Paul Kafka

10/2/13

Embedded System Design

Homework 4

**74163 Verilog**

module hw4a (LOAD, DATA\_A, DATA\_B, clk, DATA\_C, DATA\_D, CLR, ENP, ENT, Q\_A, Q\_B, Q\_C, Q\_D, RC0);

input LOAD;

input DATA\_A;

input DATA\_B;

input clk;

input DATA\_C;

input DATA\_D;

input CLR;

input ENP;

input ENT;

inout Q\_A;

inout Q\_B;

inout Q\_C;

inout Q\_D;

output RC0;

reg[5:0] a;

reg[5:0] b;

reg[5:0] c;

reg[5:0] d;

reg[3:0] temp;

reg first;

*//JK Flip Flops*

always @(negedge clk)

begin

*// Combinational Logic*

first = (ENP & ENT);

*//For Q\_A*

a[0] = (~ LOAD) | (~ CLR);

a[1] = ~(a[0] & a[3]);

a[2] = first | a[0];

a[3] = ~(~(DATA\_A & CLR) & a[0]);

a[4] = a[1] & a[2];

a[5] = a[2] & a[3];

*//For Q\_B*

b[0] = Q\_A & first;

b[1] = ~(a[0] & b[3]);

b[2] = b[0] | a[0];

b[3] = ~(~(DATA\_B & CLR) & a[0]);

b[4] = b[1] & b[2];

b[5] = b[2] & b[3];

*//For Q\_C*

c[0] = Q\_B & Q\_A & first;

c[1] = ~(a[0] & c[3]);

c[2] = c[0] | a[0];

c[3] = ~(~(DATA\_C & CLR) & a[0]);

c[4] = c[1] & c[2];

c[5] = c[2] & c[3];

*//For Q\_D*

d[0] = Q\_C & Q\_B & Q\_A & first;

d[1] = ~(a[0] & d[3]);

d[2] = a[0] | d[0];

d[3] = ~(~(DATA\_D & CLR) & a[0]);

d[4] = d[1] & d[2];

d[5] = d[2] & d[3];

*//a(4) is J and a(5) is K*

if (a[4] & ~a[5])

temp[0] = 1;

else if (~a[4] & a[5])

temp[0] = 0;

else if (a[4] & a[5])

temp[0] = ~temp[0];

*//b(4) is J and b(5) is K*

if (b[4] & ~b[5])

temp[1] = 1;

else if (~b[4] & b[5])

temp[1] = 0;

else if (b[4] & b[5])

temp[1] = ~temp[1];

*//c(4) is J and c(5) is K*

if (c[4] & ~c[5])

temp[2] = 1;

else if (~c[4] & c[5])

temp[2] = 0;

else if (c[4] & c[5])

temp[2] = ~temp[2];

*//d(4) is J and d(5) is K*

if (d[4] & ~d[5])

temp[3] = 1;

else if (~d[4] & d[5])

temp[3] = 0;

else if (d[4] & d[5])

temp[3] = ~temp[3];

end

*//Outputs*

wire Q\_A = temp[0];

wire Q\_B = temp[1];

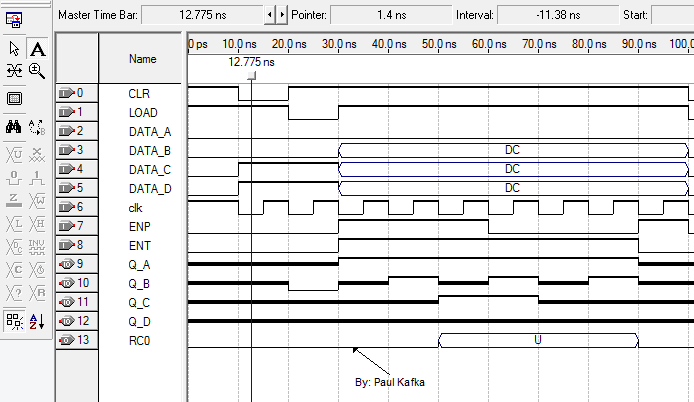
wire Q\_C = temp[2];

wire Q\_D = temp[3];

wire RC0 = Q\_D & Q\_C & Q\_B & Q\_A & ENT;

endmodule

**Simulation**



**74LS181 Verilog**

module hw4b(a, b, s, f);

input[3:0] a;

input[3:0] b;

input[3:0] s;

output[3:0] f;

reg[3:0] f;

always@(a or b or s)

begin

case(s)

4'b0000:

f = ~a;

4'b0001:

f = ~(a & b);

4'b0010:

f = ~a | b;

4'b0011:

f = "0001";

4'b0100:

f = ~(a | b);

4'b0101:

f = ~b;

4'b0110:

f = ~(a ^ b);

4'b0111:

f = a | ~b;

4'b1000:

f = ~a & b;

4'b1001:

f = a ^ b;

4'b1010:

f = b;

4'b1011:

f = a | b;

4'b1100:

f = "0000";

4'b1101:

f <= a & ~b;

4'b1110:

f = a & b;

4'b1111:

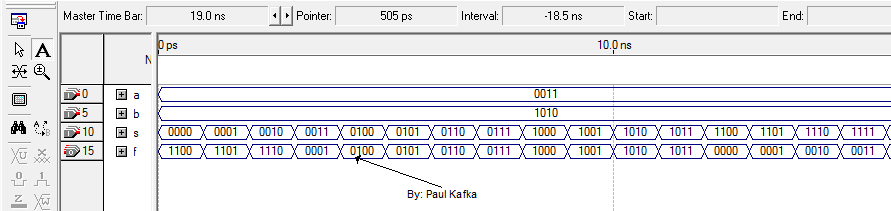
f = a;

endcase

end

endmodule

**Simulation**



**Extra Credit**

|  |  |
| --- | --- |
| VHDL – Verilog Comparison for 74163 | |
| VHDL | Verilog |
| library ieee;  use ieee.std\_logic\_1164.all; | No Libraries |
| --Ports  entity hw3 is  port  (  LOAD : in std\_logic;  DATA\_A : in std\_logic;  DATA\_B : in std\_logic;  clk : in std\_logic;  DATA\_C : in std\_logic;  DATA\_D : in std\_logic;  CLR : in std\_logic;  ENP : in std\_logic;  ENT : in std\_logic;  Q\_A : buffer std\_logic;  Q\_B : buffer std\_logic;  Q\_C : buffer std\_logic;  Q\_D : buffer std\_logic;  RC0 : out std\_logic  );  end entity; | //Ports  module hw4a (LOAD, DATA\_A, DATA\_B, clk, DATA\_C, DATA\_D, CLR, ENP, ENT, Q\_A, Q\_B, Q\_C, Q\_D, RC0);  input LOAD;  input DATA\_A;  input DATA\_B;  input clk;  input DATA\_C;  input DATA\_D;  input CLR;  input ENP;  input ENT;  inout Q\_A;  inout Q\_B;  inout Q\_C;  inout Q\_D;  output RC0; |
| --Variables  architecture rtl of hw3 is  signal a : std\_logic\_vector (5 downto 0);  signal b : std\_logic\_vector (5 downto 0);  signal c : std\_logic\_vector (5 downto 0);  signal d : std\_logic\_vector (5 downto 0);  signal temp: std\_logic\_vector (3 downto 0);  signal first : std\_logic; | //Variables  reg[5:0] a;  reg[5:0] b;  reg[5:0] c;  reg[5:0] d;  reg[3:0] temp;  reg first; |
| *-- Combinational Logic*  first <= ENP AND ENT;    *--For Q\_A*  a(0) <= (NOT LOAD) OR (NOT CLR);  a(1) <= a(0) NAND a(3);  a(2) <= first OR a(0);  a(3) <= (DATA\_A NAND CLR) NAND a(0);  a(4) <= a(1) AND a(2);  a(5) <= a(2) AND a(3);    *--For Q\_B*  b(0) <= Q\_A AND first;  b(1) <= a(0) NAND b(3);  b(2) <= b(0) OR a(0);  b(3) <= (DATA\_B NAND CLR) NAND a(0);  b(4) <= b(1) AND b(2);  b(5) <= b(2) AND b(3);    *--For Q\_C*  c(0) <= Q\_B AND Q\_A AND first;  c(1) <= a(0) NAND c(3);  c(2) <= c(0) OR a(0);  c(3) <= (DATA\_C NAND CLR) NAND a(0);  c(4) <= c(1) AND c(2);  c(5) <= c(2) AND c(3);    *--For Q\_D*  d(0) <= Q\_C AND Q\_B AND Q\_A AND first;  d(1) <= a(0) NAND d(3);  d(2) <= a(0) OR d(0);  d(3) <= (DATA\_D NAND CLR) NAND a(0);  d(4) <= d(1) AND d(2);  d(5) <= d(2) AND d(3); | *// Combinational Logic*  first = (ENP & ENT);    *//For Q\_A*  a[0] = (~ LOAD) | (~ CLR);  a[1] = ~(a[0] & a[3]);  a[2] = first | a[0];  a[3] = ~(~(DATA\_A & CLR) & a[0]);  a[4] = a[1] & a[2];  a[5] = a[2] & a[3];    *//For Q\_B*  b[0] = Q\_A & first;  b[1] = ~(a[0] & b[3]);  b[2] = b[0] | a[0];  b[3] = ~(~(DATA\_B & CLR) & a[0]);  b[4] = b[1] & b[2];  b[5] = b[2] & b[3];    *//For Q\_C*  c[0] = Q\_B & Q\_A & first;  c[1] = ~(a[0] & c[3]);  c[2] = c[0] | a[0];  c[3] = ~(~(DATA\_C & CLR) & a[0]);  c[4] = c[1] & c[2];  c[5] = c[2] & c[3];    *//For Q\_D*  d[0] = Q\_C & Q\_B & Q\_A & first;  d[1] = ~(a[0] & d[3]);  d[2] = a[0] | d[0];  d[3] = ~(~(DATA\_D & CLR) & a[0]);  d[4] = d[1] & d[2];  d[5] = d[2] & d[3]; |
| *--JK Flip Flops*  process(a,b,c,d, clk)      begin  *--a(4) is J and a(5) is K*  if (clk 'EVENT AND clk = '1') then  if (a(4) ='0' and a(5)='0') then  temp(0) <= temp(0);  elsif (a(4)='0' and a(5)='1') then  temp(0) <= '0';  elsif (a(4)='1' and a(5)='0') then  temp(0) <= '1';  elsif (a(4)='1' and a(5)='1') then  temp(0) <= not (temp(0));  end if;  end if;    if (clk 'EVENT AND clk = '1') then  if (b(4) ='0' and b(5)='0') then  temp(1) <= temp(1);  elsif (b(4)='0' and b(5)='1') then  temp(1) <= '0';  elsif (b(4)='1' and b(5)='0') then  temp(1) <= '1';  elsif (b(4)='1' and b(5)='1') then  temp(1) <= not (temp(1));  end if;  end if;    if (clk 'EVENT AND clk = '1') then  if (c(4) ='0' and c(5)='0') then  temp(2) <= temp(2);  elsif (c(4)='0' and c(5)='1') then  temp(2) <= '0';  elsif (c(4)='1' and c(5)='0') then  temp(2) <= '1';  elsif (c(4)='1' and c(5)='1') then  temp(2) <= not (temp(2));  end if;  end if;    if (clk 'EVENT AND clk = '1') then  if (d(4) ='0' and d(5)='0') then  temp(3) <= temp(3);  elsif (d(4)='0' and d(5)='1') then  temp(3) <= '0';  elsif (d(4)='1' and d(5)='0') then  temp(3) <= '1';  elsif (d(4)='1' and d(5)='1') then  temp(3) <= not (temp(3));  end if;  end if; | *// JK Flip Flops*  *//a(4) is J and a(5) is K*  if (a[4] & ~a[5])  temp[0] = 1;  else if (~a[4] & a[5])  temp[0] = 0;  else if (a[4] & a[5])  temp[0] = ~temp[0];  *//b(4) is J and b(5) is K*  if (b[4] & ~b[5])  temp[1] = 1;  else if (~b[4] & b[5])  temp[1] = 0;  else if (b[4] & b[5])  temp[1] = ~temp[1];    *//c(4) is J and c(5) is K*  if (c[4] & ~c[5])  temp[2] = 1;  else if (~c[4] & c[5])  temp[2] = 0;  else if (c[4] & c[5])  temp[2] = ~temp[2];    *//d(4) is J and d(5) is K*  if (d[4] & ~d[5])  temp[3] = 1;  else if (~d[4] & d[5])  temp[3] = 0;  else if (d[4] & d[5])  temp[3] = ~temp[3]; |
| *--Outputs*  Q\_A <= temp(0);  Q\_B <= temp(1);  Q\_C <= temp(2);  Q\_D <= temp(3);  RC0 <= Q\_D AND Q\_C and Q\_B AND Q\_A AND ENT; | *//Outputs*  wire Q\_A = temp[0];  wire Q\_B = temp[1];  wire Q\_C = temp[2];  wire Q\_D = temp[3];  wire RC0 = Q\_D & Q\_C & Q\_B & Q\_A & ENT;  endmodule |

|  |  |
| --- | --- |
| VHDL – Verilog Comparison for 74LS181 | |
| VHDL | Verilog |
| library ieee;  use ieee.std\_logic\_1164.all; | No Library |
| entity hw3b is  port  (  a : in std\_logic;  b : in std\_logic;  s : in std\_logic\_vector (3 downto 0);    f : out std\_logic  );  end entity; | module hw4b(a, b, s, f);  input[3:0] a;  input[3:0] b;  input[3:0] s;  output[3:0] f; |
| architecture logic of hw3b is  begin | reg[3:0] f; |
| process(a,b,s)    begin  case s is  when "0000" =>  f <= NOT a;  when "0001" =>  f <= NOT (a and b);  when "0010" =>  f <= NOT a or b;  when "0011" =>  f <= '1';  when "0100" =>  f <= NOT (a or b);  when "0101" =>  f <= NOT b;  when "0110" =>  f <= NOT (a xor b);  when "0111" =>  f <= a or NOT b;  when "1000" =>  f <= not a and b;  when "1001" =>  f <= a xor b;  when "1010" =>  f <= b;  when "1011" =>  f <= a or b;  when "1100" =>  f <= '0';  when "1101" =>  f <= a and not b;  when "1110" =>  f <= a and b;  when "1111" =>  f <= a;  end case;  end process; | always@(a or b or s)  begin  case(s)  4'b0000:  f = ~a;  4'b0001:  f = ~(a & b);  4'b0010:  f = ~a | b;  4'b0011:  f = "0001";  4'b0100:  f = ~(a | b);  4'b0101:  f = ~b;  4'b0110:  f = ~(a ^ b);  4'b0111:  f = a | ~b;  4'b1000:  f = ~a & b;  4'b1001:  f = a ^ b;  4'b1010:  f = b;  4'b1011:  f = a | b;  4'b1100:  f = "0000";  4'b1101:  f <= a & ~b;  4'b1110:  f = a & b;  4'b1111:  f = a;  endcase  end |
| end logic; | endmodule |