The University of New South Wales

ELEC2141: Digital Circuit Design

Tutorial 8 – Solutions

1.

```
module decoder_2_to_4_vector(EN, A, D);
   input EN;
   input [1:0] A;
   output [3:0] D;

   wire [1:0] A_n;
   wire [3:0] N;

   not gn0(A_n[0], A[0]);
   not gn1(A_n[1], A[1]);

   and ga3(N[0], A_n[0], A_n[1]);
   and ga4(N[1], A[0], A_n[1]);
   and ga5(N[2], A_n[0], A[1]);
   and ga6(N[3], A[0], A[1]);
   and ga7(D[0], N[0], EN);
   and ga8(D[1], N[1], EN);
   and ga9(D[2], N[2], EN);
   and ga10(D[3], N[3], EN);

endmodule
```

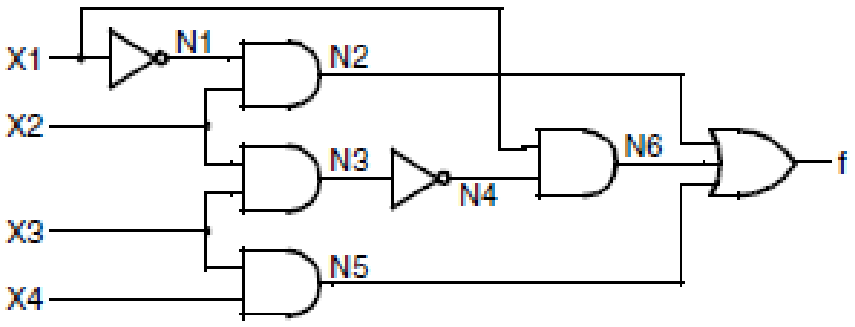
2.

```
module circuit2(A, B, C, D, X, Y);
   input A, B, C, D;
   output X, Y;

   wire n1, n2, n3, n4, n5;

   not  gn0(n1, D);
   and  ga1(n2, B, C);
   nor  gr2(n3, n1, A);
   nand gd3(n4, n1, n3);
   or   go4(n5, n1, n2);
   and  ga5(X, n4, n5);
   and  ga6(Y, n3, n5);

endmodule
```

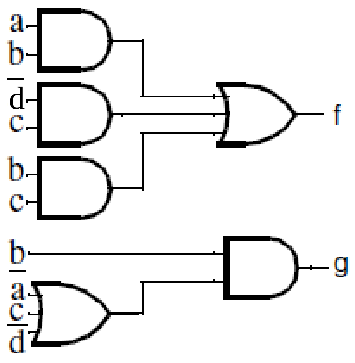
3.

```
module circuit3(X, F);
   input [2:0] X;
   output F;

   wire [1:5] w;

   nand gd1(w[1], X[2], X[1]);
   nand gd2(w[2], X[2], w[1]);
   nand gd3(w[3], X[1], w[1]);
   nand gd4(w[4], w[2], w[3], X[0]);
   nand gd5(w[5], w[3], X[0]);
   nand gd6(F, w[4], w[5]);

endmodule
```

4.



5. Use Karnaugh maps to optimize both *f* and *g* to two-level logic to obtain:



6.

```
module priority_encoder(D, A, V);
   input [3:0] D;
   output [1:0] A;
   output V;

   assign V = D[0] | D[1] | D[2] | D[3];

   assign A = D[3] ? 2'b11
                   : D[2] ? 2'b10
                   : D[1] ? 2'b01
                   : D[0] ? 2'b00
                   : 2'bxx;

endmodule
```

7.

```
module jk_ff_sr(CLK, J, K, S, R, Q);
   input CLK, J, K, S, R;
   output Q;

   reg Q;

always @(negedge CLK) begin
      if (S) Q <= 1'b1;
      else if (R) Q <= 1'b0;
      else Q <= ((J & ~Q) | (~K & Q));
   end

endmodule
```

8.

```
module mux_case(S, D, Y);
 input[1:0]  S;
 input[3:0]  D;
 output reg Y;



always @(S or D)
      case (S)
         2'b00 : Y = D[0];
         2'b01 : Y = D[1];
         2'b10 : Y = D[2];
         2'b11 : Y = D[3];
      endcase


endmodule
```

9.

```
module mux_if(S, D, Y);
   input [1:0] S;
   input [3:0] D;
   output Y;

   reg Y;

   always @(S or D) begin
      if (S == 2’b00) Y = D[0];
      else if (S == 2’b01) Y = D[1];
      else if (S == 2’b10) Y = D[2];
      else Y = D[3];
   end

endmodule
```

10.

```
module state_machine10(CLK,Clear X, Z);
  input CLK, Clear;
  input [1:2] X;
  output reg Z;

   parameter A = 2'b00, B = 2'b01, C = 2'b11, D = 2'b10;

   reg [1:0] state, next_state;

       // Next-state transition
     always @(posedge CLK, negedge Clear)
        if (clear == 0) state <= A;

        else state <= next_state;

       // Next-state inputs
always @(X or state)
      case (state)
         A: if (X == 2'b01 || X == 2'b10) next_state = B;
            else next_state = A;
         B: if (X == 2'b10 || X == 2'b11) next_state = D;
            else next_state = A;
         C: if (X == 2'b00 || X == 2'b01) next_state = A;
            else next_state = C;
         D: if (X == 2'b00 || X == 2'b11) next_state = C;
            else next_state = B;
      endcase


   // Output function
always @(X or state)
      case (state)
         A: Z = (X == 2'b10) ? 1'b1 : 1'b0;
         B: Z = (X == 2'b10 || X == 2'b11) ? 1'b1 : 1'b0;
         C: Z = (X == 2'b00 || X == 2'b10) ? 1'b1 : 1'b0;
         D: Z = (X == 2'b11) ? 1'b0 : 1'b1;
      endcase


endmodule
```

11.

```
module state_machine11(CLK,Clear X, Z);
   input CLK,Clear X;
   output reg Z;

   parameter A = 3'b000, B = 3'b001, C = 3'b010,
             D = 3'b011, E = 3'b100, F = 3'b101;

   reg [2:0] state, next_state;


// Next-state and output transitions
always @(posedge CLK or negedge Clear)
      if (clear == 0)  state <= A;
      else state <= next_state;


  // Next-state inputs
  always @(X or state)
      case (state)
         A: if (X) next_state = D; else next_state = B;
         B: if (X) next_state = C; else next_state = D;
         C: if (X) next_state = F; else next_state = A;
         D: if (X) next_state = C; else next_state = F;
         E: if (X) next_state = E; else next_state = C;
         F: if (X) next_state = F; else next_state = E;
      endcase


// Next-output function
always @(state) begin
           case (state)
         A: next_z = 0;
         B: next_z = 0;
         C: next_z = 0;
         D: next_z = 1;
         E: next_z = 1;
         F: next_z = 1;
      endcase


endmodule
```