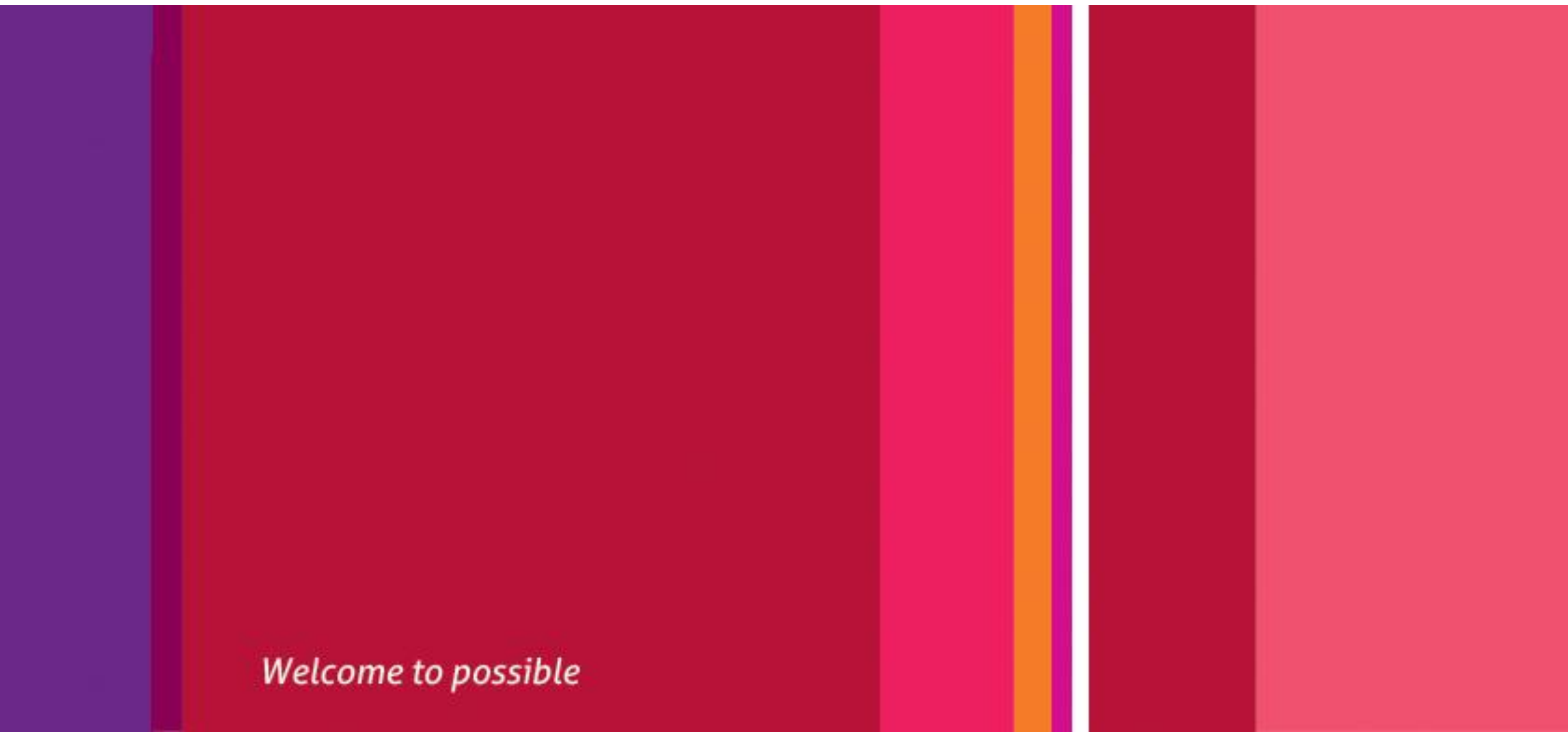
SV Assertion Based Error Signaling Checks Application across popular bus protocols

Mindtree Technical White Paper

April 2014



SV Assertion Based Error Signaling Checks Application across popular bus protocols

Narendra Kumar Kayarmar
narendra_kayarmar@mindtree.com
Nithin Kumar
nithin_kumar@mindtree.com
Mindtree Ltd
Global Village Tech Park, Bangalore, India

Abstract - **Today, many External Bus Interface Protocols have error signaling incorporated. The noise and other system anomalies prevent any standard protocol to deliver error-free data. Interconnect protocols without error signaling suffers a performance hit in this regard. Error Signaling minimizes the data loss during a bus transaction. This paper presents an effective approach to verify the compliance of the error signaling implementation with the specification.**

This paper explains how a procedural assert of an error signaling check can be nested in the action block of a concurrent assertion used to verify the related protocol, which is exemplified for a DFi™ error signaling interface.

I. BACKGROUND [1]

An assertion is basically a "statement of fact" or "claim of truth" made about a design by a design or verification engineer. An engineer will assert or "claim" that certain conditions are always true or never true about a design. If that claim can ever be proven false, then the assertion fails (the "claim" was false).

System Verilog assertions are built from sequences and properties. An assertion works by continually attempting to evaluate a sequence or property. Properties are a superset of sequences separated by implication (|=> or |->) operator. The clause to the left of the implication is called the antecedent and the clause to the right is called the consequent. Evaluation of an implication starts through repeated attempts to evaluate the antecedent. When the antecedent succeeds, the consequent is attempted, and the success of the assertion depends on the success of the consequent.

II. INTRODUCTION

The reuse of programming code is a common technique which attempts to save time and energy by reducing redundant work. Organizations can realize time to market benefits for a new product with this approach. At a time when functional verification is consuming much if not most of the IC design cycle, code reuse promises to slash the amount of verification code and the time required to build a verification environment and start debugging.

Assertions are used to verify the external bus interface protocols and to test the correctness of the signal expected behavior.

In a bus protocol, when an error is detected by a node it sends an error flag on the bus. An assertion is written to check the compliance of this error response with the specification. This assertion requires intermediate factors in the consequent of the property.

These intermediate factors can be eliminated and the related protocol assertion can be reused to check the compliance of the error response with the specification. In essence, this paper suggests methods for effectively combining assertions for error response checking, thereby reducing the assertion coding effort.

1. *Limitations*

As tested, this form of coding currently lacks support from one of the Industry leading simulators.

2. *Summary of Contributions*

The following pages contain below information needed for this report.

- Methodology
- Future Work

III. METHODOLOGY

The approach can be better explained using *Causality* [2] (also referred to as causation [3]). Causality is the relation between an event (the *cause*) and a second event (the *effect*), where the second event is understood as a consequence of the first.

Anything that affects an effect is a factor of that effect. A direct factor is a factor that affects an effect directly, that is, without any intervening factors. (Intervening factors are sometimes called "intermediate factors".) The connection between a cause(s) and an effect in this way can also be referred to as a *causal nexus* [2].

Consider three events *X*, *Y* and *Z*. Their relationship based on their occurrence is *X => Y => Z.*

The causal direction is indicated by => operator. *X* is the cause for *Y* and both *X* and *Y* are the cause for *Z*. *Y* is a direct cause factor for the effect *Z* and *X* is the direct cause factor for the effect *Y*. Here *Y* is an intermediate factor. If you eliminate *Y* and if you can make *X* as the only cause factor of *Z*, then *X* becomes a direct cause factor of *Z*. Thereby the equation reduces to *X => Z.*

Consider a protocol in which two blocks communicate using a 3 wires, consisting of a request (REQ), a grant (GRA) and an error (ERR). Requester asserts REQ to issue a request. When Granter has completed the work associated with a request, it asserts GRA and there must be a maximum RSP_TIMEOUT cycles between a request and its corresponding grant. The timing parameter ERR_RSP (greater than RSP_TIMEOUT) defines the maximum number of clock cycles that may occur from the request to the assertion of the ERR if the granter fails to assert the grant signal within the stipulated timeframe.

Fig.1 shows the block diagram of the requester and granter. Fig.2 and Fig.3 illustrates the protocol description.

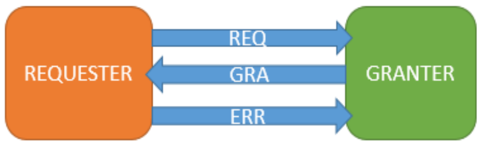


Fig 1 Direction of signaling in from a Requester/Granter

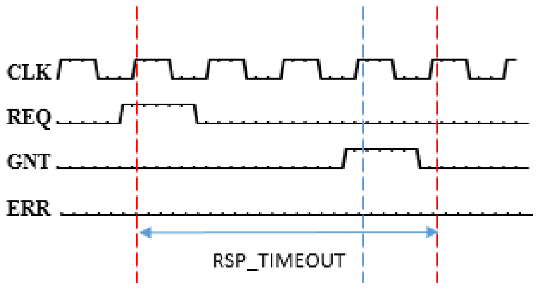


Fig 2 GRA asserts within RSP_TIMEOUT following REQ

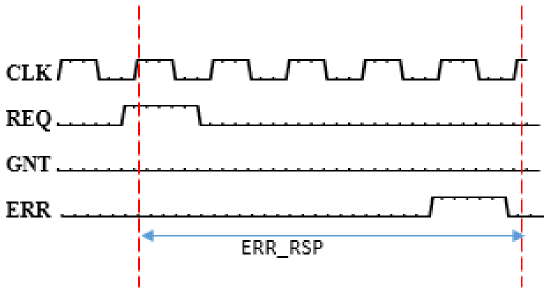


Fig 3. ERR asserts within ERR_RSP following no GRA

In order to check the timing compliance of the requester/granter protocol, you require two assertion to check the timing between

a. Assertion of REQ and assertion of GRA
b. Assertion of REQ and assertion of ERR

A pseudo-code for the requirement is shown below,

```
assert property (REQ |->## [0:RSP_TIMEOUT] GRA); (1)
assert property (REQ |-> !GRA [*RSP_TIMEOUT] |=> ERR);(2)
```

Here the error signaling check *'(2)'* is affected by two factors. One is the REQ and the other is *!GRA [*RSP_TIMEOUT*]. Significant amount of time is spent in deducing the intermediate logic *!GRA[*RSP_TIMEOUT*]. Hence, this approach is laborious considering the large number of assertions for error response scenarios.

According to the new methodology, you can eliminate the need of checking the intermediate factor *!GRA[*RSP_TIMEOUT]* and make the error signal check directly affected by REQ.

This methodology is as shown,

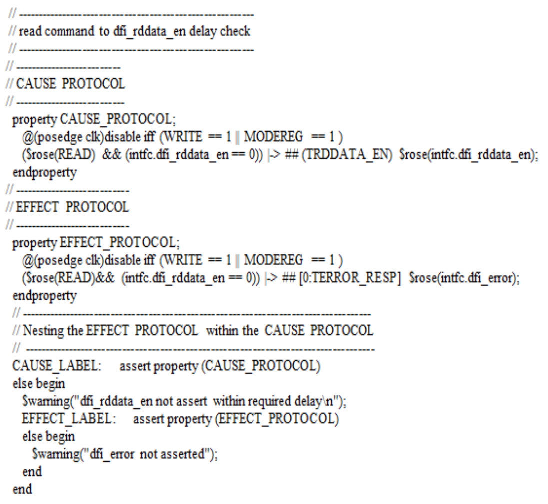
```
  assert property (REQ |-> ## [0:RSP_TIMEOUT] GRA);
else begin
  assert property (REQ |-> ## [ERR_RSP] ERR);
  end
```

This is done by nesting the error signaling check in the main protocol check assertion. Here we can see that the main protocol check *'(1)'* is reused to eliminate the intermediate factor in the error signaling check *'(2)'*. Thereby making the final effect directly affected by the first cause without any intervening factor.

This approach has been tested for DFi™[4] 3.0. DFi™ specification states that, for data errors, the timing parameter is defined as the max delay from *dfi_wrdata_en* or *dfi_rddata_en* to the assertion of the *dfi_error* signal.

For command errors, the timing parameter is defined as the max delay (*Terror_rsp*) from command. Fig 4 shows the code snippet for DFi™ error interface.

The PHY signals an error if the required PHY timing (*Trddata_en*) is not met by the MC. The error interface check is nested within the cause protocol as shown in Fig.5.



```
// -----------------------------------------------------
// read command to dfi_rddata_en delay check
// -----------------------------------------------------
// ----------------------
// CAUSE  PROTOCOL
// ----------------------
 property CAUSE_PROTOCOL;
   @(posedge clk)disable iff (WRITE  == 1 || MODEREG   == 1 )
   ($rose(READ)  && (intfc.dfi_rddata_en == 0)) |-> ## (TRDDATA_EN)  $rose(intfc.dfi_rddata_en);
 endproperty
// ------------------------
// EFFECT  PROTOCOL
// ------------------------
 property EFFECT_PROTOCOL;
   @(posedge clk)disable iff (WRITE  == 1 || MODEREG   == 1 )
   ($rose(READ)&&  (intfc.dfi_rddata_en == 0)) |-> ## [0:TERROR_RESP]  $rose(intfc.dfi_error);
 endproperty
 // ------------------------------------------------------------------------
 // Nesting the EFFECT  PROTOCOL   within the  CAUSE  PROTOCOL
 //  ------------------------------------------------------------------------
 CAUSE_LABEL:     assert property (CAUSE_PROTOCOL)
 else begin
   $warning("dfi_rddata_en not assert  within required delay\n");
   EFFECT_LABEL:     assert property (EFFECT_PROTOCOL)
   else begin
     $warning("dfi_error  not asserted");
   end
 end
```

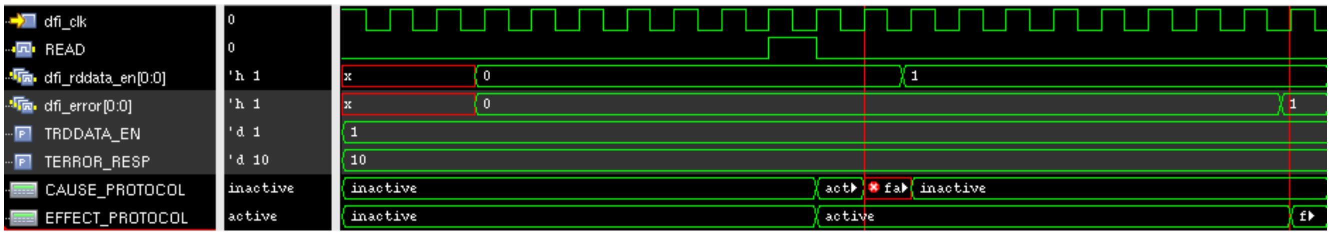


Fig 4 Code Snippet for DFI error interface check

Fig 5 Simulation waveform for theDFi error interface check

IV. CONCLUSION

We have proposed an effective approach to code assertions checks for error signaling for an external bus interface protocol. This approach involves reuse of assertion code to further reduce the coding efforts.

V. FUTURE WORK

The approach can be extended to other protocols such as AMBA–APB/AHB/AXI having error signaling and respective response timeouts.

ACKNOWLEDGEMENTS

The authors would like to thank Santosh Shivadatta (Director & Head – VLSI ,Engineering R&D Service Line , Mindtree Ltd) for his valuable comments and suggestions to improve the quality of the paper. They are also grateful to Vinod Vishwa Gadde (Senior Engineer , Mindtree Ltd) for content review of this paper. This work was supported in part by Mindtree VLSI-CoE.

REFERENCES

1. http://www.sunburst-design.com/papers/CummingsSNUG2009SJ_SVA_Bind.pdf
2. http://en.wikipedia.org/wiki/Causality
3. 'The action of causing; the relation of cause and effect' OED)
4. http://www.ddr-phy.org/

ABOUT THE AUTHORS

Narendra Kumar Kayarmar received M.Tech degree in Electronics from BMS college of Engineering Bangalore, India in 2010. Currently he is working as senior Engineer at eRnD VLSI service line in Mindtree Ltd Bangalore. He is a expertise in ASIC and SOC verification and worked in various client projects.

Nithin Kumar V R received B.E degree in Electronics and Communication Engineering from JSS Academy of Technical Education Bangalore, India in 2011. Currently he is working as senior Engineer at eRnD VLSI service line in Mindtree Ltd Bangalore. He has expertise in IP level verification.

Welcome to possible