Adopting UVM methodology for IP level Verification

The JPEG1 SV verification project

Giovanni AUDITORE, Francesco RUA'
STMicroelectronics

Jun 18, 2015

Grenoble, France

Agenda

Project overview

Verification strategy

Verification project implementation

Evaluation of Synopsys verification platform

Project overview

The primary target of this project was to fully verify the JPEG1 Digital IP built around a Synopsys JPEG codec core.

On this project we decided to insert two activities:

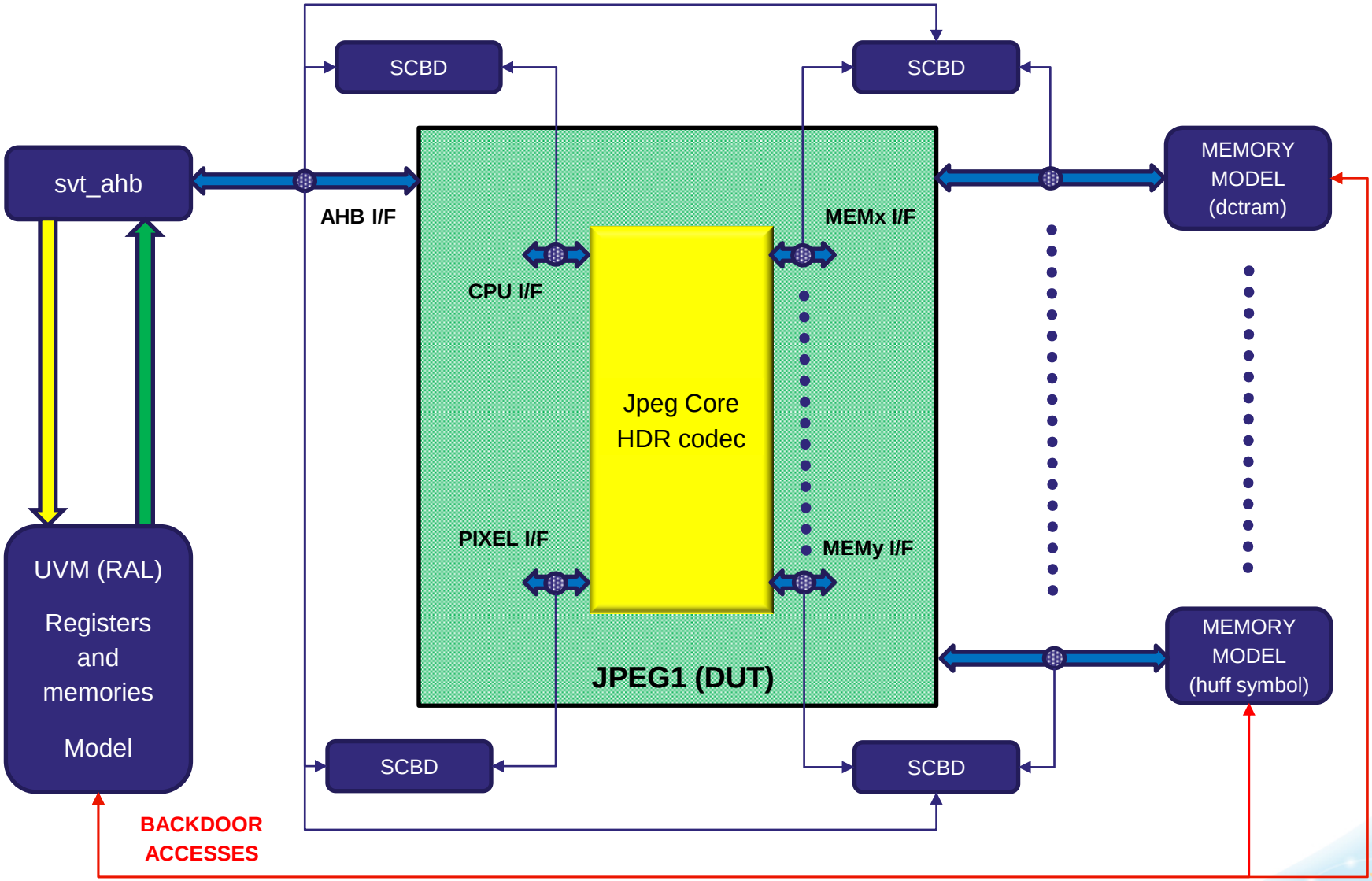
- The investigation around the System Verilog language in its UVM standard flavor and the effort it takes to adopt this verification methodology for a team already experienced in r constrained random verification techniques
- The evaluation of the Synopsys dynamic digital verification platform

Verification strategy

Focus on reuse

- Choose standard VIP svt_ahb in UVM flavor for AHB protocol implementation and checks
- Take advantage of UVM registers and memory models
- Extend existing objects of UVM library to implement the testbench scoreboards
- Implement the DUT custom interfaces following UVM guidelines
- Exploit testbench components communication and synchronization by mean of UVM events
- Configure the reusable components through UVM factory

Verification Project Implementation



Integrate the svt_ahb VIP

- Install the VIP component and its UVM use examples
- Configure the component
- Instantiate the configured component into the environment

Integrate the svt_ahb VIP

Install the component

The svt_ahb requires to be installed prior to be used.

The install command for all Synopsys VIP is dw_vip_setup.

Example install command:

SV UVM Testbench

```
dw_vip_setup -p ${SV_VERIF_BASE}/dw_vip -a ahb_system_env_svt -svtb
```

User path

VIP env

Integrate the svt_ahb VIP

Configure the component

```
class jpeg1_ahb_system_configuration extends svt_ahb_system_configuration;
  /** UVM Object Utility macro */
  `uvm_object_utils (jpeg1_ahb_system_configuration)
  /** Class Constructor */
  function new (string name = "jpeg1_ahb_system_configuration");
    super.new(name);
    this.num_masters = 1; this.num_slaves  = 2;
    /** Create port configurations.*/
    this.create_sub_cfgs(num_masters,num_slaves);
    /** Set all configuration parameters*/
    //this.master_cfg[0].<parameter> = <value>; …;
    //this.slave_cfg[0].<parameter>  = <value>;…;
    /** Set interface as AHB_LITE */
    this.ahb_lite = 1;
    /** Configure the address map */
    this.set_addr_range(0, 14'h0000, 14'h1FFF);
  endfunction
endclass
```

JPEG env specific AHB configuration

Set number of AHB masters and slaves

Allocates the master/slave configurations

Set parameters, if required

Set the address range for a specified slave (slave[0])

Integrate the svt_ahb VIP

Integrate the component

```
class jpeg1_system_configuration extends uvm_object;
  `uvm_object_utils_begin(jpeg1_system_configuration)
  `uvm_object_utils_end
  // Configuration objects
  jpeg1_ahb_system_configuration  ahb_cfg;
  // … //other_system_configuration    other_cfg;

  function new(string name = "jpeg1_system_configuration");
     super.new(name);
     ahb_cfg = jpeg1_ahb_system_configuration::type_id::create("ahb_cfg");
  endfunction // new
endclass // jpeg1_system_configuration
```

JPEG env specific AHB configuration instance and creation into JPEG env system configuartion

Integrate the svt_ahb VIP

Integrate the component

```
`class jpeg1_tb_env extends uvm_env;
  /** AHB System ENV */
  svt_ahb_system_env  ahb_system_env;
  /** JPEG1 System Configuration */
  jpeg1_system_configuration  cfg;
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    if (!uvm_config_db#(jpeg1_system_configuration)::get(this, "", "cfg",
cfg)) `uvm_fatal(…)
    uvm_config_db#(svt_ahb_system_configuration)::set(this, "ahb_system_env",
"cfg", cfg.ahb_cfg);
...
    ahb_system_env = svt_ahb_system_env::type_id::create("ahb_system_env",
this);
   endfunction
   //Connect Master and slave interfaces into connect_phase()
endclass
```

GET the global config from the DB, then propagate (SET) the sub-configs

Create sub-ENVs which, in turn, will get their config from DB, propagating specific settings to sub-components and so on

Integrate UVM registers and memories model

UVM provides standard components to model both registers and memories. Here follow the steps we adopt to implement a mirror of the IP's registers and memories:

- Write registers and memories description in RAL format
- Run ralgen to generate UVM code
- Connect the generated code to the environment

Integrate UVM registers and memories model

RAL Format example (.ralf file)

```
block jpeg1_regmodel {
  bytes 4;
  cover +a-b+f;

  register JPEG_CONFR0 @'h0000 {
    bytes 4;

    field START {
      bits 1;
      access wo;
      reset 1'b0;
      cover +b;
    }
  }
 memory HUFFENC_RAM (huffenc_ram_generate_external.huffencmem.Mem) @'h0500 {
    size 192;
    bits 32;
    access rw;
  }
}
```

Globally activate/deactivate coverage models for address, register fields and their data bits

Locally include/exclude inherited global settings

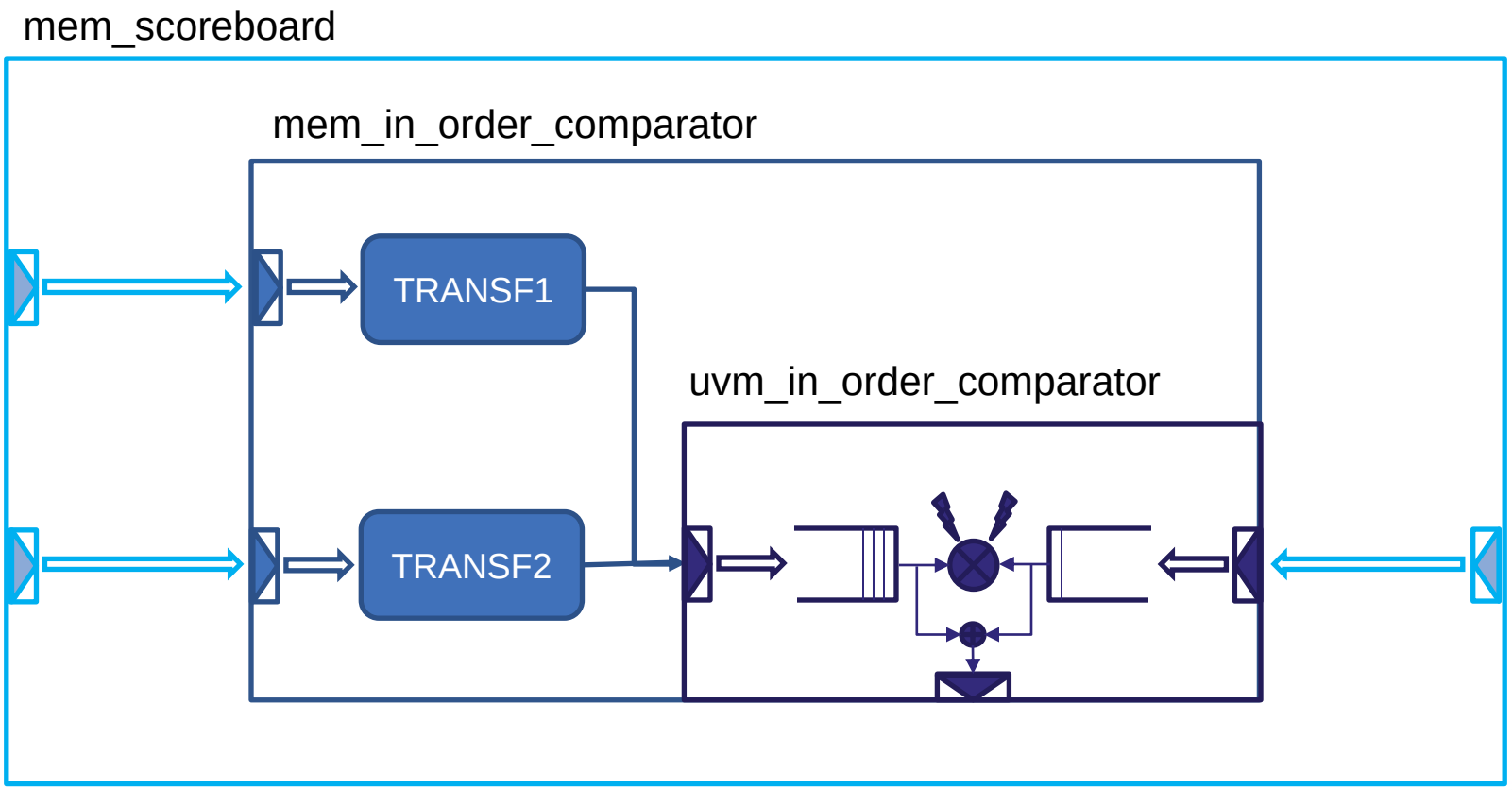
Integrate UVM registers and memories model

UVM generated code

```
class ral_reg_jpeg1_regmodel_JPEG_CONFR0 extends uvm_reg;
 rand uvm_reg_field START;
   local uvm_reg_data_t m_data;
   local uvm_reg_data_t m_be;
   local bit            m_is_read;
   covergroup cg_bits ();
   option.per_instance = 1;
   option.name = get_name();
   START: coverpoint {m_data[0:0], m_is_read} iff(m_be) {
       wildcard bins bit_0_wr_as_0 = {2'b00};
       wildcard bins bit_0_wr_as_1 = {2'b10};
       option.weight = 2;
   }
   endgroup
   covergroup cg_vals ();
   option.per_instance = 1;
       START_value : coverpoint START.value[0:0] {
   option.weight = 2;
  }
 endgroup : cg_vals
 function new(string name = "jpeg1_regmodel_JPEG_CONFR0");
  super.new(name, 32,build_coverage(UVM_CVR_REG_BITS+UVM_CVR_FIELD_VALS));
  if (has_coverage(UVM_CVR_REG_BITS))
   cg_bits = new();
  if (has_coverage(UVM_CVR_FIELD_VALS))
   cg_vals = new();
 endfunction: new
   virtual function void build();
      this.START = uvm_reg_field::type_id::create("START",,get_full_name());
      this.START.configure(this, 1, 0, "WO", 0, 1'b0, 1, 0, 1);
   endfunction: build
```

```
`uvm_object_utils(ral_reg_jpeg1_regmodel_JPEG_CONFR0)
virtual function void sample(uvm_reg_data_t data,
                             uvm_reg_data_t byte_en,
                             bit            is_read,
                             uvm_reg_map    map);
   if (get_coverage(UVM_CVR_REG_BITS)) begin
      m_data    = data;
      m_be      = byte_en;
      m_is_read = is_read;
      cg_bits.sample();
   end
 endfunction

 function void sample_values();
    super.sample_values();
    if (get_coverage(UVM_CVR_FIELD_VALS)) begin
       if(cg_vals!=null) cg_vals.sample();
    end
 endfunction
endclass : ral_reg_jpeg1_regmodel_JPEG_CONFR0
```

Scoreboards extension and reuse



Scoreboards extension and reuse

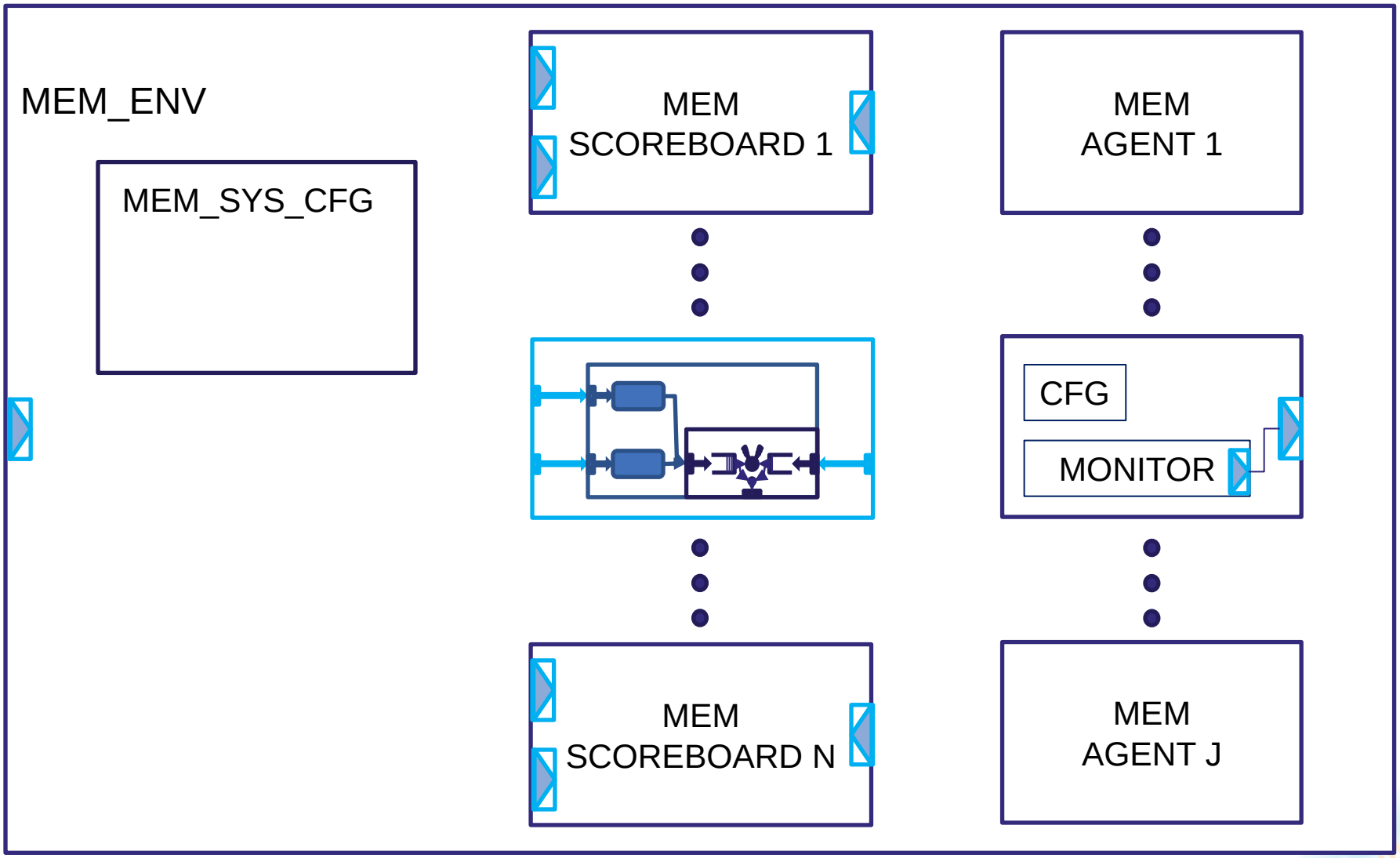
```
class mem_in_order_class_comparator #(
  string MEM_NAME   = "GENERIC MEMORY",
  type TRANSFORMER1 = ahb2mem_transformer,
  type TRANSFORMER2 = jpg2mem_transformer,
  type C            = mem_comp
) extends uvm_in_order_comparator #(
  mem_transaction,
  C,
  uvm_class_converter #(mem_transaction),
  uvm_class_pair #(mem_transaction, mem_transaction)
);
typedef mem_in_order_class_comparator #(MEM_NAME, TRANSFORMER1, TRANSFORMER2, C) this_type;
  `uvm_component_param_utils(this_type)

  const static string type_name = "mem_in_order_class_comparator #(MEM_NAME, TRANSFORMER1,
TRANSFORMER2, C)";
  uvm_analysis_imp_mem_before1 #(svt_ahb_transaction, this_type) before_export1;
  uvm_analysis_imp_mem_before2 #(mem_transaction, this_type)     before_export2;
  uvm_analysis_imp_mem_after   #(mem_transaction, this_type)     aux_after_export;
...
```

Reusable memory comparator with configurable transformers

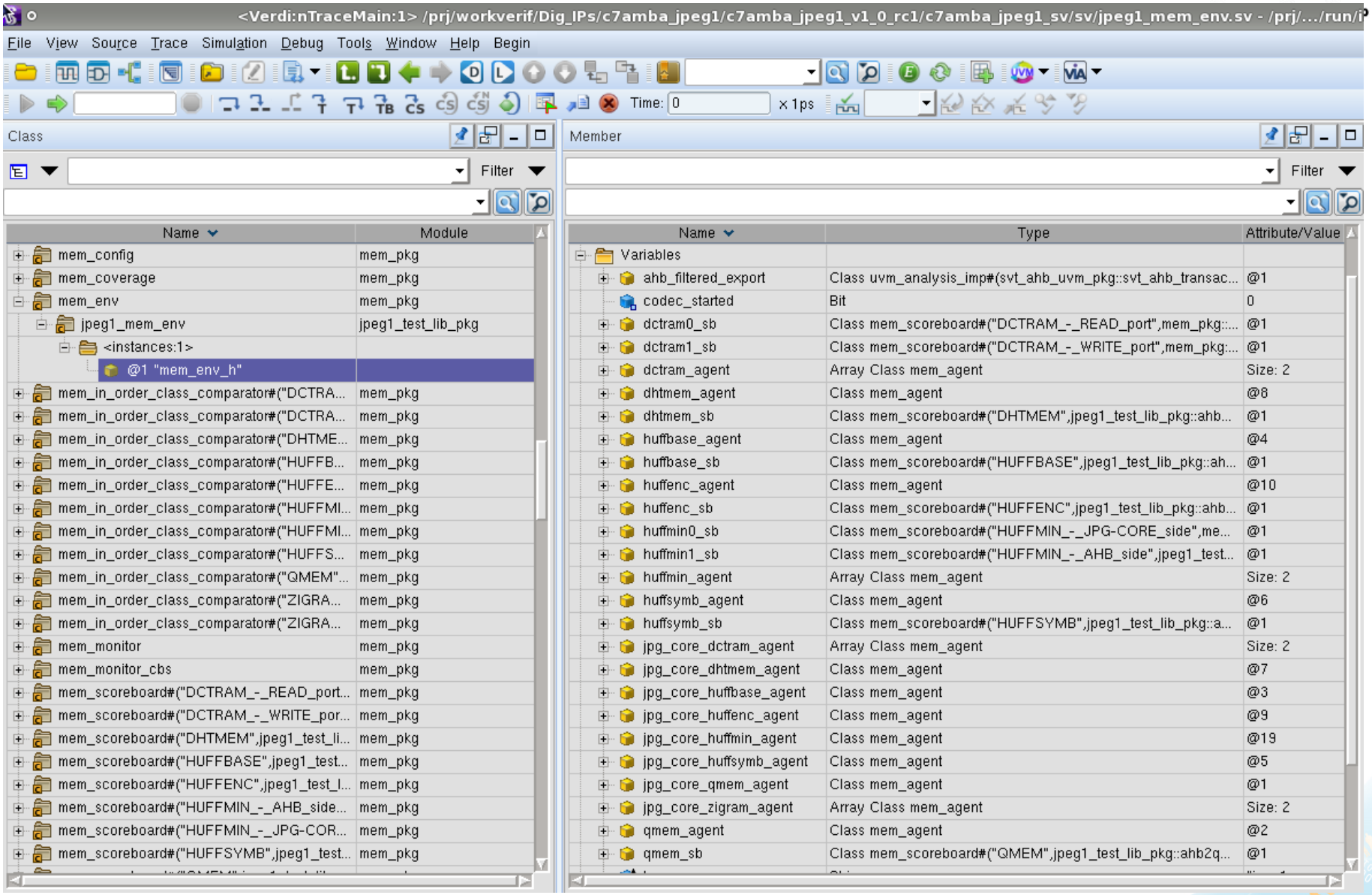
Extended from uvm_in_order_comparator class

Connection port of type implementation

MEM UVM component



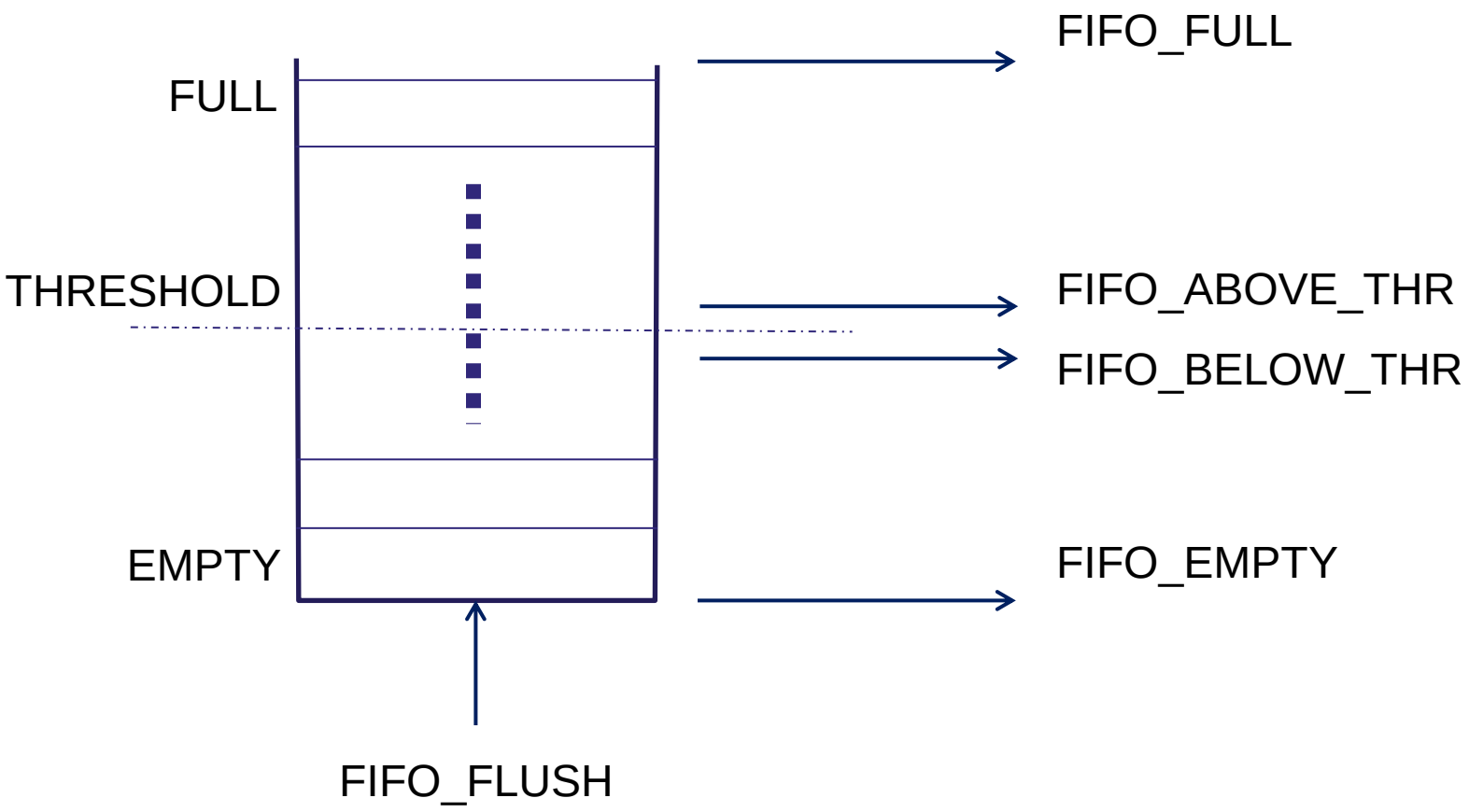
MEM UVM component



Communication and synchronization

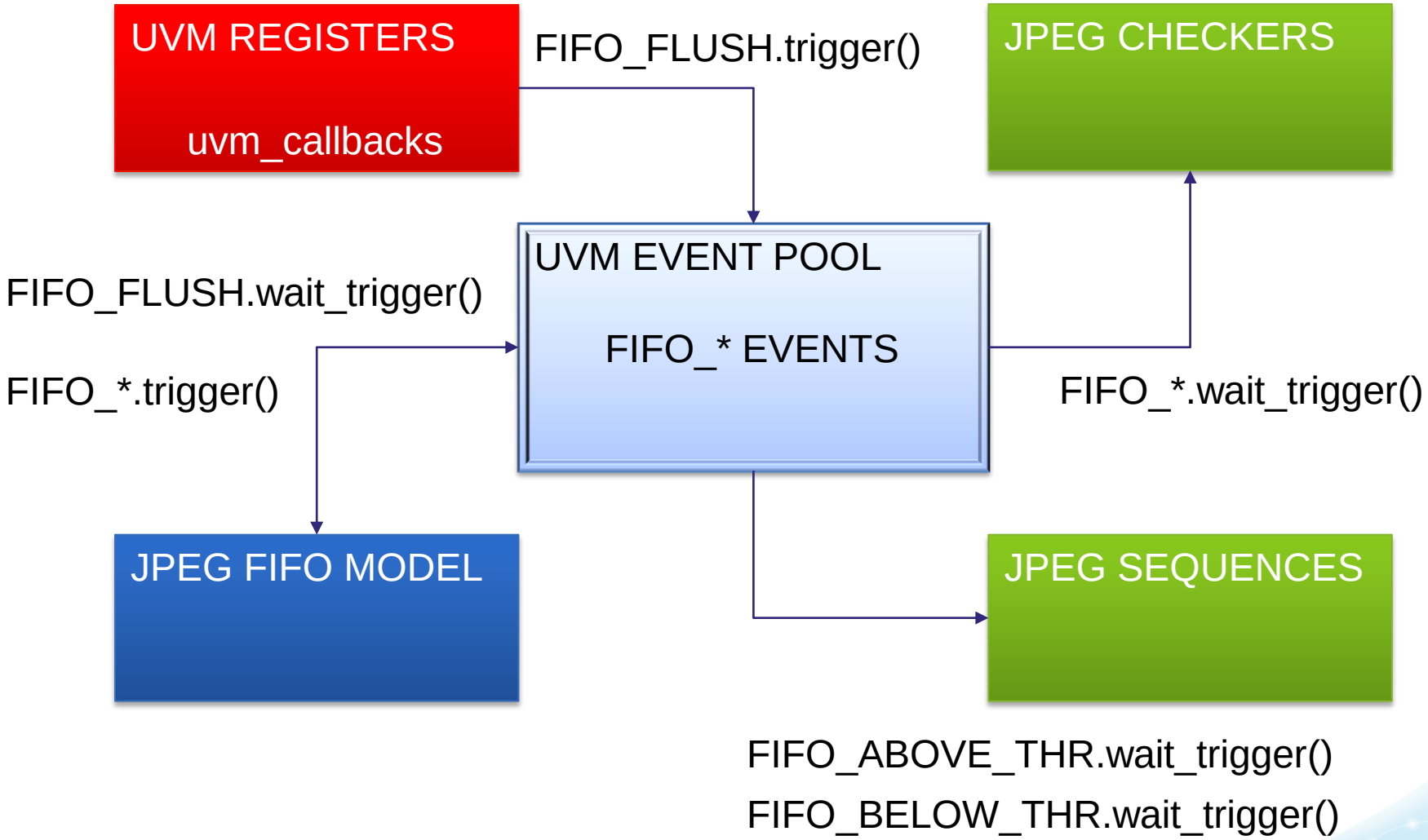
Handling FIFO controls and state changes

INPUT/OUTPUT FIFO



Communication and synchronization

The UVM event pool



Communication and synchronization

The UVM event pool

Input FIFO instance

Input FIFO UVM events

Get/create events from/into uvm_event_pool

```
188 function void fifo_model::connect_phase(uvm_phase phase);
189   string fifo_flush_s;
190   string fifo_full_s;
191   string fifo_empty_s;
192   string fifo_above_threshold_s;
193   string fifo_below_threshold_s;
194
195   super.connect_phase(phase);
196   fifo_name = get_name().toupper();
197   fifo_flush_s = $sformatf("%s_FLUSH",fifo_name);
198   fifo_flush_e = uvm_event_pool::get_global(fifo_flush_s);
199   fifo_full_s = $sformatf("%s_FULL",fifo_name);
200   fifo_full_e  = uvm_event_pool::get_global(fifo_full_s);
201   fifo_empty_s = $sformatf("%s_EMPTY",fifo_name);
202   fifo_empty_e = uvm_event_pool::get_global(fifo_empty_s);
203   fifo_above_threshold_s = $sformatf("%s_ABOVE_THR",fifo_name);
204   fifo_above_threshold_e = uvm_event_pool::get_global(fifo_above_threshold_s);
205   fifo_below_threshold_s = $sformatf("%s_BELOW_THR",fifo_name);
206   fifo_below_threshold_e = uvm_event_pool::get_global(fifo_below_threshold_s);
207   `uvm_info($sformatf("%s",fifo_name),
208             $sformatf(" \nfifo_flush_s = %s \nfifo_full_s = %s \nfifo_empty_s =
209                       fifo_flush_s,fifo_full_s,fifo_empty_s,fifo_above_thresho
```

UVM Factory

Enhance SV reuse through factory best practices

- Always use factory macros to register all objects into the factory
- Make use of uvm_config_db to:
  - Propagate top down the configuration settings into the environment through the use of set/get commands
  - Connect the agents virtual interfaces to the top environment actual interfaces
  - Overwrite the default sequence of the sequencer with the required test sequence
- Use set_instance/type_override_by_type to replace a base object with an extended object

UVM Factory

Verdi resource view

Selected a resource (configuration in this example)

<Verdi:nTraceMain:1> /prj/workverif/Dig_IPs/c7amba_jpeg1/c7amba_jpeg1_v1_0_rc1/c7amba_jpeg1_sv/mem/src/mem_scoreboard.sv - /prj/.../run/inter.fsdb (on

File View Source Trace Simulation Debug Tools Window Help Begin

Time: 0 x 1ps

<Resource_View:3>

File View Trace Tools

Name	Scope	Value	Type
cfg	uvm_test_top.env	(class jpeg1_system_configuration) @1	class jpeg1_test_lib_pkg::jpeg1_system_configuration
cfg	uvm_test_top.env.ahb_system_env	(class jpeg1_ahb_system_configuration) @1	class svt_ahb_uvm_pkg::svt_ahb_system_configuration
cfg	uvm_test_top.env.ahb_system_env.master[0]	(class svt_ahb_master_configuration) @56	class svt_ahb_uvm_pkg::svt_ahb_master_configuration
cfg	uvm_test_top.env.ahb_system_env.master[0].driver	(class svt_ahb_master_configuration) @56	class svt_ahb_uvm_pkg::svt_ahb_master_configuration
cfg	uvm_test_top.env.ahb_system_env.master[0].monitor	(class svt_ahb_master_configuration) @56	class svt_ahb_uvm_pkg::svt_ahb_master_configuration
cfg	uvm_test_top.env.ahb_system_env.master[0].sequencer	(class svt_ahb_master_configuration) @56	class svt_ahb_uvm_pkg::svt_ahb_master_configuration
cfg	uvm_test_top.env.ahb_system_env.sequencer	(class jpeg1_ahb_system_configuration) @2	class svt_ahb_uvm_pkg::svt_ahb_system_configuration
cfg	uvm_test_top.env.ahb_system_env.slave[0]	(class svt_ahb_slave_configuration) @39	class svt_ahb_uvm_pkg::svt_ahb_slave_configuration
cfg	uvm_test_top.env.ahb_system_env.slave[0].monitor	(class svt_ahb_slave_configuration) @63	class svt_ahb_uvm_pkg::svt_ahb_slave_configuration
cfg	uvm_test_top.env.ahb_system_env.slave[1]	(class svt_ahb_slave_configuration) @40	class svt_ahb_uvm_pkg::svt_ahb_slave_configuration
cfg	uvm_test_top.env.ahb_system_env.slave[1].driver	(class svt_ahb_slave_configuration) @69	class svt_ahb_uvm_pkg::svt_ahb_slave_configuration
cfg	uvm_test_top.env.ahb_system_env.slave[1].monitor	(class svt_ahb_slave_configuration) @69	class svt_ahb_uvm_pkg::svt_ahb_slave_configuration
cfg	uvm_test_top.env.ahb_system_env.slave[1].sequencer	(class svt_ahb_slave_configuration) @69	class svt_ahb_uvm_pkg::svt_ahb_slave_configuration
cfg	uvm_test_top.env.cpu_agent_h	(class cpu_config) @1	class cpu_pkg::cpu_config
cfg	uvm_test_top.env.fifo_env_h	(class jpeg1_fifo_system_configuration) @1	class fifo_pkg::fifo_system_configuration
cfg	uvm_test_top.env.fifo_env_h.decode_fifo_agent	(class fifo_config) @2	class fifo_pkg::fifo_config
cfg	uvm_test_top.env.fifo_env_h.encode_fifo_agent	(class fifo_config) @1	class fifo_pkg::fifo_config
cfg	uvm_test_top.env.mem_env_h	(class jpeg1_mem_system_configuration) @1	class mem_pkg::mem_system_configuration
cfg	uvm_test_top.env.mem_env_h.dctram_agent[0]	(class mem_config) @16	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.dctram_agent[1]	(class mem_config) @18	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.dhtmem_agent	(class mem_config) @8	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.huffbase_agent	(class mem_config) @4	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.huffenc_agent	(class mem_config) @10	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.huffmin_agent[0]	(class mem_config) @20	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.huffmin_agent[1]	(class mem_config) @21	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.huffsymb_agent	(class mem_config) @6	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.jpg_core_dctram_agent[0]	(class mem_config) @15	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.jpg_core_dctram_agent[1]	(class mem_config) @17	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.jpg_core_dhtmem_agent	(class mem_config) @7	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.jpg_core_huffbase_agent	(class mem_config) @3	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.jpg_core_huffenc_agent	(class mem_config) @9	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.jpg_core_huffmin_agent	(class mem_config) @19	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.jpg_core_huffsymb_agent	(class mem_config) @5	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.jpg_core_qmem_agent	(class mem_config) @1	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.jpg_core_zigram_agent[0]	(class mem_config) @11	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.jpg_core_zigram_agent[1]	(class mem_config) @13	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.qmem_agent	(class mem_config) @2	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.zigram_agent[0]	(class mem_config) @12	class mem_pkg::mem_config
cfg	uvm_test_top.env.mem_env_h.zigram_agent[1]	(class mem_config) @14	class mem_pkg::mem_config

Action	Scope	Value	Time	Accessor
CFGDB/SET	uvm_test_top.env	(class jpeg1_test_lib_pkg::jpeg1_system_configuration) ?	0	uvm_test_top
CFGDB/GET	uvm_test_top.env	(class jpeg1_test_lib_pkg::jpeg1_system_configuration) ?	0	uvm_test_top.env

Stopped: 0ps

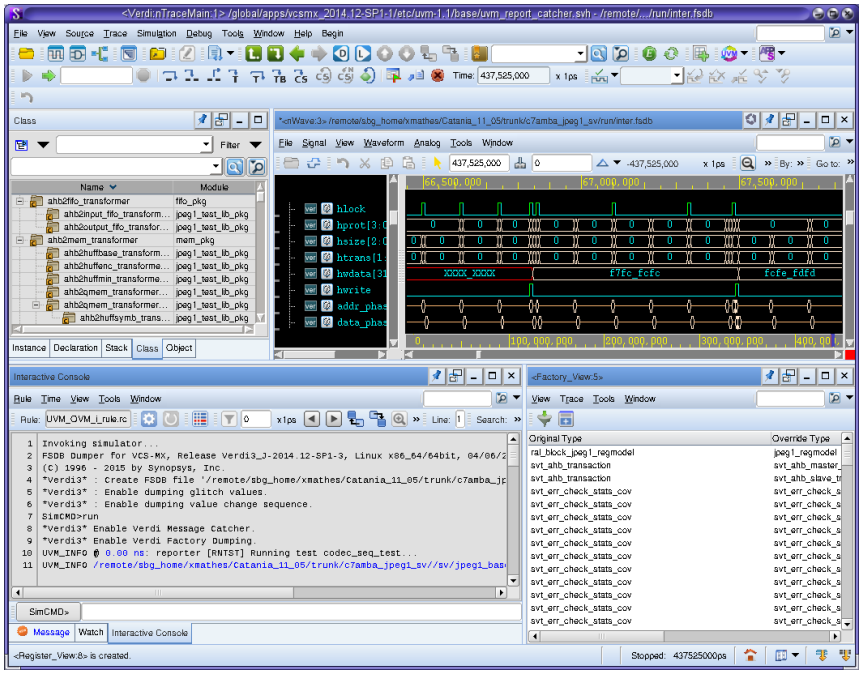
Evaluation of Synopsys verification platform

VCSMX and Verdi GUI

Synopsys Users Group

Provides test run and debug GUI capabilities

- Pros
  - Fast run time
  - Friendly GUI
  - Good UVM tools offers
  - Good step by step debug into UVM code



- Cons
  - Quite long compile and elaborate time, likely due to UVM lib overhead and to dual compilation (improved in VCSMX j-2014.12)
  - Some stability issues faced, most of them quickly solved by SNPS

Execution Manager

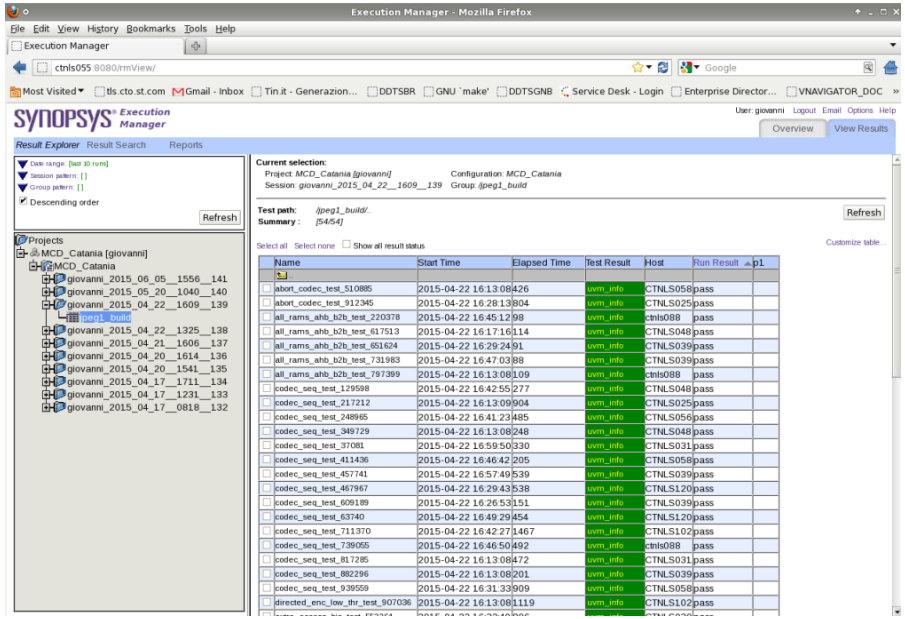
Synopsys Users Group

Provides regression handling and coverage merge capability

- Pros
  - Easy to customize

  by mean of user scripts
  - Easy to access HTML

  based regression monitor interface

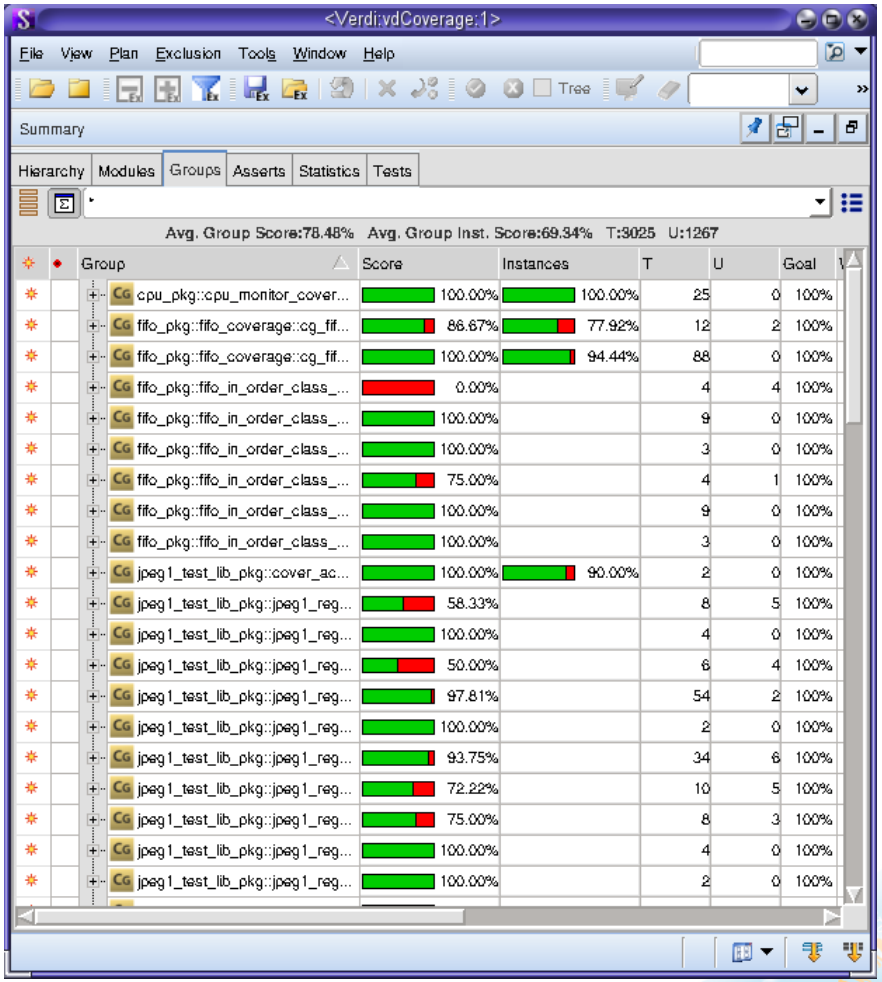


- Cons
  - Quite basic results collection and error analysis capabilities
  - Some stability issues faced into LSF interface
  - Installation flow not compliant with ST security policies

Verdi coverage

Provides coverage result analysis and management capabilities

- Pros
  - Integrated functional and code coverage results analysis
  - Very friendly GUI
  - Good coverage exclusion annotation mechanism
  - Very good support into exclusion management across different RTL versions
- Cons
  - None



Conclusions and next steps

Conclusion

- System Verilog together with UVM libraries and guidelines provide a flexible, reliable and configurable dynamic verification methodology
- Adoption of above methodology has a significant ramp-up penalty, even for verification engineers already experienced in constrained random verification techniques
- Synopsys verification platform helps into reducing this penalty with a set of UVM utilities and a good debug engine

What next?

- Improve our SV coding style target to verification reuse learning from the mistakes done
- Complete the Synopsys tools portfolio evaluation adding the missed Certitude (testbench qualification tool, trial runs already started on this) and HVP (verification plan with spec annotation capability, trials yet to be planned)

life.augmented

Synopsys Users Group

Thank You