UVM - Day25

-: UVM RAL (Register Abstraction Layer) :-

The UVM Register Layer provides a standard base class libraries that enable users to implement the object-oriented model to access the DUT registers and memories.

- UVM Register layers is also reffered to as UVM Register Abstraction Layer (UVM RAL).

UVM RAL Model ⇛
RAL blocks Contain,
- registers
- register files
- memories and other blocks.

Register Model generator ⇛
Register model generators are outside the scope of the UVM library.

- A register model can be written as a register generator application. Writing or Generating the register model is based on a design register specification.

RAL Building blocks ⇛
(1) Register block :-
The reg block is written by extending the uvm_reg_block.
- A register model is an instance of a register block, which may contain any number of register files, memories and other blocks.

(ii) Register file :-
The reg file is written by extending the uvm_reg_file.

- The reg file shall be used to group the number of registers or register files.

(iii) Register :-
The uvm register class is written by extending the uvm_reg.
- A registers represents a set of fields that are accessible as a single entity.
- Each register contains any number of fields, which mirror the values of the corresponding elements.

(iv) Register Field :-
The register field is declared with the type uvm_reg_field.

- Fields represent a contiguous set of bits. All data values are modeled as fields. A fields is contained within a single register but may have different access policies.

UVM RAL Methods ⇒
UVM RAL library classes have builtin methods implemented in it, these methods can be used for accessing the registers.

- These methods are reffered to as Register Access Methods.

The register model has methods to read, write, update and mirror DUT registers and register field values, these methods are called API (Application Programming Interface).

- APIs can either use front door access or back door access to DUT registers and register fields.

- Front door access involves using the bus interface and it is associated with the timing.

- Back door access uses simulator database access routines and this happens in 0 simulator time.

API Methods :—

- read and write ⇒
read() returns and updates the value of the DUT register
write() writes and updates the value of the DUT register
- Both read and write can be used for front door or back door access.
- In read or write value will be updated by the bus predictor on completion of the front door read or write cycle and automatically in back door read or write cycle.

- Peek and poke ⇒
peek() reads the DUT register value using a backdoor.
poke() writes a value to DUT register using backdoor.

- set and get ⇒
set() and get() writes and reads directly to the desired value.

M T W T F S S
Page No.:
Date:
YOUVA

- set and get methods operates on the register model desired value, not accesses to DUT register value. The desired value can be updated to the DUT using the update method.

update ⇛
if there is difference b/w desired value and mirrored value, update() will initiate a write to register. update() method can be used after the set method.

mirror ⇛
mirror() reads the updated DUT register values. The mirroring can be performed in the front door or back door (peek()).

randomize ⇛
randomize() randomizes register or field values with or without constraints. as per the requirement register values can be modified in post_randomize().
- ~~After~~ After randomization update() can be used to update the DUT register values.

reset ⇛
reset() sets the register desired and mirrored value to the pre-defined reset value.