# Automation of Simple Block Using YAML

This document helps to understand how an sv code is generated using UVMF automation.

## HOW IT WORKS…

UVMF is the way of automating most of the design verification code. Here we used the file ***yaml2uvmf.py*** which is basically a python file that helps in creating the required files by taking the YAML file as an input. According to the keys and value pairs the inputs are passed to the python script which are already defined.

## BLOCK TO BE IMPLEMENTED

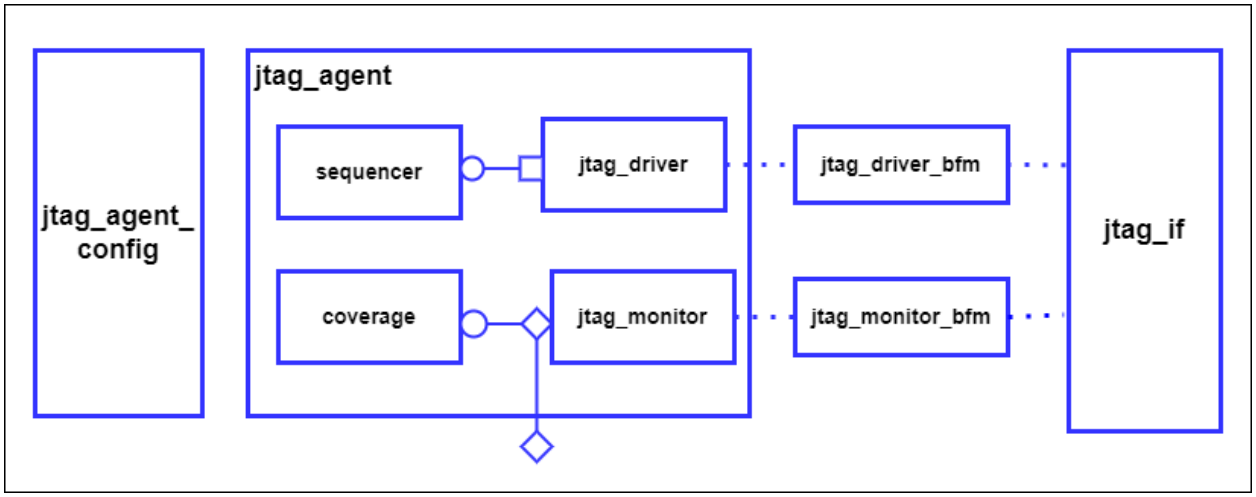


**Fig.1** Implementation Block

### Signals to be generated :

In jtag_if -

- TCK- Test Clock
- TMS- Test Mode Select
- TDI- Test Data In
- TDO- Test Data Out

In agent_config -

- is active
- no. of slaves
- has coverage

## JTAG

Joint Test Action Group provides a pins-out view for testers with every IC pad which helps in identifying any faults within a circuit board. Once this protocol is interfaced to a chip, this can attach a probe to the chip by allowing a developer to control the chip as well as its connections with other chips.

### JTAG INTERFACE SIGNALS

- **TCK**: This is a test clock pin that synchronizes the internal state machine operations in the TAP controller.
- **TDI**: This is Test Data In the pin. This data is shifted into the target device. This pin must be pulled up on a defined condition on the target board.
- **TMS**: This is the Test Mode Select pin that is pulled to determine the next condition of the state machine of the TAP controller which is sampled at the rising edge of TCK.
- **TDO**: This is the Test Data Out pin, so the data is moved out of the target device and it is valid on the falling edge of TCK.

## YAML

YAML is a simple data serialization language, used to create configuration files with any programming language. To create files for the given block, interface file is enough as Interface YAML structure creates all the files like interface,driver bfm, monitor bfm, driver proxy ,monitor proxy, agent, agent config, coverage etc.

# GENERATE CODE USING YAML

- Create a YAML file **jtag_if.yaml**
- Run the command
  **$ python yaml2uvmf.py jtag_intf.yaml --dest_dir=output**
- Reading the output, go to “dest_dir”. Here, it’s **output**

### Create a YAML file

According to the requirement add respective keys and values as shown. Below is the interface code i.e., **jtag_if.yaml**

```yaml
uvmf:
  interfaces:
    "jtag":
      # The name of the clock & reset are the only required
      # elements for each interface
      clock: "clock"
      reset: "reset"
      # Reset assertion level specified as True will make this
      # interface sensitive to an active HIGH reset
      reset_assertion_level: "True"
      veloce_ready: "True"
      ports:
        - name: "TCK"
          width: "1"
          dir: "input"
          #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"
```

```yaml
      transaction_vars:
        - name: "T_clock"
          type: "bit"
          isrand: "False"
          #iscompare: "False"
        - name: "T_MS"
          type: "bit"
          isrand: "True"
          #iscompare: "True"
        - name: "T_datain"
          type: "bit"
          isrand: "True"
          #iscompare: "True"
        - name: "T_dataout"
          type: "bit"
          isrand: "False"
            #set to true if the exists function is being checked
          #iscompare: "False"
      config_vars:
        - name: "isactive"
          type: "bit"
          isrand: "True"
          value: "1'b1"
        - name: "noofslaves"
          type: "int"
          isrand: "True"
          value: "1"
        - name: "hascoverage"
          type: "bit"
          isrand: "True"
          value: "1'b1"
      config_constraints:
        - name: "slavecount"
          value: "{ noofslaves>0;noofslaves<17 }"
```

**Fig.2** Interface yaml code

- uvmf specifies that the main key is **uvmf,** under which the sub key-value pairs are present.
- “jtag” is the interface name.
- clock and reset are system generated signals which are necessary.
- As the required interface signals are TCK,TMS,TDI and TDO they are written under “ports” key with their own width, direction and name as sub-keys.
- “Transaction_vars” is the key under which the transaction class properties that replicate the interface signals.
- “config_vars” are for agent configuration variables “isactive”,”hascoverage”,”noofslaves”.
- “config_constraints” are constraints for the config_vars.

## Command

**$ python yaml2uvmf.py jtag_intf.yaml --dest_dir=output**

Here, **yaml2uvmf.py** is the actual converter that converts the input yaml file **jtag_intf.yaml.**

After running the command you can see a directory generated with name “output” in which the following files are being generated. And it is being created in the **current directory where you are present**.

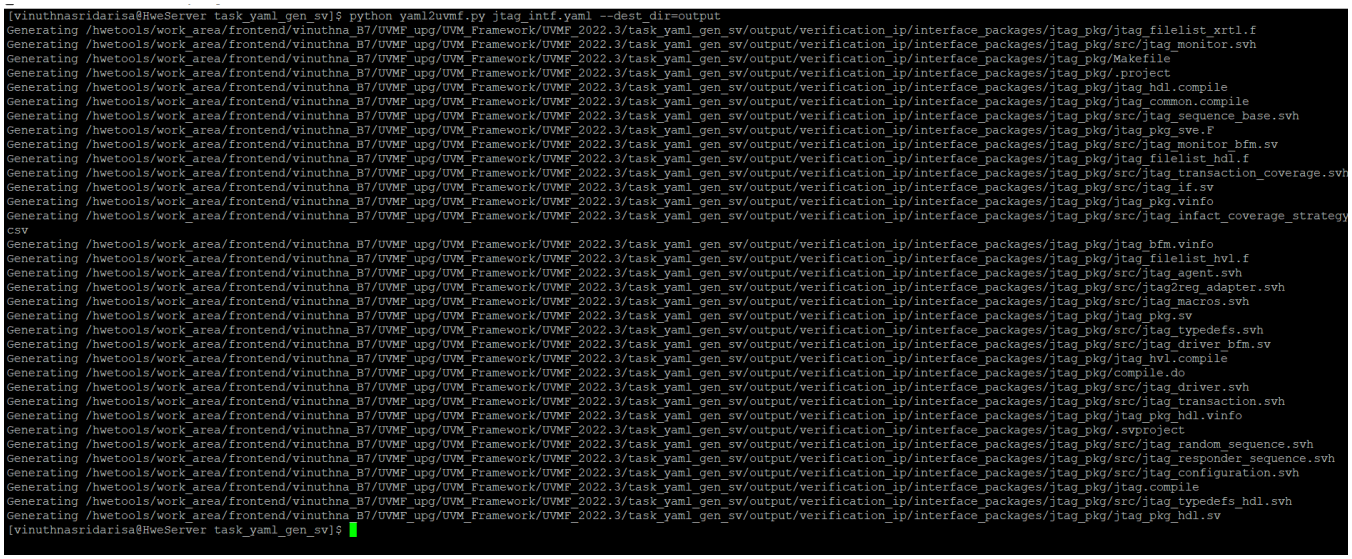


**Fig.3** Running the command to generate files

## Reading the Files

Under the path **output/verification_ip/interface_packages/jtag_pkg/src** the files reside,there are some other files that will be created with more info on the verification ip.
The detailed YAML file resides in **output/verification_ip/interface_packages/jtag_pkg/yaml.**

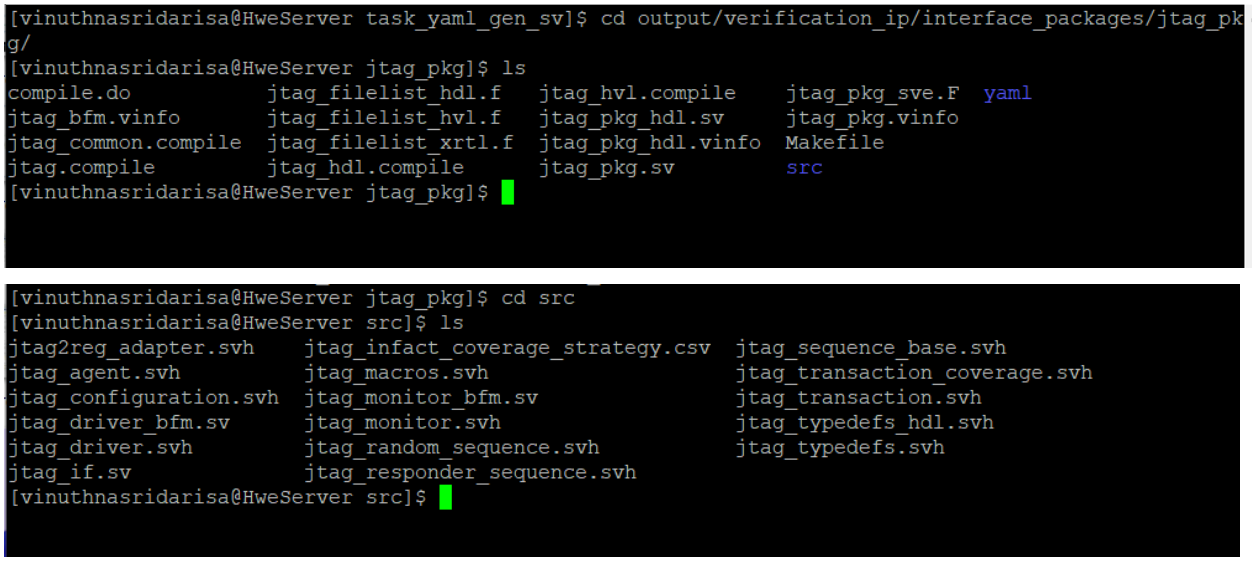


**Fig.4** Files created

## **Useful Points** :

- The file names will be created with the name of interface that we give in YAML file.
- The file name justifies the use of their creation as jtag_agent.svh is the agent base class file.

## **REFERENCES**


- UVMF_Code_Generator_YAML_Reference.pdf
- https://verificationacademy.com/sessions/uvmf-interface-code-generation