A scalable VIP component to increase robustness of co-verification within an ASIC

Mario de Matteis – Matteo Barbati
Agenda

• Co-verification adoption
• State-of-the-art Vs novel approach
• Description on the novel approach
• Overview on FW_VIP solution
• Case study analysis
• Conclusion and next step
Co-verification adoption to meet complexity

• Increasing complexity of the ASIC product (e.g. power regulator) is pushing platform to include SOC-like architectures.

• Standalone verification of the firmware with FPGA emulators doesn’t meet the signoff requirements of the complete DUT application:
  • A full-chip co-verification approach is required to qualify the device.
  • A testbench where the entire DUT (digital, analog logic plus the firmware) is instantiated should be adopted.
  • A fully featured UVM environment should be adopted.
State-of-the-art Vs novel approach

• **FPGA Centric approach**: firmware engineers verify their own code. A.k.a. “Traditional Approach”.

• **UVM-centric (VAL approach)**: “fake” register map to map verification component functionalities into firmware world and to develop test in firmware with randomization and UVM checkers in place.

• **Novel approach**: using FW_VIP component which reduces the impact of co-verification to the verification and firmware workflow.
Workflow changes

- Inserting labels within firmware to identify significant functions linked to product capabilities.
- Identifying significant variables that are linked to product capability.
- Automation of generation inputs for FW_VIP to manage changes of the firmware code.
Once the flow is in place the FW_VIP automatically translates the firmware events into UVM transactions which will be passed to the scoreboards and checkers.

The pictures is showing the some types of events the FW_VIP is able to monitor.
The verification IP topology is composed by:

- **MEMBUS VIP**: to monitor memory transactions (e.g. ahb).
- **PC IF**: to monitor addressing on Program counter.
- **MONITOR**: to translate events into transaction
- **CONFIG**: for automatic configuration to adapt to the firmware releases.
The **fw_packet** of the FW_VIP is suited to communicate to other components:

- Hardware and firmware synchronization events (e.g. interrupt calls) for which “**pc_event**” variable carries on the information.

- Variable updates within the firmware for which “**variableName**” and “**variableValue**” carries on the information.

```verilog
type def enum {pc, dut_state, variable} event_type;

class fw_packet extends uvm_sequence_item;
    event_type currentEvent;
    string pc_event;
    string variableName;
    bit [31:0] variableValue;

    `uvm_object_utils_begin(fw_packet)
        `uvm_field_enum(event_type, currentEvent, UVM_ALL_ON)
        `uvm_field_string(pc_event, UVM_ALL_ON)
        `uvm_field_string(variableName, UVM_ALL_ON)
        `uvm_field_int(variableValue, UVM_ALL_ON)
    `uvm_object_utils_end

    function new(string name = "fw_packet");
        super.new(name);
    endfunction

endclass : fw_packet
```
Configuration from case study

At the beginning of the simulation the components which are listening transactions and events related to the firmware load the values from two files.

The contents of these files are “solution dependent” and has to be defined by verification and firmware engineers together.

```
vout_max_ra 20000054E
vout_min_ra 200000550
vout_transition_rate_ra 200000556
vout_max_rb 2000005A2
vout_min_rb 2000005A4
vout_transition_rate_rb 2000005AA
```

Figure 6 – ramAddressList configuration file

```
load_configuration_end DEFAULT 00003E78
load_configuration HIDDEN 00003F09
load_user_configuration_from_OTP_end DEFAULT 00002Ef6
load_user_configuration_from_OTP HIDDEN 00002Be5
```

Figure 7 – pcValueList configuration file
Monitoring variables

The monitor of the FW_VIP operates as bridge for memory transactions between MEMBUS VIP and other parts of the verification environment.

The `ramAddressList` is the array filled up during configuration phase and contains the trigger points used to monitor the firmware variables. After the analysis of the MEMBUS VIP transaction the `fw_packet` is sent.
The monitor of the FW_VIP operates as bridge for memory transactions between program counter interface and other parts of the verification environment. The pcValueList is the array filled up from configuration phase and contains the trigger points used to monitor events related to program counter.

After the analysis of the program counter event the fw_packet is sent.
Automation process

Automation to deal with different firmware releases is achieved through configuration files which are generated by scripts.

The firmware and verification engineers work together to agree on the content of this files. Eventually label has to be inserted within the firmware.
Case Study 1: intercept pc to trigger self-check

The **fw_packet** is received through a port and it generates a trigger for the checker.
Case Study 1: copy check

The checker is started to check the correct execution of an operation.
Case Study2: variable checkers

The `fw_packet` is received through a port. The expected hardware properties (e.g. a register content) is checked according the value of the variable within the packet.
We successfully implement the novel methodology and reached all the predefined target.

## Conclusion

<table>
<thead>
<tr>
<th></th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
</table>
| FW-Centric (FPGA) approach | Can be used for Performance Analysis/Stress testing on FPGA/HW-emulator | Limited debug capabilities  
                         |                                                                     | Limited code coverage  
                         |                                                                     | No self-checking capabilities from UVM world  
                         |                                                                     | Few capabilities of analog emulation |
| UVM-Centric (VAL) approach | Full self-checking capabilities from UVM world  
                         | Easy to debug  
                         | High Code Coverage | Scenarios written in FW language  
                         |                                                                     | Verification eng. must manage aspects related to FW development (scatter file, compiler option etc.)  
                         |                                                                     | Can’t be used for Performance Analysis/Stress test on FPGA/HW-emulator |
| Novel Approach        | Full self-checking capabilities from UVM world  
                         | Easy to debug  
                         | High Coverage | Can’t be used for Performance Analysis/Stress test on FPGA/HW-emulator  
                         | Limited changes to usual FW and Verification workflows |
Next step

1. To extend the approach to a multi-core IC.

2. To implement coverage analysis of the firmware with specific covergroup and/or line code coverage
Questions

Any question is well accepted... 😊