# Directed-Simulation Assisted Formal Verification of Serial Protocol and Bridge

Saurav Gorai [#], Saptarshi Biswas, Lovleen Bhatia, Praveen Tiwari, Raj S. Mitra

[#] Mentor Graphics
Noida, India
saurav_gorai@mentor.com

Texas Instruments
Bangalore, India
{saptarshi, lovleen, ptiwari, rsm}@ti.com

## Abstract
Robust verification of protocol conversion and arbitration schemes of SoC bridges forms a significant component of the overall SoC verification. Formal verification provides a way to achieve this, but a naive approach often leads to explosion of the state space, and is impractical for most of today's protocols and bridges. This problem is further complicated in the presence of serial protocols, where control and data are mixed together and transactions continue for very great depths. White-box verification is not a feasible solution, since these bridges are often imported or generated from other sources, and internal information is not readily available. In this paper, we propose a black-box and hybrid approach to this problem, by judiciously mixing simulation and formal verification. We illustrate our approach by applying it to two dual stage bridges that perform serial to parallel protocol conversion and vice versa.

**Categories and Subject Descriptors:** B.6.3 [**Hardware**]: Logic Design - Design Aids – Verification.

**General Terms:** Verification, Experimentation.

**Keywords:** Formal verification, Model checking, Serial Protocol.

## 1. Introduction
SoC designs use complex bus protocols for high performance data transfers [1]. These protocols have highly configurable modes, extensive pipelining, and complex burst modes, and range from parallel protocols (OCP [2], AXI [3], etc) to serial protocols (I2C [4], USB [5], etc). Different IPs use different protocols for their data transfers and protocol bridges are required to interface and arbitrate between these IPs. Verification of the protocol compliance of these IPs and of the correct functionality of these bridges forms an important component of the SoC's overall verification.

Protocol compliance is typically verified by writing bus functional models (BFMs) and using them during simulation, or writing assertions which represent the protocol's semantics and verifying those using formal techniques. The latter approach is more desirable, because it guarantees completeness, but is hindered by the "capacity problem" of formal verification (FV) tools which limits the size of the module which can be verified.

* This work was done at Texas Instruments.

In recent years several research reports have been published on the formal verification of parallel protocol compliance [6-8], but relatively smaller number of papers has been published on the topic of formal verification of serial protocols [9, 10]. Formal verification of serial protocols is significantly more complex than that of parallel protocols because of the following aspects:

- The state of the protocol transaction at any instant is a function of all the previous bits transferred on the bus.
- The data and control bits are intermingled. Sometimes there is no explicit clock too.
- Formal verification would inherently involve data path logic (FIFO, counter, shift register) which are typically used to transmit/receive/descramble data bits.
- Each transaction spans hundreds of clock cycles and maintaining the entire history in the verification environment greatly increases the complexity of the formal proof process.

To formally verify serial protocol compliance, effective methodologies for all these problems have to be formulated. Structural partitioning is one way to break up a large design into smaller chunks that can be formally verified individually, by following assume-guarantee reasoning [11] at the boundaries. Although conceptually simple, there are practical limitations to this approach. In highly interleaved and complex control logic, there are heavy dependencies between the different components of the design. Even if a partitioning is done in such a case, writing the constraints to represent the boundary assumptions almost amounts to including the functionality of the other modules in their full complexity, thereby defeating the original purpose of partitioning.

In this paper, we describe a novel hybrid verification methodology, and the conditions under which it can be applied. Automated hybrid approaches to formal verification [12-13] involve use of constrained random simulation to pick starting states for formal proof engines. The effectiveness of the above approaches bank heavily on the quality of the testbench (constraints). For protocol bridges involving serial protocols, such automated hybrid approaches would need extensive randomized simulation for hitting corner cases.

Instead, our approach uses a directed simulation assisted formal verification to effectively tackle the proof complexities associated with the verification of serial protocol and bridges. Specifically, we apply this to the verification of two bridges between a parallel and serial protocol (OCP and ARM bus on one side and I2C on the other side), and achieve very high quality verification which could not have been achieved through simulation or formal verification alone.

The outline of the paper is the following: First, we describe a layered approach to writing the properties of a serial protocol which, along with assume guarantee reasoning, is critical to reducing the proof space. In section 3, we describe our hybrid methodology, which uses simulation to bring the system to some known states and then applies formal verification. Section 4 describes two industrial designs on which the methodology has been applied and the results obtained. We conclude the paper by enumerating the advantages of the proposed approach.

## 2. Aspects of Serial Protocol Verification

As mentioned earlier, Serial protocol verification is inherently a difficult task. The reason lies in the fact that a data-transaction over a serial bus spans a longer time interval as compared to parallel-bus based transaction. Remembering such a lengthy history becomes expensive and as a result, formal verification of serial protocol is challenged with capacity issues. To work around this problem, we need smarter verification techniques. The scheme proposed in this paper starts by noticing a similarity among serial protocols in their hierarchical organization. It can be observed that however unique a serial protocol might be, its functionality can still be organized into four major shells of abstraction that are referred to as "layers" henceforth. These layers are described below in a bottom-up fashion.

**Encoding (Data) Layer** - This is the lowermost layer that encodes the logical data-stream into bit-stream or voltage-transitions before putting it on the physical bus. Examples of encoding schemes are NRZ, Bipolar and Manchester.

**Transfer (Packet) Layer** - The transfer layer handles the flow of data in terms of discrete packets, rather than segregated bits. Each packet is a sequence consisting of pre-amble bits, the data to be sent, optional correction-information (parity, CRC etc.) and post-amble bits.

**Transaction (Frame) Layer** - Several packets, taken in the sequence they appear, form a meaningful "frame" of information that is communicated from the transmitter to the receiver. The transaction layer carries out the spatial and temporal splitting of a transaction into packets, controlling their granularity and sequencing.

**Semantic Layer** - The semantic layer is the topmost layer, which determines the interrelation among several transactions. It captures the entire protocol-semantics including features like pipeline, burst, modes of operation etc. Hence this is the most complex layer.

For example, the USB protocol has four different modes of operation. Each mode has a unique semantics and the Semantic layer ensures that each mode is handled in a different manner than the rest. Within a mode, each allowed transaction is called a frame. Each frame is a sequence of packets. This sequencing of packets is determined by the Transaction layer. Each packet, in turn, is a pre-determined sequence of data which is the responsibility of the Transfer layer. The data stream, thus obtained, is encoded through NRZI scheme by the Data layer and put on the physical bus.

## 2.1. Reconstruction of Serial Protocol

From a black-box verification perspective, the only information available at the protocol interface level is the raw signal-stream on the bus. Validating protocol semantics requires knowledge of transaction in the form of abstracted data entities (e.g. packets, frames). Therefore, the logical information transmitted in a serial protocol has to be reconstructed. This is carried out in a bottom up fashion, by working through the layers, building the protocol as we go. This is shown in Figure 1, and forms the basis of the proposed serial protocol validation methodology.

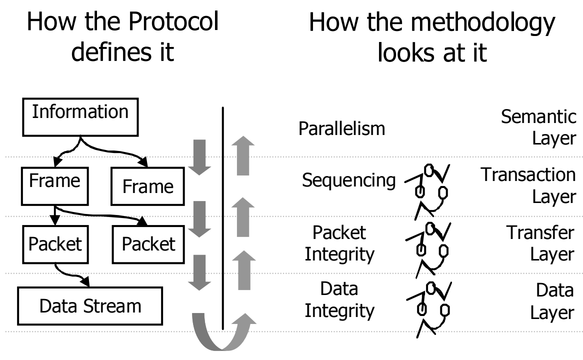

**Figure 1: Proposed Bottom-up FV Methodology**

The reconstruction of information from data is achieved using hierarchical FSMs. As Figure 1 suggests, each layer has its own decoder FSM. Data layer's FSM accepts the signal stream as its input and outputs the corresponding logical data. These data are input to the Transfer layer FSM and packets are raised as outputs. Packets are converted to Frames by the Transaction layer. Lastly, transactions are built from frames in the Semantic Layer FSM.

Real-life protocols are highly parameterized for scalability reasons. Parameters are easily accommodated in the proposed scheme by configuring FSMs according to these parameters. Clearly, these parameterizations are enforced top-down. For example, in the case of a protocol that supports a read-only mode, the corresponding read-only parameter in the Semantic layer FSM prevents write transactions in the Transaction layer FSM.

## 2.2. Property Writing and Assume-Guarantee

Decomposing a protocol into logical layers paves the way for application of Assume Guarantee reasoning [11]. We start protocol verification from the bottommost layer i.e. Data layer and move upwards. Layer-specific properties are written on the data-entities available in that particular layer. For instance, the properties in the Transfer layer check for correct sequencing of packets and do not focus on either data encoding or mode validation. Clearly, in this layered approach, the sequence of proving properties is important. The verification exercise begins with checking properties for lower layers first and gradually moves up the layered hierarchy. Once the lower layers are proven, these properties are treated as assumptions / constraints while proving the higher level properties. This cuts down the verification state-space, reduces turnaround time, and thus speeds up the overall verification.

## 2.3. State Space Consideration

The way an FSM is coded drastically influences the flop-count and it can be the deciding factor whether the validation is at all feasible in terms of proof complexity. The example in Figure 2 clarifies the point.

Here, data is encoded and transmitted through "d" only when "enable" is asserted. Once asserted, "enable" is kept high for at least 20 clock cycles. A data of "one" is indicated by line "d" going to high for a single clock cycle anytime within the 7th and the 13th clock cycle after positive edge of "enable". A "zero" is implied when the "d" lines stays at zero throughout the entire "enable" window. An unintelligent FSM scheme to decode this data can be starting a counter on positive edge of "enable", counting up to 7, then polling for 6 cycles for any activity on line "d" and then counting again 7 cycles and returning to idle state. In addition to that, we would have to check for "enable" being always high during the entire window. This requires 2 counters (one to count up to 6, another up to 7) operating for the same property.

A better way to deal with this problem is to divide it into three sub-problems or phases. First, the enable window is checked to be at least 20 cycles wide. Then it is checked that there cannot be any activity on "d" when "enable" is high except in the checking window as shown in the figure. Once these properties are proven, an FSM is created that behaves as shown in the figure. The 3 states of the FSM would require just 2 bits as against 6 bits in the previous approach. (Note that we still require the large counters in the first two phases. But their usage is limited to the Data layer only, and so they do not add to the state space exploration of the higher layers.)

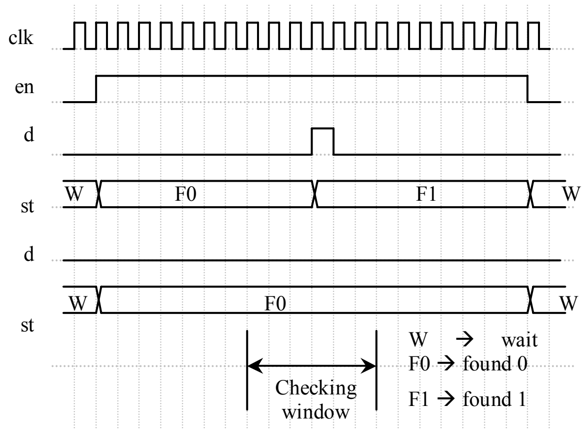

**Figure 2: State Encoding**

Another aspect of state space reduction is the detection of predefined patterns in the data stream to raise an event, for example the detection of the preamble. A naïve method to accomplish this can be to use a shift register and a comparator, but that would linearly increase the number of flops. Using an FSM to detect this pattern would result in logarithmic space complexity and is the preferred solution.

## 2.4. FIFO Verification

As discussed earlier, FIFOs are an integral part of serial protocol implementations and their omnipresence leads to making a proper formal verification methodology for them extremely crucial. One simple way to handle a FIFO is to reduce the parameters - the depth and width of the FIFO. But this may not always be possible without compromising the completeness of verification. In a simulation framework, black-box verification of FIFOs typically involves the introduction of an external FIFO which stores the same set of data and then this is used to verify data correctness and consistency. But for formal verification this would create additional states, and is not considered.

In this paper, we propose a different methodology for verifying a FIFO, which rests on the fact that most bugs in FIFO implementations occur in the control logic component of the FIFO, and it is this area that needs thorough verification. Our approach is to constrain the data bits of the FIFO in such a way that minimal logic comes into the cone of influence of the overflow or underflow property we code and yet point out bugs if any.

Suppose, we have an $N \times M$ FIFO to be verified where $N$ is the depth of the FIFO and $M$ is the width in bits. Let $n$ be the smallest integer that does not divide $N$. In our proposed methodology, we open up a minimum set of least significant bits on the data bus of the FIFO so that we can represent values from 1 through $n$. Rest of the higher bits are always tied to a constant value. Then, the input data to the FIFO is constrained to be only one of 1, 2… $n$ in a round robin fashion. In case of an overflow or underflow bug present in the FIFO, the output data will certainly deviate from the round robin behavior. This is because the data written at any FIFO location will always be different than the one previously written as $N$ is not divisible by $n$. This way, a formal verification strategy can trace bugs very quickly for FIFO-based designs without the aid of extensive simulations.

## 3. Hybrid Approach

In the above sections, we have discussed the means to tackle formal proof complexity arising from serial protocol behavior and design structures like FIFO's. These techniques can be further augmented by using the hybrid approach described below.

The directed simulation assisted formal verification approach is based on functional partitioning of the system and breaking up the entire functional space into multiple logical sub-spaces. Directed simulation is used to guide the system to some known states, and formal verification is used in each of these logical partitions of the system to attain the coverage goals. However, it must be noted that arbitrary pruning down of the total state space can defeat the purpose of performing formal verification itself. Hence, a proper functional partitioning of the system is critical to this methodology. For example, if a device is supposed to behave in, say, one out of only four functional modes of operation at a time after the system comes out of reset, it is completely justified for a formal verification plan to narrow down the entire state space of the system into four smaller state spaces and perform formal verification within each of these spaces separately.

Typical scenarios where a directed simulation can be used to bring the design to a known state could be complex register programming (software configuration), functional reset sequence after the system reset, fixing the modes of operation, loading the FIFO with predetermined pattern of data etc. Formal verification can then be carried out with appropriate pin constraints for the portion of the functionality that starts after this known state (Figure 3). For a large system, this process of combining simulation and formal verification can be fairly automated in which formal verification will happen multiple times for the same design by moving it to different initial states of interest through the use of proper design knowledge.

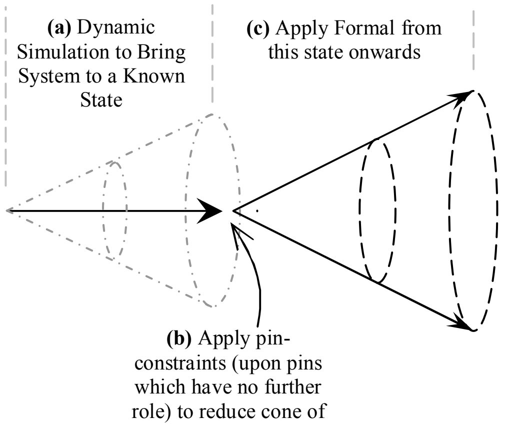
(a) Dynamic Simulation to Bring System to a Known State
(c) Apply Formal from this state onwards
(b) Apply pin-constraints (upon pins which have no further role) to reduce cone of

**Figure 3: Bringing design to a known state**

## 4. Case Study

We now describe the results of applying our methodology to two very complex designs. These designs could not be verified by formal tools alone (we have used a leading commercial tool for this purpose), nor could the coverage goals be attained by simulation alone. Hence the hybrid methodology was adopted for greater confidence in verification.

## 4.1. Application of Methodology

### Problem Description

The design under verification consisted of a two-stage-bridge intended to enable the transfer of data from an ARM7 processor to a serial I2C bus and vice-versa (Figure 4). The first stage in the bridge consisted of an ARM7-to-peripheral protocol bridge while the second stage was a peripheral protocol-to-I2C bridge. (The "peripheral protocol" was actually not a protocol, but a fairly non-standard dense interface.) The ARM7-to-peripheral protocol stage had separate FIFO buffers for the transmit and receive paths and each had a depth of eight bytes. There were several configuration registers to enable the configuration of the block as a master or slave, transmitter or receiver. The I2C is a serial protocol in which each single byte transfer spans several hundreds of clock cycles. The total flop-count of the entire two-stage-bridge system was 805. Formal verification of the block involved protocol checks, data-integrity checks and arbitration checks under various modes of operation.

Structural partitioning of the block was not feasible because the detailed information at the module boundary was not available, due to its history and legacy. This, by the way, is a fairly common problem in verification- teams are asked to verify modules which are imported from other sources and for which hardly any internal information is available.

### Configuration of Design

Directed simulation was performed for configuring the block as a master or a slave by programming the internal configuration registers via a 32-bit wide ARM7 interface. Once the mode of operation was set through simulation by register programming, further analysis was carried out using formal techniques.

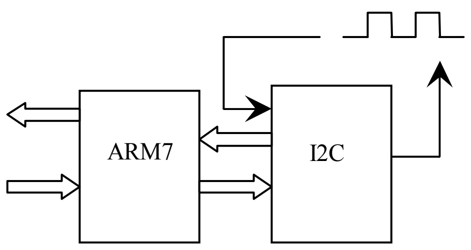

**Figure 4: ARM7-I2C Bridge**

### Protocol Checks

The properties were coded in a layered fashion for I2C protocol validation, as described in a previous section. Protocol checks were performed for both the transmission and reception modes. Protocol checks were also carried out for the parallel interface.

### Data Integrity Check

In "transmit mode" (ARM7 to I2C), data flow was regulated by the software. Further, the software was responsible for ensuring prevention of overflow or underflow conditions. Hence, the FIFO was filled up with a specific sequence of data using directed simulation. Verification was then switched to formal mode, whereby the same sequence was checked for at the output. However, for the "receive mode" no such software control was possible hence further assertions were written to detect potential overflow or underflow bugs.

### Arbitration

Modeling the arbitration of a serial protocol was a complex problem as there was not much of constraining possible on the serial input behavior because both the lines of the bus were to behave arbitrarily. Further, environment modeling had to also take into account an aspect of the design behavior, wherein after losing arbitration, reconfiguration of the registers was necessary to initiate further transfers.

Constraining of the address bits was carried out to minimize proof complexity. Certain address bits were allowed to transition only at certain specific instants in time so that all scenarios of arbitration losses could be explored and the redundant transitions were filtered with the help of appropriate constraints.

### Results

We now present quantitative statistics to highlight the results achieved with hybrid formal verification of this design using a commercially available formal tool. A few corner case bugs related to data integrity and arbitration loss scenario were detected even though the design had been verified using simulation earlier. The checks performed have been summarized in Table 1. Apart from all the times shown in the table, additional time was spent in specification reading and verification planning (7 and 5 man-days respectively). The properties were coded in a mix of the OVL and the PSL property specification languages.

## 4.2. Reuse of the Methodology

The methodology applied in the previous design was successfully reused in catching a corner case bug in another similar design having an I2C interface. The problem here was not to perform full verification, but to locate the cause of a specific bug reported by the software team, and to suggest a robust workaround for the same.

## Problem Description

The module was an I2C controller with an OCP interface at one end and an I2C interface at the other. It could be programmed as a master or slave in receive or transmit mode by programming some internal registers through the OCP interface. The bug was reported to arise "sometimes": when the module was configured in the "Master Receive" mode, and only when odd number of bytes were transferred. The bug was that one register (RRDY) was at times not getting cleared "near" the I2C "stop condition". But it was also observed that it could be cleared only when another register (ARDY) got cleared; however, the relationship between these two registers was not readily apparent from the RTL. The number of state elements in the design was more than 900, and the presence of FIFOs and the serial protocol prevented the direct use of formal verification techniques.

## Results

The I2C constraints developed for the previous design were directly reused for this design. The hybrid methodology was reused in the following way: pin constraints were applied and simulation was used to configure the design in "Master Receive" mode and to take the design to a state where the I2C address transfer phase was completed. Further functional partitioning was done, by dividing the event window (in which the RRDY could be cleared) into 4 distinct parts. Based on this setup, it was proved that RRDY could be cleared in all windows except one, and the specific condition for this was isolated. Next, an assertion was written to prove that if ARDY was cleared then RRDY could always be cleared. The coded properties are shown below:

*"If stop has occurred, MCMD = Write, MDATA(3) = 1 , MADDR = 04 , RRDY =1 and FIFO is empty, then in next cycle RRDY gets cleared"*

*"Prove that software workaround works, Clearing ARDY allows clearing of RRDY"*

```
psl rrdy_bug : assert always ( stop_flag = '1' AND
  MCMD = "001" AND MADDR = "00100" AND    -- Write
  MDATA (3) = '1' AND RRDY = '1'               -- RRDY Set
  AND fifo_empty = '1') → next [1]             -- FIFO read
  (RRDY = '0') @ (rising_edge (CLK));          -- RRDY Clear
psl rrdy_workaround : assert always ( stop_flag = '1' AND
  MCMD = "001" AND MADDR = "00100" AND    -- Write
  MDATA (3) = '1' AND RRDY = '1'                  -- RRDY Set
  AND fifo_empty = '1'                            -- FIFO read
  AND ardy_1_to_0 = '1' AND accept = '1') →  -- ARDY Clear
  next [1](RRDY = '0') @ (rising_edge (CLK));   -- RRDY Clear
```

The assertion for the workaround was also violated, and a refined workaround was suggested and then proved. This extreme corner case situation was not caught during the regular simulations performed for the module's verification. Using the hybrid methodology, the bug was caught at a depth of 2708 cycles of formal proof (Figure 5) and a cumulative depth of 4908 cycles including the initial simulation cycles. The entire exercise was concluded in less than a week.

## 5. Conclusion

We have presented an effective methodology to perform formal verification of serial protocol bridges, an area where naïve approaches fail. Based on a layered modeling of the properties and a functional partitioning of the design, we use light-weight simulation to bring the system to some known states, and then apply formal verification. We have demonstrated our methodology on two industrial designs of serial protocol bridges, and in both of them our methodology succeeds whereas formal verification and simulation applied alone have failed to achieve the verification goals. Our methodology, although applied to the verification of serial protocols, can be used to obtain results for other complex designs as well.

## References

[1] E. Salminen, V. Lahtinen, K. Kuusilinna and T. Hamalainen, "Overview of Bus-Based System-on-Chip Interconnections," IEEE International Symposium on Circuits and Systems, 2002, vol 2, pp. II-372 - II-375.

[2] Open Core Protocol 2.1 Specification, www.ocpip.org

[3] ARM Inc, AMBA AXI Protocol Specification, Mar., 2004. www.arm.com

[4] Philips Semiconductors, "The I2C-BUS specification", Version 2.1, January, 2000. http://www.cse.ucsc.edu/classes/cmpe123/Fall02/Files/I2C_BUS_SPECIFICATION.pdf

[5] USB Implementers Forum Inc, Universal Serial Bus Specification Revision 2.0, 2000. www.usb.org

[6] Abhik Roychoudhury, Tulika Mitra and S. R. Karri, "Using Formal Techniques to Debug the AMBA System-on-Chip Bus Protocol," Design, Automation and Test in Europe Conference, 2003, pp. 10828 -10833.

[7] P. Chauhan, E. Clarke, Y. Lu and D. Wang, "Verifying IP-core based system-on-chip design," Proc. IEEE ASIC. Conf., Washington, DC, USA, Sept. 1999, pp. 27-31.

[8] A. Goel, and W. R. Lee, "Formal verification of an IBM coreconnect processor local bus arbiter core," Proc. Design Automation Conf., June, 2000.

[9] Shivakumar Chonnad and Balachander Needamangalam, "A Layered Approach to Behavioral Modeling of Bus Protocols," Proceedings of the 13th Annual IEEE International ASIC/SOC Conference, 2000, pp 170 - 173.

[10] B. Plessier and C. Pixley, "Formal verification of a commercial serial bus interface," Conference Proceedings of the 1995 IEEE Fourteenth Annual International Phoenix Conference on Computers and Communications, 1995.

[11] C Norris Ip, "Formal interface compliance verification", Electronics Design Process Workshop, April 2003.

[12] Albin K, Yuan J, Aziz A, Pixley C, "Constraint synthesis for environment modelling in functional verification", Design Automation Conference, pp 296-299, 2003.

[13] Pastor E, Pena M A, "Combining simulation and guided traversal for the verification of concurrent systems", DATE Conference, pp 1158-1159, 2003.

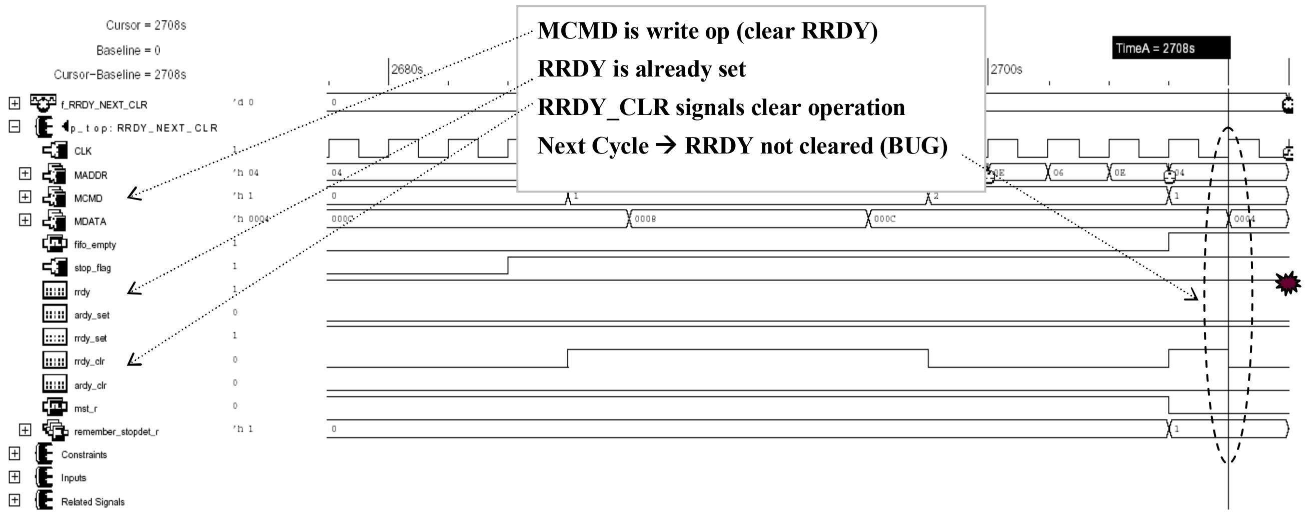

**Figure 5: Waveform for Counter-example of OCP I2C Bridge**

| Mode of Operation | Properties | Time for Modeling | FV Tool Run-time | Pass/Fail/No Conclusion | Comments |
|---|---|---|---|---|---|
| Master-Transmitter | *Protocol Checks* | 3 days | 15 mins | All Passes | Register configuration done using simulation. |
| | *Data Integrity* | 1 day | 5 min | Pass | FIFO pre-loaded with appropriate data using simulation. |
| | *Arbitration* | 3 days | 1 hour | Fail (Depth of 156 clock events) | Unlike protocol checks, some registers here were allowed to be reconfigured during the formal analysis phase. Bug caught: even after losing arbitration, the module was trying to write to the serial bus. |
| Master-Receiver | *Protocol Checks* | 3 days | 15 mins | All Passes | Register configuration done using simulation. |
| | *Data Integrity* | 1 day | 4 hours | Fail (Depth of 10,842 clock events) | Time taken because ARM7 interface was operational in static verification mode. Bug caught: FIFO overflow. |

**Table 1: Formal Verification Results for the ARM-I2C Bridge**