**4 June 2020**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Date:** | | **4 June 2020** | | | **Name:** | **Srinidhi J C** |
| **Course:** | | **Logic Design** | | | **USN:** | **4al16ec078** |
| **Topic:** | | |  | | --- | | Hardware modelling using verilog | | FPGA and ASIC Interview questions | | | | **Semester & Section:** | **8th -Sem, B-Sec** |
| **Github Repository:** | | **SrinidhiJC078** | | |  |  |
| **FORENOON SESSION DETAILS** | | | | | | |
| **Image of session**  **A screenshot of a cell phone  Description automatically generatedA picture containing bird  Description automatically generatedA picture containing circuit  Description automatically generatedA screenshot of a video game  Description automatically generatedA screenshot of a cell phone  Description automatically generatedA screenshot of a cell phone  Description automatically generated** | | | | | | |
| Report:The present trend is to standardize something called design flow means that steps that you need to follow to create a VLSI circuit or a chip. And as I said the present emphasis is number 1 on low power design and number 2 on increased performance. So, here in this diagram I am showing 2 circuits side by side. So, on the left you have the first IC planar means it is laid down in a single plain this is called a planar IC. So, you can very easily see the connections the metals and the devices which are prepared, this was a very simple circuit. And on the right side you think of one of the; you see one of the modern processors chips the Intel Quad Core Nehalem series processor. So, here as I said you have something of the order of billion transistors packed in a single chip. So, when you look at the layout in a very compacted way you see a colorful picture like This, where of course the different colors indicate the different layers of the circuit, right, ok, this is a very interesting plot, Moore’s Law is a very important law in semiconductor design you can say. The persons who are into semiconductor design, they all know about Moore’s Law. There was a person called Gordon Moore who as early as in the 1960s predicted some behavior about the growth in the semiconductor industry. So, what he had said at that time was that the number of transistors that you can put inside the chip would be increasing exponentially with time, with the number of years that pass. So, there have been some refinements to this basic you can say law or postulate. So, what it is accepted more or less today is something like this it says that every 18 months or so the number of transistors in a chip single chip will get doubled. Now, there were people who had doubted this principle or law since quite a long time in the past. This said that well the devices are becoming smaller and smaller the transistors you are making smaller, so there will be a time, a time will come where you will not be able to make the devices any further smaller. So, there will be a limit and beyond that Moore’s Law will cease to exist. But actually what has happened till today is that, because of semiconductor fabrication advances we are able to fabricate bigger chips, in the process we have been able to sustain Moore’s law, we have been put more circuits in the chip. So, if you look at this graph, so on the x axis we are plotting year started from 1970 so here it shows up to 2015, and on the y axis you see the number of transistors which is in a log scale see 1000 here up to here it is 1 billion here it is 10 billion. So, you can see there is a straight-line kind of a behavior which indicates exponential growth, and here the blue dots refer to the processors which are manufactured by the processor major Intel Corporation, the red dots are the processors manufactured by some other companies, ok. So, Moore’s Law as you can see, this has continued to hold and this trend is still a straight line behavior which indicates exponential growth over the years. Well, the technologies that have made this possible are well CMOS. You may be knowing that CMOS is the most dominant technology today, with which we are manufacturing our VLSI chips, and the CMOS transistors are becoming smaller and smaller and smaller over the years. There is something called feature size which we talk about, that is roughly that is equal to this smallest feature or the transistor that you can fabricate. Well, the state of the art CMOS technology today you can go down up to 22 nanometer, this is traditional CMOS fabrication. But there has been some very innovative kind of CMOS designs also, there is something called finfet, where the gate drain and the source they are staked vertically instead of horizontally as in the traditional case, and in this way you can pack transistors in a smaller area. So, today in the finfet state of the technology you can go down to 14 nanometer. And many of the modern chips that are coming in the market, they are actually manufactured using this finfet technology, ok. And the picture which is shown in the right, this is of course futuristic, this we do not have today, tomorrow quantum computer may come so you may be having a new technology the quantum technology.Verilog interview Questions  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 locking procedural assignment is used for combinational logic and non-blocking procedural assignment for sequential **Tell me about verilog file I/O?**  OPEN A FILE integer file; file = $fopenr("filename"); file = $fopenw("filename"); file = $fopena("filename"); The function $fopenr opens an existing file for reading. $fopenw opens a new file for writing, and $fopena opens a new file for writing where any data will be appended to the end of the file. The file name can be either a quoted string or a reg holding the file name. If the file was successfully opened, it returns an integer containing the file number (1..MAX\_FILES) or NULL (0) if there was an error. Note that these functions are not the same as the built-in system function $fopen which opens a file for writing by $fdisplay. The files are opened in C with 'rb', 'wb', and 'ab' which allows reading and writing binary data on the PC. The 'b' is ignored on Unix. CLOSE A FILE integer file, r; r = $fcloser(file); r = $fclosew(file); The function $fcloser closes a file for input. $fclosew closes a file for output. It returns EOF if there was an error, otherwise 0. Note that these are not the same as $fclose which closes files for writing.  **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 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 assignment 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 otherwise 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 of 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 are 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.   **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**  Implement a simple T Flipflop and test the module using a compiler.  moduletff(inputclk,  inputrstn,  inputt,  outputregq);  always@(posedgeclk)begin  if(!rstn)  q<=0;  else  if(t)  q<=~q;  else  q<=q;  end  endmodule  Testbenchcode  moduletb;  regclk;  regrstn;  regt;  tffu0(.clk(clk),  .rstn(rstn),  .t(t),  .q(q));  always#5clk=~clk;  initialbegin  {rstn,clk,t}<=0;  $monitor("T=%0trstn=%0bt=%0dq=%0d",$time,rstn,t,q);  repeat(2)@(posedgeclk);  rstn<=1;  for(integeri=0;i<20;i=i+1)begin  reg[4:0]dly=$random;  #(dly)t<=$random;  end  #20$finish;  end  endmodule | | | | | | | Verilog interview Questions [**Verilog interview Questions page 1**](http://www.asic.co.in/Index_files/verilog_interview_questions.htm)[**Verilog interview Questions Page 2**](http://www.asic.co.in/Index_files/verilog_interview_questions2.htm)  [**Verilog interview Questions page 3**](http://www.asic.co.in/Index_files/verilog_interview_questions3.html)[**Verilog interview Questions page 4**](http://www.asic.co.in/Index_files/verilog_interview_questions4.htm)    **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  [**Click to view more**](http://www.asic.co.in/Index_files/verilogexamples.htm)  **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  [**Click to view more**](http://www.asic.co.in/Index_files/digital_files/clock_domain_crossin.htm)  **Tell me about verilog file I/O?**   OPEN A FILE  integer file; file = $fopenr("filename"); file = $fopenw("filename"); file = $fopena("filename"); The function $fopenr opens an existing file for reading. $fopenw opens a new file for writing, and $fopena opens a new file for writing where any data will be appended to the end of the file. The file name can be either a quoted string or a reg holding the file name. If the file was successfully opened, it returns an integer containing the file number (1..MAX\_FILES) or NULL (0) if there was an error. Note that these functions are not the same as the built-in system function $fopen which opens a file for writing by $fdisplay. The files are opened in C with 'rb', 'wb', and 'ab' which allows reading and writing binary data on the PC. The 'b' is ignored on Unix.  CLOSE A FILE  integer file, r; r = $fcloser(file); r = $fclosew(file);  The function $fcloser closes a file for input. $fclosew closes a file for output. It returns EOF if there was an error, otherwise 0. Note that these are not the same as $fclose which closes files for writing.  [**Click to view more**](http://www.asic.co.in/Index_files/verilog_files/File_IO.htm)     **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   **5) What is delta simulation time?**   **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 | | |
| **Date:** | **4 June 2020** | | **Name:** | **Srinidhi J C** | | |
| **Course:** | **Python** | | **USN:** | **4al16ec078** | | |
| **Topic:** | **Application 8 Build a Web-based Financial Graph** | | **Semester & Section:** | **8th-Sem, B-Sec** | | |
| **AFTERNOON SESSION DETAILS** | | | | | | |

**Images os the sessionA group of people

Description automatically generatedA screenshot of a social media post

Description automatically generatedA screenshot of a social media post

Description automatically generatedA screenshot of a cell phone

Description automatically generated**

**Report:**

Bokeh is a powerful open source Python library that allows developers to generate JavaScript data visualizations for their web applications without writing any JavaScript. While learning a JavaScript-based data visualization library liked 3. jscan be useful, it is often far easier to knock.

Bokeh provides a variety of ways to embed plots and data into HTML documents. First, a reminder of the distinction between standalone documents and apps.

Standalone Documents:

These are Bokeh documents that are not backed by a Bokeh server. They may have many tool sand interactions (e.g. from Custom JScall backs) but are self-contained HTML, JavaScript, and CSS. They can be embedded into other HTML pages as one large document, or as a set of sub-components template individually.

Bokeh Applications:

These are Bokeh documents that are backed by a Bokeh Server. In addition to all the features of standalone documents, It is also possible to connect even sand tools to real Python callbacks that execute in the Bokeh server.

HTML files Bokeh can generate complete HTML pages for Bokeh documents using the file\_html() function. This function can emit HTML from its own generic template, or a template you provide. These files contain the data for the plot in line and are completely transportable, while still providing interactive tools (pan, zoom, etc.) for your plot. Here is an example:

from bokeh. Plotting import figure

from bokeh. Resources import CDN

frombokeh. Embed import file\_html

plot=figure ()

plot.circle([1,2],[3,4])

html=file\_html(plot, CDN,"myplot")

from flask import Flask, render\_template

app=Flask(\_\_name\_\_)

@app.route('/plot/')

def plot ():

from pandas\_datareader import data

import datetime

import fix\_yahoo\_finance as yf

yf.pdr\_override()

from bokeh.plotting import figure, show, output\_file

from bokeh.embed import components

from bokeh.resources import CDN

start=datetime.datetime(2015,11,1)

end=datetime.datetime(2016,3,10)

df=data.get\_data\_yahoo(tickers="GOOG", start=start, end=end)

def inc\_dec(c, o):

if c > o:

value="Increase"

elif c < o:

value="Decrease"

else:

value="Equal"

return value

df["Status"]=[inc\_dec(c,o) for c, o in zip(df.Close,df.Open)]

df["Middle"]=(df.Open+df.Close)/2

df["Height"]=abs(df.Close-df.Open)

p=figure(x\_axis\_type='datetime', width=1000, height=300)

p.title.text="Candlestick Chart"

p.grid.grid\_line\_alpha=0.3

hours\_12=12\*60\*60\*1000

p.segment(df.index, df.High, df.index, df.Low, color="Black")

p.rect(df.index[df.Status=="Increase"],df.Middle[df.Status=="Increase"],

hours\_12, df.Height[df.Status=="Increase"],fill\_color="#CCFFFF",line\_color="black")

p.rect(df.index[df.Status=="Decrease"],df.Middle[df.Status=="Decrease"],

hours\_12, df.Height[df.Status=="Decrease"],fill\_color="#FF3333",line\_color="black")

script1, div1 = components(p)

cdn\_js=CDN.js\_files[0]

cdn\_css=CDN.css\_files[0]

return render\_template("plot.html",

script1=script1,

div1=div1,

cdn\_css=cdn\_css,

cdn\_js=cdn\_js )

@app.route('/')

def home():

return render\_template("home.html")

@app.route('/about/')

def about():

return render\_template("about.html")

if \_\_name\_\_=="\_\_main\_\_":

app.run(debug=True)