ALU
UVM Framework
Step By Step Guide

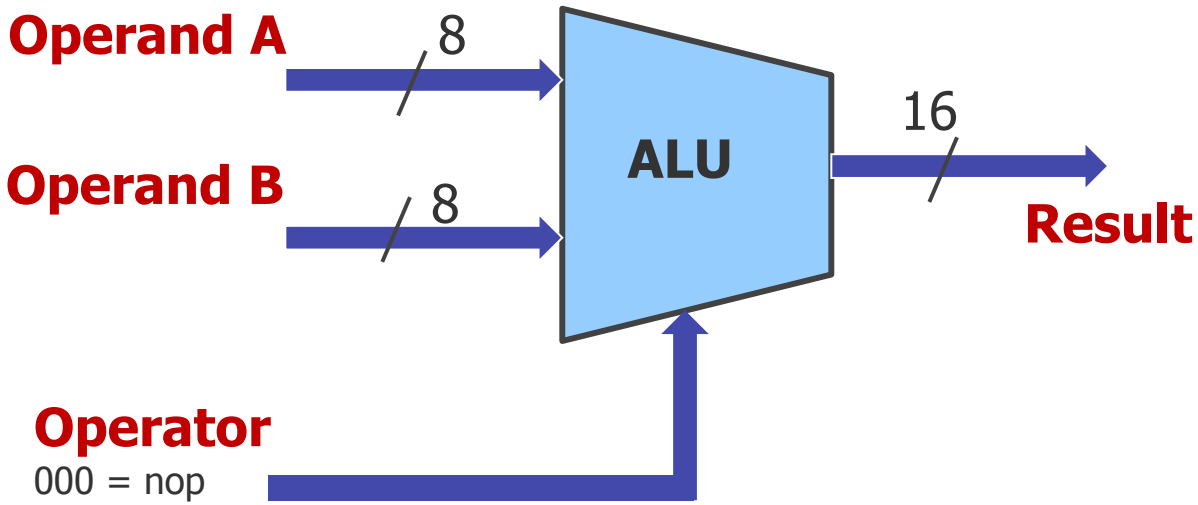
August 2017

Mentor®
A Siemens Business

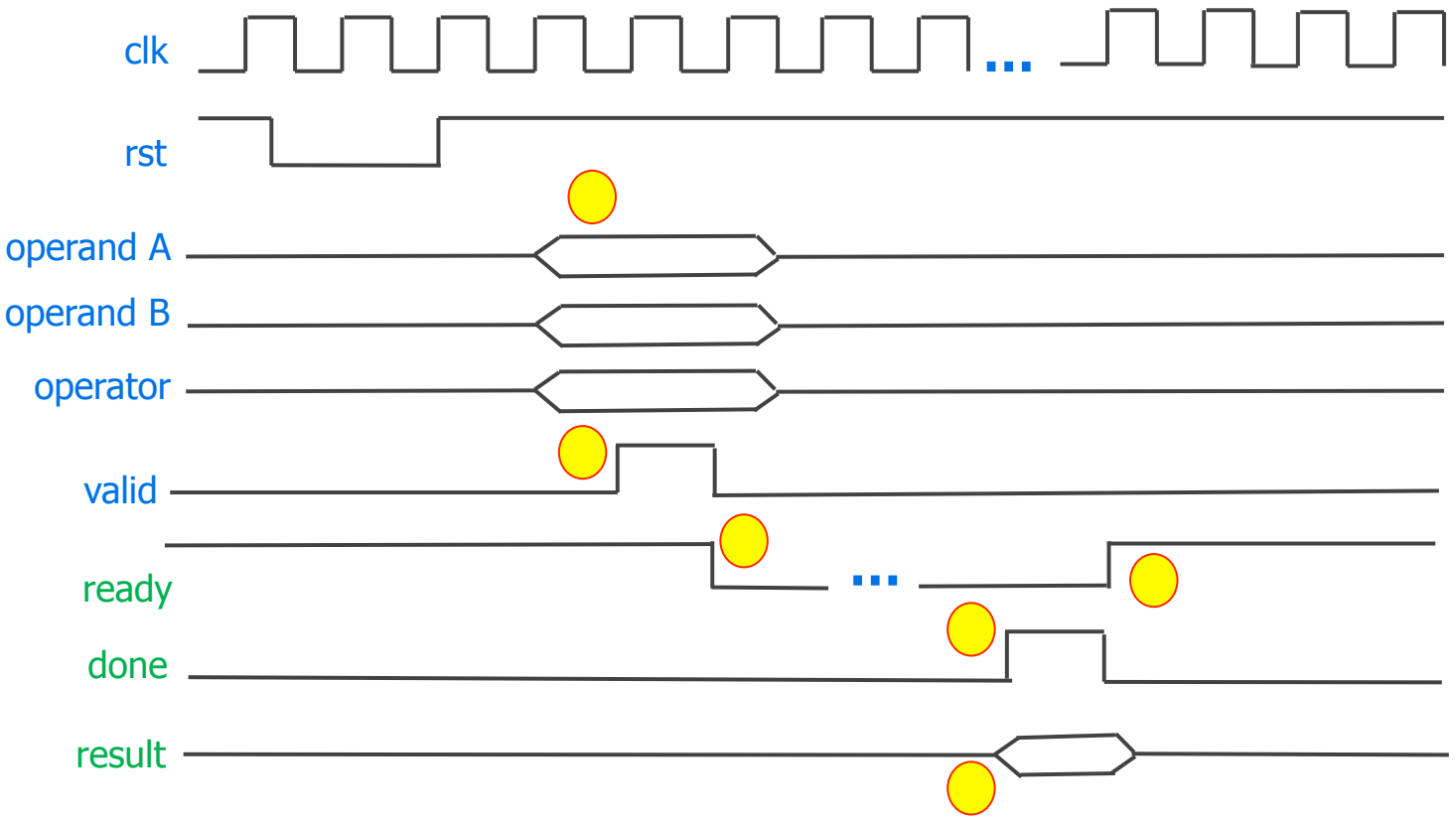
Agenda

- ALU Overview
- Config Files Explained
- Compile and Simulate Generated Code
- Adding DUT Specific Functionality

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

Python 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 each interface

- **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 specifying the Alu result

Interface Config File
alu_in_if_config.py

- **import uvmf_gen**
  - Template code that is required.
  - Imports Python code needed to interpret the rest of the config file
  - Requires $PYTHONPATH to be set as uvmf_gen.py file resides in that directory

- **intf = uvmf_gen.InterfaceClass('alu_in')**
  - The argument to the Interface Class function ('alu_in') is the name of the interface.
  - 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  #! /usr/bin/env python
2
3  import uvmf_gen
4
5  ## The input to this call is the name of the desired interface
6  intf = uvmf_gen.InterfaceClass('alu_in')
```

Interface Config File
alu_in_if_config.py

- **intf.inFactReady = False**
  - Disables generation of additional code for the Questa InFact tool
  - For 3.6f only, code is generated by default so need to disable it
  - From 3.6g onwards, the default will change to opt-in rather than opt-out

- **intf = addParamDef('ALU_IN_OP_WIDTH', 'int', '8')**
  - 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

```
 8  ## Disable generation of InFact specific files [required for 3.6f only]
 9  intf.inFactReady = False
10
11  ## Specify parameters for this interface package.
12  ## These parameters can be used when defining signal and variable sizes.
13  # addHdlParamDef(<name>,<type>,<value>)
14  intf.addParamDef('ALU_IN_OP_WIDTH','int','8')
```

Interface Config File
alu_in_if_config.py

- **intf.clock = 'clk'**
  - Specifies the interface clock signal
- **intf.reset = 'rst'**
  - Specifies the interface reset signal
- **intf.resetAssertionLevel = False**
  - Specifies the polarity of the active reset
  - False = Active Low; True = Active High

```
16  ## Specify the clock and reset signal for the interface
17  intf.clock = 'clk'
18  intf.reset = 'rst'
19  intf.resetAssertionLevel = 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_in_if_config.py

- **intf.addPort('alu_rst',1,'output)**
  - The addPort function is required for each signal in the interface.
  - The arguments to the addPort identify the name, width and direction of the signal.
  - The options for the direction are input, output and inout.

```
21  ## Specify the ports associated with this interface.
22  ## The direction is from the perspective of the test bench as an INITIATOR on the bus.
23  ##   addPort(<name>,<width>,[input|output|inout])
24  intf.addPort('alu_rst',1,'output')
25  intf.addPort('ready',1,'input')
26  intf.addPort('valid',1,'output')
27  intf.addPort('op',3,'output')
28  intf.addPort('a','ALU_IN_OP_WIDTH','output')
29  intf.addPort('b','ALU_IN_OP_WIDTH','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 intf.addParamDef directive

Interface Config File
alu_in_if_config.py

- **intf.addHdlTypedef(‘alu_in_op_t’, ‘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}’)**

  — This directive will generate an enumerated typedef for the alu operations.
  — Note that it is necessary to insert backslashes in front of the enumeration values to prevent them being processed by the Python.

```
31  ## Specify typedef for inclusion in typedefs_hdl file
32  # addHdlTypedef(<name>,<type>)
33  intf.addHdlTypedef('alu_in_op_t','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_in_if_config.py

- **intf.addTransvar('op','alu_in_op_t',israndom=True,Iscompare=True)**
  - 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 addTransVar 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.

```
36  ## Specify transaction variables for the interface.
37  ##   addTransVar(<name>,<type>)
38  ##     optionally can specify if this variable may be specified as 'rand'
39  ##     optionally can specify if this variable may be specified as used in do_compare()
40  intf.addTransVar('op','alu_in_op_t',israndom=True,iscompare=True)
41  intf.addTransVar('a','bit [ALU_IN_OP_WIDTH-1:0]',israndom=True,iscompare=True)
42  intf.addTransVar('b','bit [ALU_IN_OP_WIDTH-1:0]',israndom=True,iscompare=True)
```

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 for addTransVar
- If you need an unconstrained array, declare a fixed array in config file and modify generated code

Interface Config File
alu_in_if_config.py

- **intf.create()**
  - This tells the python generator to create all of the interface source files for the **alu_in** agent.

```
58  ## This will prompt the creation of all interface files in their specified
59  ##  locations
60  intf.create()
```

Generated UVMF Code
alu_in_if

- **Generating the Interface Code**
  - Run python on the alu_in_if_config.py file

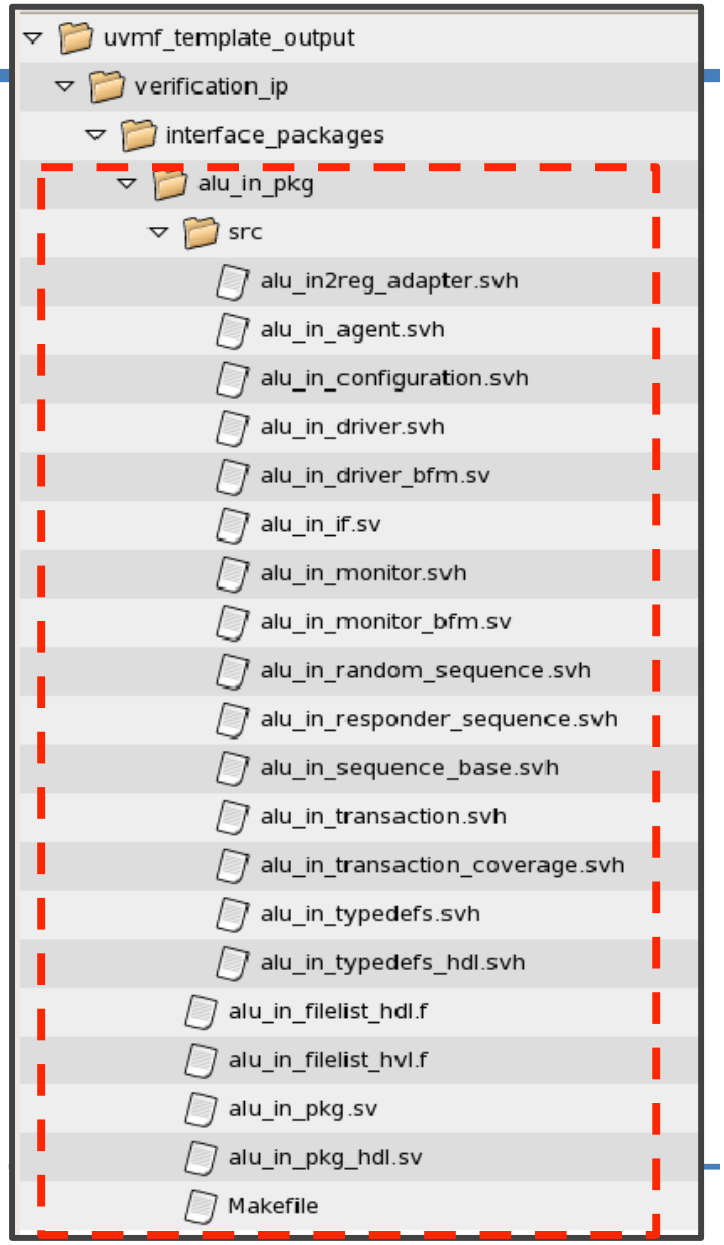
    python alu_in_if_config.py

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

```
1 @set UVMF_HOME=C:/MentorTools/questasim_10.6b/examples/UVM_Framework/UVMF_3.6f
2 @set PYTHONPATH=%UVMF_HOME%/templates/python
3
4 python alu_in_if_config.py
5 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

- **intf = addParamDef('ALU_IN_OP_WIDTH', 'int', '8')**
  - 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
```

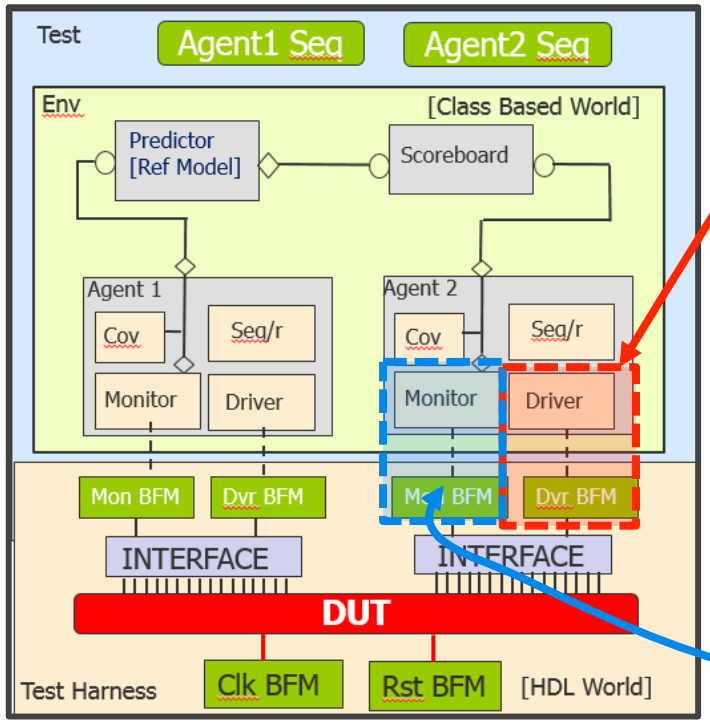
```
class alu_in_configuration  #(
      int ALU_IN_OP_WIDTH = 8
      ) extends uvmf_parameterized_agent_configuration_base
```

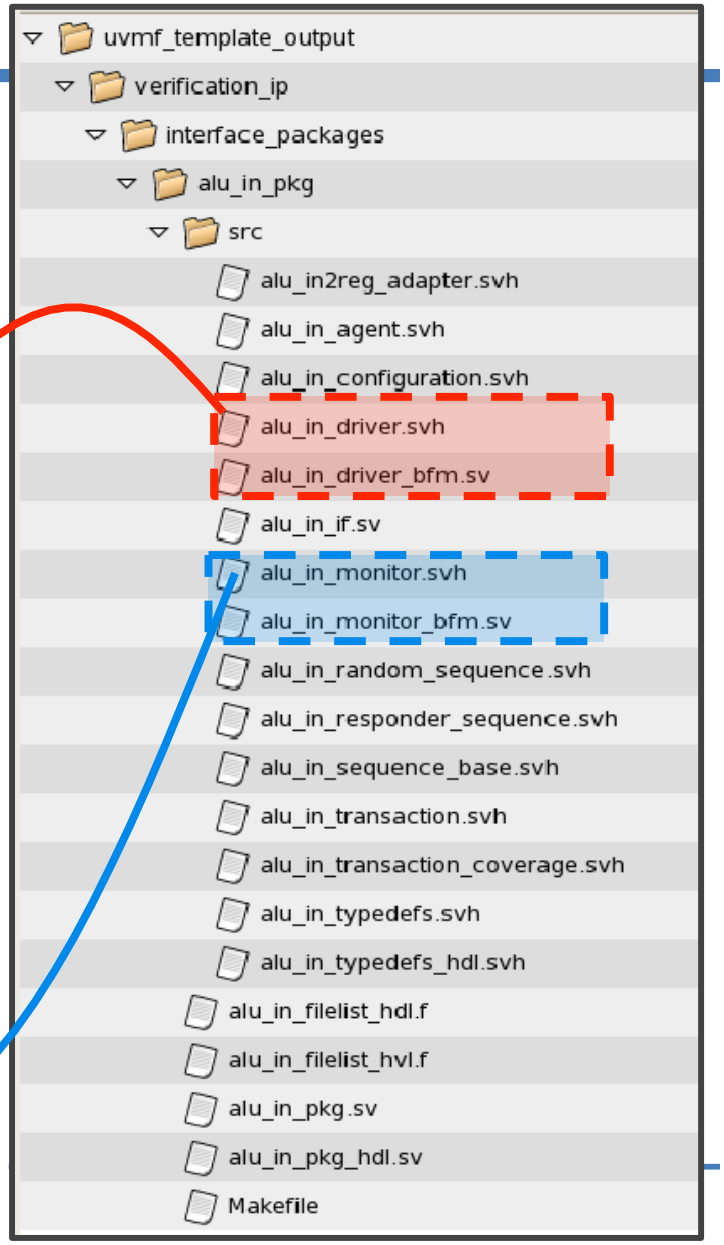
```
class alu_in_random_sequence  #(
      int ALU_IN_OP_WIDTH = 8
      )
```

```
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

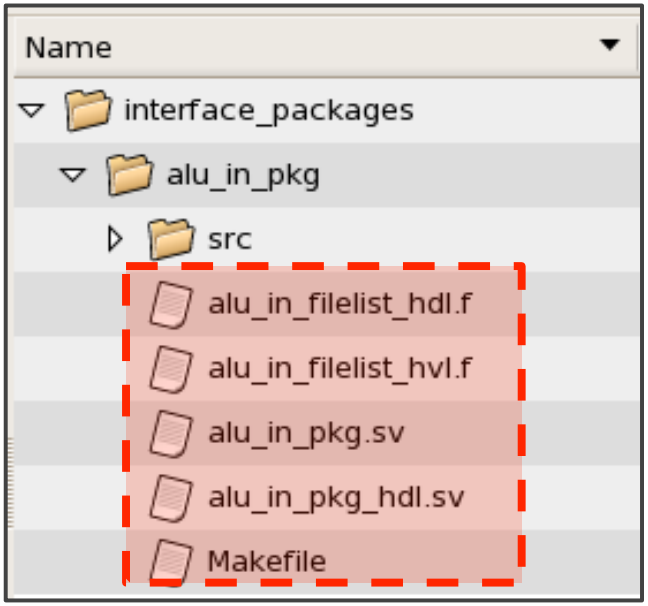
- **UVMF Transactors**
  - alu_in_driver & alu_in_monitor are class based and will be instantiated inside the agent class
  - alu_in_driver_bfm & alu_in_monitor_bfm are interfaces & will be instantiated in the top level testbench module (hdl_top)





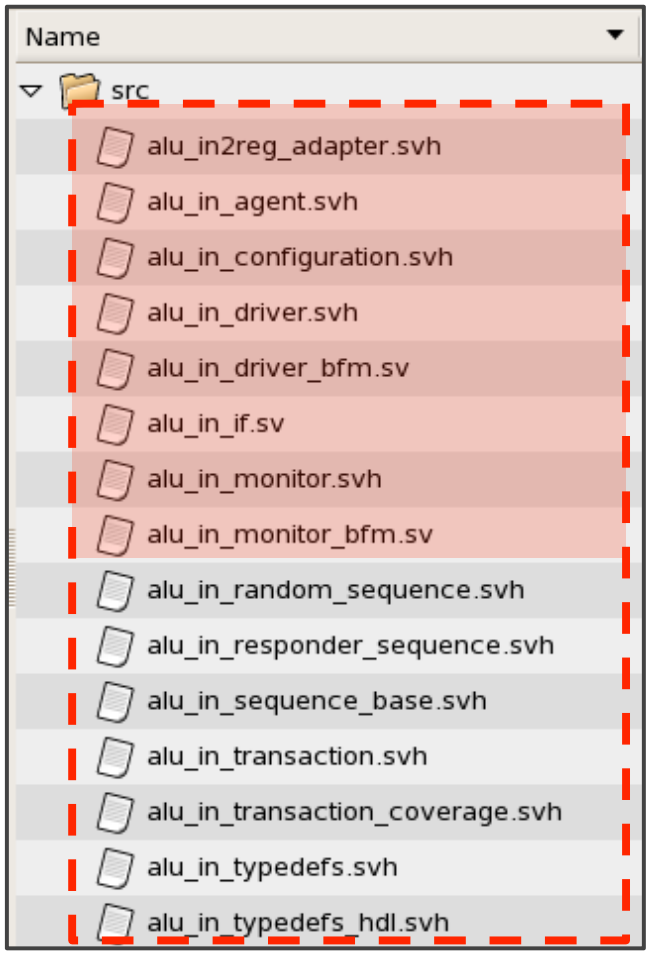
Generated UVMF Code
alu_in

- **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.



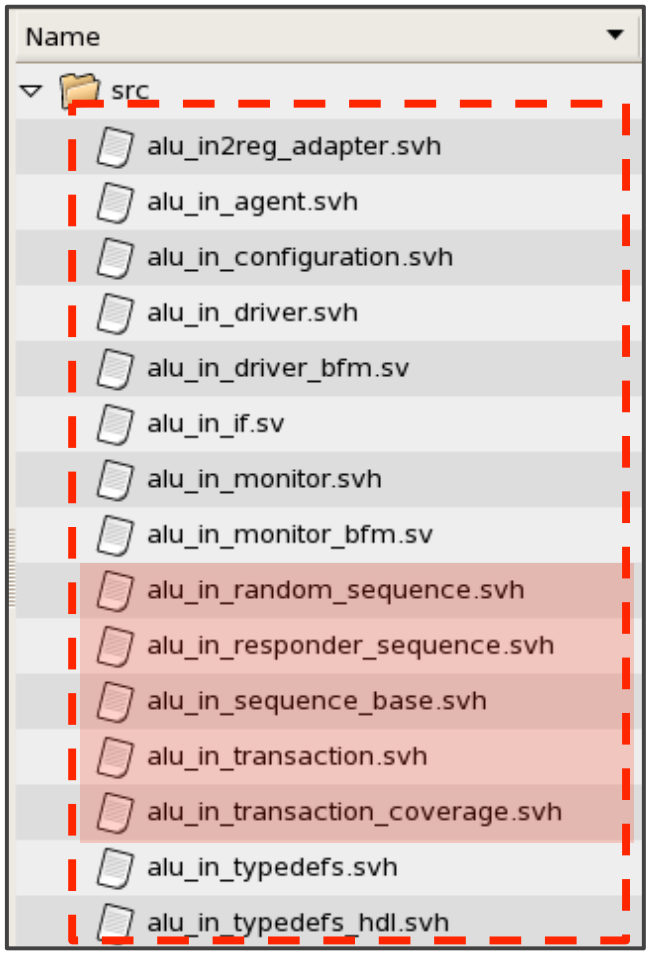
Generated UVMF Code
alu_in

- **Looking at the *alu_in_pkg/src* directory**
  - **alu_inreg_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



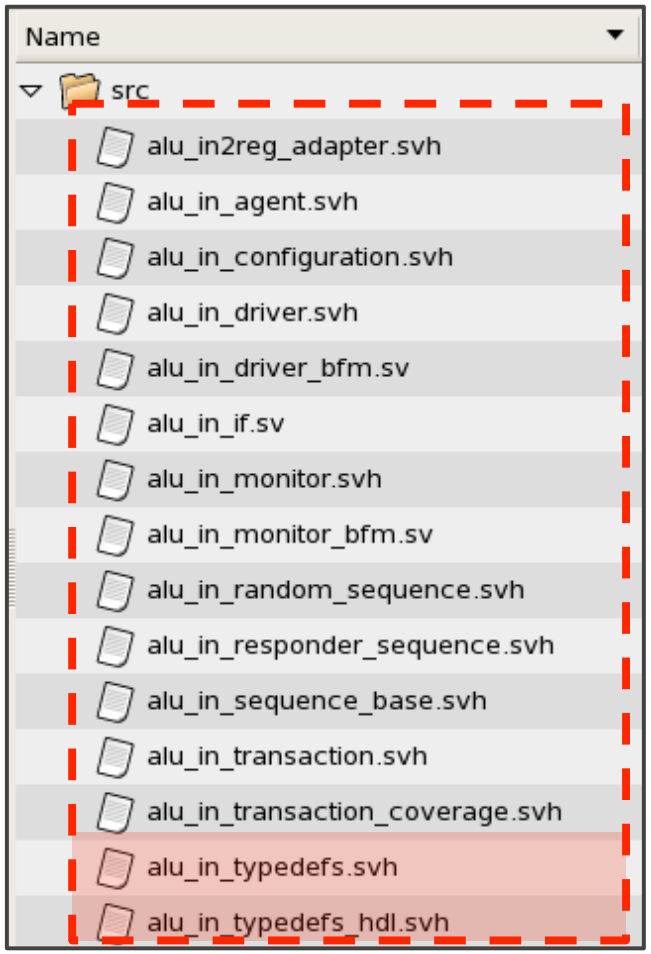
Generated UVMF Code
alu_in

- **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.



Interface Config File
alu_out_if_config.py

- **import uvmf_gen**
  - Template code that is required.
  - Imports Python code needed to interpret the rest of the config file
  - Requires $PYTHONPATH to be set as uvmf_gen.py file resides in that directory

- **intf = uvmf_gen.InterfaceClass('alu_out')**
  - The argument to the Interface Class function ('alu_out') is the name of the interface.
  - The name given will have **_pkg** appended to it to form the package name.
  - The package name (alu_out_pkg) is used to name the directory structure that contains the generated files

```
3  import uvmf_gen
4
5  ## The input to this call is the name of the desired interface
6  intf = uvmf_gen.InterfaceClass('alu_out')
7
```

Interface Config File
alu_out_if_config.py

- **intf.inFactReady = False**
  - Disables generation of additional code for the Questa InFact tool
  - For 3.6f only, code is generated by default so need to disable it
  - From 3.6g onwards, the default will change to opt-in rather than opt-out

- **intf = addParamDef('ALU_OUT_RESULT_WIDTH', 'int', '16')**
  - Specify parameters for this interface package
  - Allows creation of parameterizable agents
  - Here we define the width of the result to be generated by the alu

```
 8  ## Disable generation of InFact specific files [required for 3.6f only]
 9  intf.inFactReady = False
10
11  ## Specify parameters for this interface package.
12  ## These parameters can be used when defining signal and variable sizes.
13  # addHdlParamDef(<name>,<type>,<value>)
14  intf.addParamDef('ALU_OUT_RESULT_WIDTH','int','16')
```

Interface Config File
alu_out_if_config.py

- **intf.clock = 'clk'**
  - Specifies the interface clock signal
- **intf.reset = 'rst'**
  - Specifies the interface reset signal
- **intf.resetAssertionLevel = False**
  - Specifies the polarity of the active reset
  - False = Active Low; True = Active High

```
16  ## Specify the clock and reset signal for the interface
17  intf.clock = 'clk'
18  intf.reset = 'rst'
19  intf.resetAssertionLevel = 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_out_if_config.py

- **intf.addPort('result',ALU_OUT_RESULT_WIDTH,'input)**
  - The addPort function is required for each signal in the interface.
  - The arguments to the addPort identify the name, width and direction of the signal.
  - The options for the direction are input, output and inout.

```
21  ## Specify the ports associated with this interface.
22  ## The direction is from the perspective of the test bench as an INITIATOR
23  ##   addPort(<name>,<width>,[input|output|inout])
24  intf.addPort('done',1,'input')
25  intf.addPort('result','ALU_OUT_RESULT_WIDTH','input')
```

NOTES:

- Direction specified here is in relation to the testbench
  - i.e. 'done' is an output from the alu (DUT) and an input to the testbench
- The 'result' signal use the ALU_OUT_RESULT_WIDTH parameter which was defined using the intf.addParamDef directive

Interface Config File
alu_out_if_config.py

- **intf.addTransvar('result','bit [ALU_OUT_RESULT_WIDTH-1:0]', israndd=False, Iscompare=True)**
  - Defines a variable to be used by the transaction class.
  - The arguments required by the addTransVar 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.

```
31  ## Specify transaction variables for the interface.
32  ##   addTransVar(<name>,<type>)
33  ##     optionally can specify if this variable may be specified as 'rand'
34  ##     optionally can specify if this variable may be specified as used in do_compare()
35  intf.addTransVar('result','bit [ALU_OUT_RESULT_WIDTH-1:0]',israndd=False,iscompare=True)
```

NOTES:

- For the alu, the transaction will specify the result of the performed operation.
- israndd parameter is set to False as the testbench is not randomizing this data value, only capturing what is generated by the DUT

- Unconstrained arrays cannot be used for addTransVar
- If you need an unconstrained array, declare a fixed array in config file and modify generated code

Interface Config File
alu_out_if_config.py

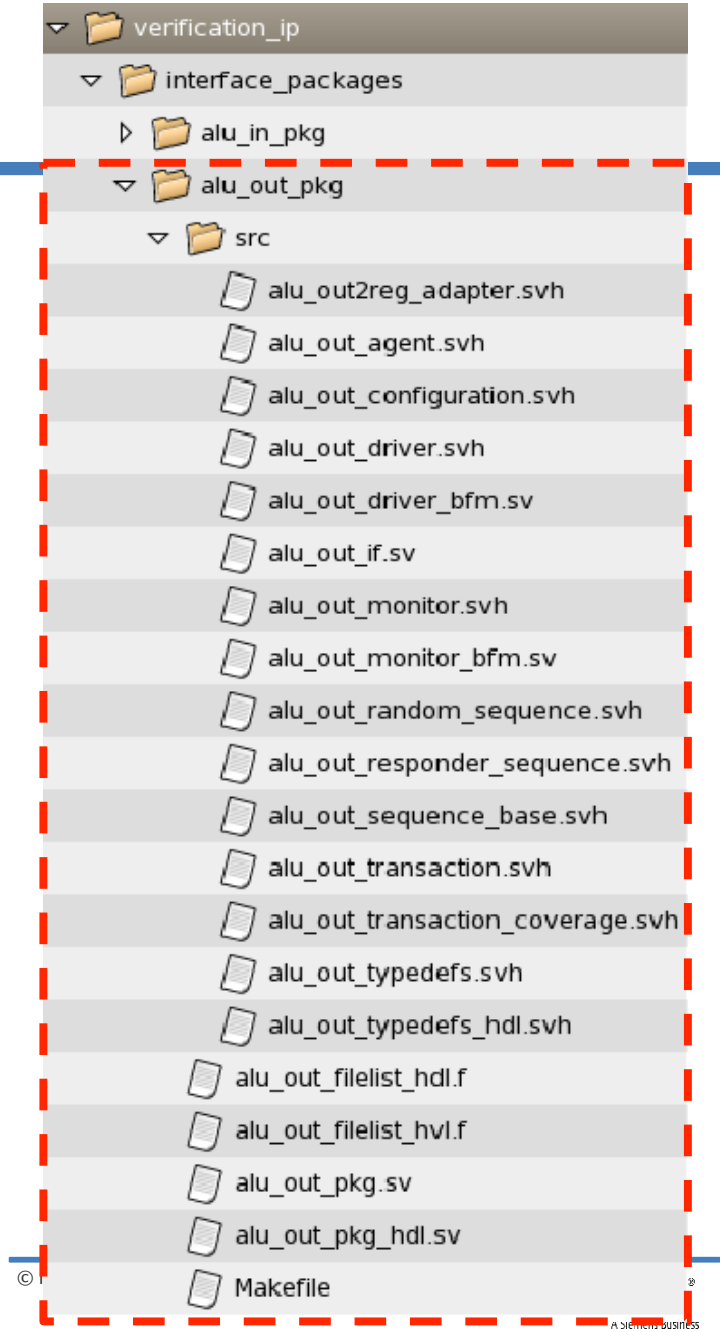
- **intf.create()**
  - This tells the python generator to create all of the interface source files for the **alu_out** agent.

```
58  ## This will prompt the creation of all interface files in their specified
59  ##  locations
60  intf.create()
```

Generated UVMF Code
alu_out_if

- **Generating the Interface Code**
  - Run python on the alu_in_if_config.py file

python alu_out_if_config.py



  - 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]

ALU Environment Config File
alu_env_config.py

- 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 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_config.py)*** in more detail

Environment Config File
alu_env_config.py

- Environment config defines the name of the environment package

```
1  #! /usr/bin/env python
2
3  import uvmf_gen
4  env = uvmf_gen.EnvironmentClass('alu')
```

Tells python to generate a environment with name 'alu_env_pkg'

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

Tells python to include the 1 x alu_in agent and 1 x alu_out agent. Defines agent instance names and the primary clock/reset.

```
12  ## Specify the agents contained in this environment
13  ##
    addAgent(<agent_handle_name>,<agent_package_name>,<clock_name>,<reset_n
    faceParameter2:value2}>, initResp = 'RESPONDER')
14  # Note: the agent_package_name will have _pkg appended to it.
15  env.addAgent('alu_in_agent',  'alu_in',   'clk', 'rst')
16  env.addAgent('alu_out_agent', 'alu_out',  'clk', 'rst')
```

Environment Config File
alu_env_config.py

- alu environment also instantiates & connects scoreboard & predictor components

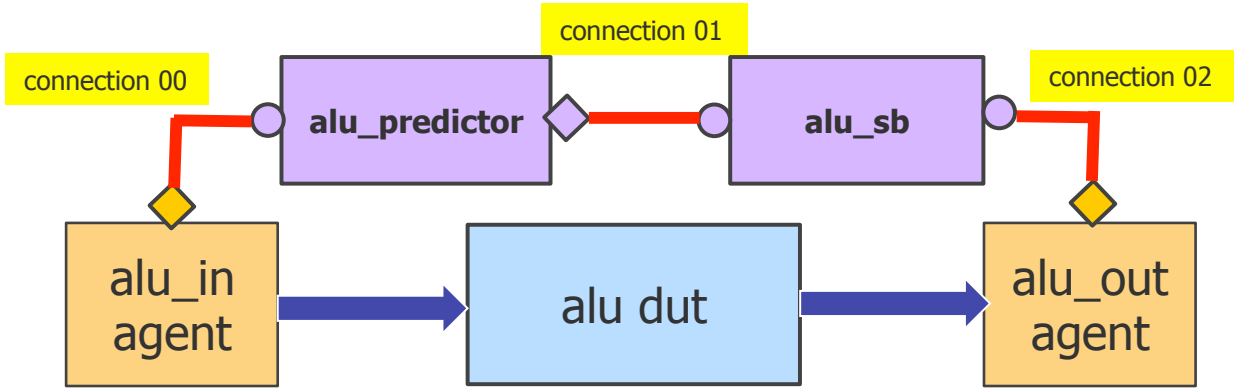
1. Command to define the predictors (defineAnalysisComponent)
   - This definition describes the component class name, list of analysis exports for receiving transactions and a list of analysis ports for broadcasting transactions.
   - The 'alu_predictor' class created by the defineAnalysisComponent API is automatically added to the environment package.
2. Command to instantiate the alu predictor (addAnalysisComponent)

```
18  ## Define the predictors contained in this environment (not instantiate, yet)
19  ## addAnalysisComponent(<keyword>,<predictor_type_name>,<dict_of_exports>,<dict_of_ports>) -
20  ## If the transaction types are parameterized and the default parameter values are desired then
21  ## the #() is required as part of defining the transaction type used by the analysis component.
22  ## doing this will add the requested analysis component to the list, enabling the use of the
23  ## given template (identified by <keyword>)                                              1
24  env.defineAnalysisComponent('predictor','alu_predictor',{'alu_in_agent_ae':'alu_in_transaction #()'},
25                                                           {'alu_sb_ap':'alu_out_transaction #()'})
26
27  ## Instantiate the components in this environment
28  ## addAnalysisComponent(<name>,<type>)                    2
29  env.addAnalysisComponent('alu_pred','alu_predictor')
```

Environment Config File
alu_env_config.py

3. Commands to add 1 x UVMF scoreboard component
4. Series of addConnection commands to connect the interface agents to the scoreboard & the predictor

```
31  ## Specify the scoreboards contained in this environment
32  ## addUvmfScoreboard(<scoreboard_handle_name>, <uvmf_scoreboard_type_name>, <transaction_type_name>)
33  env.addUvmfScoreboard('alu_sb','uvmf_in_order_scoreboard','alu_out_transaction')
34
35  ## Specify the connections in the environment
36  ## addConnection(<output_component>, < output_port_name>, <input_component>, <input_component_export_name>)
37  ## Connection 00
38  env.addConnection('alu_in_agent', 'monitored_ap', 'alu_pred', 'alu_in_agent_ae')
39  ## Connection 01
40  env.addConnection('alu_pred', 'alu_sb_ap', 'alu_sb', 'expected_analysis_export')
41  ## Connection 02
42  env.addConnection('alu_out_agent', 'monitored_ap', 'alu_sb', 'actual_analysis_export')
```



Environment Config File
alu_env_config.py

connection 00

```
env.addConnection('alu_in_agent', 'monitored_ap', 'alu_pred', 'alu_in_agent_ae')
```

alu_in_agent : instance name of agent
monitored_ap : fixed name for analysis port on all UVMF agents
alu_pred : instance name of alu predicitor
alu_in_agent_ae : name for predictor analysis export (from predictor defineAnalysisComponent API command)

connection 01

```
env.addConnection('alu_pred', 'alu_sb_ap', 'alu_sb', 'expected_analysis_export')
```

alu_pred : instance name of predictor
alu_sb_ap : fixed name for analysis port on scoreboard
['_ap' added to inst name]
alu_sb : instance name of scoreboard
expected_analysis_export : fixed name for scoreboard 'expected' analysis export

Environment Config File
alu_env_config.py

connection 02

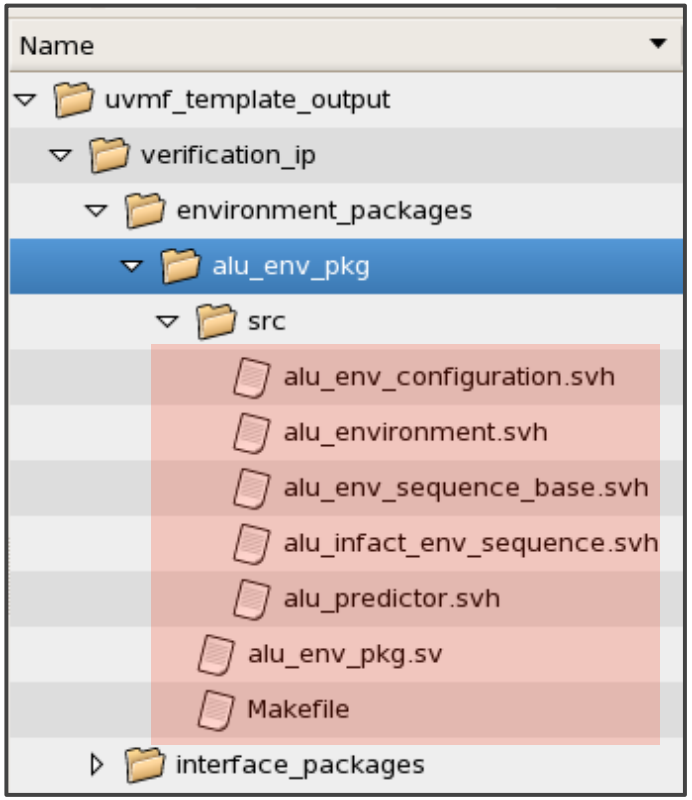
```
env.addConnection('alu_out_agent', 'monitored_ap', 'alu_sb', 'actual_analysis_export')
```

alu_out_agent : instance name of agent
monitored_ap : fixed name for analysis port on all UVMF agents
alu_sb : instance name of alu scoreboard
actual_analysis_export : fixed name for scoreboard 'actual' analysis export

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_env_configuration.svh
  - Configuration class for 'alu' environment
- alu_env.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_config.py

- 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 env (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 sequences 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_config.py)*** in more detail

ALU Bench Config File
alu_bench_config.py

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

```
3  import uvmf_gen
4
5  ## The input to this call is the name of the desired bench and the name of the top
6  ## environment package
7  ##   BenchClass(<bench_name>,<env_name>,{enviro
8  ben = uvmf_gen.BenchClass('alu','alu',{})
9
10 ## Specify parameters for this interface package.
11 ## These parameters can be used when defining signal and variable sizes.
12 # ben.addParamDef(<name>,<type>,<value>)
13
14 ## Specify the agents contained in this bench
15 ##   addBfm(<agent_handle_name>,<agent_type_name>,<clock_name>,<reset_name>,<activit
16 ben.addBfm('alu_in_agent',   'alu_in',  'clk', 'rst','ACTIVE')
17 ben.addBfm('alu_out_agent',  'alu_out', 'clk', 'rst','PASSIVE')
18
19 ## This will prompt the creation of all bench files in their specified
20 ##  locations
21 ## ben.inFactReady = False
22 ben.create()
```

Tells python to generate a bench with name 'alu'

Tells python to include the specified BFMs & defines if drivers are ACTIVE or PASSIVE

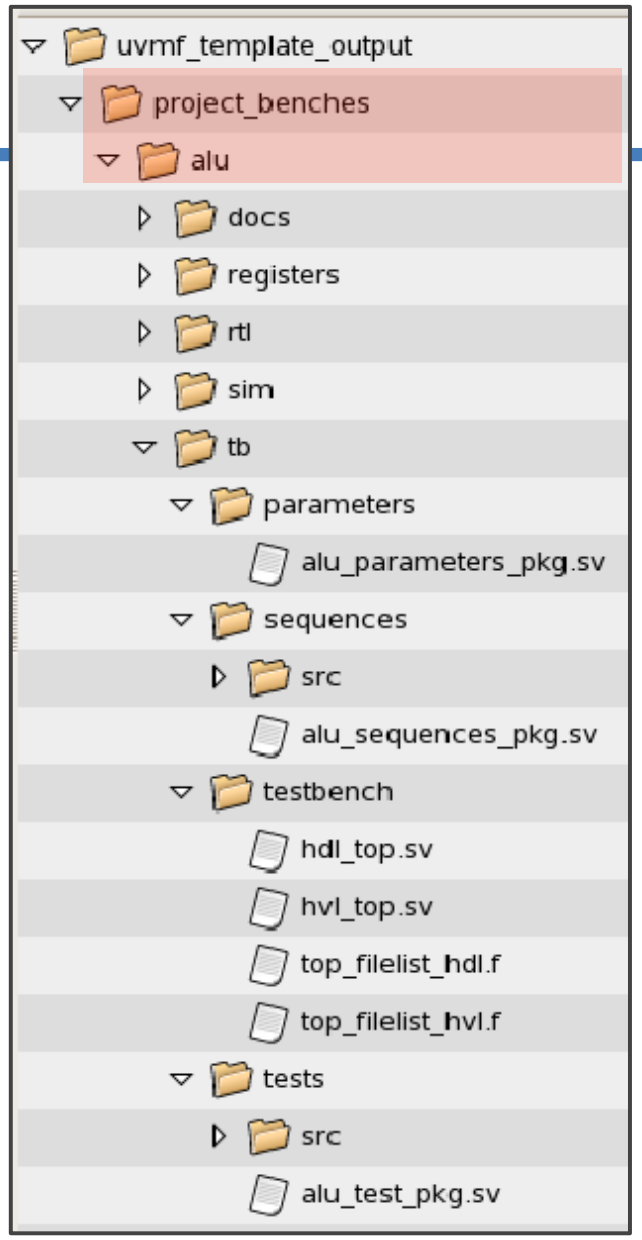
Tells python to generate the bench code

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



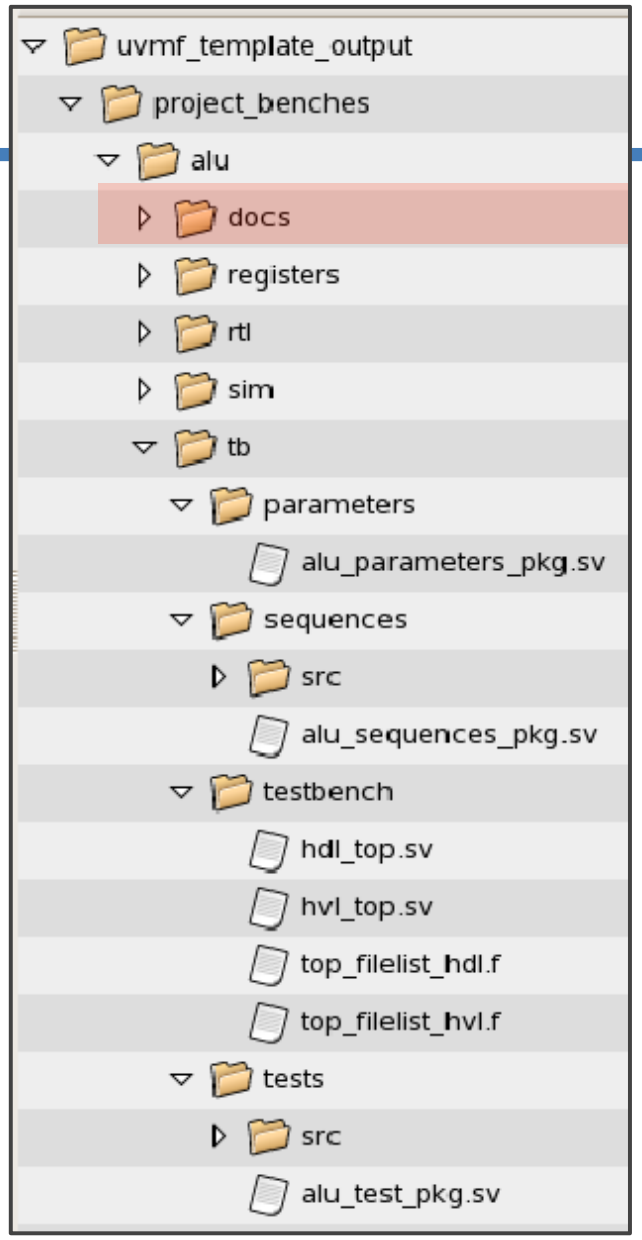
Generated UVMF Code
project_benches/alu

- Files generated under

  project_benches/alu

  - **docs**

    — Placeholder folder for user to place documentation



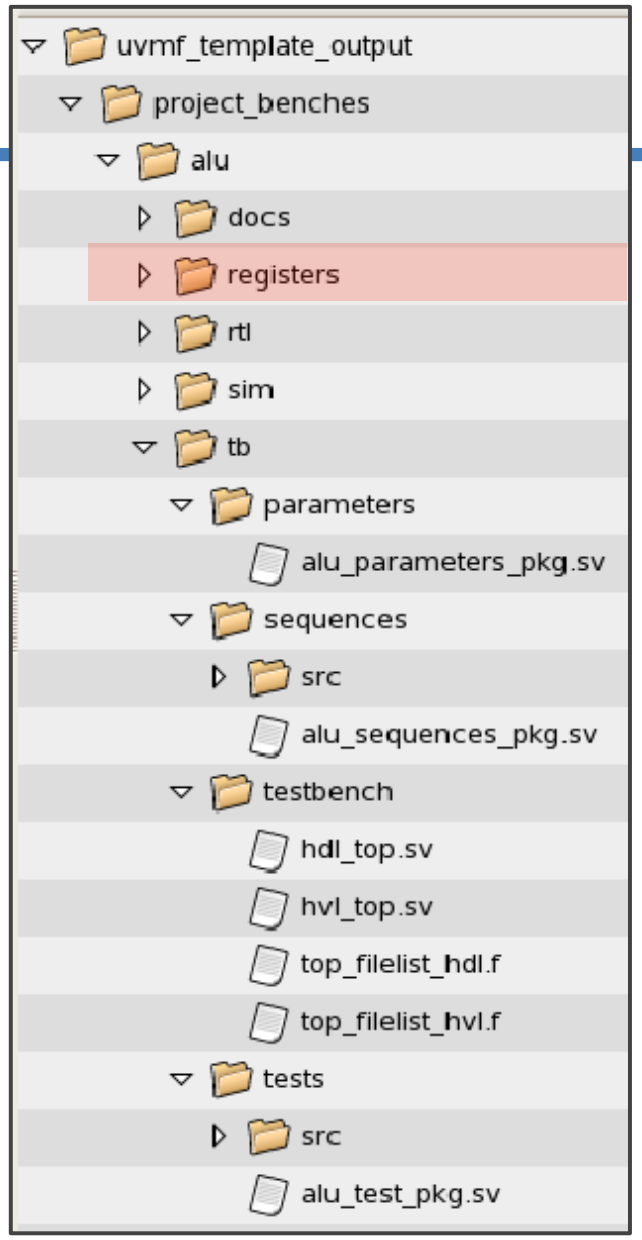
Generated UVMF Code
project_benches/alu

- Files generated under

  project_benches/alu

  - **registers**

    — Placeholder folder for user to place register layer package



Generated UVMF Code
project_benches/alu

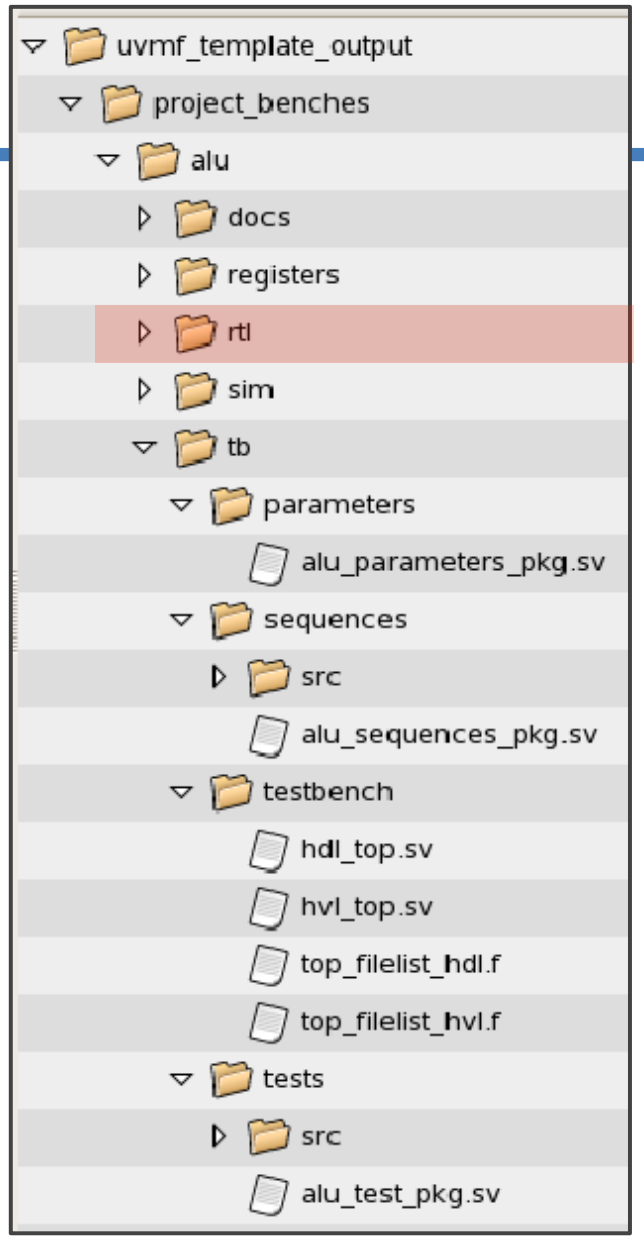
- Files generated under

  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

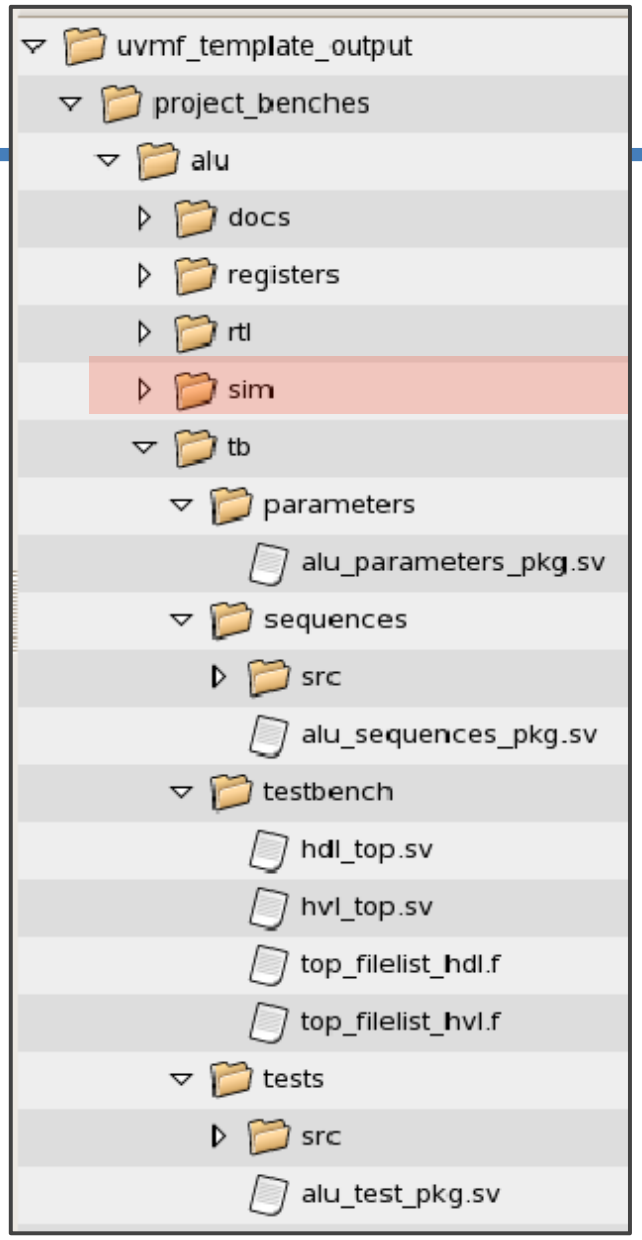
- Files generated under

  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

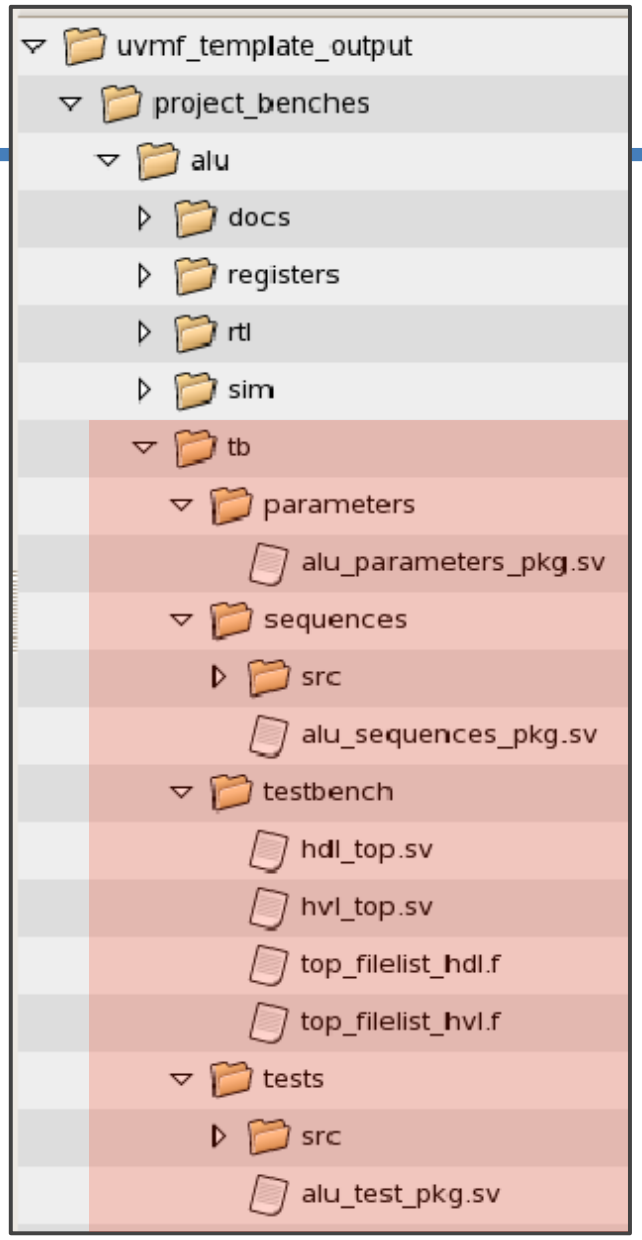
- Files generated under

  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

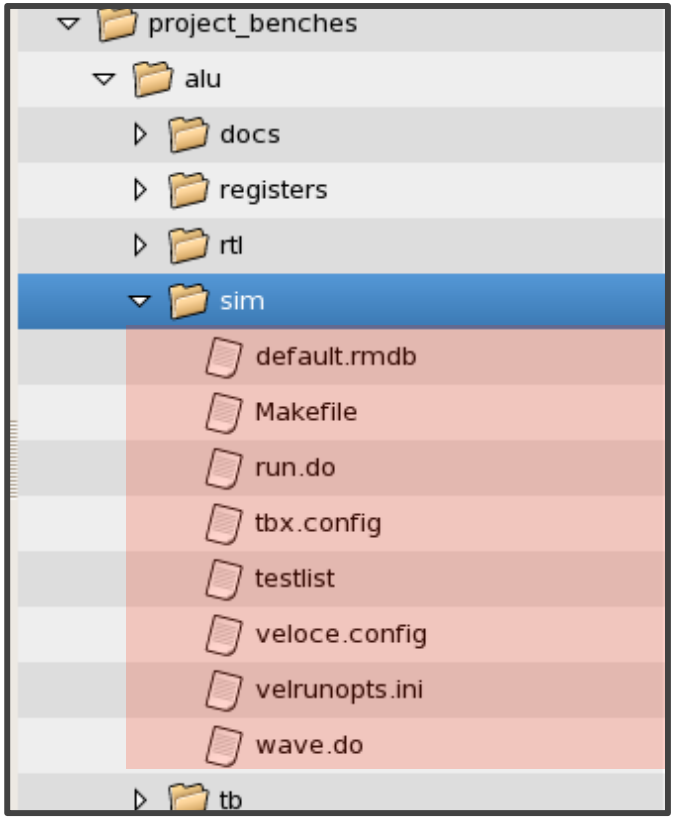


Agenda

- ALU Overview
- Config Files Explained
- Compile and Simulate Generated Code
- Adding DUT Specific Functionality

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'
```

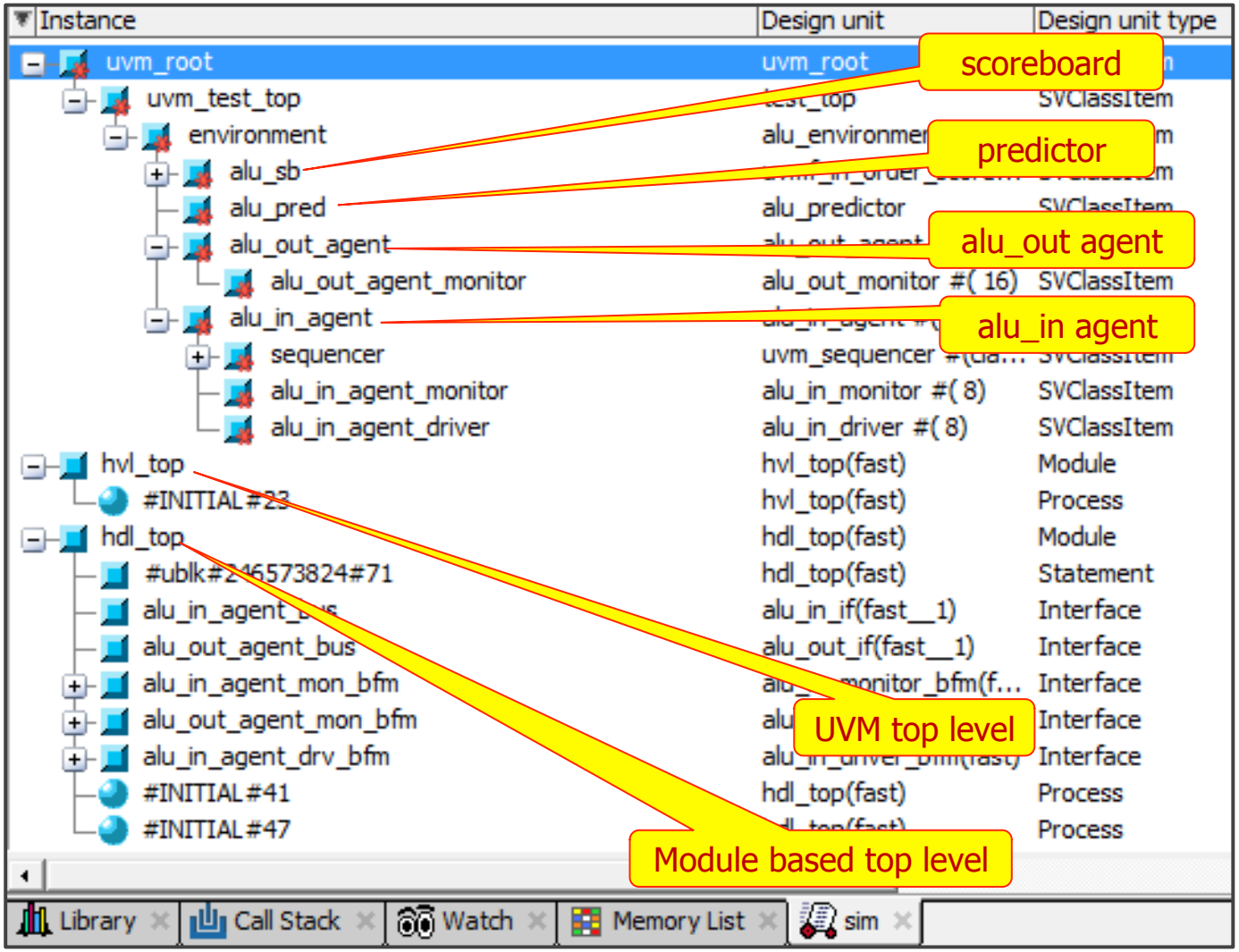
Out Of The Box Simulation
project_benches/alu/sim

- 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
```

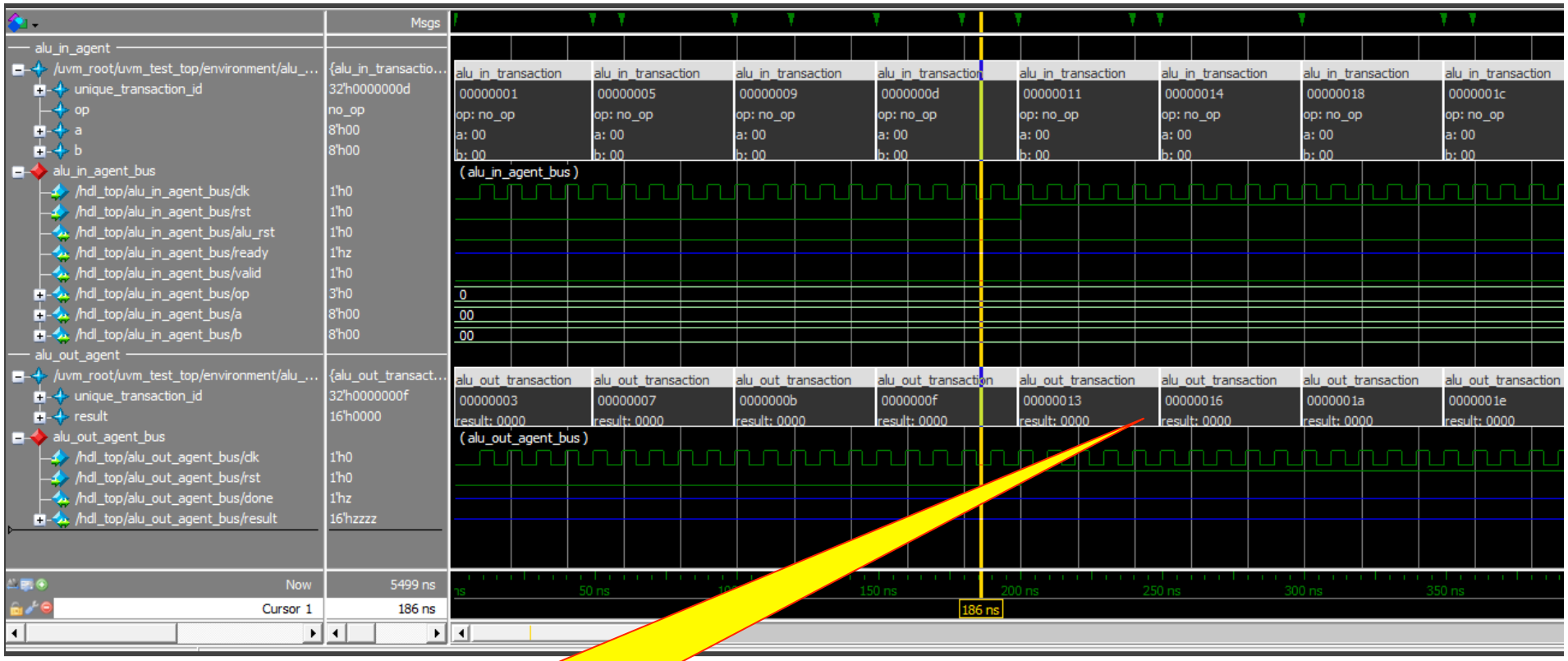
Out Of The Box Simulation

project_benches/alu/sim



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 for to run on each active agent.
    - Gets the config handles for each agent from the UVM config DB
    - Get the sequencer handles for each agent from the UVM config DB

Agenda

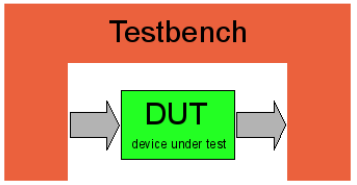
- ALU Overview
- Config Files Explained
- Compile and Simulate Generated Code
- Adding DUT Specific Functionality

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

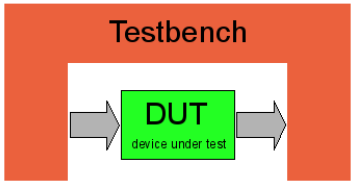
project_benches/alu/tb/testbench/hdl_top.sv



- 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
```

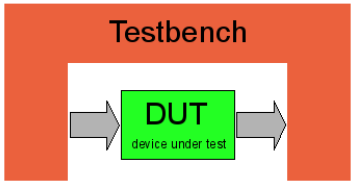
Instantiate & Wire Up the DUT
project_benches/alu/tb/testbench/hdl_top.sv



- 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 python 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.

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

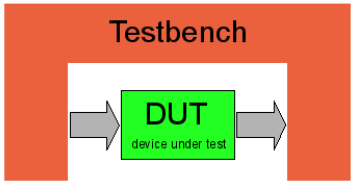
```
70  alu   #(.OP_WIDTH(8), .RESULT_WIDTH(16)) DUT  (
71        // AHB connections
72       .clk    (alu_in_agent_bus.clk ) ,
73       .rst    (alu_in_agent_bus.alu_rst ) ,
74       .ready  (alu_in_agent_bus.ready ) ,
75       .valid  (alu_in_agent_bus.valid ) ,
76       .op     (alu_in_agent_bus.op ) ,
77       .a      (alu_in_agent_bus.a ) ,
78       .b      (alu_in_agent_bus.b ) ,
79       .done   (alu_out_agent_bus.done ) ,
80       .result (alu_out_agent_bus.result ) );
81
82  initial begin  // tbx vif_binding_block
83      import uvm_pkg::uvm_config_db;
```

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 fpu_rst pin of the ali_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.

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 customers
  - **run.do** : add the vlog command to compile the alu.v source file
  - **Makefile** : uncomment line 149 which activates the comp_alu_dut target. A default filename and path is generated in the Makefile (line 122) which you would have to modify for other designs.

Makefile

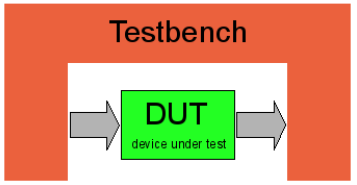
```
120 # UVMF_CHANGE_ME : Reference DUT source.
121 alu_DUT =\
122 $(UVMF_PROJECT_DIR)/rtl/verilog/alu.v
```

```
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)
```

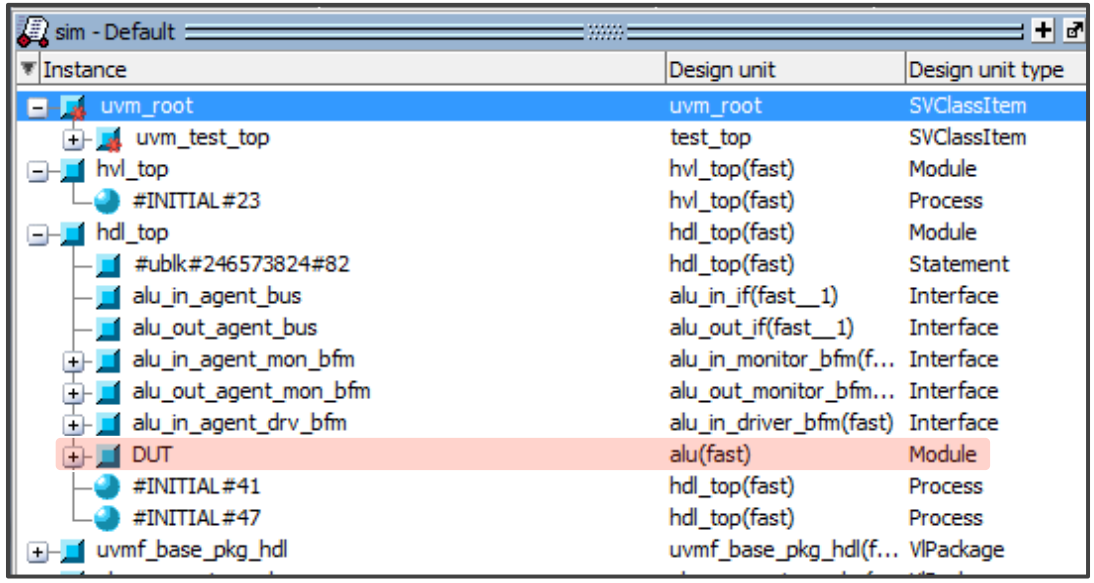
run.do

```
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 '***do_transfer***' task
  - By default the UVMF generator just has 5 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

```
129 // ****************************************************************
130   task do_transfer(                input alu_in_op_t op,
131                    input bit [ALU_IN_OP_WIDTH-1:0] a,
132                    input bit [ALU_IN_OP_WIDTH-1:0] b                );
133   // UVMF_CHANGE_ME : Implement protocol signaling.
154
155
156   @(posedge clk_i);
157   @(posedge clk_i);
158   @(posedge clk_i);
159   @(posedge clk_i);
160   @(posedge clk_i);
161   $display("alu_in_driver_bfm: Inside do_transfer()");
162 endtask
```

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 5 consecutive clock cycle delays with the following code

Modified Code

```
157        @(posedge clk_i);
158        case (op)
159           rst_op   : do_assert_rst(op);
160           default : alu_in_op(op, a, b);
161        endcase
162        $display("alu_in_driver_bfm: Inside do_transfer()");
163
164     endtask
```

NOTES:
The reset operation does only drives the alu reset Pin and is there handled separately in it's own task.

  - Add the following 2 tasks to the module

```
165  // ****************************************************
166    task do_assert_rst(input alu_in_op_t op);
167    $display("%g ***************   Starting Reset", $time);
168       op_o <= op;
169       alu_rst_o <= 1'b0;
170       repeat (10) @(posedge clk_i);
171       alu_rst_o <= 1'b1;
172       repeat (5) @(posedge clk_i);
173    $display("%g ***************   Ending Reset", $time);
174    endtask
```

New Code

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.

```
177  // ****************************************************
178    task alu_in_op(input alu_in_op_t op,
179                   input bit [ALU_IN_OP_WIDTH-1:0] a,
180                   input bit [ALU_IN_OP_WIDTH-1:0] b);
181
182      while ( ready_i == 1'b0 ) @(posedge clk_i) ;
183      valid_o <= 1'b1;
184      op_o <= op;
185      a_o <= a;
186      b_o <= b;
187
188      @(posedge clk_i);
189      valid_o <= 1'b0;
190      op_o <= {3{1'bx}};
191      a_o <= {ALU_IN_OP_WIDTH{1'bx}};
192      b_o <= {ALU_IN_OP_WIDTH{1'bx}};
193
194    endtask
```

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

```
76  // INITIATOR mode output signals
77    tri          alu_rst_i;
78    bit          alu_rst_o;
```

Original Code

```
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;
```

```
76  // INITIATOR mode output signals
77    tri          alu_rst_i;
78    bit          alu_rst_o = 1'b1;
```

Modified Code

```
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:

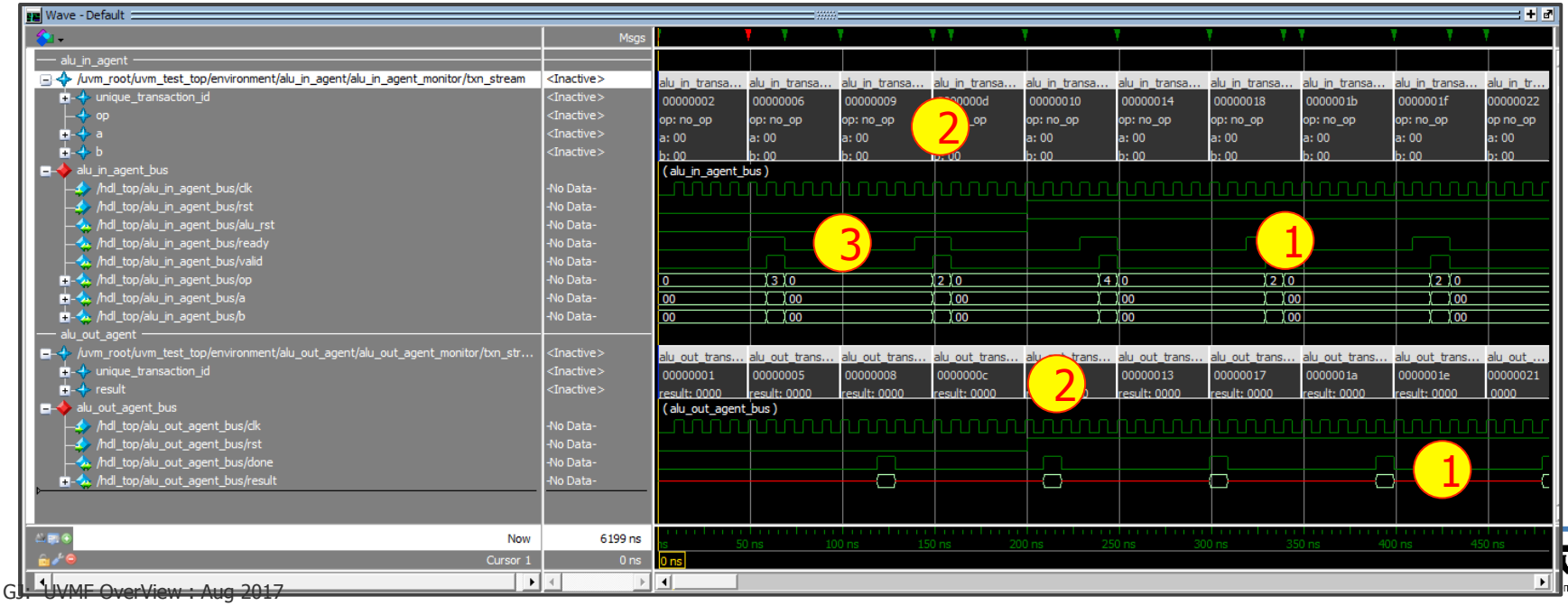
- alu_rst is active low so we initialize the default value of alu_rst_o to be '1'.
- We want to reset the alu if either the top level reset (rst_i) is active or when a RST_OP operation is received which drives alu_rst_i low. So we logically AND the 2 reset driving signals alu_rst_

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

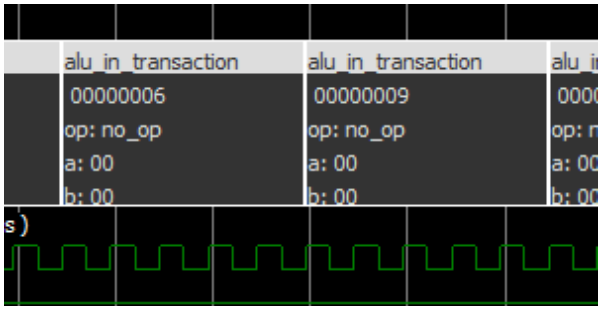
verification_ip/interface_packages/alu_in_pkg/src/alu_in_monitor_BFM.sv

- 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

```
124  // ****************************************************************
125      task do_monitor(
126                      output alu_in_op_t op,
127                      output bit [ALU_IN_OP_WIDTH-1:0] a,
128                      output bit [ALU_IN_OP_WIDTH-1:0] b
129  // UVMF_CHANGE_ME : Implement protocol monitoring.
138
139      @(posedge clk_i);
140      @(posedge clk_i);
141      @(posedge clk_i);
142      @(posedge clk_i);
143
144      endtask
```

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

```
124 // ****************************************************************************
125     task do_monitor(
126                     output alu_in_op_t op,
127                     output bit [ALU_IN_OP_WIDTH-1:0] a,
128                     output bit [ALU_IN_OP_WIDTH-1:0] b                    );
129 // UVMF_CHANGE_ME : Implement protocol monitoring.
138
139     //-start_time = $time;
140       // Hold here until signal event happens to capture bus values
141     while (valid_i == 1'b0 && alu_rst_i == 1'b1) begin
142       @(posedge clk_i);
143     end
144     op = alu_in_op_t'(op_i);
145     a  = a_i;
146     b  = b_i;
147
148     if (alu_rst_i == 1'b0) begin
149        while (alu_rst_i == 1'b0) @(posedge clk_i);
150     end else
151        @(posedge clk_i);
152     //-end_time = $time;
153
154     endtask
```

Modified Code

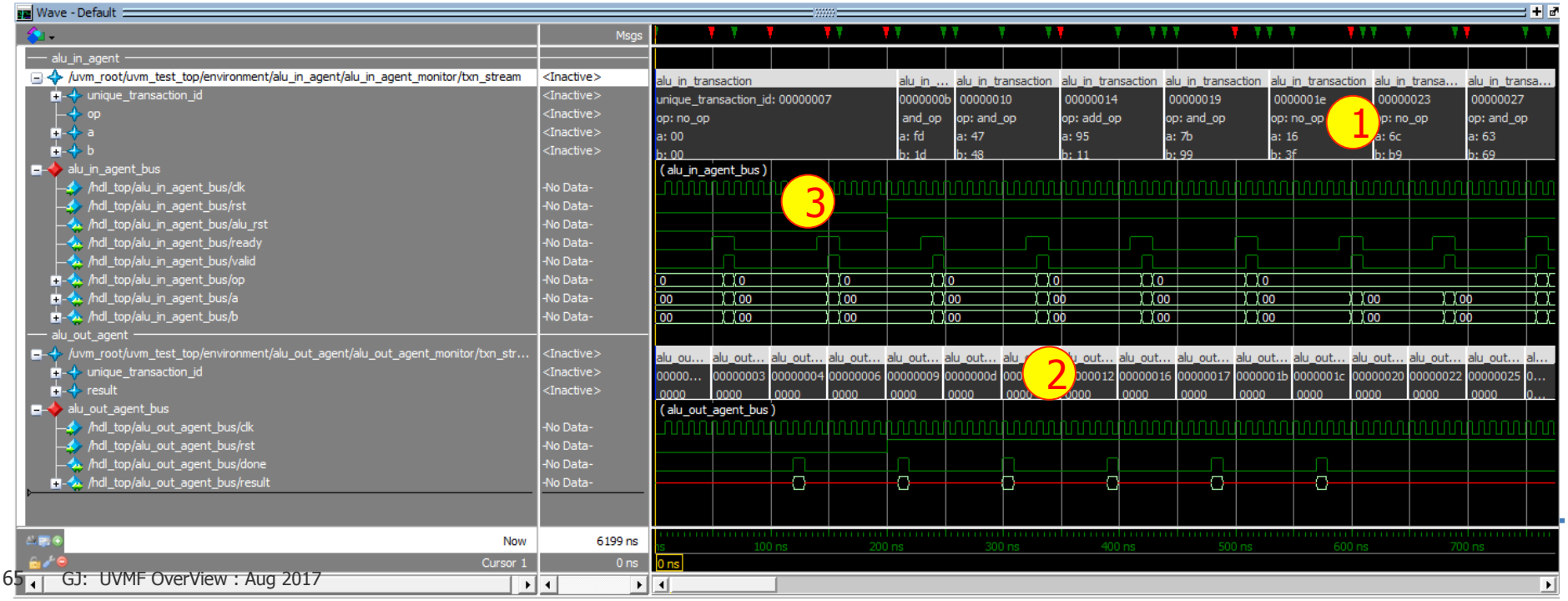
NOTES
- Wait until either valid goes high or reset goes active
- Read bus values
- If reset then wait until reset becomes inactive

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.



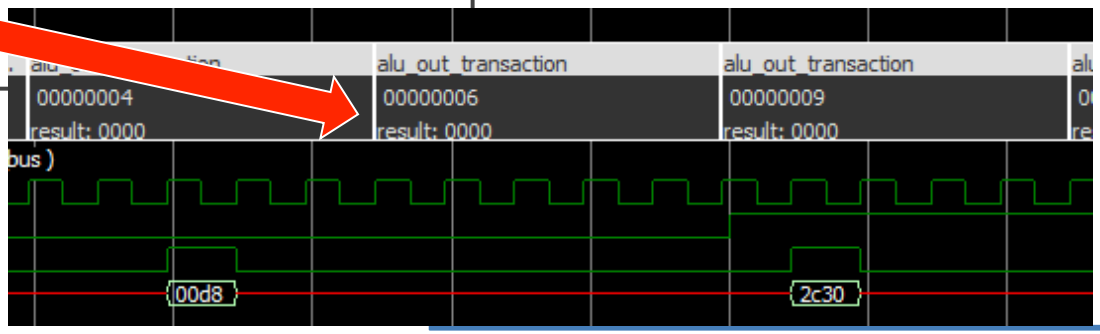
Adding Protocol Information To The Monitor BFM

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

```
110  // ****************************************************************************
111       task do_monitor(
112                       output bit [ALU_OUT_RESULT_WIDTH-1:0] result                );
113  // UVMF_CHANGE_ME : Implement protocol monitoring.
118
119       @(posedge clk_i);
120       @(posedge clk_i);
121       @(posedge clk_i);
122       @(posedge clk_i);
123
124       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

Mentor A Siemens Business

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

```
110  // ****************************************************************************
111       task do_monitor(
112                       output bit [ALU_OUT_RESULT_WIDTH-1:0] result);
113  // UVMF_CHANGE_ME : Implement protocol monitoring.
118
119       //-start_time = $time + 1;
120       while ( done_i == 1'b0 ) @(posedge clk_i);
121       result = result_i;
122       //-end_time = $time - 1;
123
124       endtask
```

Modified Code

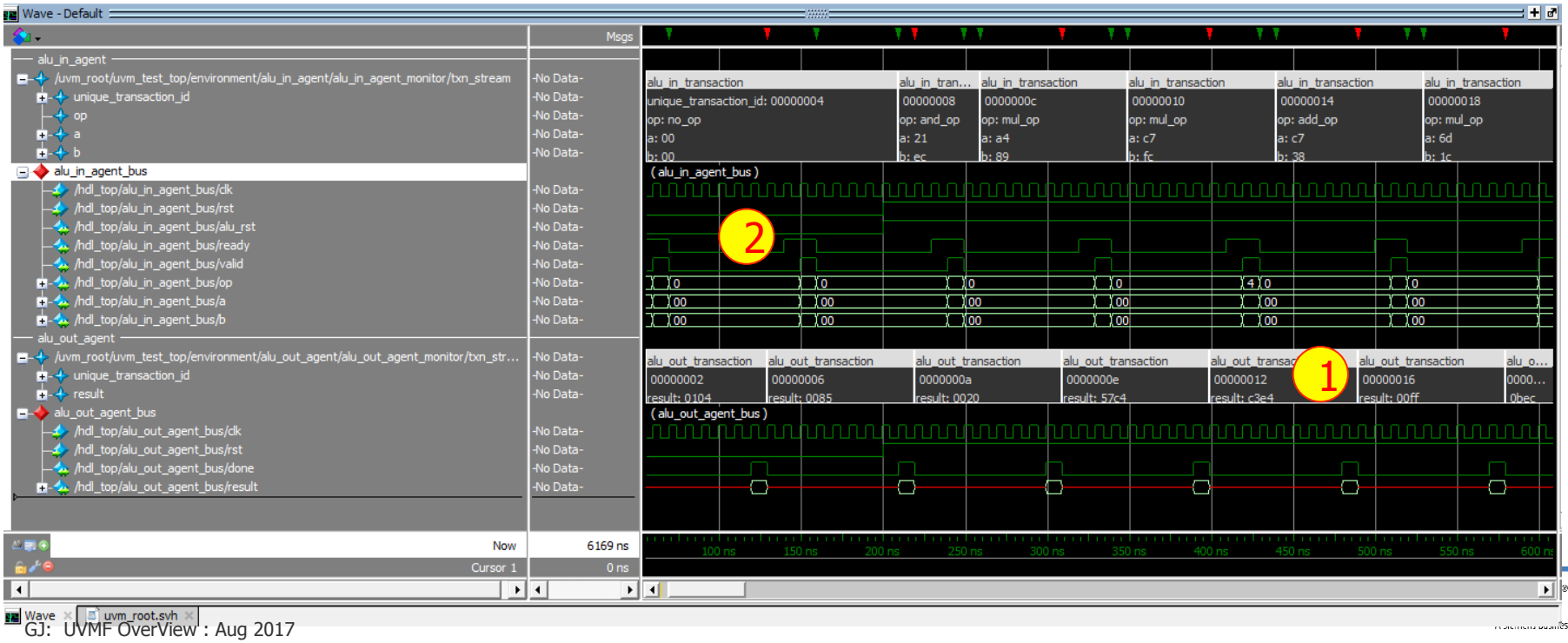
NOTES
- Wait until done_i goes high
- Read result value

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

```
68  // ****************************************************************************
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
```

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 task provide 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

```
68  // ****************************************************************************
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    // Delay start of sequences til lreset has ended and then wait a few clocks after that
75     alu_in_agent_config.wait_for_reset();
76     alu_in_agent_config.wait_for_num_clocks(10);
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
```

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_transaction***' task
  - 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 functonality

Original Code

```
68  virtual function void write_alu_in_agent_ae(alu_in_transaction #() t);
69    `uvm_info("alu_predictor", "Transaction Received through alu_in_agent_ae", UVM_MEDIUM)
70    `uvm_info("alu_predictor", {"            Data: ",t.convert2string()}, UVM_FULL)
71
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    // Code for sending output transaction out through alu_sb_ap
76    alu_sb_ap.write(alu_sb_ap_output_transaction);
77  endfunction
```

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).

```
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
```

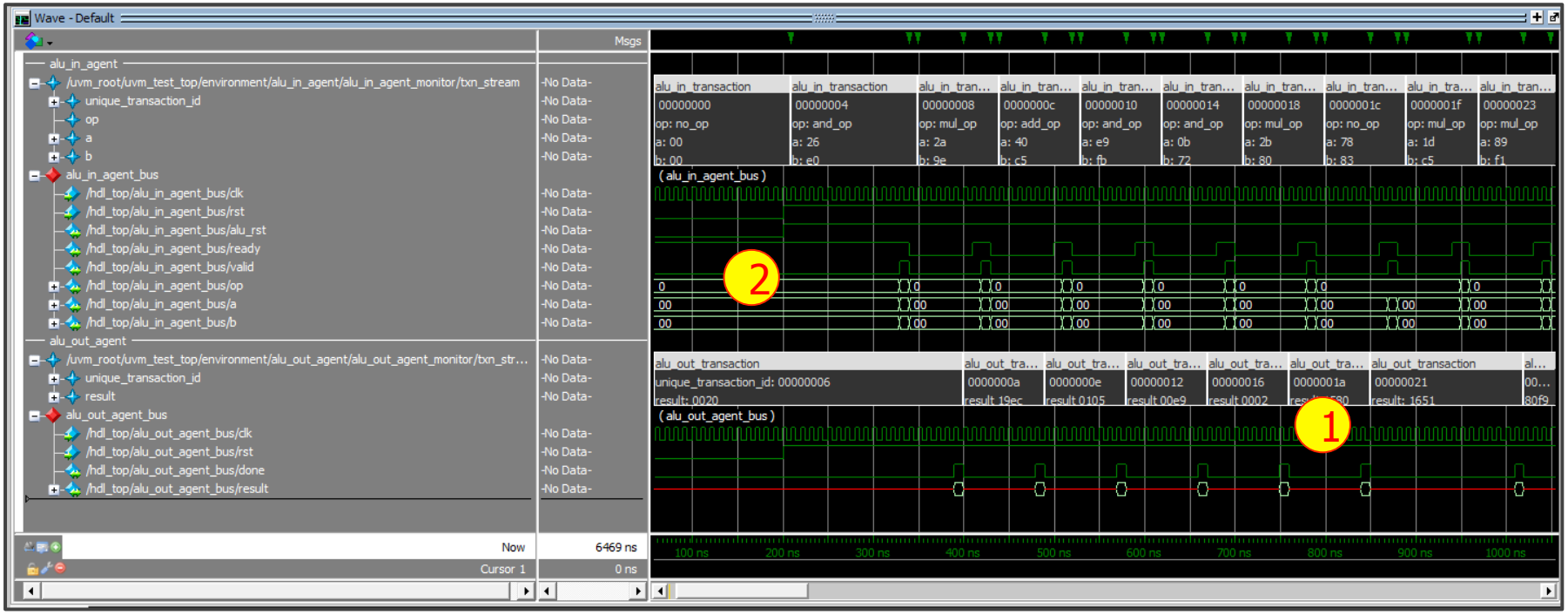
Modified Code

Adding DUT Behaviour To The Predictor

verification_ip/environment_packages/alu_env_pkg/src/alu_predictor.svh

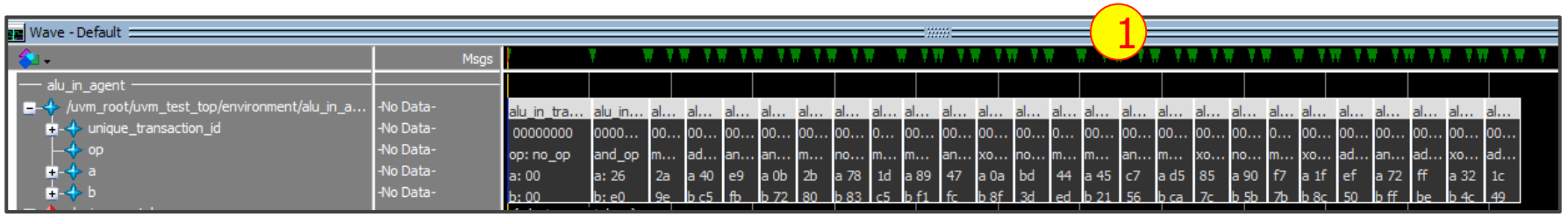
- Checking the predictor code changes
  - Use the run.do or the Makefile (make debug) to check that there are no compilation errors in the code you have modified/added.

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

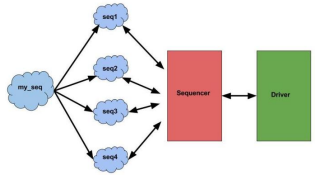
- Basic ALU operation appears to be working
  1. There should be no errors at this stage. In the wave window there should be no red triangles, which would indicate UVM Errors from the scoreboard



- 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 python 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

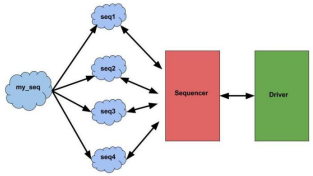
3. Copy the alu_in_random_sequence.svh file to alu_in_reset_sequence.svh

4. Edit the sequence and change all references to ali_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

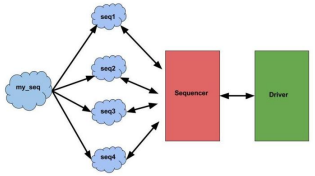



```
20  class alu_in_reset_sequence  #(
21        int ALU_IN_OP_WIDTH = 8
22        )
23  extends alu_in_sequence_base  #(
24                                .ALU_IN_OP_WIDTH(ALU_IN_OP_WIDTH)
25                               ) ;
26
27    `uvm_object_param_utils( alu_in_reset_sequence #(
28                             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

```
36  package alu_in_pkg;
37
38      import uvm_pkg::*;
39      import questa_uvm_pkg::*;
40      import uvmf_base_pkg_hdl::*;
41      import uvmf_base_pkg::*;
42      import alu_in_pkg_hdl::*;
43      `include "uvm_macros.svh"
44
45      export alu_in_pkg_hdl::*;
46
47
48      `include "src/alu_in_typedefs.svh"
49      `include "src/alu_in_transaction.svh"
50
51      `include "src/alu_in_configuration.svh"
52      `include "src/alu_in_driver.svh"
53      `include "src/alu_in_monitor.svh"
54
55      `include "src/alu_in_transaction_coverage.svh"
56      `include "src/alu_in_sequence_base.svh"
57      `include "src/alu_in_random_sequence.svh"
58      `include "src/alu_in_reset_sequence.svh"
59
60      `include "src/alu_in_responder_sequence.svh"
61      `include "src/alu_in2reg_adapter.svh"
62
63      `include "src/alu_in_agent.svh"
64
65      typedef uvm_reg_predictor #(alu_in_transaction) alu_in2reg_predictor;
66
67
68  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/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

```
40  class alu_random_sequence #(int ALU_IN_OP_WIDTH = 8) extends alu_bench_sequence_base;
41
42    `uvm_object_utils( alu_random_sequence );
43
44    typedef alu_in_reset_sequence #(ALU_IN_OP_WIDTH) alu_reset_sequence_t;
45    alu_reset_sequence_t alu_in_reset_s;
46
47
48  // ****************************************************************************
49    function new( string name = "" );
50       super.new( name );
51    endfunction
52
53  // ****************************************************************************
54    virtual task body();
55       alu_in_agent_random_seq = alu_in_random_sequence#()::type_id::create("alu_in_agent_random_seq");
56       alu_in_reset_s  = alu_in_reset_sequence#()::type_id::create("alu_in_reset_s");
57
58       alu_in_agent_config.wait_for_reset();
59       alu_in_agent_config.wait_for_num_clocks(10);
60
61       repeat (10) alu_in_agent_random_seq.start(alu_in_agent_sequencer);
62       alu_in_reset_s.start(alu_in_agent_sequencer);
63       repeat (5) alu_in_agent_random_seq.start(alu_in_agent_sequencer);
64
65       alu_in_agent_config.wait_for_num_clocks(50);  // 50 = 1000ns/20ns
66
67    endtask
68
69  endclass
```

3

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

```
26  package alu_sequences_pkg;
27
28      import uvm_pkg::*;
29      import questa_uvm_pkg::*;
30      import uvmf_base_pkg::*;
31      import alu_in_pkg::*;
32      import alu_out_pkg::*;
33      import alu_parameters_pkg::*;
34
35
36      `include "uvm_macros.svh"
37
38      `include "src/alu_bench_sequence_base.svh"
39      `include "src/alu_random_sequence.svh"
40      `include "src/infact_bench_sequence.svh"
41      `include "src/example_derived_test_sequence.svh"
42
43  endpackage
```

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_derviced_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***.

```
36  class alu_random_test extends test_top;
37
38    `uvm_component_utils( alu_random_test );
39
40    function new( string name = "", uvm_component parent = null );
41      super.new( name, parent );
42    endfunction
43
44    virtual function void build_phase(uvm_phase phase);
45      alu_bench_sequence_base::type_id::set_type_override(alu_random_sequence #(8)::get_type());
46      super.build_phase(phase);
47    endfunction
48
49  endclass
```

New Code

Mentor A Siemens Business

Adding a New UVM Test

project_benches/alu/tb/tests
alu_test_pkg.sv

- Add the new test to the alu_test_pkg

```
24  package alu_test_pkg;
25
26      import uvm_pkg::*;
27      import questa_uvm_pkg::*;
28      import uvmf_base_pkg::*;
29      import alu_parameters_pkg::*;
30      import alu_env_pkg::*;
31      import alu_sequences_pkg::*;
32
33
34      `include "uvm_macros.svh"
35
36      `include "src/test_top.svh"
37      `include "src/alu_random_sequence.svh"
38      `include "src/example_derived_test.svh"
39
40  endpackage
```

Modified Code

Simulating The New Test

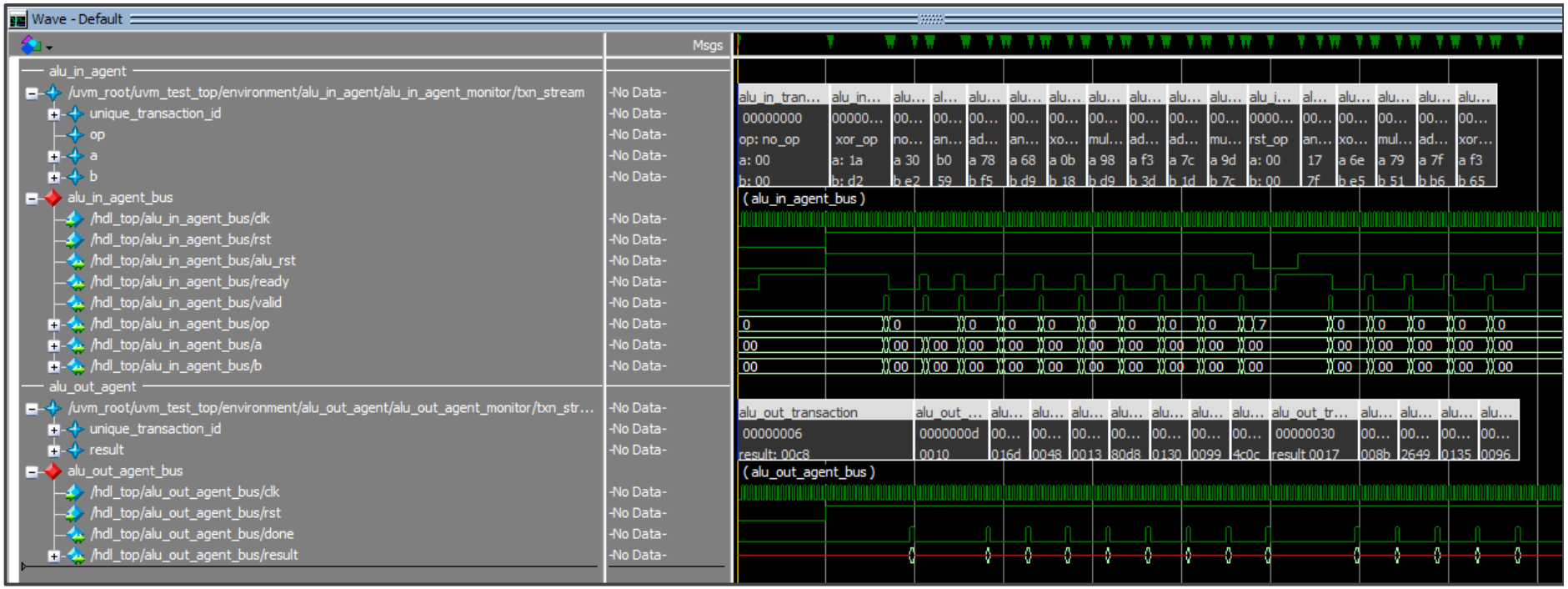
- Go to the ***project_benches/alu/sim*** folder
- Windows Users
  - Edit run.do and modify the modify the last line where +UVM_TESTNAME specifies the test to run

```
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 Colour To The Transactions

verification_ip/interface_packages/alu_in_pkg/src
alu_in_transaction.svh

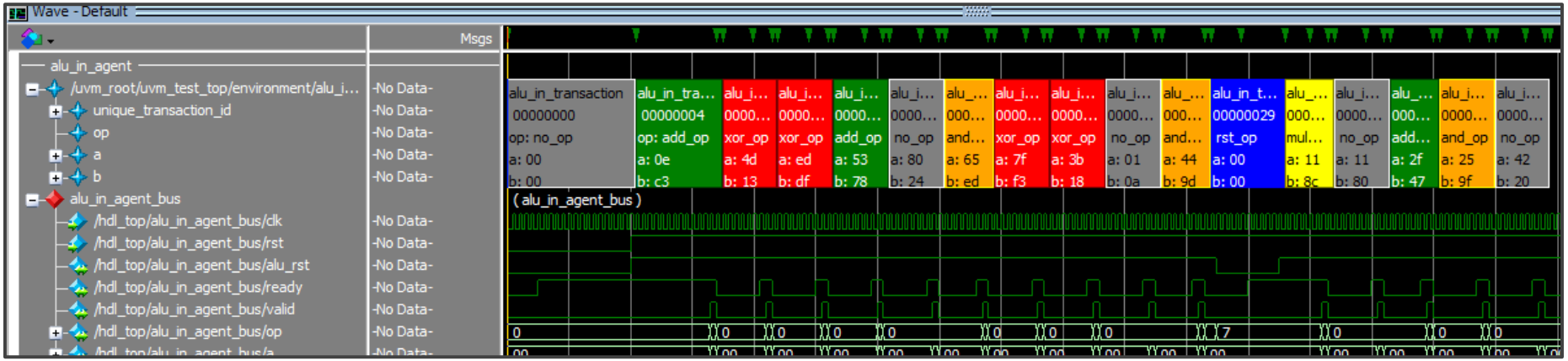
- Edit the file alu_in_transaction.svh
- Find the function called 'add_to_wave'
  - It contains some comments about adding colour to the trnasactions
  - Add the code highlighted below to the function which will assign a different colour to the transactions depending on the opcode value

```
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 colour coded depending on the op code
2. The colours 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

www.mentor.com