Reuse C test and UVM sequence utilizing TLM2, register model and interrupt handler

HongLiang Liu  andy.liu@amd.com
Teng-Fei Gao  teng-fei.gao@amd.com
Abstract – Where we are

• Reuse C test case and UVM sequence in SOC

• Reuse happens **at three layers** in this paper
Problem Statement

• how to reuse SOC C test cases when system bus protocol changes?

• how to reuse IP UVM sequence to SOC verification environment?
Prior Work

• When SOC’s system bus protocol changes, such as from PCIE to AXI, SOC C based test bench need be updated

• IP UVM is **passive** in SOC verification environment, not **active**
  – SOC verification engineer creates new C test cases to verify new IP’s register and interrupt
Approach / Ideas

• test stimulus is software, think in software way

• a simple communication software prototype

Diagram:

- Application layer
- Transaction layer
- Physical layer
Three-layer C-UVM Reuse Structure

Application Layer

C Test

Transaction Layer

PCIE_TLM2 API
send()/receive()

UVM Sequence

Bus Layer

PCIE BFM

PCIE iUVC

AXI iUVC

interface UVC

fully reuse
Three-layer C-UVM Reuse Structure

- C test sits on top layer, it is isolated from bus layer by transaction layer, thus, C tests are fully reused regardless of system bus protocol.

- Transaction layer is implemented with TLM2, including **general part** and **protocol part**.

- Bus layer is a **interface UVC**, reused from IP UVM environment.
Transaction Layer Flow

PCIe Struct → TLM2 Generic Payload → Bus Protocol Extensions → UVM Bus Sequence Item → iUVC

tlm_generic_payload *ptr_tlm_trans = new tlm_generic_payload;
gp_extensions_axi *ptr_tlm_extensions = new gp_extensions_axi;

ptr_tlm_trans->set_write(pkt.is_write());
ptr_tlm_trans->set_address(pkt.addr);
ptr_tlm_trans->set_data_length(pkt.len);
ptr_tlm_trans->set_data_ptr(pkt.data);

ptr_tlm_extensions->set_id();
ptr_tlm_extensions->set_burst();
ptr_tlm_extensions->set_cache();
ptr_tlm_extensions->set_user();

Protocol scalable

general reuse
Simple C and UVM Interactive

• C is responsible for simulation ending control

• C packet is converted to UVM sequence item in transaction layer with TLM2

• C test case and the reused IP UVM sequence run on the same iUVC
iUVC works in **active mode** at IP block level

iUVC works in **passive mode** or **active mode** at SOC level
iUVC Reuse

- **active mode:** executes UVM sequence, generates master request or slave response stimulus
- **passive mode:** monitor, protocol checker, coverage collector
iUVC uses virtual interface to connect RTL and UVM
Connection Reuse

- Connect **static RTL world** and **dynamic UVM world**
  - **static RTL world:** real interface
    - **dynamic UVM world:** virtual interface
  - bind the real interface to RTL module
  - `uvm_resouce_db` pass the real interface handle to virtual interface
Connection Reuse

- Bind connection separate s DV code from RTL, better for reuse
- Code segments for bind and uvm_resource_db

```verilog
bind `AMBA_MODULE svt_axi_slave_if axi_slave_if
(   .araddr(AXI_SLV0_araddr),

module
initial begin
uvmpkg::uvm_resource_db#(virtual
  svt_axi_if)::set("interface_registry ",`AMBA_MODULE.axi_slave_if, `AMBA_MODULE.axi_slave_if);
end
endmodule

uvm_pkg::uvm_resource_db#( virtual svt_axi_if)
  ::read_by_name(" interface_registry ",`AMBA_MODULE.axi_slave_if, axi_slave_if)
```
Problem Statement

• how to reuse SOC C test cases when system bus protocol changes?

• how to reuse IP UVM sequence to SOC verification environment?
UVM Sequence Reuse Challenge

- both sequence and sequencer are adhered to sequence item type – sometimes, a bus protocol

```verilog
module ip_uvm_seq#(ip_sequence_item)
  extends uvm_sequence;
endmodule

module ip_uvm_sequencer #(ip_sequence_item)
  extends uvm_sequencer;
endmodule
```

- If IP and SOC bus protocol are **same**, IP sequence can be directly reused to SOC
- If IP and SOC bus protocol are **different**, only general IP sequence can be reused to SOC
  - RAL register sequence
  - Interrupt sequence
adapter is the key point when reuse RAL register sequence
RAL Register Sequence Structure

```java
ip_reg_model.reg0.write(status, value, .parent(this));
```

- access a register - reg0 by register model hierarchy

```java
ip_reg = reg_model.get_reg_by_name(reg0);
ip_reg.write(status, value, .parent(this));
```

- access a register – reg0 by get_reg_by_name()
- make sure the register name is unique
IP Interrupt Sequence Structure

- **uvm_resource_db** is static and globally visible.
- **uvm_resource** provides `wait_modified()`.

```cpp
uvm_resource #(int) intr_rsrc;
intr_rsrc = uvm_resource_db#(int)::get_by_name("","int_id0");
intr_rsrc.wait_modified();
```
SOC Interrupt Sequence Structure

- Replace IP interrupt monitor with SOC interrupt monitor
- Interrupt service sequence are directly reused from IP block to SOC
Summary

- SOC engineer doesn’t need modify legacy C tests, when system bus changes its protocol

- SOC engineer can directly reuse IP UVM RAL register sequence, interrupt sequence, save effort to create new C tests

- With this approach, IP UVM environment is active in SOC now.
Acknowledge

- Bodmer, Thomas
- Whiting, Karl
- Chen, Beryl
- Zhang Japheth
- Yu, Rocky
- Gong XiaoHua