SOLUTION NOTES

ECAD 2001 Paper 4 Question 6 (SWM)

(a) When designing clocked circuits there are times when asynchronous inputs have to be sampled which may result in metastable behavior in state holding elements. How might metastability be avoided when sampling asynchronous inputs?

[5 marks]

Ans: When sampling asynchronous data with a D-latch, the D-latch is liable to enter a metastable state if the D input changes whilst the clock is rising (assuming the D-latch stores data on the rising edge of the clock). It takes time for metastability to resolve. By latching the data twice in a shift register arrangement (figure would be helpful...) then the first D-latch is given time to resolve metastability before the second D-latch latches the data. This does not guarantee that metastability will be resolved in time, but statistically it is far better than simply latching the data once.

(b) An optical shaft encoder (e.g. used on the internal rollers of mechanical mouse) consists of a disk with an evenly spaced alternating transparent and opaque grating around the circumference. Two optical sensors are positioned such that when one sensor is at the middle of an opaque region, the other is at the edge. Consequently, the following Grey code sequence is produced, depending upon the direction of rotation:

positive rotation	negative rotation	
00	00	|
01	10	|
11	11	|
10	01	V time

A shaft decoder module is required to convert the Grey code into an 8-bit position. The 8-bit position should be incremented every time the input changes from one state to another in a positive direction (e.g. from 00 to 01, or from 10 to 00). Similarly, the 8-bit position should be decremented every time the input changes from one state to another in a negative direction (e.g. from 00 to 10, or from 01 to 00).

Write and comment a Verilog module which performs the function of a shaft decoder.

[15 marks]

Ans:

```
module ShaftDecoder(clk, encoded, pos);
  input clk;              // input clock
  input [1:0] encoded;    // shaft encoding
  output [7:0] pos;   // 8-bit position

  reg [7:0] pos;   // make pos registered
  reg [1:0] encodeA, encodeB, encodeHistory;

  always @(posedge clk) begin
    encodeA <= encoded;
    encodeB <= encodeA;   // sample encoded twice before use
    encodeHistory <= encodeB;
    case({encodeHistory,encodeB})  // concatenate previous and current states
      4'b0001 : pos<=pos+1;
      4'b0111 : pos<=pos+1;
      4'b1000 : pos<=pos+1;
      4'b1110 : pos<=pos+1;
      4'b0010 : pos<=pos-1;
      4'b0100 : pos<=pos-1;
      4'b1011 : pos<=pos-1;
      4'b1101 : pos<=pos-1;
      default : pos<=pos;
    endcase
  end
endmodule
```