**Verilog**

**What is difference between blocking (=) and non-blocking (<=) statements? *(This is a very common question)***

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. Non-blocking statements allow you to schedule assignments without blocking the procedural flow. You can use the non-blocking procedural statement whenever you want to make several register assignments within the same time step without regard to order or dependence upon each other. It means that non-blocking statements resemble actual hardware more than blocking assignments.

**Write a Verilog code for synchronous and asynchronous reset? *(This is a very common question)***

Synchronous reset, synchronous means clock dependent so reset must not be present in sensitivity disk e.g.:   
always @ (posedge clk )  
begin if (reset)  
. . . end  
Asynchronous means clock independent so reset must be present in sensitivity list.  
E.g.:   
Always @ (posedge clock or posedge reset)  
begin  
if (reset)  
. . . end  
Asynchronous Reset : works immediately, as soon as it is applied. Sync reset: waits for the clock to have effect. Neither is superior, it depends upon application.  
I would use Sync Resets everywhere in the design to make it fully synchronous. But in some cases, when the chip is being powered up, you don’t have a clock yet, in those cases asynchronous reset would help.

**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

**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. 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.

**Specify few Verilog constructs which are not supported by the synthesis tool.**

initial, delays, real and time data types, force and release, fork join.

**Difference between task and function?**

tasks can have any number of inputs and outputs. A task shall not return a value.  
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)

functions cannot include timing delays, like posedge, negedge, # delay, which means that functions should be executed in "zero" time delay.

functions can have any number of inputs but only one output.  
Within a function, no event, delay or timing control statements are permitted   
module simple\_task();

**task convert;**

**input [7:0] temp\_in;**

**output [7:0] temp\_out;**

**begin**

**temp\_out = (9/5) \*( temp\_in + 32)**

**end**

**endtask**

**endmodule**

Verilog functions are synthesizable today and Verilog tasks are synthesizable as long as there are no timing controls in the body of the task, such as @(posedge clk). The #delay construct is ignored by synthesis tools.

**Difference between $monitor, $display & $strobe?**   
These commands have the same syntax, and display text on the screen during simulation. $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.

**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 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."   
  
**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. 

**Variable and signal which will be updated first?**  
Signals.  
  
**What is sensitivity list?**  
The sensitivity list indicates that when a change occurs to any one of elements in the list, begin…end statement inside that always block will get executed.   
  
**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.   
  
**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 world"); --- output: Hello world  
$display($time) // current simulation time.  
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

**Tell me about Verilog file I/O?**   
OPEN A FILE  
integer file;  
file = $fopen("filename", “r”);  
file = $fopenw("filename", “w”);  
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.

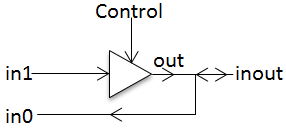
**Input output port**

case (control)

1: out = in1;

default out = z;

endcase



**Structural vs. Behavioral**

Verilog can be structural or behavioral. Structural definition specifies the gates and their connections explicitly.

Behavioral definition specifies the functionality of a design, and does not contain any structural information such as transistors or gates. Logic synthesis software implements the structural.