Specman e wrapper for UVM Ethernet VIP

Test and Verification Solutions
The Verification Experts

Presented by:
Basavaraj Naik
CONTENTS

• Introduction
• UVC Configuration
• Data Item Generation
• Data Item Processing Flow
• E Proxy sequencer
• “mltypemap” utility
• References
INTRODUCTION

• The E-wrapper is adapted in order to reuse the existing UVC in specman-e verification environment.
• Eliminates the effort of recreating the same environment in specman-e.
• Mixed language utility helps in the configuration, stimulus generation, data item exchange across the language.
• Checking and coverage tasks can also be reused.
Existing Ethernet UVC
The E wrapper Environment

- UVC's sequences mapped to e
- Mapped sv seq
- Proxy sequencer executes mapped and new (pure e) sequences
- New e seq
- Virtual Sequencer
- UVM Tvs Ethernet ENV
  - ML sequencer stub
  - UVM Ethernet Agent
    - Agent sequencer
    - Agent Monitor
    - Agent Driver
  - UVM WRAPPER

T&VS Confidential
Now the fields of the config object and transaction item should be converted to specman e fields. This is done using “mltypemap” utility.

Configuration is propagated from the upper layer TB(e) and the tests can control the configuration.

The configuration object is passed to SV-UVC from top eTB.

The following is the configuration struct and has three fields as shown:

```plaintext
struct tvs_ethernet_config like any_struct {
!%enable : bit;        // to enable monitor checking signal in monitor
%is_active : ethernet_active_passive_t; // setting the agent is active or passive
!%has_scoreboard : bit; // to enable or disable the SV scoreboard
}; // end struct tvs_ethernet_config
```

```plaintext
class tvs_ethernet_config extends uvm_object;
    bit enable;        // to enable monitor checking signal in monitor
    uvm_active_passive_enum is_active=UVM_ACTIVE; // setting the agent is active or passive
    bit has_scoreboard=1;
endclass
```
To pass the configuration object from e to SV

- An “in” UVM method port is defined in the e layer.
- The configuration object is pulled from the SV TB.

```c
-- instantiate the SVE in sys
extend sys {
    eosv_tvs_tb : eosv_tvs_tb_u is instance;
    keep eosv_tvs_tb.agent() == "NCSV";

    // setting hdl_path for any signal access
    keep eosv_tvs_tb.hdl_path() == "-tb_top";

    // setting UVM path to corresponding SystemVerilog component instance
    keep eosv_tvs_tb.external_uvm_path() == "eosv_tvs_ethernet_tb";
}; // extend sys

-- E-over-CB SVE unit
unit eosv_tvs_tb_u {
    // ETHERNET configuration (for the DUT)
    tvs_ethernet_config : tvs_ethernet_config;
    // ETHERNET UVC configuration
    keep soft tvs_ethernet_config.is_active == UVM_ACTIVE;

    eosv_tvs_env : ETHERNET0 eosv_tvs_ethernet_env_u is instance;
    keep eosv_tvs_env.config == tvs_ethernet_config;
    // setting UVM path to point to corresponding SystemVerilog component instance
    keep eosv_tvs_env.external_uvm_path() == "ethernet0";
    // virtual sequencer
    virt_seqr : eosv_vir_seqr_u is instance;
    // Multi-language sequences adapter config (needs to be instantiated only once)
    ethernet_adapter_config is instance;

    connect_pointers() is also {
        virt_seqr.ethernet_proxy_seqr = eosv_tvs_env.ethernet_seqr;
    }; // connect_pointer...
};
```
The SV sequence item class is mapped to an e sequence item.
A SV sequencer stub component is created from the provided library ml_uvm_seq::ml_uvm_sequencer_stub.
Declare a sequencer stub class using ml_uvm_seq::ml_uvm_sequencer_stub_base as base class.
Instantiate the sequencer stub class in SV ENV (tvs_ethernet_env) and connect to SV sequencer using assign_sequencer() function.
An e proxy sequencer is created using the provided library base type ml_proxy_seq_driver.
The e proxy sequencer and SV sequencer stub are instantiated and connected in the ENV.
Data item processing flow

1. An e do action is allowed to proceed and generates an item.

2. The item is passed to SV, and the sequencer stub’s prepare_item_to_send() function is invoked, to process the item including any additional randomization.

3. The item is passed on to the actual UVC sequencer.
struct tvs_ethernet_item like any_sequence_item {
    // RGB_T pstruct;
    %destination_addr : uint(bits:48);  // The destination address field
    %source_addr : uint(bits:48);       // The source address field
    %len_or_type : uint(bits:16);       // The length or type field
    %packet : int;                     // Packet number
    %type_of_packet : uint(bits:4);
    // The type of packet (Basic type implementation)
    %type_of_payload_data : uint(bits:2); // The type of payload data (sequential or random)

    // monitor fields
    !%preamble : uint(bits:56);               // Preamble field
    !%sfd : byte;                            // start frame delimiter field
    !%payload_data : list of bit;           // payload data
    !%pad_data : list of bit;               // pad data
    !%extension_data : list of bit;         // extension data (if frame length<512 bytes)
    !%fcs : uint;                           // frame checksum
    !%payload_size : int;                   // payload size
    !%pad_size : int;                       // pad size
    !%extension_size : int;                 // extension size
}; // end struct tvs_ethernet_item
e sequencer proxy

- Derived from the provided base type. Does not maintain its own do queue.
- E proxy sequencer propagates all calls to SV stub
- Extends from a special library component.
- Single do queue is managed by the original UVC sequencer.
• Define interface ports in e testbench.
• Add logic SV testbench to register the SV_TLM interfaces with the ml_uvm library using calls to ml_uvm::external_if().
• Connect the TLM sv ports to e interface ports.
The `mltypemap` utility

- Automatically maps data item types across languages
  - Takes a struct/class declaration in a source language (SV)
  - Generates code in the target language (e):
    # Matching struct/class declaration
    # Serialization/de-serialization code (for TLM ports)

- To map the UVC objects
  - Compile and elaborate just the SV-UVC
  - Invoke `mltypemap` to map the required classes
THANK YOU