ALU
UVM Framework
Step By Step Guide

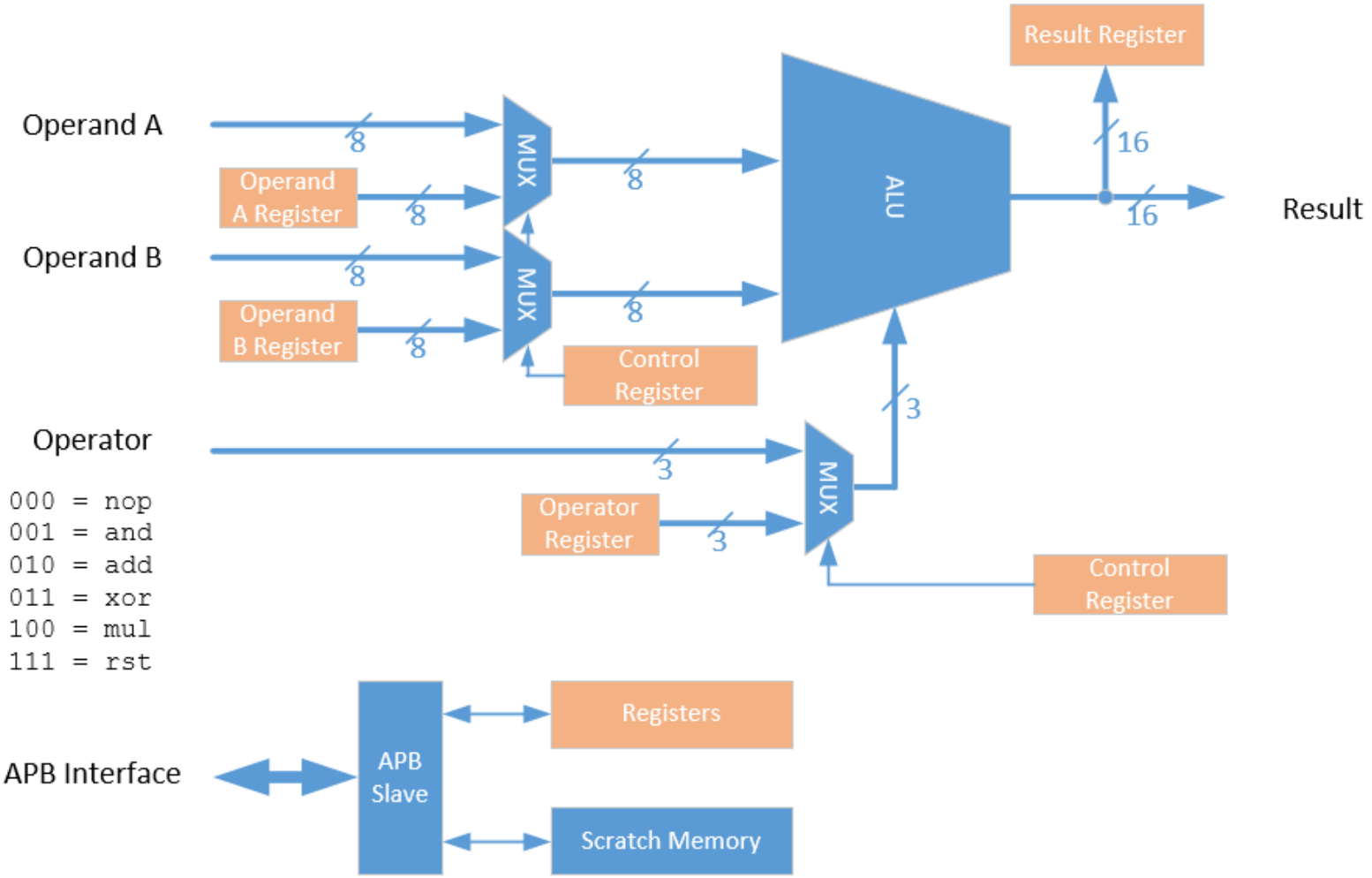
January 2020

Mentor®
A Siemens Business

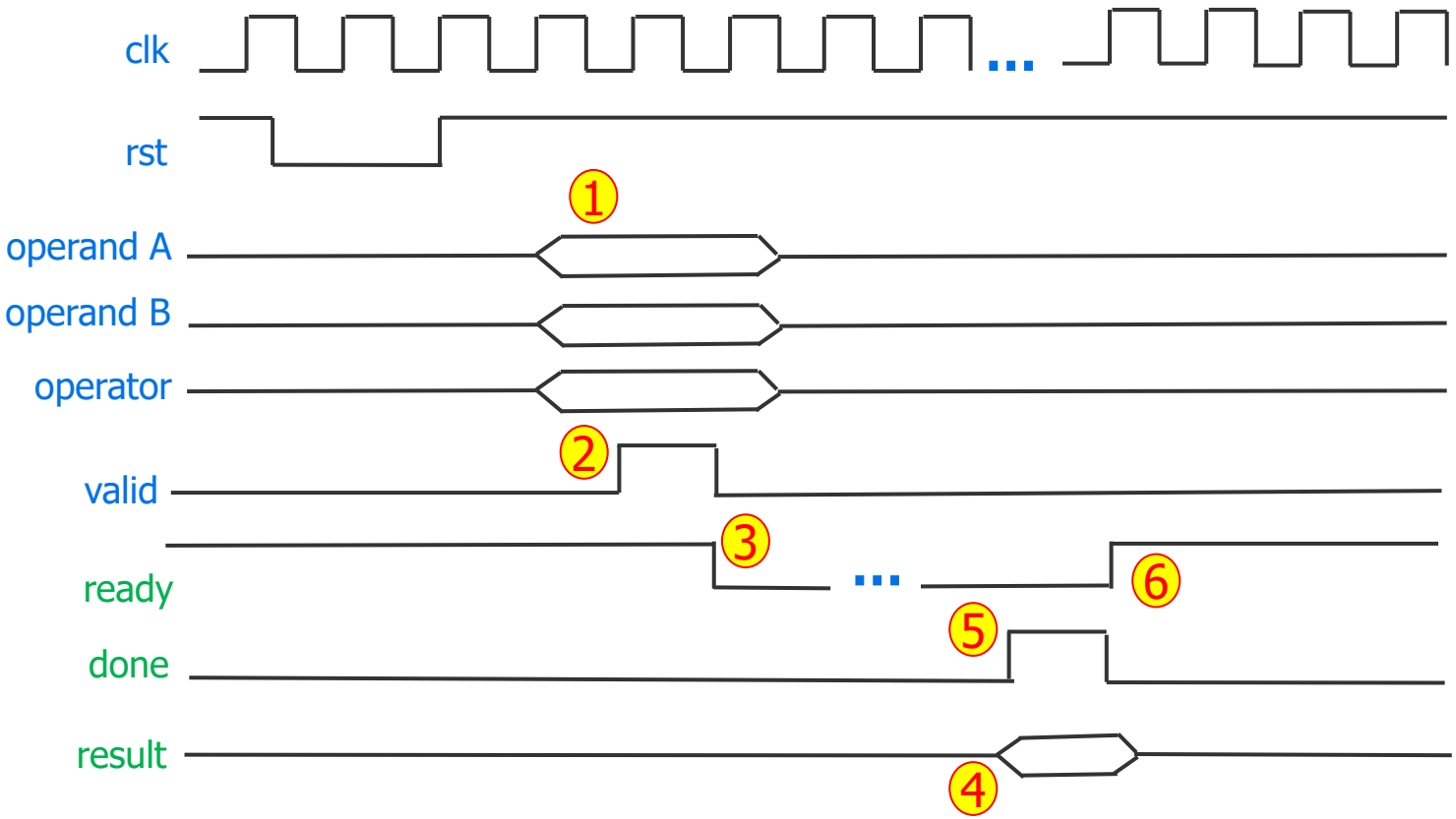
Agenda

- ALU Overview
- Config Files Explained
- Compile and Simulate Generated Code
- Adding DUT Specific Functionality
- Generate and integrate the Register Model
- Add functional coverage

ALU : Block Diagram



ALU : Timing Diagram



1. Apply operands & operation on i/p pins
2. Raise valid for 1 cycle (start)
3. ALU drops ready signals
4. After X cycles, ALU presents result
5. ALU raises done for 1 cycle
6. ALU raises ready signal

Agenda

- ALU Overview
- Config Files Explained
- Compile and Simulate Generated Code
- Adding DUT Specific Functionality
- Generate and integrate the Register Model
- Add functional coverage

YAML Config Files

- **Number Of Agents**
  - First need to determine how many interfaces there are for the ALU.
  - Group signals into interfaces
  - Create a config file for the interfaces

- **Interface Config files**
  - **ALU_IN Interface**
    - All signals that are associated with specifying the ALU operation to perform
  - **ALU_OUT Interface**
    - All signals that are associated with retrieving the ALU result

Interface Config File
alu_if_cfg.yaml

- **uvmf:**
  - First line of the YAML should be uvmf with no indention.
  - Imports Python code needed to parse the YAML config file
  - Requires $PYTHONPATH to be set since the Python packages reside in this directory

- **interfaces: (Must be indented from 'uvmf:')**
  - The next line after interfaces ("alu_in") is the name of the interface and this line must be indented from 'interfaces:'.
  - The name given will have **_pkg** appended to it to form the package name.
  - The package name (alu_in_pkg) is used to name the directory structure that contains the generated files.

```
1 uvmf:
2   interfaces:
3     "alu_in":
```

Interface Config File
alu_if_cfg.yaml

- **clock: "clk"**
  - Specifies the interface clock signal
- **reset: "rst"**
  - Specifies the interface reset signal
- **reset_assertion_level: "False"**
  - Specifies the polarity of the active reset
  - False = Active Low; True = Active High

```
4         clock: "clk"
5         reset: "rst"
6         reset_assertion_level: "False"
```

NOTES:

- The names for the clock and the reset will be used in all the generated code for the agent, including the interface.

Interface Config File
alu_if_cfg.yaml

- **parameters:**
  - Specify parameters for this interface package
  - Allow creation of parameterizable agents
  - Here we define the width of the operand to be processed by the ALU
- **hdl_typedefs:**
  - This directive will generate a typedef for the ALU operations.

```
 7      parameters:
 8        - name: "ALU_IN_OP_WIDTH"
 9          type: "int"
10          value: "8"
11      hdl_typedefs:
12        - name:  "alu_in_op_t"
13          type:  "enum bit[2:0] {no_op = 3'b000, add_op = 3'b001, and_op = 3'b010, xor_op = 3'b011, mul_op = 3'b100, rst_op = 3'b111}"
```

Interface Config File
alu_if_cfg.yaml

- **ports:**
  - The arguments to the ports: directive identify the name, width and direction of the signal.
  - The options for the direction are input, output and inout.

```
14        ports:
15          - name: "alu_rst"
16            width: "1"
17            dir: "output"
18          - name: "ready"
19            width: "1"
20            dir: "input"
21          - name: "valid"
22            width: "1"
23            dir: "output"
24          - name: "op"
25            width: "3"
26            dir: "output"
27          - name: "a"
28            width: "ALU_IN_OP_WIDTH"
29            dir: "output"
30          - name: "b"
31            width: "ALU_IN_OP_WIDTH"
32            dir: "output"
```

NOTES:

- Direction specified here is in relation to the testbench.
  - i.e. 'alu_rst' is an output from the testbench and an input pin on the DUT.
- The agent has to be able to execute a 'rst_op' operation and will need to drive the ALU reset pin in response to such a request.
- The 'a' & 'b' use the ALU_IN_OP_WIDTH parameter which was defined using the parameters: directive.

Interface Config File
alu_if_cfg.yaml

- **transaction_vars:**
  - Defines a variable to be used by the transaction class.
  - Variables in the transaction class reflect the untimed information used during a transfer on the bus.
  - The arguments required by the transaction_vars: directive include the name of the variable, variable type, an indication if this variable is to be of a randomized type & if the variable is to be compared in the do_compare method.

```
33        transaction_vars:
34          - name: "op"
35            type: "alu_in_op_t"
36            isrand: "True"
37            iscompare: "True"
38          - name: "a"
39            type: "bit [ALU_IN_OP_WIDTH-1:0]"
40            isrand: "True"
41            iscompare: "True"
42          - name: "b"
43            type: "bit [ALU_IN_OP_WIDTH-1:0]"
44            isrand: "True"
45            iscompare: "True"
46        transaction_constraints:
47          - name: "valid_op_c"
48            value: "{ op inside {no_op, add_op, and_op, xor_op, mul_op}; }"
```

NOTES:

- For the ALU, the transaction will specify the operation and the a & b operands.
- Each of these values can be randomized.
- Unconstrained arrays cannot be used with the transaction_vars: directive.
- If you need an unconstrained array, declare a fixed array in config file and modify the generated code.

Interface Config File
alu_if_cfg.yaml

- **Additional Interfaces**
  - Add the alu_out interface to the same config file as the alu_in interface.
  - Align "alu_out": interface name on line 49 to the same indention as the "alu_in": interface name on line 3.
  - Follow the same steps from the "alu_in" interface configuration slides to define the "alu_out" interface as shown below.

```
49      "alu_out":
50        clock: "clk"
51        reset: "rst"
52        reset_assertion_level: "False"
53        parameters:
54          - name: "ALU_OUT_RESULT_WIDTH"
55            type: "int"
56            value: "16"
57        ports:
58          - name: "done"
59            width: "1"
60            dir: "input"
61          - name: "result"
62            width: "ALU_OUT_RESULT_WIDTH"
63            dir: "input"
64        transaction_vars:
65          - name: "result"
66            type: "bit [ALU_OUT_RESULT_WIDTH-1:0]"
67            isrand: "False"
68            iscompare: "True"
```

Generated UVMF Code
alu_in_if

- **Generating the Interface Code**
  - Run yaml2uvmf.py on the alu_if_cfg.yaml file

$UVMF_HOME/scripts/yaml2uvmf.py alu_if_cfg.yaml

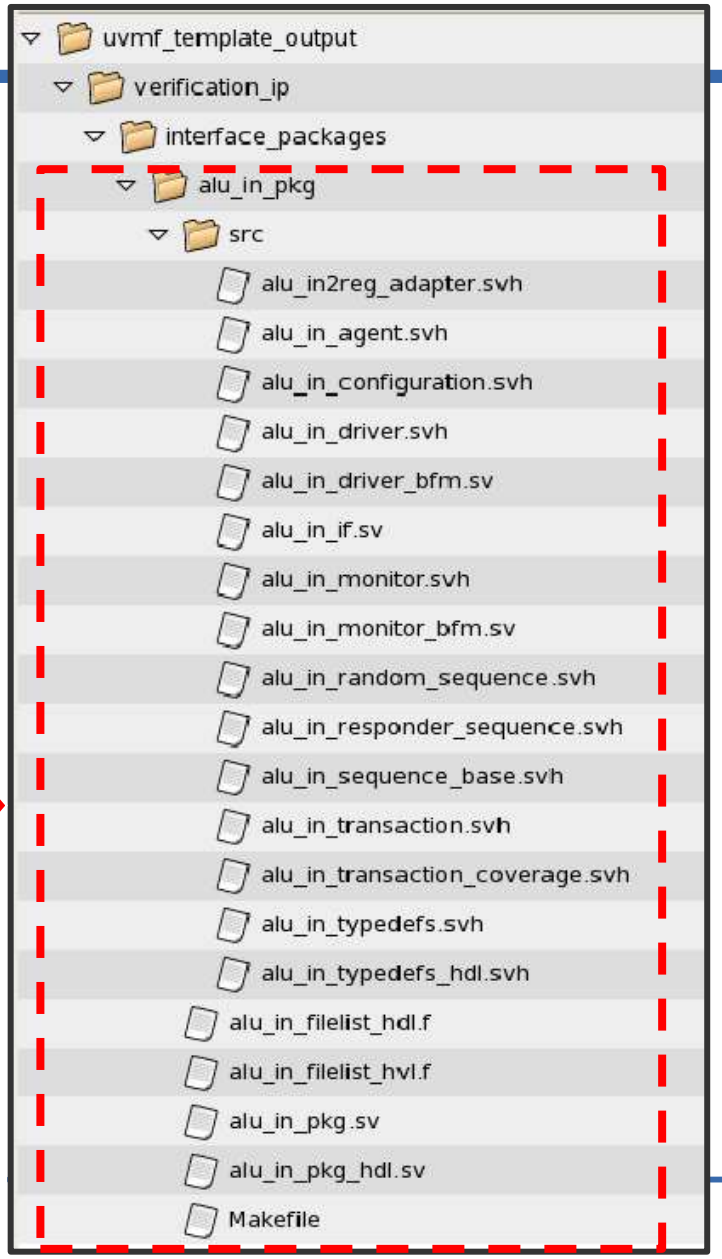
  - You can create a simple .bat file on Windows to set the $UVMF_HOME & $PYTHONPATH variables and then call python on your config file

```
@set UVMF_HOME=C:/MentorTools/questasim_10.6b/examples/UVM_Framework/UVMF_3.6f
@set PYTHONPATH=%UVMF_HOME%/templates/python

python alu_in_if_config.py
pause
```

  - All UVMF agent code is placed under **uvmf_template_output / verification_ip / interface_packages**

  - All generated code for the alu_in agent will be saved under the **alu_in_pkg** folder [as shown opposite]



Interface Config File
parameterized ALU agent

- **'parameters:' directive in alu_if_cfg.yaml**
  - All generated code for the ALU agent will be parameterized

```
interface  alu_in_if          #(
      int ALU_IN_OP_WIDTH = 8
      )
(
 input tri clk,
```

```
class alu_in_monitor  #(
      int ALU_IN_OP_WIDTH = 8
      ) extends uvmf_monitor_base
```

```
class alu_in_transaction        #(
      int ALU_IN_OP_WIDTH = 8
      ) extends uvmf_transaction_base;
```

```
class alu_in_driver  #(
      int ALU_IN_OP_WIDTH = 8
      ) extends uvmf_driver_base
```

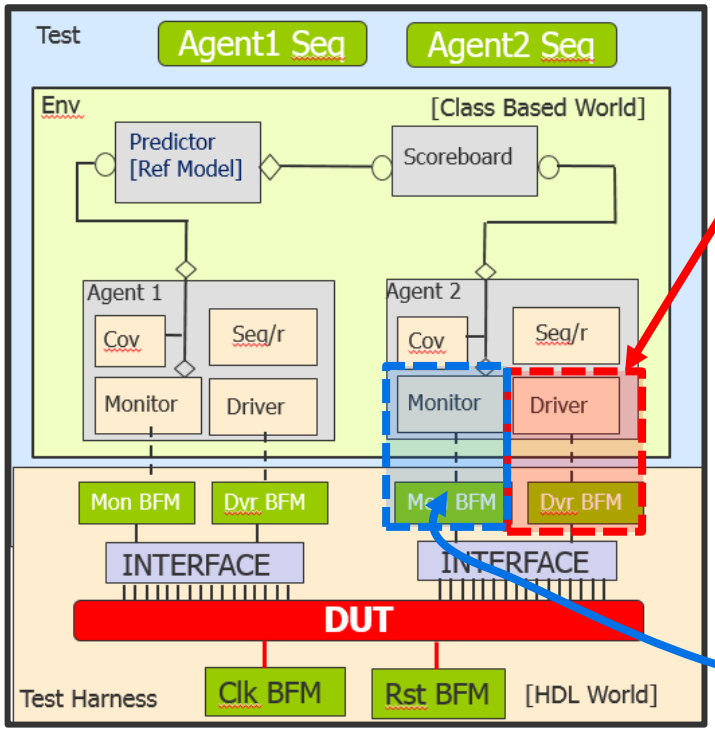
```
class alu_in_configuration  #(
      int ALU_IN_OP_WIDTH = 8
      ) extends uvmf_parameterized_agent_configuration_base
```

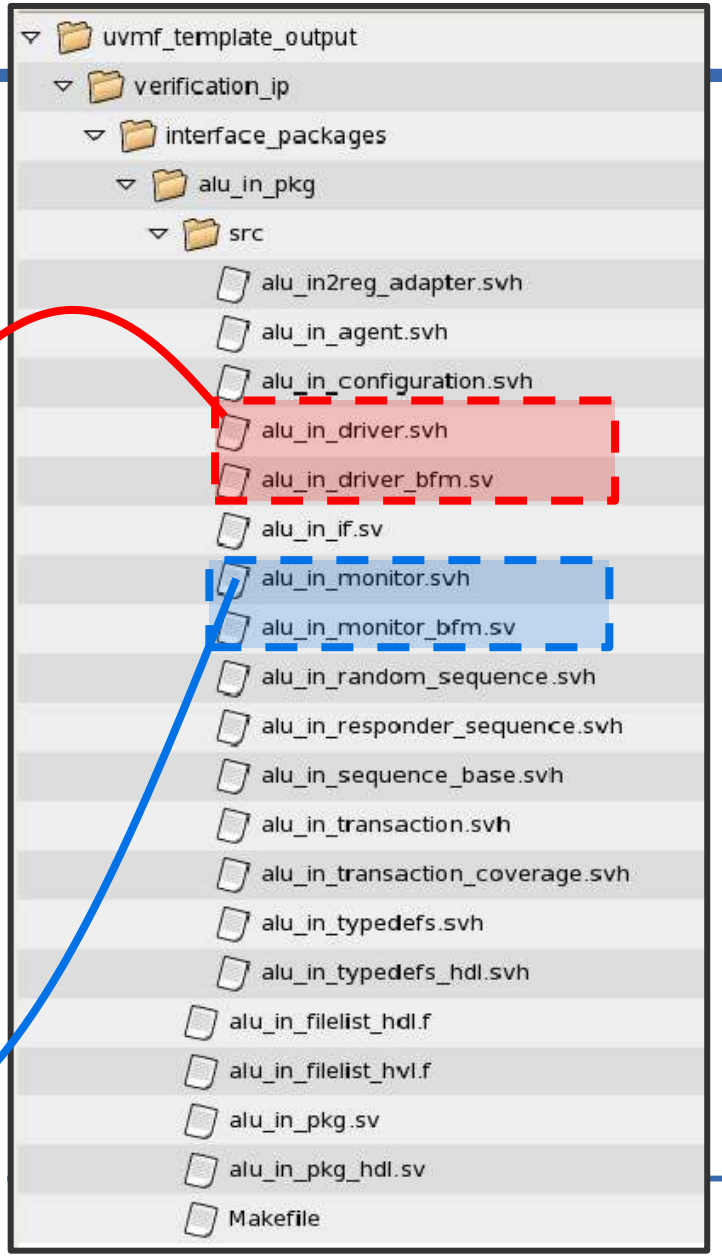
```
class alu_in_random_sequence  #(
      int ALU_IN_OP_WIDTH = 8
      )
```

```
class alu_in_agent #( int ALU_IN_OP_WIDTH = 8)extends uvmf_parameterized_agent #(
                      .CONFIG_T(alu_in_configuration #(.ALU_IN_OP_WIDTH(ALU_IN_OP_WIDTH))),
                      .DRIVER_T(alu_in_driver #(.ALU_IN_OP_WIDTH(ALU_IN_OP_WIDTH))),
```

Generated UVMF Code
alu_in

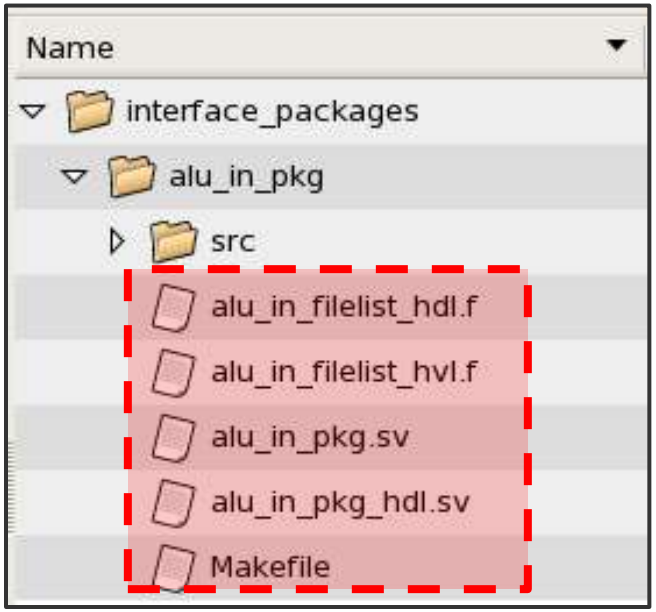
- **UVMF Transactors**
  - alu_in_driver & alu_in_monitor are class based and will be instantiated inside the agent class
  - alu_in_driver_bfm & alu_in_monitor_bfm are interfaces & will be instantiated in the top level testbench module (hdl_top)





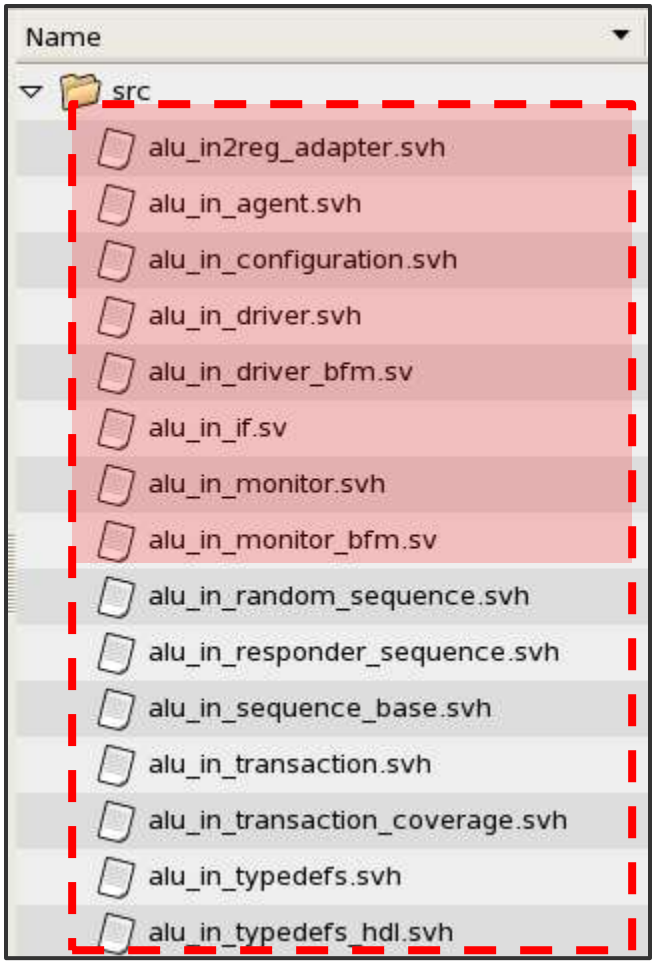
Generated UVMF Code
alu_in

- **Looking at the *alu_in_pkg* directory**
  - **Makefile**
    Contains the compile commands for the generated agent
  - **alu_in_filelist_hdl.f**
    Compilation list of HDL files (the interface and the 2 BFMs)
  - **alu_in_filelist_hvl.f**
    Compilation list of HVL files (all other files)
  - **alu_in_pkg.sv**
    This is the verification package (HVL) that `includes all the generated classes for our VIP agent (all from directory src)
  - **alu_in_pkg_hdl.sv**
    This package will be used for the HDL part of the VIP. The HDL part is synthesized by the emulator.



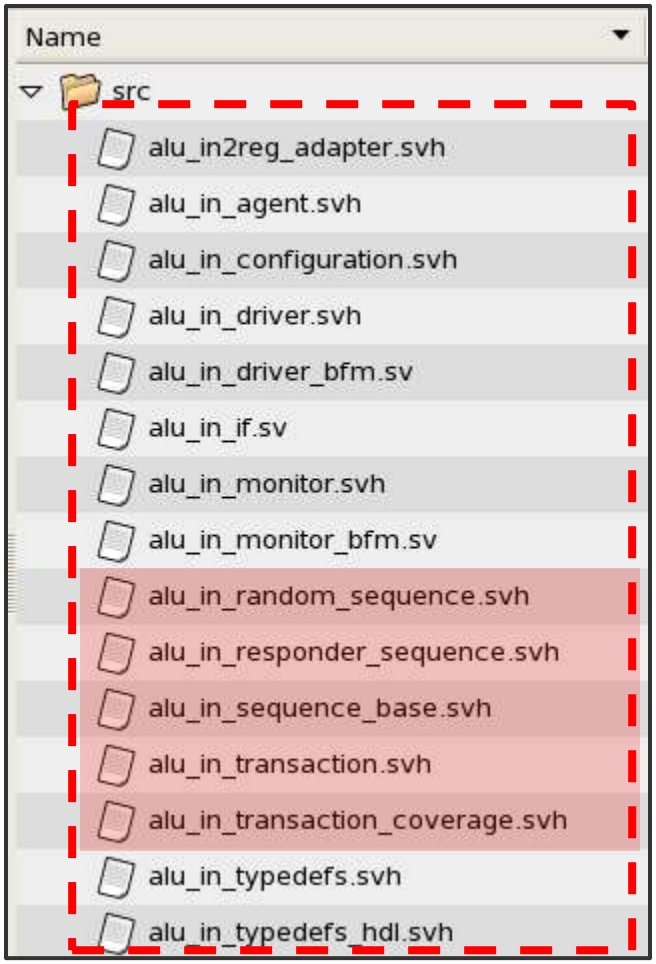
Generated UVMF Code
alu_in

- **Looking at the *alu_in_pkg/src* directory**
  - **alu_in2reg_adaptor.svh**
    Template adaptor for UVM register layer. Requires user to fill in functionality.
  - **alu_in_agent.svh**
    Agents class (parameterized)
  - **alu_in_configuration.svh**
    Configuration class for the agent
  - **alu_in_driver.svh**
    Driver class to be instantiated in the agent
  - **alu_in_driver_bfm.sv**
    Bus functional model to convert transactions to protocol pin wiggles. Requires user to fill in functionality
  - **alu_in_if.sv**
    Signal interface for the agent. User can optionally add protocol assertions in here.
  - **alu_in_monitor.svh**
    Monitor class to be instantiated in the agent
  - **alu_in_monitor_bfm.sv**
    Bus functional model to convert the protocol pin wiggles to transactions. Requires user to fill in functionality



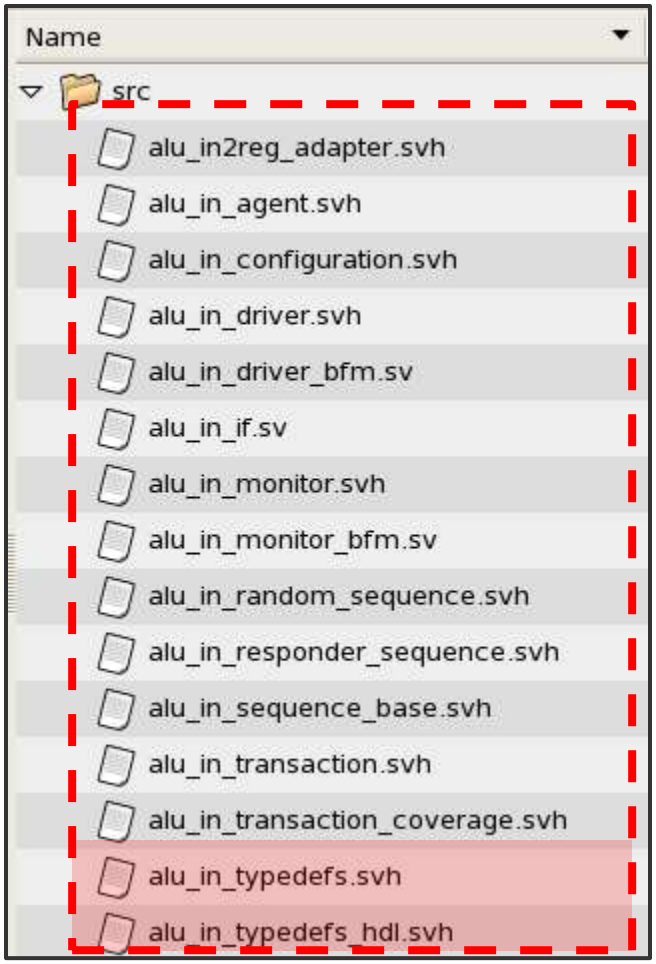
Generated UVMF Code
alu_in

- **Looking at the *alu_in_pkg/src* directory**
  - **alu_in_random_sequence.svh**
    Starter sequence. Randomizes 1 instance of the alu_in transaction class and sends to sequencer.
    Extended from alu_in_sequence_base
  - **alu_in_responder_sequence.svh**
    This sequence class can be used to provide stimulus when an interface has been configured to run in a responder mode.
    Requires user to fill in functionality
  - **alu_in_sequence_base.svh**
    Base class with useful methods that all inherited sequences can utilize
  - **alu_in_transaction.svh**
    This is the sequence item that we will use in our sequences.
    Extends from 'uvmf_transaction_base.svh' which contains global "id" which holds a unique number for every transaction.
    Also contains several methods for printing, comparing, etc
  - **alu_in_transaction_coverage.svh**
    This class records alu_in transaction information using a covergroup named alu_in_transaction_cg.
    An instance of this coverage component is instantiated in the uvmf_parameterized_agent if the has_coverage flag is set.



Generated UVMF Code
alu_in

- **Looking at the *alu_in_pkg/src* directory**
  - **alu_in_typedefs.svh**
    This file contains defines and typedefs used only in the testbench (HVL) side of the testbench. Package may not contain any defines or typedefs after but will still be generated.
  - **alu_in_typedefs_hdl.svh**
    This file contains defines and typedefs used by the interface package performing transaction level simulation activities. This package is used by the driver/monitor BFMs.



Generated UVMF Code
alu_out_if

- **Interface Code was generated from**

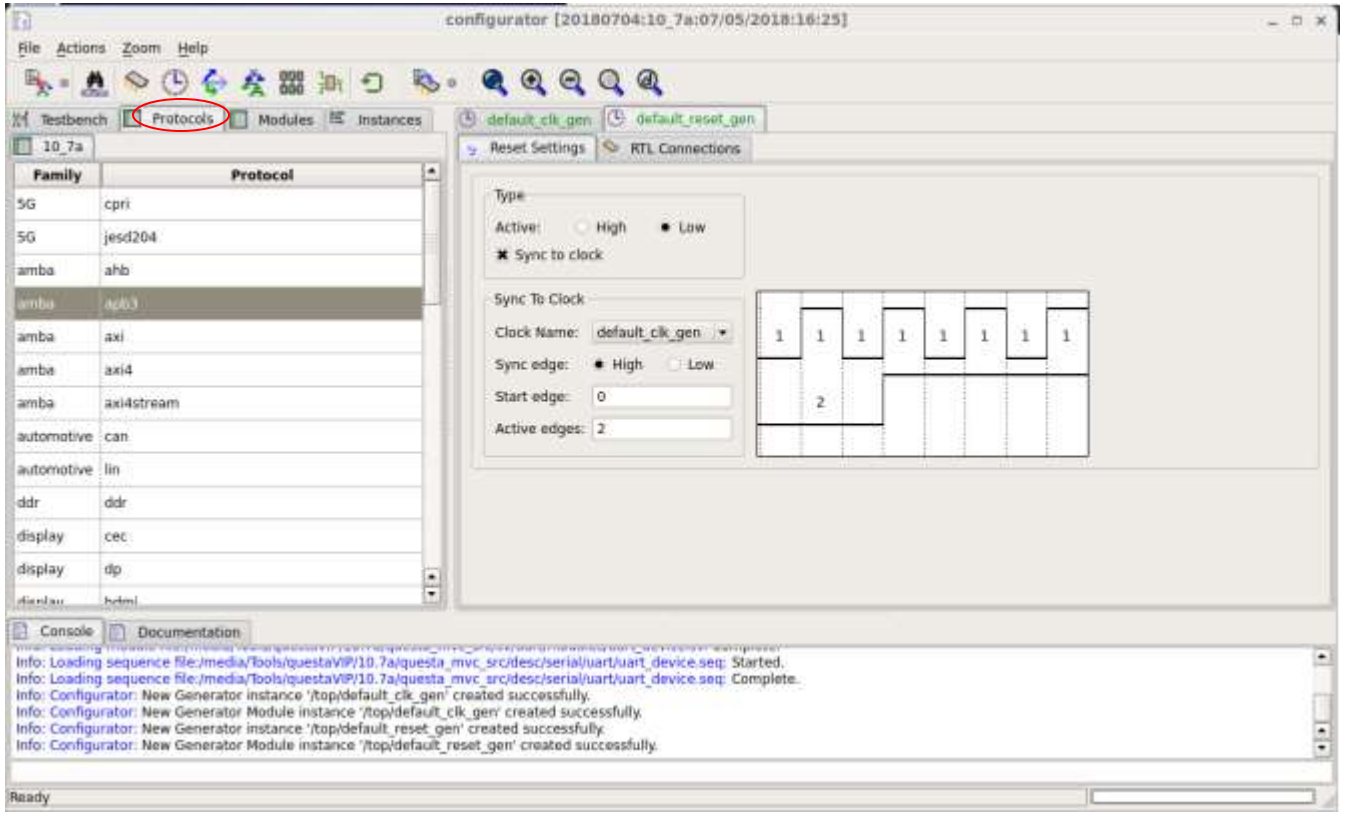
  $UVMF_HOME/scripts/yaml2uvmf.py alu_if_cfg.yaml

— All UVMF agent code is placed under **uvmf_template_output / verification_ip / interface_packages**

— All generated code for the alu_out agent will be saved under the **alu_out_pkg** folder [as shown opposite]

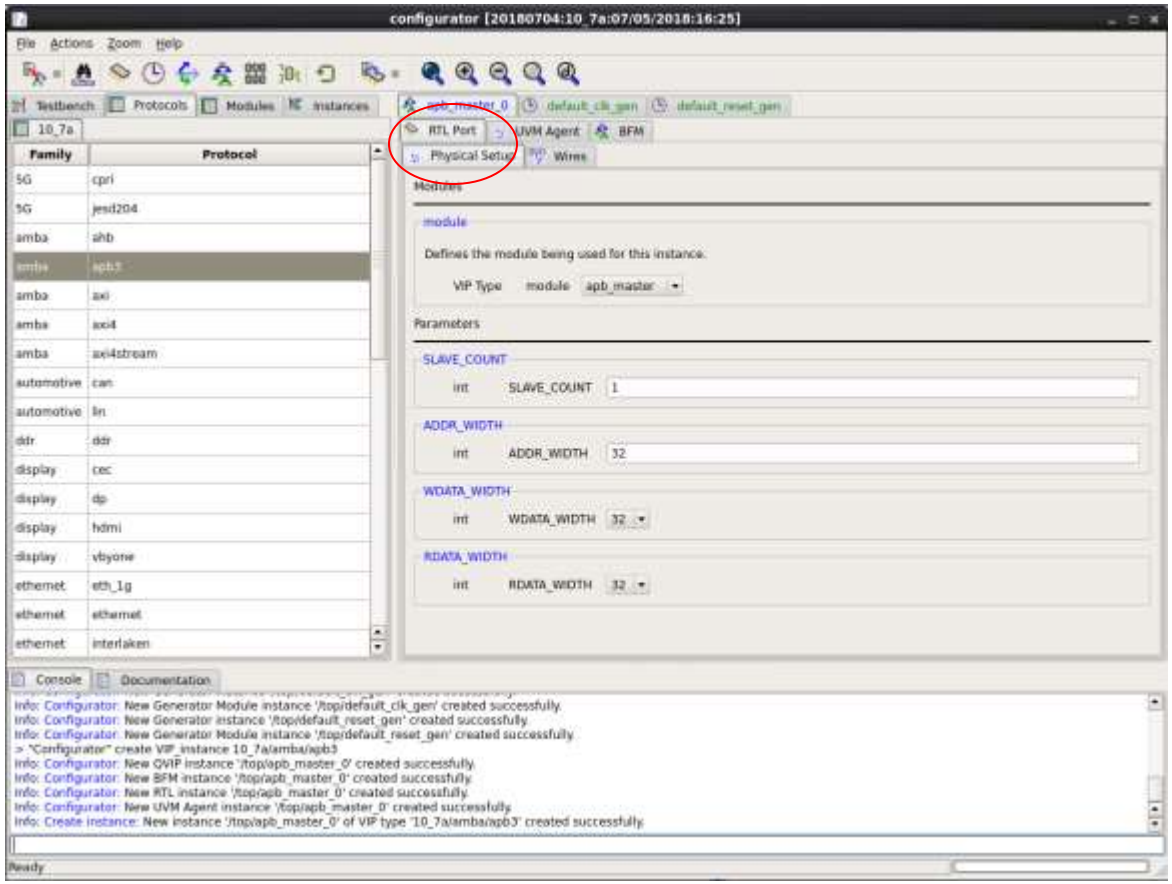
QVIP Configurator
Protocols Tab

- Use the QVIP Configurator to create a separate UVM environment for all of your QVIP agents.
  - Launch qvip_configurator from the directory of your choice and once open select the protocols tab and then double click the protocol you wish to use. (you may select more than one protocol but we will walk through only configuring one protocol for this example).



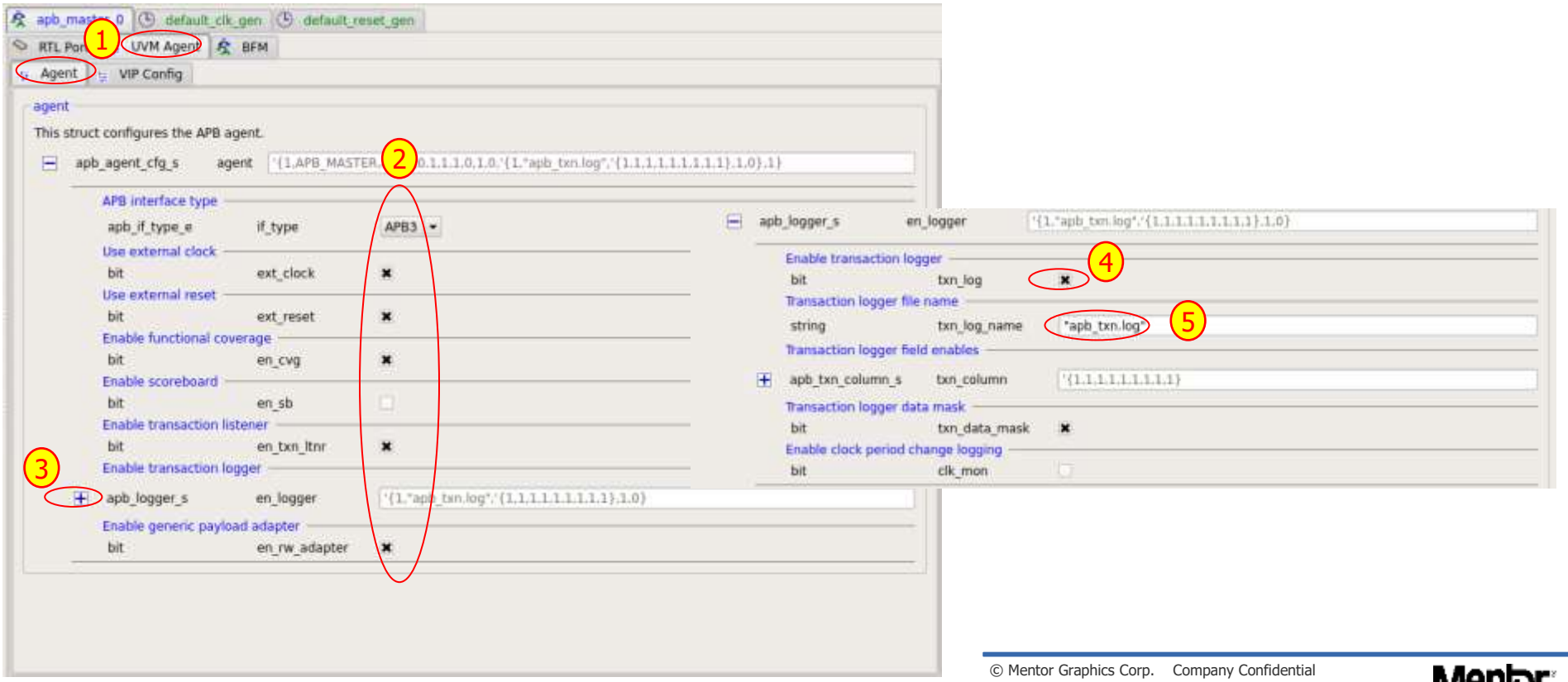
QVIP Configurator
RTL Port -> Physical Setup Tab

- Agent configuration
  - Select the Physical Setup tab and make changes as necessary. We will use all of the defaults in our example.



QVIP Configurator
UVM Agent -> Agent Tab

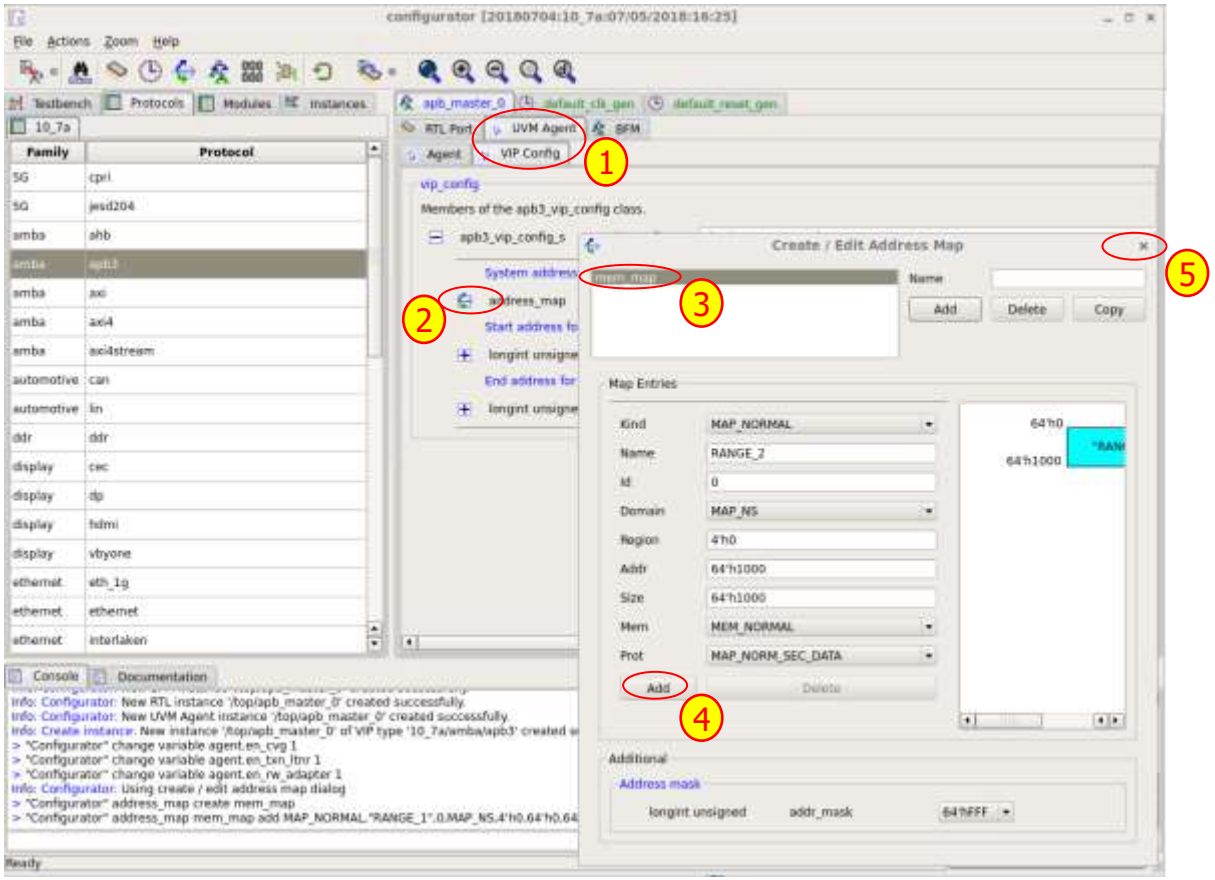
1. Click on the 'UVM Agent' then the 'Agent' tab.
2. Select settings as shown below.
3. Click the 'plus' symbol to the left of apb_logger_s.
4. Select txn_log.
5. Name your transaction log.



QVIP Configurator

UVM Agent -> VIP Config Tab

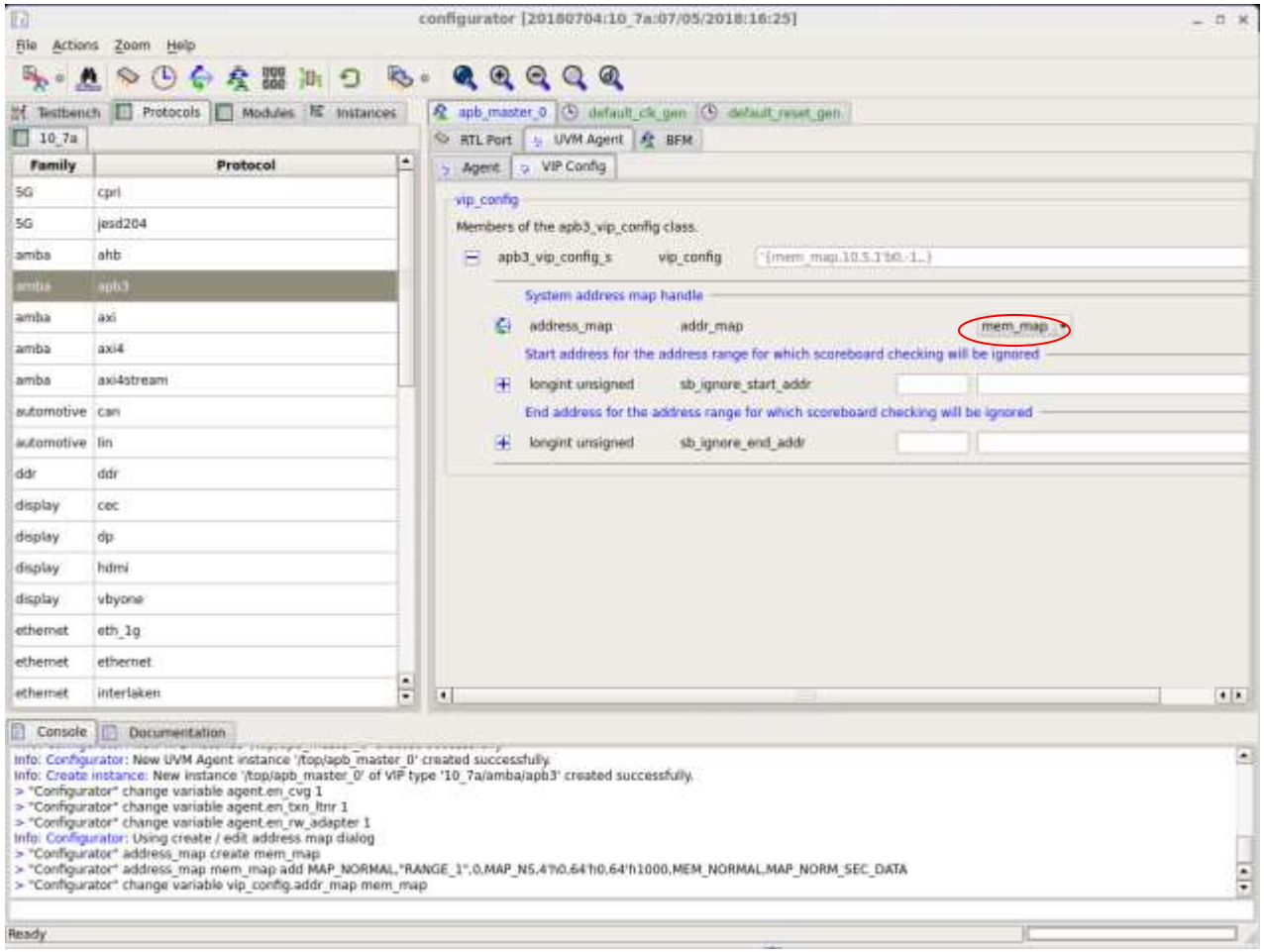
1. Click on the 'VIP Config' tab.
2. Click on the icon to the left of 'address_map'.
3. Name your new address map (i.e. mem_map).
4. Add your map entry.
5. Close the dialog box.



QVIP Configurator

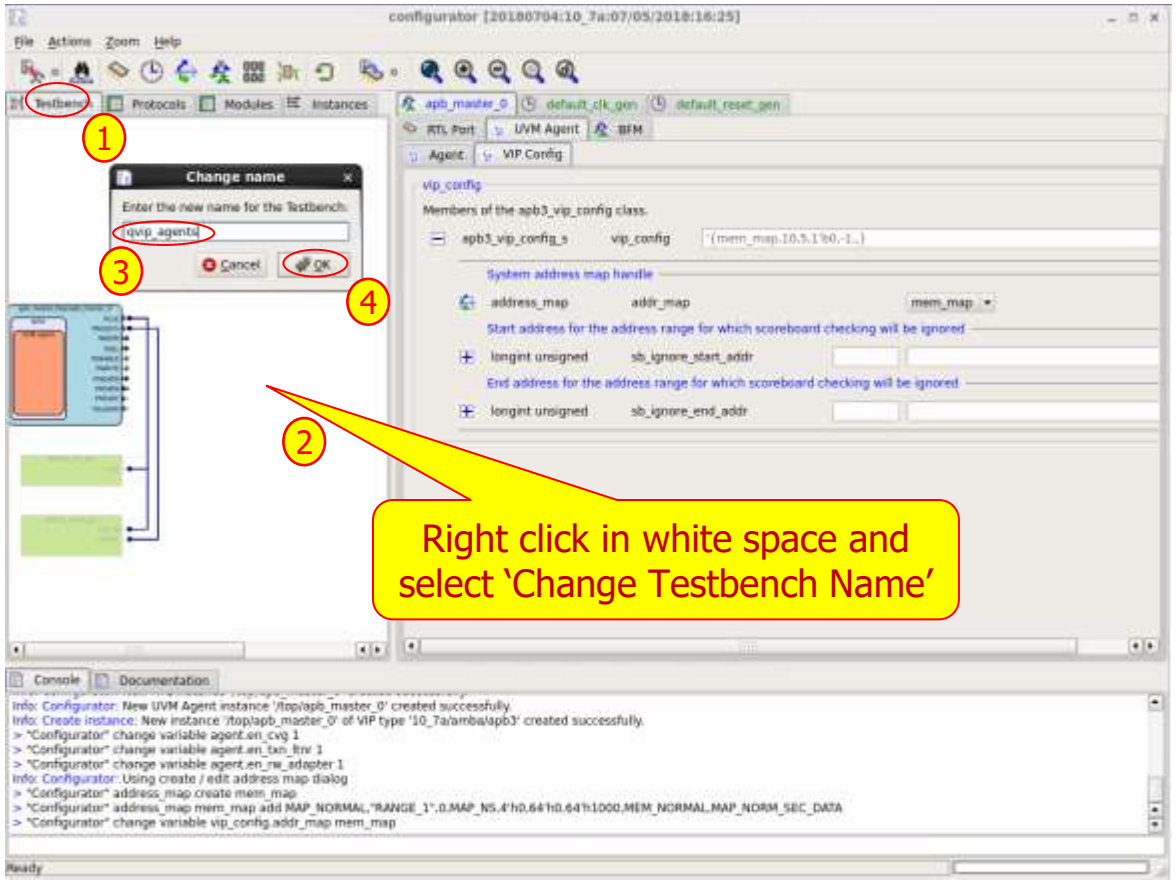
UVM Agent -> VIP Config Tab

- Use the pull down menu to select the ‘mem_map’ that you just created.



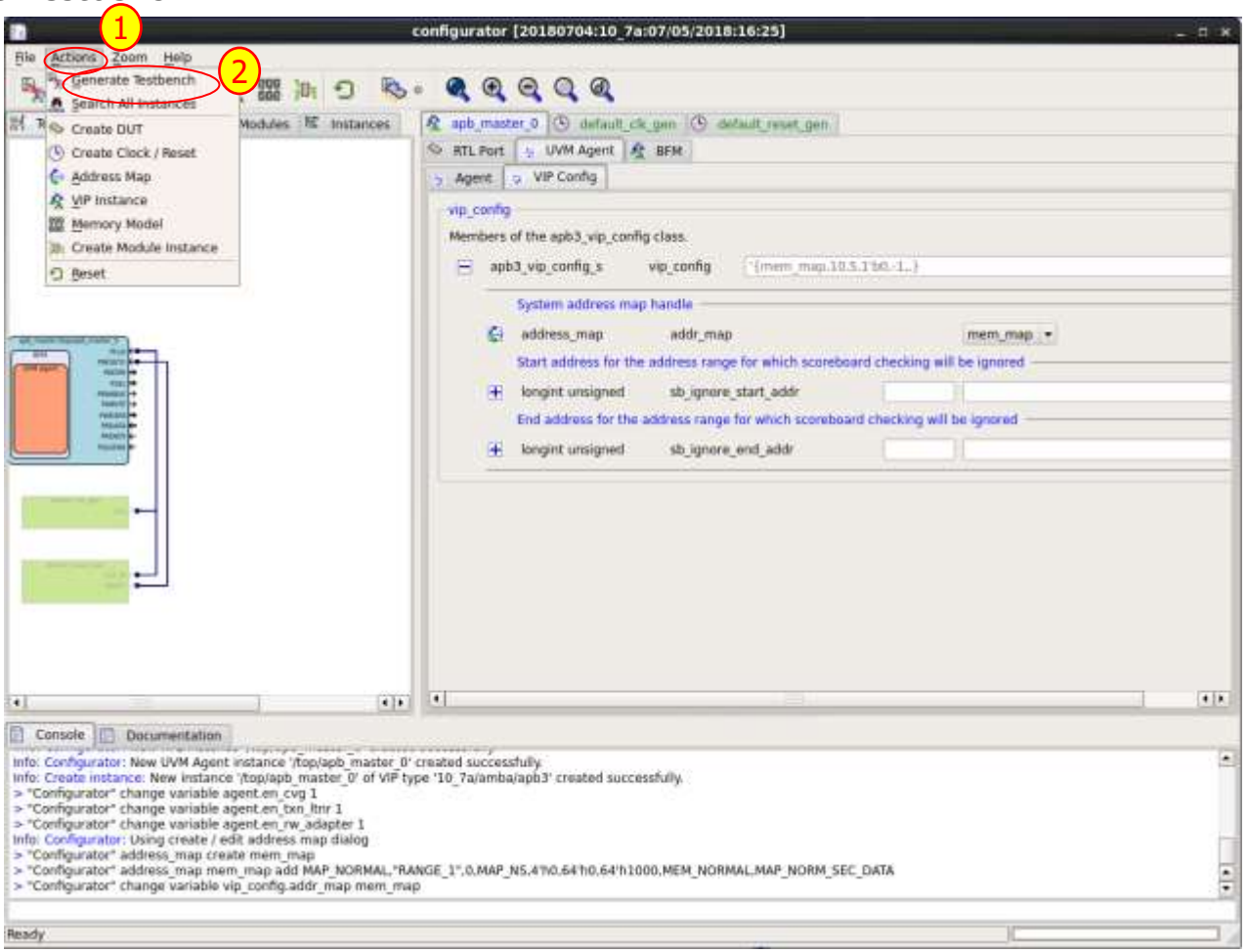
QVIP Configurator
UVM Agent -> VIP Config Tab

1. Click on the 'Testbench' tab
2. Right click in the white space of the Testbench diagram
3. Name your new QVIP testbench environment (i.e. qvip_agents)
4. Select OK



QVIP Configurator
Agent Tab

1. Select 'Actions'
2. Select 'Generate Testbench'



ALU Utility Components File
utility_components.yaml

- UVMF Utility Components
  - The utility components file will be used to generate predictor and coverage components for use in the environment.
- Lets look at the utility components file ***(utility_components.yaml)*** in more detail

Utility Comonents File

utility_components.yaml

- Define the component name and type
- Define the analysis exports and analysis ports to the component
- Add any parameters needed for the transaction types

```
uvmf:
  util_components:
    "alu_predictor" :
      type: "predictor"
      analysis_exports :
        - name: "alu_in_agent_ae"
          type: "alu_in_transaction #(.ALU_IN_OP_WIDTH(ALU_IN_OP_WIDTH))"
      analysis_ports :
        - name: "alu_sb_ap"
          type: "alu_out_transaction #(.ALU_OUT_RESULT_WIDTH(ALU_OUT_RESULT_WIDTH))"
      parameters:
        - name: "ALU_IN_OP_WIDTH"
          type: "int"
          value: "8"
        - name: "ALU_OUT_RESULT_WIDTH"
          type: "int"
          value: "16"
```

ALU Environment Config File
alu_env_cfg.yaml

- UVMF Environment Packages
  - The environment config file will be used to generate an environment package
  - The environment package will comprise the environment class, the environment configuration class and an environment sequence class. It can also optionally include predictor classes if they have been specified.
  - The environment package can be reused when a block level UVMF testbench is being used as part of a subsystem/chip level testbench.
- Lets look at environment config file ***(alu_env_cfg.yaml)*** in more detail

Environment Config File
alu_env_cfg.yaml

- Environment config defines the name of the environment package

Generate an environment with name 'alu_env_pkg'.
NOTE: the '_env_pkg' string is appended to the 'alu' environment name defined on line 3 of the YAML.

- Environment config file defines the number of instances of each agent to be used
- alu_env instantiates 2 agents
  - 1 x alu_in
  - 1 x alu_out

```
1 uvmf:
2   environments:
3     "alu" :
4       imports:
5         - name: "mvc_pkg"
6         - name: "mgc_apb3_v1_0_pkg"
```

Environment Config File
alu_env_cfg.yaml

- **parameters:**
  - Specify parameters for use within this environment package

```
 7        parameters:
 8          - name: "ALU_IN_OP_WIDTH"
 9            type: "int"
10            value: "8"
11          - name: "ALU_OUT_RESULT_WIDTH"
12            type: "int"
13            value: "16"
14          - name: "APB_ADDR_WIDTH"
15            type: "int"
16            value: "32"
17          - name: "APB_WDATA_WIDTH"
18            type: "int"
19            value: "32"
20          - name: "APB_RDATA_WIDTH"
21            type: "int"
22            value: "32"
```

- **imports:**
  - Import packages needed for QVIP sub environments.

```
 4        imports:
 5          - name: "mvc_pkg"
 6          - name: "mgc_apb3_v1_0_pkg"
```

Environment Config File
alu_env_cfg.yaml

ALU environment also instantiates agents, predictor components, and scoreboards

- **agents:**
  - Specify agents along with their parameters.

```
23        ## Agents are defined in a LIST so that the order is maintained. This is important
24        ## because of how the BFMs are passed in at the bench utilize this same order when
25        ## this environment's initialize() routine is called.
26        agents :
27          - name: "alu_in_agent"
28            type: "alu_in"
29            parameters:
30              - {name: "ALU_IN_OP_WIDTH", value: "ALU_IN_OP_WIDTH"}
31          - name: "alu_out_agent"
32            type: "alu_out"
33            parameters:
34              - {name: "ALU_OUT_RESULT_WIDTH", value: "ALU_OUT_RESULT_WIDTH"}
```

Include 1 x alu_in agent and 1 x alu_out agent. Defines agent instance names and sets the agent's parameters.

Environment Config File
alu_env_cfg.yaml

ALU environment also instantiates agents, predictor components, and scoreboards

- **analysis_components:**
  - Specify predictor.
- **scoreboards:**
  - Specify scoreboard.

```
38        ## Analysis components are defined externally in a 'util_components' structure.
39        ## They are instantiated here.
40        analysis_components :
41          - name: "alu_pred"
42            type: "alu_predictor"
43        ## Each scoreboard is keyed by the scoreboard instantiation name with
44        ## information on the scoreboard type and what type of transaction it
45        ## will be parsing
46        scoreboards :
47          - name: "alu_sb"
48            sb_type: "uvmf_in_order_scoreboard"
49            trans_type: "alu_out_transaction #(.ALU_OUT_RESULT_WIDTH(ALU_OUT_RESULT_WIDTH))"
```

Environment Config File
alu_env_cfg.yaml

Define connections between ALU agents and analysis components.

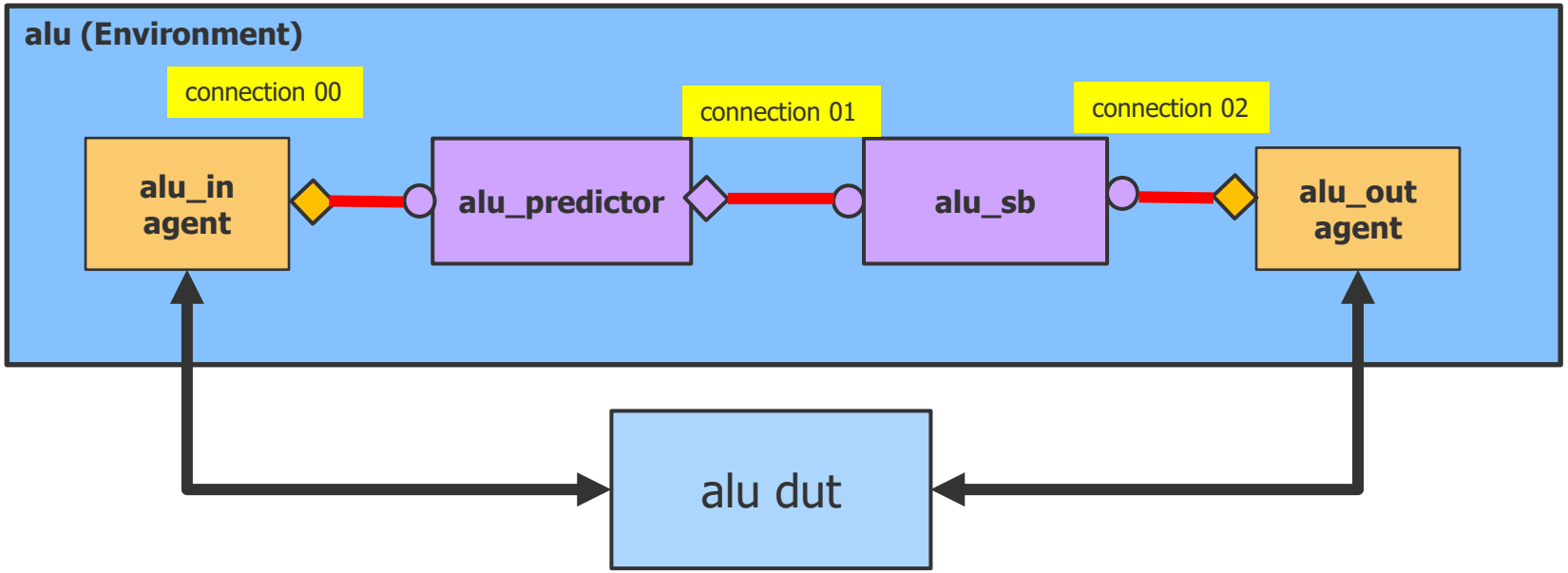
- **tlm_connections:**
  - Specify connections between agents and analysis components.

```
50      tlm_connections :
51        - driver:   "alu_in_agent.monitored_ap"
52          receiver: "alu_pred.alu_in_agent_ae"
53        - driver:   "alu_pred.alu_sb_ap"
54          receiver: "alu_sb.expected_analysis_export"
55        - driver:   "alu_out_agent.monitored_ap"
56          receiver: "alu_sb.actual_analysis_export"
```

connection 00

connection 01

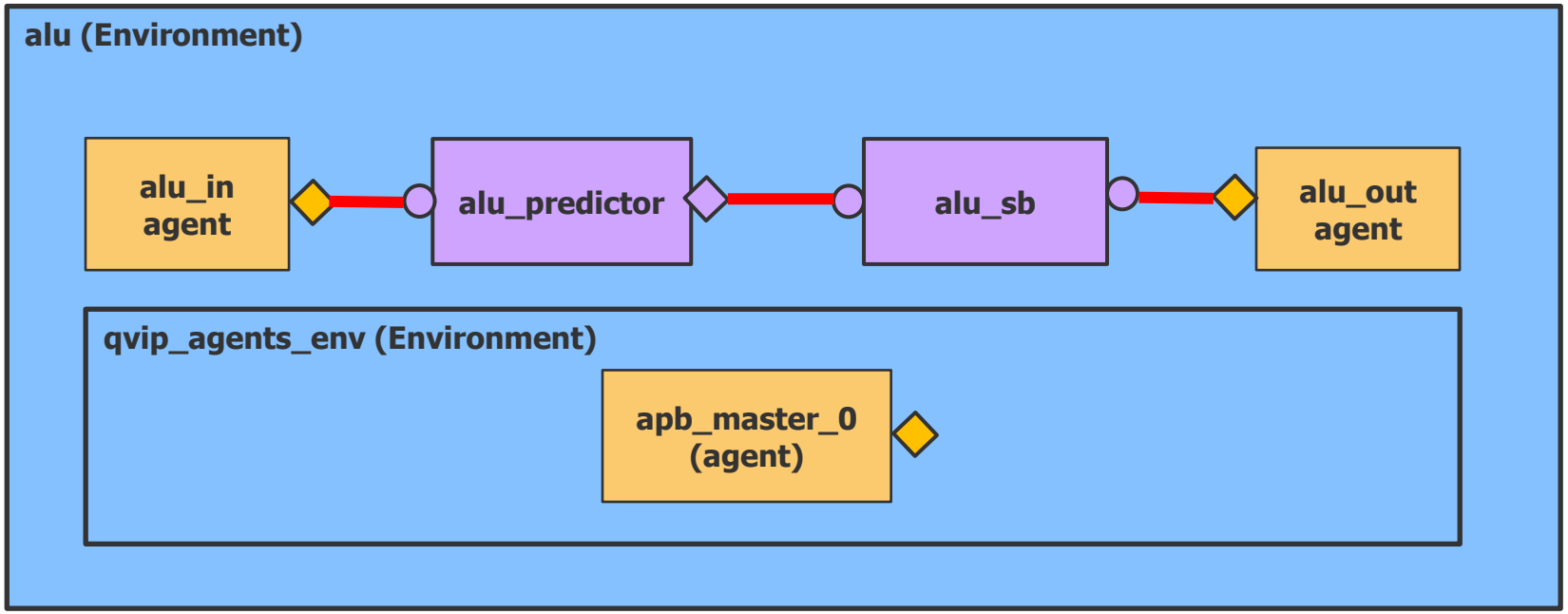
connection 02



Environment Config File
alu_env_cfg.yaml

- **qvip_subenvs:**
  - Instantiate sub environments created with qvip_configurator.
  - Be sure to include QVIP parameters in the parameters: directive shown above.

```
35        qvip_subenvs:
36          - name: "qvip_agents_env"
37            type: "qvip_agents"
```



Environment Config File
alu_env_cfg.yaml

ALU environment also instantiates agents, predictor components, and scoreboards

- **register_model:**
  - Specify predictor.

```
57        register_model :
58          use_adapter: "True"
59          use_explicit_prediction: "True"
60          maps:
61            - { name: "apb_map", interface: "alu_in_agent" }
```

Generated UVMF Code
alu_env_pkg

- **Generating the Environment and Test Bench Code**
  - Copy ./qvip_agents_dir/uvmf/qvip_agents_subenv_config.yaml to your current working directory where the rest of your YAML files reside.
  - Run yaml2uvmf.py on the alu_bench_cfg.yaml, alu_env_cfg.yaml, utility_components.yaml, and the qvip_agents_subenv_config.yaml files

$UVMF_HOME/scripts/yaml2uvmf.py alu_bench_cfg.yaml alu_env_cfg.yaml utility_components.yaml qvip_agents_subenv_config.yaml

  - After the code is generated cd to the ./qvip_agents_dir/uvmf and type the following on the Linux command line:

setenv QVIP_AGENTS_DIR_NAME `pwd

  - This allows the UVMF to find the QVIP subenvironment.

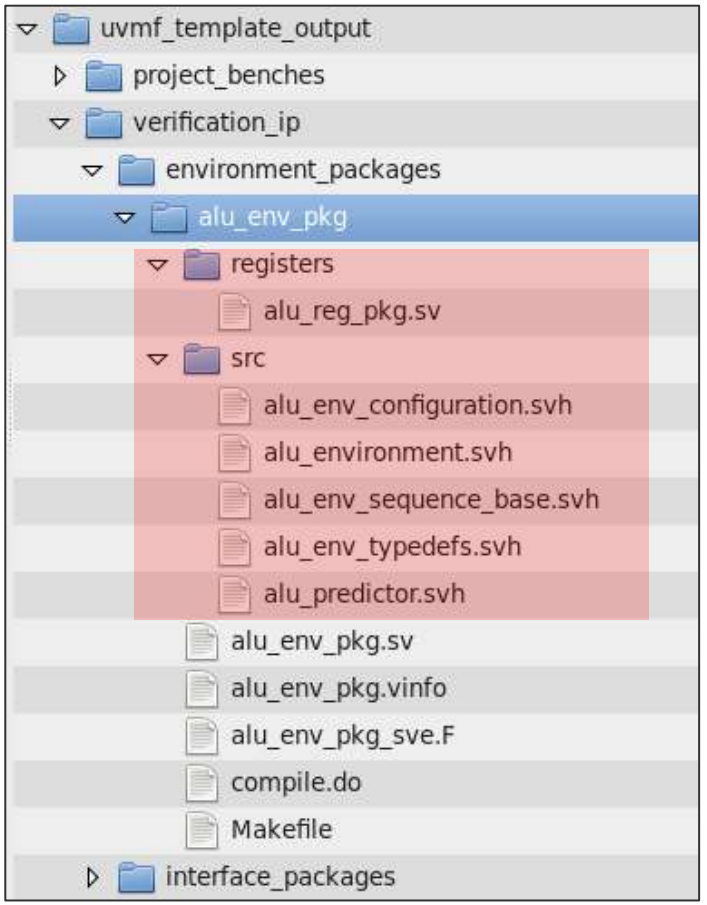
Generated UVMF Code
alu_env_pkg

- Files get generated under verification_ip/environment_packages/alu_env_pkg

- Makefile
  - Compiles the environment package
- alu_env_pkg.sv
  - ALU Environment package
- alu_reg_pkg.sv
  - Register package with a register model place holder

  User will need to add their own register model
- alu_env_configuration.svh
  - Configuration class for ALU environment
- alu_environment.svh
  - ALU environment class that instantiates the agents, scoreboards, predictors & connects them
- alu_env_sequence_base.svh
  - Base sequence class for any environment level sequences
- alu_predictor.svh
  - Generated predictor class for ALU environment.

  User will need to add code to model the function to predict



ALU Bench Config File
alu_bench_cfg.yaml

- UVMF Top Level Testbench
  - The bench config file will be used to generate the UVMF top-level testbench
  - The top level testbench will instantiate the ALU environment (which in turn instantiates the ALU interface agents as well as the environment configuration class
  - It facilitates the top-down configuration of the environment, which in turn configures the agents.
  - It provides a default sequence and a default test to run
  - It provides a simulation directory and makefile/run.do file for compiling and simulating the generated code
  - The code generated from the bench level config file is specific to the DUT it is testing and in general will be non-reusable code.
- Lets look at the bench config file ***(alu_bench_cfg.yaml)*** in more detail

ALU Bench Config File
alu_bench_cfg.yaml

- Short config file
- List of BFMs **MUST** be in same order as their corresponding agents were defined in the environment config file

```
uvmf:
  benches:
    "alu" :
      top_env: "alu"
      clock_half_period: "5ns"
      clock_phase_offset: "9ns"
      reset_assertion_level: "False"
      reset_duration: "200ns"
      imports:
        - name: "mgc_apb3_v1_0_pkg"
      parameters:
        - name: "TEST_ALU_IN_OP_WIDTH"
          type: "int"
          value: "8"
        - name: "TEST_ALU_OUT_RESULT_WIDTH"
          type: "int"
          value: "16"
        - name: "TEST_APB_ADDR_WIDTH"
          type: "int"
          value: "32"
        - name: "TEST_APB_WDATA_WIDTH"
          type: "int"
          value: "32"
        - name: "TEST_APB_RDATA_WIDTH"
          type: "int"
          value: "32"
      top_env_params:
        - name: "ALU_IN_OP_WIDTH"
          value: "TEST_ALU_IN_OP_WIDTH"
        - name: "ALU_OUT_RESULT_WIDTH"
          value: "TEST_ALU_OUT_RESULT_WIDTH"
        - name: "APB_ADDR_WIDTH"
          value: "TEST_APB_ADDR_WIDTH"
        - name: "APB_WDATA_WIDTH"
          value: "TEST_APB_WDATA_WIDTH"
        - name: "APB_RDATA_WIDTH"
          value: "TEST_APB_RDATA_WIDTH"
      interface_params:
        - bfm_name: "alu_in_agent"
          value:
            - {name: "ALU_IN_OP_WIDTH", value: "TEST_ALU_IN_OP_WIDTH"}
        - bfm_name: "alu_out_agent"
          value:
            - {name: "ALU_OUT_RESULT_WIDTH", value: "TEST_ALU_OUT_RESULT_WIDTH"}
      active_passive:
        - {bfm_name: "alu_in_agent", value: "ACTIVE"}
        - {bfm_name: "alu_out_agent", value: "PASSIVE"}
```

Generate a bench with name 'alu'

Instantiate 'alu' environment previously defined in alu_env_cfg.yaml

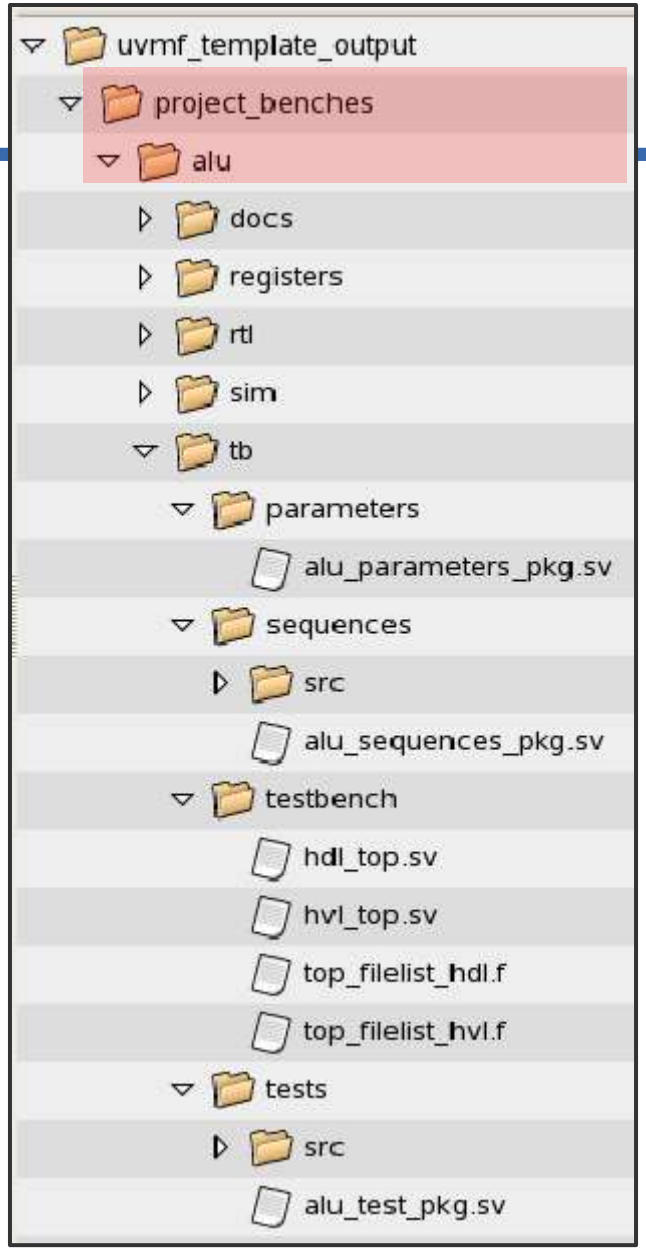
Include the specified BFMs & defines if drivers are ACTIVE or PASSIVE

Generated UVMF Code
project_benches/alu

- Generates the top level UVMF testbench plus scripts for compiling and running the simulation

- Files generated under

  project_benches/alu



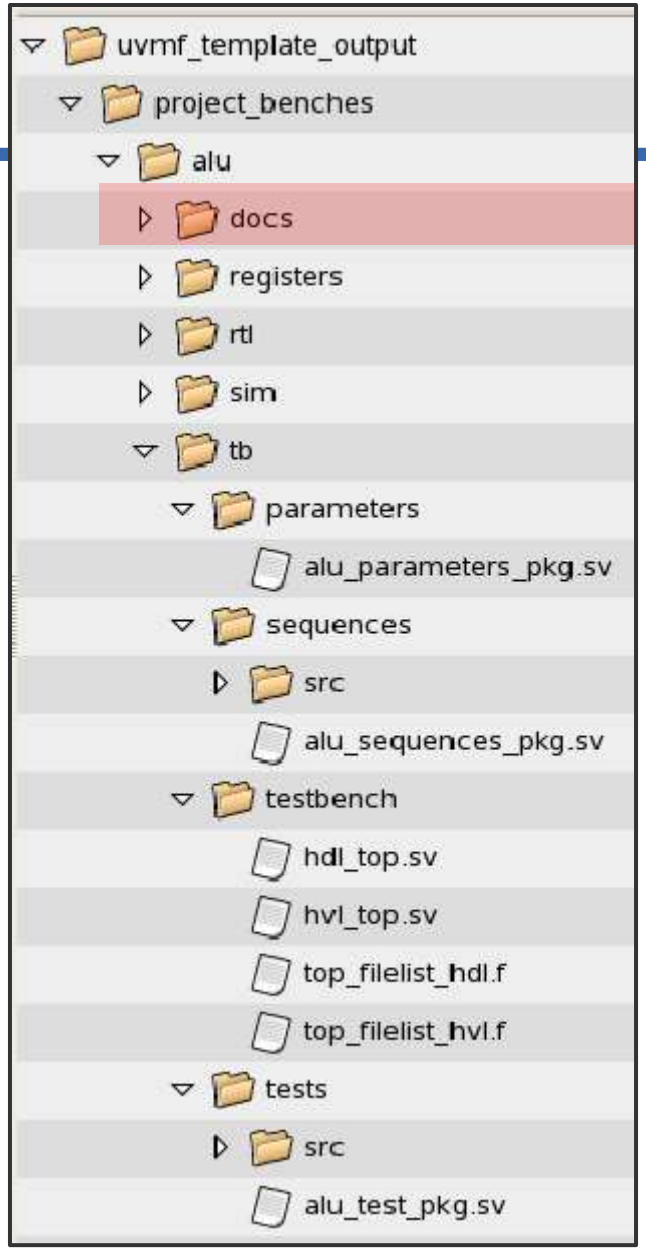
Generated UVMF Code
project_benches/alu

- Files generated under

  project_benches/alu

  - **docs**

    — Placeholder folder for user to place documentation



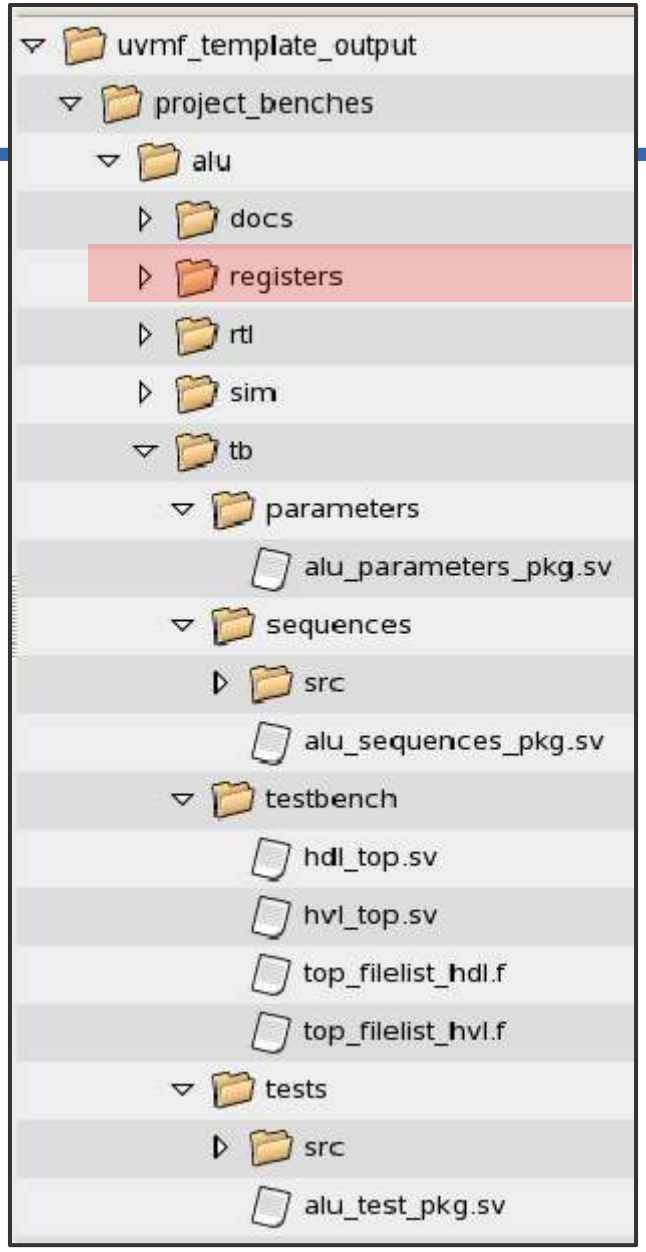
Generated UVMF Code
project_benches/alu

- Files generated under

  project_benches/alu

  - **registers**

    — Placeholder folder for user to place register layer package



Generated UVMF Code
project_benches/alu

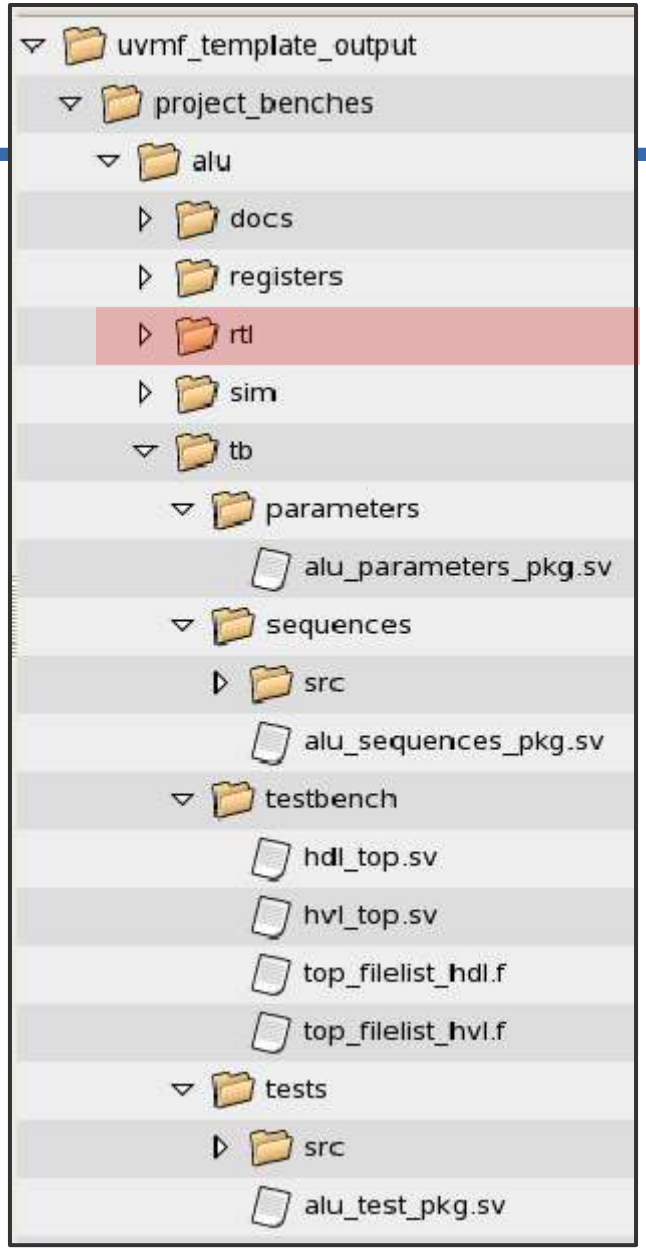
- Files generated under

  project_benches/alu

  - **rtl**

    — Placeholder folder for user to place RTL DUT code

    — Optional can place your DUT code any where you want.



Generated UVMF Code
project_benches/alu

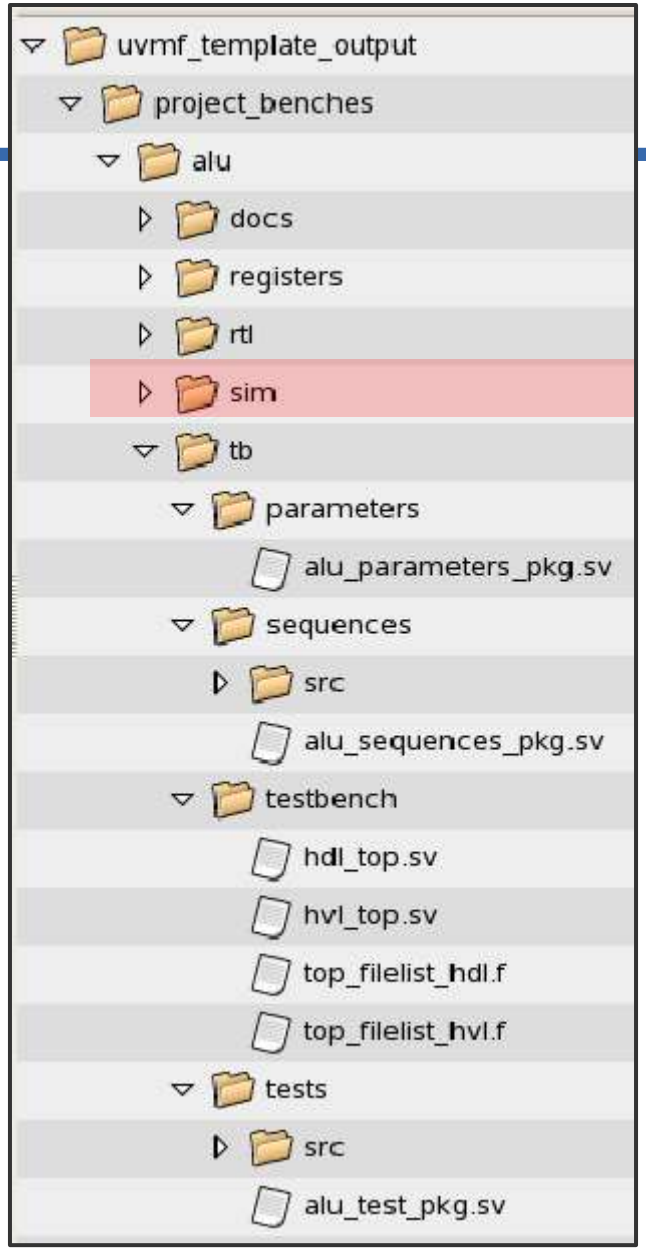
- Files generated under

  project_benches/alu

  - **sim**

    — Directory where user should run simulations.
    — Contains makefile for Linux users to compile & run testbench
    — Contains run.do for Windows users to compile & run testbench
    — Contains wave.do which is populated with agent transactions

    — Also contain some other support files for emulation users plus a default RMDB for Questa VRM users.



Generated UVMF Code
project_benches/alu

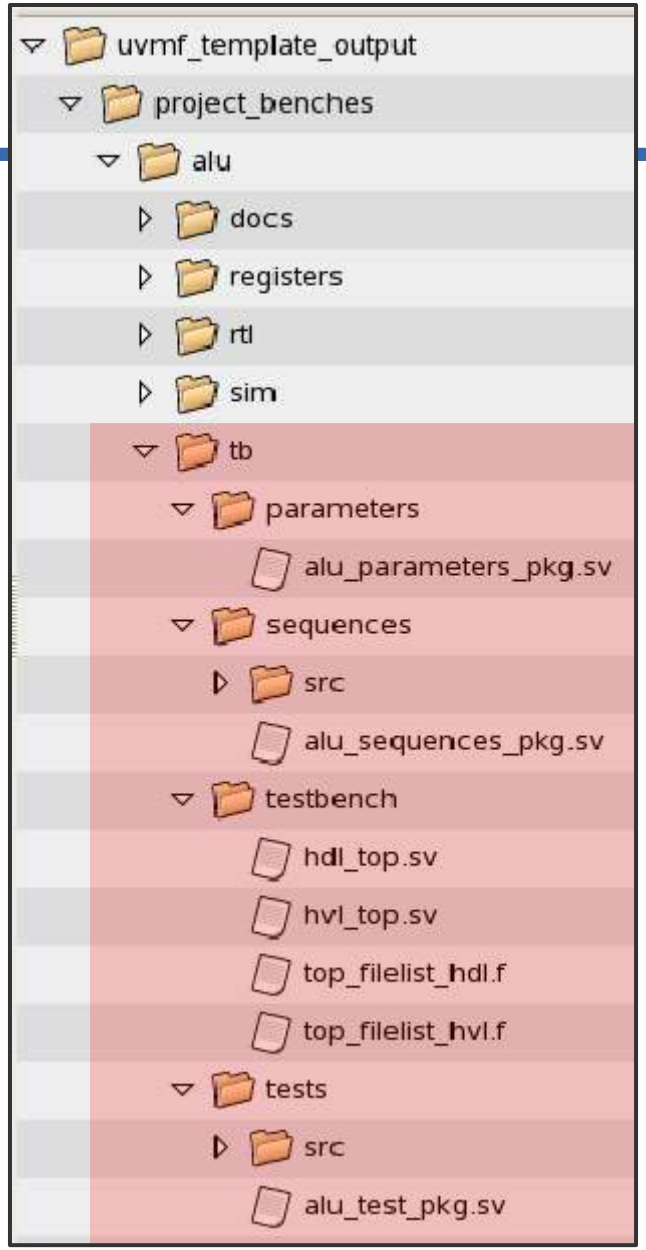
- Files generated under

  project_benches/alu

  - **tb**

    — Multiple sub-folders for testbench

  - **Parameters**
    - — Top level testbench params package (i/f names, etc)
  - **Sequences**
    - — Example top level sequence and a sequence base class
    - — Sequence package
  - **testbench**
    - — hdl_top.sv : top level module based TB
    - — hvl_top.sv : non-synthesizable parts of top level TB
  - **tests**
    - — Example top level test, extended from test base class
    - — Test Package

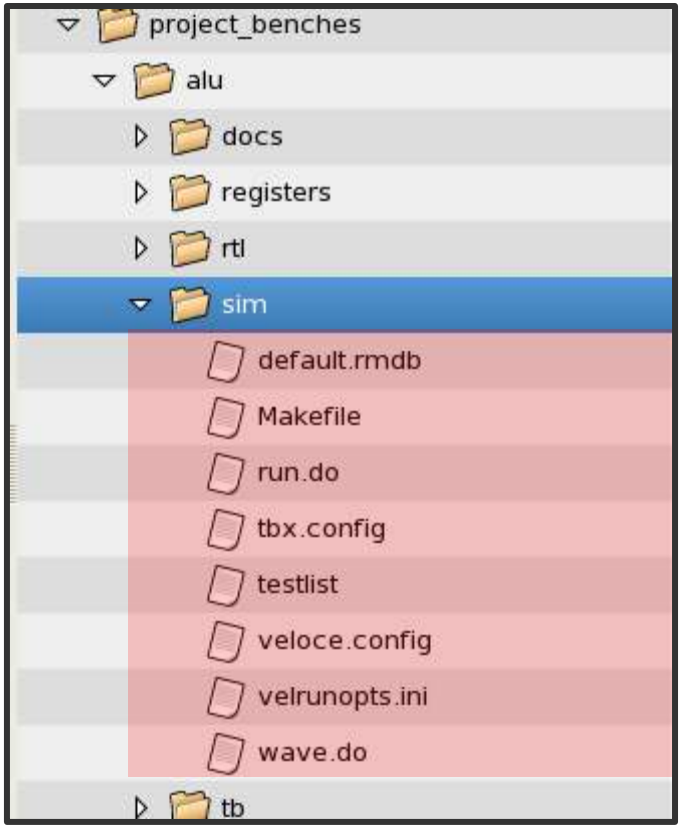


Agenda

- ALU Overview
- Config Files Explained
- Compile and Simulate Generated Code
- Adding DUT Specific Functionality
- Generate and integrate the Register Model
- Add functional coverage

Out Of The Box Simulation
project_benches/alu/sim

- Bench level config file
  - Generates Makefile for Linux users
  - Generates run.do for Windows users
  - Other files used in simulation
    - wave.do : contains signals & transactions from each of the agents
  - Remaining files can be ignored for regular simulation
    - default.rmdb : Questa VRM regression file
    - testlist : Questa VRM testlist
    - tbx.config : Veloce emulation file
    - veloce.config : Veloce emulation file
    - velrunopts.ini : Veloce emulation file



Out Of The Box Simulation
project_benches/alu/sim

- OS Considerations

  - Linux
    - make build : compiles the generated code
    - make debug : compiles and loads the generated code

  - Windows
    - do run.do : compiles and loads the generated code

  - Makefile sets default OS architecture to 32 bit
    - If running on the 64 bit version of Questa on Linux, you can change the OS setting to 64 bit as follows;

```
make build MACHINE_ARCH='-64'
make debug MACHINE_ARCH='-64'
```

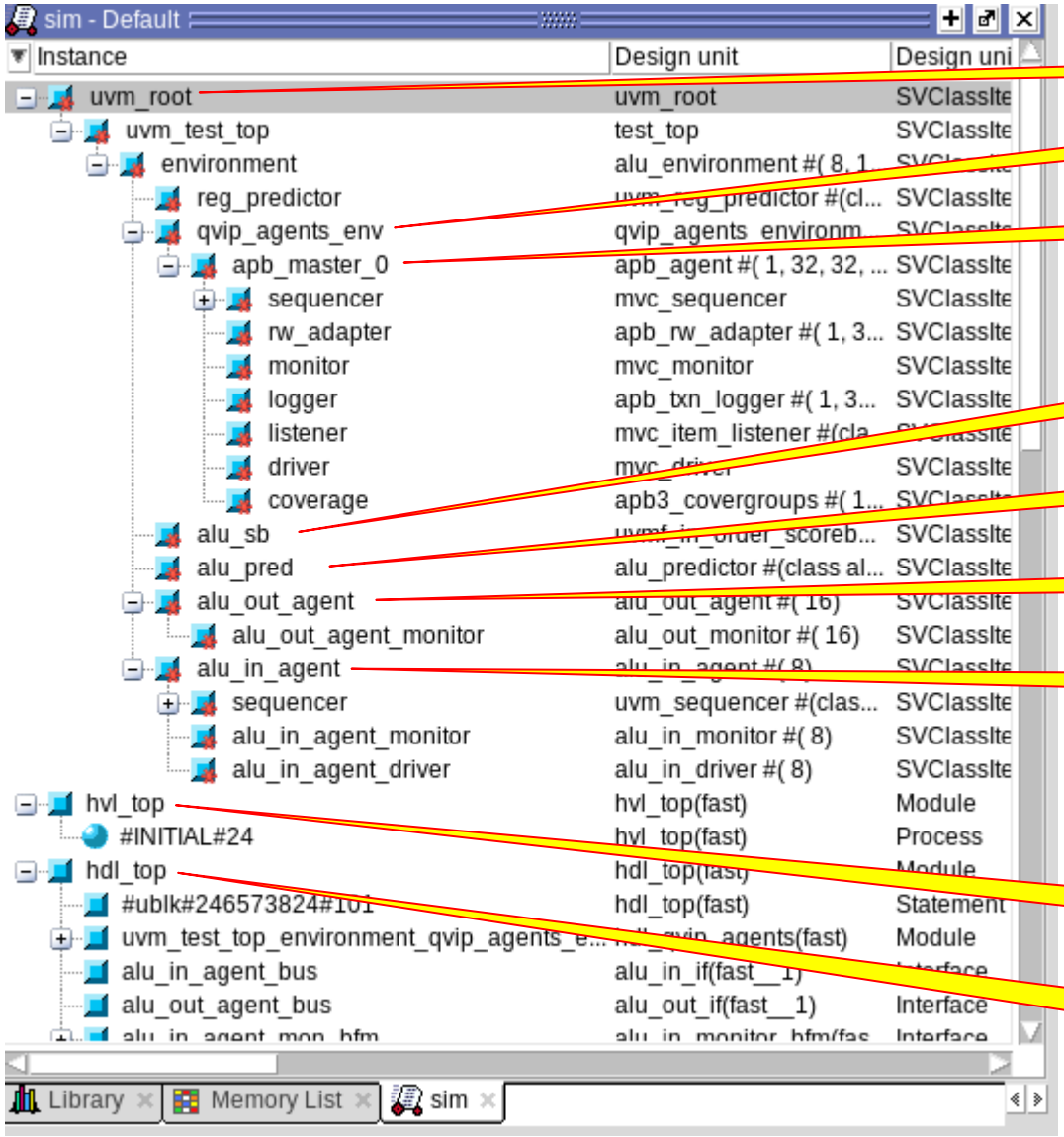
Out Of The Box Simulation
project_benches/alu/sim

- Bench level config file
  - Generated Makefile / run.do invoke vsim with +UVM_TESTNAME=test_top
  - Also applies some other switches to vsim that are required to run the UVMF simulation. ***Do not remove them***

```
vsim -i  -32  -sv_seed random +UVM_TESTNAME=test_top \
              +UVM_VERBOSITY=UVM_HIGH  \
              -permit_unmatched_virtual_intf +notimingchecks \
              -suppress 8887 -uvmcontrol=all -msgmode both \
              -classdebug –assertdebug \
              +uvm_set_config_int=*,enable_transaction_viewing,1 \
              -do " set NoQuitOnFinish 1; onbreak {resume}; run 0; \
                    do wave.do; set PrefSource(OpenOnBreak) 0; \
                    radix hex showbase; " optimized_debug_top_tb
```

Out Of The Box Simulation

project_benches/alu/sim



UVM top level

QVIP subenv

QVIP agent

scoreboard

predictor

alu_out agent

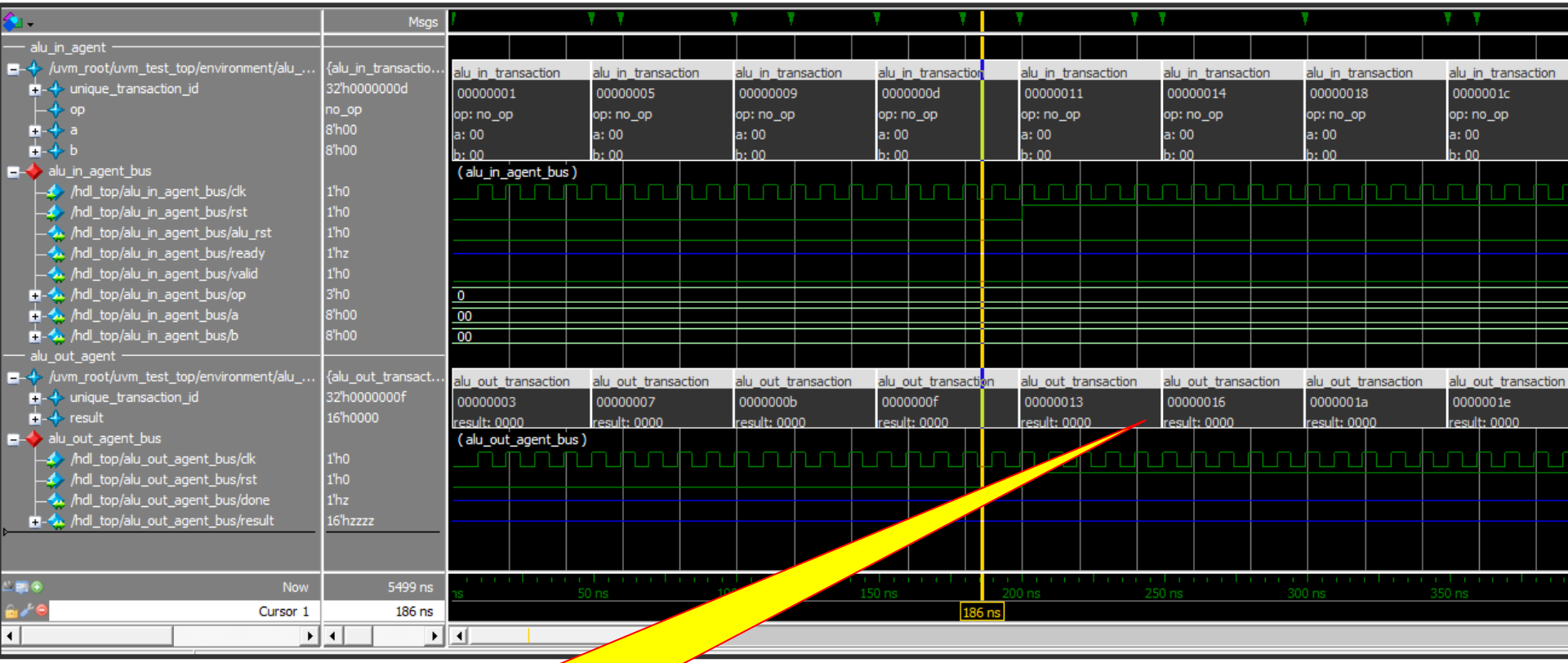
alu_in agent

Non - Synthesizable Module based top level

Synthesizable Module based top level

Out Of The Box Simulation

project_benches/alu/sim



Transactions coming from the monitors
No DUT connected yet….
Just showing default values

Out Of The Box Simulation

project_benches/alu/tb/tests/src

- Default test : top_test.svh
  - Extends from uvmf_test_base
  - Is parameterized with the configuration, environment and sequence to use
  - Build phase kicks off top down configuration

```
23  typedef alu_env_configuration alu_env_configuration_t;
24  typedef alu_environment alu_environment_t;
25
26  class test_top extends uvmf_test_base #(.CONFIG_T(alu_env_configuration_t),
27                                          .ENV_T(alu_environment_t),
28                                          .TOP_LEVEL_SEQ_T(alu_bench_sequence_base));
29    `uvm_component_utils( test_top );
30
31  // ****************************************************************************
36    function new( string name = "", uvm_component parent = null );
37      super.new( name ,parent );
38    endfunction
39
40  // ****************************************************************************
51    virtual function void build_phase(uvm_phase phase);
52
53      super.build_phase(phase);
54      configuration.initialize(BLOCK, "uvm_test_top.environment", alu_parameters_pkg::interface_names, null,
        alu_parameters_pkg::interface_activities);
55    endfunction
56
57  endclass
```

This is the default top level virtual sequence that gets executed

Out Of The Box Simulation

project_benches/alu/tb/sequences/src

- Default sequence : alu_bench_sequence_base.svh

```
69  virtual task body();
70
71  // Construct sequences here
72   alu_in_agent_random_seq     = alu_in_agent_random_seq_t::type_id::create("alu_in_agent_random_seq");
73
74  // Start RESPONDER sequences here
75   fork
76   join_none
77
78  // Start INITIATOR sequences here
79   fork
80       repeat (25) alu_in_agent_random_seq.start(alu_in_agent_sequencer);
81   join
82
83   // UVMF_CHANGE_ME : Extend the simulation XXX number of clocks after
84   // the last sequence to allow for the last sequence item to flow
85   // through the design.
86
87  fork
88    alu_in_agent_config.wait_for_num_clocks(400);
89    alu_out_agent_config.wait_for_num_clocks(400);
90  join
91
92  endtask
```

Out Of The Box Simulation
project_benches/alu/tb/sequences/src

- Default sequence : alu_bench_sequence_base.svh
  - Sequence body (shown on previous slide)
    - creates agent sequences
      - The alu_out agent is passive so it has no default sequence to run
    - Repeats each agent sequence to run 25 times
      - The default random sequence randomizes and generates 1 transaction.
    - Uses utility methods in agent configs to wait for specified number of clocks
  - Sequence code (not shown)
    - Extends from uvmf_sequence_base
    - Defines sequence handles to run on each active agent.
    - Gets the config handles for each agent from the UVM config_db
    - Gets the sequencer handles for each agent from the UVM config_db

Agenda

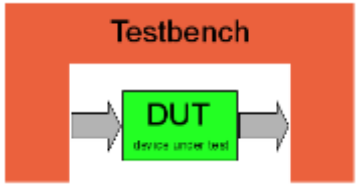
- ALU Overview
- Config Files Explained
- Compile and Simulate Generated Code
- Adding DUT Specific Functionality
- Generate and integrate the Register Model
- Add functional coverage

Completing the UVMF Testbench

- User Modifications To the UVMF Generated Code
  - Having generated the UVMF testbench structure using the Python config files, the user now has to modify certain files to add DUT specific functionality.
  - These modification steps include
    1. Adding the DUT & wiring it up to the BFMs and the clock/reset
    2. Adding protocol specific information to the driver BFMs
    3. Adding protocol specific information to the monitor BFMs
    4. Adding DUT specific behavior to the predictor
  - Then the user will need to create additional tests & sequences to exercise the DUT functionality, which requires the following steps
    1. Extending the default test to create a new test which overrides the default sequence
    2. Extending the default sequence to create a new sequence that generates the desired stimulus for the test.
- The following slides will look at the code changes required to implement each of the above steps

Instantiate & Wire Up the DUT

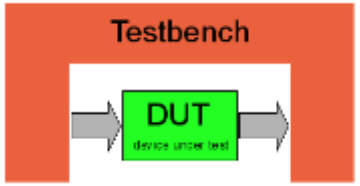
project_benches/alu/tb/testbench/hdl_top.sv



- Clocks & Resets
  - The hdl_top module contains simple clock and reset generation code that the user can modify to change frequencies, add more clocks, etc depending on the need for their specific DUT
  - In the case of the ALU IP we can leave this code unmodified.

```
33  module hdl_top;
34  // pragma attribute hdl_top partition_module_xrtl
35
36
37  bit rst = 0;
38  bit clk;
39     // Instantiate a clk driver
40     // tbx clkgen
41     initial begin
42        #9ns;
43        clk = ~clk;
44        forever #5ns clk = ~clk;
45     end
46     // Instantiate a rst driver
47     initial begin
48        #200ns;
49        rst <= ~rst;
50     end
```

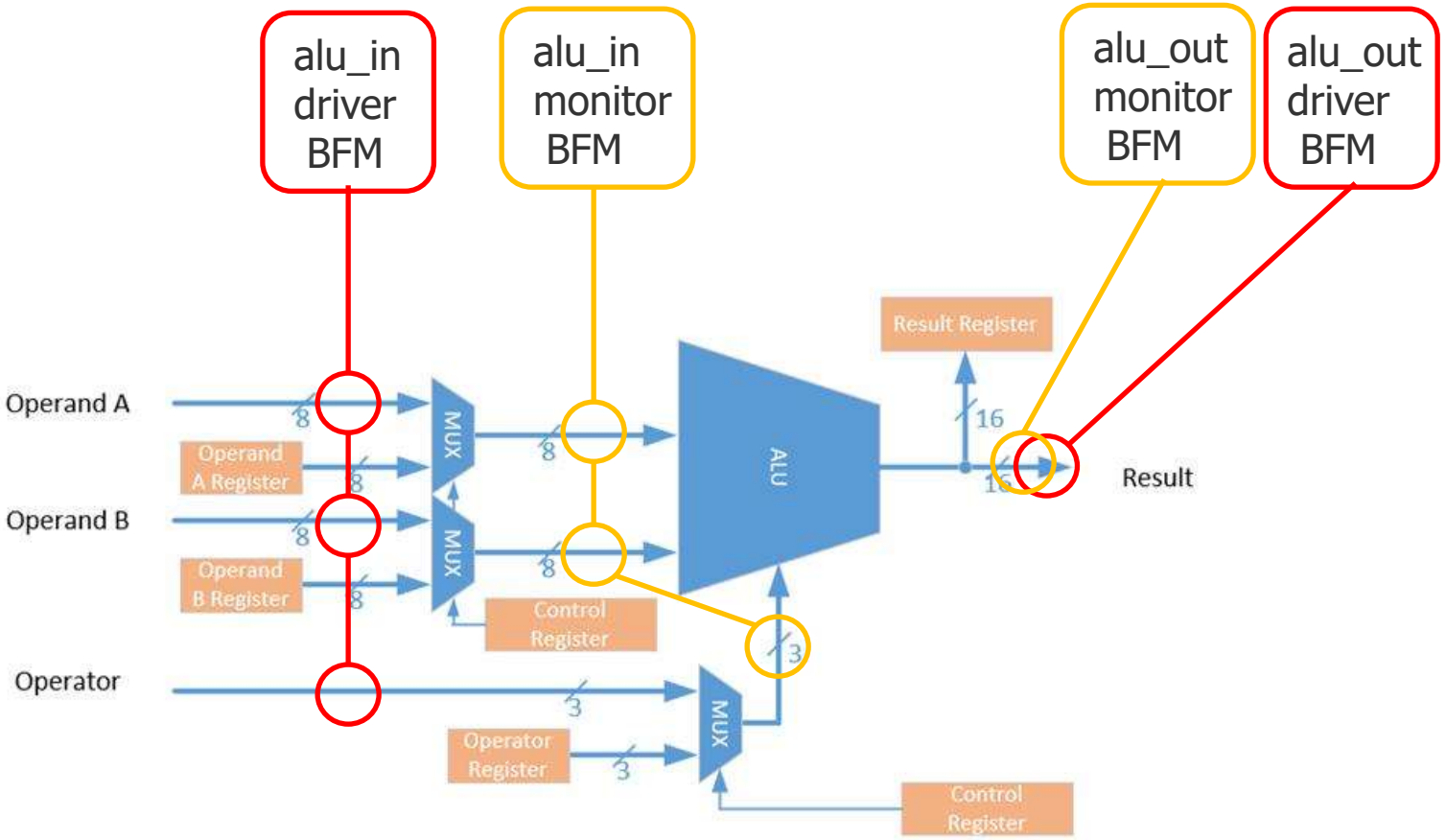
Instantiate & Wire Up the DUT
project_benches/alu/tb/testbench/hdl_top.sv



- The DUT
  - The DUT RTL model is located at ***project_benches/alu/rtl/verilog/alu.v*** [in the version shipped with the UVMF in the Questa install tree]
  - You can copy this file into the corresponding rtl folder under your ***uvmf_template_output/project_benches/alu/rtl/verilog*** which was created when you ran your YAML config files

    NOTE: You do not have to place the DUT RTL code in this this directory. This is merely a placeholder location for DUT source code but it can reside anywhere on disk.

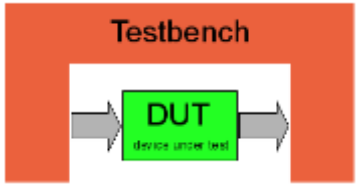
DUT-Test Bench Connections



ALU behavior checked using predictor
MUX behavior checked using SVA

Instantiate & Wire Up the DUT

project_benches/alu/tb/testbench/hdl_top.sv



- Instantiate The DUT
  - Edit the file ***project_benches/alu/tb/testbench/hdl_top.sv***
  - Find the comment that says "Instantiate DUT here"
  - Add instance of the ALU and wire up ports to the corresponding agent interface

Original Code

```
68  // UVMF_CHANGE_ME : Add DUT and connect
    to signals in _bus interfaces listed above
69  // Instantiate DUT here
70
71  initial begin  // tbx vif_binding_block
72      import uvm_pkg::uvm_config_db;
```

Modified Code

```
33 module hdl_top;
34   // pragma attribute hdl_top partition_module_xrtl
35   hdl_qvip_agents #(.UNIQUE_ID("uvm_test_top.environment.qvip_agents_env."),.EXT_CLK_RESET(1)
67   // UVMF_CHANGE_ME : Add DUT and connect to signals in _bus interfaces listed above
68   assign uvm_test_top_environment_qvip_agents_env_qvip_hdl.default_clk_gen_CLK     = clk;
69   assign uvm_test_top_environment_qvip_agents_env_qvip_hdl.default_reset_gen_RESET = rst;
70
71   // Instantiate your DUT here
72   alu #(.OP_WIDTH    (TEST_ALU_IN_OP_WIDTH),
73         .RESULT_WIDTH (TEST_ALU_OUT_RESULT_WIDTH)
74   )
75     DUT (.clk     ( alu_in_agent_bus.clk                                    ),
76          .rst     ( alu_in_agent_bus.alu_rst                                ),
77          .ready   ( alu_in_agent_bus.ready                                  ),
78          .valid   ( alu_in_agent_bus.valid                                  ),
79          .op      ( alu_in_agent_bus.op                                     ),
80          .a       ( alu_in_agent_bus.a                                      ),
81          .b       ( alu_in_agent_bus.b                                      ),
82          .done    ( alu_out_agent_bus.done                                  ),
83          .result  ( alu_out_agent_bus.result                                ),
84          .PADDR   ( uvm_test_top_environment_qvip_agents_env_qvip_hdl.apb_master_0_PADDR   ),
85          .PSEL    ( uvm_test_top_environment_qvip_agents_env_qvip_hdl.apb_master_0_PSEL    ),
86          .PENABLE ( uvm_test_top_environment_qvip_agents_env_qvip_hdl.apb_master_0_PENABLE ),
87          .PWRITE  ( uvm_test_top_environment_qvip_agents_env_qvip_hdl.apb_master_0_PWRITE  ),
88          .PWDATA  ( uvm_test_top_environment_qvip_agents_env_qvip_hdl.apb_master_0_PWDATA  ),
89          .PRDATA  ( uvm_test_top_environment_qvip_agents_env_qvip_hdl.apb_master_0_PRDATA  ),
90          .PREADY  ( uvm_test_top_environment_qvip_agents_env_qvip_hdl.apb_master_0_PREADY  ),
91          .PSLVERR ( uvm_test_top_environment_qvip_agents_env_qvip_hdl.apb_master_0_PSLVERR )
92   );
93
```

NOTES:

To get list of interface signals, then the interface files are located at :

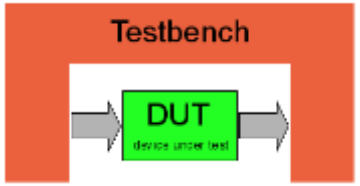
- uvmf_template_output/verification_ip/interfaces/alu_in_pkg/src/alu_in_if.sv
- uvmf_template_output/verification_ip/interfaces/alu_out_pkg/src/alu_out_if.sv

The testbench reset is passed in to the agent BFM interfaces but the alu rst pin needs to be driven from the alu_rst pin of the alu_in BFM. This is required since there in a RST_OP transaction that can be actioned where the driver will need to activate the reset signal to the DUT.

Need to connect the clk and rst signal from this testbench down into the clock and reset signals of the QVIP environment (see lines 68 and 69 in the 'Modified Code').

Move the alu_in_monitor_bfm interface connections

project_benches/alu/tb/testbench/hdl_top.sv



- The alu_in_monitor_bfm is normally connected directly to the alu_in_driver_bfm
  - We need to move the interface connects to the outputs of the ALU input muxes. Please refer to the block diagram on slide 3.

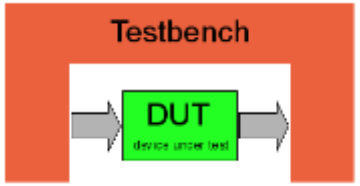
Original Code

```
65  // Instantiate the signal bundle, monitor bfm and driver bfm for each interface.
66  // The signal bundle, _if, contains signals to be connected to the DUT.
67  // The monitor, monitor_bfm, observes the bus, _if, and captures transactions.
68  // The driver, driver_bfm, drives transactions onto the bus, _if.
69  alu_in_if #(
70        .ALU_IN_OP_WIDTH(TEST_ALU_IN_OP_WIDTH)
71        ) alu_in_agent_bus(
72     // pragma uvmf custom alu_in_agent_bus_connections begin
73     .clk(clk), .rst(rst)
74     // pragma uvmf custom alu_in_agent_bus_connections end
75     );
76  alu_out_if #(
77        .ALU_OUT_RESULT_WIDTH(TEST_ALU_OUT_RESULT_WIDTH)
78        ) alu_out_agent_bus(
79     // pragma uvmf custom alu_out_agent_bus_connections begin
80     .clk(clk), .rst(rst)
81     // pragma uvmf custom alu_out_agent_bus_connections end
82     );
83  alu_in_monitor_bfm #(
84        .ALU_IN_OP_WIDTH(TEST_ALU_IN_OP_WIDTH)
85        ) alu_in_agent_mon_bfm(alu_in_agent_bus.monitor_port);
86  alu_out_monitor_bfm #(
87        .ALU_OUT_RESULT_WIDTH(TEST_ALU_OUT_RESULT_WIDTH)
88        ) alu_out_agent_mon_bfm(alu_out_agent_bus.monitor_port);
89  alu_in_driver_bfm #(
90        .ALU_IN_OP_WIDTH(TEST_ALU_IN_OP_WIDTH)
91        ) alu_in_agent_drv_bfm(alu_in_agent_bus.initiator_port);
```

Modified Code

```
65  // Instantiate the signal bundle, monitor bfm and driver bfm for each interface.
66  // The signal bundle, _if, contains signals to be connected to the DUT.
67  // The monitor, monitor_bfm, observes the bus, _if, and captures transactions.
68  // The driver, driver_bfm, drives transactions onto the bus, _if.
69  alu_in_if #(
70        .ALU_IN_OP_WIDTH(TEST_ALU_IN_OP_WIDTH)
71        ) alu_in_agent_bus(
72     // pragma uvmf custom alu_in_agent_bus_connections begin
73     .clk(clk), .rst(rst)
74     // pragma uvmf custom alu_in_agent_bus_connections end
75     );
76  alu_out_if #(
77        .ALU_OUT_RESULT_WIDTH(TEST_ALU_OUT_RESULT_WIDTH)
78        ) alu_out_agent_bus(
79     // pragma uvmf custom alu_out_agent_bus_connections begin
80     .clk(clk), .rst(rst)
81     // pragma uvmf custom alu_out_agent_bus_connections end
82     );
83  // pragma uvmf custom alu_in_monitor_bus_connections begin
84  alu_in_if #(.ALU_IN_OP_WIDTH(TEST_ALU_IN_OP_WIDTH)
85            ) alu_in_mon_bus(.clk(clk), .rst(rst));
86
87  assign alu_in_mon_bus.alu_rst = alu_in_agent_bus.alu_rst;
88  assign alu_in_mon_bus.ready   = DUT.ready_o;
89  assign alu_in_mon_bus.valid   = DUT.valid_i;
90  assign alu_in_mon_bus.op      = DUT.op_i;
91  assign alu_in_mon_bus.a       = DUT.a_i;
92  assign alu_in_mon_bus.b       = DUT.b_i;
93
94  alu_in_monitor_bfm #(
95        .ALU_IN_OP_WIDTH(TEST_ALU_IN_OP_WIDTH)
96        ) alu_in_agent_mon_bfm(alu_in_mon_bus.monitor_port);
97  // pragma uvmf custom alu_in_monitor_bus_connections end
98  alu_out_monitor_bfm #(
99        .ALU_OUT_RESULT_WIDTH(TEST_ALU_OUT_RESULT_WIDTH)
100       ) alu_out_agent_mon_bfm(alu_out_agent_bus.monitor_port);
101 alu_in_driver_bfm #(
102       .ALU_IN_OP_WIDTH(TEST_ALU_IN_OP_WIDTH)
103       ) alu_in_agent_drv_bfm(alu_in_agent_bus.initiator_port);
```

Instantiate & Wire Up the DUT
project_benches/alu/tb/testbench/hdl_top.sv



- Compiling The DUT
  - Go to the folder ***project_benches/alu/sim***
  - There is a Makefile for Linux users and a run.do for Windows users
  - **run.do** : add the vlog command to compile the alu.v source file
  - **Makefile** : uncomment line 152 which activates the comp_alu_dut target. A default filename and path is generated in the Makefile (line 123) which you would have to modify for other designs.

Makefile

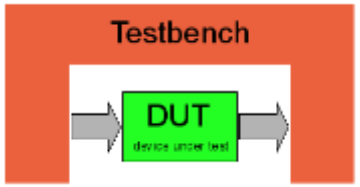
```
120 # UVMF_CHANGE_ME : Reference DUT source.
121 alu_DUT =\
122 $(UVMF_PROJECT_DIR)/rtl/verilog/alu.v
```

```
146 # UVMF_CHANGE_ME : Add make target to compile your dut here
147 comp_alu_dut:
148     echo "Compile your DUT here"
149     $(HDL_COMP_CMD) $(alu_DUT)
```

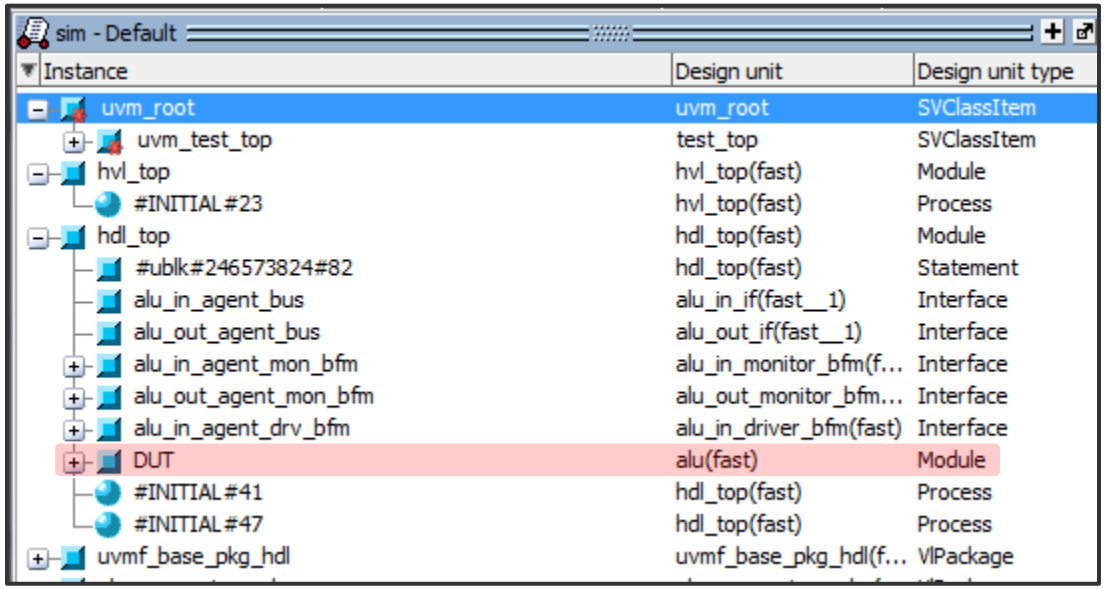
run.do

```
27 echo "Compile your DUT here"
28 vlog -sv $env(UVMF_PROJECT_DIR)/rtl/verilog/alu.v
```

Instantiate & Wire Up the DUT
project_benches/alu/tb/testbench/hdl_top.sv



- Simulating with the ALU DUT
  - Go to the folder ***project_benches/alu/sim***
  - Use either the run.do or the Makefile (make debug) to compile and load the simulation
  - Check that there are no compile errors and that the ALU (instance name DUT) appears in the hierarchy of hdl_top



Adding Protocol Information To The Driver BFM

verification_ip/interface_packages/alu_in_pkg/src/alu_in_driver_bfm.sv

- Modifying the alu_in driver BFM
  - Go to the folder ***verification_ip/interface_packages/alu_in_pkg/src***
  - Edit the file ***alu_in_driver_bfm.sv*** and locate the '***initiate_and_get_response***' task
  - By default the UVMF generator just has 4 consecutive clock delays inserted in to the driver. No data is actually driven onto the alu_in bus interface
  - This code needs to be modified to implement the interface protocol

Original Code

```
178 // pragma uvmf custom initiate_and_get_response begin
179 // ****************************************************************************
180 // UVMF_CHANGE_ME
181 // This task is used by an initator.  The task first initiates a transfer then
182 // waits for the responder to complete the transfer.
183     task initiate_and_get_response(
184         // This argument passes transaction variables used by an initiator
185         // to perform the initial part of a protocol transfer.  The values
186         // come from a sequence item created in a sequence.
187         input alu_in_initiator_s alu_in_initiator_struct,
188         // This argument is used to send data received from the responder
189         // back to the sequence item.  The sequence item is returned to the sequence.
190         output alu_in_responder_s alu_in_responder_struct
191         );// pragma tbx xtf
```

```
223     // Initiate a transfer using the data received.
224     @(posedge clk_i);
225     @(posedge clk_i);
226     // Wait for the responder to complete the transfer then place the responder data into
227     // alu_in_responder_struct.
228     @(posedge clk_i);
229     @(posedge clk_i);
```

Adding Protocol Information To The Driver BFM

verification_ip/interface_packages/alu_in_pkg/src/alu_in_driver_bfm.sv

- Modifying the alu_in driver BFM
  - Replace the 4 consecutive clock cycle delays with the following code

Modified Code

```
223     // Initiate a transfer using the data received.
224     @(posedge clk_i);
225     case (alu_in_initiator_struct.op)
226       rst_op   : do_assert_rst(alu_in_initiator_struct);
227       default  : alu_in_op(alu_in_initiator_struct);
228     endcase
229
230     $display("alu_in_driver_bfm: Inside do_transfer()");
231   endtask
```

NOTES:
The reset operation only drives the ALU reset pin and is therefore handled separately in it's own task.

  - Add the following 2 tasks to the module

```
233 // ****************************************************************************
234   task do_assert_rst(input alu_in_initiator_s alu_in_initiator_struct);
235     $display("%g ***************  Starting Reset", $time);
236     op_o <= alu_in_initiator_struct.op;
237     alu_rst_o <= 1'b0;
238     repeat (10) @(posedge clk_i);
239     alu_rst_o <= 1'b1;
240     repeat (5) @(posedge clk_i);
241     $display("%g ***************  Ending Reset", $time);
242   endtask
```

New Code

```
244 // ****************************************************************************
245   task alu_in_op(input alu_in_initiator_s alu_in_initiator_struct);
246
247     while (ready_i == 1'b0) @(posedge clk_i);
248     valid_o <= 1'b1;
249     op_o    <= alu_in_initiator_struct.op;
250     a_o     <= alu_in_initiator_struct.a;
251     b_o     <= alu_in_initiator_struct.b;
252
253     @(posedge clk_i);
254     valid_o <= 1'b0;
255     op_o    <= {3{1'bx}};
256     a_o     <= {ALU_IN_OP_WIDTH{1'bx}};
257     b_o     <= {ALU_IN_OP_WIDTH{1'bx}};
258
259   endtask
```

NOTES:
For the reset operation we need to drive the correct op code on to the bus so that the monitor can recognize the reset operation.

Adding Protocol Information To The Driver BFM

verification_ip/interface_packages/alu_in_pkg/src/alu_in_driver_bfm.sv

- Modifying the alu_in driver BFM
  - Modify the code that generates the alu_rst_o signal as shown below

```
78   // INITIATOR mode input signals
79   tri  ready_i;
80   reg  ready_o = 'bz;
81
82   // INITIATOR mode output signals
83   tri  alu_rst_i;
84   reg  alu_rst_o = 'bz;
```

Original Code

```
144  // ****************************************************
145  // Always block used to return signals to reset
146  always @( negedge rst_i )
147     begin
148       // RESPONDER mode output signals
149       ready_o <= 'bZ;
150       // INITIATOR mode output signals
151       alu_rst_o <= 'bZ;
```

```
100  // These are signals marked as 'output' by the config file, but the outputs will
101  // not be driven by this BFM unless placed in INITIATOR mode.
102  assign bus.alu_rst = (initiator_responder == INITIATOR) ? alu_rst_o : 'bz;
103  assign alu_rst_i = bus.alu_rst;
```

```
78   // INITIATOR mode input signals
79   tri  ready_i;
80   reg  ready_o = 'b1;
81
82   // INITIATOR mode output signals
83   tri  alu_rst_i;
84   reg  alu_rst_o = 'b1;
```

Modified Code

```
144  // ****************************************************
145  // Always block used to return signals to reset
146  always @( negedge rst_i )
147     begin
148       // RESPONDER mode output signals
149       ready_o <= 'b1;
150       // INITIATOR mode output signals
151       alu_rst_o <= 'b1;
152       valid_o <= 'bz;
```

```
100  // These are signals marked as 'output' by the config file, but the outputs will
101  // not be driven by this BFM unless placed in INITIATOR mode.
102  assign bus.alu_rst = (initiator_responder == INITIATOR) ? (alu_rst_o && rst_i) : 'bz;
103  assign alu_rst_i = bus.alu_rst;
```

NOTES:

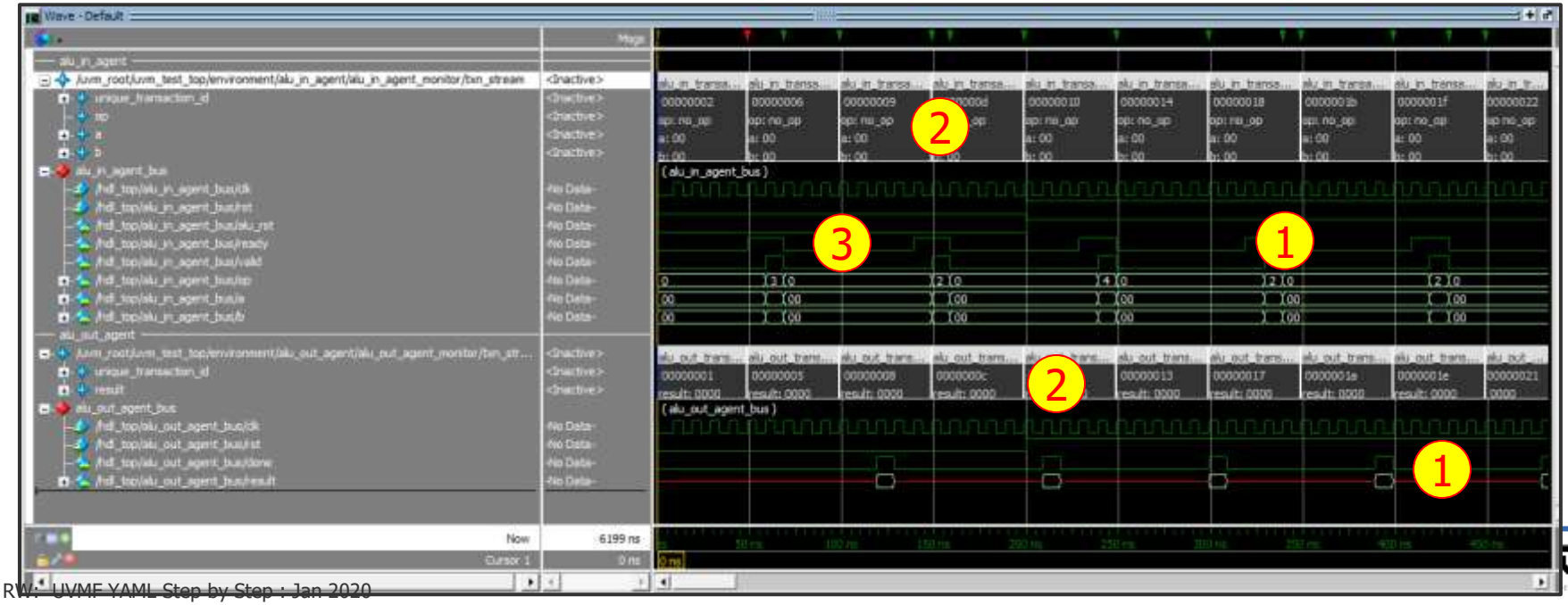
- alu_rst is active low so we initialize the default value of alu_rst_o to be '1'.
- We want to reset the ALU if either the top level reset (rst_i) is active or when a RST_OP operation is received which drives alu_rst_i low. So we logically AND the 2 reset driving signals alu_rst_

Adding Protocol Information To The Driver BFM

verification_ip/interface_packages/alu_in_pkg/src/alu_in_driver_bfm.sv

- Checking the driver BFM code changes
  - Use the run.do or the Makefile (make debug) to check that there are no compilation errors in the code you have modified/added.

1. If you look at the ALU signals (alu_in_agent_bus & alu_out_agent_bus) you will see that the testbench is sending operations to the ALU and that results are being generated.
2. The transactions are still showing incorrect value since the monitor code has not been modified yet. We will fix this next.
3. Some transactions are being sent during the reset period. We will fix this later.



Adding Protocol Information To The Monitor BFM

verification_ip/interface_packages/alu_in_pkg/src/alu_in_monitor_bfm.sv

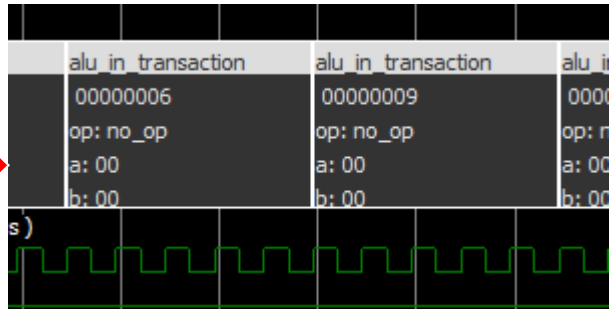
- Modifying the alu_in monitor BFM
  - Go to the folder ***verification_ip/interface_packages/alu_in_pkg/src***
  - Edit the file ***alu_in_monitor_bfm.sv*** and locate the '***do_monitor***' task
  - By default the UVMF generator just has 4 consecutive clock delays inserted in to the monitor. No data is actually read from the alu_in bus interface
  - This code needs to be modified to implement the interface protocol

Original Code

```
136   // ****************************************************************************
137
138   task do_monitor(output alu_in_monitor_s alu_in_monitor_struct);
```

```
157     // pragma uvmf custom do_monitor begin
158     // UVMF_CHANGE_ME : Implement protocol monitoring.  The commented reference code
159     // below are examples of how to capture signal values and assign them to
160     // structure members.  All available input signals are listed.  The 'while'
161     // code example shows how to wait for a synchronous flow control signal.  This
162     // task should return when a complete transfer has been observed.  Once this task is
163     // exited with captured values, it is then called again to wait for and observe
164     // the next transfer. One clock cycle is consumed between calls to do_monitor.
165     @(posedge clk_i);
166     @(posedge clk_i);
167     @(posedge clk_i);
168     @(posedge clk_i);
169     // pragma uvmf custom do_monitor end
```

NOTES
This is why we see the alu_in_transactions are all 4 cycles long and the displayed data values are just the language type defaults



Adding Protocol Information To The Monitor BFM

verification_ip/interface_packages/alu_in_pkg/src/alu_in_monitor_bfm.sv

- Modifying the alu_in monitor BFM
  - Replace the 4 consecutive clock cycle delays with the following code

```
166     while (alu_rst_i == 1'b0) @(posedge clk_i);
167
168     @(posedge valid_i or negedge alu_rst_i);
169     alu_in_monitor_struct.op = alu_in_op_t'(op_i);
170     alu_in_monitor_struct.a  = a_i;
171     alu_in_monitor_struct.b  = b_i;
172
173     // pragma uvmf custom do_monitor end
174   endtask
```

Modified Code

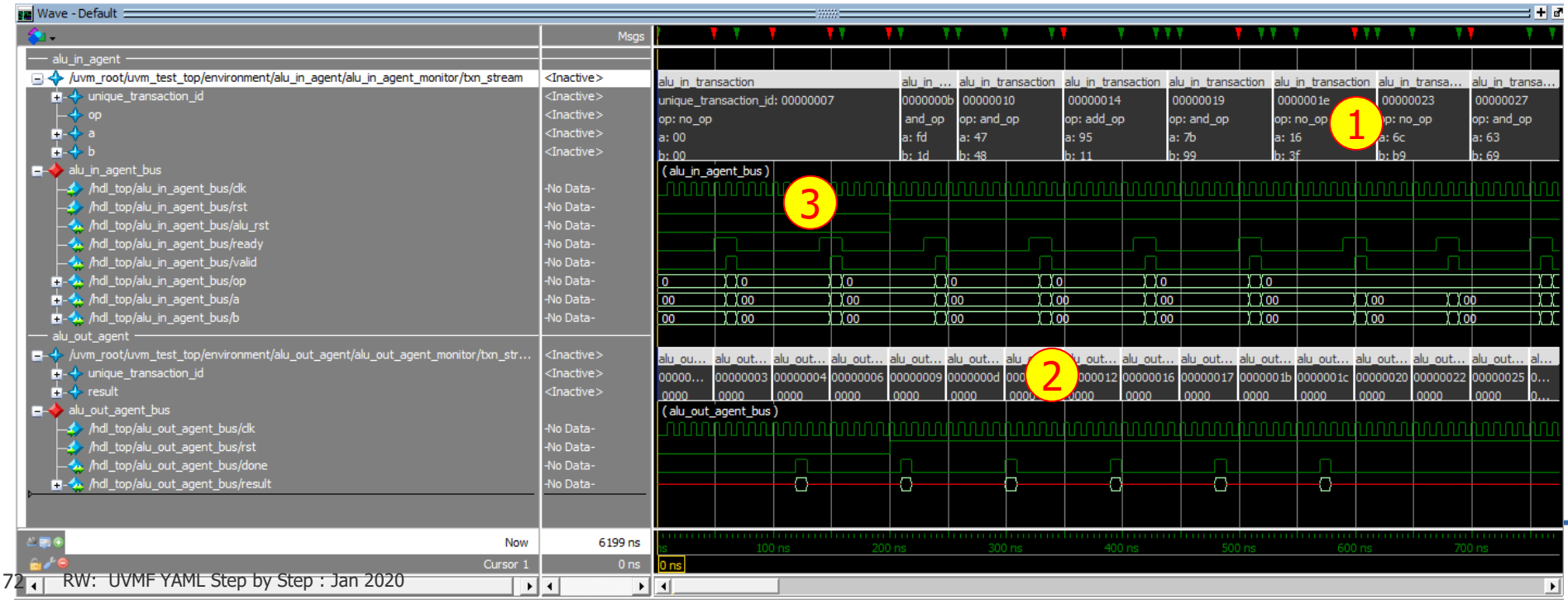
NOTES
- Wait until reset goes active
- Read bus values when valid goes high

Adding Protocol Information To The Monitor BFM

verification_ip/interface_packages/alu_in_pkg/src/alu_in_monitor_bfm.sv

- Checking the monitor BFM code changes
  - Use the run.do or the Makefile (make debug) to check that there are no compilation errors in the code you have modified/added.

1. The alu_in transactions are now showing the actual stimulus data values and the transaction lengths match the corresponding pin signal activity.
2. The alu_out_transactions are still showing default values since we have not modified the alu_out_driver BFM code yet.
3. Some transactions are being sent during the reset period. We will fix this later.



Adding Protocol Information To The Monitor BFM

verification_ip/interface_packages/alu_out_pkg/src/alu_out_monitor_bfm.sv

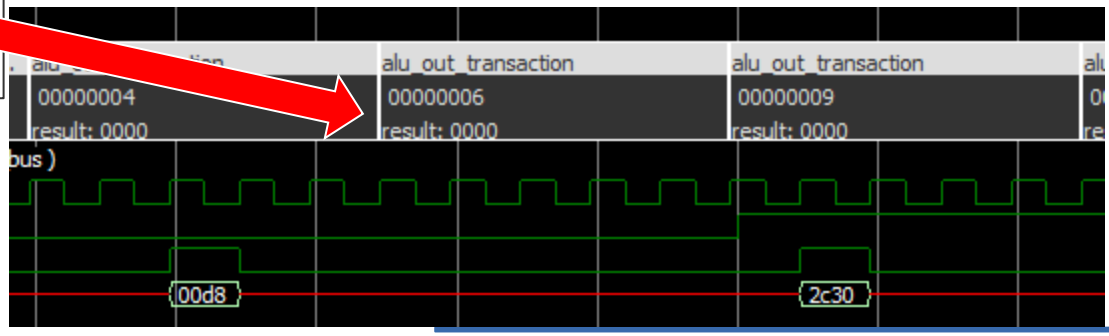
- Modifying the alu_out monitor BFM
  - Go to the folder ***verification_ip/interface_packages/alu_out_pkg/src***
  - Edit the file ***alu_out_monitor_bfm.sv*** and locate the '***do_monitor***' task
  - By default the UVMF generator just has 4 consecutive clock delays inserted in to the monitor. No data is actually read from the alu_out bus interface
  - This code needs to be modified to implement the interface protocol

Original Code

```
128   // ****************************************************************************
129
130   task do_monitor(output alu_out_monitor_s alu_out_monitor_struct);
```

```
151     @(posedge clk_i);
152     @(posedge clk_i);
153     @(posedge clk_i);
154     @(posedge clk_i);
155     // pragma uvmf custom do_monitor end
156   endtask
```

NOTES
This is why we see the alu_out_transactions are all 4 cycles long and the displayed 'result' data values are just the language type defaults



Adding Protocol Information To The Monitor BFM

verification_ip/interface_packages/alu_out_pkg/src/alu_out_monitor_bfm.sv

- Modifying the alu_out monitor BFM
  - Replace the 4 consecutive clock cycle delays with the following code

Modified Code

```
150     // the next transfer. One clock cycle is consumed between calls to do_monitor.
151     while ( done_i == 1'b0 ) @(posedge clk_i);
152     alu_out_monitor_struct.result = result_i;
153
154     // pragma uvmf custom do_monitor end
155   endtask
```

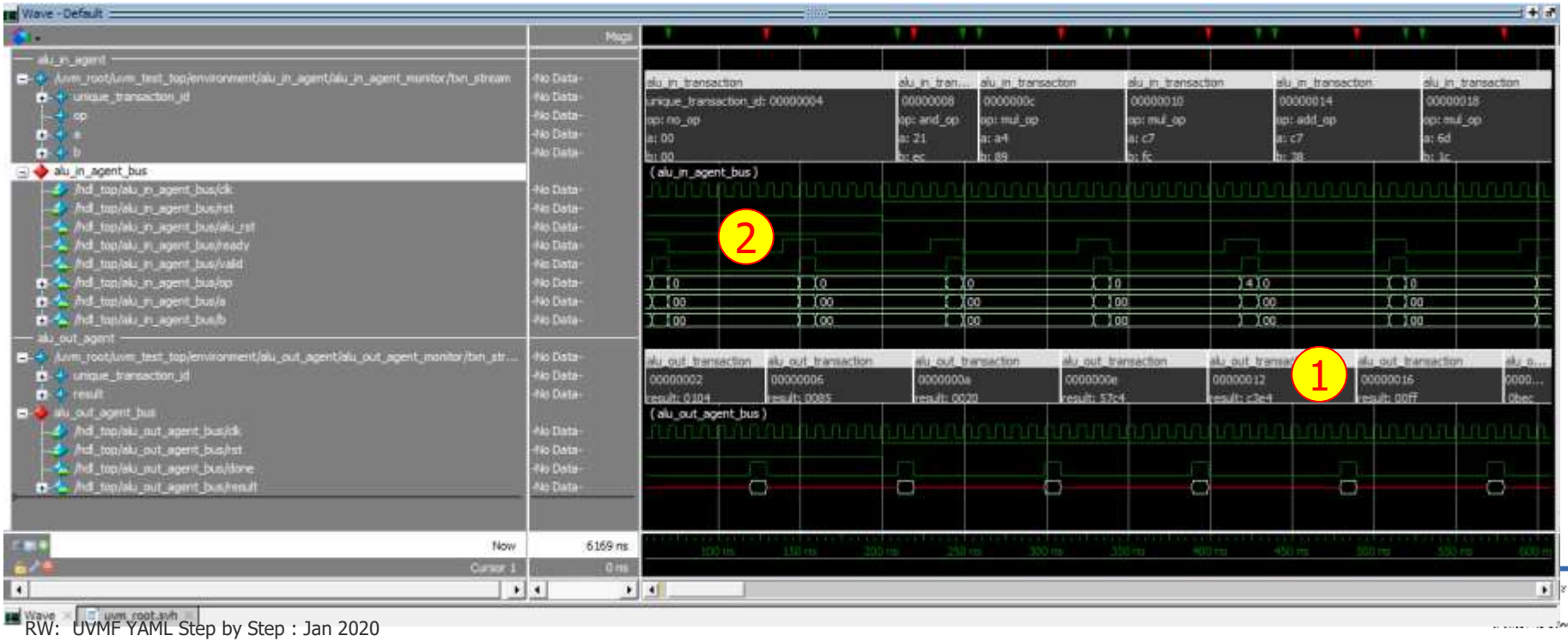
NOTES
- Wait until done_i goes high
- Read result value

Adding Protocol Information To The Monitor BFM

verification_ip/interface_packages/alu_out_pkg/src/alu_out_monitor_bfm.sv

- Checking the monitor BFM code changes
  - Use the run.do or the Makefile (make debug) to check that there are no compilation errors in the code you have modified/added.

1. The alu_out transactions are now showing the actual result data values and the transaction lengths match the corresponding pin signal activity.
2. Some transactions are still being sent during the reset period. We will fix this next.



Delaying Sequence Activity During Reset
project_benches/alu/tb/sequences/src/ alu_bench_sequence_base.svh

- Modifying the ALU default sequence
  - Go to the folder ***project_benches/alu/tb/sequences/src***
  - Edit the file ***alu_bench_sequence_base.svh*** and locate the `***body***` task
  - A repeat loop starts the sequence on the alu_in_agent at time 0
  - We need to add some code to delay starting the stimulus generation until the reset is inactive

Original Code

```
81   // ****************************************************************************
82   virtual task body();
83     // Construct sequences here
84     alu_in_agent_random_seq     = alu_in_agent_random_seq_t::type_id::create("alu_in_agent_random_seq");
85
86     fork
87       alu_in_agent_config.wait_for_reset();
88       alu_out_agent_config.wait_for_reset();
89     join
90     reg_model.reset();
91     // Start RESPONDER sequences here
92     fork
93     join_none
94     // Start INITIATOR sequences here
95     fork
96       repeat (25) alu_in_agent_random_seq.start(alu_in_agent_sequencer);
97     join
98     // UVMF_CHANGE_ME : Extend the simulation XXX number of clocks after
99     // the last sequence to allow for the last sequence item to flow
100    // through the design.
101    fork
102      alu_in_agent_config.wait_for_num_clocks(400);
103      alu_out_agent_config.wait_for_num_clocks(400);
104    join
```

Delaying Sequence Activity During Reset

project_benches/alu/tb/sequences/src/alu_bench_sequence_base.svh

- Modifying the ALU default sequence
  - Replace the RESPONDER fork/join with the code shown below
  - This calls some utility tasks provided in the agent configuration class

wait_for_reset : waits until the reset signal has been deasserted
wait_for_num_clocks : waits for the specified number of clock cycles

```
106     // Delay start of sequence until reset has ended and then wait a few clocks after that
107     alu_in_agent_config.wait_for_reset();
108     alu_in_agent_config.wait_for_num_clocks(10);
109
110     reg_model.reset();
111     // Start RESPONDER sequences here
112     fork
```

Modified Code

Adding DUT Behaviour To The Predictor
verification_ip/environment_packages/alu_env_pkg/src/alu_predictor.svh

- Modifying the ALU predictor
  - Go to the folder ***verification_ip/environment_packages/alu_env_pkg/src***
  - Edit the file ***alu_predictor.svh*** and locate the '***write_alu_in_agent_ae***' function
  - Transactions received through alu_in_agent_ae initiate the execution of this function
  - This function performs prediction of DUT output values based on DUT input, configuration and state
  - This code needs to be modified to implement the DUT functionality

Original Code

```
77    // FUNCTION: write_alu_in_agent_ae
78    // Transactions received through alu_in_agent_ae initiate the execution of this function.
79    // This function performs prediction of DUT output values based on DUT input, configuration and state
80    virtual function void write_alu_in_agent_ae(alu_in_transaction #(.ALU_IN_OP_WIDTH(ALU_IN_OP_WIDTH)) t);
81      // pragma uvmf custom alu_in_agent_ae_predictor begin
82      `uvm_info("PRED", "Transaction Received through alu_in_agent_ae", UVM_MEDIUM)
83      `uvm_info("PRED", {"            Data: ",t.convert2string()}, UVM_FULL)
84      // Construct one of each output transaction type.
85      alu_sb_ap_output_transaction = alu_sb_ap_output_transaction_t::type_id::create("alu_sb_ap_output_transaction");
86      //  UVMF_CHANGE_ME: Implement predictor model here.
87      `uvm_info("UNIMPLEMENTED_PREDICTOR_MODEL", "**********************************************************************
   *",UVM_NONE)
88      `uvm_info("UNIMPLEMENTED_PREDICTOR_MODEL", "UVMF_CHANGE_ME: The alu_predictor::write_alu_in_agent_ae function needs
   iction model",UVM_NONE)
89      `uvm_info("UNIMPLEMENTED_PREDICTOR_MODEL", "**********************************************************************
   *",UVM_NONE)
```

Adding DUT Behaviour To The Predictor

verification_ip/environment_packages/alu_env_pkg/src/alu_predictor.svh

- Modifying the ALU predictor
  - Insert the following case statement into the task to implement the ALU operations
  - Note that we deliberately ignore the RST_OP op code, taking care not to write a transaction out to the analysis export (which is connected to the scoreboard). Also note the removal of lines 75 and 76 from the original code.

```
72    // Construct one of each output transaction type.
73    alu_sb_ap_output_transaction = alu_out_transaction #()::type_id::create("alu_sb_ap_output_transaction");
74
75        case (t.op)
76          add_op: begin
77                      alu_sb_ap_output_transaction.result = t.a + t.b;
78                      `uvm_info("PREDICT",{"ALU_OUT: ",alu_sb_ap_output_transaction.convert2string()},UVM_MEDIUM);
79                      // Code for sending output transaction out through alu_sb_ap
80                      alu_sb_ap.write(alu_sb_ap_output_transaction);
81                  end
82          and_op: begin
83                      alu_sb_ap_output_transaction.result = t.a & t.b;
84                      `uvm_info("PREDICT",{"ALU_OUT: ",alu_sb_ap_output_transaction.convert2string()},UVM_MEDIUM);
85                      // Code for sending output transaction out through alu_sb_ap
86                      alu_sb_ap.write(alu_sb_ap_output_transaction);
87                  end
88          xor_op: begin
89                      alu_sb_ap_output_transaction.result = t.a ^ t.b;
90                      `uvm_info("PREDICT",{"ALU_OUT: ",alu_sb_ap_output_transaction.convert2string()},UVM_MEDIUM);
91                      // Code for sending output transaction out through alu_sb_ap
92                      alu_sb_ap.write(alu_sb_ap_output_transaction);
93                  end
94          mul_op: begin
95                      alu_sb_ap_output_transaction.result = t.a * t.b;
96                      `uvm_info("PREDICT",{"ALU_OUT: ",alu_sb_ap_output_transaction.convert2string()},UVM_MEDIUM);
97                      // Code for sending output transaction out through alu_sb_ap
98                      alu_sb_ap.write(alu_sb_ap_output_transaction);
99                  end
100       endcase // case (op_set)
101
102   endfunction
```

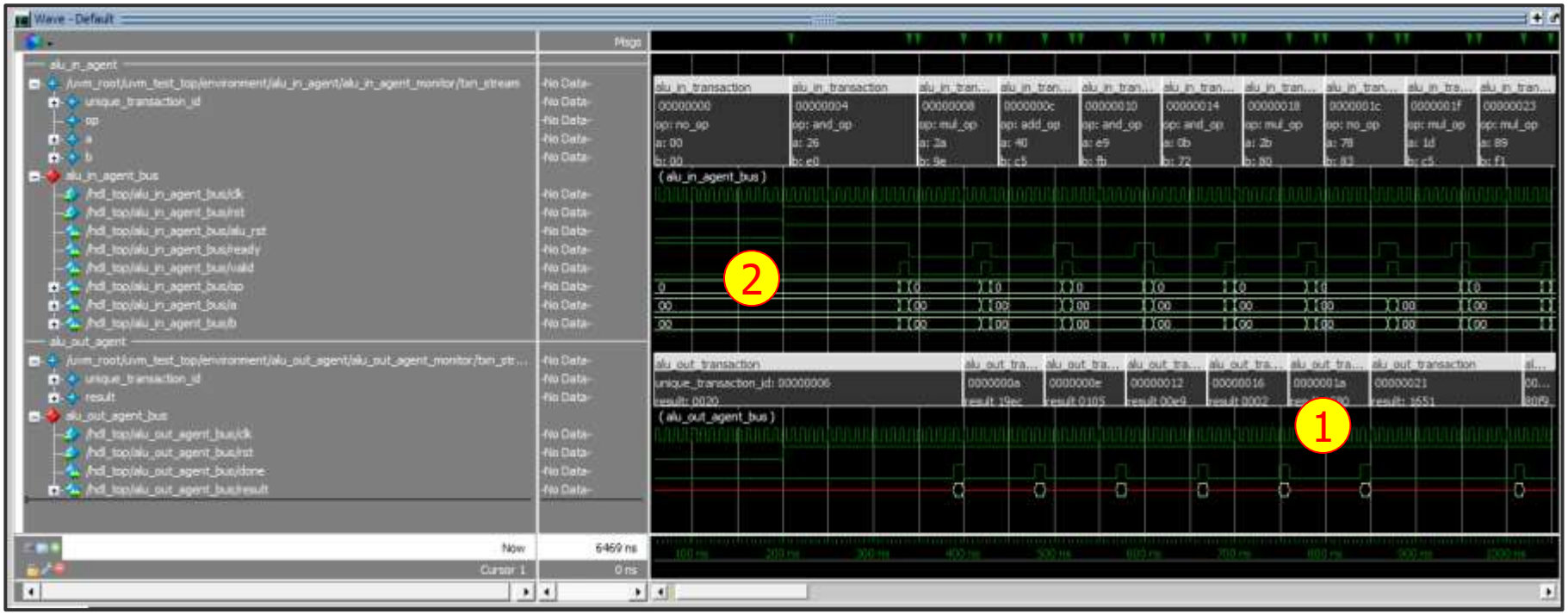
Modified Code

Adding DUT Behaviour To The Predictor

verification_ip/environment_packages/alu_env_pkg/src/alu_predictor.svh

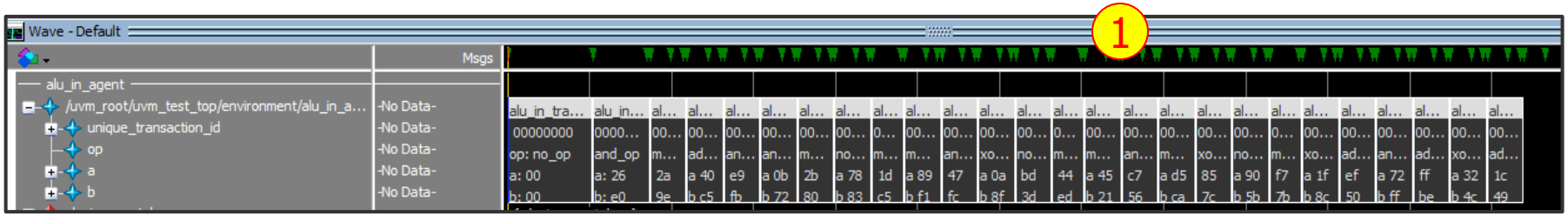
- Checking the predictor code changes
  - Use the run.do or the Makefile (make debug) to check that there are no compilation errors in the code you have modified/added.

1. The alu_out transactions are now showing the actual result data values and the transaction lengths match the corresponding pin signal activity.
2. Transactions are no longer being sent during the reset period.



Status So Far

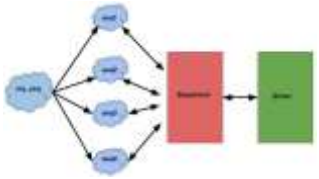
- Basic ALU operation appears to be working
  1. There should be no errors at this stage. In the wave window there should be no red triangles, which would indicate UVM Errors from the scoreboard



- Still To Do
  - Create a new test & sequence to exercise the RST_OP which is currently not being tested.
  - This is due to the constraint we specified back in the YAML config file for the alu_in interface which only selects from the following ALU operations.

```
34  //Constraints for the transaction variables:
35    constraint valid_op_c { op inside {no_op, add_op, and_op, xor_op, mul_op}; }
```

Creating an interface reset sequence
verification_ip/interface_packages/alu_in_pkg/src
alu_in_reset_sequence.svh



- UVMF Generated Sequence

1. alu_in agent was generated with the following sequence

   interface_packages/alu_in_pkg/src/alu_in_random_sequence.svh

2. It randomizes ALU operations, selecting from no_op, add_op, and_op, xor_op & mul_op

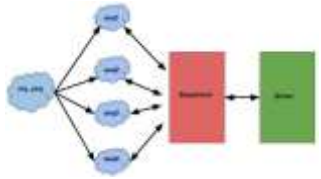
3. Copy the alu_in_random_sequence.svh file to alu_in_reset_sequence.svh

4. Edit the sequence and change all references to alu_in_random_sequence to alu_in_reset_sequence.

5. After the randomization of the alu_in_transaction, set the ALU op = RST_OP

Creating an interface reset sequence

verification_ip/interface_packages/alu_in_pkg/src
alu_in_reset_sequence.svh



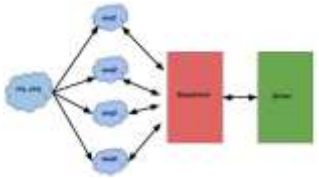
```
20  class alu_in_reset_sequence  #(
21        int ALU_IN_OP_WIDTH = 8
22        )
23  extends alu_in_sequence_base  #(
24                                .ALU_IN_OP_WIDTH(ALU_IN_OP_WIDTH)
25                               ) ;
26
27    `uvm_object_param_utils( alu_in_reset_sequence #(
28                             ALU_IN_OP_WIDTH
29                              ))
30
31  //------------------------------------------------------------------
32    function new(string name = "");
33      super.new(name);
34    endfunction: new
35
36  // ****************************************************************************
41    task body();
42
43      begin
44        // Construct the transaction
45        req=alu_in_transaction #(
46                  .ALU_IN_OP_WIDTH(ALU_IN_OP_WIDTH)
47                 ) ::type_id::create("req");
48
49        start_item(req);
50        // Randomize the transaction
51        if(!req.randomize()) `uvm_fatal("RANDOMIZE_FAILURE", "alu_in_reset_sequence::body()-alu_in_transaction")
52        // force the operation to be a reset.
53        req.op = rst_op;
54        // Send the transaction to the alu_in_driver_bfm via the sequencer and alu_in_driver.
55        finish_item(req);
56        `uvm_info("reset_seq response from driver", req.convert2string(),UVM_MEDIUM)
57      end
58
59    endtask: body
60
61  endclass: alu_in_reset_sequence
```

Modified Code

Creating an interface reset sequence

verification_ip/interface_packages/alu_in_pkg

alu_in_pkg.sv



STEPS

- Update the alu_in_pkg to include the newly created alu_in_reset_sequence.svh file
- The compilation script will compile this package and therefore all files that it includes

```
31 package alu_in_pkg;
32
33     import uvm_pkg::*;
34     import uvmf_base_pkg_hdl::*;
35     import uvmf_base_pkg::*;
36     import alu_in_pkg_hdl::*;
37
38     `include "uvm_macros.svh"
39
40     // pragma uvmf custom package_imports_additional begin
41     // pragma uvmf custom package_imports_additional end
42
43     `include "src/alu_in_macros.svh"
44
45     export alu_in_pkg_hdl::*;
46
47
48
49     // Parameters defined as HVL parameters
50
51     `include "src/alu_in_typedefs.svh"
52     `include "src/alu_in_transaction.svh"
53
54     `include "src/alu_in_configuration.svh"
55     `include "src/alu_in_driver.svh"
56     `include "src/alu_in_monitor.svh"
57
58     `include "src/alu_in_transaction_coverage.svh"
59     `include "src/alu_in_sequence_base.svh"
60     `include "src/alu_in_random_sequence.svh"
61
62     `include "src/alu_in_responder_sequence.svh"
63     `include "src/alu_in2reg_adapter.svh"
64
65     `include "src/alu_in_agent.svh"
66
67     // pragma uvmf custom package_item_additional begin
68     // UVMF_CHANGE_ME : When adding new interface sequences to the src directory
69     //     be sure to add the sequence file here so that it will be
70     //     compiled as part of the interface package.  Be sure to place
71     //     the new sequence after any base sequences of the new sequence.
72     `include "src/alu_in_reset_sequence.svh"
73     // pragma uvmf custom package_item_additional end
74
75 endpackage
```

Modified Code

Creating a new bench virtual sequence
project_benches/alu/tb/sequences/src
alu_random_sequence.svh

- UVMF Generated Sequence

1. alu_in bench was generated with the following virtual sequence

   project_benches/alu/tb/sequences/src/alu_bench_sequence_base.svh

2. This is the default sequence that gets ran by the default test.

3. Extend this sequence to create a new sequence called alu_random_sequence.

4. We have the handle for the alu_in_random sequence from the base class, but we need to define a handle for the new alu_in_reset_sequence

5. In the body of the sequence we will generate some random ALU operations, followed by a reset operation and then we will generate some more random ALU operations.

Creating a new bench virtual sequence

project_benches/alu/tb/sequences/src
alu_random_sequence.svh

```
18 class alu_random_sequence extends alu_bench_sequence_base;
19
20   `uvm_object_utils( alu_random_sequence )
21
22   // Instantiate sequences here
23   typedef alu_in_reset_sequence #(
24         .ALU_IN_OP_WIDTH(TEST_ALU_IN_OP_WIDTH)
25         ) alu_in_agent_reset_seq_t;
26   alu_in_agent_reset_seq_t alu_in_agent_reset_seq;
27
28   // ****************************************************************************
29   function new( string name = "" );
30     super.new( name );
31   endfunction
32
33   // ****************************************************************************
34   virtual task body();
35
36     // Construct sequences here
37     alu_in_agent_random_seq    = alu_in_agent_random_seq_t::type_id::create("alu_in_agent_random_seq");
38     alu_in_agent_reset_seq     = alu_in_agent_reset_seq_t::type_id::create("alu_in_agent_reset_seq");
39
40     // Delay start of sequence until reset has ended and then wait a few clocks after that
41     alu_in_agent_config.wait_for_reset();
42     alu_in_agent_config.wait_for_num_clocks(10);
43
44     repeat (10) alu_in_agent_random_seq.start(alu_in_agent_sequencer);
45     alu_in_agent_reset_seq.start(alu_in_agent_sequencer);
46     repeat (5) alu_in_agent_random_seq.start(alu_in_agent_sequencer);
47
48     // UVMF_CHANGE_ME : Extend the simulation XXX number of clocks after
49     // the last sequence to allow for the last sequence item to flow
50     // through the design.
51     alu_in_agent_config.wait_for_num_clocks(50);
52
53   endtask
```

3

4

4

5

New Sequence Code

Creating a new bench level sequence

project_benches/alu/tb/sequences
alu_sequence_pkg.sv

STEPS

- Update the alu_sequences_pkg to include the newly created alu_random_sequence.svh file
- The compilation script will compile this package and therefore all files that it includes

```
22 package alu_sequences_pkg;
23   import uvm_pkg::*;
24   import uvmf_base_pkg::*;
25   import mvc_pkg::*;
26   import mgc_apb3_v1_0_pkg::*;
27   import alu_in_pkg::*;
28   import alu_in_pkg_hdl::*;
29   import alu_out_pkg::*;
30   import alu_out_pkg_hdl::*;
31   import alu_parameters_pkg::*;
32   import alu_env_pkg::*;
33   import qvip_agents_params_pkg::*;
34   import alu_reg_pkg::*;
35   `include "uvm_macros.svh"
36
37   // pragma uvmf custom package_imports_additional begin
38   // pragma uvmf custom package_imports_additional end
39
40   `include "src/alu_bench_sequence_base.svh"
41   `include "src/register_test_sequence.svh"
42   `include "src/example_derived_test_sequence.svh"
43
44   // pragma uvmf custom package_item_additional begin
45   // UVMF_CHANGE_ME : When adding new sequences to the src directory
46   //    be sure to add the sequence file here so that it will be
47   //    compiled as part of the sequence package.  Be sure to place
48   //    the new sequence after any base sequences of the new sequence.
49   `include "src/alu_random_sequence.svh"
50   // pragma uvmf custom package_item_additional end
```

Modified Code

Adding a New UVM Test

project_benches/alu/tb/tests/src
alu_random_test.svh

- Example derived test provided in

  tests/src/example_dervied_test.svh

1. Create a new test, ***alu_random_test*** & extend it from ***test_top***

2. In the build phase, specify a factory override of the default sequence (which is ***alu_bench_sequence_base***) to replace it with the new sequence ***alu_random_sequence***.

```
19 class alu_random_test extends test_top;
20
21   `uvm_component_utils( alu_random_test );
22
23   function new( string name = "", uvm_component parent = null );
24     super.new( name, parent );
25   endfunction
26
27   virtual function void build_phase(uvm_phase phase);
28     // The factory override below is an example of how to replace the alu_bench_sequence_base
29     // sequence with the example_derived_test_sequence.
30     alu_bench_sequence_base::type_id::set_type_override(alu_random_sequence::get_type());
31     // Execute the build_phase of test_top AFTER all factory overrides have been created.
32     super.build_phase(phase);
33     // pragma uvmf custom configuration_settings_post_randomize begin
34     // UVMF_CHANGE_ME Test specific configuration values can be set here.
35     // The configuration structure has already been randomized.
36     // pragma uvmf custom configuration_settings_post_randomize end
37   endfunction
38
39 endclass
```

New Code

Adding a New UVM Test

project_benches/alu/tb/tests
alu_test_pkg.sv

- Add the new test to the alu_test_pkg

```
21 package alu_tests_pkg;
22
23    import uvm_pkg::*;
24    import uvmf_base_pkg::*;
25    import alu_parameters_pkg::*;
26    import alu_env_pkg::*;
27    import alu_sequences_pkg::*;
28   import alu_in_pkg::*;
29   import alu_in_pkg_hdl::*;
30   import alu_out_pkg::*;
31   import alu_out_pkg_hdl::*;
32    import qvip_agents_pkg::*;
33    import mvc_pkg::*;
34    import mgc_apb3_v1_0_pkg::*;
35
36
37    `include "uvm_macros.svh"
38
39   // pragma uvmf custom package_imports_additional begin
40   // pragma uvmf custom package_imports_additional end
41
42    `include "src/test_top.svh"
43    `include "src/register_test.svh"
44    `include "src/example_derived_test.svh"
45
46   // pragma uvmf custom package_item_additional begin
47   // UVMF_CHANGE_ME : When adding new tests to the src directory
48   //    be sure to add the test file here so that it will be
49   //    compiled as part of the test package.  Be sure to place
50   //    the new test after any base tests of the new test.
51    `include "src/alu_random_test.svh"
52   // pragma uvmf custom package_item_additional end
```

Modified Code

Simulating The New Test

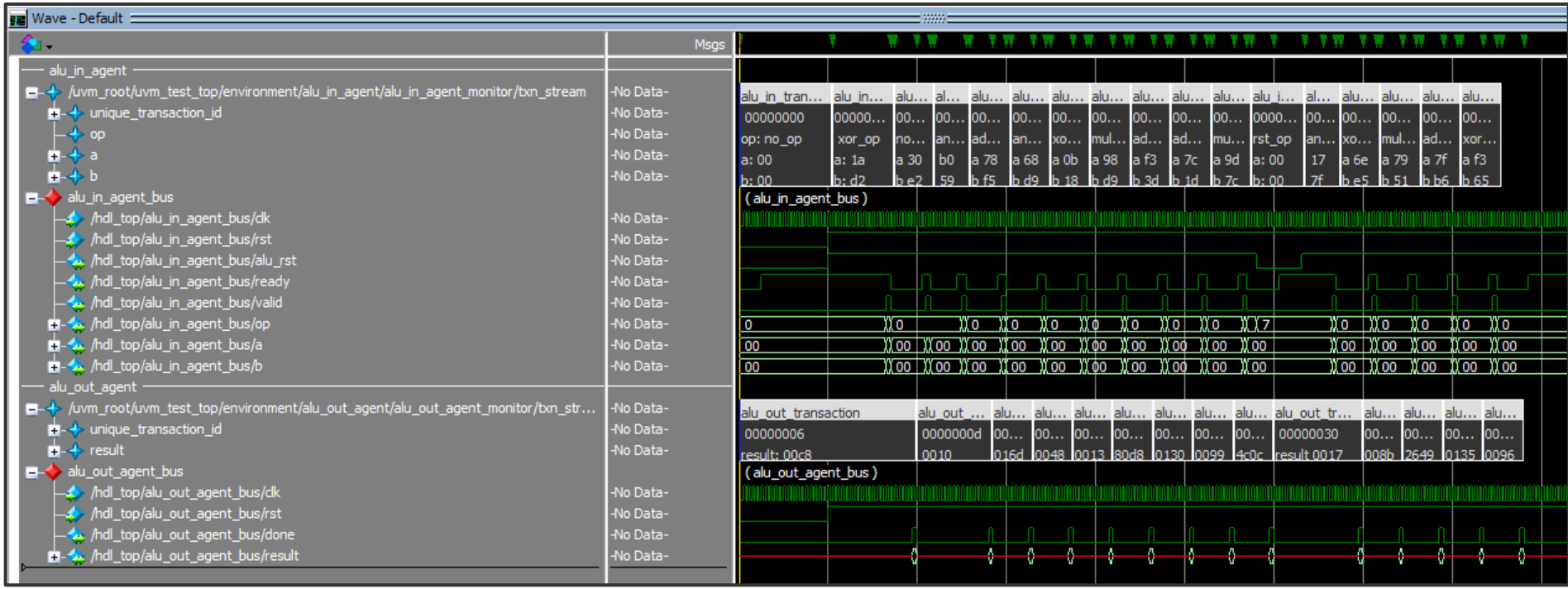
- Go to the ***project_benches/alu/sim*** folder
- Windows Users
  - Edit run.do and modify the modify the last line where +UVM_TESTNAME specifies the test to run

```
32  vopt -32  +acc    hvl_top hdl_top   -o optimized_debug_top_tb
33  vsim -i  -32  -sv_seed random +UVM_TESTNAME=alu_random_test  +UVM_VERBOSITY=UVM_HIGH
34
```

- Linux Users
  - Execute 'make debug TEST_NAME=alu_random_test'

Simulating The New Test

- Observe from the wave window
  1. No operations occur during the reset
  2. 10 random operations are then applied to the ALU
  3. A reset is then applied to the ALU
  4. 5 further random operations are then applied to the ALU



Adding Color To The Transactions

verification_ip/interface_packages/alu_in_pkg/src
alu_in_transaction.svh

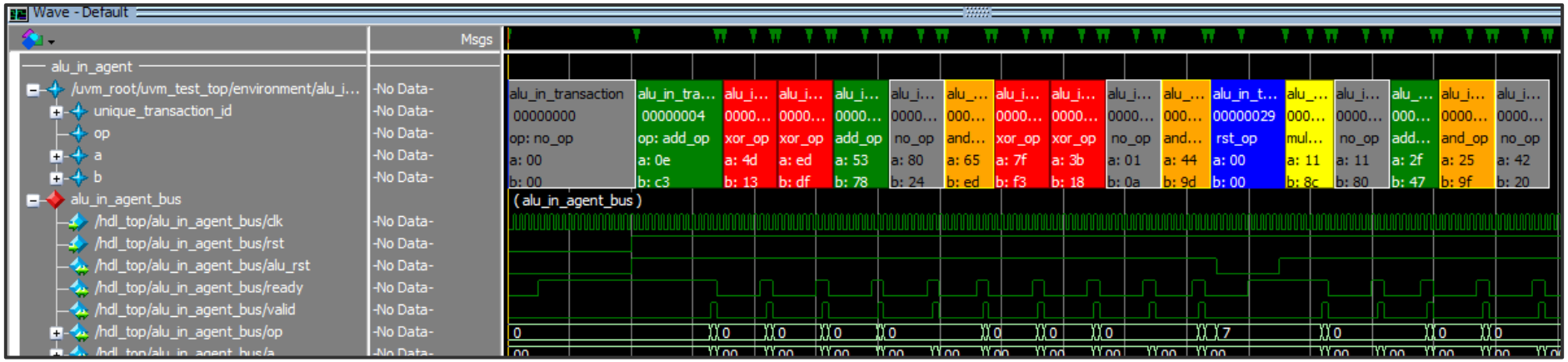
- Edit the file alu_in_transaction.svh
- Find the function called 'add_to_wave'
  - It contains some comments about adding color to the transactions
  - Add the code highlighted below to the function which will assign a different color to the transactions depending on the opcode value

```
92   virtual function void add_to_wave(int transaction_viewing_stream_h);
93     if (transaction_view_h == 0)
94       transaction_view_h = $begin_transaction(transaction_viewing_stream_h,"alu_in_transaction",start_time);
95     case (op)
96       no_op :   $add_color(transaction_view_h,"grey");
97       add_op :  $add_color(transaction_view_h,"green");
98       and_op :  $add_color(transaction_view_h,"orange");
99       xor_op :  $add_color(transaction_view_h,"red");
100      mul_op :  $add_color(transaction_view_h,"yellow");
101      rst_op :  $add_color(transaction_view_h,"blue");
102      default : $add_color(transaction_view_h,"grey");
103    endcase
104    super.add_to_wave(transaction_view_h);
105// UVMF_CHANGE_ME : Eliminate transaction variables not wanted in transaction viewing in the waveform viewer
106    $add_attribute(transaction_view_h,op,"op");
107    $add_attribute(transaction_view_h,a,"a");
108    $add_attribute(transaction_view_h,b,"b");
109    $end_transaction(transaction_view_h,end_time);
110    $free_transaction(transaction_view_h);
111  endfunction
```

Re-Simulate The New Test

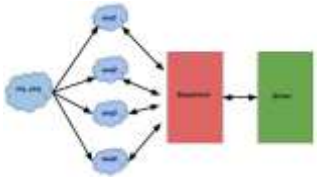
- Observe from the wave window

  1. The alu_in transactions are now color coded depending on the op code
  2. The colors match those specified in the add_to_wave function of the alu_in_transaction class



- That Completes The Steps To Get The UVMF Environment Running

Modifying an APB QVIP sequence
project_benches/alu/tb/sequences/src
apb3_random_sequence.svh



- APB QVIP Example Sequence

1. QVIP ships with example tests and sequences for all supported protocols

   $QUESTA_MVC_HOME/examples

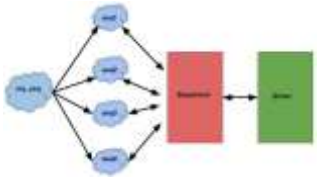
2. Copy the $QUESTA_MVC_HOME/examples/apb3/register_layering/apb_bus_sequence.svh file to apb3_random_sequence.svh (see path in slide title)

3. Edit the sequence and change all references from apb_bus_sequence to apb3_random_sequence. We will use this sequence to write to the ALU memory.

4. We will then modify the address randomization to only allow addresses from 0x10 to 0x1000.

Modifying an APB QVIP sequence

project_benches/alu/tb/sequences/src
apb3_random_sequence.svh



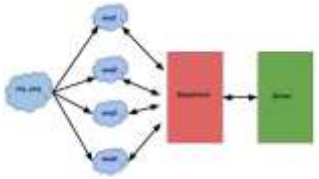
3

```
19 class apb3_random_sequence #( int SLAVE_COUNT   = 1 ,
20                               int ADDRESS_WIDTH = 32,
21                               int WDATA_WIDTH   = 32,
22                               int RDATA_WIDTH   = 32 ) extends mvc_sequence;
23
24   typedef apb3_host_write    #(SLAVE_COUNT,
25                                ADDRESS_WIDTH,
26                                WDATA_WIDTH,
27                                RDATA_WIDTH) wr_txn_t;
28
29   typedef apb3_host_read     #(SLAVE_COUNT,
30                                ADDRESS_WIDTH,
31                                WDATA_WIDTH,
32                                RDATA_WIDTH) rd_txn_t;
33
34   typedef apb3_random_sequence #(SLAVE_COUNT,
35                                ADDRESS_WIDTH,
36                                WDATA_WIDTH,
37                                RDATA_WIDTH) this_t;
38
39   typedef apb3_vip_config    #(SLAVE_COUNT,
40                                ADDRESS_WIDTH,
41                                WDATA_WIDTH,
42                                RDATA_WIDTH) cfg_t;
43
44   `uvm_object_param_utils( this_t )
```

Modified Code

Modifying an APB QVIP sequence

project_benches/alu/tb/sequences/src
apb3_random_sequence.svh



```
74   task body();
75
76     write_item_t write_item = write_item_t::type_id::create("write_seq");
77     read_item_t  read_item = read_item_t::type_id::create("read_seq");
78
79     apb3_config = cfg_t::get_config(m_sequencer);
80
81
82     apb3_config.wait_for_reset();
83     apb3_config.wait_for_clock();
84
85   start_item( write_item );
86   if (!write_item.randomize() with {
87                                     'h10 <= write_item.addr && write_item.addr < 'h1000;
88                                    }
89      ) `uvm_fatal("apb3_random_sequence","Randomization failure in write transaction")
90   finish_item( write_item );
91
92   start_item( read_item );
93   if (!read_item.randomize() with {read_item.slave_id == write_item.slave_id;
94                                    read_item.addr     == write_item.addr;
95                                   }
96      ) `uvm_fatal("apb3_random_sequence","Randomization failure in write transaction")
97
98   finish_item( read_item );
99
100  endtask
```

4

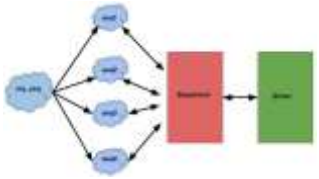
3

3

Modified Code

Modifying an APB QVIP sequence

project_benches/alu/tb/sequences/ alu_sequences_pkg.sv



STEPS

- Update the alu_sequences_pkg to include the newly created apb3_random_sequence.svh file
- The compilation script will compile this package and therefore all files that it includes

```
22 package alu_sequences_pkg;
23   import uvm_pkg::*;
24   import uvmf_base_pkg::*;
25   import mvc_pkg::*;
26   import mgc_apb3_v1_0_pkg::*;
27   import alu_in_pkg::*;
28   import alu_in_pkg_hdl::*;
29   import alu_out_pkg::*;
30   import alu_out_pkg_hdl::*;
31   import alu_parameters_pkg::*;
32   import alu_env_pkg::*;
33   import qvip_agents_params_pkg::*;
34   import alu_reg_pkg::*;
35   `include "uvm_macros.svh"
36
37   // pragma uvmf custom package_imports_additional begin
38   // pragma uvmf custom package_imports_additional end
39
40   `include "src/alu_bench_sequence_base.svh"
41   `include "src/register_test_sequence.svh"
42   `include "src/example_derived_test_sequence.svh"
43
44   // pragma uvmf custom package_item_additional begin
45   // UVMF_CHANGE_ME : When adding new sequences to the src directory
46   //    be sure to add the sequence file here so that it will be
47   //    compiled as part of the sequence package.  Be sure to place
48   //    the new sequence after any base sequences of the new sequence.
49   `include "src/alu_random_sequence.svh"
50   `include "src/apb3_random_sequence.svh"
51   // pragma uvmf custom package_item_additional end
52
53 endpackage
```

Modified Code

Creating a new bench virtual sequence
project_benches/alu/tb/sequences/src
apb3_alu_random_sequence.svh

- APB QVIP Example Sequence

1. Copy the apb3_random_sequence.svh to apb3_alu_random_sequence.svh (see path in slide title)

2. Edit the sequence and change all references of apb3_random_sequence to apb3_alu_random_sequence. We will use this sequence to perform ALU operations.

3. In the body of the sequence we will write a random ALU operand to Operand A and Operand B, then we will write a random ALU operation to the ALU, and finally we will write a 0x1 to the ALU control register to force the ALU to perform the operation. We will then write and read some random locations in the APB slave memory.

Creating a new bench virtual sequence

project_benches/alu/tb/sequences/src
apb3_alu_random_sequence.svh

2

```
19 class apb3_alu_random_sequence #( int SLAVE_COUNT   = 1 ,
20                                   int ADDRESS_WIDTH = 32,
21                                   int WDATA_WIDTH   = 32,
22                                   int RDATA_WIDTH   = 32 ) extends mvc_sequence;
23
24   typedef apb3_host_write    #(SLAVE_COUNT,
25                                ADDRESS_WIDTH,
26                                WDATA_WIDTH,
27                                RDATA_WIDTH) wr_txn_t;
28
29   typedef apb3_host_read     #(SLAVE_COUNT,
30                                ADDRESS_WIDTH,
31                                WDATA_WIDTH,
32                                RDATA_WIDTH) rd_txn_t;
33
34   typedef apb3_alu_random_sequence #(SLAVE_COUNT,
35                                ADDRESS_WIDTH,
36                                WDATA_WIDTH,
37                                RDATA_WIDTH) this_t;
38
39   typedef apb3_vip_config    #(SLAVE_COUNT,
40                                ADDRESS_WIDTH,
41                                WDATA_WIDTH,
42                                RDATA_WIDTH) cfg_t;
43
44   `uvm_object_param_utils( this_t )
```

2

New Sequence Code

Creating a new bench virtual sequence

project_benches/alu/tb/sequences/src
apb3_alu_random_sequence.svh

```
74   task body();
75
76     write_item_t write_item = write_item_t::type_id::create("write_seq");
77     read_item_t  read_item = read_item_t::type_id::create("read_seq");
78
79     apb3_config = cfg_t::get_config(m_sequencer);
80
81
82     apb3_config.wait_for_reset();
83     apb3_config.wait_for_clock();
84
85     // Write Operand A through APB
86     start_item( write_item );
87       if ( !write_item.randomize() with {write_item.addr == 'h0;}
88           ) `uvm_error("APB3_ALU_RANDOM_SEQUENCE", "Randomization failure for write transaction")
89     finish_item( write_item );
90
91     // Write Operand B through APB
92     start_item( write_item );
93       if ( !write_item.randomize() with {write_item.addr == 'h4;}
94           ) `uvm_error("APB3_ALU_RANDOM_SEQUENCE", "Randomization failure for write transaction")
95     finish_item( write_item );
96
97     // Write Operation through APB
98     start_item( write_item );
99       if ( !write_item.randomize() with {write_item.addr == 'h8;
100                                         'h0 < write_item.wr_data && write_item.wr_data <= 'h4;}
101          ) `uvm_error("APB3_ALU_RANDOM_SEQUENCE", "Randomization failure for write transaction")
102    finish_item( write_item );
103
104    // Write Control Register through APB to perform the Operation
105    start_item( write_item );
106      if ( !write_item.randomize() with {write_item.addr    == 'h10;
107                                         write_item.wr_data == 'h1;}
108          ) `uvm_error("APB3_ALU_RANDOM_SEQUENCE", "Randomization failure for write transaction")
109    finish_item( write_item );
110
111    // Execute write followed by read to the same address and slave ID
112    start_item( write_item );
113      if ( !write_item.randomize() with {
114                                         'h10 <= write_item.addr && write_item.addr < 'h1000;
115                                        }
116          ) `uvm_error("APB3_ALU_RANDOM_SEQUENCE",
117                       "Randomization failure for write transaction")
118    finish_item( write_item );
119    start_item  ( read_item );
120      if ( !read_item.randomize() with {
121                                       read_item.addr     == write_item.addr;
122                                       read_item.slave_id == write_item.slave_id;
123                                      }
124          ) `uvm_error("APB3_ALU_RANDOM_SEQUENCE",
125                       "Randomization failure for read transaction")
126    finish_item ( read_item );
127
128  endtask
```

2

3

New Sequence Code

Creating a new bench level sequence

project_benches/alu/tb/sequences
alu_sequence_pkg.sv

STEPS

- Update the alu_sequences_pkg to include the newly created apb3_alu_random_sequence.svh file
- The compilation script will compile this package and therefore all files that it includes

```
22 package alu_sequences_pkg;
23   import uvm_pkg::*;
24   import uvmf_base_pkg::*;
25   import mvc_pkg::*;
26   import mgc_apb3_v1_0_pkg::*;
27   import alu_in_pkg::*;
28   import alu_in_pkg_hdl::*;
29   import alu_out_pkg::*;
30   import alu_out_pkg_hdl::*;
31   import alu_parameters_pkg::*;
32   import alu_env_pkg::*;
33   import qvip_agents_params_pkg::*;
34   import alu_reg_pkg::*;
35   `include "uvm_macros.svh"
36
37   // pragma uvmf custom package_imports_additional begin
38   // pragma uvmf custom package_imports_additional end
39
40   `include "src/alu_bench_sequence_base.svh"
41   `include "src/register_test_sequence.svh"
42   `include "src/example_derived_test_sequence.svh"
43
44   // pragma uvmf custom package_item_additional begin
45   // UVMF_CHANGE_ME : When adding new sequences to the src directory
46   //    be sure to add the sequence file here so that it will be
47   //    compiled as part of the sequence package.  Be sure to place
48   //    the new sequence after any base sequences of the new sequence.
49   `include "src/alu_random_sequence.svh"
50   `include "src/apb3_random_sequence.svh"
51   `include "src/apb3_alu_random_sequence.svh"
52   // pragma uvmf custom package_item_additional end
53
54 endpackage
```

Modified Code

Modify the bench virtual sequence

project_benches/alu/tb/sequences/src
alu_random_sequence.svh

- UVMF Generated Sequence

1. Add a new handle for the apb3_random_sequence and the apb3_alu_random_sequence.

2. Add the random APB sequences after the existing ALU sequences.

Creating a new bench virtual sequence

project_benches/alu/tb/sequences/src
alu_random_sequence.svh

```
28  typedef apb3_random_sequence #(.SLAVE_COUNT    (1),
29                                 .ADDRESS_WIDTH  (TEST_APB_ADDR_WIDTH),
30                                 .WDATA_WIDTH    (TEST_APB_WDATA_WIDTH),
31                                 .RDATA_WIDTH    (TEST_APB_RDATA_WIDTH) ) apb3_random_seq_t;
32  apb3_random_seq_t apb3_random_seq;
33
34  typedef apb3_alu_random_sequence #(.SLAVE_COUNT    (1),
35                                     .ADDRESS_WIDTH  (TEST_APB_ADDR_WIDTH),
36                                     .WDATA_WIDTH    (TEST_APB_WDATA_WIDTH),
37                                     .RDATA_WIDTH    (TEST_APB_RDATA_WIDTH) ) apb3_alu_random_seq_t;
38  apb3_alu_random_seq_t apb3_alu_random_seq;
```

1

```
46  virtual task body();
47
48    // Construct sequences here
49    alu_in_agent_random_seq = alu_in_agent_random_seq_t::type_id::create("alu_in_agent_random_seq");
50    alu_in_agent_reset_seq  = alu_in_agent_reset_seq_t::type_id::create("alu_in_agent_reset_seq");
51    apb3_random_seq         = apb3_random_seq_t::type_id::create("apb3_random_seq");
52    apb3_alu_random_seq     = apb3_alu_random_seq_t::type_id::create("apb3_alu_random_seq");
53
54
55
56    // Delay start of sequence until reset has ended and then wait a few clocks after that
57    alu_in_agent_config.wait_for_reset();
58    alu_in_agent_config.wait_for_num_clocks(10);
59
60    repeat (10) alu_in_agent_random_seq.start(alu_in_agent_sequencer);
61    alu_in_agent_reset_seq.start(alu_in_agent_sequencer);
62    repeat (5) alu_in_agent_random_seq.start(alu_in_agent_sequencer);
63
64    // Test the ALU memory
65    repeat (5) apb3_random_seq.start(uvm_test_top_environment_qvip_agents_env_apb_master_0_sqr);
66
67    // Test the ALU APB operand and operation interface
68    repeat (5) apb3_alu_random_seq.start(uvm_test_top_environment_qvip_agents_env_apb_master_0_sqr);
69
70    // UVMF_CHANGE_ME : Extend the simulation XXX number of clocks after
71    // the last sequence to allow for the last sequence item to flow
72    // through the design.
73    alu_in_agent_config.wait_for_num_clocks(50);
74
75  endtask
```

1

2

New Sequence Code

Simulating The New Test

- Go to the ***project_benches/alu/sim*** folder
- Windows Users
  - Edit run.do and modify the modify the last line where +UVM_TESTNAME specifies the test to run

```
32  vopt -32  +acc    hvl_top hdl_top   -o optimized_debug_top_tb
33  vsim -i  -32  -sv_seed random +UVM_TESTNAME=alu_random_test  +UVM_VERBOSITY=UVM_HIGH
34
```

- Linux Users
  - Execute 'make debug TEST_NAME=alu_random_test'

Simulating The New Test

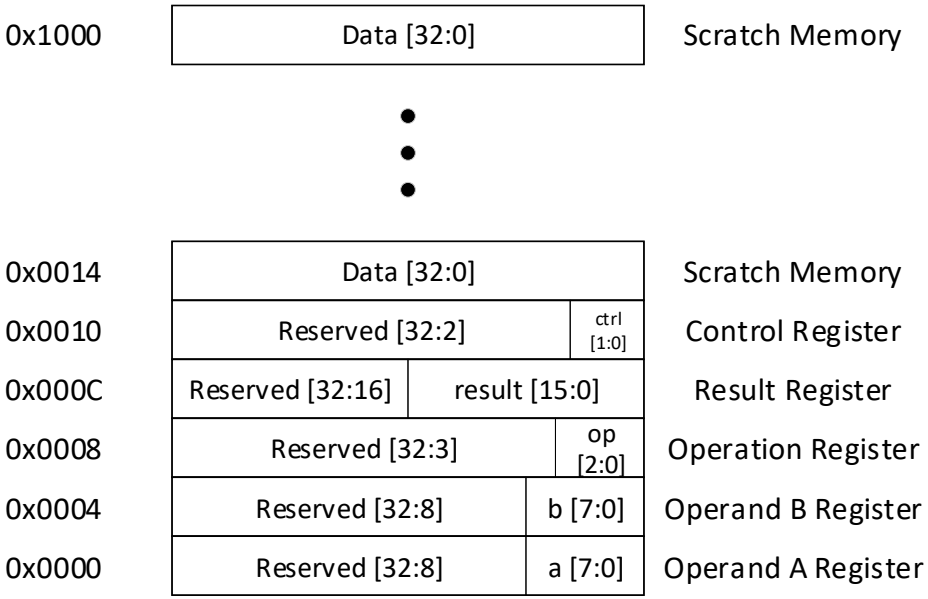
- Observe from the wave window
  1. No operations occur during the reset
  2. 10 random operations are then applied to the ALU
  3. A reset is then applied to the ALU
  4. 5 further random operations are then applied to the ALU
  5. APB write to Control Register, Operand A, Operand B, Operation, and then Control Register again.

Agenda

- ALU Overview
- Config Files Explained
- Compile and Simulate Generated Code
- Adding DUT Specific Functionality
- Generate and integrate the Register Model
- Add functional coverage

Memory Map

- The ALU contains 5 registers and a single scratch pad memory.
- We will use the information contained in this memory map to generate our uvm_reg compliant register model.



Register Definition

- Questa is bundled with the Register Assistant Utility (RUVM) which will automatically create your uvm_reg compliant register model from the CSV templates which you are about to fill in.
- Copy the example CSV files from

$QUESTA_HOME/RUVM_4.x/examples/uvm/csv

- Modify the *sw_regs.csv* to define the ALU registers as follows:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Register Name	Register Description	Register Address	Register Width	Register Access	Register Reset Value	Register Constraints	Register Custom Type	Field Name	Field Description	Field Offset	Field Width	Field Access	Field Reset Value	Field is Covered	Field is Reserved	Field is Volatile	Field Constraints
2	a_reg	Operand A Register	0x0	32	RW	0x0												
3	a_reg								rsvrd	Reserved	8	24	RO	0x0	FALSE	TRUE		
4	a_reg								a	Operand A	0	8	RW	0x0	FALSE			constraint op_a_c {a.value[7:0] < 5;}
5	b_reg	Operand B Register	0x04	32	RW	0x0												
6	b_reg								rsvrd	Reserved	8	24	RO	0x0	FALSE	TRUE		
7	b_reg								b	Operand B	0	8	RW	0x0	FALSE			constraint op_b_c {b.value[7:0] < 15;}
8	op_reg	Operation Register	0x08	32	RW	0x0												
9	op_reg								rsvrd	Reserved	3	29	RO	0x0	FALSE	TRUE		
10	op_reg								op	Operation	0	3	RW	0x0	FALSE			
11	result_reg	Result Register	0x0C	32	RO	0x0												
12	result_reg								rsvrd	Reserved	16	16	RO	0x0	FALSE	TRUE		
13	result_reg								rslt	Result	0	16	RO	0x0	FALSE			
14	ctrl_reg	Control Register	0x010	32	RW	0x0												
15	ctrl_reg								rsvrd	Reserved	2	30	RO	0x0	FALSE	TRUE		
16	ctrl_reg								mode	Switch from ALU interface to APB interface	1	1	RW	0x0	FALSE			
17	ctrl_reg								strt	Perform Operation	0	1	RW	0x0	FALSE			

Memory Definition

- Modify the *sw_mems.csv* to define the ALU memory as follows:

	A	B	C	D	E	F	G
1	Memory Name	Memory Description	Memory Address	Memory Width	Memory Range	Memory Access	Memory Constraints
2	scratch_mem	Memory	0x14	32	4076	RW	

Block Definition

- Modify the *sw_blocks.csv* to define the ALU register and memory architecture as follows:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	Block Name	Block Description	Block Coverage	Block Backdoor	Block Component Name	Block Instance Name	Block Instance Type	Block Instance Description	Block Instance Dimension	Block Instance Backdoor	Block Instance No Reg Tests	Block Instance No Reg Access Test	Block Instance No Reg Shared Access Test	Block Instance No Reg Bit Bash Test	Block Instance No Reg HW Reset Test	Block Instance No Mem Tests	Block Instance No Mem Access Test	Block Instance No Mem Shared Access Test	Block Instance No Mem Walk Test	Project Extra Imports
2	alu_reg_model	Top block for the ALU design		hdl_top.DUT	alu_block	alu	block													
3																				
4	alu_block	ALU	UVM_CVR_ADDR_MAP		a_reg	a_reg_reg		Operand A												
5	alu_block				b_reg	b_reg_reg		Operand B												
6	alu_block				op_reg	op_reg_reg		Operation												
7	alu_block				result_reg	result_reg_reg		Result				1								
8	alu_block				ctrl_reg	ctrl_reg_reg		Control												
9	alu_block				scratch_mem	scratch_mem_mem		MEM instance												

Block Map Definition

- Modify the *sw_maps.csv* to define how the ALU uvm_reg blocks map to the register and memory instances as follows:

	A	B	C	D	E	F	G	H
1	Block Name	BlockMap Name	BlockMap Description	BlockMap is default	BlockMap Instance Name	BlockMap Instance Address	BlockMap Instance Access	BlockMap Address Offset
2	alu_reg_model	apb_map	ALU APB top block map	TRUE	alu.apb_map	0x0		
3								
4	alu_block	apb_map	ALU sub block map	TRUE	a_reg_reg	0x00		
5	alu_block	apb_map			b_reg_reg	0x04		
6	alu_block	apb_map			op_reg_reg	0x08		
7	alu_block	apb_map			result_reg_reg	0x0C		
8	alu_block	apb_map			ctrl_reg_reg	0x10		
9	alu_block	apb_map			scratch_mem_mem	0x14		

```
57        register_model :
58          use_adapter: "True"
59          use_explicit_prediction: "True"
60          maps:
61            - { name: "apb_map", interface: "alu_in_agent" }
```

Note that the map name you used in the YAML must match the Block Map Name in the maps.csv file.

Generate your Register Model

- Now run the *vreguvm* utility to generate your register model.

$QUESTA_HOME/RUVM_2020.1/vreguvm -uvmout alu_reg_pkg.sv -csvin alu_regs.csv alu_mems.csv alu_blocks.csv alu_maps.csv -block alu_reg_model

- Your register model will be named *alu_reg_pkg.sv* and will be located in the directory where you launched *vreguvm*.
- Now copy your register model to *verification_ip/environment_packages/alu_env_pkg/src/registers* and overwrite the *alu_reg_pkg.sv* file that was already there. This file was an empty model which was generated as a place holder.

Connect the APB3 QVIP Register Adapter to the Register Model

- The YAML connected the register model to the alu_in register adapter by default.
- We need to disconnect the alu_in register adapter and connect the APB3 QVIP register adapter.
- Edit the ALU environment file as noted in the following slides:
  verification_ip/environment_packages/alu_env_pkg/src/alu_environment.svh

Connect the APB3 QVIP Register Adapter to the Register Model

- The YAML connected the register model to the alu_in register adapter by default.

```
75    // Instantiate register model adapter and predictor
76    typedef alu_in2reg_adapter#(.ALU_IN_OP_WIDTH(ALU_IN_OP_WIDTH))    reg_adapter_t;
77    reg_adapter_t    reg_adapter;
78    typedef uvm_reg_predictor #(alu_in_transaction#(.ALU_IN_OP_WIDTH(ALU_IN_OP_WIDTH))) reg_predictor_t;
79    reg_predictor_t    reg_predictor;
```

Replace the ALU register adapter and the ALU register predictor with the APB3 QVIP register adapter and APB3 QVIP register predictor. We also need an APB3 QVIP Transaction.

```
75  // Instantiate register model adapter and predictor
76  typedef apb3_host_apb3_transaction #(1 ,
77                                       APB_ADDR_WIDTH ,
78                                       APB_WDATA_WIDTH ,
79                                       APB_RDATA_WIDTH) apb3_host_apb3_transaction_t;
80
81  typedef reg2apb_adapter #(apb3_host_apb3_transaction_t,
82                            1,
83                            APB_ADDR_WIDTH,
84                            APB_WDATA_WIDTH,
85                            APB_RDATA_WIDTH) reg2apb_adapter_t;
86
87  typedef apb_reg_predictor #(apb3_host_apb3_transaction_t,
88                              1,
89                              APB_ADDR_WIDTH,
90                              APB_WDATA_WIDTH,
91                              APB_RDATA_WIDTH) apb_reg_predictor_t;
92
93  reg2apb_adapter_t    reg_adapter;
94  apb_reg_predictor_t  reg_predictor;
```

Connect the APB3 QVIP Register Adapter to the Register Model


```
103   // Build register model predictor if prediction is enabled
104   if (configuration.enable_reg_prediction) begin
105     reg_predictor = reg_predictor_t::type_id::create("reg_predictor", this);
106   end
```

```
128   // Create register model adapter if required
129   if (configuration.enable_reg_prediction ||
130       configuration.enable_reg_adaptation)
131     reg_adapter = reg_adapter_t::type_id::create("reg_adapter");
132   // Set sequencer and adapter in register model map
133   if (configuration.enable_reg_adaptation)
134     configuration.reg_model.apb_map.set_sequencer(alu_in_agent.sequencer, reg_adapter);
135   // Set map and adapter handles within uvm predictor
136   if (configuration.enable_reg_prediction) begin
137     reg_predictor.map     = configuration.reg_model.apb_map;
138     reg_predictor.adapter = reg_adapter;
139     // temp alu_in_agent.monitored_ap.connect(reg_predictor.bus_in);
140   end
```

Replace the ALU register adapter and the ALU register predictor with the APB3 QVIP register adapter and APB3 QVIP register predictor. We also need an APB3 QVIP Transaction.

```
123   // Build register model predictor if prediction is enabled
124   if (configuration.enable_reg_prediction) begin
125     reg_predictor = apb_reg_predictor_t::type_id::create("reg_predictor", this);
126   end
```

```
152   // Create register model adapter if required
153   if (configuration.enable_reg_prediction ||
154       configuration.enable_reg_adaptation)
155     reg_adapter = reg2apb_adapter_t::type_id::create("reg_adapter");
156   // Set sequencer and adapter in register model map
157   if (configuration.enable_reg_adaptation)
158     configuration.reg_model.apb_map.set_sequencer(qvip_agents_env.apb_master_0.m_sequencer, reg_adapter);
159     configuration.reg_model.apb_map.set_auto_predict(0);
160   // Set map and adapter handles within uvm predictor
161   if (configuration.enable_reg_prediction) begin
162     reg_predictor.map     = configuration.reg_model.apb_map;
163     reg_predictor.adapter = reg_adapter;
164     qvip_agents_env.apb_master_0.ap["trans_ap"].connect(reg_predictor.bus_item_export);
165   end
```

Does your Register Model include coverage?

- If you enabled 'block coverage' and/or 'field coverage' in your CSV files then you need to include the coverage models in your environment.
- Add the following code to your ***project_benches/aiu/tb/tests/src/test_top.svh***.

```
69  virtual function void build_phase(uvm_phase phase);
70
71    // Turn on coverage for the Register Model
72    uvm_reg::include_coverage("*", UVM_CVR_ALL);
73
74    super.build_phase(phase);
75    configuration.initialize(BLOCK, "uvm_test_top.environment", interface_names, null, interface_activities);
76
77  endfunction
```

The UVMF Register Sequence
project_benches/alu/tb/sequences/src
register_test_sequence.svh

- You may want to disable some of the default register model tests as shown below.

```
49      // Reset the register model
50      reg_model.reset();
51      // Identify the register model to test
52      uvm_register_test_seq.model = reg_model;
53      // Perform the register test
54      // Disable particular tests in sequence by commenting options below
55      uvm_register_test_seq.tests = {
56      // pragma uvmf custom register_test_operation begin
57                                      UVM_DO_REG_HW_RESET       |
58                                      UVM_DO_REG_BIT_BASH       |
59                                      UVM_DO_REG_ACCESS         |
60                                      UVM_DO_MEM_ACCESS         |
61                                      UVM_DO_SHARED_ACCESS      |
62                                      UVM_DO_MEM_WALK           |
63                                      UVM_DO_ALL_REG_MEM_TESTS
64      // pragma uvmf custom register_test_operation end
65                                    };
66
67      uvm_register_test_seq.start(null);
```

Original Code

Modified Code

```
49      // Reset the register model
50      reg_model.reset();
51      // Identify the register model to test
52      uvm_register_test_seq.model = reg_model;
53      // Perform the register test
54      // Disable particular tests in sequence by commenting options below
55      uvm_register_test_seq.tests = {
56      // pragma uvmf custom register_test_operation begin
57                                      UVM_DO_REG_HW_RESET       |
58                                    // UVM_DO_REG_BIT_BASH       |
59                                    // UVM_DO_REG_ACCESS         |
60                                    // UVM_DO_MEM_ACCESS         |
61                                    // UVM_DO_SHARED_ACCESS      |
62                                       UVM_DO_MEM_WALK           |
63                                    // UVM_DO_ALL_REG_MEM_TESTS
64      // pragma uvmf custom register_test_operation end
65                                    };
66
67      uvm_register_test_seq.start(null);
```

The UVMF Register Test
project_benches/alu/tb/tests/src
register_test.svh

- You will need to disable the UVMF scoreboard for the register model test since they do not follow the ALU protocol and will fail to generate transactions for the scoreboard to compare.

Original Code

```
35   virtual function void end_of_elaboration_phase(uvm_phase phase);
36     super.end_of_elaboration_phase(phase);
37     // pragma uvmf custom register_test_scoreboard_control begin
38
39     // These UVMF scoreboards may need to be disabled for the register test.
40
41     // environment.alu_sb.disable_scoreboard();
42     // environment.alu_sb.disable_end_of_test_activity_check();
43
44     // pragma uvmf custom register_test_scoreboard_control end
45   endfunction
```

Modified Code

```
35   virtual function void end_of_elaboration_phase(uvm_phase phase);
36     super.end_of_elaboration_phase(phase);
37     // pragma uvmf custom register_test_scoreboard_control begin
38
39     // These UVMF scoreboards may need to be disabled for the register test.
40
41     environment.alu_sb.disable_scoreboard();
42     environment.alu_sb.disable_end_of_test_activity_check();
43
44     // pragma uvmf custom register_test_scoreboard_control end
45   endfunction
```

Running The Register Test

- Go to the ***project_benches/alu/sim*** folder
- Windows Users
  - Edit run.do and modify the modify the last line where +UVM_TESTNAME specifies the test to run

```
9 quietly set cmd [format "vsim -i -sv_seed random +UVM_TESTNAME=register_test
```

- Linux Users
  - Execute 'make debug TEST_NAME=register_test'

Agenda

- ALU Overview
- Config Files Explained
- Compile and Simulate Generated Code
- Adding DUT Specific Functionality
- Generate and integrate the Register Model
- Add functional coverage

Add Functional Coverage to your ALU Agent

verification_ip/interface_packages/alu_in_pkg/src
alu_in_transaction_coverage.svh

- The UVMF YAML code generators created an alu_in_transaction_coverage class with a default cover group where you can easily add coverage bins, crosses, and exclusions as needed.

```
33  covergroup alu_in_transaction_cg;
34    // pragma uvmf custom covergroup begin
35    // UVMF_CHANGE_ME : Add coverage bins, crosses, exclusions, etc. according to coverage needs.
36    option.auto_bin_max=1024;
37    option.per_instance=1;
38    op: coverpoint coverage_trans.op;
39    a: coverpoint coverage_trans.a;
40    b: coverpoint coverage_trans.b;
41    // pragma uvmf custom covergroup end
42  endgroup
```

```
33  covergroup alu_in_transaction_cg;
34    // pragma uvmf custom covergroup begin
35    // UVMF_CHANGE_ME : Add coverage bins, crosses, exclusions, etc. according to coverage needs.
36    option.auto_bin_max=1024;
37    option.per_instance=1;
38    op: coverpoint coverage_trans.op { bins noop   = { 3'b000 };
39                                       bins addop  = { 3'b001 };
40                                       bins andop  = { 3'b010 };
41                                       bins xorop  = { 3'b011 };
42                                       bins mulop  = { 3'b100 };
43                                       bins rstop  = { 3'b111 };
44                                       bins others = default;
45                                     }
46    a: coverpoint coverage_trans.a { bins low    = {   [0:63] };
47                                     bins mid    = { [64:126] };
48                                     bins high   = {      127 };
49                                     bins others = default;
50                                   }
51    b: coverpoint coverage_trans.b { bins low    = {   [0:63] };
52                                     bins mid    = { [64:126] };
53                                     bins high   = {      127 };
54                                     bins others = default;
55                                   }
56    // pragma uvmf custom covergroup end
57  endgroup
```

Modified Code

Create your Test Plan

- If you plan to use Microsoft Excel to write your test plan.
  - If you have not done so, you can install the Questa Excel Add in to assist you with creating test plans.
  - Details on how to install the Questa Excel Add in can be found under your Questa installation at the following path:

    $QUESTA_HOME/vm_src

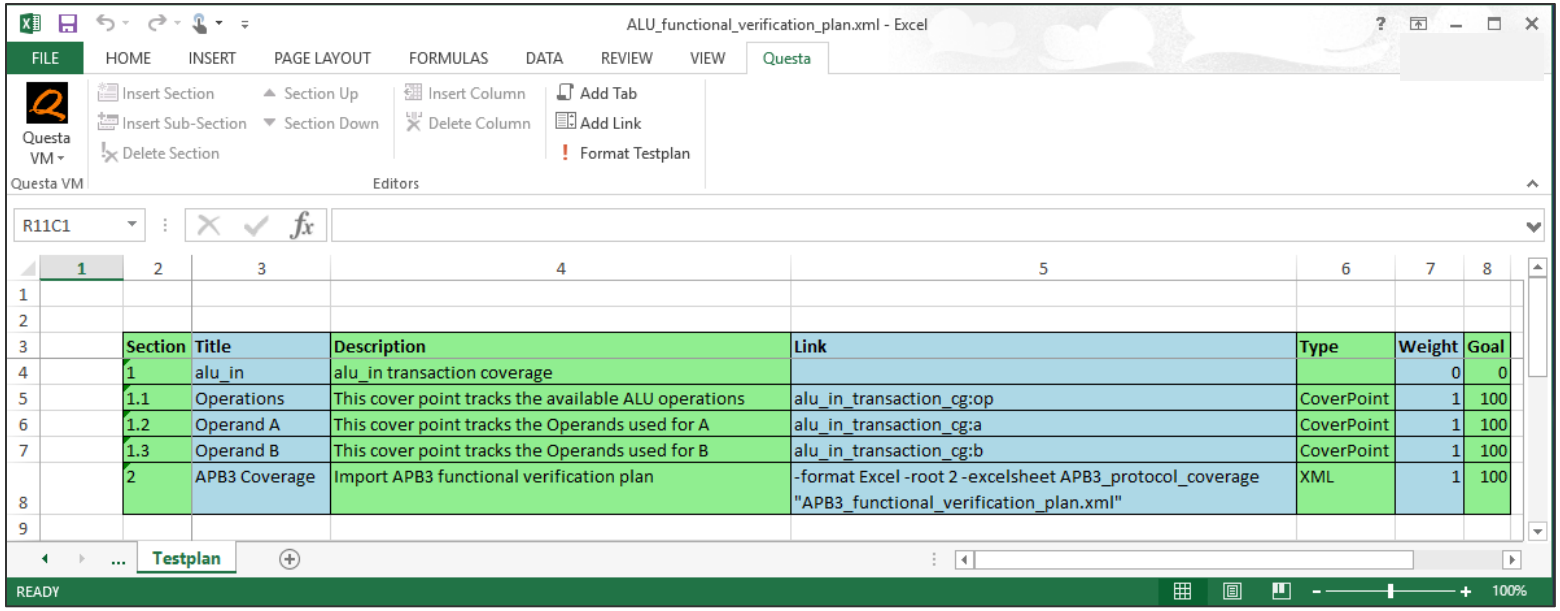
- Or, if you plan to use Microsoft Word to write your test plan.
  - Jump to slide 116 for an example of the test plan written in Microsoft Word.
- Otherwise please refer to the Questa Verification Management User's Manual found in your Questa installation at the following path:

  $QUESTA_HOME/docs/pdfdocs/questa_sim_vm.pdf

Add Functional Coverage to your Test Plan

If you are using Microsoft Excel

- Once you have the Questa VM add in installed then you can easily create a test plan.
- The example below shows the cover points we added to the alu_in_transaction_coverage.svh class on slide 113.
- We are also adding a link to the APB3 QVIP test plan so we can track the functional coverage of the APB protocol.



Section	Title	Description	Link	Type	Weight	Goal
1	alu_in	alu_in transaction coverage			0	0
1.1	Operations	This cover point tracks the available ALU operations	alu_in_transaction_cg:op	CoverPoint	1	100
1.2	Operand A	This cover point tracks the Operands used for A	alu_in_transaction_cg:a	CoverPoint	1	100
1.3	Operand B	This cover point tracks the Operands used for B	alu_in_transaction_cg:b	CoverPoint	1	100
2	APB3 Coverage	Import APB3 functional verification plan	-format Excel -root 2 -excelsheet APB3_protocol_coverage "APB3_functional_verification_plan.xml"	XML	1	100

Add Functional Coverage to your Test Plan

If you are using Microsoft Excel

- Modify lines 72 and 73 of your default.rmdb file to enable your Excel Test Plan options.

```
66  <runnable name="run" type="task" foreach="{%TESTCASES_FOR_BUILD%}" index="testname">
67    <parameters>
68      <parameter name="TESTCASES_FOR_BUILD" type="tcl">[GetTestcases {%build%} {%COLLAPSE%}]</parameter>
69      <parameter name="UVM_TESTNAME" type="tcl">[lindex [split {%testname%} "-"] 1]</parameter>
70      <parameter name="ucdbfile">{%testname%}.ucdb</parameter>
71      <parameter name="seed" type="tcl">[lindex [split {%testname%} "-"] 3]</parameter>
72      <parameter name="tplanoptions">-format Excel -datafields "Section,Title,Description,Link,Type,Weight,Goal"</parameter>
73      <parameter name="tplanfile">{%RMDBDIR%}/ALU_Functional_verification_plan.xml</parameter>
74      <parameter name="TIMEOUT">3600</parameter>
75      <parameter name="DEBUGMODE">0</parameter>
76      <parameter name="EXTRARUNCMD" type="tcl">[GetRunCmd {%build%}]</parameter>
77      <parameter name="MVCHOME" type="tcl">[if {[FindMVCHome "{%RMDBDIR%}/Makefile"]} { return "-mvchome $::env(QUESTA_MVC_HOME)" }]</parameter>
78      <parameter name="UVM_VERBOSITY" type="tcl">[ if { {%DEBUGMODE%} } { return "UVM_HIGH" } else { return "UVM_LOW" }]</parameter>
79      <parameter name="TOP">optimized_batch_top_tb</parameter>
80      <parameter name="CODE_COVERAGE" type="tcl">[ if { {%CODE_COVERAGE_ENABLE%} } { return "-coverage" } else { return "" }]</parameter>
81      <parameter name="VIS_ARGS" type="tcl">[ if { {%USE_VIS%} &amp;&amp; { {%DEBUGMODE%} || {%DUMPWAVES%} } } { return "-qwavedb=+{%VIS_DUMP_OPTIONS%}" } else { return "" }]</parameter>
82    </parameters>
```

Add Functional Coverage to your Test Plan

If you are using Microsoft Word

Test Plan For ALU

1 alu_in (1)
WEIGHT: 0
GOAL: 0
DESCRIPTION:
alu_in transaction coverage.

1.1 Operations (2)
WEIGHT: 1
GOAL: 100
TYPE: CoverPoint
LINK: alu_in_transaction:op;
DESCRIPTION:
This cover point tracks the available ALU operations.

1.2 Operand A
WEIGHT: 1
GOAL: 100
TYPE: CoverPoint (3)
LINK: alu_in_transaction:a;
DESCRIPTION:
This cover point tracks the Operands used for A.

1.3 Operand B
WEIGHT: 1
GOAL: 100
TYPE: CoverPoint
LINK: alu_in_transaction:b;
DESCRIPTION:
This cover point tracks the Operands used for B.

1.4 APB3 Coverage
WEIGHT: 1
GOAL: 100
TYPE: XML
LINK: -format Excel -root 2 -excelsheet APB3_protocol_coverage "APB3_functional_verification_plan.xml"
DESCRIPTION:
Import APB3 functional verification plan.

1. Test Plan sections should use 'Heading1' style.
2. Test Plan sub-sections should use the next highest heading style i.e. 'Heading2'.
3. The paragraphs following the test plan sections should be 'Normal' style and should start with one of the following labels in no particular order:
   - DESCRIPTION:
   - GOAL:
   - LINK:
   - TYPE:
   - WEIGHT:

- Please refer to the 'Guidelines for Word Documents' in the Questa SIM Verification Manager User's Manual for more details.
- To export to XML first select 'File -> Save As' and then select 'Word 2003 XML Document (*.xml)' as the 'Save as type'.

NOTE: You must disable automatic correction of straight quotes to smart quotes and hyphens with dash. See KB article for further details.
https://support.mentor.com/en/knowledge-base/MG603966

Add Functional Coverage to your Test Plan

If you are using Microsoft Word

- Modify lines 72 and 73 of your default.rmdb file to enable your Word Test Plan options.

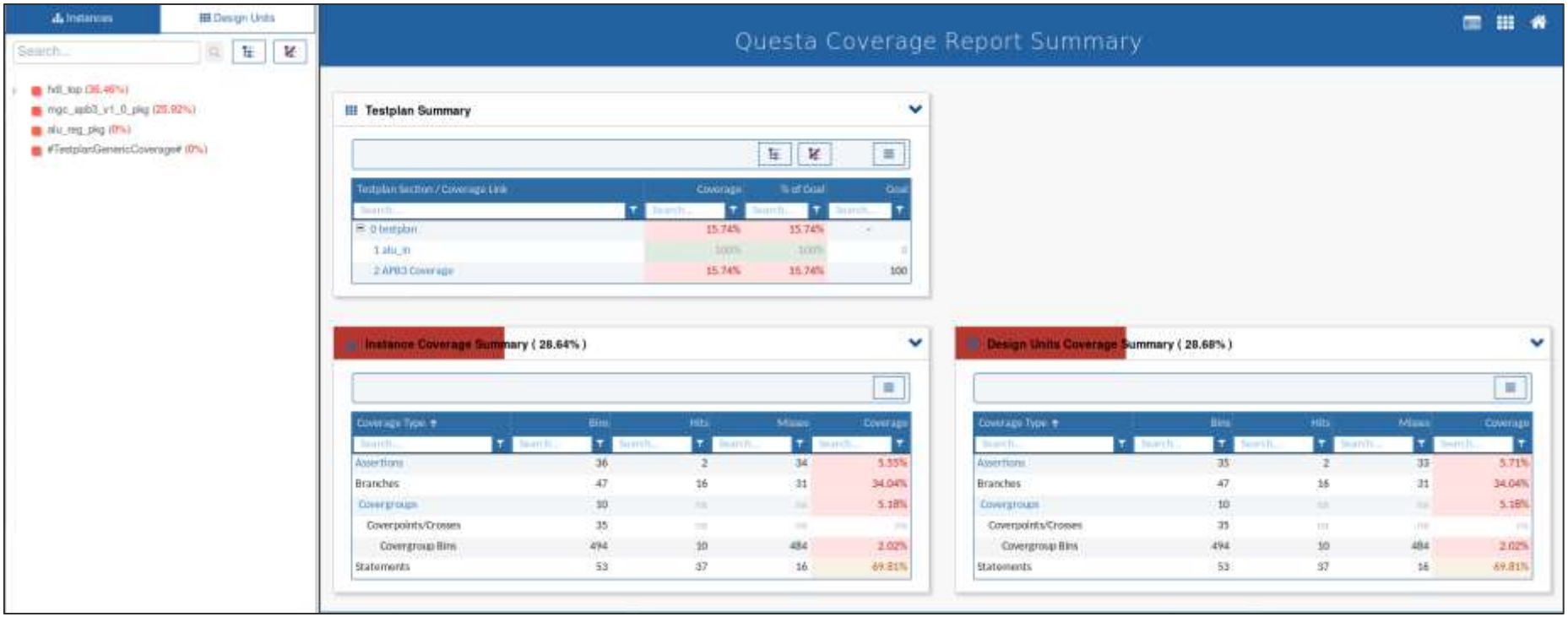
```
66  <runnable name="run" type="task" foreach="{%TESTCASES_FOR_BUILD%}" index="testname">
67    <parameters>
68      <parameter name="TESTCASES_FOR_BUILD" type="tcl">[GetTestcases {%build%} {%COLLAPSE%}]</parameter>
69      <parameter name="UVM_TESTNAME" type="tcl">[lindex [split {%testname%} "-"] 1]</parameter>
70      <parameter name="ucdbfile">{%testname%}.ucdb</parameter>
71      <parameter name="seed" type="tcl">[lindex [split {%testname%} "-"] 3]</parameter>
72      <parameter name="tplanoptions">-format Word -datalabels "WEIGHT,GOAL,TYPE,LINK,DESCRIPTION"</parameter>
73      <parameter name="tplanfile">{%RMDBDIR%}/ALU_functional_verification_plan_word.xml</parameter>
74      <parameter name="TIMEOUT">3600</parameter>
75      <parameter name="DEBUGMODE">0</parameter>
76      <parameter name="EXTRARUNCMD" type="tcl">[GetRunCmd {%build%}]</parameter>
77      <parameter name="MVCHOME" type="tcl">[if {[FindMVCHome "{%RMDBDIR%}/Makefile"]} { return "-mvchome $::env(QUESTA_MVC_HOME)" }]</parameter>
78      <parameter name="UVM_VERBOSITY" type="tcl">[ if { {%DEBUGMODE%} } { return "UVM_HIGH" } else { return "UVM_LOW" }]</parameter>
79      <parameter name="TOP">optimized_batch_top_tb</parameter>
80      <parameter name="CODE_COVERAGE" type="tcl">[ if { {%CODE_COVERAGE_ENABLE%} } { return "-coverage" } else { return "" }]</parameter>
81      <parameter name="VIS_ARGS" type="tcl">[ if { {%USE_VIS%} &amp;&amp; ( {%DEBUGMODE%} || {%DUMP_WAVES%} ) } { return "-qwavedb=({%VIS_DUMP_OPTIONS%})" } else { return "" }]</parameter>
82    </parameters>
```

Collect your coverage

- You can use the Questa Verification Run Manager (VRM) to manage your simulations to collect your coverage. VRM also automatically merges all passing tests and generates your coverage reports.
- To run VRM make sure you are in *uvmf_template_output/project_benches/alu/sim* and type *vrun –GCODE_COVERAGE_ENABLE=1*.

View your coverage

- To view your coverage report make sure you are in *uvmf_template_output/project_benches/alu/sim* and type *firefox VRMDATA/top/report/coverage_report/index.html &*.



www.mentor.com