**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **4 June 2020** | **Name:** | **Shreya poojary** |
| **Course:** | **HDL design** | **USN:** | **4al16ec074** |
| **Topic:** | **Hardware modelling using Verilog**  **FPGA and ASIC Interview questions** | **Semester & Section:** | **8- B** |
| **Github Repository:** | **shreya-test** |  |  |

|  |
| --- |
|  |
| **Image of session** |
| **Report**  **Verilog code to swap contents of two registers with and without a temporary register**  With temp reg ;  always @ (posedge clock)  begin  temp=b;  b=a;  a=temp;  end  Without temp reg;  always @ (posedge clock)  begin  a <= b;  b <= a;  end  **Difference between task and function**  Function:  A function is unable to enable a task however functions can enable other functions.  A function will carry out its required duty in zero simulation time. ( The program time will not be incremented during the function routine)  Within a function, no event, delay or timing control statements are permitted  In the invocation of a function their must be at least one argument to be passed.  Functions will only return a single value and can not use either output or inout statements.  Tasks:  Tasks are capable of enabling a function as well as enabling other versions of a Task  Tasks also run with a zero simulation however they can if required be executed in a non zero simulation time.  Tasks are allowed to contain any of these statements.  A task is allowed to use zero or more arguments which are of type output, input or inout.  A Task is unable to return a value but has the facility to pass multiple values via the output and inout statements .  **The following section explains clock domain interfacing**  One of the biggest challenges of system-on-chip (SOC) designs is that different blocks operate on independent clocks. Integrating these blocks via the processor bus, memory ports, peripheral busses, and other interfaces can be troublesome because unpredictable behavior can result when the asynchronous interfaces are not properly synchronized  A very common and robust method for synchronizing multiple data signals is a handshake technique as shown in diagram below This is popular because the handshake technique can easily manage changes in clock frequencies, while minimizing latency at the crossing. However, handshake logic is significantly more complex than standard synchronization structures.   FSM1(Transmitter) asserts the req (request) signal, asking the receiver to accept the data on the data bus. FSM2(Receiver) generally a slow module asserts the ack (acknowledge) signal, signifying that it has accepted the data.  it has loop holes: when system Receiver samples the systems Transmitter req line and Transmitter samples system Receiver ack line, they have done it with respect to their internal clock, so there will be setup and hold time violation. To avoid this we go for double or triple stage synchronizers, which increase the MTBF and thus are immune to metastability to a good extent. The figure below shows how this is done. |