A 360 Degree View of UVM Events

Vikas Billa
vikas.billa@intel.com
SoC Verification Engineer
Intel India Pvt. Ltd

ABSTRACT

❖UVM saw a great adoption in the past few years and has been accepted as the most popular methodology choice by the verification community. Since its inception, the community and its users have been constantly adding and enhancing its library, thereby offering rich and flexible data structure choices to the test-bench developers.

❖The UVM events has seen a tremendous enhancement over few years due to its widespread applications and usage.

❖This paper attempts to give a 360-degree view of uvm events with use cases.

SYSTEMVERILOG EVENTS

❖ A SystemVerilog event is a data type which has no storage. An identifier declared with data type event is called a named event. As represented in Figure 1, named event can be triggered using the "->" operator and an event triggering occurrence can be captured by using "@" operator or inbuilt .triggered method. The named event and event control provides the communication and synchronization between two or more concurrent process.

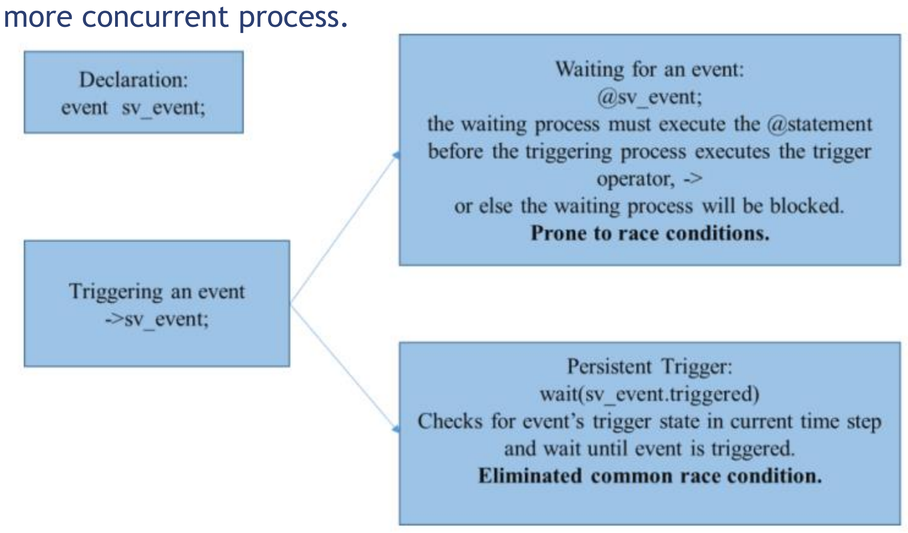

Figure 1: System Verilog Events

UVM EVENTS

❖uvm_event is a parametrized wrapper class created using the System Verilog event construct. It provides some additional services such as setting call-backs, data delivery, and maintaining number of waiters, on-off state and timing information.

```
class uvm_event#(type T=uvm_object) extends uvm_event_base;
```

❖The uvm_event class is an extension of the abstract uvm_event_base class, the hierarchy of uvm_event in UVM class library is shown below:

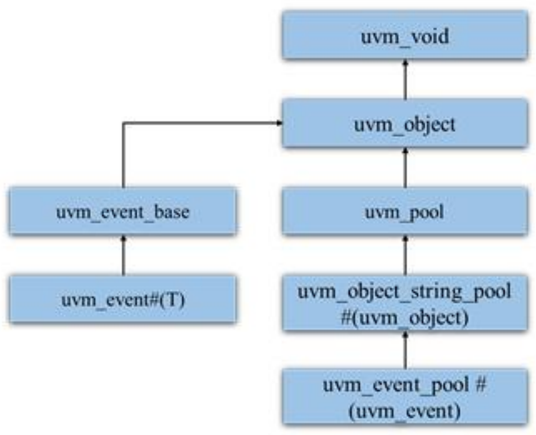

Figure 2: UVM Event Hierarchy

UVM EVENTS: RAL

❖Register Abstraction Layer is one of the popular feature in UVM. UVM_RAL provides some standard callback methods like pre_write, pre_read, post_write, post_read, post_predict, encode and decode. By using these callback methods the desired functionality can be added at the required position during register access.

❖uvm_event can be used for communication from register callback class to other components like scoreboard,reference model, sequences etc. Adopting the uvm_event for RAL to TB communication requires minimal testbench code changes and also the data delivery through uvm_event helps a great deal especially in score-boarding and reference modeling.

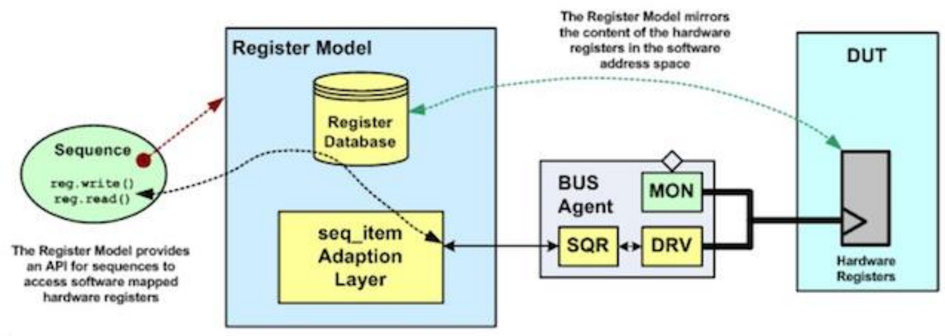

Figure 3: UVM RAL

UVM EVENTS: A Reset aware Testbench

❖ Handling on-the-fly reset is one of the challenges in testbench design. The UVM methodology does not define how an on-the-fly reset must be handled. The reset is a major disruptive event which can occur at any point of time, it's very important to ensure that the chip exists out of reset and resumes normal operation without any issue.

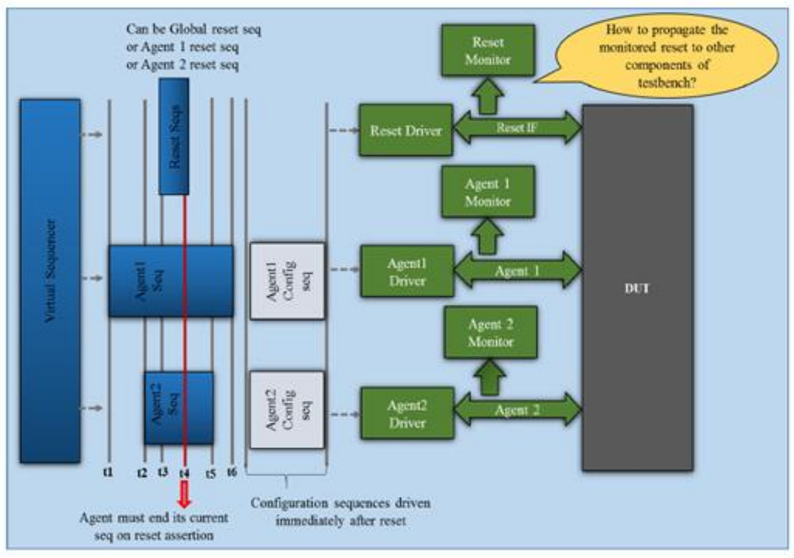

Figure 4 : Rest aware Testbench

UVM EVENTS: Interrupts

❖ Interrupt handling is a must have feature of any verification environment. Interrupt is an event that is triggered by a Design or an IP block once certain conditions are fulfilled; the CPU has to service these events. The actions that have to be taken by CPU while servicing the interrupt is collectively called as Interrupt Service Routines (ISR).

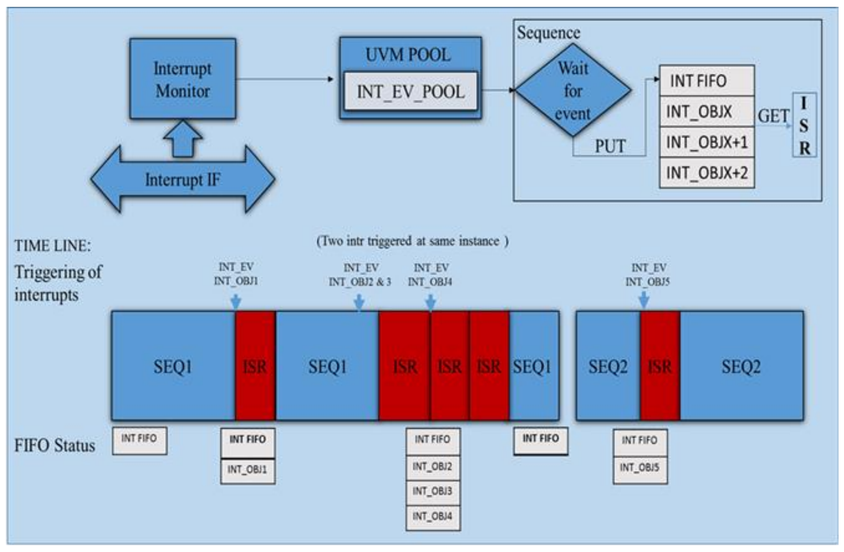

Figure 5 : Interrupt Mechanism using UVM Events

CONCLUSION

The uvm_event and event pool provides synchronization between multiple threads or concurrent processes in the verification environment. It is observed that by using uvm_event we can achieve better synchronization without making major changes to the exiting testbench.

The uvm_event wrapper class around the traditional systemverilog event with added methods make uvm_event to be applicable to a broader and complex application than just synchronization.