The explored results were obtained with 1 hour tool effort per property)

According to this comparison we can see an improvement in all categories of results, especially when we are considering that the total number increased in the new ABVIP. Many previously explored assertions moved to a conclusive Pass or Fail state, and the exploration depth of the remaining ones increased significantly (not shown here).

7 Future Enhancements

This positive experience with formal scoreboarding and ABVIP is encouraging us to count on mixed formal and simulation-based tools in future verification projects. The quality of the results was a tremendous improvement over our prior methodology, and in parallel the performance was such that these new methods can reduce the effort spent in other environments. However, some bounded proofs can still remain, and it will take some understanding and design knowledge to interpret an explored depth in order to be confident in

the results. In short, while this is not a completely push-button flow, we believe anyone will also see similar benefits from replicating our approach.

8 Summary

The experience described is one of several successful applications of formal verification on complicated designs like GALS bridges, and represents a very positive experience of datapath verification using formal. We found a serious show-stopper bug that was missed by other methods (inspiring a significant redesign of our DUT). Additionally, the completeness of this setup – including both protocol compliance and end-to-end functional datapath checking of both sides of the bridge – gave us complete confidence that we didn't need to spend more resources on verification.

Overall, we estimate that the specific AXI protocol and functional verification task is three-times faster using formal analysis, formal scoreboard and ABVIP vs. a testbench simulation with dynamic VIP and scoreboard approach.

9 References

1. **Cadence.** *AXI Assertion-Based VIP User Guide.* s.l. : support.cadence.com. Product Version 1.0.

2. —. *Incisive Enterprise verifier XL User Guide.* s.l. : support.cadence.com, 2012. Product Version 12.2.

3. *Expressing interesting properties of programs in propositional temporal logic.* **Wolper, Pierre.** New York : Proc. POPL '86, pp. 184–193, 1986.

4. *Applying Formal Verification with Protocol Compiler.* **Stangier, Christian and Holtmann, Ullrich.** s.l. : Proc. Euromicro Symp. Digital Systems Design, 2001.

5. **Mueller, Joerg.** Quickly Find Data Transport Bugs with Formal Scoreboarding. *www.cadence.com.* [Online] 11 17, 2011. http://www.cadence.com/cadence/events/Pages/event.aspx?eventid=560.

6.6. *Formal verification of an asynchronous STBUS bridge with IFV.* **Steffenhagen, Arthur and Mueller, Joerg.** s.l. : CDNLive EMEA, 2010.

7. **ARM.** *AMBA AXI Protocol Specification v1.* s.l. : www.arm.com, ARM IHI 0022B.

10 Acknowledgements

The authors would like to thank Rachida El Idrissi, Chris Komar, and Joseph Hupcey III for providing feedback during the creation of this paper.