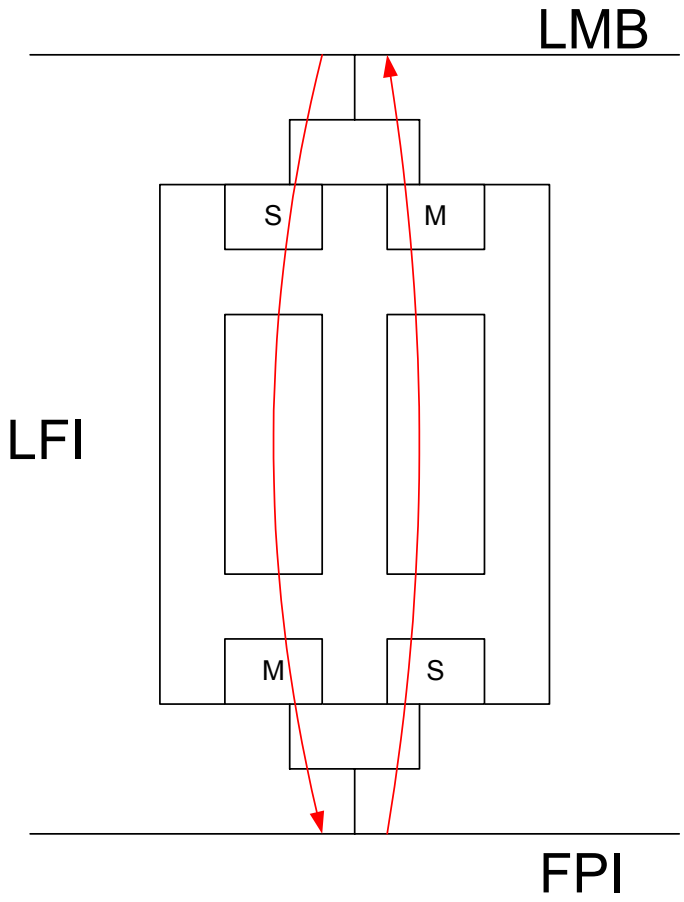
Bus Bridge Verification

Darren Galpin

Infineon Technologies UK Ltd

The LFI – LMB-FPI Interface



Two-way bridge between two Infineon busses. Supports

- Pipelining
- Early termination by masters, slaves and arbiter
- Retries and splits
- Read-Modify-Write Operation
- Different clocks on each bus

Around 30k gates

Why is this difficult to verify?

- Each bus has:
  - 4-bit opcode (16 possibilities) – single and burst types.
  - 2-bit acknowledge code (4 possibilities) – Rty, Split (LMB only – Res on FPI) , ERR, OK.
  - Read, write and RMW transaction types
- LMB has 32-bit address bus, and 64-bit data bus. FPI has 32-bit address bus, and 32-bit data bus. LFI will have to translate data and opcodes.
- Can have any clock speed on FPI as long as edge is coincident with the LMB clock.
- Not only verifying that the device obeys the two protocols, but also need to test the *interaction* between the two protocols, especially as the LFI can hold transactions within its pipelines.

Directed Testing – Approach 1

- Test bench
  - HDL Bus Functional Models (BFMs) for LMB and FPI masters and slaves.
  - Protocol Checker on each bus.
  - Regression of tests.

- Coverage
  - Compliance with predefined test specification
    - Written by verification team.
  - Various metrics (statement, branch, toggle) with predefined targets (100% or near).
    - May require white-box tests.

Directed Testing – Approach 1

Advantages

- Exact control over test and what is driven.
- Easy to code a test to hit a specific point.

Disadvantages

- Difficult to hit all corner cases – would require a lot of hand written transactions to hit all combinations.
- Limited to scenarios thought of by the verification engineer. What about those not though of?
- Relies of structural coverage. This only tells you what you haven't tested, not what you have.

Pseudo-Random Testing – Approach 2

Used on a previous project where 100% statement and toggle coverage had been achieved – found over 30 more bugs!

Test bench

- 'Verification Components' for LMB and FPI masters, slaves, protocol checkers and arbitration units.

Coverage – Functional

- Transaction Coverage. Cross-cover each transaction with each termination condition, byte lane, wait cycles etc.
- FSM transition. All legal transitions in the internal state machines occur.
- May require directing of tests.

Functional coverage is based on expected behaviour of the device, either at the interface (black box) or internally (white box). It does not take into account code structure.

Pseudo-Random Testing – Approach 2

Advantages:

- Run far more tests.
  - Number of transactions is higher due to automation - 32.5 million.
  - Achieve functional coverage targets quicker than directed testing - not as tedious, and less prone to human error.
- Scenarios ran not limited by pre-conceived notions of what should be tested and the device operation.

Disadvantages:

- Requires increased compute power to run increased number of simulations.
- Functional coverage is only as good as the coverage points written.
- Still need to direct tests to hit certain coverage points.

Formal Verification – Approach 3

First use of the approach. Mathematically based, and should offer 100% proof of what you are testing.

Test bench

- Constraints. Inputs to obey bus protocol.
- Properties
  - Bus compliance.
  - Transactions complete correctly.

Coverage

- Properties undergo independent review.

Formal Verification – Approach 3

Limitations:

- Property checker considers only finite time windows and does no reachable state analysis. So choose from two possibilities:
    1. User needs to define reachable states. Constraints on internal signals.
    2. Start from known state - reset.

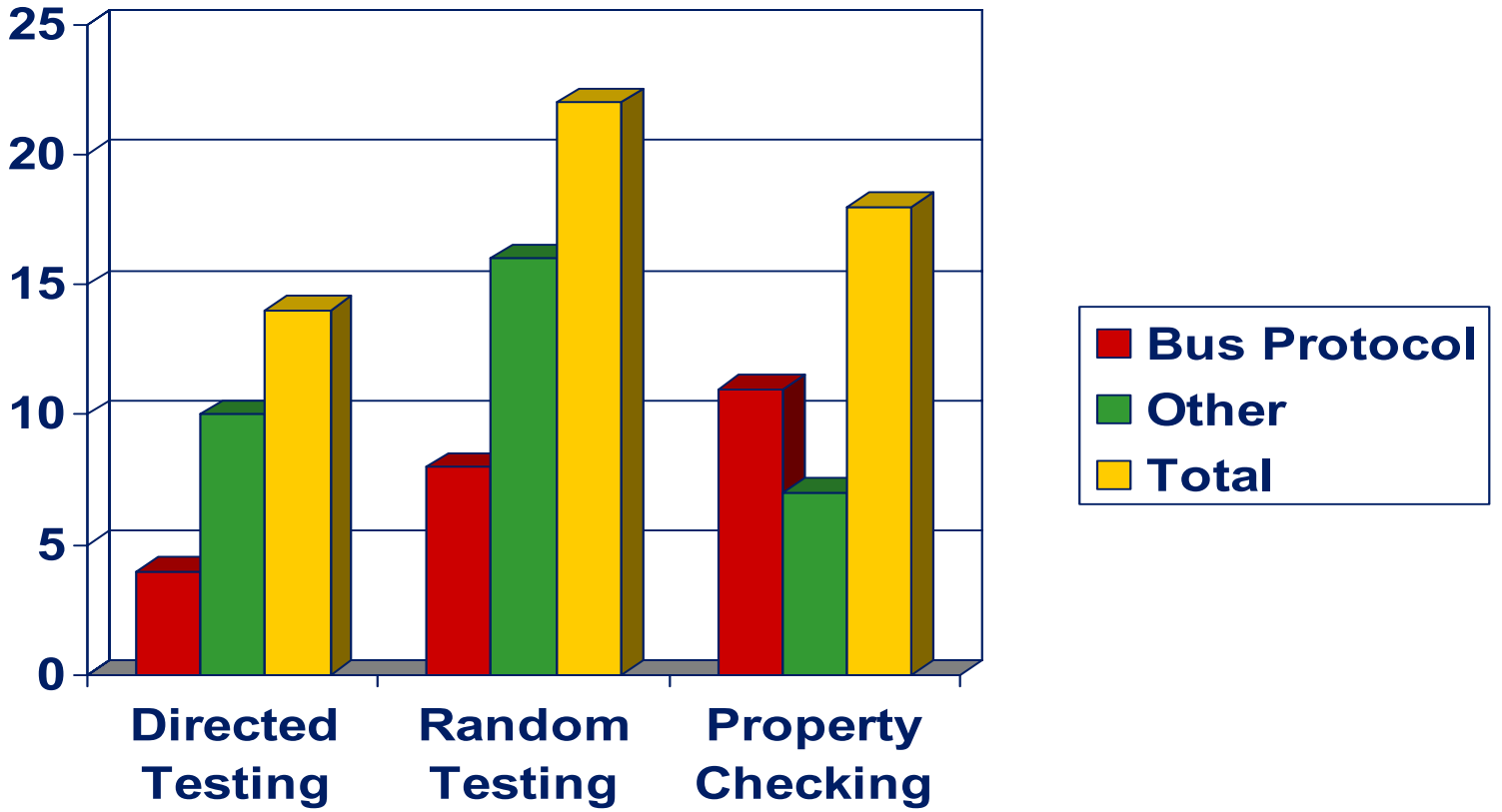
Took latter approach

- Design had complex internal dependencies (7 interacting FSMs, FIFOs with complex pointer behaviour).
- Maintaining internal constraints across design changes proved time-consuming.

Advantages:

- Extremely thorough verification of design just after reset.
- Bus protocol properties: for all legal/allowable inputs the design obeys bus protocol.
    - Bugs resulting in breaches of bus protocol manifest themselves close to reset?
- Transaction properties: check first (pair of) transaction(s) after reset.
    - No restrictions on other legal bus behaviour.
    - Symbolic check that addresses/data pass through bridge correctly.

The Bug Chart



No bugs found since design release.

That was then, but what now?

Assertions

- Single line checks within the code which express design intent.
- Used to check interfaces and critical internal logic where assumptions have been made.
- Can be written by both verification and design.
- Can be run in any simulator (with OVL. With PSL too “soon”).
- Are inherited with the block into systems.

Re-use eVC’s at the system level

- Re-using FPI eVC’s on TC2SV to generate traffic and protocol checking.

That was then, but what now?

Combine the techniques

- Simulations (directed and random) are susceptible to ‘bug hiding’ – you may find a bug and fix it for that test, but have you really fixed it in all cases, or simply moved the entry condition, or not fixed all entry conditions? Property checking is more persistent, so deduce properties from failing simulations to prove completeness of fix.
- We found that scenarios missed in the directed test specification were found (or suggested) by failing random tests. These can then be added to the directed test regression suite (although we abandoned directed testing entirely for the next bus bridge).
- When designers perform their code review, get them to suggest white box properties. Increases confidence in the design.
- Properties can be re-used as run time checks within the random environment. Allows you to explore the state space beyond the proof-radius with the property.
- Create coverage points on the assertions. Assertions only tell you when they have failed, but you need to know whether you have tested them or not.

The Challenges of the Job

1. Getting the tools to run together
   - Frequent software patches – far fewer Specman users than Windows users.
   - Have to cross-compile shared object libraries when using more than two programs together, e.g. VNavigator, Specman and NCSim.
2. Specification Changes
   - May be due to consequences of a bug or customer change. Need to ensure that verification environment reflects the latest requirements.
3. Multi-site working
   - Colleagues may be in Munich or Singapore. Need to ensure that everything (especially specifications) are clear and unambiguous. (You do not want to argue about the meaning of a comma, or shall…)!
4. Rapidly changing techniques and tools
   - New EDA tools released regularly with new methodologies. Need to be able to quickly sift through them to get the new ideas – you can always use methodology developed for one tool with another.
5. Designers
   - May see the verification engineer as being purely destructive. Need to convince them that you are working together.

The Challenges of the Job (Continued…..)

1. Managers.
   - You never ever have the time, budget or head-count……