CSE370 Final Exam Solution (11 December 2006)

Please read through the entire examination first! This exam was designed to be completed in 110 minutes (one hour and 50 minutes) and, hopefully, this estimate will be reasonable.

There are 3 problems for a total of 100 points. The point value of each problem is indicated in the table below. Each problem and sub-problem is on a separate sheet of paper. Write your answer neatly in the space provided. If you need more space (you shouldn't), you can write on the back of the sheet where the question is posed, but please make sure that you indicate clearly the problem to which the comments apply. Do NOT use any other paper to hand in your answers. If you have difficulty with part of a problem, move on to the next one. They are mostly independent of each other.

The exam is CLOSED book and CLOSED notes. Please do not ask or provide anything to anyone else in the class during the exam. Make sure to ask clarification questions early so that both you and the others may benefit as much as possible from the answers.

Good luck and have a great holiday break.

Name:

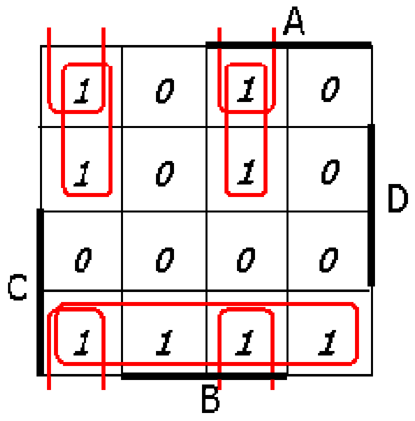
ID#:

Problem	**Max Score**	**Score**
1	25	***25***
2	50	***50***
3	25	***25***
TOTAL	**100**	***100***

1. Combinational Logic (25 points)

(a – 5 pts) Map the following function on the Karnaugh-map provided below and re-write it in canonical sum-of-products form. Please use minterm notation (e.g., m_3) instead of the algebraic form (e.g., A'B'CD).

$$Z = ((A' \text{ xor } B) + C)\,(CD)'$$



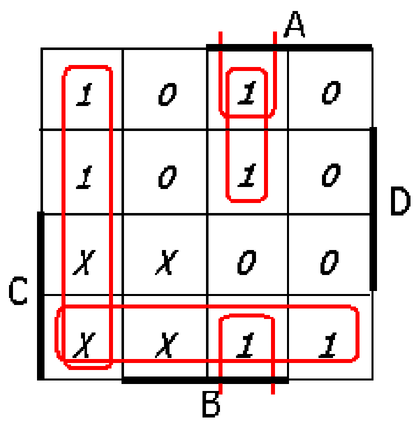
A'B'C'   A'B'D'   ABC'   ABD'   C'D'

$\boldsymbol{m_0 + m_1 + m_2 + m_6 + m_{10} + m_{12} + m_{13} + m_{14}}$

(b – 5 pts) Re-write the function in minimized sum-of-products form and highlight each term that represents an essential prime implicant by underlining it. Make sure to circle each term on the K-map above.

Z = $\underline{\boldsymbol{A'B'C'}} + \underline{\boldsymbol{ABC'}} + \underline{\boldsymbol{C'D'}}$

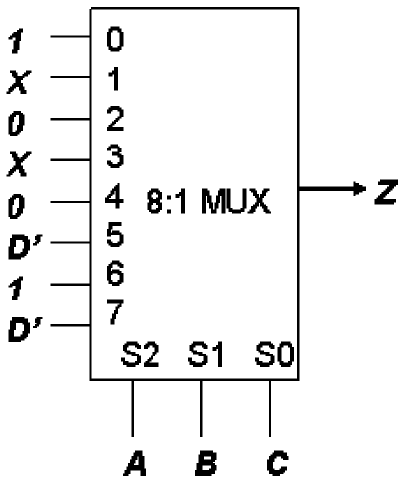
(c – 5 pts) Add don’t cares to the K-map you derived in part (a) for the cells covered by A’C. Redraw you new K-map below, circle all prime implicants, and then list the essential and non-essential prime implicants for this new function.



Essential: ***C’D’ A’B’ ABC’***

Non-essential: ***ABD’***

(d – 10pts) Implement the function of part (c) using an 8:1 multiplexer (provided below) and at MOST one inverter. S2 is the most significant bit of the control signals, S0 is the least significant (e.g., S2=1, S1=1, S0=0 selects input #6).

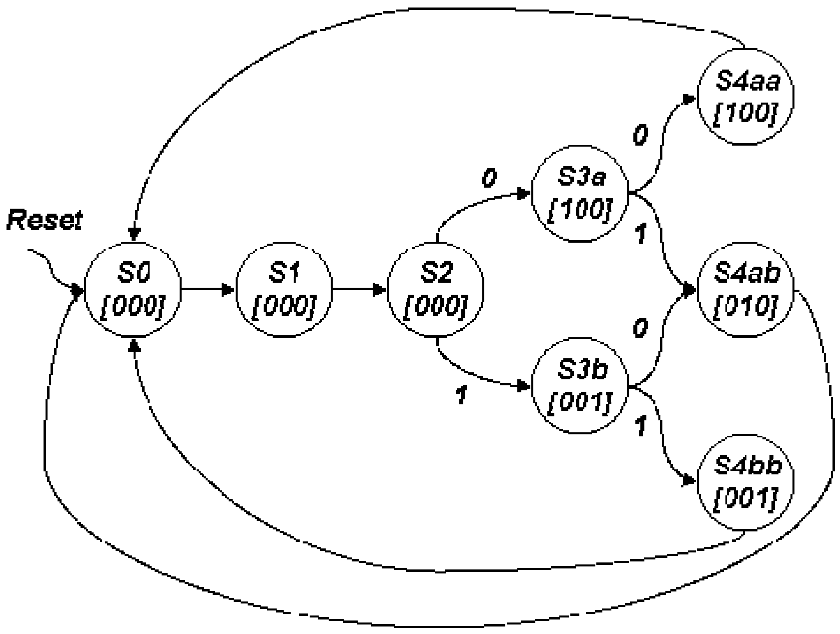


2. Finite State Machines (45 points)

The following Verilog was found among old papers in a dusty drawer of a now defunct dot-com company. Unfortunately, there were no comments in the code.

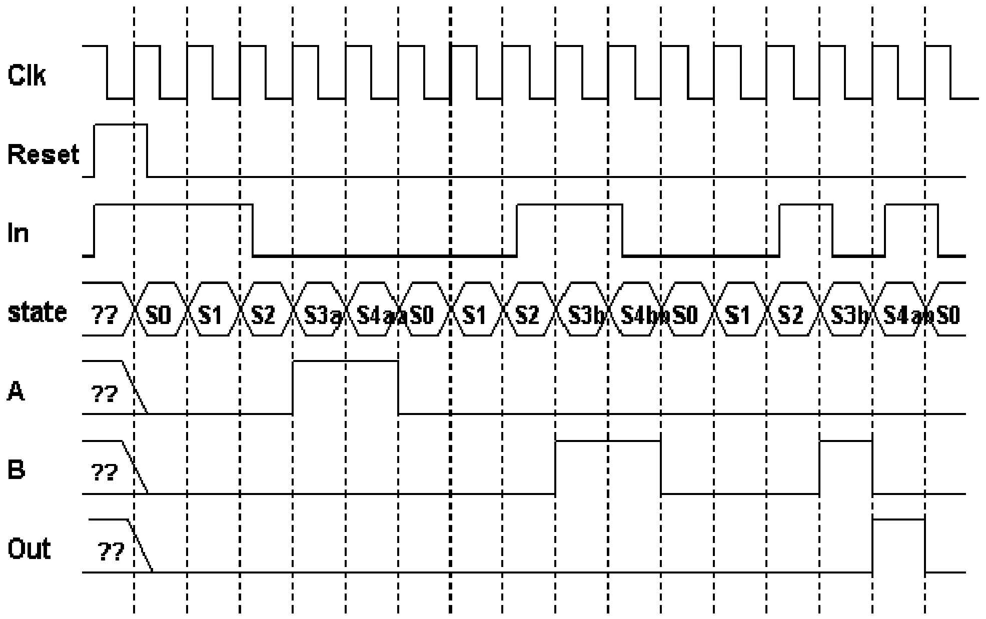
```
module Mystery (In, Clk, Reset, A, B, Out);
  input      In, Clk, Reset;
  output     A, B, Out;

  reg  [5:0]  state;
  reg  [5:0]  next_state;
  wire [2:0]  count;

  parameter S0   = 6'b000000;
  parameter S1   = 6'b000001;
  parameter S2   = 6'b000010;
  parameter S3a  = 6'b100011;
  parameter S3b  = 6'b001011;
  parameter S4aa = 6'b100100;
  parameter S4ab = 6'b010100;
  parameter S4bb = 6'b001100;

  always @(posedge Clk) begin
      if (Reset) begin state = `S0;        end
      else       begin state = next_state; end
  end

  always @(In or state) begin
      case(state)
        `S0:   next_state = `S1;
        `S1:   next_state = `S2;
        `S2:   if (In) next_state = `S3b;  else next_state = `S3a;
        `S3a:  if (In) next_state = `S4ab; else next_state = `S4aa;
        `S3b:  if (In) next_state = `S4bb; else next_state = `S4ab;
        `S4aa: next_state = `S0;
        `S4ab: next_state = `S0;
        `S4bb: next_state = `S0;
      endcase
  end

  assign count = state[2:0];
  assign A     = state[5];
  assign Out   = state[4];
  assign B     = state[3];

endmodule
```

(a – 10 pts) What type of state machine is this? Circle: Mealy, [✓ Moore], or synchronous-Mealy. Derive its state diagram and clearly label all transitions and the values for A, B, and Out in each state. Use the template below. Clearly label all transitions and outputs.



Output = [A, Out, B]

(b – 15 pts) Simulate the state machine for the following sample input waveforms. You are provided the values for the input signals, Reset and In. Fill in the details for the signals A, B, and Out. Also, please indicate the state the FSM is in for each clock cycle (i.e., write the symbolic name of the state in each cycle). Assume that the FSM is initially in an unknown state. Assume that all FFs are positive edge-triggered.



(c – 15pts) When we look to see how this state machine was actually implemented, we find that there is a 3-bit binary counter in the circuit and 2 individual D flip-flops with some very simple logic around them. We need to figure out if we have an up to date description for this module. The Verilog to describe the actual circuit implementation is provided below.

```
module MysteryImplementation (In, Clk, Reset, A, B, Out);
  input      In, Clk, Reset;
  output     A, B, Out;

  reg  [2:0]  counter;
  reg         A, B;

  always @(posedge Clk) begin
     if (Reset | counter[2]) counter = 0;
     else                    counter = counter + 1;
  end

  always @(posedge Clk) begin
     if (Reset | counter[2]) A = 0;
     else                    A = A | (counter[1] & ~In);
  end

  always @(posedge Clk) begin
     if (Reset | counter[2]) B = 0;
     else                    B = B | (counter[1] & In);
  end

  assign Out = A & B;

endmodule
```

(c – 15 pts continued) Is this an MysteryImplementation a correct implementation of the Mystery module? It turns out it is! In fact, Mystery and MysteryImplementation describe the same state diagram. What is the correspondence of states between them? For each of the states of the Mystery module list the states for the sequential elements of the MysteryImplementation module – note that there are 3 state elements in the MysteryImplementation: the counter (3 bits), the A FF (1 bit), and the B FF (1 bit).

Fill in the table below.

Mystery	**MysteryImplementation**		
state [5:0]	**counter [2:0]**	**A**	**B**
S0 = 6'b000000	*000*	*0*	*0*
S1 = 6'b000001	*001*	*0*	*0*
S2 = 6'b000010	*010*	*0*	*0*
S3a = 6'b100011	*011*	*1*	*0*
S3b = 6'b001011	*011*	*0*	*1*
S4aa = 6'b100100	*100*	*1*	*0*
S4ab = 6'b010100	*100*	*1*	*1*
S4bb = 6'b001100	*100*	*0*	*1*

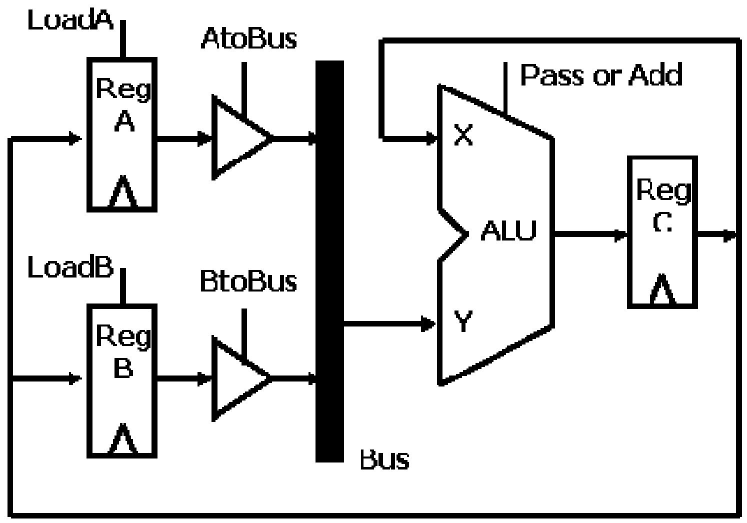
Since there are 3 separate state machines in MysteryImplementation (the counter with 8 states and 2 FFs with 2 states each) and the product of the states in each of the three would imply there are 32 states in this implementation. But that would only be true if all combinations of states are possible. That is clearly not the case as the counter goes right back to 0 when it reaches 4, so it only reaches 5 of its 8 states. The FF for A can only change to 1 if the counter is at 2 or 3. Thus, it can only be two different values for the counter's states 3 and 4. Similarly for B. However, both A and B can't change in the same cycle as they both rely on In. In the end, we have 5 states for the counter. In counter states 0, 1, and 2, the A and B FFs will both be 0. In counter state 3, the A and B FFs could be 01 or 10. In counter state 4, the A and B FFs could be 01, 10, or 11. That means we have a total of 3+2+3 states for a total of 8 which is the same as our original state diagram.

(d – 10 pts) Our next task is to verify that the output of the two machines is the same. Note that in MysteryImplementation, the output Out is specified using an assign statement. Describe in words, why this will be asserted in the same way as in the original module and why it is still the same type of FSM (Moore, Mealy, or synchronous-Mealy).

Out is still a Moore output, it is only a function of state bits. It will be true only in those states where both and A and B are true. That is only the case for MysteryImplementation in the second to the last row of the table above which is exactly when Out was true as part of the state of the Mystery machine.

3. Basic Computer Organization (25 points)

You are given the data-path below. Note that there are three registers. Two of these have a load control input (A and B), while the other (C) loads a new value on every clock cycle. There are two tri-state drivers that connect the outputs of registers A and B to a common bus. Finally, there is an ALU that can perform two operations: pass Y, add X and Y. X is always the output of register C, while Y is the value on the bus.



(a – 5 pts) Show the register-transfer operations needed to implement an instruction that swaps the contents of the A and B registers (*SWAP A, B*). Make sure to clearly indicate how many states will be needed to implement the instruction and the value of each control signal in each state. Assume the instruction is already in the instruction register so there is no need to worry about fetching the instruction or incrementing a program counter.

cycle	register-transfer operations	AtoBus	BtoBus	ALU	LoadA	LoadB
1	C ← A	1	0	Pass	0	0
2	*B ← C; C ← B*	*0*	*1*	*Pass*	*X*	*1*
3	*A ← C*	*0*	*0*	*X*	*1*	*0*

Comment [gb1]: This can be a don't care since we load A on the next cycle and don't need its current value any longer.

Comment [gb2]: Can't be don't cares because both could end up being 1 and that could be damaging!

(b – 5 pts) Show the register-transfer operations needed to implement an instruction that doubles the contents of the A register (`TWOX A`). Make sure to clearly indicate how many states will be needed to implement the instruction and the value of each control signal in each state. Assume the instruction is already in the instruction register so there is no need to worry about fetching the instruction or incrementing a program counter.

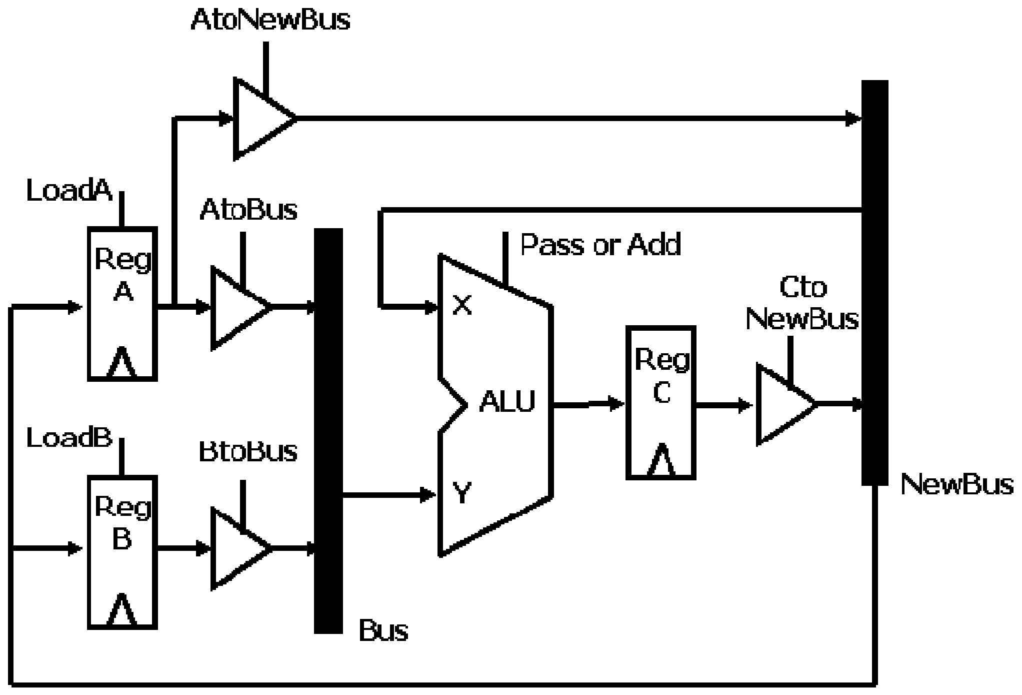
cycle	**register-transfer operations**	**AtoBus**	**BtoBus**	**ALU**	**LoadA**	**LoadB**
1	*C ← A*	*1*	*0*	*Pass*	*0*	*0*
2	*C ← C + A*	*1*	*0*	*Add*	*X*	*0*
3	*A ← C*	*0*	*0*	*X*	*1*	*0*

Comment [gb3]: The values in this column should not be don't cares because even though we don't use the value of B for this instruction, we shouldn't change it in case it is being used for something else.

Comment [gb4]: This can be a don't care for the same reason as in the instruction above.

Comment [gb5]: Can't be don't cares because both could end up being 1 and that could be damaging!

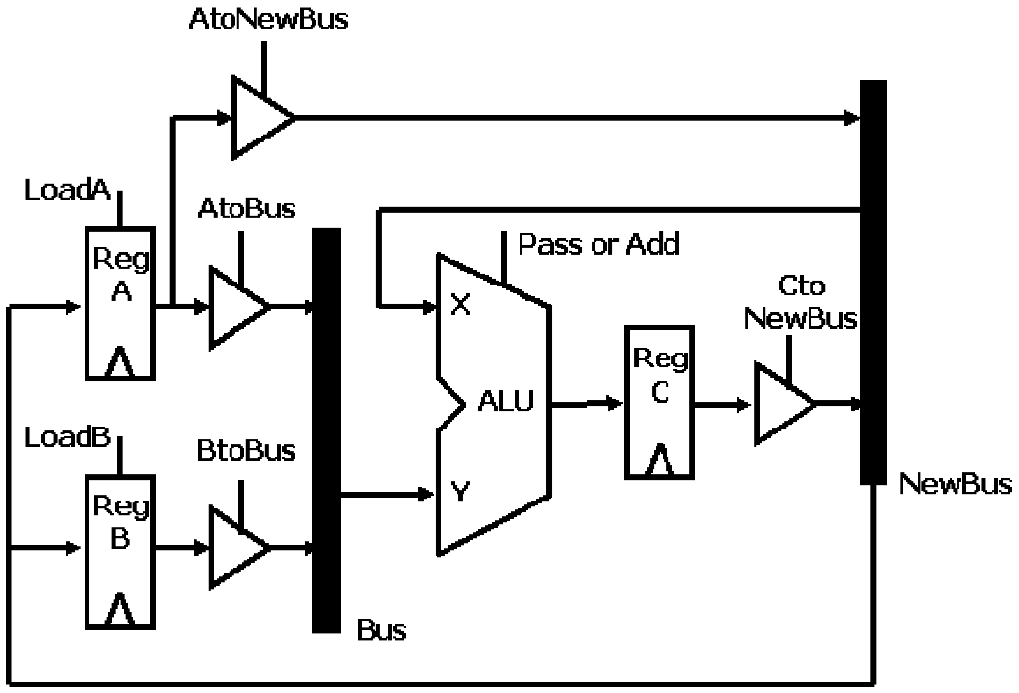
(c – 10 pts) How would you minimally change the architecture above so that you could perform the SWAP instruction in two cycles? You don't need to be concerned with the TWOX instruction for this question. Alter the data path diagram below.



The data path is modified with a new bus at the output of register C that also be driven from A. On the first cycle we load A into B using this new bus while B is moved to C. In the next cycle the contents of C (originally from B) is moved to A completing the swap.

cycle	register-transfer operations	Ato Bus	Bto Bus	ALU	Ato New Bus	Cto New Bus	Load A	Load B
1	*C ← B; B ← A;*	*0*	*1*	*Pass*	*1*	*0*	*0*	*1*
2	*A ← C;*	*X*	*X*	*X*	*0*	*1*	*1*	*0*

(d – 5 pts) How would you minimally change the architecture above so that you could perform the TWOX instruction in two cycles? You don't need to be concerned with the SWAP instruction for this question. Alter the data path diagram below.



Using the same data path as part (c), we enable A to the X input of the ALU (through the new bus) at the same as we connect A to Y through the original bus. After the result is stored in C, we simply use the next cycle to move it to A.

cycle	register-transfer operations	Ato Bus	Bto Bus	ALU	Ato New Bus	Cto New Bus	Load A	Load B
1	*C ← A + A;*	*1*	*0*	*Add*	*1*	*0*	*0*	*0*
2	*A ← C;*	*X*	*X*	*X*	*0*	*1*	*1*	*0*