2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

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

Mario de Matteis – Matteo Barbati

onsemi™

accellera SYSTEMS INITIATIVE

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

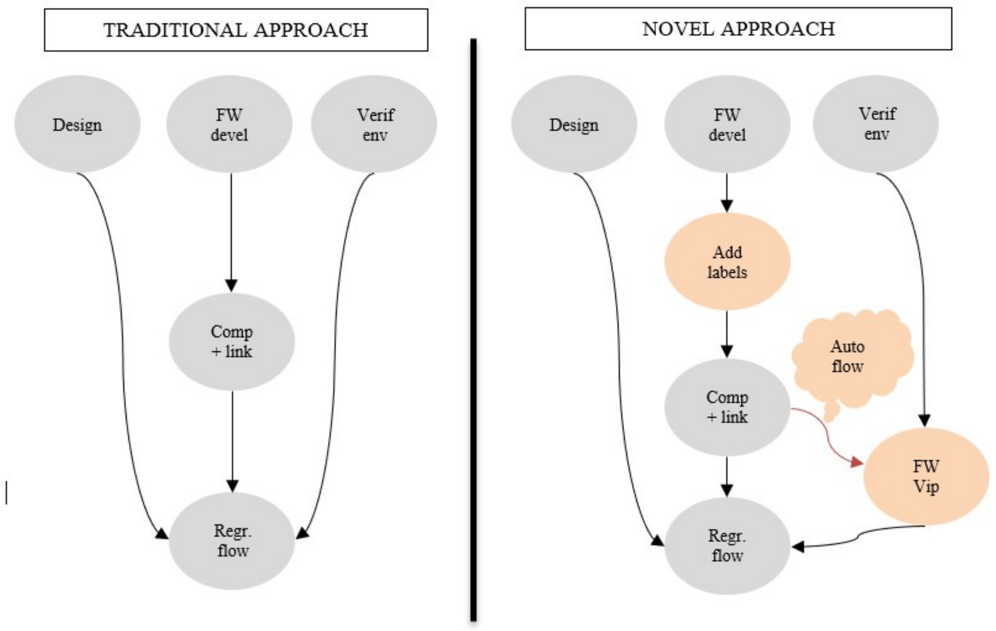
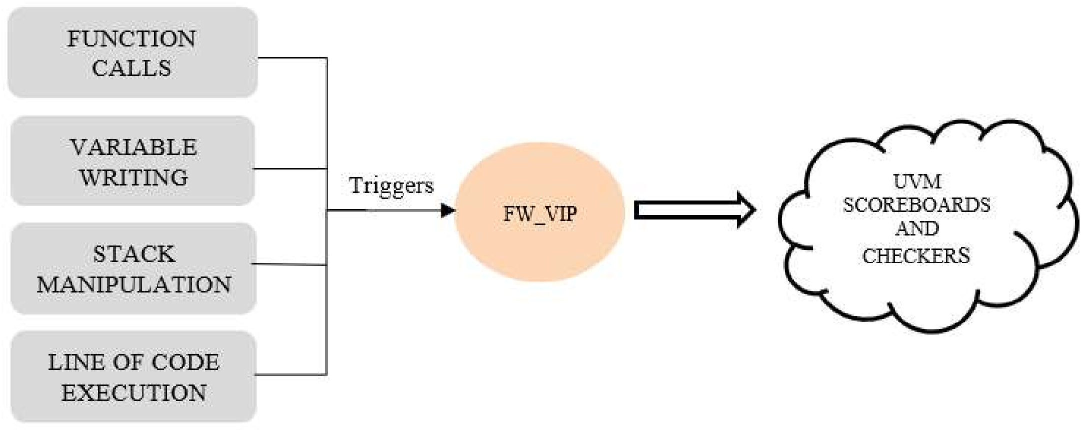


Figure 1 - Workflow changes

- 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.

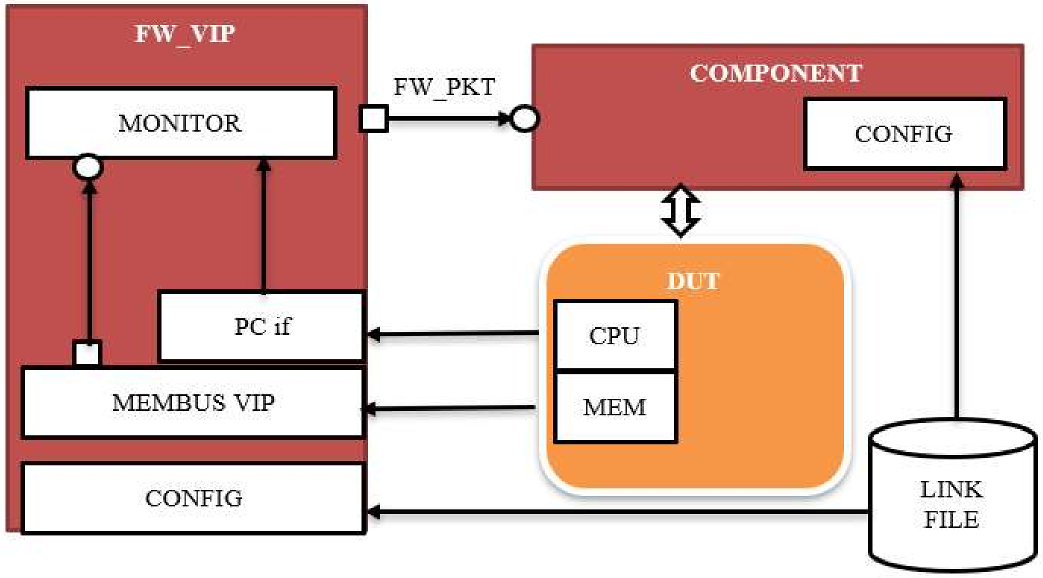
FW_VIP: monitored events



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.

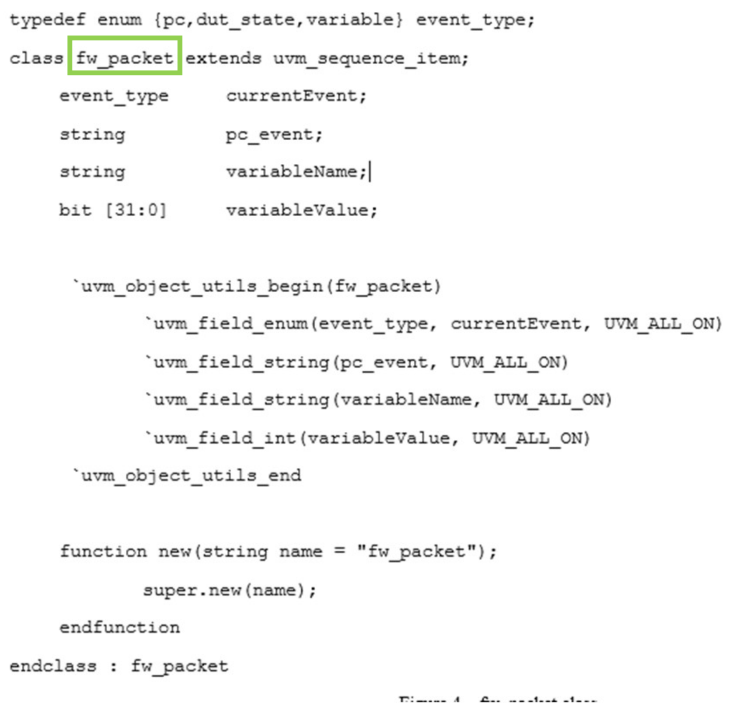
FW_VIP: block description



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.

FW_VIP: the packet

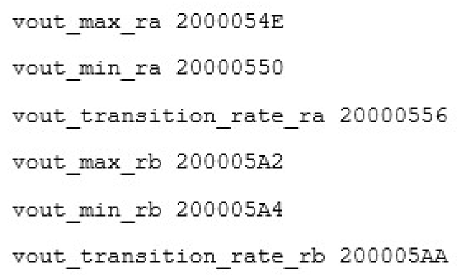


```
typedef enum {pc,dut_state,variable} event_type;
class fw_packet extends uvm_sequence_item;
    event_type      currentEvent;
    string          pc_event;
    string          variableName;
    bit [31:0]      variableValue;

     `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
```

The **fw_packet** of the FW_VIP is suited to communicate to other components:

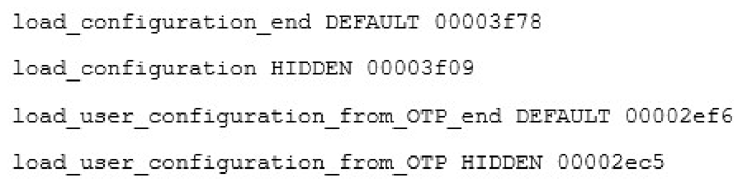
- Hardware and firmware synchronization events (e.g. interrupt calls) for which "**pc_event**" variable carries on the information.
- Variable updates within the firmware for which "**variableName**" and "**variableValue**" carries on the information.

Configuration from case study



```
vout_max_ra 2000054E

vout_min_ra 20000550

vout_transition_rate_ra 20000556

vout_max_rb 200005A2

vout_min_rb 200005A4

vout_transition_rate_rb 200005AA
```

Figure 6 – ramAddressList configuration file



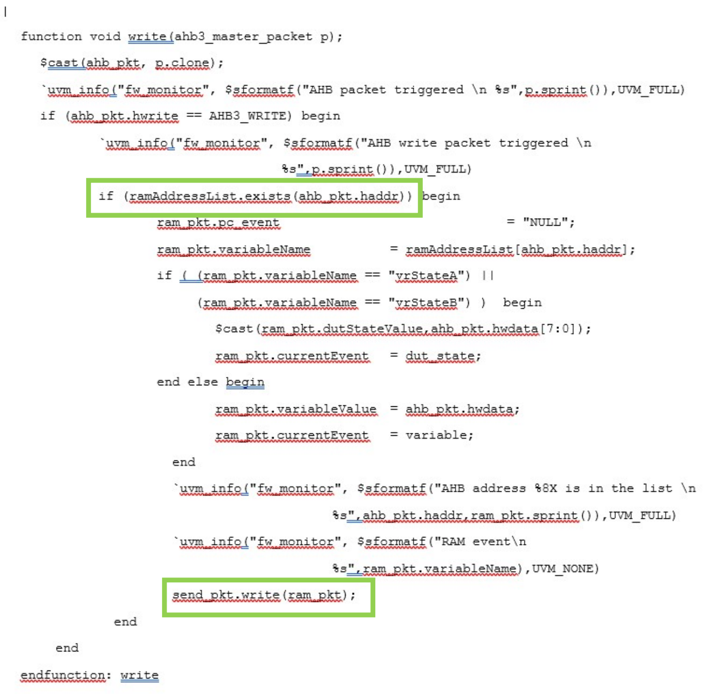
```
load_configuration_end DEFAULT 00003f78

load_configuration HIDDEN 00003f09

load_user_configuration_from_OTP_end DEFAULT 00002ef6

load_user_configuration_from_OTP HIDDEN 00002ec5
```

Figure 7 – pcValueList configuration file

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.

Monitoring variables

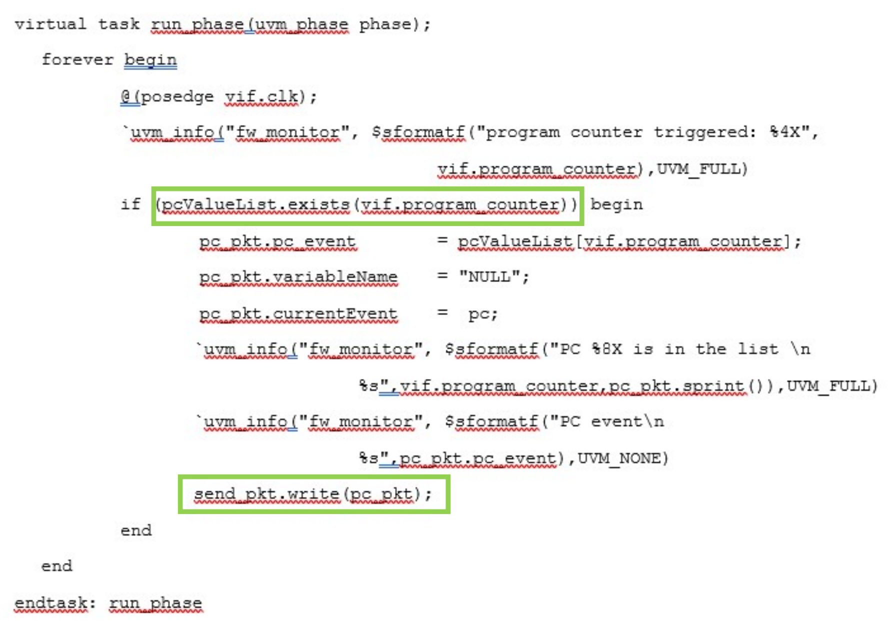


```
function void write(ahb3_master_packet p);
   $cast(ahb_pkt, p.clone);
   `uvm_info("fw_monitor", $sformatf("AHB packet triggered \n %s",p.sprint()),UVM_FULL)
   if (ahb_pkt.hwrite == AHB3_WRITE) begin
           `uvm_info("fw_monitor", $sformatf("AHB write packet triggered \n
                                %s",p.sprint()),UVM_FULL)
           if (ramAddressList.exists(ahb_pkt.haddr)) begin
                 ram_pkt.pc_event                          = "NULL";
                 ram_pkt.variableName          = ramAddressList[ahb_pkt.haddr];
                 if ( (ram_pkt.variableName == "vrStateA") ||
                      (ram_pkt.variableName == "vrStateB") )  begin
                       $cast(ram_pkt.dutStateValue,ahb_pkt.hwdata[7:0]);
                       ram_pkt.currentEvent   = dut_state;
                 end else begin
                       ram_pkt.variableValue  = ahb_pkt.hwdata;
                       ram_pkt.currentEvent   = variable;
                   end
                  `uvm_info("fw_monitor", $sformatf("AHB address %8X is in the list \n
                                    %s",ahb_pkt.haddr,ram_pkt.sprint()),UVM_FULL)
                  `uvm_info("fw_monitor", $sformatf("RAM event\n
                                    %s",ram_pkt.variableName),UVM_NONE)
                  send_pkt.write(ram_pkt);
           end
      end
endfunction: write
```

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.

Monitoring Program Counter



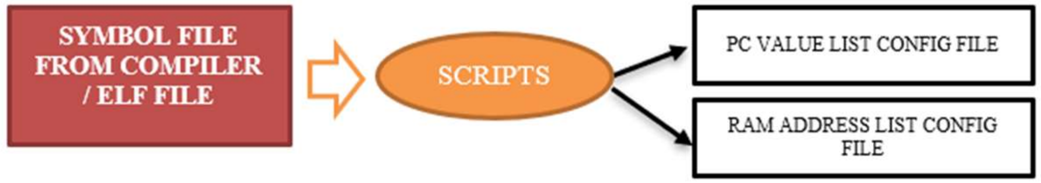
```
virtual task run_phase(uvm_phase phase);
  forever begin
        @(posedge vif.clk);
        `uvm_info("fw_monitor", $sformatf("program counter triggered: %4X",
                                          vif.program_counter),UVM_FULL)
        if (pcValueList.exists(vif.program_counter)) begin
              pc_pkt.pc_event        = pcValueList[vif.program_counter];
              pc_pkt.variableName    = "NULL";
              pc_pkt.currentEvent    =  pc;
              `uvm_info("fw_monitor", $sformatf("PC %8X is in the list \n
                          %s",vif.program_counter,pc_pkt.sprint()),UVM_FULL)
              `uvm_info("fw_monitor", $sformatf("PC event\n
                          %s",pc_pkt.pc_event),UVM_NONE)
             send_pkt.write(pc_pkt);
        end
  end
endtask: run_phase
```

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.

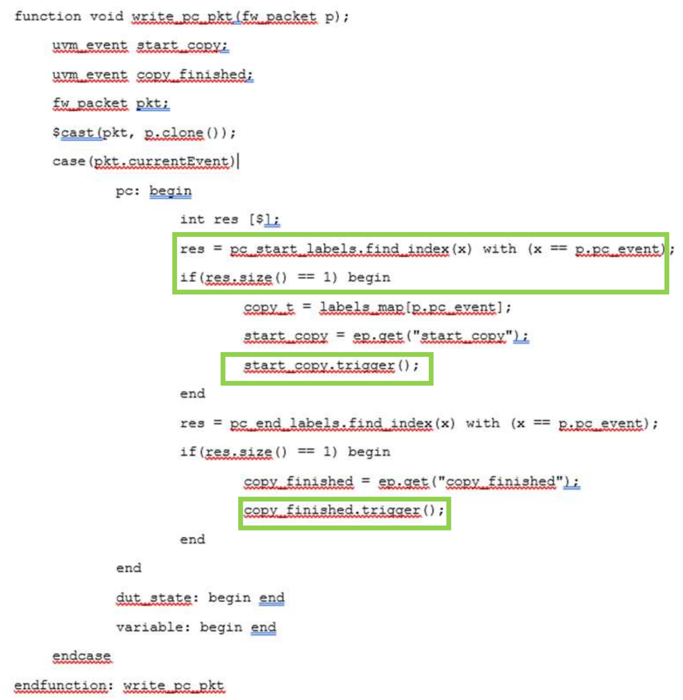
Automation process


SYMBOL FILE FROM COMPILER / ELF FILE
SCRIPTS
PC VALUE LIST CONFIG FILE
RAM ADDRESS LIST CONFIG FILE

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

The firmware and verification engineers works together to agree on the content of this files. Eventually label has to be inserted within the firmware.

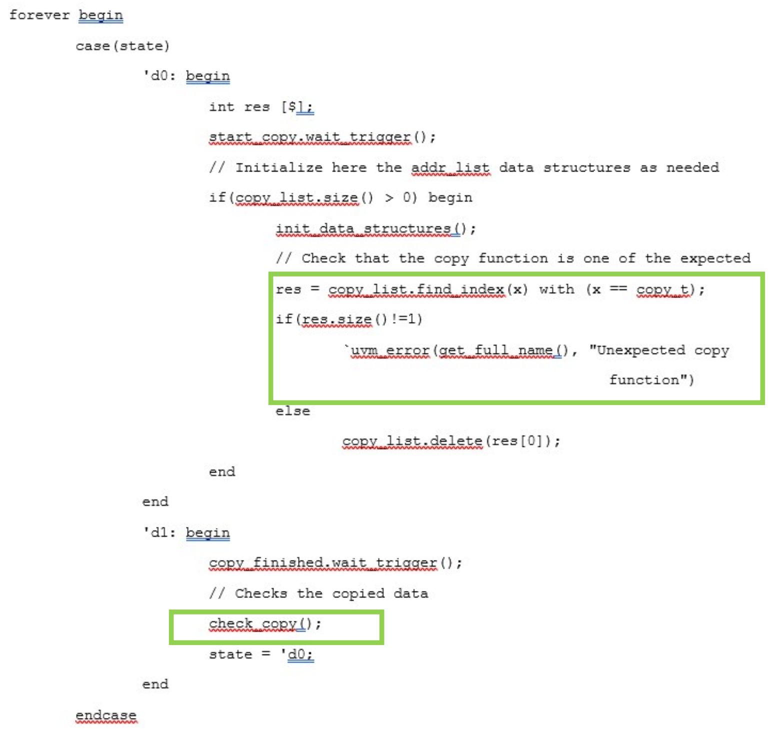
Case Study1: intercept pc to trigger self-check



```
function void write_pc_pkt(fw_packet p);
    uvm_event start_copy;
    uvm_event copy_finished;
    fw_packet pkt;
    $cast(pkt, p.clone());
    case(pkt.currentEvent)
            pc: begin
                    int res [$];
                    res = pc_start_labels.find_index(x) with (x == p.pc_event);
                    if(res.size() == 1) begin
                            copy_t = labels_map[p.pc_event];
                            start_copy = ep.get("start_copy");
                            start_copy.trigger();
                    end
                    res = pc_end_labels.find_index(x) with (x == p.pc_event);
                    if(res.size() == 1) begin
                            copy_finished = ep.get("copy_finished");
                            copy_finished.trigger();
                    end
            end
            dut_state: begin end
            variable: begin end
    endcase
endfunction: write_pc_pkt
```

The **fw_packet** is received through a port and it generates a trigger for the checker.

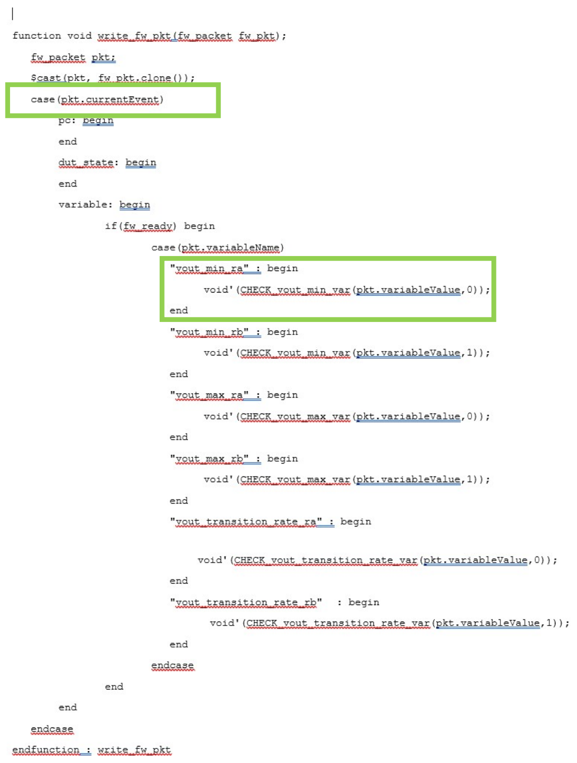
Case Study1: copy check



```
forever begin
    case(state)
        'd0: begin
            int res [$];
            start_copy.wait_trigger();
            // Initialize here the addr_list data structures as needed
            if(copy_list.size() > 0) begin
                init_data_structures();
                // Check that the copy function is one of the expected
                res = copy_list.find_index(x) with (x == copy_t);
                if(res.size()!=1)
                    `uvm_error(get_full_name(), "Unexpected copy
                                                function")
                else
                    copy_list.delete(res[0]);
            end
        end
        'd1: begin
            copy_finished.wait_trigger();
            // Checks the copied data
            check_copy();
            state = 'd0;
        end
    endcase
```

The checker is started to check the correct execution of an operation.

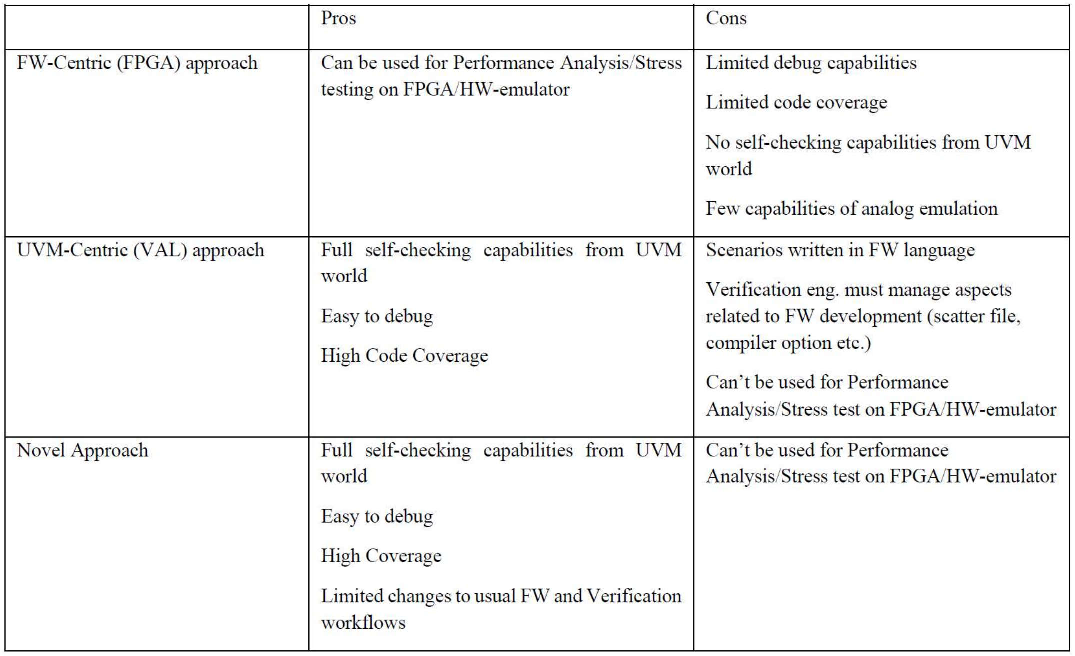
Case Study2: variable checkers



```
function void write_fw_pkt(fw_packet fw_pkt);
   fw_packet pkt;
   $cast(pkt, fw_pkt.clone());
   case(pkt.currentEvent)
       pc: begin
       end
       dut_state: begin
       end
       variable: begin
              if(fw_ready) begin
                     case(pkt.variableName)
                      "vout_min_ra" : begin
                           void'(CHECK_vout_min_var(pkt.variableValue,0));
                      end
                      "vout_min_rb" : begin
                           void'(CHECK_vout_min_var(pkt.variableValue,1));
                      end
                      "vout_max_ra" : begin
                           void'(CHECK_vout_max_var(pkt.variableValue,0));
                      end
                      "vout_max_rb" : begin
                           void'(CHECK_vout_max_var(pkt.variableValue,1));
                      end
                      "vout_transition_rate_ra" : begin

                          void'(CHECK_vout_transition_rate_var(pkt.variableValue,0));
                      end
                      "vout_transition_rate_rb"   : begin
                            void'(CHECK_vout_transition_rate_var(pkt.variableValue,1));
                      end
                     endcase
              end
       end
   endcase
endfunction : write_fw_pkt
```

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 .

Conclusion



	Pros	Cons
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 Limited changes to usual FW and Verification workflows	Can't be used for Performance Analysis/Stress test on FPGA/HW-emulator

We successfully implement the novel methodology and reached all the predefined target.

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… ☺