Generic Testbench/Portable Stimulus/Promotability

Revati Bothe
Jesvin Johnson

sondrel
success through partnership

SYSTEMS INITIATIVE

2019 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

- Introduction
  - Verification Environment - Reusability
- Generic Transactor testbench structure
  - SCEMI based Transactors
  - SCEMI pipes Communication
  - Immutable functions
  - Variable functions
  - Test case Promotability
- Generic re-usable testbench
  - Sondrel Image processing Subsystem verification architecture
  - Re-use across different platforms & Testbenches
- API for re-use
- Portable Stimulus
- Summary

Introduction

- With SoC complexity increasing many of the internal subsystems gets reused more often. This also opens an option to reuse the verification environment. At times some of the verification efforts can also be shared between various platform other than just Simulator. One may choose to emulate or have the design on an FPGA.

- Having a unified testbench for all these various platforms will make it easier to maintain the verification environment. UVM provides excellent capability of reusing the test sequence, providing greater flexibility such as overriding various aspects of the testbench via the UVM factory and these approaches can scale well across various levels of the design i.e. from IP to subsystem to system level.

- These verification components may not be reused well across multiple verification platform like and emulator as these transactor can't be realized or synthesized to these platforms.

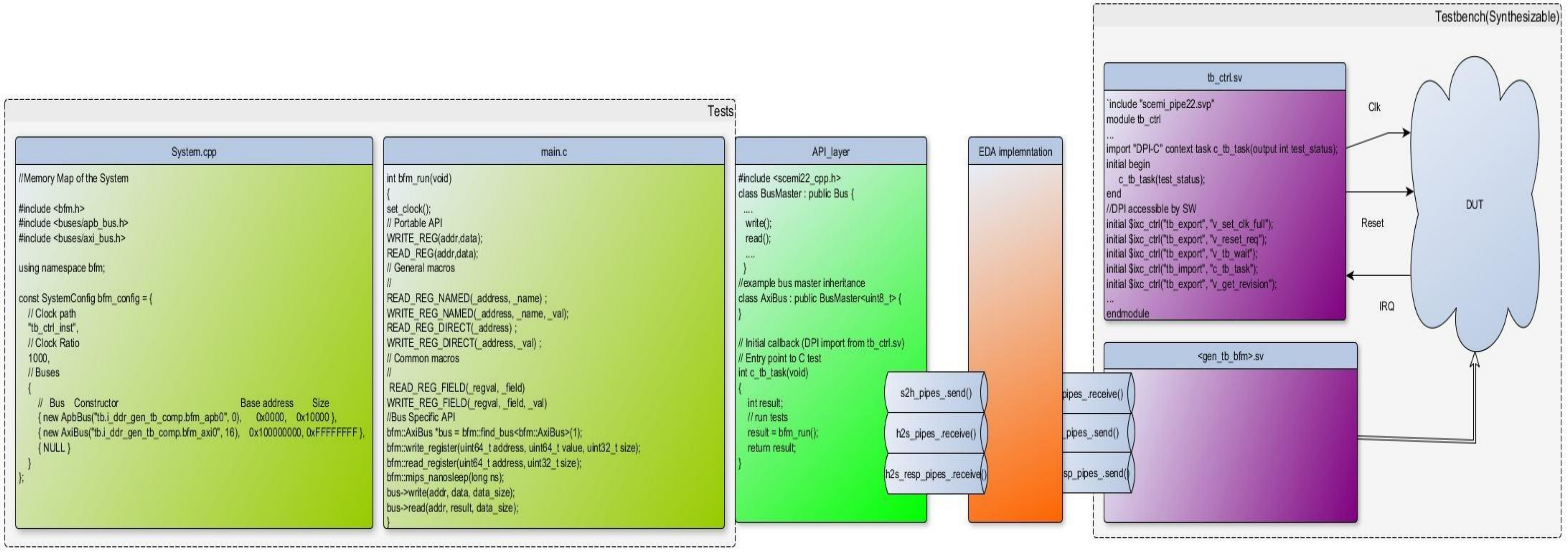
Introduction

- Accellera SCEMI standards addresses the above issue and provides the capability of having synthesizable Bus Functional Models that can be reused on various platforms and can scale well between IP level verification to system level verification.

- A SCEMI based transactors comprises of a BFM layer implemented in HDL which is synthesizable on platforms like FPGA and emulator and API layer that provides necessary interface to the test writer and this can be implemented in HDL or programming languages like C, C++ etc.

Generic transactor testbench structure(1/4)

Figure 1: Generic transactor testbench structure



System.cpp

```
//Memory Map of the System

#include <bfm.h>
#include <buses/apb_bus.h>
#include <buses/axi_bus.h>

using namespace bfm;

const SystemConfig bfm_config = {
   // Clock path
   "tb_ctrl_inst",
   // Clock Ratio
   1000,
   // Buses
   {
      //  Bus   Constructor                                         Base address      Size
      { new ApbBus("tb.i_ddr_gen_tb_comp.bfm_apb0", 0),      0x0000,    0x10000 },
      { new AxiBus("tb.i_ddr_gen_tb_comp.bfm_axi0", 16),   0x100000000, 0xFFFFFFFF },
      { NULL }
   }
};
```

main.c

```
int bfm_run(void)
{
set_clock();
// Portable API
WRITE_REG(addr,data);
READ_REG(addr,data);
// General macros
//
READ_REG_NAMED(_address, _name) ;
WRITE_REG_NAMED(_address, _name, _val);
READ_REG_DIRECT(_address) ;
WRITE_REG_DIRECT(_address, _val) ;
// Common macros
//
 READ_REG_FIELD(_regval, _field)
WRITE_REG_FIELD(_regval, _field, _val)
//Bus Specific API
bfm::AxiBus *bus = bfm::find_bus<bfm::AxiBus>(1);
bfm::write_register(uint64_t address, uint64_t value, uint32_t size);
bfm::read_register(uint64_t address, uint32_t size);
bfm::mips_nanosleep(long ns);
bus->write(addr, data, data_size);
bus->read(addr, result, data_size);
}
```

API_layer

```
#include <scemi22_cpp.h>
class BusMaster : public Bus {
  ....
   write();
   read();
   ....
  }
//example bus master inheritance
class AxiBus : public BusMaster<uint8_t> {
}

// Initial callback (DPI import from tb_ctrl.sv)
// Entry point to C test
int c_tb_task(void)
{
   int result;
   // run tests
   result = bfm_run();
   return result;
}
```

tb_ctrl.sv

```
`include "scemi_pipe22.svp"
module tb_ctrl
...
import "DPI-C" context task c_tb_task(output int test_status);
initial begin
    c_tb_task(test_status);
end
//DPI accessible by SW
initial $ixc_ctrl("tb_export", "v_set_clk_full");
initial $ixc_ctrl("tb_export", "v_reset_req");
initial $ixc_ctrl("tb_export", "v_tb_wait");
initial $ixc_ctrl("tb_import", "c_tb_task");
initial $ixc_ctrl("tb_export", "v_get_revision");
...
endmodule
```

Generic transactor testbench structure(2/4)

- Figure 1 depicts a typical generic testbench architecture testbench architecture illustrating the flow of transactions from testcase (C/C++ / SV) to Scemi pipes and ultimately to a synthesizable transactor (purple) which fetches transaction from Scemi pipe and drives the test environment .

- These scemi transactor based testbench contains an instance of a testbench control module which imports a DPI hdl2c() which is invoked within an initial block , similar to having a run_test() for accessing UVM test form testbench. The hdl2c() call is blocking and will transfers the execution thread control from HDL testbench side to C and executes the test sequence. C to HDL synchronizations are controlled via DPI or polling for certain status flags from a testbench register.

Generic transactor testbench structure(3/4)

- At the end of C test hdl2c passed control back to the testbench control (tb_ctrl) which then raises a shutdown request flag and waits for a shutdown acknowledge from test enviornment which is in granted state by default, unless its controlled via external threads (e.g. UVM run_phase) to schedule simulation. If no external threads are active tb_ctrl will immediately stop the simulation. Usually UVM run_phase withhold the shutdown acknowledge to tb_ctrl so that the UVM check_phase and report phase can perfrom additional analysis and generate status of the simulation.

- The test writer at each level focuses on implementing *actions* and multiple actions become a *scenario* or the test objective. An *action* can be a combination of immutable and variable function.

Generic transactor testbench structure(4/4)

- Immutable functions are set of process like for e.g. programming an IP for a certain mode of operation and this behavior will not change regardless of the platform or whether testing at IP level or system level.

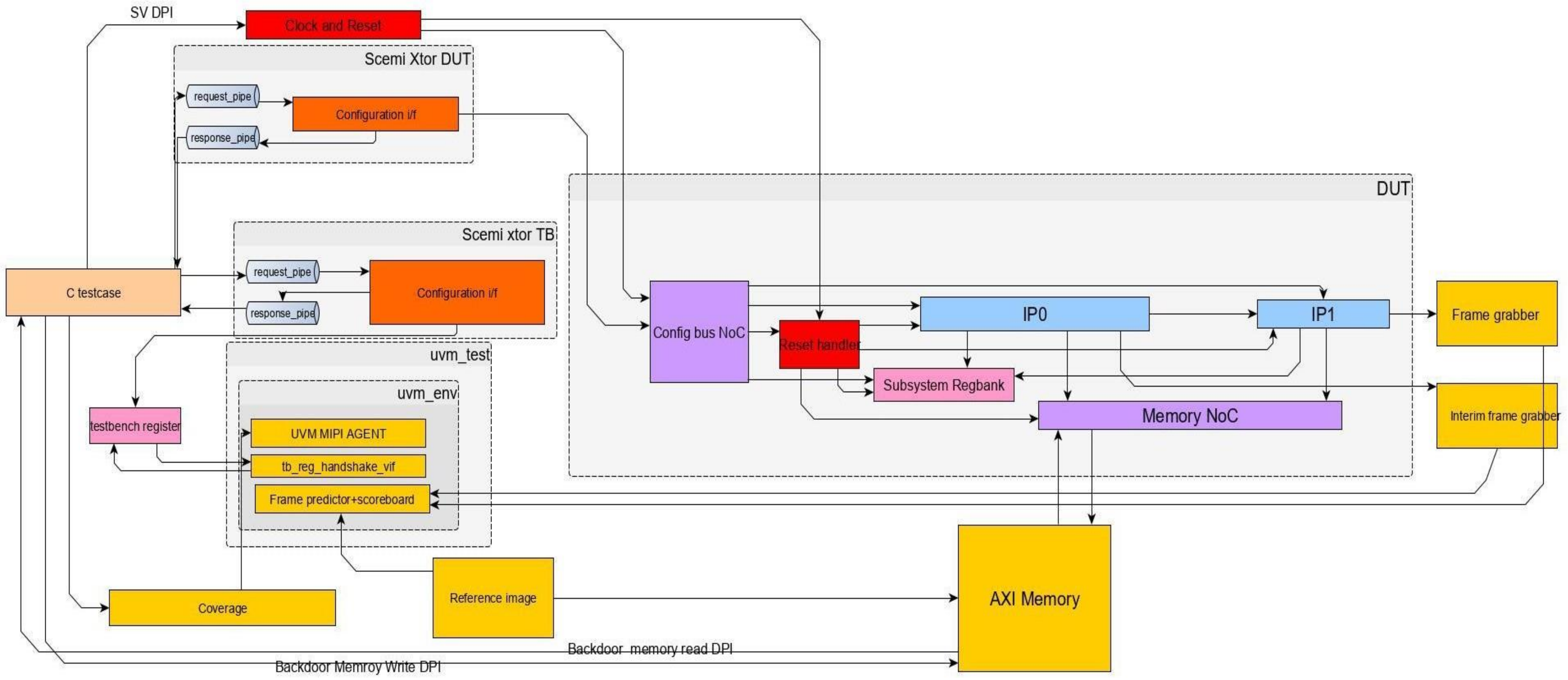
- A Variable function or hook function can morph its behavior based on platform or testbench. For example as a preamble to programming the IP one may choose to enable the clocks and bring the system out of reset, this process may vary depending on the testbench as at IP level this can be a signal connected straight to the I/Os of the IP, although at the subsystem level there might be a clock gate and some external clock controller register needs configuring to enable the clock to IP . Exposing these hook function will provide the flexibility to adapt the action for a given platform. Details on example implementation will be discussed below.

Generic re-usable testbench (1/6)

- In this section we will explain a re-usable generic testbench that we have used to verify Subsystem and SOC. The subsystem here is an Image Processing Subsystem.

Generic re-usable testbench (2/6)

Figure 2: Typical Subsystem Verification Testbench



Generic re-usable testbench (3/6)

- Key:

  - Light Blue: IPs
  - Red: Clock and Reset Blocks
  - Orange: Configuration Interfaces (Synthesizable SCEMI BFMs) Yellow: Testbench elements
  - Pink: Register Banks
  - Peach: C testcases
  - Purple: interconnect

- Figure 2 depicts block diagram of Subsystem verification architecture. The IP in case of this particular Image Processing Subsystem would be delivering the IP level tests and these tests will be re-used at subsystem and SoC level with some modifications, like commenting the commands which are related to IP's internal data generator since we will be using the external imager in form of UVC/models. We also commented the 'test models' related commands because we will not be using the IP delivered test models at the Subsystem, We had to add some additional commands required to access the testbench registers used for the synchronization between the software and the external imager sequence (uvm) in case of Subsystem and SoC Level.

- Additional verification code was to configure various subcomponents and backdoor access to IPs eg descriptors. A C framework was used for the backdoor access in the testbench. Also, C testcases are written in a way that the CPU can run them, however during early integration testing, not all testcases have the CPU live. C part was written with consideration of generic test bench approach so that same tests can be used for FPGA/Emulator with little modifications.

Generic re-usable testbench (4/6)

- In order to provide a common verification environment across different SoC, subsystems and different platforms some generic reusable testbench components have been developed The standard generic components are synthesizable BFMs for AXI4 interfaces, Company Standard Interfaces.

- These BFMs are bus masters and can drive standard slave interfaces that are compliant with AXI4 and Company Standard Interface. Additional reusable components developed were for clock and reset generation (tb_ctrl),

- In order to re-use tests and test sequences, the Subsystem test sequence are layered where the platform, the testbench and the subsystem specific code are structured in layers. This allows seamless porting to different platforms like an FPGA or emulator, ports to a different testbench as well, without expecting any changes to be made to the tests

Generic re-usable testbench (5/6)

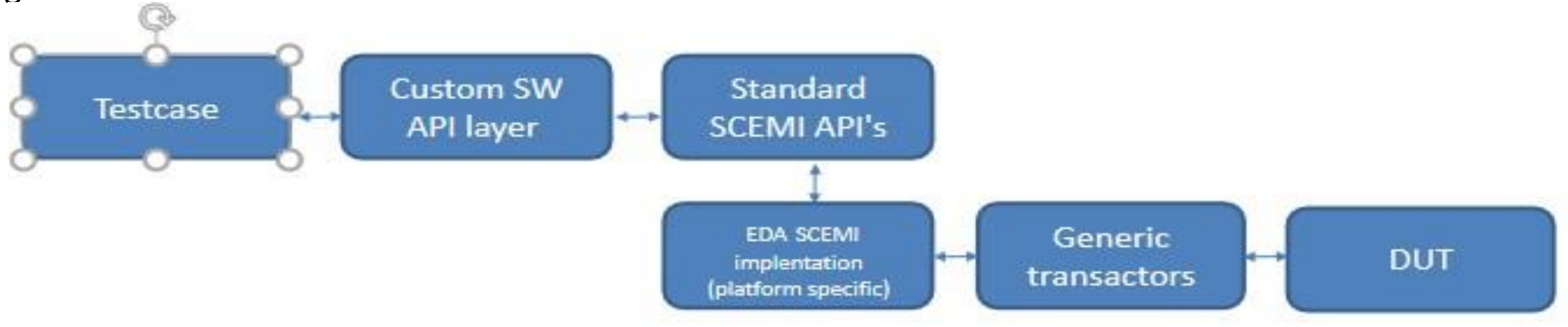
- At the SoC Level some additional verification code is required to configure various other subcomponents and a C test framework will be used for the same. C tests will be written with consideration of generic test bench approach so that same tests can be used for FPGA/emulator with little modifications. This includes use of SCE-MI pipes.

- Verification environment at SoC level will have additional memory interface, Vendor specific master/slave port and some IP port will be connected to Memory Data Interconnect. Register interface Interconnect was connected to Configuration Data Interconnect

Generic re-usable testbench (6/6)

The figure 3 shows a typical case of re-use across different platforms and testbench.

Figure 3: API for Re-use



PORTABLE STIMULUS (1/2)

- Our paper addresses the underlying infrastructure which Portable stimulus standard compliant verification synthesis engine will use

- Portable stimulus defines high-level verification intent. A verification synthesis engine consumes that description and creates test cases targeting different execution environments, such as simulation, emulation, hardware prototypes, and real silicon.

- Portable Stimulus enhances the existing SV/UVM methodology. It targets systems and the interaction between hardware and software. At the block level, it's possible to combine the strengths of both languages—SV/UVM provides transactors and legacy verification intellectual property (VIP).

PORTABLE STIMULUS (2/2)

- Portable stimulus standard is a set of semantics for a Verification Intent Model and from that model tools will be able to synthesize testbenches that could target any number of execution engines. The model captures the intended behaviors of the design in a way that complete testbenches can be generated, including stimulus and checkers.

- The generated testbench could either drive UVM models, or could generate code that executes on the embedded processors or Python/C languages driving bus masters like embedded processors or a combination of both. Results of the run can be annotated onto the model for notions of system-level coverage. And the creation of those testbenches also includes notions of randomization, so nothing is lost in terms of methodology.

- Our future work will involve how to map the verification intent model into our generic tb based verification infrastructure.

Summary

- We discussed about –

- Verification Environment - Reusability
- Generic Transactor testbench structure
  - SCEMI based Transactors
  - SCEMI pipes Communication
  - Immutable functions
  - Variable functions
  - Test case Promotability
- Generic re-usable testbench
  - Sondrel Image processing Subsystem verification architecture
  - Re-use across different platforms & Testbenches
- API for re-use
- Portable Stimulus

Questions