Faculty of Engineering, Mathematics and Science
School of Computer Science & Statistics

Integrated Computer Science Programme
Trinity Term 2018
Year 1 Annual Examinations

CS1026 – Digital Logic Design

Tuesday, 8\textsuperscript{th} May 2018  
RDS Main Hall  
09:30 – 12:30

Prof. Paula Roberts and Prof. John Waldron

Instructions to Candidates:

Section A is worth 50 marks. You must answer TWO questions from Section A. Each question is worth 25 marks. A correct answer to each question in Section B is worth 5 marks, a wrong answer loses 1 mark. Enter your answers on the special optical mark recognition answer sheet provided. Fill in the circle corresponding to your answer for each question \(\bullet\), preferably using a black pen. You may not start this examination until you are instructed to do so by the Invigilator. Exam Paper is not to be removed from venue.

Materials permitted for this examination:

Non-programmable calculators are permitted for this examination — please indicate the make and model of your calculator on each answer book used. To be accompanied by an optical mark recognition answer sheet.
Section B

Q B.1
// For the inputs ABC = 000, 001, 010, ... ,111
// which sequence matches output E
//
module Structural_3 (A, B, C, D, E);
output D, E;
input A, B, C;
wire w1;
    xnor G1 (w1, A, B);
    xor G2 (E, A, C);
    or G3 (D, w1, E);
endmodule

Q B.2
// For the inputs ABCD = 0000, 0001, 0010, ... ,1111
// which sequence matches output F1
//
module Structural_2 (input A, B, C, D, output F1, F2);
wire A_bar = !A;
wire B_bar = !B;
wire T1, T2, T3, T4;
    and (T1, B_bar, C);
    and (T2, A_bar, B);
    or (T3, A, T1);
    xor (T4, T2, D);
    or (F1, T3, T4);
    or (F2, T2, D);
endmodule

Q B.3
// For the inputs A B = 00 00, 00 01, 00 10, ... ,11 11
// which sequence matches output A_eq_B
//
module mag_compare ( input [1: 0] A, B, output A_lt_B, A_eq_B, A_gt_B);
    assign A_lt_B = (A < B);
    assign A_gt_B = (A > B);
    assign A_eq_B = (A == B);
endmodule

Page 2 of 6
Q B.4
// For the inputs ABCD = 0000, 0001, 0010, ... ,1111
// which sequence matches output F
//
module Dataflow_2 (input A, B, C, D, output F);
endmodule

( A) 1010011100010010 ( B) 1010010100011010
( C) 1010010100010010 ( D) 1010010100000010
( E) 1010010100010000 ( F) OTHER

Q B.5
// For the inputs A = 0000, 0001, 0010, ... ,1111
// which sequence matches output carry
//
module Incrementer (output [3: 0] sum, output carry, input [3: 0] A);
    assign {carry, sum} = A + 1;
endmodule

( A) 0000000000000001 ( B) 0000000000010001
( C) 0000000100000001 ( D) 0000000010000001
( E) 0000000000000011 ( F) OTHER

Q B.6
// For the inputs ABCD = 0000, 0001, 0010, ... ,1111
// which sequence matches output F1
//
module Behaviour_0 (input A, B, C, D, output reg F1, F2, V);
    always @ (A, B, C, D)
    begin
        casex ({A, B, C, D})
            4'b0000:{F1, F2, V} = 3'b000;
            4'b1000:{F1, F2, V} = 3'b001;
            4'bx100:{F1, F2, V} = 3'b011;
            4'bx110:{F1, F2, V} = 3'b101;
            4'bx111:{F1, F2, V} = 3'b111;
            default:{F1, F2, V} = 3'b000;
        endcase
    end
endmodule

( A) 0111011101110101 ( B) 0101011101110111
( C) 0111011101110111 ( D) 1111011101110111
( E) 0111011101110111 ( F) OTHER
Q B.7
// For the inputs sig_in = 0000, 0001, 0010, ... ,1111
// which sequence matches output sig_out[0]
//
module shift_right (output reg [3: 0] sig_out, input [3: 0] sig_in);
always @ (sig_in)
    sig_out = {sig_in[3], sig_in[3], sig_in[3: 2]};
endmodule

(A) 0000111100001111 (B) 0000111100001110
(C) 0000111100001101 (D) 0000111100101111
(E) 0000111010000111 (F) OTHER

Q B.8
// A = 8'b11111111; B = 8'b01010010; sel = 3'b101;
// which sequence matches output y
//
module ALU (output reg [7: 0] y, input [7: 0] A, B, input [2: 0] Sel);
always @ (A, B, Sel) begin
    y = 0;
    case (Sel)
        3'b000:y = 8'b0;
        3'b001:y = A & B;
        3'b010:y = A | B;
        3'b011:y = A ^ B;
        3'b100:y = A + B;
        3'b101:y = A - B;
        3'b110:y = ~A;
        3'b111:y = 8'hFF;
    endcase
end
endmodule

(A) 10101001 (B) 10101101
(C) 10101100 (D) 10111101
(E) 10100101 (F) OTHER
Q B.9
// For the inputs
initial begin clock = 0; forever #5 clock = ~clock; end
initial fork
#0 reset = 0;
#0 x_in = 0;
#2 reset = 1;
#10 x_in = 0;
#30 x_in = 0;
#40 x_in = 0;
#50 x_in = 0;
join
initial #55 $finish;
// which sequence matches output y_out[0]
// at each clock change
//
module Moore_Model (output [1: 0] y_out, input x_in, clock, reset);
reg [1: 0] state;
parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
always @(posedge clock, negedge reset)
   if (reset == 0) state <= S0; // Initialize to state S0
   else case (state)
      S0: if (x_in) state <= S2; else state <= S0;
      S1: if (x_in) state <= S3; else state <= S2;
      S2: if (x_in) state <= S1; else state <= S0;
      S3: if (x_in) state <= S2; else state <= S0;
   endcase
assign y_out = state;
endmodule
(A) 000100000000 (B) 000000000000
(C) 000001000000 (D) 000000000001
(E) 000010000000 (F) OTHER
Q B.10
// For the inputs
initial begin clock = 0; forever #5 clock = ~clock; end
initial fork
#0 reset = 0;
#0 in_x = 1; in_y = 1;
#2 reset = 1;
#10 in_x = 1;
#10 in_y = 0;
#30 in_x = 0;
#30 in_y = 0;
#40 in_x = 1;
#40 in_y = 1;
#50 in_x = 1;
#50 in_y = 1;
join
initial #55 $finish;
// which sequence matches output z
// at each clock change
//
module FSM (output out_z, input in_x, in_y, clk, reset_b);
reg [1:0] state, next_state;
assign out_z = ((state == 2'b11) || (state == 2'b10));
always @ (posedge clk, negedge reset_b)
  if (reset_b == 1'b0) state <= 2'b00;
  else state <= next_state;
always @ (state, in_x, in_y)
case (state)
  2'b00: if ({in_x, in_y} == 2'b00) next_state = 2'b00;
         else if ({in_x, in_y} == 2'b01) next_state = 2'b11;
         else if ({in_x, in_y} == 2'b10) next_state = 2'b01;
         else next_state = 2'b10;
  2'b01: if ({in_x, in_y} == 2'b00) next_state = 2'b11;
         else if ({in_x, in_y} == 2'b01) next_state = 2'b00;
         else if ({in_x, in_y} == 2'b10) next_state = 2'b01;
         else next_state = 2'b10;
  2'b10: if ({in_x, in_y} == 2'b00) next_state = 2'b11;
         else if ({in_x, in_y} == 2'b01) next_state = 2'b10;
         else if ({in_x, in_y} == 2'b10) next_state = 2'b00;
         else next_state = 2'b10;
  2'b11: if ({in_x, in_y} == 2'b00) next_state = 2'b00;
         else if ({in_x, in_y} == 2'b01) next_state = 2'b10;
         else if ({in_x, in_y} == 2'b10) next_state = 2'b00;
         else next_state = 2'b01;
endcase
endmodule

(A) 011000011101 (B) 011000111001
(C) 011000010001 (D) 001000011001
(E) 011000011001 (F) OTHER

© THE UNIVERSITY OF DUBLIN 2018