

Semaphore: A Crucial Entity in Verification

- By Milan Pujara, ASIC Engineer, einfochips

Verification of any ASIC/Soc includes mainly creation of heavy traffic on chip interfaces to find any hidden corner bugs either on chip interfaces or deep inside the chip. To achieve it, it is very important to share common resources (like bus functional model between multiple higher level drivers), to synchronize between two different entities in the testbench etc.

Problem Overview To create an effective testbench, many times we need an element to work as either common resource allocator or synchronizing element between two entities.

Solution Overview Instead of developing our own algorithm in the testbench like round robin algorithm for common resource allocation or local storage and access to local storage in the testbench for synchronization, the best approach is to use semaphore to serve these algorithms.

Key Topics Semaphore is a crucial entity which serves all these purposes to create effective testbench to generate heavy traffic on chip interfaces.

Let us see the usage of semaphore as

- Common resource allocator called mutex,
- Synchronizing element between producer and consumer.

Common Resource allocator: Mutex

The best example for common resource allocation is between well-known testbench components like driver/generator and bus functional models. To create stimulus and drive it on any chip interface, the verification engineer needs to develop basic components like drivers/generators and bus functional models (BFM) respectively.

Semaphore as a common resource allocator can be used between multiple drivers and single bus functional models as shown in below diagram.

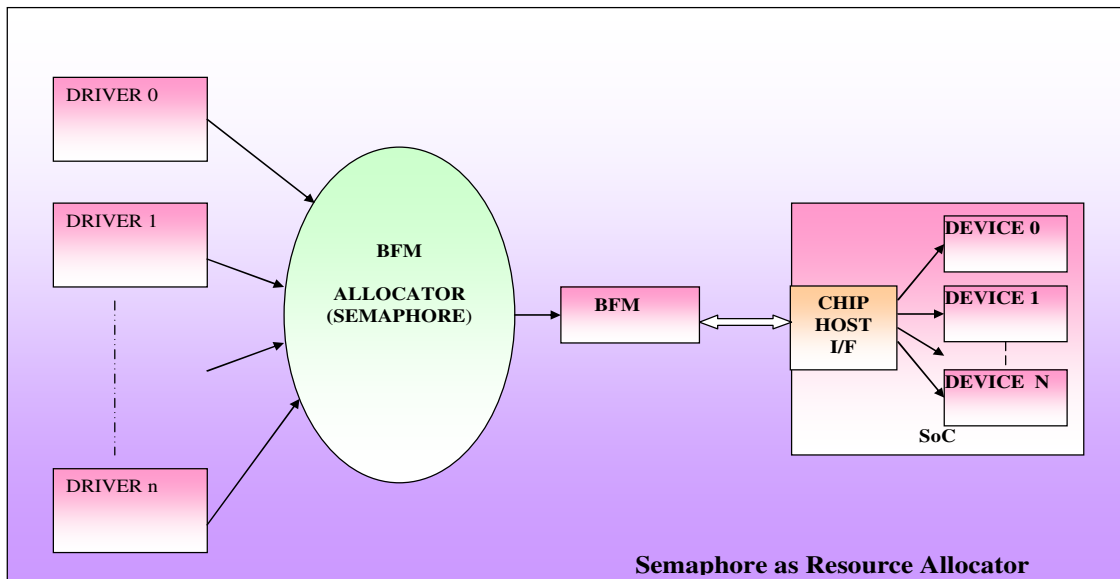


Figure 1. Semaphore as a common resource allocator

As shown in above diagram, generally any SoC might have multiple devices like memory, DMA, processor etc residing in it and chip interface like MDIO, PCI etc. to configure this devices. This kind of interfaces are generally called host interface.

The ideal testbench for this kind of Soc on host interface is to have driver for each device to configure device or read the status of each device dynamically, bus functional model to perform packet transfer or read/write transaction based on protocol and one BFM allocator to allocate one BFM to multiple driver at a time. Here, we need a mutual exclusion for using BFM between multiple drivers. Semaphore can be used to have a mutual exclusion (called mutex) as shown below.

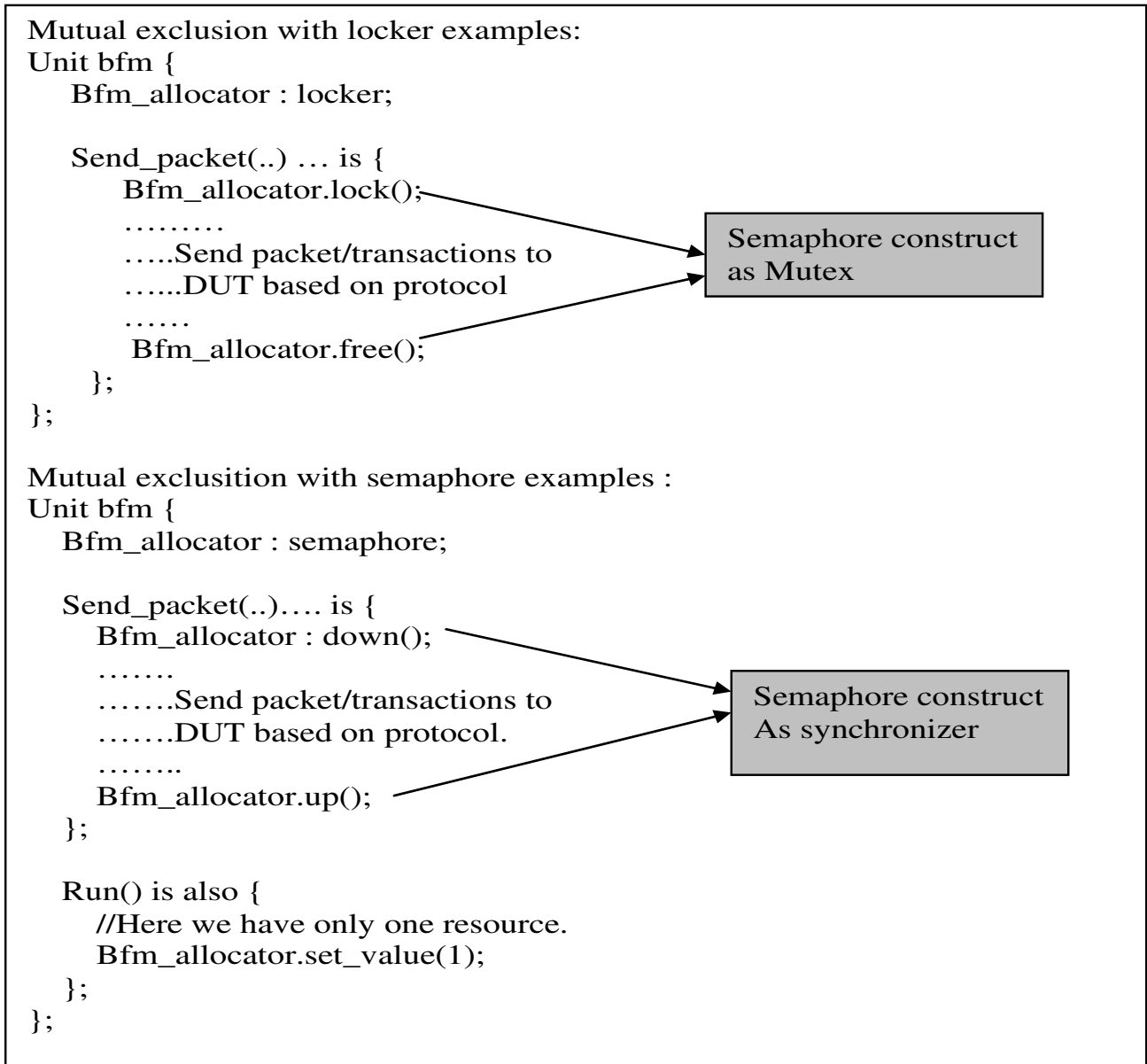


Figure 2 Semaphore as mutex/synchronizer

Synchronizing element between producer and consumer

We use semaphore as a synchronizing element in developing testbench in one or other ways. Those who are familiar to eRM methodology to develop testbench in specman might have used `try_next_item /get_next_item` as a synchronizing element between producer and consumer. Those who have used RVM methodology to develop testbench in specman might have used channel concept to have a proper synchronization between producer and consumer.

Let us understand semaphore functionality in rvm_channel used in vera's RVM methodology :

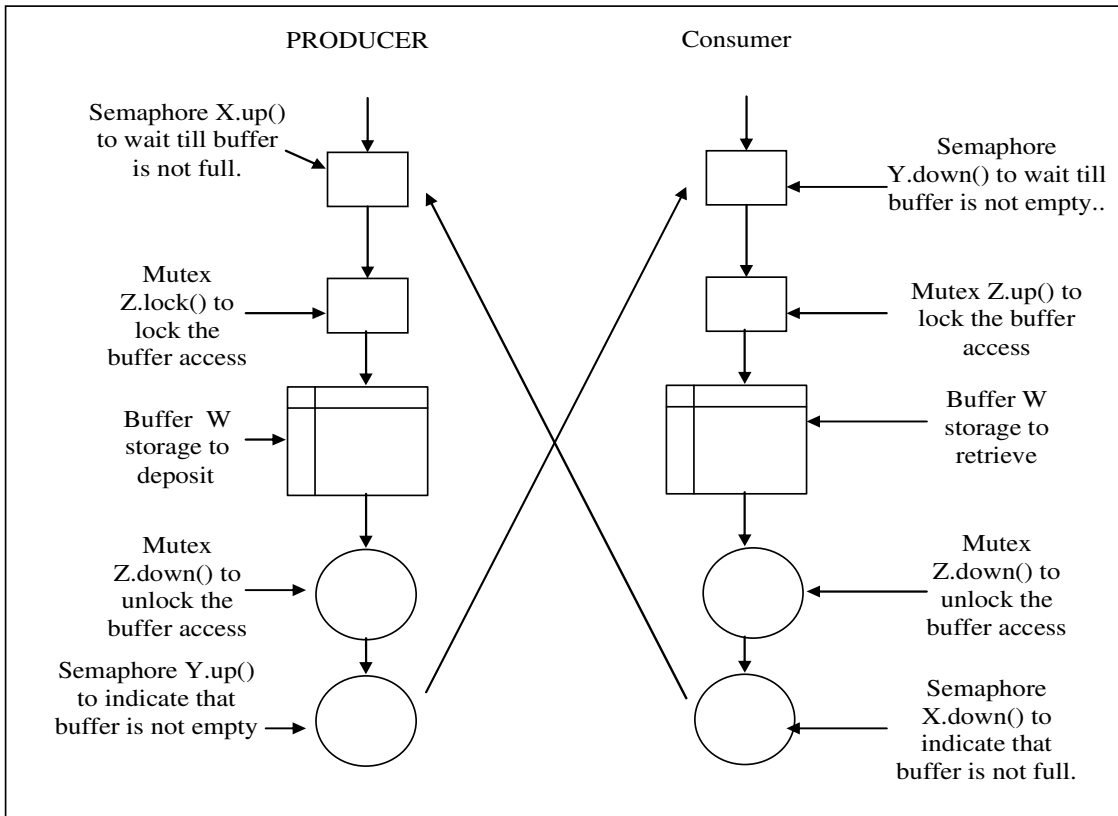


Figure 3 rvm_channel implementation using Semaphore

As shown in above figure, we need to have two semaphores X and Y and one mutex Z and one buffer W.

Semaphore X and Y are used to synchronize between buffer full and empty condition respectively and mutex Z (binary semaphore) is used to allow access to buffer one at a time between producer and consumer.

Conclusion : Semaphore is very crucial element in developing harness to create heavy traffic on any interface by behaving as a common resource allocator (mutex) and synchronizing element between producer and consumer by behaving as a semaphore. As a verification engineer, it is always important to use semaphore to cover all possible exhaustively exercise chip interfaces.