Simplifying SoC Verification using a Generic Approach

David Robinson
Verilab

david.robinson@verilab.com

Overview

I'm talking about....

How Verilab are simplifying SoC verification

A self generating eVC for generic SoC verification

- **Provides automatic bus verification (infrastructure and interconnect)**
- **Provides a platform for the rest of the verification**
- **Provides mechanisms to isolate the tests from the design**

Overview

Something for everyone....

Part 1 (For managers)

- Solving my top four problems in SoC verification
- The value of this approach

Part 2 (For engineers)

- Technical overview

Verification Problem 1

Problem

It's hard to find good verification engineers who are also language experts

Solution

Remove the need to be a language expert

Verification Problem 2

Problem

It takes too long to write the verification environment

Solution

Automatically generate the verification environment from a high-level description

Verification Problem 3

Problem

Previous knowledge is not captured or reused

Solution

Build this knowledge into the verification environment generator

Verification Problem 4

Problem

Designers have a decision making problem

Solution

Accept that change will occur, and build the verification environment to cope

The Added Value

Cheaper projects

Better quality

Easier projects

Safer projects

Cheaper Projects

Projects cost less because....

- Fewer people are needed
- They can be less skilled
- The verification phase can be shorter
- There are minimal delays when change occurs

Better Quality Verification

Quality improves because....

- Automatic reuse of verification knowledge
- Tried and tested solution used

Easier Projects

Projects get easier because....

- Easier to find the right people
- Easier to get a verification environment running
- Many of the technical and methodological issues of setting up a complex testbench simply go away
- Reduced pressure on the verification team because they start finding bugs earlier

Safer Project

Projects get safer because....

- Uses pre-built and pre-verified verification environment that contains a wealth of project experience and expert knowledge
- Less chance of resource shortages
- Functional bugs caught early (while the designers are still working on the code)
- Change from the design team does not mean problems and delays for the verification team

Part 1: Summary

This is valuable

On the latest design we found the first RTL bug within *one day* of receiving the RTL and starting the verification

- 1xAHB multi-master bus
- 1xAHB Bus Matrix (multi-layer point-to-point)
- 2xAPB peripheral busses
- 3 bridges
- 6 masters
- 20 slaves

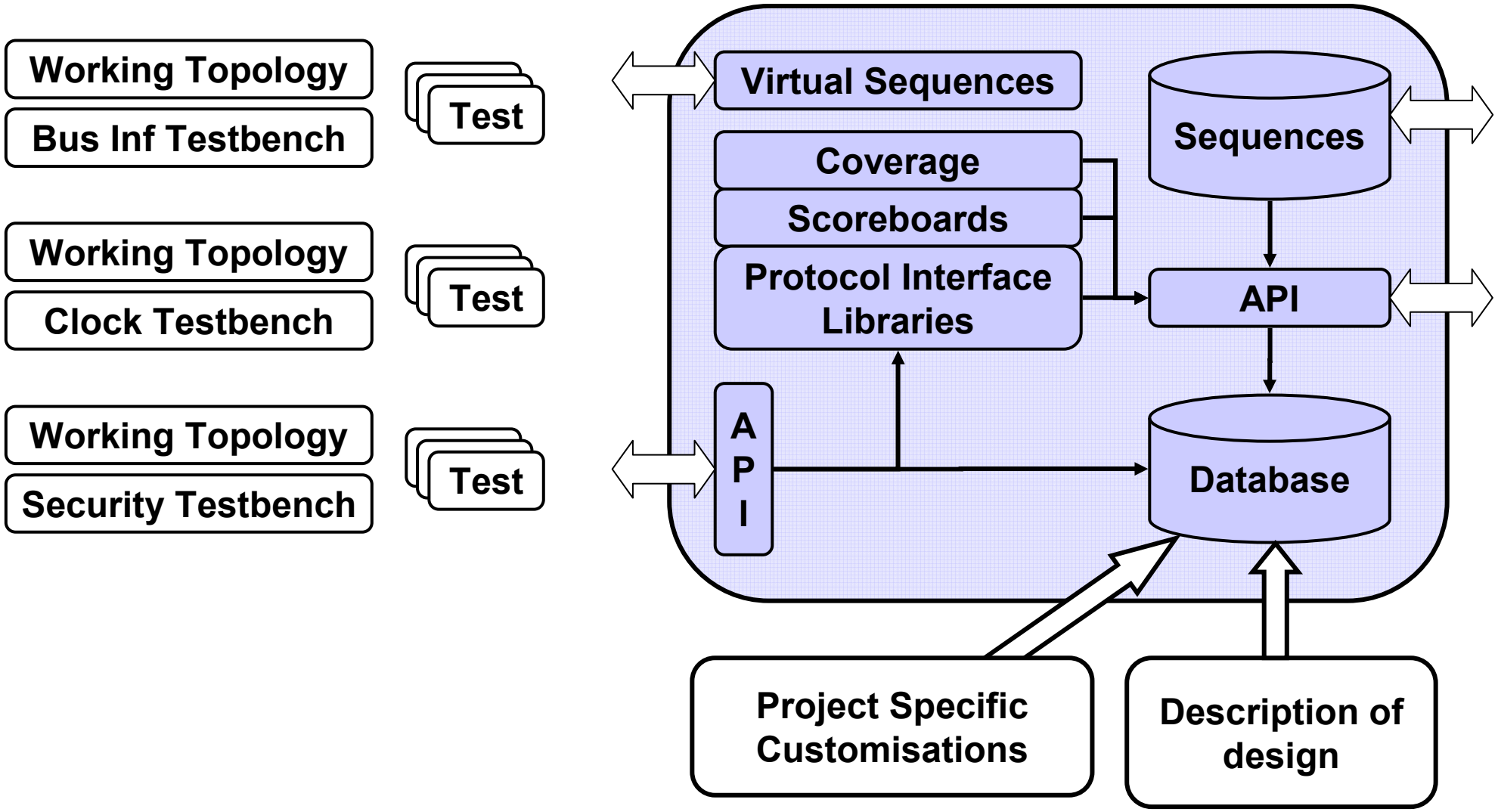
Part 2: Features

What does it give me?

- A high-level text format for specifying the design's topology, protocol limitations, address maps, access limitations, safe and unsafe memory regions, etc
- Instantiation of the required 3rd party bus eVCs
- Bus traffic scoreboards
- Customised functional coverage

- A sequence library for bus infrastructure tests
- APIs to access design information without knowing design details
- A working topology definition

eVC Usage and Structure



Defining an AHB Master

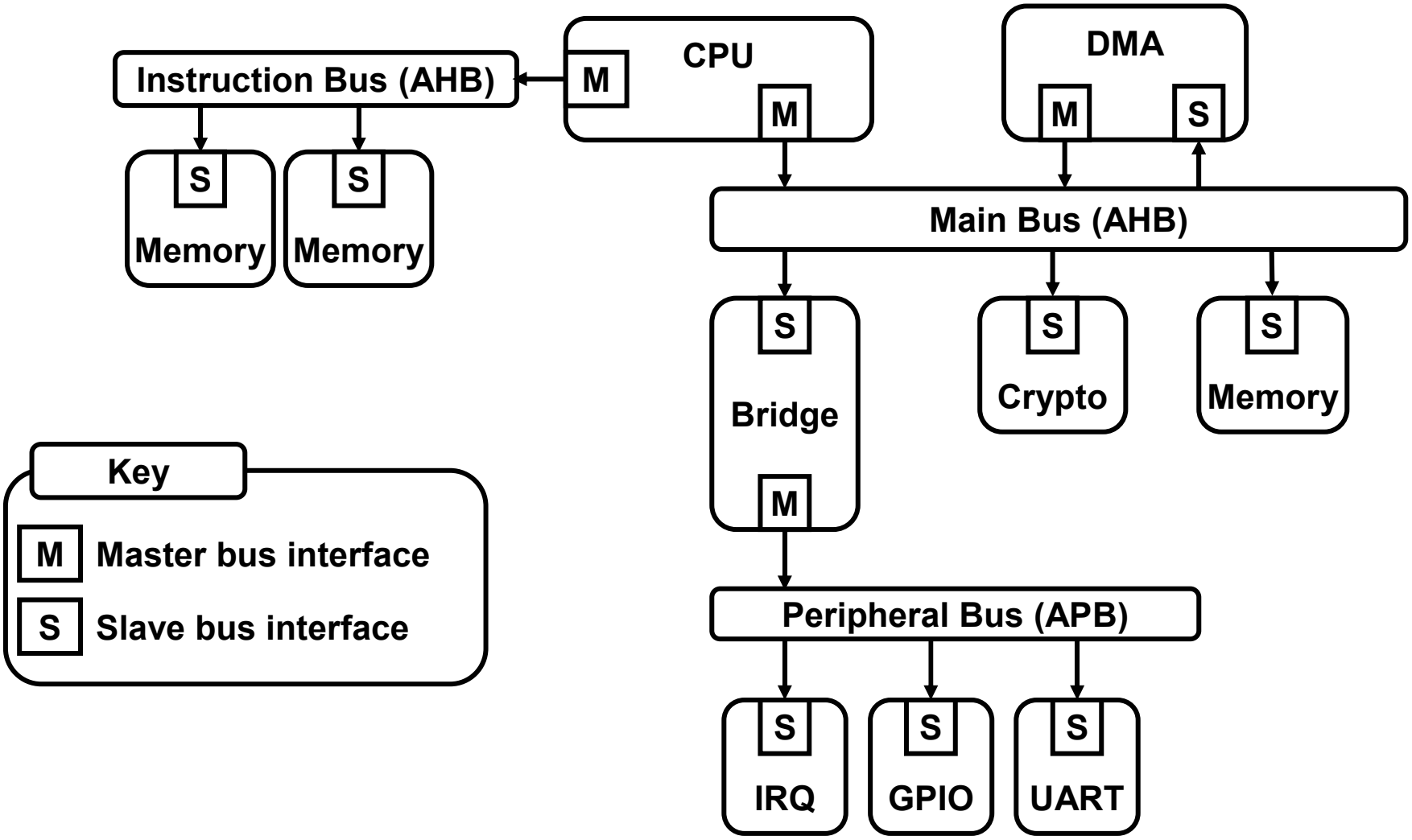
```
add master bus_interface dut_instance=<INST>{
 name      : CPU__INST_IF;
 component: CPU;
 bus       : INST_BUS;
 protocol  : AHB;

 <limitations>;
  add direction: WRITE;
 </limitations>;

 <routing_table>;
  SRAM__S_IF;
  INST_BUS__DEFAULT_SLAVE_IF;
 </routing_table>;
};

ahb master INST_BUS CPU__INST_IF 0 FALSE
                                  "hresetn" "hclk"
```

The Working Topology

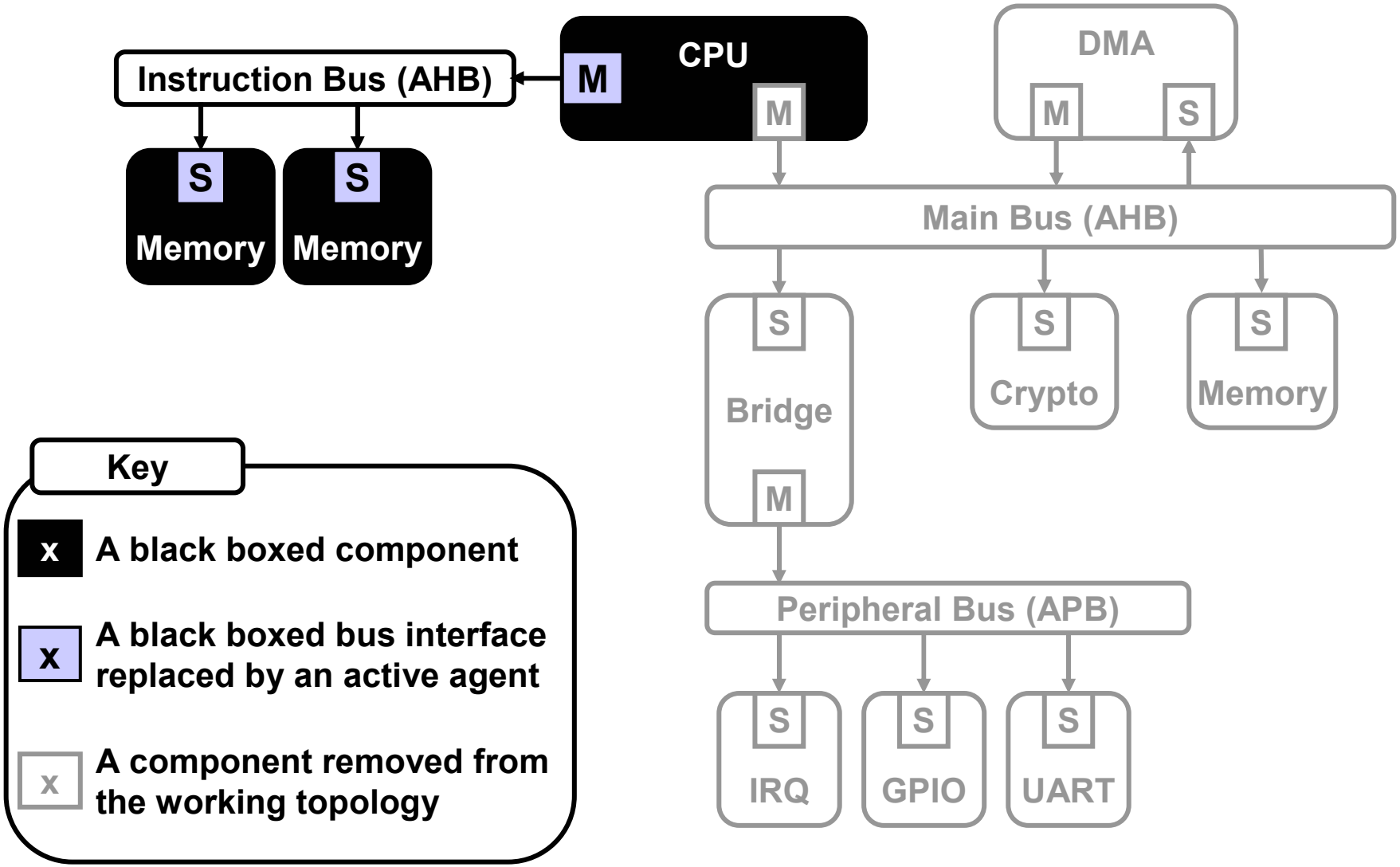
The what?

- Different testbenches need to see different views of the design
- Or alternatively: Different testbenches *do not* need to see *the same* view of the design
- The working topology file lets you specify:
  - Which subsystems are to be included
  - Which items are black boxed in the HDL
  - Which eVCs to use

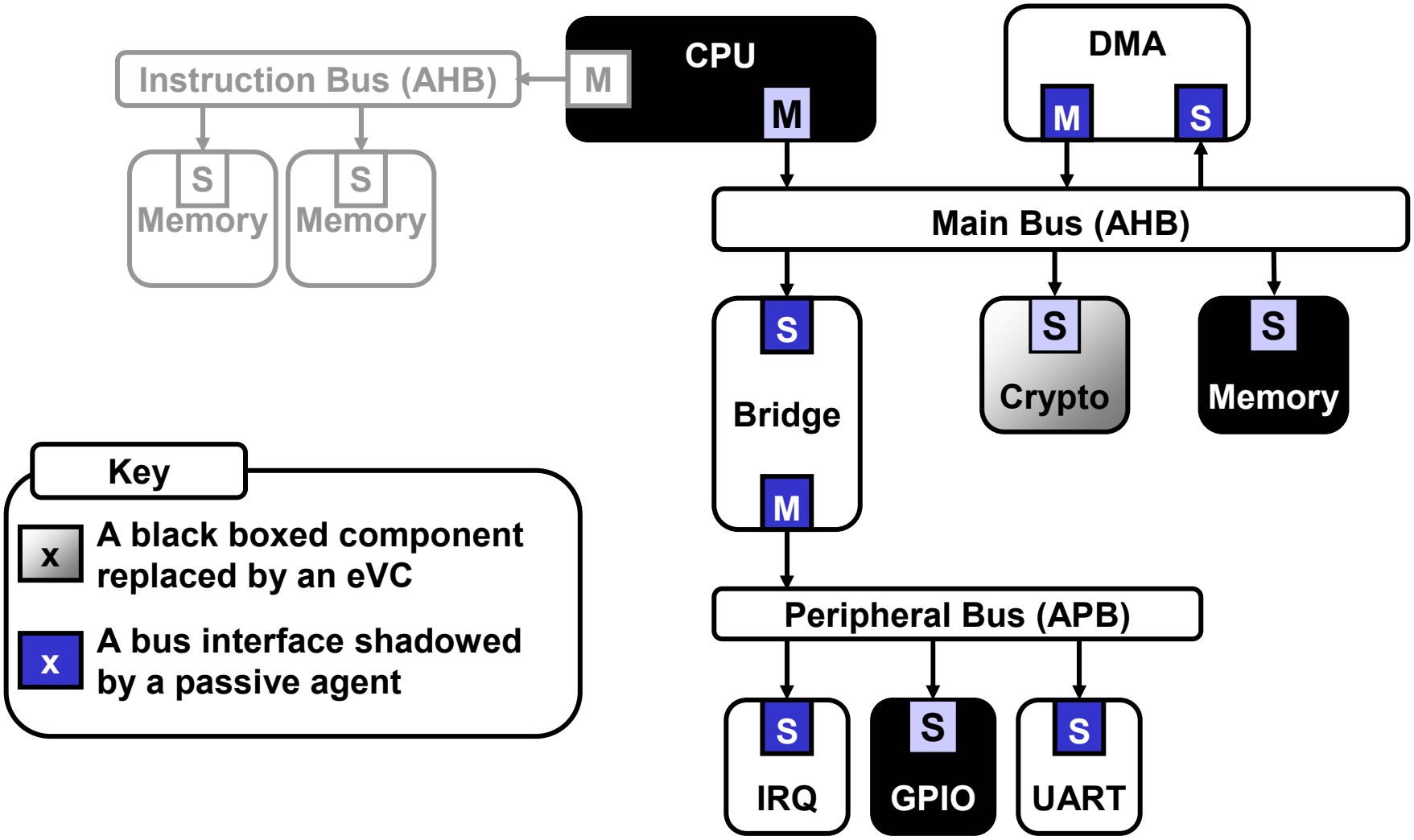
Example: Full Topology



Example: Instruction Subsystem Only



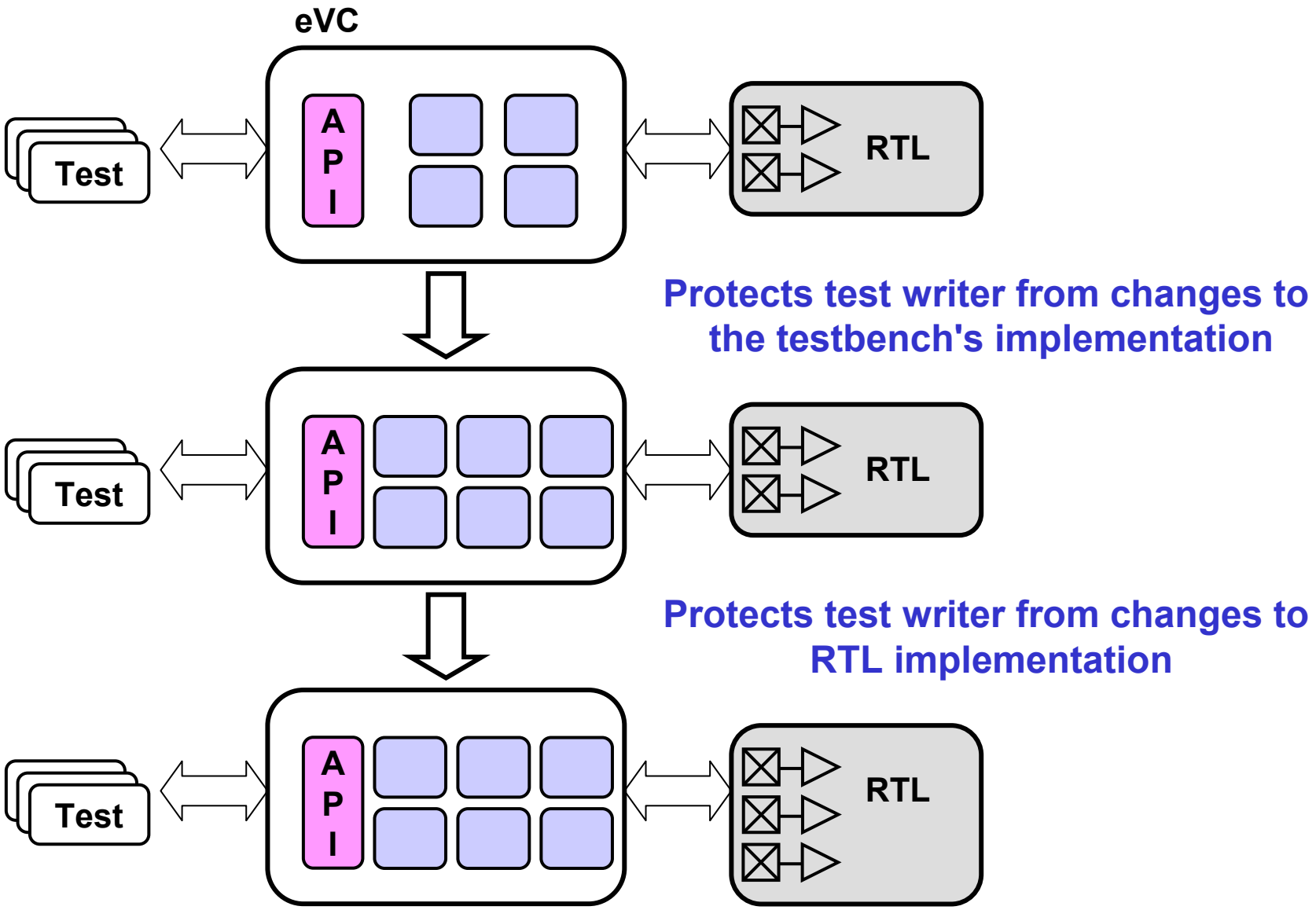
Example: Use Case Testing



Example Working Topology

```
<use>
   AHB  t;  // use the eVC t[rue]/f[alse]
   APB  f;  // use the eVC t[rue]/f[alse]
</use>

<subsystem>
   MAIN_BUS   t; // include the sub system t[rue]/f[alse]
   INST_BUS   f; // include the sub system t[rue]/f[alse]
   PERIPH_BUS t; // include the sub system t[rue]/f[alse]
</subsystem>

<topology_item>
   bus hdl_included MAIN_BUS;
   bus hdl_included INST_BUS;
   com hdl_excluded CPU;
   bif hdl_excluded CPU__DATA_IF;
   bif hdl_excluded CPU__INST_IF;
</topology_item>
```

Dealing with Change



Example API

Accessing testbench components

```
get_bus_sequence_driver(name)
get_interface_sequence_driver(name)
get_register_sequence_driver(name)
```

Accessing database information

```
get_bus_interfaces(filter)
get_address_ranges_for_slave(slave_name)
get_all_reachable_endpoints(master_name, filter)
get_all_masters_who_can_reach_me(slave_name, filter)
```

Summary

Current Status:

- Still adding new features
- Currently being used on three designs by different teams
  - 1-4 days to first simulation
- Attracting attention from other design teams
- Influencing eVC and RTL IP design guidelines

It's bringing real value to SoC verification projects

Download the paper from www.verilab.com