Modem Design: UVM-to-UVMA Migration: Update #3

Contents

1 Overview

1.1 Target files

The following files are chosen by Yungi Um for migration. (files in RED are available in VCAD chamber)

- **Common interface**:
  - **sec_modem/lib/common/common_intf.sv**
- **Interfaces**:
  - **sec_modem/lib/rxf/rxf_intf.sv**
  - **sec_modem/lib/symbproc4g/symbproc4g_intf.sv**
  - **sec_modem/lib/symbproc4gc/symbproc4gc_intf.sv**
  - **sec_modem/lib/demod_4g/demod_4g_intf.sv**
  - **sec_modem/lib/ifrgen/ifrgen_intf.sv**
  - **sec_modem/lib/dm/dm_intf.sv**
  - **uvc/sec_dm/v201408/sv/sec_dm_intf.sv**
  - **uvc/sec_demod/v201408/sv/sec_demod_intf.sv**
  - **uvc/sec_rxf/v201408/sv/sec_rxf_intf.sv**
- **Interface instantiations**:
  - **sec_modem/tb/top/intf_inst.sv**
- **Drivers**:
  - **uvc/sec_rxf/v201408/sv/sec_rxf_driver.sv**
- **Monitors**:
  - **sec_modem/lib/common/base_lib_mon.sv**
  - **sec_modem/lib/symbproc4gc/symbproc4gc_mon.sv**
- **Sequences**:
  - **uvc/sec_modem/lib/rxf/rxf_{seq,vseq}_lib.sv**
  - **uvc/sec_modem/lib/symbproc4g/symbproc4g_{seq,vseq}_lib.sv**
  - **uvc/sec_modem/lib/symbproc4gc/symbproc4gc_{seq,vseq}_lib.sv**
  - **uvc/sec_modem/lib/demod_4g/demod_4g_{seq,vseq}_lib.sv** (only seq_lib given)

  - **uvc/sec_modem/lib/dm/dm_{seq,vseq}_lib.sv**
  - **uvc/sec_modem/lib/ifrgrn/ifrgen_{seq,vseq}_lib.sv**
- **Checkers**:
  - **checkers/demod_4g/NonCol_Compare.inc**
  - **checkers/demod_4g/compare_fdi.inc**
  - **checkers/demod_4g/Pbch_Compare.inc**
  - **checkers/demod_4g/compare_iw_rcal_done.inc**
  - **checkers/demod_4g/ce_pp_compare.inc**
  - **checkers/demod_4g/compare_mmsewg.inc**
  - **checkers/demod_4g/ce_y_compare.inc**
  - **checkers/demod_4g/compare_tdi.inc**
  - **checkers/demod_4g/compare_ToneDemapper.inc**
  - **checkers/demod_4g/drs_dmrs_ext.inc**
  - **checkers/demod_4g/compare_chbridge_mimoout.inc**
  - **checkers/demod_4g/fdagc_checker.inc**
  - **checkers/demod_4g/compare_chbridge_mimooutCCD.inc**
  - **checkers/demod_4g/fft_checker.inc**
  - **checkers/demod_4g/compare_chbridge_mimooutCCDHICH.inc**
  - **checkers/demod_4g/front_out_checker.inc**
  - **checkers/demod_4g/compare_chbridge_mimooutPBCH.inc**
  - **checkers/demod_4g/pdp_checker.inc**
  - **checkers/demod_4g/compare_chbridge_pdcch_llr.inc**
  - **checkers/demod_4g/sinr_checker.inc**
  - **checkers/demod_4g/compare_csi_postproc.inc**
  - **checkers/demod_4g/tc_checker.inc**
  - **checkers/demod_4g/compare_csi_preproc.inc**

1.2 Code organization

The following “initial” code organization is assumed.

1.2.1 TB

```
module incr_top;
  //‘include "env_intf_inst.sv"
  intf intf(.nReset(top.nReset), .SystemClock(top.SystemClock, .*);

  // interface instantiations
  // interface could be instantiated either in DUT or TB initially;
  // if instantiated inside DUT, should be set as dutexcl to map it to SW
  // initially; eventually, H-interface portion of rxf_intf will be mapped
  // to HW
  rxf_intf rxf_intf(.nReset(top.nReset),
                    .SystemClock(top.SystemClock),
                    .clk(top.clkrst_if.clk245),
                    .clkx2(top.clkrst_if.clk245),
                    .clkx4(top.clkrst_if.clk245),
                    .reset_n(top.clkrst_if.rsn));

  initial begin
    uvm_config_db#(virtual intf)::set(null, "uvm_test_top.*", "vintf",
                                      incr_top.intf);
  end
endmodule
```

1.2.2 DUT

```
module top;
  // clock/reset interface
  clkrst_intf clkrst_if(...);

  // DUT wrapper
  dut dut;
endmodule
module dut;
  Modem A_Modem;
endmodule
module A_Modem
endmodule
```

2 Migration Steps

2.1 STEP #1: From signal-based to transaction-based

2.1.1 Action items

- Move all tasks used inside DRIVER and MONITOR into interfaces. In particular,
  - `drive_transfer` task should be moved out of DRIVER class and put inside target SV interface
    - See SECTION 5:STEP#1, for details.
  - `collect_type0` and `collect_type1` task should be moved out of MONITOR class and put inside target SV interface
    - See SECTION 6:STEP#1, for details.

2.1.2 Notes

- At this point, no need to worry about synthesizability, since the SV interface will be still in SW-space. If the interface, say `sec_rxf_intf` is instantiated under DUT (or we can still leave them under INCR_TOP), then we should map it to SW by providing IXCOM option +`dutexcl`+`sec_rxf_intf`.

2.2 STEP #2: Clean-up common interface

2.2.1 Action items

- **COMMON_INTF**: Split `common_intf.sv` into two parts: `common_intf_S.sv` and `common_intf_H.sv`.

    - `common_intf_S.sv` will contain all objects which is non-synthesizable or is non-essential for DUT computation or checking
    - `common_intf_H.sv` will contain all objects which are actually driven DUT and will be used inside DUT (especially for checking, etc.)
    - all SV interfaces which include `common_intf.sv` must be changed so that they will include both `common_intf_H.sv` and `common_intf_S.sv`.
        * See SECTION 3, for details.
- **CHECKERS**: Move checkers into DUT

2.2.2 Notes

- Entire checker module should be included into DUT. Moving checkers into DUT is a good exercise for STEP #3 and #4. Also, would give good introduction how to make non-synthesizable module into PXP HW.
- After STEP #2, HW-run MUST pass.

2.3 STEP #3: Split S-interface and H-interface

2.3.1 Action items

- **STEP #3.1: Split interfaces**: Now, each actual interface should be split into S-interface and H-inteface. For example, `rxf_intf.sv` should be split into `rxf_intf_S.sv` and `rxf_intf_H.sv`.
    - First, for each sig1nal defined in SV interface, decide where to put it: HW (`rxf_intf_H`) or SW (`rxf_intf_S`)
    - Next, for each process (INITIAL or ALWAYS) and task/function, decide where to put it: HW (`rxf_intf_H`) or SW (`rxf_intf_S`)
        * Some process touches both signals in HW (`rxf_intf_H`) and signals in (`rxf_intf_S`).
        * Then, such process need to be rewritten so that it will be split into two parts: HW-part, and SW-part.
        * Also, some logic may need to be inserted to connect the HW-part of the process and SW-part of the process.
    - `rxf_intf_H.sv` will be compiled by IXCOM (in DUT compilation)
        * But still, we will map `rxf_intf_H` into SW since it will still contain nonsynthesizable constructs
        * `rxf_intf_H` interface should only include `common_intf_H.sv`
    - `rxf_intf_S.sv` will be compiled by IES (in TB compilation)
        * `rxf_intf_S` interface should only include `common_intf_S.sv`
- **STEP #3.2: Remove non-synthesizable constructs from H-interfaces**:
    - All uses of non-synthesizable objects in H-interface (e.g. queues, dynamic arrays, etc.) should be remodeled using synthesizable objects.
    - After this SUBSTEP, remove all `+dutexcl+<intf>` options. Make sure IXCOM compilation works and all H-interfaces are mapped to HW. HW-run or SW-run would not pass since we haven't connected S-interface and H-inteface yet.
- **STEP #3.3: Connect S-interface with H-interface using DPIMAP**:
    - After H-interfaces are mapped to HW, even SW-run would not pass. To make this pass, we need to bridge two using DPIMAP.
    - Also, any memory transfer should be modified to use `dpiMap::put/getBytes()` function calls.

2.3.2 Notes

- After STEP #3.1, HW-run MUST pass.
- After STEP #3.2, SW-run MUST pass.
- After STEP #3.3, HW-run MUST pass.

2.4 STEP #4: Performance Optimization

2.4.1 Action items

- Iteration: profiling and performance optimization.

3 Common Interface: common_intf.sv

Common interface objects defined in `common_intf.sv` can be classified into two: a) objects to put into HW and b) objects to put into SW.

For example, `check_data` which are bound to actual DUT signals should be put into HW for performance. If we put it in SW, every time the corresponding DUT signal changes, the value change should be propagated to `check_data` which is sitting in SW. However, some objects should be put into SW either because they cannot be synthesized in HW (e.g. strings) or they actually are not needed for DUT computation – `check_int_key_name` appears to fall into this category.

NOTE: From now on, when a code is displayed, the following color coding is used.

- **RED means that the code should be put into HW for performance**
- **BLUE means that the code should be put in SW, either because it's non-synthesizable or there's little benefit in putting it to HW.**
- **GREEN means that it's not clear where to put the code.**
- **BOLD comments were added by Cheoljoo** and comments in normal font is original comment in the code.

3.1 CODE

```
// (i) PULSE GEN
//   - Q: what is the purpose of this pulse signal? also, where are the loads of this signal?
//        why vclk?
bit vclk;
always #2 vclk = ~vclk;
`define INT_PULSE_GEN(NAME,SRC) \
  bit r``NAME; \
  always @ (posedge vclk) r``NAME <= SRC; \
  wire NAME``Pulse = SRC&~r``NAME;

// (ii) METAINFO of REFERNCE (GOLDEN) DATA: filled up from FILE at time 0 and compared against
//      DUT-generated data later
//   - it appears that folowing BLUE code are mostly for logging purpose and  put only into q
//     SW-side (e.g. S-interface) and access them only from SW-side
//   - there are MAX_CHECK_NUM check points, where each check point is specified by a single file
//   - MAX_CHECK_NUM is an interface parameter
string check_point_name[MAX_CHECK_NUM];

// the reference file name
string check_ref_file_name[MAX_CHECK_NUM];
```

```
// reference file type (0: 1 column, non zero: 2 columns)
// 1: (i,q) pair
// 2: (addr,data) pair
int check_ref_file_type[MAX_CHECK_NUM];

// reference file existence
int check_ref_file_exists[MAX_CHECK_NUM];

// reference data size (only for collect_type1)
int check_ref_data_max_num[MAX_CHECK_NUM];
initial foreach (check_ref_data_max_num[i]) check_ref_data_max_num[i] = -1;

// key information to extract reference data from file
string check_int_key_name[MAX_CHECK_NUM][$];
int check_int_key_value[MAX_CHECK_NUM][$];
string check_str_key_name[MAX_CHECK_NUM][$];
string check_str_key_value[MAX_CHECK_NUM][$];


// (iii) CHECK_CLOCK: used to clock checking processes, etc.
//   - driven from DUT (Q: can you confirm?)
//   - used in TB; uses of CHECK_CLOCK are:
//     . MONITOR code uses this clock to fetch one DUT output item, which is put into a queue
//       inside a transaction
//     . for updating CHECK_REF_DATA_IDX inside a process in interfaces, (CHECK_REF_DATA_IDX
//       value is eventually used inside MONITOR code, to number DUT output item
// clock used for capturing the sample
logic check_clk[MAX_CHECK_NUM];

// Q: when CHECK_CLK_SKIP_NUM is NOT 0, where do we get the non-0 value?
//   - is it a constant or value is read from a file?
// number of clocks we should skip after capturing the sample (usually 0)
int check_clk_skip_num[MAX_CHECK_NUM];

// (iv) CHECK_START: indicates that now DUT emits outputs that need to be checked
//   - driven from DUT
//   - on CHECK_START, TB code will populate parameters of the given check point.
//     . after parameters are set, CHECK_PARAM_SET_END will be set
//   - MONITOR code (collect_type) waits for CHECK_START and CHECK_PARAM_SET_END, and then
//     start to fetch DUT output item one by one into transaction
// signal indicating the start of the related logic (once or multiple times throughout simulation)
// if we use multiply-generated start signal, should use "collect_type0" function at monitor.
// else if we use one-time-generated start singal, should use "collect_type1" function at monitor.
logic check_start[MAX_CHECK_NUM];

// (v) CHECK_DATA_EN: level signal which is asserted while DUT generates output items
//   - processes clocked on CHECK_CLK are guarded by this
// in-time data enable signal used for capturing the sample
logic check_data_en[MAX_CHECK_NUM];
```

```
// (vi) CHECK_DATA: actual DUT output items
//   - driven through XMR
// DUT signal to be captured
logic [127:0] check_data[MAX_CHECK_NUM];
logic [127:0] check_data_i[MAX_CHECK_NUM];
logic [127:0] check_data_q[MAX_CHECK_NUM];

// (vii) CHECK_MASK
//  - mostly constant value (driven through cont asgn
//  - only read by TB (in collect_type in MONITOR)
//  - Q: can you confirm that it's only used in collect_type task?
// this is used for masking the invalid bit within 128-bit.
logic [127:0] check_mask[MAX_CHECK_NUM];

// (viii) CHECK_REF_DATA_IDX: during checking, multiple output items to be checked are
// generated by DUT, this index is used to number them
//   - typically initialized to 0 on CHECK_START
//   - incremented by 1 on CHECK_CLOCK edge iff CHECK_DATA_EN
//   - only used in collect_type task in MONITOR code
//   - still, better to update this value in HW since the update logic is clocked;
//     if we update this value in SW, we need to stop on every CHECK_CLK posedge
// symbol index within key group in testvector synchronizing with the currently being
// captured DUT data
int check_ref_data_idx[MAX_CHECK_NUM];

// (ix) CHECK_DONE: opposite of CHECK_START
//   - used to indicate the end of one CHECK BURST
//   - see collect_type0 code in MONITOR how it's used
//   - for performance, we better put this to HW; for details, see section on MONITOR
// if some burst processing has its start-done pair, we should describe its done signal.
// this is only valid when multiple start signals exist ("collect_type0")
logic check_done[MAX_CHECK_NUM];

// if some burst processing generates multiple done signals at a single start signal,
// we should define the number of done signal.
// this is only valid when multiple start signals exist  ("collect_type0")
int check_done_num[MAX_CHECK_NUM];

// (x) CHECK_PARAM_SET_END
//   - on CHECK_START, some SW-code for inserting ‘‘markers’’, which marks the beginning of new
//     check data is inserted
//   - CHECK_PARAM_SET_END is triggered to tell such insertion is finished
//   - after CHECK_PARAM_SET_END (and CHECK_START) actually, collecting begins
// we should trigger this event, when we finish setting all parameters of checking point.
// this starts monitor to capture the samples.
event check_param_set_end[MAX_CHECK_NUM];

// (xi) CHECKER_ON: looks like another guard which enables/disables CHECKING
bit checker_on;
// this indicates whether this block is enabled.
// we trigger on at body task of monitor.
// (xi) TEST_ON
//   - Q: what is this for?
//   - Q: why is the value set after #5000?
//   - if we need to put this to HW, may be need to remodel as in:
//     S_intf: initial #5000ns H_intf.set_test_on();
//     H_intf: function void set_test_on; test_on = checker_on; endfuncion
bit test_on;
// this indicates whether reset is released.
initial #5000ns test_on = checker_on;
```

3.2 Details

- **STEP #1**: Split `common_intf.sv` into two: one for SW-side, the other for HW-side
  1. Create two files: `common_intf_S.sv` and `common_intf_H.sv`.

     ```
     // common_intf_S.sv
     ```
  2. Change the original `common_intf.sv` file.

     ```
     ‘include "common_intf_S.sv"
     ‘include "common_intf_H.sv"
     ```
- **STEP #2**: No further steps needed for this file. Later, when we split an actual interface into S-interface and H-interface, we can have S-interface include `common_intf_S.sv` and have H-interface include `common_intf_H.sv`.

4 INTERFACES

Interfaces contain objects and processes. For acceleration, we need to split the set of objects and processes into two: one for HW and the other for SW. For this, remodeling may be needed. Also, code which bridges between HW and SW objects/processes may need to be added.

4.1 demod_4g_intf.sv

- This interface is mostly empty.

4.2 symbproc4gc_intf.sv

- **SETUP**:
  - There are 49 check items: `Check [0]` through `Check [48]`.
  - For each check item there are code which performs following (consider `Check [0]` for example):

    ```
    // binds real ‘‘DUT signals’’ to ‘‘generic’’ interface signals; should be put into HW
       - Q: Can you confirm sp4gc_Clk245, sp4gc_BchStrt are DUT signals? How are they driven?
            In a continuous assignment such as "assign sp4gc_Clk245 = dut.x.clk"?
    assign  check_clk[0]            = sp4gc_Clk245;
    initial check_ref_file_name[0]  = $sformatf("lspcch_dscr_a0301_in.txt");
    assign  check_start[0]          = sp4gc_BchStrt;
    assign  check_clk_skip_num[0]   = 0;
    assign  check_data_en[0]        = sp4gc_BchDataEn;
    assign  check_mask[0]           = 6’h3f;
    assign  check_done_num[0]       = 1;
    ```

```
assign  check_done[0]           = sp4gc_WagWrDone;
assign  check_data[0]           = sp4gc_BchData;

// most of the following code should be executed in SW space; either
//   i) can be transformed into
//        always @(posedge check_start[0] iff checker_on)
//          do_check_param_set(HenbTtiOn, ...);
//      where do_check_param_set is defined in SW (say, common_intf_sw.sv)
// or
//   ii) just put the entire process into S-interface and set check_start, checker_on,
//       sp4gc_* signals as export_read. If value changes are frequent on these signals,
//       i) would be more efficient.
always@(posedge check_start[0] iff checker_on)
begin
  if(sp4gc_PbchDecOn&&sp4gc_start_cnt==0)
    check_ref_data_idx[0] = 0;
  check_int_key_name[0].delete();
  check_int_key_value[0].delete();
  check_str_key_name[0].delete();
  check_str_key_value[0].delete();

  if (sp4gc_EicicOn) begin
    check_point_name[0]     = $sformatf("lspcch_a0301_in_bch (for Regen)");
  end else if (HenbTtiOn) begin
    check_point_name[0]     = $sformatf("lspcch_a0301_in_bch (for Henb)");
  end else begin
    check_point_name[0]     = $sformatf("lspcch_a0301_in_bch");
  end

  if (sp4gc_EicicOn) begin
  end else begin
    if (HenbTtiOn) begin
      check_int_key_name[0].push_back("asfr");
      check_int_key_value[0].push_back(10);
    end else begin
      check_int_key_name[0].push_back("fr");
      check_int_key_value[0].push_back(fr_idx_pbch);
    end
  end

  check_int_key_name[0].push_back("cc");
  check_int_key_value[0].push_back(0);

  if(sp4gc_PbchDecOn) begin
    check_str_key_name[0].push_back("chan");
    check_str_key_value[0].push_back("PBCH");
    if (sp4gc_oPdcchSibUnknown) begin
      check_int_key_name[0].push_back("decGrp");
      check_int_key_value[0].push_back(sp4gc_oPdcchSibUnknownIdx-1);
    end else begin
      check_int_key_name[0].push_back("decGrp");
      check_int_key_value[0].push_back(sp4gc_pbch_decgrp);
    end
  end
    ->check_param_set_end[0];
end

// should be put into HW
always@(posedge sp4gc_Clk245 iff checker_on)
  if(sp4gc_CchDecDone)
    sp4gc_start_cnt <= 0;
  else if(check_done[0])
    sp4gc_start_cnt <= sp4gc_start_cnt+1;

always@(posedge sp4gc_Clk245 iff checker_on)
  if(check_data_en[0])
    check_ref_data_idx[0] <= check_ref_data_idx[0]+1;
```

- **STEP #1**: Create a task to be used between S-interface and H-interface.
  - For each Check[*i*], change the BLUE code as follows.

```
// add appropriate types for task parameters
//   - will be put into S-interface, in a later STEP
function void do_check_param_set(input sp4gc_EicicOn, HenbTtiOn, sp4gc_PbchDecOn,
                                 sp4gc_oPdcchSibUnknown);
  check_int_key_name[0].delete();
  check_int_key_value[0].delete();
  check_str_key_name[0].delete();
  check_str_key_value[0].delete();
  if (sp4gc_EicicOn) begin
    check_point_name[0]     = $sformatf("lspcch_a0301_in_bch (for Regen)");
  end else if (HenbTtiOn) begin
    check_point_name[0]     = $sformatf("lspcch_a0301_in_bch (for Henb)");
  end else begin
    check_point_name[0]     = $sformatf("lspcch_a0301_in_bch");
  end

  if (sp4gc_EicicOn) begin
  end else begin
    if (HenbTtiOn) begin
      check_int_key_name[0].push_back("asfr");
      check_int_key_value[0].push_back(10);
    end else begin
      check_int_key_name[0].push_back("fr");
      check_int_key_value[0].push_back(fr_idx_pbch);
    end
  end

  check_int_key_name[0].push_back("cc");
  check_int_key_value[0].push_back(0);

  if(sp4gc_PbchDecOn) begin
```

```
        check_str_key_name[0].push_back("chan");
        check_str_key_value[0].push_back("PBCH");
        if (sp4gc_oPdcchSibUnknown) begin
          check_int_key_name[0].push_back("decGrp");
          check_int_key_value[0].push_back(sp4gc_oPdcchSibUnknownIdx-1);
        end else begin
          check_int_key_name[0].push_back("decGrp");
          check_int_key_value[0].push_back(sp4gc_pbch_decgrp);
        end
      end
        ->check_param_set_end[0];
    endfunction

    // Always process will be put into HW
    //   - will be put into H-interface, later
    always@(posedge check_start[0]) begin
      if (checker_on) begin
        if(sp4gc_PbchDecOn&&sp4gc_start_cnt==0)
          check_ref_data_idx[0] = 0;
        do_check_param_set(sp4gc_EicicOn, HenbTtiOn, sp4gc_PbchDecOn, sp4gc_oPdcchSibUnknown);
      end
    end
```

- Validate the code change.

- **STEP #2**: Partition the code into H-interface code and S-inteface code.
  - To be continued.

4.3 rxf_intf.sv

5 DRIVERS

Most driver UVCs use some template which eventually calls `drive_transfer`. Move `drive_transfer` into corresponding interface (e.g. `sec_rxf_intf`) and put the instance of `sec_rxf_intf` in HW.

5.1 sec_rxf_driver.sv

- **SETUP**: This interface is quite synthesizable and appears that we could use single-interfaces solution. Also, interface contains mostly synthesizable objects and does not require clean-up.
- **CODE**: The following shows some representative part of the driver.

```
1
class sec_rxf_driver_c extends uvm_driver#(sec_rxf_trans_c);
   virtual task run_phase(uvm_phase phase);
      super.run_phase(phase);
      fork
         do_run_wrapper();
         monitor_reset();
      join
   endtask: run_phase
```

```
// the main BFM
task do_run_wrapper();
   fork
      begin
         @(posedge vintf.reset_n iff is_first_reset == TRUE);
         ‘uvm_info(inst_name, $psprintf("Reset is ended."), UVM_NONE)
         reset_has_started = FALSE;
         run_pid = process::self();
         ‘uvm_info(inst_name, $psprintf("Current process id = %0d", %run_pid), UVM_FULL)
         main_bfm();
         wait fork;
      end
   join_none
endtask: do_run_wrapper

// the task to monitor reset
virtual task monitor_reset();
   forever begin
      // Detect the first reset start.
      if (is_first_reset == FALSE) begin
         @(vintf.clkx2 iff (vintf.reset_n == 0 && reset_has_started == FALSE));
         ‘uvm_info(inst_name, $psprintf("Reset is started."), UVM_MEDIUM)
         is_first_reset = TRUE;
         reset_has_started = TRUE;
         init_bfm();
      end
      // Detect the seconad or later reset start.
      else begin
         @p_seqr.reset_started_e;
         ‘uvm_info(inst_name, $psprintf("Triggered reset start event from sqr."), UVM_HIGH)
         ‘uvm_info(inst_name, $psprintf("Reset is started."), UVM_LOW)
         reset_has_started = TRUE;
         // If reset is detected, execute rerun.
         rerun();
      end
   end
endtask: monitor_reset

// rerun when multiple reset is asserted.
task rerun();
   ‘uvm_info(inst_name, $psprintf("Rerun is started."), UVM_LOW)
   if(run_pid) begin
      run_pid.kill();
      ‘uvm_info(inst_name, $psprintf("%0d process is killed.", run_pid), %UVM_MEDIUM)
   end
   // Initialize variables and cleanup when multiple reset is asserted.
   init_bfm();
   // Execute main bfm.
   do_run_wrapper();
endtask: rerun
```

```
//----------------------------------------
// Main logic for BFM
//----------------------------------------
virtual task main_bfm();
   `uvm_info(inst_name, $psprintf("Start main_bfm."), UVM_FULL)
   seq_item_port.get_next_item(req);
   `uvm_info(inst_name, $psprintf("Get items: %s", req.sprint()), %UVM_MEDIUM)
   drive_transfer(req);
   $cast(rsp, req.clone());
   `uvm_info(inst_name, $psprintf("Get rsp: %s", rsp.sprint()), %UVM_FULL)
   rsp.set_name("rsp");
   rsp.set_id_info(req);
   seq_item_port.item_done();
endtask: main_bfm

//----------------------------------------
// Main logic for driving phase
//----------------------------------------
protected task drive_transfer(sec_rxf_trans_c req);
   @(posedge vintf.clkx2);

   case (p_cfg.rat_mode_cfg)
      "4g" : begin
               fork
                 if (req.in_on[0][0]) drv_rxf_4g(0,0);
                 if (req.in_on[1][0]) drv_rxf_4g(1,0);
                 if (req.in_on[0][1]) drv_rxf_4g(0,1);
                 if (req.in_on[1][1]) drv_rxf_4g(1,1);
                 if (req.in_on[0][2]) drv_rxf_4g(0,2);
                 if (req.in_on[1][2]) drv_rxf_4g(1,2);
               join
             end
      "3gf" : begin
               fork
                 begin
                   // input driving offset for 3G
                   repeat (p_cfg.rx_start_offset) @ (posedge vintf.clkx2);
                   if (p_cfg.rat_mode_cfg!="4g") begin
                     // synchronizing even/odd phase offset
                     @ (posedge vintf.Tclk[0] iff
                     vintf.iRxfDCR0VClkEnable_0 & vintf.iRxfDCF0VClkEnable_0);
                     repeat (4) @ (posedge vintf.clkx2);
                   end
                   vintf.RX_START <= 1;
                   repeat (16) @ (posedge vintf.clkx2);
                   vintf.RX_START <= 0;
                 end
                 if (req.in_on[0][0]) drv_rxf (0,0);
                 if (req.in_on[1][0]) drv_rxf (1,0);
                 if (req.in_on[0][1]) drv_rxf (0,1);
                 if (req.in_on[1][1]) drv_rxf (1,1);
                 if (req.in_on[0][2]) drv_rxf (0,2);
                 if (req.in_on[1][2]) drv_rxf (1,2);
               join
             end
      ...
   endcase

   // implement driving logic here.
   # (5000);
   if (p_cfg.rat_mode_cfg=="3gt" && p_cfg.srch_3gt_on==1)
   #50_000_000;
endtask: drive_transfer

task automatic drv_rxf_4g (int ant, int c);
   int n = 2*c + ant;
   int pre_bw;

   `uvm_info(inst_name,$sformatf("drv_rxf_4g waiting.... (ant%0d,c%0d)", ant,c),UVM_NONE)
   @(posedge vintf.TtiTick[n]);
   `uvm_info(inst_name,$sformatf("TtiTick[%0d] released.... (ant%0d,c%0d)",n,...,UVM_NONE)

   pre_bw = vintf.CurBW[n];
   `uvm_info(inst_name,$sformatf("drv_rxf_4g driving waiting.... (ant%0d,c%0d)",...,UVM_NONE)

       `ifdef UVM_TB_s333ap
        if ((vintf.SarMode[n] && (vintf.CurBW[n] > 2)) ||
            (!vintf.SarMode[n] && (vintf.CurBW[n] > 4)))
       `else
        if (vintf.CurBW[n] > 4)
       `endif
        @(posedge vintf.InClk[n]);

        repeat (p_cfg.rf_in_offset) @ (posedge vintf.InClk[n]);
        `uvm_info(get_type_name(),$sformatf("drv_rxf_4g driving starts....'', ...)

        vintf.DriveEn[n] = 1;

        for (int i=0;i< req.in_data[ant][c].size();) begin
           `ifdef UVM_TB_s333ap
           if ((vintf.SarMode[n] && ((vintf.PreBW[n] >= 3) &&
                 (vintf.GapEn[n] || ((vintf.AgapEn[n])&(vintf.AgapBW[n]<=2))))) ||
               (vintf.SarMode[n] && ((vintf.PreBW[n] == 4) &&
                 (vintf.GapEn[n] || ((vintf.AgapEn[n])&(vintf.AgapBW[n]==5))))) ||
                (!vintf.SarMode[n] && ((vintf.PreBW[n] == 5) &&
                  (vintf.GapEn[n] || ((vintf.AgapEn[n])&(vintf.AgapBW[n]!=5)))))) begin
           `else
               if ((vintf.PreBW[n] == 5) && ((vintf.GapEn[n]) |
                    ((vintf.AgapEn[n])&(vintf.AgapBW[n]!=5)))) begin
           `endif
                  vintf.AntRxAdc[2*c+ant] = req.in_data[ant][c][i++];
               end
            `ifdef UVM_TB_s333ap
                if ((vintf.SarMode[n] && ((vintf.PreBW[n] >= 3) &&
                      (vintf.CurBW[n]<=2)&&(vintf.GapInfo[c]==0)&&(vintf.GapHold[c]==1))) ||
                    (!vintf.SarMode[n] && ((vintf.PreBW[n] == 5) &&
                      (vintf.CurBW[n]!=5)&&(vintf.GapInfo[c]==0))) ) begin
                     vintf.AntRxAdc[2*c+ant] = req.in_data[ant][c][i++];
                end
            `else
                if ((vintf.PreBW[n] == 5) &&
                    (vintf.CurBW[n]!=5)&&(vintf.GapInfo[c]==0)) begin
                   vintf.AntRxAdc[2*c+ant] = req.in_data[ant][c][i++];
                end
            `endif

            `ifdef UVM_TB_s333ap
               if ((vintf.SarMode[n] &&
                    ((~((vintf.PllSel[n]==3) && vintf.GapHold[c]
                     && (vintf.GapInfo[c]==0) && (vintf.PreBW[n]==0)))&
                    (~((vintf.CurBW[n]==5) && ((vintf.PreBW[n]!=3)&& ...) begin
            `else
               if ((~((vintf.PllSel[n]==3) && vintf.GapHold[c] &&
                   (vintf.GapInfo[c]==0) && (vintf.PreBW[n]==0)))& ...) begin
            `endif
                   vintf.AntRxAdc[2*c+ant] = req.in_data[ant][c][i++];
             end
                   @(posedge vintf.InClk[n]);
         end   // foreach end

         vintf.DriveEn[n] <= 0;
         @(posedge vintf.InClk[n]);
   endtask: drv_rxf_4g

   //----------------------------------------
   // Initialization of signals
   //----------------------------------------
   virtual function void init_bfm();
      `uvm_info(inst_name, $psprintf("Initialize signals."), UVM_MEDIUM)
      vintf.AntRxAdc[0] <= 0;
      vintf.AntRxAdc[1] <= 0;
      vintf.AntRxAdc[2] <= 0;
      vintf.AntRxAdc[3] <= 0;
      vintf.AntRxAdc[4] <= 0;
      vintf.AntRxAdc[5] <= 0;
      vintf.RX_START <= 0;
   endfunction: init_bfm
endclass: sec_rxf_driver_c
```

- **STEP #1**: Move the `drive_transfer` function into `sec_rxf_intf` interface.
  - For example,

```
interface sec_rxf_intf;
```

```
    task drive_transfer(sec_rxf_trans_c req); ... endtask

    task drv_rxf_4g(int ant, int c); ...  endtask
  endinterface
```

  - We don't need to worry whether `sec_rxf_intf` is synthesizable or not at this point (we will continue to map it to SW, for now).
  - To achieve this,
    1. All tasks called from inside `drive_transfer` needs to be moved into `sec_rxf_intf`. (e.g. `drv_rxf_4g`)
    2. All variables which is defined inside `sec_rxf_driver_c` class but used inside the body of `drive_transfer` task (or any task called from `drive_transfer` transitively, like `drv_rxf_4g`) need to be passed as arguments.
       For example, `sec_rxf_config_c p_cfg` is used inside the body of `drive_transfer` as follows.
       ```
       protected task drive_transfer(sec_rxf_trans_c req);
        @(posedge vintf.clkx2);
        case (p_cfg.rat_mode_cfg)
           "4g" : begin ... end
           "3gf": begin ... end
           "3gt": begin ... end
        endcase
       ```
       For this, we need to pass `p_cfg` object from the driver class, when calling the `drive_transfer` task defined in `sec_rxf_intf`. One possible way is to do
       ```
       vintf.drive_transfer(req, p_cfg);
       ```
    3. All references relative to vintf (e.g. `vintf.CurBW[n]`) needs to be rewritten (e.g. `CurBW[n]`).
    4. UVM macros such as `` `uvm_info `` should be commented out.
  - **Validate the code change before moving on to STEP #2.**
- **STEP #2**: Remove all dynamic objects in task definition.
  - To be continued.

6 MONITORS

6.1 base_lib_mon.sv

6.1.1 collect_type0

- **SETUP**:
  - Essentially, for each call to `collect_type0`, dynamically creates a **check_data collector** (see **CODE** below in RED).
  - It appears it's best to put **check_data collector** into HW for performance. For this the forever statement of this collector should be translated into an always process. Since it's impossible to dynamically create an always process, we need to statically define one process for each `collect_type0` call. For this, we could generate
- **CODE**:

```
task automatic collect_type0 (
   CHECK_TRANS trans,
   int check_point_idx = 0,
   int show_success = 0,
```

```
   int dbg = 0,
   int num_diff_print = 0
);
   string name;
   int cnt = 0;
   trans.show_success = show_success;
   trans.num_diff_print_flag = num_diff_print;
   name = vintf.check_point_name[check_point_idx];
   trans.data_type = vintf.check_ref_file_type[check_point_idx];
   if (dbg)
     $display("[DBG] (%0s) (%0s) collect_type0 starts...", get_type_name(), name);
   forever begin
      // wait transaction's starting
      fork
        wait (vintf.check_start[check_point_idx]);
        wait (vintf.check_param_set_end[check_point_idx].triggered);
      join
      name = vintf.check_point_name[check_point_idx];
      if (dbg)
        $display("[DBG] (%0s) (%0s) start and param_set_end is triggered.",
                 get_type_name(), name);
      // start recording
      void'(begin_tr(trans, vintf.check_point_name[check_point_idx]));
      // check point's name
      trans.point_name = vintf.check_point_name[check_point_idx];
      if (dbg)
        $display("[DBG] (%0s) (%0s) check_point_name: %0s",
                 get_type_name(), name, trans.point_name);
      // set reference file name
      trans.ref_file_name = vintf.check_ref_file_name[check_point_idx];
      if (dbg)
        $display("[DBG] (%0s) (%0s) ref_file_name: %0s",
                 get_type_name(), name, trans.ref_file_name);
      // get key value
      trans.int_key_name = vintf.check_int_key_name[check_point_idx];
      trans.int_key_value = vintf.check_int_key_value[check_point_idx];
      trans.str_key_name = vintf.check_str_key_name[check_point_idx];
      trans.str_key_value = vintf.check_str_key_value[check_point_idx];
      if (dbg)
        foreach(trans.int_key_name[i])
          $display("[DBG] (%0s) (%0s) int_key_name[%0d] = %0s",
                   get_type_name(),name,i,trans.int_key_name[i]);
      if (dbg)
        foreach(trans.int_key_value[i])
          $display("[DBG] (%0s) (%0s) int_key_value[%0d] = %0d",
                   get_type_name(),name,i,trans.int_key_value[i]);
      if (dbg)
        foreach(trans.str_key_name[i])
          $display("[DBG] (%0s) (%0s) str_key_name[%0d] = %0s",
                   get_type_name(),name,i,trans.str_key_name[i]);
      if (dbg)
```

```
  foreach(trans.str_key_value[i])
    $display("[DBG] (%0s) (%0s) str_key_value[%0d] = %0s",
              get_type_name(),name,i,trans.str_key_value[i]);
// empty queue to collect
trans.rtl_data.delete();
trans.rtl_data_i.delete();
trans.rtl_data_q.delete();
trans.mask.delete();
trans.ref_data_idx.delete();
trans.inst_time.delete();
cnt = 0 ;
// capture rtl values
fork
  // #1 check_data collector
  //  - best to execute in HW (i.e. H_interface)
  //  - collect data in HW array and notify SW to fetch them
  //  - i.e. HW call a task defined in monitor UVC
  //  - actual notification may occur in process #2
  forever begin
   // Q: what is the typical number of iterations of this forerver body?
    @(posedge vintf.check_clk[check_point_idx] iff vintf.check_data_en[check_point_idx]);
    trans.mask.push_back(vintf.check_mask[check_point_idx]);
    trans.ref_data_idx.push_back(vintf.check_ref_data_idx[check_point_idx]);
    // how to handle $time() in HW?
    trans.inst_time.push_back($time());
    if (trans.data_type == 0) begin
      trans.rtl_data.push_back(vintf.check_data[check_point_idx]);
      if (dbg)
        $display("[DBG] (%0s) (%0s) get data[%0d] (ref_idx:%0d): 0x%0h @ %0d ns",
                 get_type_name(),name,cnt,trans.ref_data_idx[cnt],
                 trans.rtl_data[cnt]&trans.mask[cnt],trans.inst_time[cnt++]);
    end else begin
      trans.rtl_data_i.push_back(vintf.check_data_i[check_point_idx]);
      trans.rtl_data_q.push_back(vintf.check_data_q[check_point_idx]);
      if (dbg)
        $display("[DBG] (%0s) (%0s) get data[%0d] (ref_idx:%0d): (0x%0h,0x%0h) @ %0d ns",
                 get_type_name(),name,cnt,trans.ref_data_idx[cnt],
                 trans.rtl_data_i[cnt]&trans.mask[cnt],
                 trans.rtl_data_q[cnt]&trans.mask[cnt],
                 trans.inst_time[cnt++]);
    end
    repeat (vintf.check_clk_skip_num[check_point_idx])
      @(posedge vintf.check_clk[check_point_idx]);
  end

  // #2 check_done waiter: for killing fork when done
  // - could be put into H-interface; but can be mapped to HW or SW
  // - if check_done_num is 1 mostly (as observed in symbproc4gc)
  //   can be mapped to SW (by creating a TB_TASK task), to avoid creating
  //   multiple copies of waiter processes
  begin
```

```
           repeat (vintf.check_done_num[check_point_idx])
             @(negedge vintf.check_done[check_point_idx]);
           if (dbg)
             $display ("[DBG] (%0s) (%0s) check_done is triggered.",
                       get_type_name(), name);
         end
       join_any
       disable fork;
      // send colleted item
       item_collected_port.write (trans);
       // end recording
       end_tr (trans);
    end
 endtask : collect_type0
```

- **STEP #1**: Move **part of** collect_type0 into interface (e.g. symbproc4gc_intf.sv)

```
// base_lib_mon.sv
class base_lib_mon_c;
  task collect_type0(CHECK_TRANS trans, ...);
    string name;
    int cnt = 0;
    ...
    forerver begin
      vintf.do_collect_type0(trans, ...);
      aport_write(trans);
    end
  endtask
endclass

// common_intf.sv
task do_collect_type0(inout CHECK_TRANS trans, ...);
  fork wait(check_start[check_point_idx];
       wait(check_param_set_end[check_point_idx].triggered;
  join;
  ...
  fork;
  join_any
  disable fork
end

// symbproc_4gc_mon.sv
// - NO change for subclasses of base_lib_mon class
fork
  if (rear_top_flag == 1 || modem_top_flag == 0)
    collect_type0(lspcch_dscr_a0301_in_bch, 0, 0, 0);
  ...
join_none
```

- **STEP #2**: Split into S-interface and H-interface
  - Rough idea about remodelling **collect_type0** is:

```
// S-interface                      // H-interface
event collect_done[];               always @(posedge check_clk iff)
task do_collect_type0(idx);           collect check_data int local mem;
  forever begin
    fork wait(check_start);         task start(int idx);
         wait (...);
    join
    prepare trans;                  repeat(check_done_num[idx])
    hintf.start(idx);                 @(negedge check_done[idx]);
    @collect_done[idx];               sintf.finish(idx);
    dpiMap::getBytes();             endtask
    populate trans;                 initial $ixc_ctrl("tb_task" "start");
    aport.write(trans);
  end
  task finish(idx)
    ->collect_done[idx];
  endtask
endtask
```

  – on H-inteface-side we need to create as many always processes as the number of collect_type0 calls

- **STEP #3**: Make H-interface synthesizable

6.1.2 collect_type1

- Similar to collect_type0

6.2 symbproc4gc_mon.sv

- No need to change

7 SEQUENCES

7.1 demod_4g_vseq_lib.sv

8 REGISTER LAYER CLASSES

9 CHECKERS

9.1 NonCol_Compare.inc

9.1.1 compare_noncol_ch

- **CODE**:

```
intial
  fork
    compare_noncol_ch(0, 0, 0, 0);
    compare_noncol_ch(0, 0, 0, 1);
    ...
  join
task automatic compare_noncol_ch(int Intf, int Cc, int Tx, int Rx);
```

```
    parse_file#(.DATA_WIDTH(11))    parse_file_h;
    parse_item#(.DATA_WIDTH(11))    parse_item_q[$];

    parse_file_h = new;
    parse_file_h.read_file("./VEC/demd4g_cecan_a0002_crs_ch.txt",
                           DATA_FILE, parse_item_q);

    foreach (parse_item_q[i]) begin
      if ((parse_item_q[i].int_key["itfk"]  == Intf) &&
          (parse_item_q[i].int_key["cck"]  == Cc) &&
          (parse_item_q[i].int_key["txk"]  == Tx) &&
          (parse_item_q[i].int_key["rxk"]  == Rx))
        fork
          begin
            for (int j = 0; j < parse_item_q[i].data[0].size(); j=j+1) begin
              @(posedge iClk iff ( w_noncol_ch_en &&
                                   r_noncol_intf_idx == Intf &&
                                   r_noncol_cc_idx == Cc &&
                                   r_noncol_port_idx == Tx &&
                                   r_noncol_rx_idx == Rx));

              r_noncol_ch_ref_re = parse_item_q[i].data[0][j];
              r_noncol_ch_ref_im = parse_item_q[i].data[1][j];
              if (w_noncol_ch_dt !=={r_noncol_ch_ref_re[10:0],
                                     r_noncol_ch_ref_im[10:0]}) begin
                $display("CE NONCOL(CHIN) ERROR %1d ns Intf %1d Cc %1d Tx "
                         "%1d Rx %1d subfr %1d %3dth Vec : %3h %3h, RTL: "
                         "%3h %3h",
                         $time,
                         parse_item_q[i].int_key["itfk"],
                         parse_item_q[i].int_key["cck"],
                         parse_item_q[i].int_key["txk"],
                         parse_item_q[i].int_key["rxk"],
                         parse_item_q[i].int_key["sfr"],
                         j,
                         r_noncol_ch_ref_re,
                         r_noncol_ch_ref_im,
                         w_noncol_ch_dt[21:11],
                         w_noncol_ch_dt[10:0]);
              end
            end
          end
        join
    end
  endtask
```

- **STEP #1**: Extract comparison logic from `compare_noncol_ch`. Also, copy reference value into HW memory.

  1. **Determine upper bound on the number of per-queue reference data items**: Need to know the upper bound on the `parse_item_q[i].data[0].size()`. That is, we need to determine the bound of the for-loop, across all possible queues. Let this**constant** value be 100.

     Also, we see there are four index values – `Intf`, `Cc`, `Tx`, `Rx`. We can easily determine the maximum value used in this file, observing the calls to compare_noncol_ch. Intf takes a value from $[-1,1]$, Cc takes a value from $[0,2]$, Tx takes a value from $[0,3]$, and Rx takes a value from $[0,1]$.

2. **Create HW memory to store reference values**: Given a call compare_noncol_ch(1, 2, 3, 4), we need to create a HW memory which will store the reference values inside HW.

   We create 4 additional packed dimensions so that we can index into the corresponding reference value using 4 index values (Intf, Cc, Tx, Rx).

```
// memory to be put into HW
//  [Intf] [Cc ] [Tx ] [Rx ]
reg [-1:1] [0:2] [0:3] [0:1] [10:0] r_noncol_ch_ref_re_mem[99/*upperbound*/:0];
reg [-1:1] [0:2] [0:3] [0:1] [10:0] r_noncol_ch_ref_im_mem[99/*upperbound*/:0];
```

3. **Create HW vector to store for-loop indices**: Given a call compare_noncol_ch(1, 2, 3, 4), we need to create a HW vector which will store the values to be used as an index to the reference memory. Its usage will be clear shortly.

```
// memory to be put into HW
//  [Intf] [Cc ] [Tx ] [Rx ]
reg [-1:1] [0:2] [0:3] [0:1] [31:0] noncol_ch_idx = 'b0;
```

4. **Add code to copy reference data into HW memory**: When a reference data is read from a file, these value needs to be transferred to HW, so that HW process will use them to compare DUT output. If we only store the reference data inside the queue, we will have to switch between HW and SW for comparison (say, get DUT value from HW and compare the value with reference data in SW queue).

```
task automatic compare_noncol_ch(int Intf, int Cc, int Tx, int Rx);
  parse_file#(.DATA_WIDTH(11))    parse_file_h;
  parse_item#(.DATA_WIDTH(11))    parse_item_q[$];

  parse_file_h = new;
  parse_file_h.read_file("./VEC/demd4g_cecan_a0002_crs_ch.txt",
                          DATA_FILE, parse_item_q);
  foreach (parse_item_q[i]) begin
    if ((parse_item_q[i].int_key["itfk"]  == Intf) &&
        (parse_item_q[i].int_key["cck"]  == Cc) &&
        (parse_item_q[i].int_key["txk"]  == Tx) &&
        (parse_item_q[i].int_key["rxk"]  == Rx))
      fork
        begin
          for (int j = 0; j < parse_item_q[i].data[0].size(); j=j+1) begin
            // copy reference data to HW memory;
            // in next STEP, we will use MARG for SW-to_HW copy
            // FROM: r_noncol_ch_ref_re = parse_item_q[i].data[0][j];
            r_noncol_ch_ref_re_mem[j][Intf][Cc][Tx][Rx] = parse_item_q[i].data[0][j];
            // FROM: r_noncol_ch_ref_im = parse_item_q[i].data[1][j] ;
            r_noncol_ch_ref_im_mem[j][Intf][Cc][Tx][Rx] = parse_item_q[i].data[1][j];
          end
        end
      join
  end
```

```
endtask
```

5. **Create HW process which compares result with reference data**: Given a call `compare_noncol_ch(1, 2, 3, 4)`, we need to create a HW always process.

```
always @(posedge iClk) begin
  if (w_noncol_ch_en && w_noncol_intf_idx == 1 &&
                        w_noncol_cc_idx == 2 &&
                        w_noncol_port_idx == 3 &&
                        w_noncol_rx_idx == 4)) begin
    r_noncol_ch_ref_re = r_noncol_ch_ref_re_mem[noncol_ch_idx[1][2][3][4]] [1][2][3][4];
    r_noncol_ch_ref_im = r_noncol_ch_ref_im[noncol_ch_idx[1][2][3][4]]  [1][2][3][4];
    if (w_noncol_ch_dt !=={r_noncol_ch_ref_re[10:0],
                           r_noncol_ch_ref_im[10:0]}) begin
      noncol_ch_error(1, 2, 3, 4,
                      noncol_ch_idx[1][2][3][4],
                      r_noncol_ch_ref_re,
                      r_noncol_ch_ref_im,
                      w_noncol_ch_dt[21:11],
                      w_noncol_ch_dt[10:0]);
    end
  end
end

task noncol_ch_error(input int Intf, int Cc, int Port, int Rx,
                      int idx,
                      input [10:0] r_noncol_ch_ref_re,
                      input [10:0] r_noncol_ch_ref_im,
                      input [10:0] w_noncol_ch_dt_re,
                      input [10:0] w_noncol_ch_dt_im)
      $display(‘‘CE NONCOL(CHIN) ERROR %1d, ...’’,
               $time,
               parse_item_q[noncol_ch_idx[1][2][3][4]].int_key["itfk"],
               parse_item_q[noncol_ch_idx[1][2][3][4]].int_key["cck"],
               ...
               parse_item_q[noncol_ch_idx[1][2][3][4]].int_key["sfr"],
               j,
               r_noncol_ch_ref_re,
               r_noncol_ch_ref_im,
               w_noncol_ch_dt_re,
               w_noncol_ch_dt_im);
endtask
initial $ixc_ctrl("tb_import", "noncol_ch_error");
```

To avoid manual efforts, macro can be used:

```
‘define COMPARE_NONCOL_CH_HW(INTF, CC, PORT, RX) \
   initial noncol_ch_idx[INTF][CC][PORT][RX] = 0; \
   always @(posedge iClk) begin \
     if (w_noncol_ch_en && w_noncol_intf_idx == INTF && \
                           w_noncol_cc_idx == CC && \
              w_noncol_port_idx == PORT && \
              w_noncol_rx_idx == RX)) begin \
   r_noncol_ch_ref_re = \
    r_noncol_ch_ref_re_mem[noncol_ch_idx[INTF][CC][PORT][RX]][INTF][CC][PORT][RX]; \
   r_noncol_ch_ref_im = \
    r_noncol_ch_ref_im_mem[noncol_ch_idx[INTF][CC][PORT][RX]][INTF][CC][PORT][RX]; \
   if (w_noncol_ch_dt !=={r_noncol_ch_ref_re[10:0], \
              r_noncol_ch_ref_im[10:0]}) begin \
    noncol_ch_error(INTF, CC, PORT, RX, \
            noncol_ch_idx[INTF][CC][PORT][RX]], \
            r_noncol_ch_ref_re, \
            r_noncol_ch_ref_im, \
            w_noncol_ch_dt[21:11], \
            w_noncol_ch_dt[10:0]); \
  end \
 end \
end
```

6. **NOTE**: Before proceeding to STEP #2, be sure to validate using SW-run that all transformation is correct. After STEP #1, HW-run will not be correct since we are not actually performing HW-SW value transfer inside `compare_noncol_ch` task (i.e. the code in BLUE in STEP #1.4.

- **STEP #2**: Add MARG function calls for actual memory transfer (i.e. populate memory for reference data).

  1. **Create MARG handles for the HW memories**: For each HW memory, we creatd in STEP #1, create MARG handles.

```
int r_noncol_ch_ref_re_mem_mah, r_noncol_ch_ref_im_mah;
int r_noncol_ch_ref_re_vmh, r_noncol_ch_ref_im_vmh;
initial begin
 r_noncol_ch_ref_re_mem_mah = marg_new(72*11/*width*/, 100 /*depth*/);
 r_noncol_ch_ref_im_mem_mah = marg_new(72*11/*width*/, 100 /*depth*/);
 r_noncol_ch_ref_re_mem_vmah = marg_vmem_handle("r_noncol_ch_ref_re_mem");
 r_noncol_ch_ref_im_mem_vmah = marg_vmem_handle("r_noncol_ch_ref_im_mem");
end
```

  2. **Use MARG function calls to copy reference data into HW memory**: The BLUE code in STEP #1.3 will only populate the SW-side copy of the memory. To actually populate the HW-side memory, we need to explicitly use MARG functions. For this we need to execute the following code.

```
for (int i = 0; i < 100; i++) begin
 marg_write(r_noncol_ch_ref_re_mah, i, r_noncol_ch_ref_re_mem[i]);
 marg_write(r_noncol_ch_ref_im_mah, i, r_noncol_ch_ref_im_mem[i]);
end
marg_put(r_noncol_ch_ref_re_mah, 0, 100, r_noncol_ch_ref_re_vmh, 0);
marg_put(r_noncol_ch_ref_im_mah, 0, 100, r_noncol_ch_ref_im_vmh, 0);

```

This code can be executed only once regardless of the numer of calls to the `compare_noncol_ch` task However, this code must be executed only after after all `compare_noncol_out` tasks have finished it execution. For this, put the code after the `fork-join` statement.

```

initial
 fork
  compare_noncol_out(0, 0, 0, 0);
```

```
      ...
      compare_noncol_y(1, 0, 2, 1);
    join
    r_noncol_ch_ref_re_mem_mah = marg_new(72*11/*width*/, 100 /*depth*/);
    r_noncol_ch_ref_im_mem_mah = marg_new(72*11/*width*/, 100 /*depth*/);
    r_noncol_ch_ref_re_mem_vmah = marg_vmem_handle("r_noncol_ch_ref_re_mem");
    r_noncol_ch_ref_im_mem_vmah = marg_vmem_handle("r_noncol_ch_ref_im_mem");
    for (int i = 0; i < 100; i++) begin
      marg_write(r_noncol_ch_ref_re_mah, i, r_noncol_ch_ref_re_mem[i]);
      marg_write(r_noncol_ch_ref_im_mah, i, r_noncol_ch_ref_im_mem[i]);
    end
    marg_put(r_noncol_ch_ref_re_mah, 0, 100, r_noncol_ch_ref_re_vmh, 0);
    marg_put(r_noncol_ch_ref_im_mah, 0, 100, r_noncol_ch_ref_im_vmh, 0);
  end
```

9.1.2 compare_noncol_out

- Simliar to `compare_noncol_ch` but 6 memories are needed for reference data (`r_noncol_ref0_exp`, `r_noncol_ref0_re`, `r_noncol_ref0_im`, `r_noncol_ref1_exp`, `r_noncol_ref1_re`, `r_noncol_ref1_im`).

9.1.3 compare_noncol_y

- Simliar to `compare_noncol_out`.

9.2 Pbch_compare.inc

- Similar to the handling of `NonCol_Compare.inc`.

9.3 ce_pp_compare.inc

- Similar to the handling of `NonCol_Compare.inc`.

9.4 ce_y_compare.inc

- Similar to the handling of `NonCol_Compare.inc`.

9.5 compare_ToneMapper.inc

-

9.6 fft_checker.inc

- Similar to the handling of `NonCol_Compare.inc`, except that the task is not parameteized here.

9.7 pdp_checker.inc

- **NOTE**: It appears that this code does not have any observable behavior. It performs comparison but the always processes at the end of the file, which performs ERROR reporting, is commented out. Unless waveform itself is used for regression testing, this chcker code looks like DEAD CODE.