Question 1 1/2

a) Sensitivity lists contain signals that trigger a procedural block. Some types of procedural blocks, e.g. always_latch and always_comb have implicit sensitivities derived automatically from the code.

Example:
always_ff @(posedge clk, reset)
  if (reset)
    Q <= 1'b0;
  else Q <= D;

  a) Inferred hardware

```
  c
  |
  v
  y

  b
  |
  v
  y

  a
```

The code infers a 1-bit transparent latch. always_latch and does not require an explicit sensitivity list as the inputs to the block can be derived from the code.

(c) combinational logic cannot be derived due to the incomplete if statement
Question 1 2/2

modification:

```verilog
always_comb
    begin
        if (a == 1'b1)
            if (b == 1'b1)
                y = 1'b0;
            else
                y = c;
        else
            y = 1'b1; // added to complete if statement
    end;
```

Inferred logic:
**Question 2 1/2**

(a) A testbench is used for testing and verification
   - it is not synthesised, used for simulation only
   - it is a top module with no ports
   - it typically contains one or more initial blocks with input waveform specification

(b) module testbench;

```
logic[7:0] Q,D;
logic clk,ctrl,load,reset;

// register instance
reg r (.*);

initial // clock block
begin
  clk = 1'b0;
  forever #10 clk = ~clk;
end;

initial
begin
  D <= 8'hAA; // sample data
  ctrl = 1'b1; // drive output
  load = 1'b1; // load D
  reset = 1'b1; // clear register
  #50 reset = 1'b0; // remove reset
  #50 ctrl = 1'b0; // remove ctrl, outputs go hi-z
  ... etc
end;
```
Question 2 2/2

c) modified testbench with two regs and potential bus conflict detection
module testbench;

logic[7:0] Q1,D1,Q2, D2;
logic clk,ctrl1,ctr2,load,reset;

// register instances
regs r1 (.Q(Q1),.D(D1),.ctrl(ctrl1),.load(load),.reset(reset),.clk(clk));
regs r1 (.Q(Q2),.D(D1),.ctrl(ctrl2),.load(load),.reset(reset),.clk(clk));

initial // clock block
begin
    clk = 1'b0;
    forever #10 clk = ~clk;
end;

// concurrent assertion to detect if both outputs are driven
assert property ( ~(Q1==8'hZZ | Q2== 8'hZZ)) // check if at least one reg is disconnected.
    $error("failed at time %0t", $time);

initial
begin
    D1 <= 8'hAA; // sample data
    D2 <= 8'11;
    ctrl = 1'b1; // drive output 1
    ctr1 = 1'b0; // disconnect output 2
    load = 1'b1; // load D1,D2
    reset = 1'b1; // clear registers
    #50 reset = 1'b0; // remove reset
    #50 ctrl1 = 1'b0; // remove ctrl, outputs go hi-z
    #50 ctrl1 = 1'b1; ctrl2 = 1'b1; // both registers driven, assertion issues error
    ... etc
end;
endmodule;
Question 3 1/1

(a) Assertions bring significant benefits in both the design and verification processes of digital hardware. The primary benefit is that assertions help to detect more functional bugs, detect them earlier in the process and detect them closer to their original cause. This leads in turn to fewer bugs remaining undetected in production, shorter verification times and faster debugging. Another benefit is that the very act of formulating and writing assertions can give the designer a better understanding of the design, and hence uncover bugs in the specification or else avoid introducing bugs into the design in the first place.

(b)

```verilog
property p
  @posedge(clk) (a! = b) ##2 b;
endproperty;

assert property(p)
  $display("assertion of property p passed\n");
extelse
  $display("assertion of property p failed\n");
```

This assertion passes if:
1. a and b are different on each rising edge of the clock, and:
2. b = 1'b1 one or two clock periods later

(c) priority casex requires evaluation of cases in the order listed and asserts that all the cases are covered. In the example this is clearly not the case as the case `a=3'b001` is not covered. So this assertion will fail.

(cont).
Priority asserts that cases should be evaluated in the order listed and that one and only one case is selected at any time. This assertion will fail if \( a = 3'b0x0 \) as this is not covered by the case statement.

```verilog
module testbench;
logic[2:0] a;
logic y;

// module instance
coder c(.*)

// this concurrent assertion detects if \( y \) is ever \( x \) or \( z \)
assert property (y == 1'b0 | y == 1'b1) // assert that \( y \) must be 0 or 1
$display("y is 0 or 1 \n ");
else
$error("y is \( x \) or \( z \); assertion failed at time %0t", $time);
initial
begin
a=3'b000;
#50 a= 3'b100; // assertion passes
#50 a =3'b001; // assertion fails as behaviour for this case is not defined so \( y \) must be \( x \)
endmodule;
```