**Question A1**

(a) An initial block produces waveforms which generally cannot be synthesized into hardware

(b) A 2-to-1 mux where a is the select input, an or gate for c | d when 0 is selected and b when a is selected

(c) `===` is a wild equate, where don’t cares can be used in logic, whereas `==` equates each bit of two logic objects literally, e.g. `1’bx == 1’b1` has the value of 0.

(d) The reset is asynchronous as it overrides the clock. The synthesiser determines from the code that a is reset and, as q is not directly dependent on b in the code, b must be the clock.

(e) Blocking assignments execute immediately, non-blocking assignments are deferred until all blocks are completely executed at the current cycle.

```
always_comb
begin
    out = a ^ b;
    out = a & b;
end
```

The value of `out` in this case will be the same, regardless which type of assignment is used.
module craps(input logic clk, throw, reset, logic [3:0] D, output logic win, lose, play);
    logic[3:0] point; // memory for the point
    typedef logic [1:0] enum (start, playing, won, lost) state;
    state present, next;

always_ff @ (posedge clk or posedge reset)
begin
    if (reset)
        point <= 4'b0000;
        present <= start;
    else
        begin
            if(state == start && throw)
                point = D; // always save throw, comb logic will determine if it’s a point
                present <= next;
        end
end

always_comb
begin
    play = 1; win=0; lose=0; //default outputs
    case(present)
        start: if (throw)
            begin
                if (D==7 | D == 11)
                    next = won;
                else if (D== 2 | D==3 | D==12)
                    next = lost;
                else
                    next = playing;
            end
        playing: if (throw)
            begin
                if (D==7)
                    next <= lost;
                else if( D==point)
                    next <= won;
                else
                    next <= playing;
            end
        lost:
            begin
                lose = 1; play = 0;
                next <= lost; // end game and wait for reset
            end
        won:
            begin
                win = 1; play = 0;
                next <= win; // end game and wait for reset
            end
    endcase
end
endcase
end
endmodule;

b)
module testbench
logic clk, throw, reset;
logic [3:0] D;
logic win, lose, play;
craps c(.*);
initial
begin
 clk = 0;
 forever #10ns clk = ~clk;
end

initial // test scenario
begin
 reset = 1;
 throw = 0;
 D = 0;
 #100 reset = 0; // remove reset
 #100 throw = 1; D= 6; // throw 6
 #100 throw = 0;
 #100 throw = 1; D=10; // throw 10
 #100 throw = 0;
 #100 throw =1l D = 6; // throw the point and win
end
endmodule
Question A3

a) immediate assertions occur within other procedural block and execute immediately after the evaluation of the assert expression, e.g.
   assert (A == B) $display("A equals B – this is expected");
   else $error("A is not equal to B, something has gone wrong");

Concurrent assertions are normally procedural blocks in their own right, although they can also occur in sequential code. They can be used to check timed behaviour and relationships between signals driven by other blocks. SV concurrent assertions provide a powerful debugging mechanism in hardware descriptions. E.g.

assert property (@(posedge clk) Request |-> ##[1:2] Ready; else $error(“Request failed as Ready did not arrive in time”);

The above code asserts that when Request is raised, Ready must be raised at the next or the following clock cycle. Both signals can be driven by two separate blocks.

b)

This can be checked, for example, using the $past function

property dffdelay1;
   @(posedge clk)
      Q == $past(D)
endproperty;

assert property (dffdelay1); // this assertion fails when Q changes immediately after D and passes when Q changes after D at a clock tick

An alternative way of checking the dff behaviour could be:

property dffdelay2;
   clk =
      Q == $past(D)
endproperty;

; // note that == works better here than the more usual |->
   // as we want to check that Q follows D
   // at the next clock tick for both states, 0 or 1
endproperty;

Evaluation of concurrent assertions starts at every clock cycle with a new incarnation of a check. The check is postponed into the future until a definite conclusion on pass or fail can be made.

assert property (@(posedge clk) disable iff (!rst_n) (q==$past(d)))
ANSWER FORM

Paper: ELEC2014 2011-12
Author: T.J Kazmierski

Proportion of answer derived from:
Lectures: 80%
Laboratories:
Prescribed Reading: 20%
Independent Reading: