DAY-67
#100DAYSRTL

"System Verilog:- IPC: Semaphore"

"Introduction":-

We have seen that in System Verilog various processes can run in parallel. This can create a problem if 2 processes access some variables. Semaphore provides a solution to this problem. semaphores can also help in synchronization of different processes mainly used in test bench design where different components of the test bench need to be synchronized with each other.

"Semaphore":-

Semaphores are basically a way to prevent unintended access to shared variables by different processes. This can be considered as a bucket having keys. The process will first need to get a key from this bucket and then only it can continue to execute the remaining code lines. If there are no keys left in the bucket, the process will have to wait until a key is back in the bucket.

```
semaphore sem;
 sem = new();
```

"Methods of Semaphore":-

1. **new(int key count = 0)-** This method creates a new handle of semaphore. This handle will be used to get or put keys back into the semaphore. The number of keys stored by semaphore can be passed as an argument.
2. **get(int keyCount = 1)-** This method gets the number of keys passed as an argument from the semaphore. If the mentioned

number of keys is not present in the semaphore then it will wait for the specified number of keys to be available in the semaphore.

3. **try_get(int keyCount = 1) -** this is a non-blocking function that returns 0 if the number of keys is not available it will return 1. This will not wait for the number of keys to be available.
4. **put(int key count = 1) -** this is a task that will put the specified number of keys back into the semaphore. This is also a blocking method that will block the execution of further statements until the specified number of keys are returned to the semaphore.

Code practicing:-

```
module tb;
   semaphore key;
   task getRoom (bit [1:0] id);
      $display ("[%0t] Trying to get a room for id[%0d] ...", $time, id);
      key.get (1);
      $display ("[%0t] Room Key retrieved for id[%0d]", $time, id);
   endtask
   task putRoom (bit [1:0] id);
      $display ("[%0t] Leaving room id[%0d] ...", $time, id);
      key.put (1);
      $display ("[%0t] Room Key put back id[%0d]", $time, id);
   endtask
   task personA ();
      getRoom (1);
      #10 putRoom (1);
   endtask
   task personB ();
      getRoom (2);
      #10 putRoom (2);
   endtask
   initial begin
      key = new (1);
      fork
         personA ();
         personB ();
      join_none
   end
endmodule
```

Result:-

```
[0] Trying to get a room for id[1] ...
[0] Room Key retrieved for id[1]
[0] Trying to get a room for id[2] ...
[10] Leaving room id[1] ...
[10] Room Key put back id[1]
[10] Room Key retrieved for id[2]
[20] Leaving room id[2] ...
[20] Room Key put back id[2]
Simulation has finished. There are no more test vectors to simulate.
```

Note:-

In semaphore, the no of keys given inside the new is just the initial no of keys. More no of keys can be added to the semaphore pushing put() task. If the no of keys inside put() is more than what we have removed from the semaphore the total no of keys gets increased as seen in the below code.

Code practicing:-

```
module sem_example_2;
  semaphore sema;
  task method1;
    sema.get(4);
        $display("[%0t] Method 1 Got control", $time);
        #30;
    sema.put(5);
    endtask
    task method2;
      sema.get(1);
        $display("[%0t] Method 2 Got control", $time);
        #50;
        sema.put(4);
    endtask
    task method3;
        if(sema.try_get(4))
            $display("[%0t] Method 3 Got control", $time);
        else
            $display("[%0t] Method 3 failed to get control", $time);
        #50;
        sema.put(4);
    endtask
    initial begin
        sema = new(1);
        fork
            method1;
            method2;
            method3;
        join
    end
endmodule
```

Result:-

```
[0] Method 2 Got control
[0] Method 3 failed to get control
[50] Method 1 Got control
Simulation has finished. There are no more test vectors to simulate.
```