UVM-FM: Reusable Extension Layer for UVM to Simplify Functional Modeling

Ahmed Kamal
Mentor, a Siemens Business
ahmed_kamal@mentor.com
Introduction

The Universal Verification Methodology (UVM) is a powerful standardized tool for implementing a reusable verification environment, especially for single-layer protocols such as memories and simple communication buses. However, layered protocols have become key elements in the SoC industry; where the protocol defines several layers that contribute together to perform the overall protocol functionality. Most industry modern protocols, such as PCIe, SATA, SAS, and UniPro, are designed with this stacked approach.

UVM does not standardize how to implement a verification environment for layered protocols, nor does it recommend implementing the verification environment using either a single agent or multiple agents, where each agent represents a layer from the protocol stack. Consequently, several approaches have been proposed to address this lack of a standardized solution. This presentation evaluates some of the suggested approaches and introduces our proposed solution (UVM-FM).

Paper Subject :
- How to functionally model stacked protocols using UVM
- How to standardize the functional modeling of stacked protocols
Stacked Protocols Overview

A lot of the modern protocols are stacked (multi-layered)

- Some have a management layer
- Others have a tree structure

SATA Simple Stack

UniPro Stack with Management Layer

SAS Stack with Tree Structure

Complexity of Functional Modeling
Stacked Protocols Modeling
Related Work: Multi-Agent Approach

- Complete agent modeling for the leaf layer
- Complete agent modeling for the non-leaf layers
- The driver in non-leaf layers starts sequences on a lower-layer pass-thru sequencer
- Layer modeling is done inside the driver
- This approach manages the synchronization between the downstream and upstream paths
- Good approach for individual layers and complete stack verification

Open Questions:
- Is it a suitable approach for modeling protocols with management layer or tree structure?
- Is it scalable enough?

Reference:
Stacked Protocols Modeling
Related Work: Translation Agent

- Complete agent modeling for the leaf layer
- Uses non-attached to driver sequencers to forward the upper-layer sequence to the lower-layer sequencers after translation
- Uses translation sequences per each layer to model the layer functionally
- Uses the monitor path to translate leaf-layer-detected sequence items to upper-layer sequence items
- Simple technique without the overhead of implementing an agent for each layer

Open Questions:
- Is this approach helpful in individual layer verification?
- Is synchronization between the downstream and upstream paths easy?
- Is it a suitable approach for modeling protocols with management layer or tree structure?

Reference:
- Tom Fitzpatrick, “Layering in UVM”, Verification Horizons
- Doulous, “Requests, Responses, Layered Protocols and Layered Agents”, Online resources
Stacked Protocols Modeling

The UVM-FM Approach

- Backward compatible extension layer for UVM
- Aims to standardize the functional modeling architecture for the stacked protocols
- Changes the agent architecture to a network-like topology
- Defines UVM communicator node: a new UVM component that acts like a network router
- Defines UVM communicator node, which manages intra-communication for each layer and the inter-communication between different layers
- Re-defines the role of UVM driver to be like an interface between two layers
- Introduces the concept of Node/Block, which is a UVM component attached to a communicator node
- Node/Block could do one of the following roles: traditional driver, traditional monitor, or an internal block inside the layer
UVM-FM Approach
Communication Inside the Layer

- Defines the concept of a standard UVM sequence item that contains the message type and its properties
- If the user has another data structure, it could be packed and sent inside the message payload field
- Each node has a unique name inside the layer
- When a node/block wants to communicate with another node inside the layer, it writes the destination node name inside the message and defines the message type
- The block is connected to the communicator node through a pair of TLM ports
- The block sends the outgoing messages by using predefined subroutine: send_msg()
- Each block has a predefined subroutine called message_decoder(), which is called automatically when a message is received by the block
- User should override message_decoder() and decide needed actions for different received message types

The Standard Sequence Item for UVM-FM

MSG   ARGS   SRC   DST   SIZE   PAYLOAD

Simple Stacked Protocol Modeled by UVM-FM
Example of sending messages inside the layer: Block B sends a message to Block C
UVM-FM Approach
Communication between Layers

- When a block wants to send a message to a lower or upper layer a new message is constructed.

- The destination field inside the message holds the name of the driver that is responsible for communicating with the adjacent layer: “TUL” (to upper layer driver) or “TTL” (to lower layer driver).

- TUL and TLL forward the message to the corresponding layer.

- When “FUL” (from upper layer) driver or “FLL” (from lower layer) driver receives a message, it forwards the message to the corresponding node using the communicator block.

- FUL and FLL have a user programmed lookup table used to decide which layer node to forward the received message to.

UVM-FM standardizes the communication between the different layers, which simplifies modeling complex layering structures.
UVM-FM Package

- UVM-FM Package contains:
  - `fm_message` implementation: used for the standard message
  - `fm_component` implementation: used for the nodes/blocks
  - `fm_communicator` implementation: used for the communicator node
  - `fm_driver` implementation: used for the driver interfaces attached to the a communicator node
  - Easy-to-use SystemVerilog macros to simplify the system connections
  - Logging mechanism

This paper defines the specifications of the package but there is no open source implementation for it.
Example Code

Building the communicator node

```cpp
class user_comp extends fn_component;
    `uvm_component_utils(user_comp);
    event e_M1_is_here;
    event e_M2_is_here;
    function new(string name = "", `uvm_component parent = null);
        super.new(name, parent);
    endfunction
    //msg decoder - user should override this function to inform the node
    //how to handle the incoming messages
    virtual function msg_decoder(int msg_type);
        case (msg_type)
            M1 : if (M1_is_here);
            M2 : if (M2_is_here);
            default: begin
                $error("[Comp B] : unexpected msg type", msg_type);
            end
        endcase
    endtask
    virtual task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            task begin
                fork
                    if (M1_is_here);
                        $display("[Comp B] : receiving M1");
                end
                begin
                    if (M2_is_here);
                        $display("[Comp B] : receiving M2");
                end
            endjoin
            // sends a message to the upper layer (TUL)
            // sends the req to the communicator to get handled by this layer
            seq_item_port.get_next_item(req);
            seq_item_port.item_done();
        end
    endtask
endclass
```

Building the agent

```cpp
class my_full_driver extends fn_driver #(`uvm_msg);
    `uvm_component_utils(my_full_driver);
    // Standard UVM Methods:
    function new(string name="my_full_driver", `uvm_component parent = null);
        super.new(name, parent);
    endfunction
    virtual task run_phase(uvm_phase phase);
        super.run_phase(phase);
        //reqs are sent to nodes A and B
        register req, msg;
        forever begin
            seq_item_port.get_next_item(req);
            seq_item_port.item_done();
        end
    endtask
endclass
```

Building a system Node/Block

```cpp
class my_tul_driver extends fn_driver #(`uvm_msg);
    `uvm_component_utils(my_tul_driver);
    // Standard UVM Methods:
    function new(string name="my_tul_driver", `uvm_component parent = null);
        super.new(name, parent);
    endfunction
    virtual task run_phase(uvm_phase phase);
        super.run_phase(phase);
        // reqs are sent to nodes A and B
        seq_item_port.get_next_item(req);
        seq_item_port.item_done();
    endtask
endclass
```

Building TUL and FUL drivers
Future Work

UVM-FM Structure Automation Flow

Testing UVM-FM Approach on More Complex Stacked Protocols

UVM-FM provides theoretical capabilities to test complex layered architectures. These capabilities are tested on an industry simple-layered architecture project. More testing is needed on complex-layered architectures.
Summary

- This paper describes the UVM-FM approach and provides specifications for the new package.
- UVM-FM provides UVM backward-compatible layer to simplify modeling the communication between the different layers in stacked protocols.
- UVM-FM is a step forward to standardize modeling the communication between the different layers in stacked protocols.
- UVM-FM has capabilities to extend and support modeling layered stacks with a tree structure.
- UVM-FM changes the UVM agent architecture to a network-like topology.
- Since the system communication is now centralized in the communication nodes, UVM-FM provides capabilities to debug testbenches by logging transferred messages.
- UVM-FM is a step toward an easy testbench automation flow.