Module: SV for Verification
Section: Threads & Interprocess Communication **Task:** Wait Fork

Task 4

Wait Fork

- **Code:**

```
module fork_wait ();

  initial begin
    fork
        begin
          $display("Thread 1 Task Starting  @ %0t",$time);
          #30;
          $display("Thread 1 Task Finished  @ %0t",$time);
        end

        begin
          $display("Thread 2 Task Starting  @ %0t",$time);
          #15;
          $display("Thread 2 Task Finished  @ %0t",$time);
        end
    join_none

    //Add code here to wait for all forked threads to finish
    wait fork;

    #5;
    $display("Program Finished @ %0t",$time);
    $finish;
  end
endmodule
```

- **Before Wait Fork:**
  - **Output:**

```
Log   Share
Contains Synopsys proprietary information.
Compiler version U-2023.03-SP2_Full64; Runtime version U-2023.03-SP2_Full64;  Sep  7 02:21 2024
Thread 1 Task Starting  @ 0
Thread 2 Task Starting  @ 0
Program Finished @ 5
$finish called from file "testbench.sv", line 23.
$finish at simulation time                   5
           V C S   S i m u l a t i o n   R e p o r t
Time: 5 ns
CPU Time:      0.470 seconds;       Data structure size:   0.0Mb
Sat Sep  7 02:21:27 2024
Done
```

- **After Wait Fork:**
  - **Output:**

```
Log   Share
Compiler version U-2023.03-SP2_Full64; Runtime version U-2023.03-SP2_Full64;  Sep  7 02:27 2024
Thread 1 Task Starting  @ 0
Thread 2 Task Starting  @ 0
Thread 2 Task Finished  @ 15
Thread 1 Task Finished  @ 30
Program Finished @ 35
$finish called from file "testbench.sv", line 23.
$finish at simulation time                  35
           V C S   S i m u l a t i o n   R e p o r t
Time: 35 ns
CPU Time:      0.430 seconds;       Data structure size:   0.0Mb
Sat Sep  7 02:27:28 2024
```

- **Observation:**
  With the **wait fork;** statement, the program waits for all forked threads to complete before proceeding, ensuring that both threads finish execution before the final output is printed. In contrast, without **wait fork;**, the program moves forward immediately after the fork begins, potentially finishing before the threads complete. This leads to premature termination, with the final output being printed before either thread fully executes. In essence, **wait fork;** ensures proper synchronization, while its absence can cause incomplete execution of parallel tasks.