UVM Framework
JTAG Interface

By: Silicon Crew

Team members :

Nayana T R
Mahadevaswamy B N
Nancy dimry
Gopal Vamsi
Piyush Agrawal
Hemanth V
Naveen V
Sukeerthana B
Radhika Joshi

Contents

UVM Framework

The UVM Framework is a reuse methodology that verification teams can leverage. It supports component level verification reuse across projects and environment reuse from block through chip to system level simulation.

Features:-

- UVM Code Generation.
- UVM JumpStart.
- UVM Reuse Methodology.

YAML (YAML Ain’t Markup Language.)

- YAML is a case-sensitive language. Indentations are to be taken care in Yaml.
- It generates more reusability. Reduces line of code.
- YAML configuration files are used as the inputs to the UVM Framework code generators and they determine the content of the generated code

Data in YAML is generally defined by:

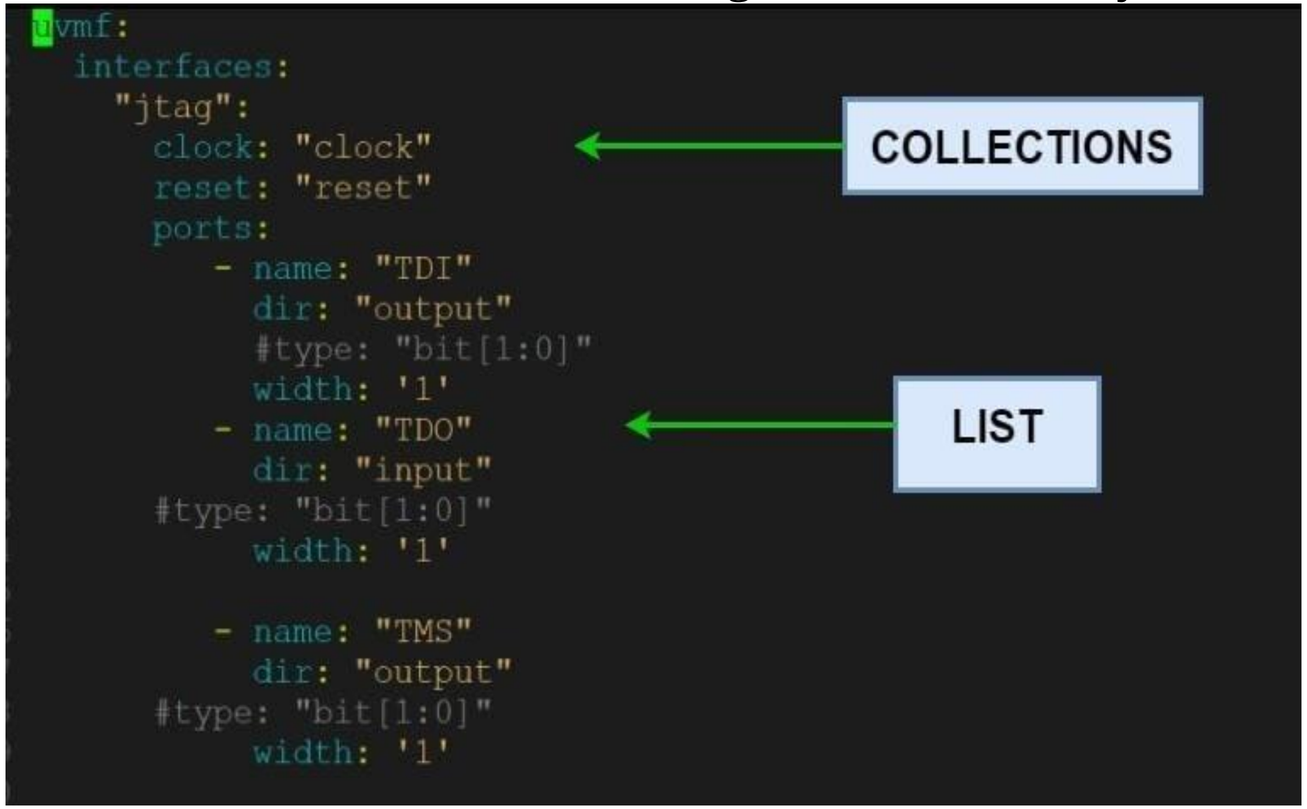
➢ Order list : Eg: clock: “clock”

Here, key is clock and value is “clock”

➢ Key value pair : Eg: - name: “TDO”

Here, the ‘-‘ symbol donates a list.

- Comments start with the number sign (#), can start anywhere on a line and continue until the end of the line.



```
uvmf:
  interfaces:
    "jtag":
      clock: "clock"
      reset: "reset"
      ports:
         - name: "TDI"
           dir: "output"
           #type: "bit[1:0]"
           width: '1'
         - name: "TDO"
           dir: "input"
      #type: "bit[1:0]"
           width: '1'

         - name: "TMS"
           dir: "output"
      #type: "bit[1:0]"
           width: '1'
```

JTAG

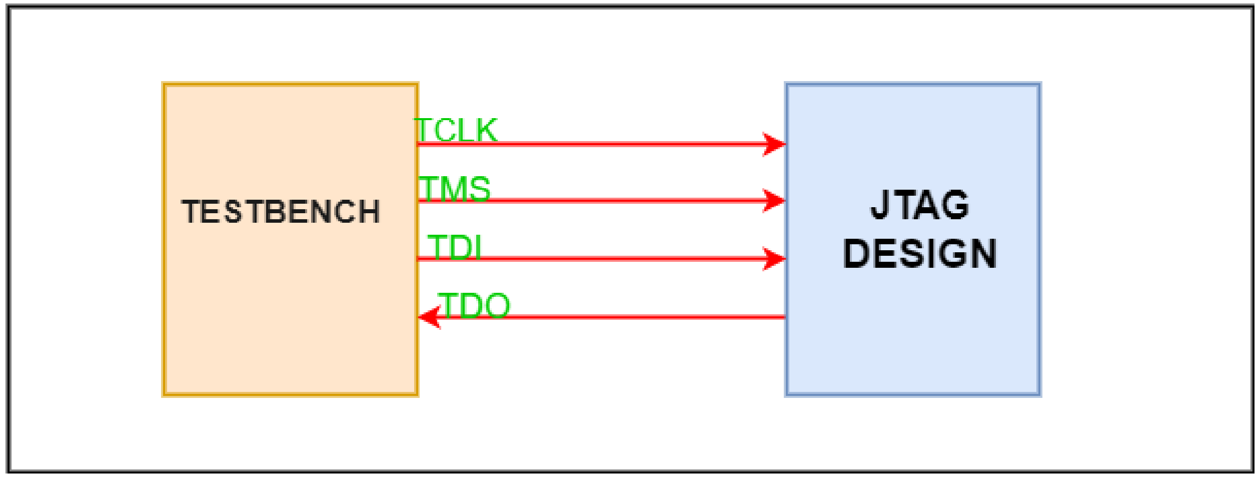
- Jtag stands for Joint Test Action Block.
- Advances in Silicon design need to increase device density; more recently, BGA packaging has reduced the efficiency of traditional testing methods.
- In order to overcome these problems JTAG protocol introduce.

Interface Signals

- The JTAG interface, collectively known as a Test Access Port, or TAP, uses the following signals to support the operation of the boundary scan.
- **TCK** (Test Clock) – this signal synchronises the internal state machine operations.
- **TMS** (Test Mode Select) – this signal is sampled at the rising edge of TCK to determine the next state.

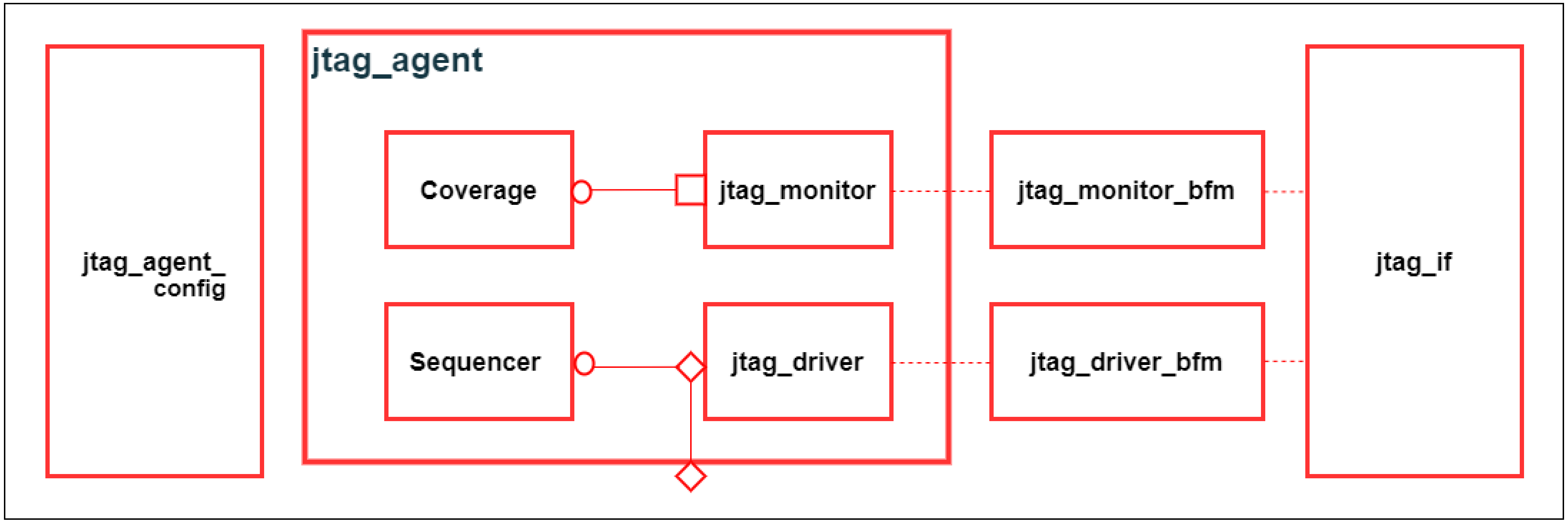
JTAG

- **TDI** (Test Data In) – this signal represents the data shifted into the device's test or programming logic. It is sampled at the rising edge of TCK when the internal state machine is in the correct state.
- **TDO** (Test Data Out) – this signal represents the data shifted out of the device's test or programming logic and is valid on the falling edge of TCK when the internal state machine is in the correct state.
- **TRST** (Test Reset) – This is an optional pin that, when available, can reset the TAP controller's state machine.



Problem statement

Write a YAML file to generate the following interface diagram?



JTAG Agent/Interface code generator
jtag_interface.yaml

The below structure shows the structure for “jtag” interface. This is saved under directory name jtag_yaml with the file name jtag_interface.yaml

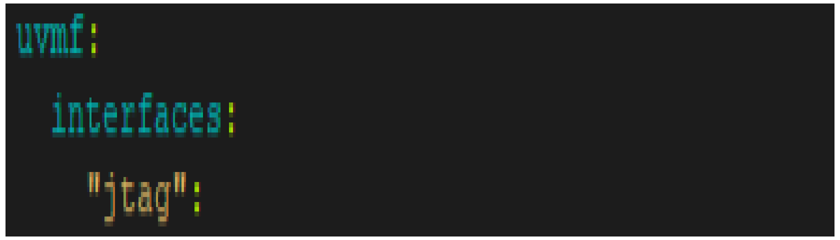
uvmf:

- Top level key, file being passed is in describes valid uvmf component

interfaces:

"jtag":

- One or more interfaces can be described under interfaces, here “jtag” is the individual interface described.



```
uvmf:
  interfaces:
    "jtag":
```

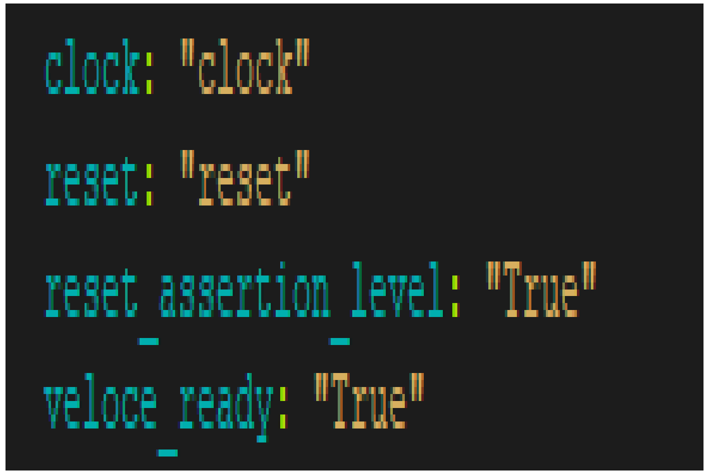
JTAG Agent/Interface code generator
jtag_interface.yaml

```
clock: "clock"

reset: "reset"

reset_assertion_level: "True"
```

- Identifies the primary clock and primary reset to be used in the interface agent as “clock” and “reset” with active HIGH reset polarity.



```
clock: "clock"
reset: "reset"
reset_assertion_level: "True"
veloce_ready: "True"
```

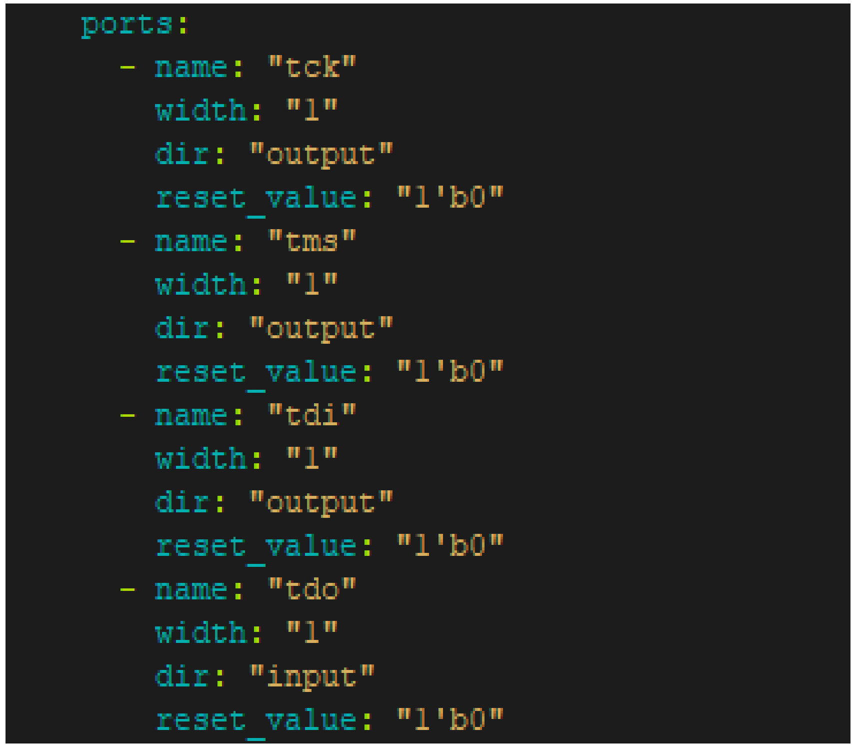
```
veloce_ready: "True"
```

- Default is True, if it is true the generated framework can be implemented on Veloce (Emulation tool)

JTAG Agent/Interface code generator
jtag_interface.yaml

```
ports:
    - name: "tck"
      width: "1"
      dir: "output"
      reset_value: "1'b0"
    - name: "tms"
      width: "1"
      dir: "output"
      reset_value: "1’b0”
```

- The port list describes what goes as interface inputs; direction is defined from the test bench or interface perspective.

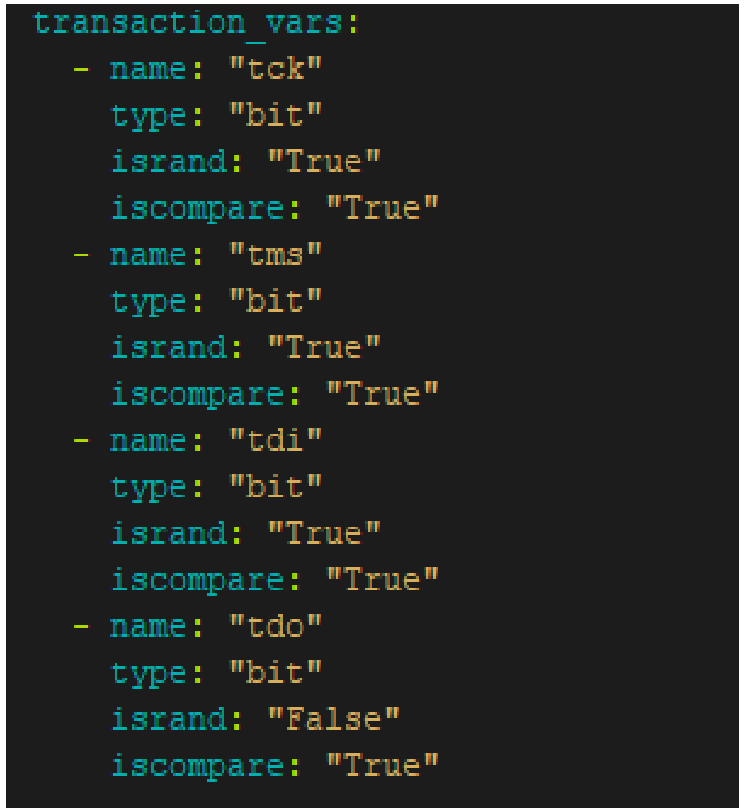


```
ports:
  - name: "tck"
    width: "1"
    dir: "output"
    reset_value: "1'b0"
  - name: "tms"
    width: "1"
    dir: "output"
    reset_value: "1'b0"
  - name: "tdi"
    width: "1"
    dir: "output"
    reset_value: "1'b0"
  - name: "tdo"
    width: "1"
    dir: "input"
    reset_value: "1'b0"
```

JTAG Agent/Interface code generator
jtag_interface.yaml

```
transaction_vars:
      - name: "tck"
        type: "bit"
        isrand: "True"
        iscompare: "True"
      - name: "tms"
        type: "bit"
        isrand: "True"
        iscompare: "True"
```

- Defines any variable to be used by the transaction class.
- Variables in the transaction class reflect the untimed information used during a transfer on the bus
- isrand specifies whether we can randomize or not
- iscompare specifies whether we can compare that variable or not



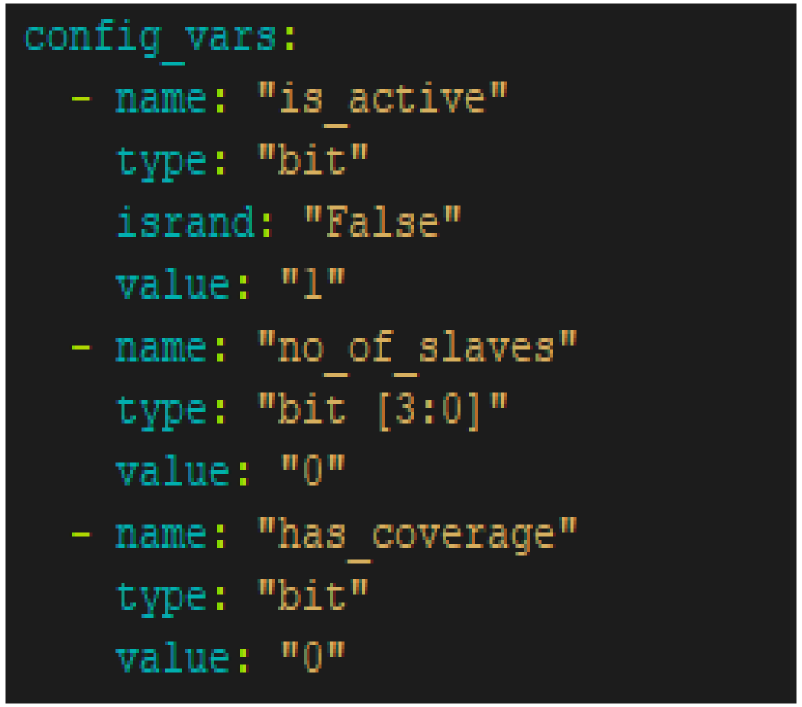
```
transaction_vars:
  - name: "tck"
    type: "bit"
    isrand: "True"
    iscompare: "True"
  - name: "tms"
    type: "bit"
    isrand: "True"
    iscompare: "True"
  - name: "tdi"
    type: "bit"
    isrand: "True"
    iscompare: "True"
  - name: "tdo"
    type: "bit"
    isrand: "False"
    iscompare: "True"
```

JTAG Agent/Interface code generator

jtag_interface.yaml

```
config_vars:
    - name: "is_active"
      type: "bit"
      isrand: "False"
      value: "1"
    - name: "no_of_slaves"
      type: "bit [3:0]"
      value: "0"
    - name: "has_coverage"
      type: "bit"
      value: "1"
```

- Defines any variable to be used by the Config class.

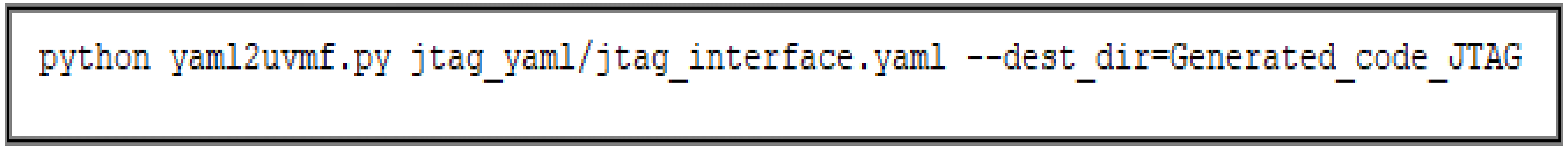


```
config_vars:
  - name: "is_active"
    type: "bit"
    isrand: "False"
    value: "1"
  - name: "no_of_slaves"
    type: "bit [3:0]"
    value: "0"
  - name: "has_coverage"
    type: "bit"
    value: "0"
```

Generating the JTAG Agent/Interface Code

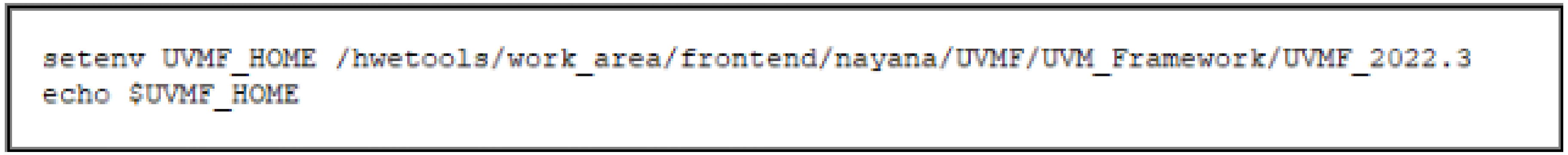
- Command to generate the JTAG Interface code

```
python $UVMF_HOME/scripts/yaml2uvmf.py  ./jtag_yaml/jtag_interface.yaml
```



```
python yaml2uvmf.py jtag_yaml/jtag_interface.yaml --dest_dir=Generated_code_JTAG
```

$UVMF_HOME is a path from the root directory to the present UVMF_2022.3



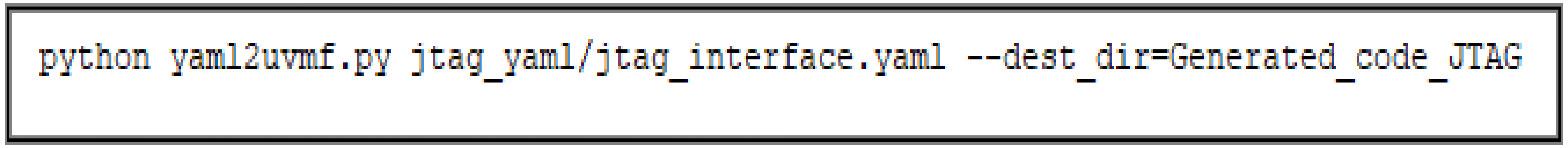
```
setenv UVMF_HOME /hwetools/work_area/frontend/nayana/UVMF/UVM_Framework/UVMF_2022.3
echo $UVMF_HOME
```

➢ By default the generated code will be created under **uvmf_template_output** directory, but we can change the destination directory name using:

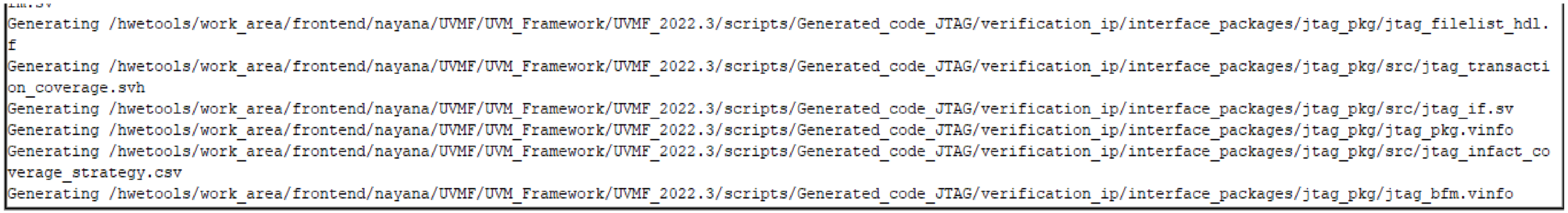
Generating the JTAG Agent/Interface Code

- using command:

  yaml2uvmf.py <filename> --dest_dir=<dir_name>



```
python yaml2uvmf.py jtag_yaml/jtag_interface.yaml --dest_dir=Generated_code_JTAG
```

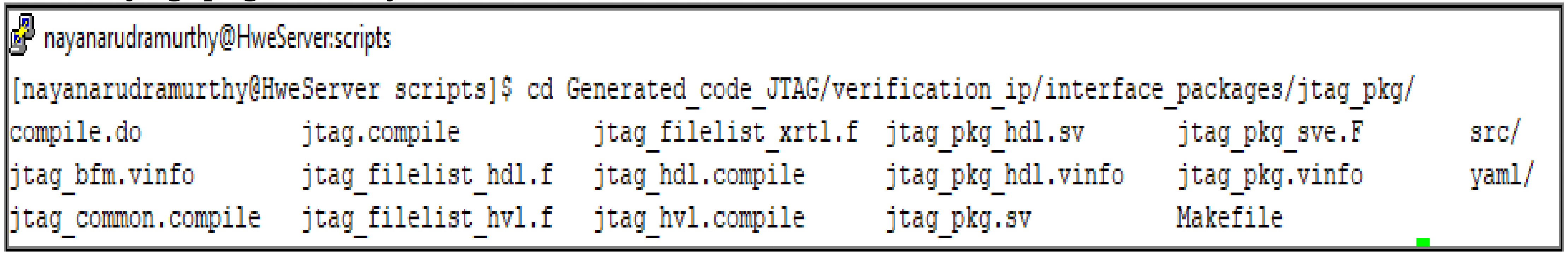


```
Generating /hwetools/work_area/frontend/nayana/UVMF/UVM_Framework/UVMF_2022.3/scripts/Generated_code_JTAG/verification_ip/interface_packages/jtag_pkg/jtag_filelist_hdl.
f
Generating /hwetools/work_area/frontend/nayana/UVMF/UVM_Framework/UVMF_2022.3/scripts/Generated_code_JTAG/verification_ip/interface_packages/jtag_pkg/src/jtag_transacti
on_coverage.svh
Generating /hwetools/work_area/frontend/nayana/UVMF/UVM_Framework/UVMF_2022.3/scripts/Generated_code_JTAG/verification_ip/interface_packages/jtag_pkg/src/jtag_if.sv
Generating /hwetools/work_area/frontend/nayana/UVMF/UVM_Framework/UVMF_2022.3/scripts/Generated_code_JTAG/verification_ip/interface_packages/jtag_pkg/jtag_pkg.vinfo
Generating /hwetools/work_area/frontend/nayana/UVMF/UVM_Framework/UVMF_2022.3/scripts/Generated_code_JTAG/verification_ip/interface_packages/jtag_pkg/src/jtag_infact_co
verage_strategy.csv
Generating /hwetools/work_area/frontend/nayana/UVMF/UVM_Framework/UVMF_2022.3/scripts/Generated_code_JTAG/verification_ip/interface_packages/jtag_pkg/jtag_bfm.vinfo
```

➢ All generated code for the **jtag** agent will be saved under the **Generated_code_JTAG/verification_ip/interface_packages/jtag_pkg** directory

Generating the JTAG Agent/Interface Code

- Inside jtag_pkg directory



```
nayanarudramurthy@HweServer:scripts
[nayanarudramurthy@HweServer scripts]$ cd Generated_code_JTAG/verification_ip/interface_packages/jtag_pkg/
compile.do          jtag.compile        jtag_filelist_xrtl.f  jtag_pkg_hdl.sv     jtag_pkg_sve.F      src/
jtag_bfm.vinfo      jtag_filelist_hdl.f jtag_hdl.compile      jtag_pkg_hdl.vinfo  jtag_pkg.vinfo      yaml/
jtag_common.compile jtag_filelist_hvl.f jtag_hvl.compile      jtag_pkg.sv         Makefile
```

➢ jtag_filelist_hdl.f

Compilation list of hdl files which includes the text files related to interface and the 2 BFMs.

➢ jtag_filelist_hvl.f

Compilation list of hdl files which includes the text files related to all other files.

➢ jtag_pkg.sv

This is the verification package of HVL side that includes all the generated classes for our VIP agent (present id directory src).

Generating the JTAG Agent/Interface Code

- compile.do

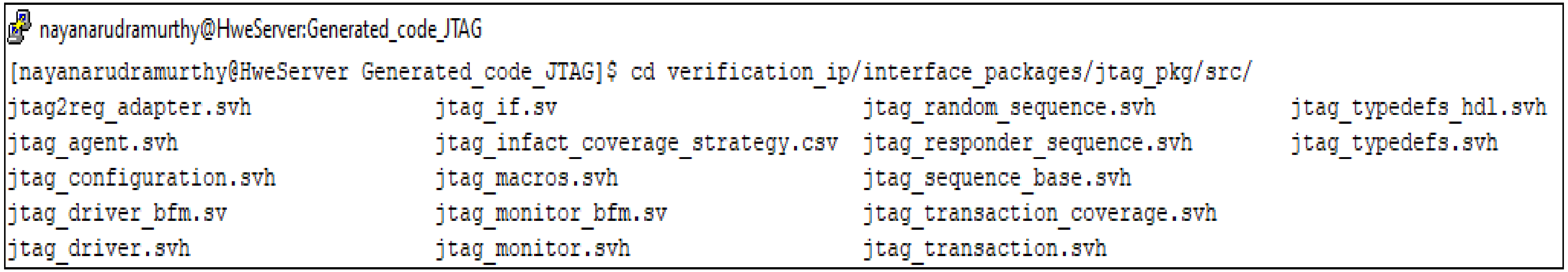
  This file Contains the compile command for the generated agent. This file can be used on Windows or Linux.

- Makefile

  Contains the compile commands for the generated agent. This file can be used on Linux.

Generating the JTAG Agent/Interface Code

- Inside jtag_pkg/src directory



```
[nayanarudramurthy@HweServer Generated_code_JTAG]$ cd verification_ip/interface_packages/jtag_pkg/src/
jtag2reg_adapter.svh          jtag_if.sv                        jtag_random_sequence.svh          jtag_typedefs_hdl.svh
jtag_agent.svh                jtag_infact_coverage_strategy.csv jtag_responder_sequence.svh       jtag_typedefs.svh
jtag_configuration.svh        jtag_macros.svh                   jtag_sequence_base.svh
jtag_driver_bfm.sv            jtag_monitor_bfm.sv               jtag_transaction_coverage.svh
jtag_driver.svh               jtag_monitor.svh                  jtag_transaction.svh
```

➢ jtag_reg_adaptor.svh

This file includes template adaptor for UVM register layer. User has to fill their own functionality into this file.

➢ jtag_agent.svh

Agent class code in uvm.

➢ jtag_configuration.svh

This is agent configuration class.

Generating the JTAG Agent/Interface Code

- jtag_driver.svh

  Driver class to be instantiated in the agent.

- jtag_driver_bfm.svh

  This file has bus functional model to convert transactions to protocol pin wiggles. User has to fill their own functionality into this file.

- jtag_if.sv

  This file contains the signal interface for the agent. User can optionally add protocol assertions in here.

- jtag_macros.sv

  Defines structs type packets and configuration variables which are used to pass data between classes, HVL, BFMs and HDL.

- jtag_monitor.sv

  Monitor class to be instantiated in the agent.

Generating the JTAG Agent/Interface Code

- jtag_monitor_bfm.sv

This file has bus functional model to convert the protocol pin wiggles to transactions. User has to fill their own functionality into this file.

- jtag_random_sequence.svh

This file contains Start sequence. Randomizes the transaction class fields and transfers it to sequencer. This is Extended from jtag_sequence_base

- jtag_responder_sequence.svh

This sequence class can be used to generate stimulus when an interface has been configured to run. User has to fill their own functionality into this file .

- jtag_sequence_base.svh

This is a Base class with useful methods that all inherited sequences has the permission to utilize it.

Generating the JTAG Agent/Interface Code

- jtag_transaction.svh

  This is the sequence_item class 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

- jtag_transaction_coverage.svh

  This class records jtag transaction information using a covergroup named jtag_transaction_coverage.

- jtag_typedefs.svh

  This file contains the defined typedefs which are used in the testbench (HVL) side of the testbench. Package may not contain any defines or typedefs after but will still be generated.

- jtag_typedefs_hdl.svh

  This file contains the defined typedefs used by the interface package performing transaction level simulation activities. This package is used by the driver/monitor BFMs.