**DAILY ASSESSMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **04/06/2020** | **Name:** | **Dhavala** |
| **Course:** | **DIGITAL DESIGN USING HDL** | **USN:** | **4AL17EC027** |
| **Topic:** | * **Hardware modelling using verilog** * **FPGA and ASIC Interview questions** | **Semester & Section:** | **6TH SEM & A Section** |
| **Github Repository:** | **Dhavala27** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
|  |

|  |
| --- |
| **Report** |
| **1) Write a 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  **2) Difference between blocking and non-blocking?**(Verilog interview questions that is most commonly asked)  The Verilog language has two forms of the procedural assignment statement: blocking and non-blocking. The two are distinguished by the = and <= assignment operators. The blocking assignment statement (= operator) acts much like in traditional programming languages. The whole statement is done before control passes on to the next statement. The non-blocking (<= operator) evaluates all the right-hand sides for the current time unit and assigns the left-hand sides at the end of the time unit. For example, the following Verilog program  // testing blocking and non-blocking assignment  module blocking; reg [0:7] A, B; initial begin: init1 A = 3; #1 A = A + 1; // blocking procedural assignment B = A + 1;  $display("Blocking: A= %b B= %b", A, B ); A = 3; #1 A <= A + 1; // non-blocking procedural assignment B <= A + 1; #1 $display("Non-blocking: A= %b B= %b", A, B ); end endmodule  produces the following output: Blocking: A= 00000100 B= 00000101 Non-blocking: A= 00000100 B= 00000100  The effect is for all the non-blocking assignments to use the old values of the variables at the beginning of the current time unit and to assign the registers new values at the end of the current time unit. This reflects how register transfers occur in some hardware systems. blocking procedural assignment is used for combinational logic and non-blocking procedural assignment for sequential  **3) 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 .  **4) Difference between inter statement and intra statement delay?**  //define register variables reg a, b, c;  //intra assignment delays initial begin a = 0; c = 0; b = #5 a + c; //Take value of a and c at the time=0, evaluate //a + c and then wait 5 time units to assign value //to b. end  //Equivalent method with temporary variables and regular delay control initial begin a = 0; c = 0; temp\_ac = a + c; #5 b = temp\_ac; //Take value of a + c at the current time and //store it in a temporary variable. Even though a and c //might change between 0 and 5, //the value assigned to b at time 5 is unaffected. end   **6) Difference between $monitor,$display & $strobe?**  These commands have the same syntax, and display text on the screen during simulation. They are much less convenient than waveform display tools like cwaves?. $display and $strobe display once every time they are executed, whereas $monitor displays every time one of its parameters changes. The difference between $display and $strobe is that $strobe displays the parameters at the very end of the current simulation time unit rather than exactly where it is executed. The format string is like that in C/C++, and may contain format characters. Format characters include %d (decimal), %h (hexadecimal), %b (binary), %c (character), %s (string) and %t (time), %m (hierarchy level). %5d, %5b etc. would give exactly 5 spaces for the number instead of the space needed. Append b, h, o to the task name to change default format to binary, octal or hexadecimal. Syntax: $display (“format\_string”, par\_1, par\_2, ... ); $strobe (“format\_string”, par\_1, par\_2, ... ); $monitor (“format\_string”, par\_1, par\_2, ... );   **7) What is difference between Verilog full case and parallel case?**  A "full" case statement is a case statement in which all possible case-expression binary patterns can be matched to a case item or to a case default. If a case statement does not include a case default and if it is possible to find a binary case expression that does not match any of the defined case items, the case statement is not "full." A "parallel" case statement is a case statement in which it is only possible to match a case expression to one and only one case item. If it is possible to find a case expression that would match more than one case item, the matching case items are called "overlapping" case items and the case statement is not "parallel."  **8) What is meant by inferring latches,how to avoid it?**  Consider the following : always @(s1 or s0 or i0 or i1 or i2 or i3) case ({s1, s0}) 2'd0 : out = i0; 2'd1 : out = i1; 2'd2 : out = i2; endcase  in a case statement if all the possible combinations are not compared and default is also not specified like in example above a latch will be inferred ,a latch is inferred because to reproduce the previous value when unknown branch is specified. For example in above case if {s1,s0}=3 , the previous stored value is reproduced for this storing a latch is inferred. The same may be observed in IF statement in case an ELSE IF is not specified. To avoid inferring latches make sure that all the cases are mentioned if not default condition is provided.  **9) Tell me how blocking and non blocking statements get executed?**  Execution of blocking assignments can be viewed as a one-step process: 1. Evaluate the RHS (right-hand side equation) and update the LHS (left-hand side expression) of the blocking assignment without interruption from any other Verilog statement. A blocking assignment "blocks" trailing assignments in the same always block from occurring until after the current assignment has been completed  Execution of nonblocking assignments can be viewed as a two-step process: 1. Evaluate the RHS of nonblocking statements at the beginning of the time step. 2. Update the LHS of nonblocking statements at the end of the time step.  **10) Variable and signal which will be Updated first?**  Signals  **11) What is sensitivity list?**  The sensitivity list indicates that when a change occurs to any one of elements in the list change, begin…end statement inside that always block will get executed.  **12) In a pure combinational circuit is it necessary to mention all the inputs in sensitivity disk?** if yes, why?  Yes in a pure combinational circuit is it necessary to mention all the inputs in sensitivity disk other wise it will result in pre and post synthesis mismatch.  **13) Tell me structure of Verilog code you follow?**  A good template for your Verilog file is shown below.  // timescale directive tells the simulator the base units and precision of the simulation `timescale 1 ns / 10 ps module name (input and outputs); // parameter declarations parameter parameter\_name = parameter value; // Input output declarations input in1; input in2; // single bit inputs output [msb:lsb] out; // a bus output // internal signal register type declaration - register types (only assigned within always statements). reg register variable 1; reg [msb:lsb] register variable 2; // internal signal. net type declaration - (only assigned outside always statements) wire net variable 1; // hierarchy - instantiating another module reference name instance name ( .pin1 (net1), .pin2 (net2), . .pinn (netn) ); // synchronous procedures always @ (posedge clock) begin . end // combinatinal procedures always @ (signal1 or signal2 or signal3) begin . end assign net variable = combinational logic; endmodule  **14) Difference between Verilog and vhdl?**  Compilation VHDL. Multiple design-units (entity/architecture pairs), that reside in the same system file, may be separately compiled if so desired. However, it is good design practice to keep each design unit in it's own system file in which case separate compilation should not be an issue.   Verilog. The Verilog language is still rooted in it's native interpretative mode. Compilation is a means of speeding up simulation, but has not changed the original nature of the language. As a result care must be taken with both the compilation order of code written in a single file and the compilation order of multiple files. Simulation results can change by simply changing the order of compilation.   Data types VHDL. A multitude of language or user defined data types can be used. This may mean dedicated conversion functions are needed to convert objects from one type to another. The choice of which data types to use should be considered wisely, especially enumerated (abstract) data types. This will make models easier to write, clearer to read and avoid unnecessary conversion functions that can clutter the code. VHDL may be preferred because it allows a multitude of language or user defined data types to be used.  Verilog. Compared to VHDL, Verilog data types a re very simple, easy to use and very much geared towards modeling hardware structure as opposed to abstract hardware modeling. Unlike VHDL, all data types used in a Verilog model are defined by the Verilog language and not by the user. There are net data types, for example wire, and a register data type called reg. A model with a signal whose type is one of the net data types has a corresponding electrical wire in the implied modeled circuit. Objects, that is signals, of type reg hold their value over simulation delta cycles and should not be confused with the modeling of a hardware register. Verilog may be preferred because of it's simplicity.  Design reusability VHDL. Procedures and functions may be placed in a package so that they are avail able to any design-unit that wishes to use them.  Verilog. There is no concept of packages in Verilog. Functions and procedures used within a model must be defined in the module. To make functions and procedures generally accessible from different module statements the functions and procedures must be placed in a separate system file and included using the `include compiler directive.  **15) What are different styles of Verilog coding I mean gate-level,continuous level and others explain in detail?**  **16) Can you tell me some of system tasks and their purpose?** $display, $displayb, $displayh, $displayo, $write, $writeb, $writeh, $writeo. The most useful of these is $display.This can be used for displaying strings, expression or values of variables. Here are some examples of usage. $display("Hello oni"); --- output: Hello oni $display($time) // current simulation time. --- output: 460 counter = 4'b10; $display(" The count is %b", counter); --- output: The count is 0010 $reset resets the simulation back to time 0; $stop halts the simulator and puts it in interactive mode where the user can enter commands; $finish exits the simulator back to the operating system   **17) Can you list out some of enhancements in Verilog 2001?** In earlier version of Verilog ,we use 'or' to specify more than one element in sensitivity list . In Verilog 2001, we can use comma as shown in the example below. // Verilog 2k example for usage of comma always @ (i1,i2,i3,i4) Verilog 2001 allows us to use star in sensitive list instead of listing all the variables in RHS of combo logics . This removes typo mistakes and thus avoids simulation and synthesis mismatches, Verilog 2001 allows port direction and data type in the port list of modules as shown in the example below module memory ( input r, input wr, input [7:0] data\_in, input [3:0] addr, output [7:0] data\_out );   **18)Write a Verilog code for synchronous and asynchronous reset?** Synchronous reset, synchronous means clock dependent so reset must not be present in sensitivity disk eg: always @ (posedge clk )  begin if (reset) . . . end Asynchronous means clock independent so reset must be present in sensitivity list. Eg Always @(posedge clock or posedge reset) begin if (reset) . . . end  **19) What is pli?why is it used?** Programming Language Interface (PLI) of Verilog HDL is a mechanism to interface Verilog programs with programs written in C language. It also provides mechanism to access internal databases of the simulator from the C program. PLI is used for implementing system calls which would have been hard to do otherwise (or impossible) using Verilog syntax. Or, in other words, you can take advantage of both the paradigms - parallel and hardware related features of Verilog and sequential flow of C - using PLI.  **20) There is a triangle and on it there are 3 ants one on each corner and are free to move along sides of triangle what is probability that they will collide?** Ants can move only along edges of triangle in either of direction, let’s say one is represented by 1 and another by 0, since there are 3 sides eight combinations are possible, when all ants are going in same direction they won’t collide that is 111 or 000 so probability of not collision is 2/8=1/4 or collision probability is 6/8=3/4  **Task**  **T flip flop**  Module t\_ff (t,q,clk);  Input t, clk;  Output reg q=0;  Always @ (posedge clk)  Begin  If (t==1)  Begin  q=~q;  end  else  begin  q=q;  end  endmodule |
|  |

**DAILY ASSESSMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **04/06/2020** | **Name:** | **Dhavala** |
| **Course:** | **PYTHON** | **USN:** | **4AL17EC027** |
| **Topic:** | * **Application 7: Scrape Real Estate Property Data from the Web** | **Semester & Section:** | **6TH SEM & A Section** |
| **Github Repository:** | **Dhavala27** |  |  |

|  |
| --- |
| **AFTERNOON SESSION DETAILS** |
|  |

|  |
| --- |
| **Report**-  The study aims to inspect the stableness of interactive affinity between search interest of prices of the  stock and evidence stock market out comes on worldwide equity market indices. This study represents  and develop former exploration into financial graph by registering the attributes and magnitudes of the  graph use and embarkment from representational impartiality.such a paradox could also be derived  through investors behavior and degree of disclosure inclusion  •Python script to plot stock market data using bokeh library and deploy the bokeh plot to a live website.  •A ticker symbol or stock symbol is an abbreviation used to uniquely identify publicly traded shares of a particular stock on a particular stock market.  A stock symbol may consist of letters, numbers or a combination of both. "Ticker symbol" refers to the symbols that were printed on the ticker tape of a ticker tape machine.  •Some of the examples are:  NYSE (New York Stock Exchange) uses the ticker symbol with 3 letters or few  such as ‘NYT’ for the New York Times Co. or ‘T’ for AT&T.  Symbols with 4 or more letters generally denote securities traded on the American stock exchange and NASDAQ.  Those ending in ‘X’ indicate mutual funds.  There are also certain symbols that denote specific status or type of security say,  tickers ending in ‘Q’ indicate issuers which are under bankruptcy and letter ‘Y’  denotes security is an ADR.  •Some of the functions used under bokeh library:  It is possible to ask Bokeh to return the individual components of a standalone  document for individual embedding using the components ()function under  bokeh .embed module. This function returns a <script> that contains the data for your plot, together with an accompanying <div> tag that the plot view is loaded into. These tags can be used in HTML documents however you like.  The resources module provides the Resources class for easily configuring how  BokehJS code and CSS resources should be located, loaded, and embedded in Bokeh documents.  Additionally, functions for retrieving Sub resource Integrity hashes for Bokeh  JavaScript files are provided here.  Content delivery network (CDN): Load minified BokehJS from CDN |

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **03/06/2020** | **Name:** | **Dhavala** |
| **Course:** | **Bonus session** | **USN:** | **4AL17EC027** |
| **Topic:** | * **What is electrical engineering?** | **Semester & Section:** | **6TH SEM & A Section** |
| **Github Repository:** | **Dhavala27** |  |  |





