University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Science

EECS 150
Fall 2005

R. H. Katz, Instructor
Po-Kai Chen, Head TA

SECOND MIDTERM EXAMINATION
Wednesday, 9 November 2005

INSTRUCTIONS—READ THEM NOW! This examination is CLOSED BOOK/CLOSED NOTES. There is no need for calculations, and so you will not require a calculator, Palm Pilot, laptop computer, or other calculation aid. Please put away your (camera) cell phones too! You MAY use one 8.5" by 11" double-sided crib sheet, as densely packed with notes, formulas, and diagrams as you wish. The examination has been designed for 55 minutes/55 points (1 point = 1 minute, so pace yourself accordingly). Nevertheless, you will have the full 80 minutes to work on it. All work should be done on the attached pages.

In general, if something is unclear, write down your assumptions as part of your answer. If your assumptions are reasonable, we will endeavor to grade the question based on them. If necessary, of course, you may raise your hand, and a TA or the instructor will come to you. Please try not to disturb the students taking the examination around you.

We will post solutions to the examination as soon as possible, and will grade the examination as soon as practical, usually within a week. Requests for regrades should be submitted IN WRITING, explaining why you believe your answer was incorrectly graded, within ONE WEEK of the return of the examination in class. We try to be fair, and do realize that mistakes can be made during the regarding process. However, we are not sympathetic to arguments of the form "I got half the problem right, why did I get a quarter of the points?"

SID: _____

(Signature)

(Name—Please Print!)

QUESTION	POINTS ASSIGNED	POINTS OBTAINED
1	5	
2	10	
3	5	
4	5	
5	10	
6	20	
TOTAL	55	

Student Name: __ SID:

Question 1. *Projects and Laboratories* (5 Points)

(a) Give one advantage of using Chipscope over the Logic Analyzer for purposes of debugging your design (1 point).

(b) Give one advantage of using the Logic Analyzer over Chipscope for the purposes of debugging your design (1 point).

(c) In your project checkpoint, why does the VideoEncoder need to have reset high for longer than 1 clock cycle (1 point)?

(d) Assume your Project Checkpoint #2 works perfectly so it has no bug[redacted] see scrolling when hooking it up with your implementation [redacted].

Question 2. *Verilog Synthesis* (10 points)
Consider the following Verilog description of a digital subsystem.

```
module X(EN, I, A, E0);
   input EN; input [ 3:0] I;
   output [ 1:0] A; output E0;
   reg [ 1:0] A; reg E0;

   always @ (EN or I or A or E0)
      begin
         if (EN == 0) begin E0 = 0; A = 0; end;
         else
            begin
               E0 = 1; A = 0;
               if (I[ 0] == 1) begin E0 = 0; A = 0; end;
               if (I[ 1] == 1) begin E0 = 0; A = 1; end;
               if (I[ 2] == 1) begin E0 = 0; A = 2; end;
               if (I[ 3] == 1) begin E0 = 0; A = 3; end;
            end
      end
end module
```

(a) To make sure that you understand the function that is being specified, complete the following table for the given inputs. NOTE: This is not a complete truth table! (1 point):

I_3	I_2	I_1	I_0	EN	A_1 A_0 E0
0	0	0	0	0	
0	0	0	0	1	
0	0	0	1	0	
0	0	0	1	1	
0	0	1	0	0	
0	0	1	0	1	
1	0	0	0	0	
1	0	0	0	1	
1	0	0	1	0	
1	0	0	1	1	

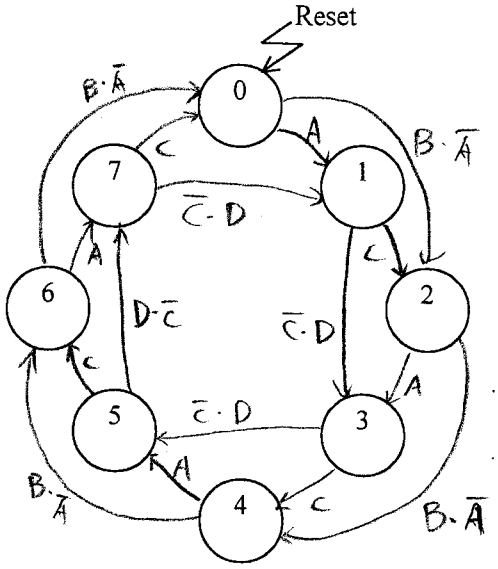
(b) Succinctly state *in a single succinct sentence* what function is being specified (1 point):

(c) The procedure for how Verilog synthesis handles `if-then-else` statements is to use 2:1 multiplexers to select one input if the IF condition is true and the other if it is false. Draw a schematic of this function using ONLY these muxes (8 points):

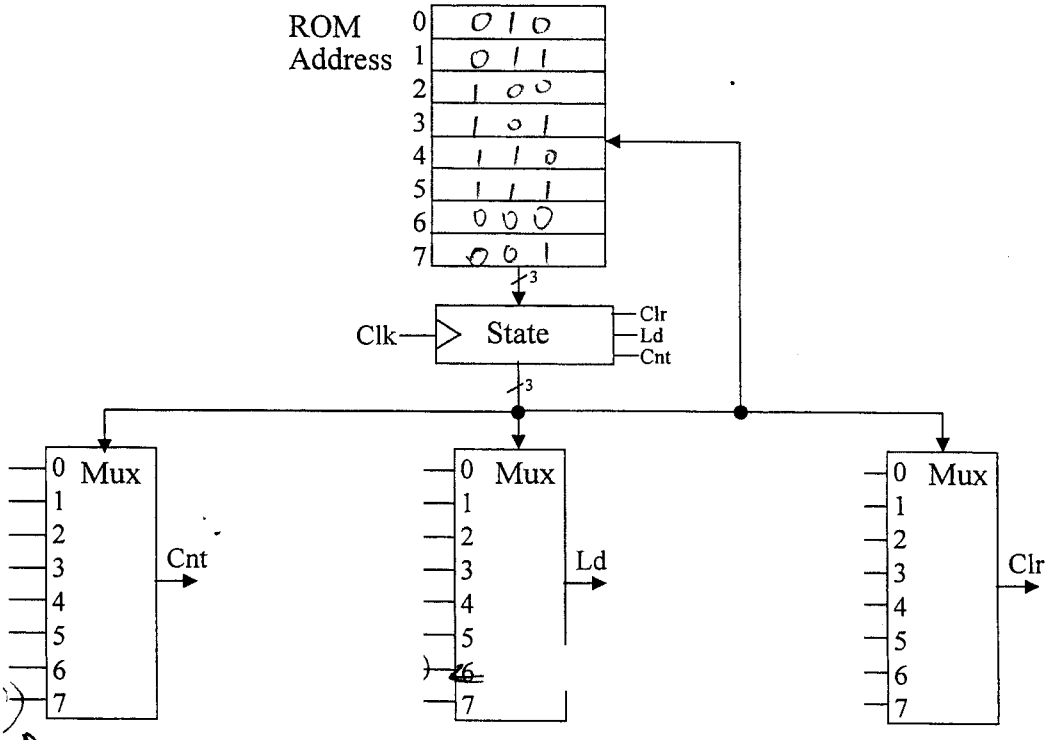
Question 3. *State Machine Implementation* (5 Points)

Consider the following state machine description for a 3-bit "even-odd by one or two up counter." The counter has 4 data inputs, denoted A, B, C, D, and reset. When reset is asserted, the counter is set to 0. When A is asserted, the counter counts up by 1 if it is in an even state. When B is asserted, the counter counts up by 2 if it is in an even state. When C is asserted, it counts up by 1 if it is in an odd state. When D is asserted, it counts up by 2 if it is in an odd state. Input A has priority over B, while C has priority over D, if more than one input is asserted at the same time. The 8 states form a ring, with the counter wrapping from high states to low states while counting.

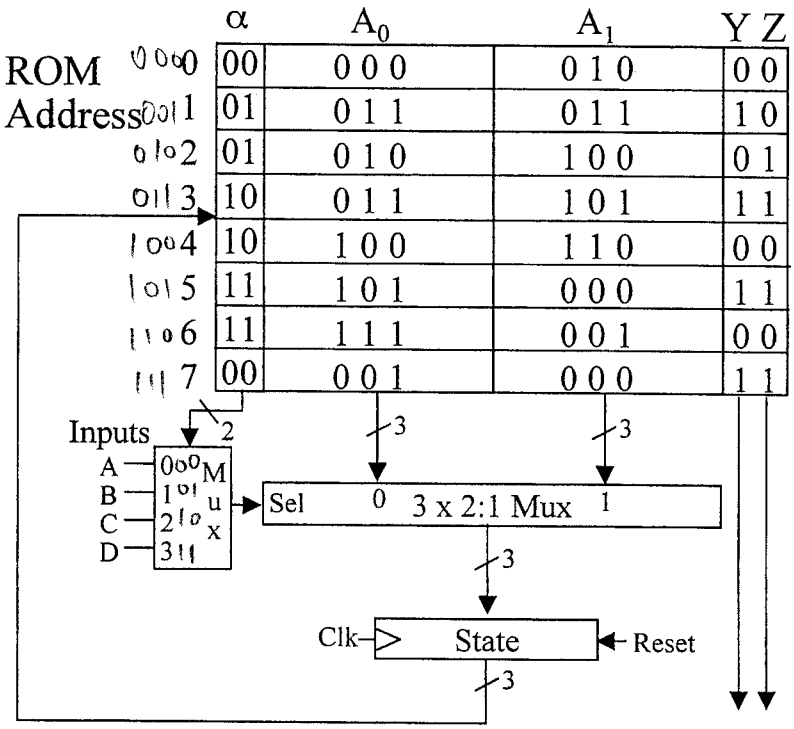
(a) To make sure you understand the problem, complete the following state diagram (1 points):

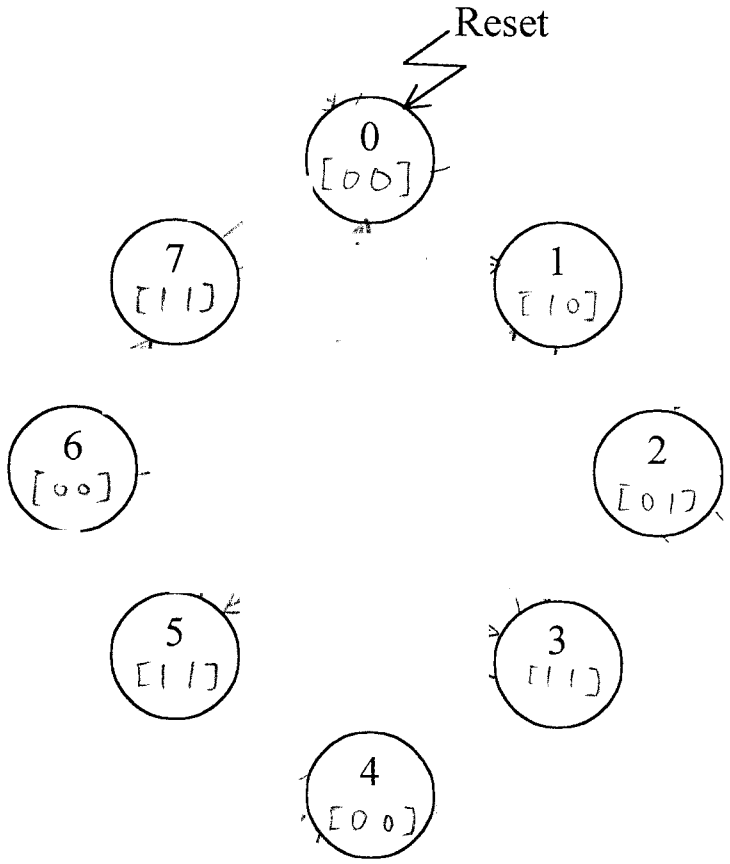


(b) Implement this state machine as a pure jump counter, showing how you wire the multiplexers and the contents of the jump ROM (4 points):



Question 4. *Reverse Engineering* (5 Points)
Given the following controller implementation and ROM contents, derive the state diagram it implements:





Question 5. *Controller Design* (10 Points)
You are to design the control for a datapath that performs the summation of two 8-bit sign and magnitude (S&M) numbers. The high order bit determines the sign, and the low order seven bits determine the magnitude.

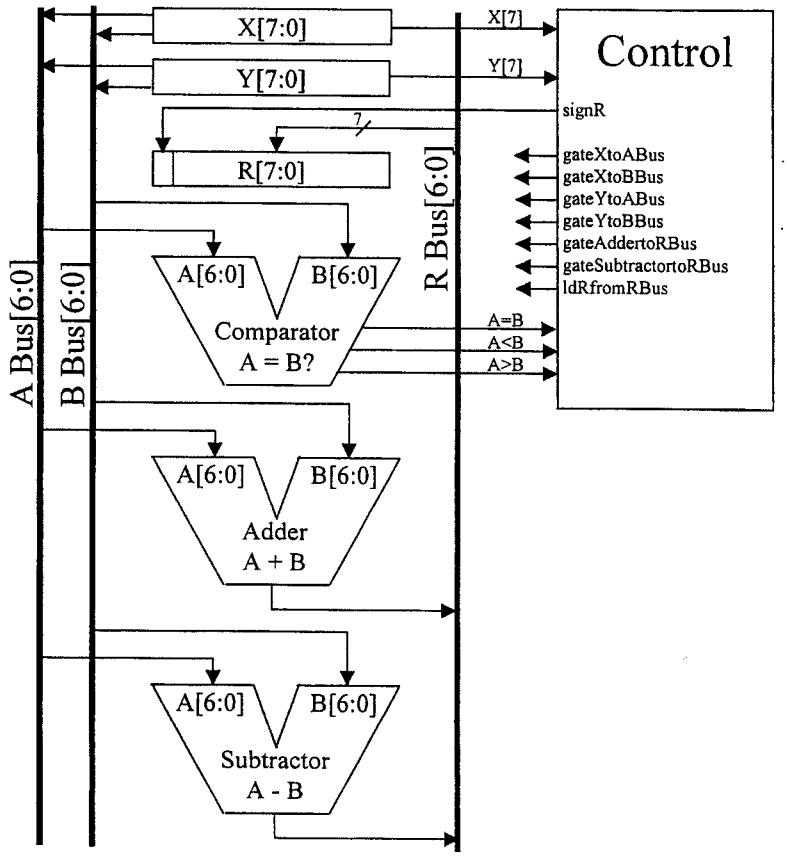
Consider the following cases:

(i) +3 + +4 = +7
(ii) –3 + –4 = –7
(iii) +3 + –4 = –1
(iv) –3 + +4 = +1

These cases illustrate the following two succinct rules:

(1) If the signs are the same, add the magnitudes of the operands, and assign their sign to the result;
(2) If the signs differ, subtract the smaller magnitude from the larger, and assign the sign of the larger magnitude to the result.

You are given the following datapath for this problem:



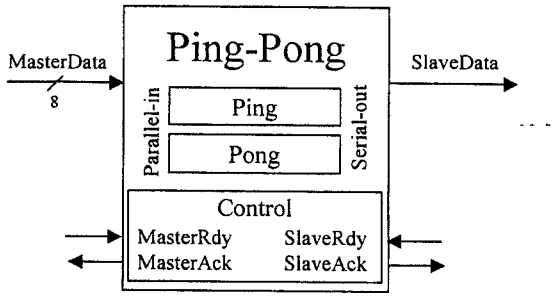
The control signals are defined as follows:

signR:	R[7] ← signR (R[7] written with new value when register R is loaded)
gateXtoABus:	ABus ← X[6:0]
gateXtoBBus:	BBus ← X[6:0]
gateYtoABus:	ABus ← Y[6:0]
gateYtoBBus:	BBus ← Y[6:0]
gateAddertoRBus:	RBus ← Adder[6:0]
gateSubtractortoRBus	RBus ← Subtractor[6:0]
ldRfromRBus:	R[6:0] ← RBus

Draw a MOORE MACHINE state diagram that adds two sign and magnitude numbers in registers X and Y, and places the correct result in register R. Abbreviate the signal names to save time, using the following notation for the eight control signals of the prior page: sR, gXA, gXb, gYb, gAR, gSR, lRR.

Question 6. *Subsystem Design* (20 Points)

You are to design a "Ping-Pong" register subsystem to the following specification. Its purpose is to provide a double-buffered interface between a byte parallel system on the one hand (MASTER) and a bit serial system on the other (SLAVE). Data flows only from master to slave.

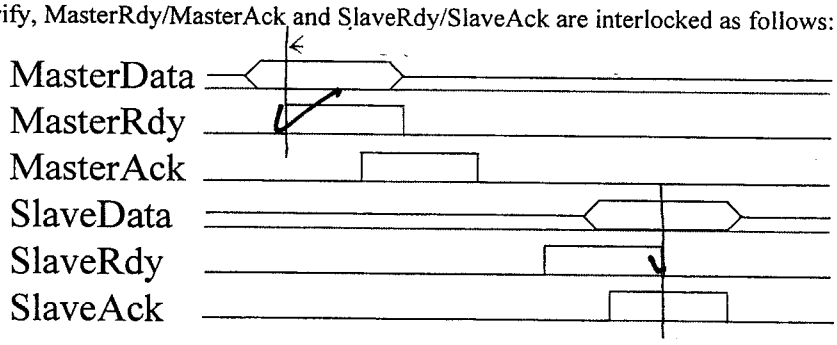


The subsystem consists of two 8-bit parallel-load, serial-shift registers named PING and PONG. It works as follows. The MASTER places data on the MasterData lines, and asserts MasterRdy. When the subsystem detects that MasterRdy is true, it latches the input data into one of PING or PONG, if one is available, then asserts MasterAck when the data is safely latched. To insure that the Master has seen the Ack and that both the Master and the subsystem have reset themselves in anticipation of another transaction, the Master first unasserts MasterRdy and the subsystem responds by unasserting MasterAck. If neither PING nor PONG are available for new data, the subsystem simply defers asserting MasterAck until one is available and the master data has been latched into it.

The slave side works in a similar manner. The SLAVE asserts SlaveRdy when it needs the next data bit. The subsystem has to keep track of which one of PING or PONG has it, and then shift it out onto SlaveData. When there is good data on the SlaveData line, the subsystem asserts SlaveAck. The SLAVE will indicate that it has latched the data by unasserting SlaveRdy. The subsystem responds by unasserting SlaveAck, and the process can start anew.

Note that the MASTER and SLAVE sides of the subsystem can operate independently as long as there is no need to access the same register from both sides.

To further clarify, MasterRdy/MasterAck and SlaveRdy/SlaveAck are interlocked as follows:



(a) To test your understanding of the signaling conventions, indicate on the above timing diagram the earliest time it is safe for the subsystem to latch MasterData and the earliest time it is safe for the subsystem to stop asserting SlaveData (2 points).

(b) Design a register-level datapath for the subsystem (i.e., you need only represent storage registers, shifters, counters, etc. their interconnections via busses, tri-states, and multiplexers, and their control signals like LD, SHFT, CNT; you do not have to provide detailed designs for the register, shifter, counter, etc. internals). Tabulate your control signals and briefly describe what they do. (6 points).

Datapath Diagram (3 points):

Briefly explain your scheme for interlocking access to PING and PONG while reading from the MASTER and writing to the SLAVE (1 point):

Tabulate your register-transfer operations and briefly describe their functions (2 points)

(c) Show a NEAT state diagram for the Master-side state machine as a MEALY MACHINE (6 points).

(d) Show a NEAT state diagram for the Slave-side state machine as a MEALY MACHINE (6 points).