Protocol Compliance Verification Using The AMBA AXI Properties

Diego Hdez (diego@yosyshq.com)

Revision	Comment	Date
0.1	Initial draft.	16/11/20
0.2	Updated full working properties	20/11/20
0.3	Minimal edits	25/02/21

Contents

Part I
Introduction

The new AMBA AXI Properties for AXI4-Stream can be used to verify protocol compliance of digital designs that uses the AXI4-Stream bus, as well as to check another Verification IP (VIP) implementations. The goal of this example is to show how to instantiate and use the new AMBA AXI Properties component to verify and measure improvements over the current solution of the AXI4-Stream that is currently integrated in Symbiotic EDA Suite.

Three problems where found using the new AMBA AXI Properties in the current AXI4-Stream VIP that comes in the official release of Symbiotic EDA Suite, two of them can be classified as mild, while a third can be classified as severe (failure to detect lost packets during transmission). This demonstrates the value of the new VIP.

Part II
Technical Specification

1 Architecture

One simple method to verify the current implementation of the AXI4-Stream VIP (named, faxis_slave component) is to connect the new AMBA AXI Properties in the "opposite direction", that is, since the faxis_slave component behaves as sink element (provides constraints or assumptions), this constraint model can be used to restrict the behavior at the input of the new AMBA AXI Properties connected as source. The source component therefore will validate that the constrains behave as intended or if assumptions are missing (the source can also verify over-constraint by checking the vacuity of the assertions).

The Figure 1 shows a block diagram of the architecture for this demonstration.

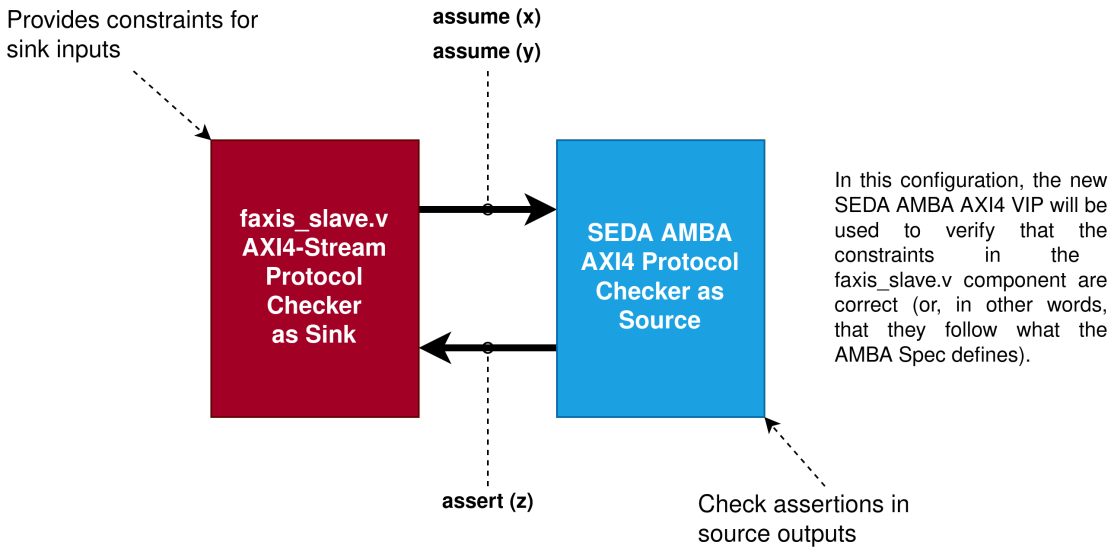


Figure 1: Block diagram representing the architecture of the test.

This method can be seen as a way to measure the differences that exist in two formal models, as the constraints of the sink that influences the cone of logic of the assertions of the source will make such assertions to pass trivially (as they are tautologies). The solver will find a path to a counter example in each of the assertions of the source if there exists logical differences such as divergent property density (more variables affecting the COI of the properties in the source compared with the sink) or missing behaviors that are affected by primary inputs, among others.

2 Environment

2.1 Directory Organisation

The demo files are organised as follows:

- **doc/:** The source of this document.
- **src/:** The HDL source files such as the wrapper to instantiate both verification components, and the scripts for SymbiYosys.

2.2 Configuration Parameters

Two HDL models are needed for this test, the faxis_slave component and the amba_axi4_stream VIP as well as a wrapper to connect both VIPs. The following table sumarises the parameter configuration of both components for this specific test.

Component	Parameter	Value	Component	Parameter	Value
AMBA AXI4 Stream	AXI_BUS_TYPE	1 (source)	faxis_slave	F_MAX_ PACKET	0
	GEN_WITNESS	0		F_MIN_ PACKET	0
	ARM_RECOMMENDED	1		F_MAX_ STALL	0
	MAXWAITS	16		C_S_AXI_ DATA_WIDTH	8
	CHECK_SETUP	1		C_S_AXI_ ID_WIDTH	1
	RESET_CHECKS	1		C_S_AXI_ ADDR_WIDTH	4
	CHECK_XPROP	0 (not supported yet by Tabby CAD)		C_S_AXI_ USER_WIDTH	0
	VERIFY_VIP	1		F_ LGDEPTH	32

2.3 The HDL Wrapper

The Figure 2 shows the amba_axi4_stream_verify.sv wrapper that will be used for this example. It instantiates the faxis_slave as a sink element, and the amba_axi4_stream component as a source element.

```
`default_nettype none
import amba_axi4_stream_seda_pkg::*;

module amba_axi4_stream_seda_verify
  (input wire axi4s_aclk    ACLK,
   input wire axi4s_aresetn ARESETn,
   input wire axi4s_data    TDATA,
   input wire axi4s_strb    TSTRB,
   input wire axi4s_keep    TKEEP,
   input wire axi4s_last    TLAST,
   input wire axi4s_id      TID,
   input wire axi4s_dest    TDEST,
   input wire axi4s_user    TUSER,
   input wire axi4s_valid   TVALID,
   input wire axi4s_ready   TREADY);

   faxis_slave
     #(.F_MAX_PACKET(0),
       .F_MIN_PACKET(0),
       .F_MAX_STALL(0),
       .C_S_AXI_DATA_WIDTH(AXI4_STREAM_DATA_WIDTH_BYTES*8),
       .C_S_AXI_ID_WIDTH(AXI4_STREAM_ID_WIDTH),
       .C_S_AXI_ADDR_WIDTH(AXI4_STREAM_DEST_WIDTH), // ???
       .C_S_AXI_USER_WIDTH(AXI4_STREAM_USER_WIDTH),
       .F_LGDEPTH(32))
   faxis_slave_top
     (.i_aclk(ACLK),
      .i_aresetn(ARESETn),
      .i_tvalid(TVALID),
      .i_tready(TREADY),
      .i_tdata(TDATA),
      .i_tstrb(TSTRB),
      .i_tkeep(TKEEP),
      .i_tlast(TLAST),
      .i_tid(TID),
      .i_tdest(TDEST),
      .i_tuser(TUSER),
      .f_bytecount(), // free
      .f_routecheck()); // free

   // New Symbiotic EDA VIP
   amba_axi4_stream_seda
     #(.BUS_TYPE(1))
   source_checker_helper (.*);

endmodule // amba_axi4_stream_seda_top
```

Figure 2: The amba_axi4_stream_verify HDL wrapper for this example.

2.4 The SBY Script

In the SBY script amba_axi4_stream_verify.sby, two main tasks are defined: **provemode** and **covermode**. The **provemode** task uses the smtbmc and boolector engines for K-induction for validity (checking assertions). The **covermode** uses the smtbmc engine for satisfiability (reachability of cover points). The amba_axi4_stream_verify defines some cover scenarios for the VALID/READY handshaking process and the position bytes and null bytes defined in the AXI4-Stream spec, whose reachability analysis is performed in the **covermode** task. The validity is performed in the **provemode** task.

```
1  [tasks]
2  provemode
3  covermode
4
5  [options]
6  provemode: mode prove
7  covermode: mode cover
8
9  [engines]
10 provemode: smtbmc boolector
11 covermode: smtbmc
12
13 [script]
14 provemode: read -sv amba_axi4_stream_seda_pkg.sv
15 provemode: read -sv amba_axi4_stream_seda.sv
16 provemode: read -sv amba_axi4_stream_seda_verify.sv
17 provemode: read -sv faxis_slave.v
18 provemode: prep -flatten -top amba_axi4_stream_seda_verify
19
20 covermode: read -sv amba_axi4_stream_seda_pkg.sv
21 covermode: read -sv amba_axi4_stream_seda.sv
22 covermode: read -sv amba_axi4_stream_seda_verify.sv
23 covermode: read -sv faxis_slave.v
24 covermode: verific -import -autocover amba_axi4_stream_seda_verify
25 covermode: prep -flatten -top amba_axi4_stream_seda_verify
26
27 [files]
28 ../../../amba_axi4_stream_seda_pkg.sv
29 ../../../amba_axi4_stream_seda.sv
30 amba_axi4_stream_seda_verify.sv
31 faxis_slave.v
```

Figure 3: The amba_axi4_stream_verify.sby script.

3 Execution

The only requisite for this demonstration is license of Symbiotic EDA Suite version, that is needed to synthesize the SVA assertions of the amba_axi4_stream VIP. The execution of the test is done by executing the command shown in Figure 4.

```
1 sby -f amba_axi4_stream_seda_verify.sby
```

Figure 4: Command to perform Formal Property Verification with SBY for this example.

For users with no previous experience in Formal Property Verification, Figure 5 shows a flowchart that can be used as reference to identify and fix issues that could exists.

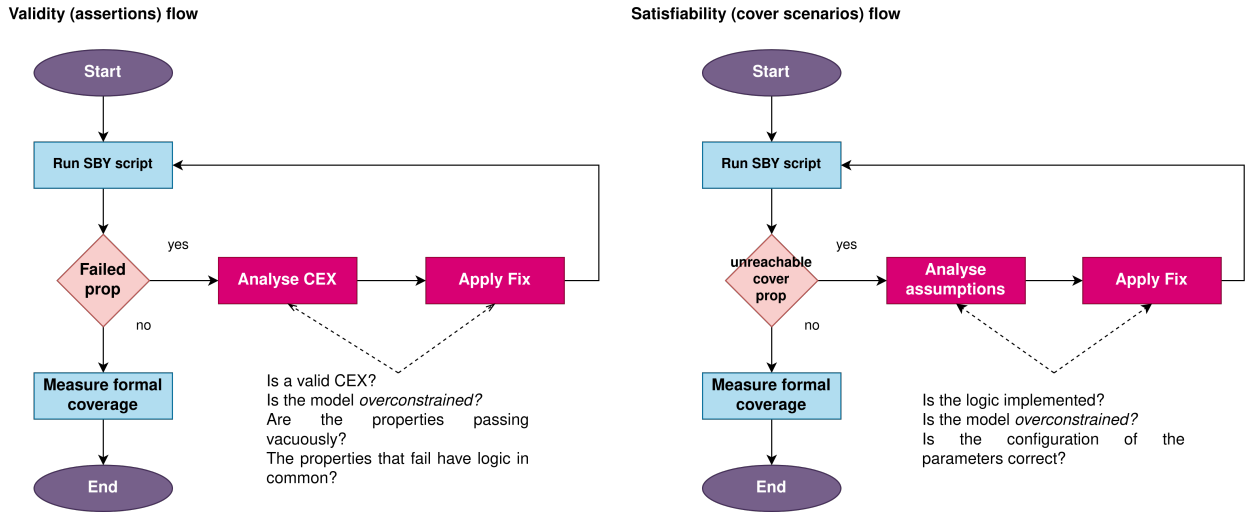


Figure 5: Debug flow.

Part III
Results

4 Protocol Violations

4.1 Satisfiability (cover)

All cover scenarios are reachable.

4.2 Validity (assert)

The table below shows some of the protocol violations (missing/not implemented constraints) that the AMBA VIP found in the faxis_slave component.

Issue	Reference in the AMBA Specification	Details
Assert failed in assert_SRC_OPTIONAL_TUSER_TIEOFF	If TUSER is not present, TUSER must be stable (3.1.4 Optional TID, TDEST, and TUSER, p3-3).	Not implemented in current VIP.
Assert failed in assert_SRC_TKEEP_TSTRB_RESERVED	A combination of TKEEP LOW and TSTRB HIGH must not be used (2.4.3 TKEEP and TSTRB combinations, p2-9, Table 2-2).	The current constraint using i_tvalid as antecedent. Although this is partially correct, the solver found a trace where, after exiting reset, the TKEEP and TSTRB invalid signals propagated.
Assert failed in assert_SRC_STABLE_TDATA	Once the master has asserted TVALID, data and control information from master must remain stable [TDATA] (2.2.1, p2-3, Figure 2-1).	Not implemented in current VIP for TDATA.

4.3 Issue I: Assert failed in assert_SRC_OPTIONAL_TUSER_TIEOFF

4.3.1 CEX

The Figure 6 shows the point where the sideband port TUSER is able to change its state despite of not being used (as defined in the test parameter AXI4_STREAM_USER_WIDTH = 0).

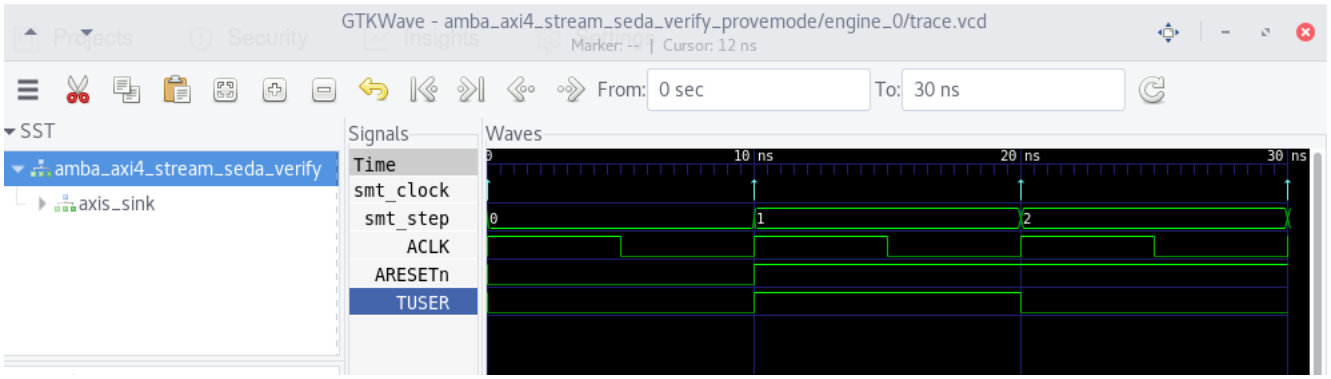


Figure 6: Not stable TUSER when is not used.

4.3.2 Description

The faxis_slave does not seems to implement optional AXI4-Stream signal checks that are defined in the spec. This assertion failed because the test defined that TUSER is not used, but faxis_slave does not have those rules implemented such, therefore creating a window for a structural bug to escape.

- The USER_WIDTH parameter is defined in amba_axi4_stream_pkg.sv as:

```
1  *          control and data AXI-Stream ports.
```

Figure 7: Definition of the TUSER width port in the AMBA AXI4-Stream VIP.

- An propagated to the test amba_axi4_stream_verify.sv wrapper as:

```
1      .i_tvalid(TVALID),
```

Figure 8: The TUSER width parameter in the amba_axi4_stream_verify component.

- But the only rule for stable signals defined in faxis_slave.v is this one, that is different from the optional checks:

```
        // If TVALID but not TREADY, then the master isn't allowed to change
        // anything until the slave asserts TREADY.
        always @(posedge i_aclk)
        if ((f_past_valid)&&($past(i_aresetn))
                &&($past(i_tvalid))&&(!$past(i_tready)))
        begin
                `SLAVE_ASSUME(i_tvalid);
                `SLAVE_ASSUME($stable(i_tstrb));
                `SLAVE_ASSUME($stable(i_tkeep));
                `SLAVE_ASSUME($stable(i_tlast));
                `SLAVE_ASSUME($stable(i_tid));
                `SLAVE_ASSUME($stable(i_tdest));
                `SLAVE_ASSUME($stable(i_tuser));
        end
```

Figure 9: The TUSER width parameter in the faxis_slave component.

4.4 Issue II: Assert failed in assert_SRC_TKEEP_TSTRB_RESERVED

4.4.1 CEX

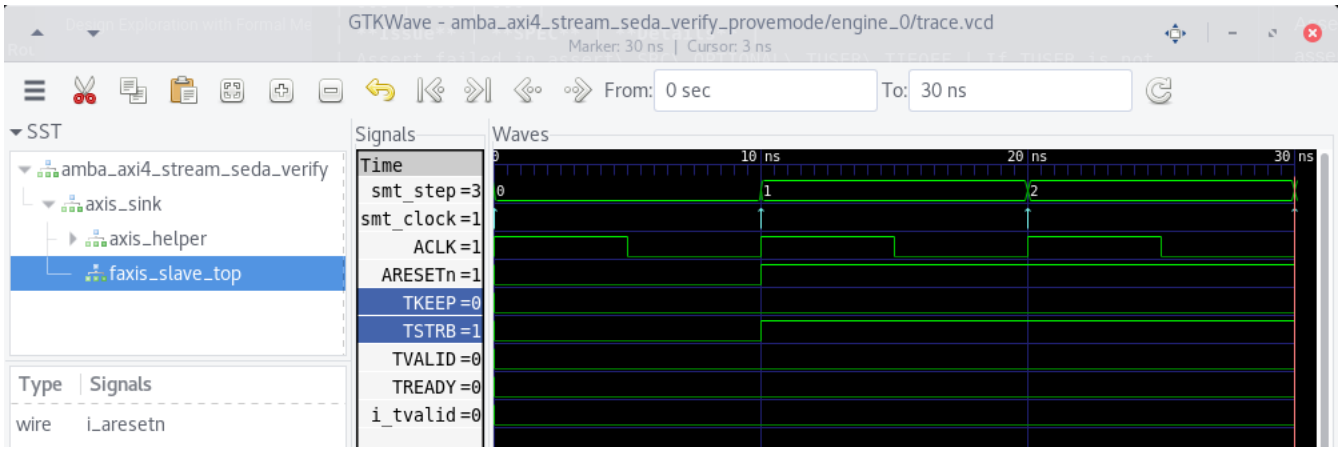


Figure 10: A possible scenario where invalid TKEEP/TSTRB combination is present.

4.4.2 Description

The solver found a way to propagate a reserved AXI state due to the antecedent requirement of i_tvalid in the following assertion:

- Invalid combination of TKEEP and TSTRB defined in the faxis_slave:

```
1        // TKEEP == LOW and TSTRB == HIGH is reserved per the spec, and
2        // must not be used
3        always @(posedge i_aclk)
4        if (i_tvalid)
5                `SLAVE_ASSUME((~i_tkeep & i_tstrb)==0);
```

Figure 11: Antecedent for the rule of TKEEP/TSTRB reserved scenario in the faxis_slave component.

The AMBA AXI4 Stream protocol spec does not explicitly mention that the TKEEP and TSTRB combinations depends on the TVALID/TREADY handshake (it can be argued that is an obvious assumption, but the spec is clear mentioning that behaviors that are not explicitly declared shall not be assumed). For a protocol checker, it is better to enforce that no reserved behaviors occur at any time, regardless of the time window where it appears, and if that time window is a valid data meant to be propagated or not.

4.5 Issue III: Assert failed in assert_SRC_STABLE_TDATA

4.5.1 CEX

As can be seen in the CEX of Figure 12, the optional TDATA is used and it changed the information that the source intended to transmit, but the sink did not capture any of the packets because it was not in a ready state. In other words, a valid packet was dropped. This is an important issue and the faxis_slave must have enforced the check for this data drop packet problem correctly.

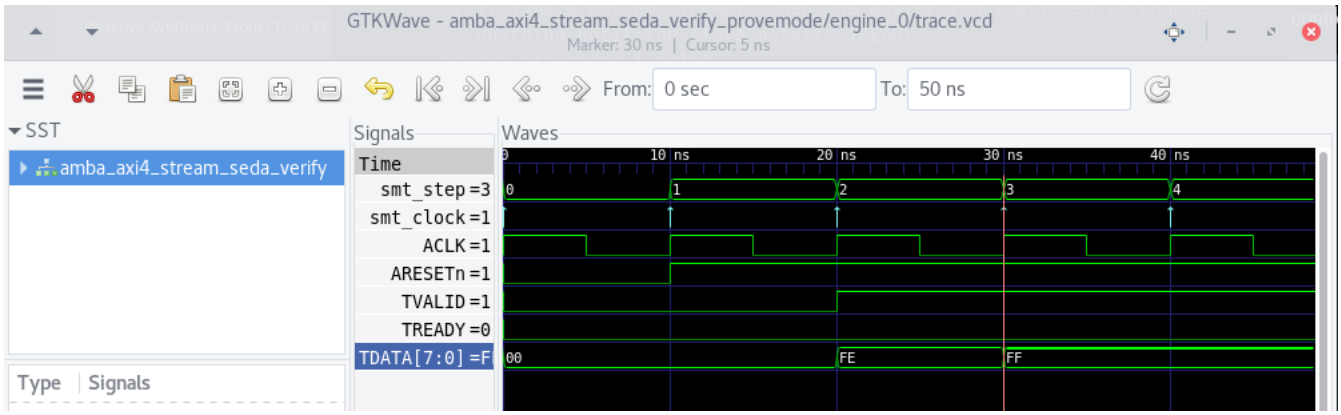


Figure 12: The source is able to change (drop) a valid packet (TDATA) while the sink is not ready (TREADY).

4.5.2 Description

The faxis_slave **does no implement a very critical check** that is defined from the spec as follows: "Once the master has asserted TVALID, the data or control information from the master must remain unchanged until the slave drives the TREADY signal HIGH, indicating that it can accept the data and control information".

- The current implementation does not check for TDATA when the spec says that both data and control should remain stable:

```
        // If TVALID but not TREADY, then the master isn't allowed to change
        // anything until the slave asserts TREADY.
        always @(posedge i_aclk)
        if ((f_past_valid)&&($past(i_aresetn))
                &&($past(i_tvalid))&&(!$past(i_tready)))
        begin
                `SLAVE_ASSUME(i_tvalid);
                `SLAVE_ASSUME($stable(i_tstrb));
                `SLAVE_ASSUME($stable(i_tkeep));
                `SLAVE_ASSUME($stable(i_tlast));
                `SLAVE_ASSUME($stable(i_tid));
                `SLAVE_ASSUME($stable(i_tdest));
                `SLAVE_ASSUME($stable(i_tuser));
        end
```

Figure 13: Data and control information checks in the faxis_slave component.

5 Improvements

The following patches have been implemented in the files amba_axi4_stream_verify_fixed.sby and faxis_slave_fixed.sv.

5.1 Fix I: Assert failed in assert_SRC_OPTIONAL_TUSER_TIEOFF

The faxis_slave should implement the optional interface checks. For this specific issue the implementation could be as shown below, with all optional TDATA checks implemented (unoptimised):

```
1        // diego's fixes
2        // Issue I: Assert failed in assert_SRC_OPTIONAL_TUSER_TIEOFF (FIX), for
             an optimised version see: amba_axi4_stream_seda.sv
3        property_fix_optional_TDATA_I: `SLAVE_ASSUME property (@(posedge i_aclk)
             disable iff (!i_aresetn) UW == 0 |-> $stable(i_tuser));
4        property_fix_optional_TDATA_II: `SLAVE_ASSUME property (@(posedge i_aclk)
              disable iff (!i_aresetn) AW == 0 |-> $stable(i_tdest));
5        property_fix_optional_TDATA_III: `SLAVE_ASSUME property (@(posedge i_aclk
             ) disable iff (!i_aresetn) IDW == 0 |-> $stable(i_tid));
```

Figure 14: The TUSER width parameter in the faxis_slave component.

5.2 Fix II: Assert failed in assert_SRC_TKEEP_TSTRB_RESERVED

As discussed in Section 4.2 Issue II: Assert failed in assert_SRC_TKEEP_TSTRB_RESERVED, the rule should not involve any handshake signal:

```
1        always @(posedge i_aclk)
2        // Issue II: ssert failed in assert_SRC_TKEEP_TSTRB_RESERVED (FIX)
3        // if (i_tvalid)
4                `SLAVE_ASSUME((~i_tkeep & i_tstrb)==0);
```

Figure 15: The TUSER width parameter in the faxis_slave component.

5.3 Fix III: Assert failed in assert_SRC_STABLE_TDATA

The assertion that manages stability of data and control needs to look for TDATA as well:

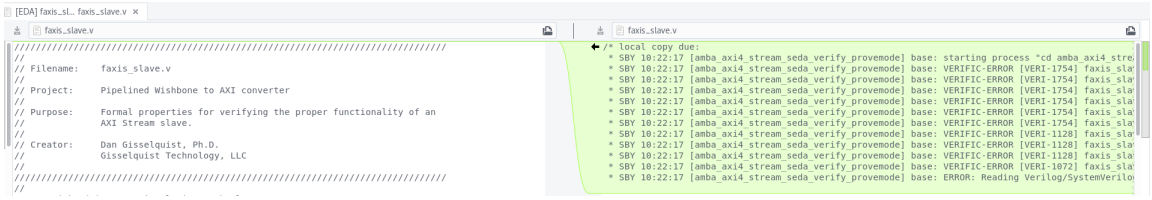
```
        //
        // If TVALID but not TREADY, then the master isn't allowed to change
        // anything until the slave asserts TREADY.
        always @(posedge i_aclk)
        if ((f_past_valid)&&($past(i_aresetn))
                &&($past(i_tvalid))&&(!$past(i_tready)))
        begin
                `SLAVE_ASSUME(i_tvalid);
                `SLAVE_ASSUME($stable(i_tstrb));
                `SLAVE_ASSUME($stable(i_tkeep));
                `SLAVE_ASSUME($stable(i_tlast));
                `SLAVE_ASSUME($stable(i_tid));
                `SLAVE_ASSUME($stable(i_tdest));
                `SLAVE_ASSUME($stable(i_tuser));
                // Issue III: Assert failed in assert_SRC_STABLE_TDATA (FIX)
                `SLAVE_ASSUME($stable(i_tdata));
```

Figure 16: The TUSER width parameter in the faxis_slave component.

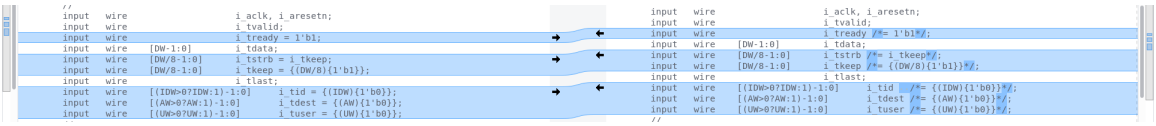
Appendix A: Differences of the Local Copy of faxis_slave

The faxis_slave used in this demonstration was obtained from symbiotic-20200902A SEDA release, and it was modified as shown below with the purpose of making it synthesizable, using Verific's feedback:

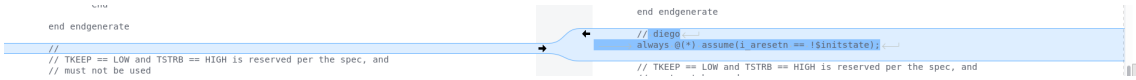
- Lines 1 to 14, comments of the errors shown by Verific:



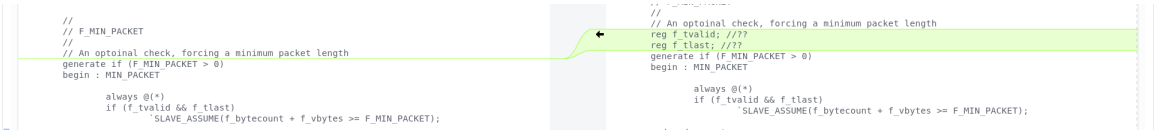
- Lines 80, 82-83, 85-87, removing non-ansi port declaration:



- Lines 158 to 160, adding $initstate, a Yosys extension to avoid looking traces generated during reset:



- Lines 255 to 256, declaring f_tvalid and f_tlast because they had not been declared anywhere:



- Line 246, renaming undeclared stall_count to declared f_stall_count: