EE224: End-semester Test

24 April 2014, 1730-2030 hrs

All questions have 5 marks. Make suitable assumptions if needed, and mention them clearly.

1. Suppose $f(x_1, x_2, x_3, x_4)$ is a function of four variables. Derive a procedure to check if it is possible to factorize

$$f(x_1, x_2, x_3, x_4) = g(x_1, x_2).h(x_3, x_4)$$

where g and h are functions of two variables (5 marks). Apply your procedure to find suitable factors g, h (or show that no such factors exist) if

$f(x_1, x_2, x_3, x_4) = x_1.(x_4.x_2.\overline{x}_3 + x_3.\overline{x}_4.x_2) + \overline{x}_1.(\overline{x}_3.\overline{x}_2.\overline{x}_4 + \overline{x}_2.\overline{x}_3.x_4)$.

   - If such a factorization is possible then, for every substitution of values for x_1, x_2 in f one should end up with either 0 or h. For the example given above, $f_{x_1x_2} = x_3 \oplus x_4$, $f_{\overline{x}_1x_2} = 0$, $f_{x_1\overline{x_2}} = 0$ and $f_{\overline{x}_1\overline{x}_2} = \overline{x}_3$. Thus, the condition is not satisfied and f cannot be factorized.

2. Consider the two state-machines shown in Figure 1. The left-machine has states $\{S0, S1\}$, input symbols $\{rL, f, s\}$ and output symbols $\{rR, u, v\}$. The right-machine has states $\{T0, T1\}$, input symbols $\{rR, u, v\}$ and output symbols $\{d, n\}$. The output of the left-machine is the input to the right-machine. Taken together, the pair of state machines is equivalent to a single FSM with four states, input symbols $\{rL, f, s\}$ and output symbols $\{d, n\}$. Draw the state-transition graph of this equivalent FSM.

   - The resulting machine is shown in Figure 1.

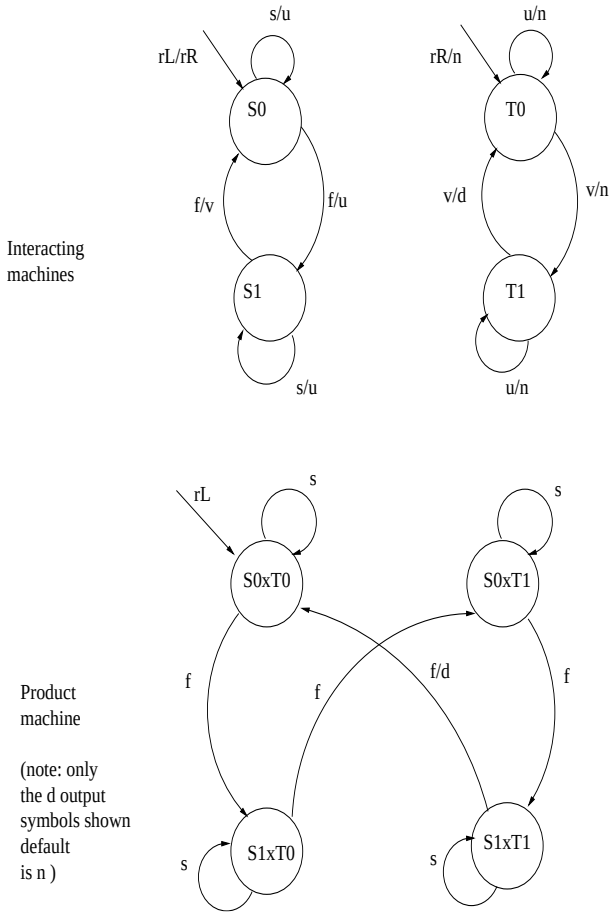

Figure 1: Interacting FSMs (with product FSM shown)

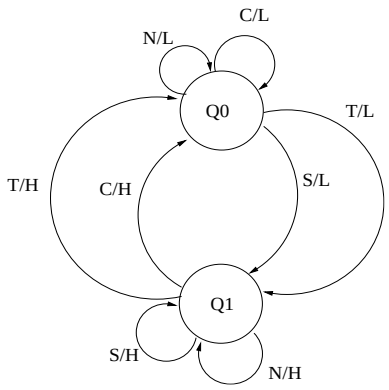

Figure 2: FSM specification

3. Using two input NAND gates, inverters and D-flipflops, implement the state machine shown in Figure 2. The input symbols are $\{S, C, T, N\}$, the output symbols are $\{H, L\}$ and the states are $\{Q0, Q1\}$. Clearly show your input, output and state encodings and any intermediate work you do in getting to your implementation. Assume that all gates have unit delay, and the D-flipflop has delay, setup and hold times which are all equal to 1 unit. For your solution, find the

   - input setup times relative to clock (for each input).
   - input hold times relative to clock (for each input).
   - delay from clock to output.

   - We will encode the input symbols using two variables J, K as follows

Symbol	J	K
S	1	0
C	0	1
T	1	1
N	0	1

     the output symbols by a single variable y

Symbol	y
H	1
L	0

     and the state symbols by a single variable q

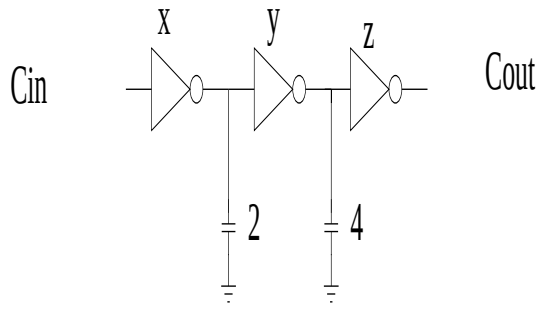

Figure 3: Inverter Chain

Symbol	q
Q0	0
Q1	1

Then, the next-state equation becomes

$$J(\overline{K} + \overline{q}) = \overline{\overline{J.\overline{K}.q}}$$

and the output equation becomes

$$y = q$$

The setup time relative to the positive edge of clock is then 4 units (delay from K to FF-input is 3 units), and the hold time is -1 units (delay from J to FF-input is 2 units). The delay from clock to q is 1 unit. The minimum clock period (due to FF-to-FF delay which is 4 units) is 5 units.

4. Consider the circuit shown in Figure 3. The capacitances shown correspond to wire capacitances. The variables x, y, z are input capacitances of the three gates. For the circuit, it is required that $C_{in} = 1$ and $C_{out} = 4$ units. Write the total delay of the circuit as a function of x, y, z (in τ units). For what values of x, y, z is the minimum delay achieved (assume that x, y, z are positive real numbers)? What is the minimum delay?

   - Clearly, we can assume that $x = 1$. Thus the delay (assume that the parasitic delay of each inverter is 1 unit) is:

$$(2 + y) + \frac{z + 4}{y} + \frac{4}{z} + 3$$

For a given $y > 0$, this delay is minimized when $z = 2\sqrt{y}$. Substitute into the equation, and we need to find $y > 0$ to minimize

$$(2 + y) + \frac{2\sqrt{y} + 4}{y} + \frac{4}{2\sqrt{y}} + 3$$

which is minimized somewhere near $y = 3$ and thus $z = 3.4$ (you can search a bit near $y = 3$ to find the "best" value).

5. The pseudo-code in Figure 4 describes an 8-bit divider. Simulate the pseudo code (calculate the state and register values at the different k instants) until the div_done output signal becomes 1. The system starts in div_reset at $k = 0$ and at $k = 1$, div_start is 1. The divisor input is maintained at 00011100 and the dividend input is maintained at 10101110 throughout the period of interest. Does it work?

   - Yes, it works.

6. Consider the VHDL description of a piece of logic.

```
-- a, b, c, d are bits.
process(a,b,c,d, clk)
  variable x,y: bit;
begin
  x := (a or b);
  if(b = '1') then
    x := (not x);
 else
    x := (x or c);
 end if;
 if(c = '1') then
   x := (x and d);
 end if;
 e <= x;
 y := (not (x and (a or b)));
 if(clk'event and clk = '1') then
   q <= y;
 end if;
end process;
```

   Draw a logic network which is equivalent to this process statement (you may use AND, OR, NOT gates and D-flip-flops).

```
// divisor, dividend are 8-bit inputs.
// remainder, dividend_by_2 shifted_divisor,
// quotient are all 8-bits wide registers.
reset:
 if div_start then
    goto div_outerloop/
          {remainder = dividend || quotient = 0}
 else
    goto div_reset
 endif

div_outerloop:
 if (divisor <= remainder) then
    goto div_innerloop/
        { curr_quotient = "0000000000000001" ||
            dividend_by_2 = (remainder >> 1) ||
            shifted_divisor = divisor}
 else
    goto div_completed
 endif

div_innerloop:
 if (shifted_divisor < dividend_by_2) then
    goto div_innerloop/
      { shifted_divisor = (shifted_divisor << 1) ||
        curr_quotient = (curr_quotient << 1) }
 else
    goto div_update
 endif

div_update:
 goto div_outerloop/
      {quotient = (quotient OR curr_quotient) ||
       remainder = (remainder - shifted_divisor)}

div_completed:
 goto div_reset/{emit div_done}
```

Figure 4: Divider

- Working from the beginning to the end, we find that when e is assigned to x, its value in terms of a, b, c, d is

$$e = t$$

where

$$t = (c?((b?((a+b)+c) : \overline{a+b}) + d) \ : \ (b?((a+b)+c) \ : \ \overline{a+b}))$$

which is a combinational logic network consiting of three multiplexors and some OR's and a NOR. This can be simplified a bit

$$\begin{aligned} u &= (a+b) \\ v &= (u+c) \\ p &= \overline{u} \\ w &= (b?v : p) \\ r &= (w+d) \\ t &= (c?r : w) \end{aligned}$$

which uses three OR2 gates, one NOT gate and two multiplexors (each multiplexor can be implemented using a NOT gate and three NAND2 gates). Then q is

$$\begin{aligned} q &= (clk \uparrow ? \ \overline{t.(a+b)} \ : \ q) \\ &= (clk \uparrow ? \ m \ : \ q) \\ m &= \overline{t.u} \end{aligned}$$

which is implemented with a D-flipflop whose input is m.

7. You are asked to design a system which receives a sequence of bits $\{x(k)\}$ and produces an output sequence of bits $\{y(k)\}$ such that $y(k) = 1$ if and only if the bits $x(0), x(1), \ldots x(k)$ form a palindrome (that is a sequence which reads the same from left to right as it does from right to left, for example $1, 1, 0, 1, 1$). Can you implement such a system using a Mealy finite state machine? If yes, give the state transition graph of the FSM. If not, why not? (5 marks).

- Clearly, the system will have to have infinite memory, because at each k it will have to remember *all* the input bits up to k (otherwise it will not be able to distinguish a palindrome from a non-palindrome). It cannot be implemented as a finite-state machine.

8. Suppose you are given a collection of combinational gates each of which is a four-input programmable logic block which can implement any four-input Boolean function. Using these gates, implement a majority voter function which has eight input bits and outputs a 1 if and only if least 5 of its input bits are 1 (5 marks).

    - There are several ways to do it. Note that $\overline{f}$ and f can be implemented using the same structure of interconnected gates (inverters are "free"). Instead of implementing the majority function we implement the minority function which is 1 if and only if at most three of its inputs are 1. Divide the input into two halves. Let l_0, l_1, l_2 be functions of the lower 4 bits which are 1 if at most $0, 1, 2$ of the lower four bits is 1. Similarly, let h_0, h_1, h_2 be the corresponding functions of the upper four bits. Then the minority function is $l_0(h_0 + h_1 + h_2) + l_1(h_1 + h_2) + l_2 h_0$. The h and the l functions can be implemented using 6 blocks and the final combination can be done using 3 blocks. Maybe you can do better?

9. Design a CMOS gate to implement the Boolean function $\overline{(A + B).(C + D)}$ with a single size parameter W and size it so that the pull-up and pull-down networks are equivalent to a $2W$ PMOS transistor and W NMOS transistor respectively. What is the logical effort of the gate relative to that of a standard inverter?

    - This is easy. The logical effort relative to the standard inverter is 2 (for the same drive strength as the standard $2W/W$ inverter, each input of the gate sees a load which is $6W$).

10. Do you agree or disagree with the following statements? In each case, give a short justification for your opinion.

    - In a well designed CMOS circuit, all nodes should have approximately the same rise/fall times: Yes, because this implies that all are operating at similar efforts, and also, the crossover or short-circuit currents are controlled.
    - In a well designed CMOS circuit, all paths should have approximately the same delays: Yes, otherwise, it is possible to improve the circuit by increasing the delay of the non-critical paths.
    - In a well designed CMOS circuit, all gates should be minimum size: No, this can lead to terrible circuit performance. One must size the gates so that efforts are equally distributed.

- A digital system with many clocks is easier to implement and verify than one with a single clock: No, clock skew is difficult to control.
- When implementing a digital system, most of the time should be spent in VHDL/Verilog coding: No, most of the time should be spent in perfecting the algorithm and proving its correctness. VHDL/Verilog coding should be a routine implementation of the algorithm.

1 Further Reading after this Course

To those of you are interested in reading further, the following books are likely to be useful.

- To go deeper into Boolean algebra, switching theory, finite state machines, Turing machines etc., check out M.A. Harrison, "Introduction to Switching and Automata Theory," McGraw-Hill. It goes pretty deep. Kohavi's book is good for an introduction.
- For a more "systems" perspective, see Hill and Peterson, "Digital Systems: Hardware Organization and Design," Wiley.