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

EECS 150 D. E. Culler
Fall 2007

Problem Set #3: FSMs
Assigned 9/14/2007, Due 9/21/2007 at 2 PM

Problem 1. In this problem you are going to design a combinational logic system that maps from notes in a scale to tones in the octave. An octave has 12 tones, such as

C, C#, D, D#, E, F, F#, G, G#, A, A#, B and then starts again at C.

You can think of these as all the keys on the piano, starting at C.
A major scale includes seven of the notes in an octave (before returning to the next tonic) that have the intervals W,W,H,W,W,W,H where W (a whole step) is two tones (+2) and H (a half step) is one tone (+1), such as the C major scale

C, D, E, F, G, A, B and then starts again at C.

These would be the white keys on the piano, starting at C and skipping the black keys. We'll work with octaves and scale abstractly using Boolean values.

You circuit has three inputs, I_2, I_1, I_0, representing the notes in the scale with <0,0,0> being the root and <1,1,1> being the 7th. It has four outputs <O_3, O_2, O_1, O_0> representing the tone in the octave. It maps each note in the major scale to its corresponding tone in the octave.

1.a. Specify your circuit as a truth table with three input columns and four output columns.

I2I1I0	O3O2O1O0
000	0000
001	0010
010	0100
011	0101
100	0111
101	1001
110	1011

1.b. Produce a sum or products circuit and corresponding Boolean expression for each output.

O3 = I2I1I0' + I2I1'I0
O2 = I2'I1 + I2I1'I0'
O1 = I2I0' + I2'I1'I0
O0 = I2I0'+I2I1'+I2'I1I0

1.c. Optimize each expression to minimize the number of gates, using only NAND and NOR gates and give the corresponding Boolean expression.

O3 = I2I1 + I2I0
O2 = I2'I1 + I2I1'I0'
O1 = I2I0' + I2'I1'I0
O0 = I2+I2'I1I0

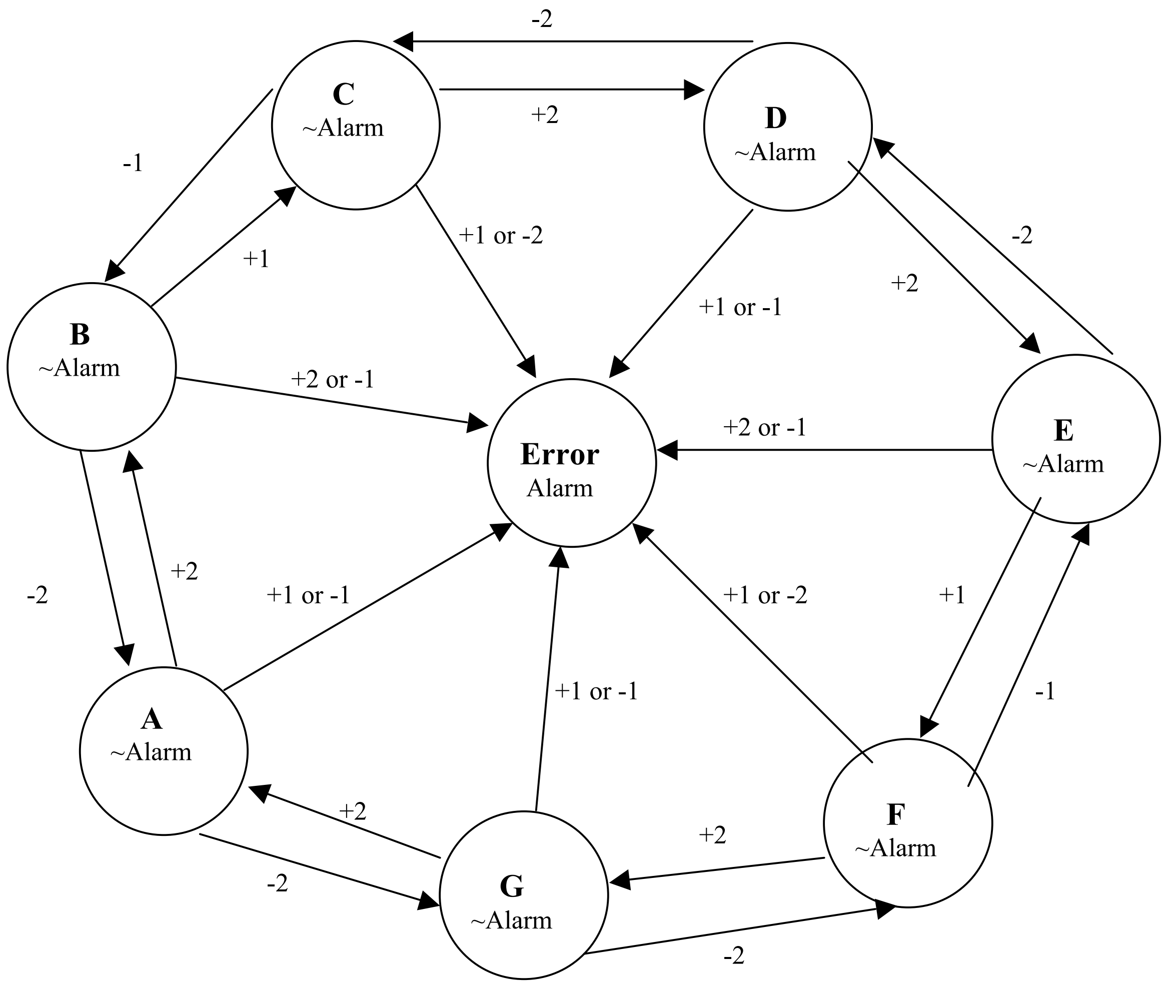
1.d. Give a behavioral module for your design in Verilog.

```
module ( clk, I, O);
  input clk;
  input[2:0] I;
  output[3:0] O;

  always @(posedge clk)
  begin
    case (I)
      3'b000: O <= 4'b0000;
      3'b001: O <= 4'b0010;
      3'b010: O <= 4'b0100;
      3'b011: O <= 4'b0101;
      3'b100: O <= 4'b0111;
      3'b101: O <= 4'b1001;
      3'b110: O <= 4'b1011;
    endcase
  end
endmodule
```

Problem 2. In this problem we are going to build a scale-watcher Moore Machine. An interval is the number of tones between two notes, so C-C# is +1, C-D is +2, and C-B is -1. Your FSM is given a melody that starts on the root and proceeds with a series of whole and half steps up and down, and repeated nodes (0 steps). Your FSM must raise an alarm if the melody ever leaves the major scale.

2.a. Describe or solution as a state transition diagram.



2.b. Give the symbolic state transition table for your solution.

Current State	NextState				
	-2	-1	0	1	2
C	Error	B	C	Error	D
D	C	Error	D	Error	E
E	D	Error	E	F	Error
F	Error	E	F	Error	G
G	F	Error	G	Error	A
A	G	Error	A	Error	B
B	A	Error	B	C	Error
Error	Error	Error	Error	Error	Error

2.c. Produce a (hopefully elegant) state encoding and show the state transition table.

Current State	NextState				
	-2=110	-1=111	0=000	1=001	2=010
C=0000	1111	1011	0000	1111	0010
D=0010	0000	1111	0010	1111	0100
E=0100	0010	1111	0100	0101	1111
F=0101	1111	0100	0101	1111	0111
G=0111	0101	1111	0111	1111	1001
A=1001	0111	1111	1001	1111	1011
B=1011	1001	1111	1011	0000	1111
Error=1111	1111	1111	1111	1111	1111

2.d. Draw a structural implementation of your circuit.

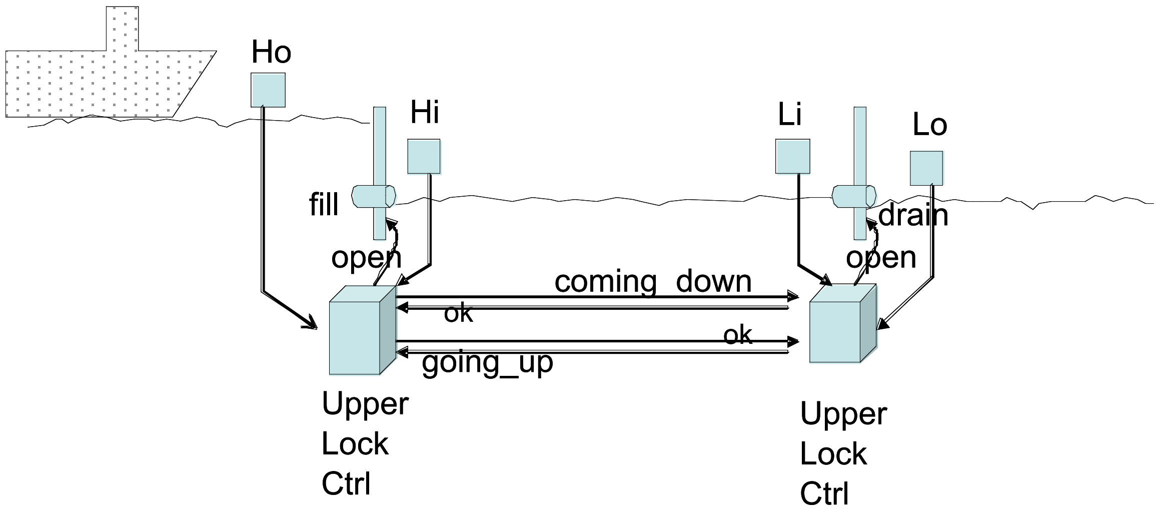
2.e. Give a behavioral Verilog implementation of your solution.

```
module ScaleWatcher(Alarm, clk, interval);
  input clk;
  input [2:0] interval;
  output alarm;
  reg [4:0] CurrentState, NextState;

  always @ (posedge clk)
    begin
    if(Nextstate == 4’b1111 || Nextstate == 4’b0001 || Nextstate == 4’b0011 || Nextstate
== 4’b0110|| Nextstate == 4’b1000 || Nextstate == 4’b1010 || Nextstate == 4’b1101)
      Alarm <= 1;
     CurrentState <= 4’b1111;
    else if (Nextstate == 4’b1100) begin
      Alarm <= 0;
      CurrentState < = 4’b0000;
    else begin
      Alarm <= 0;
      CurrentState<=NextState;
    end
  end

  always @ (CurrentState or interval)
    begin
    case(interval)
      3’b110: NextState = NextState -2;
      3’b111: NextState = NextState -1;
      3’b010: NextState = NextState +2;
      3’b001: NextState = NextState +1;
      3’b000: NextState = NextState +0;
    endcase
  end
endmodule
```

Problem 3. You have been asked to design a pair of controllers for the gates on a lock on the Berkeley-Stanford canal shown below. The upper lock has a High-outside ship detector (Ho) and a High-Inside (Hi) ship detector. It has two control outputs on the lock: fill and open/~open. And it has a "full" signal indicating that the lock is at the upper level. Similarly, the lower lock control has similar ship detectors and controls drain and open/~open. Each can signal the other when it is ready to start a ship moving through the lock and gets an OK acknowledgment from the other when it is OK to do so. Ships going down have priority.

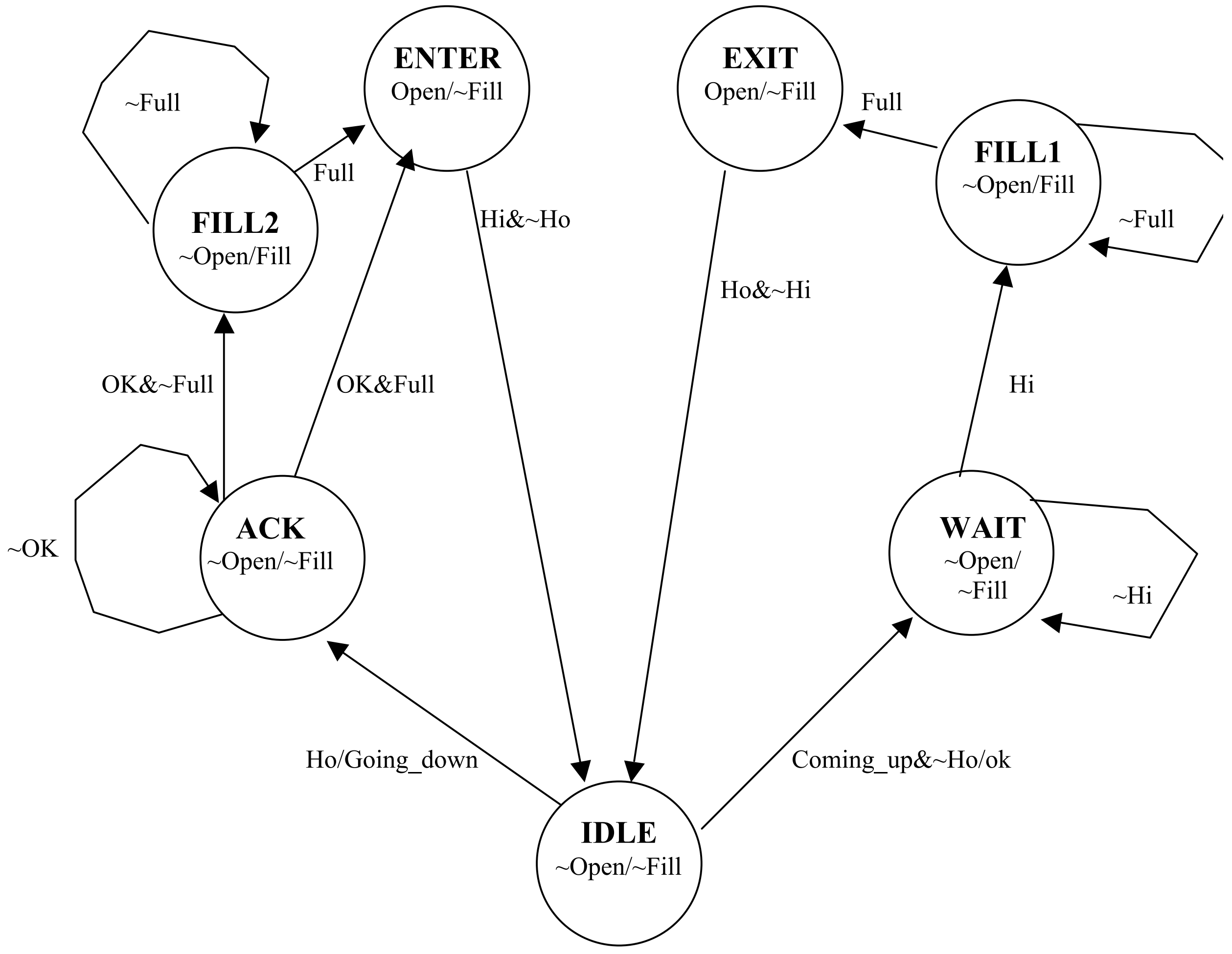


When a ship arrives at the upper lock (Ho) and no ship is in the lock going up, the upper controller requests permission to move it into the lock. The lower controller must ensure that no ship is going down and its gate is closed before giving its OK. Then the upper controller fills the canal and opens the gate. Once the ship has moved through, it closes the gate. When the ship reaches the lower lock, the lower controller drains the canal and opens the gate for the ship to pass out. Similarly, a ship going up requests the lower controller to gain permission, drain the canal, and let the ship pass through its gate.

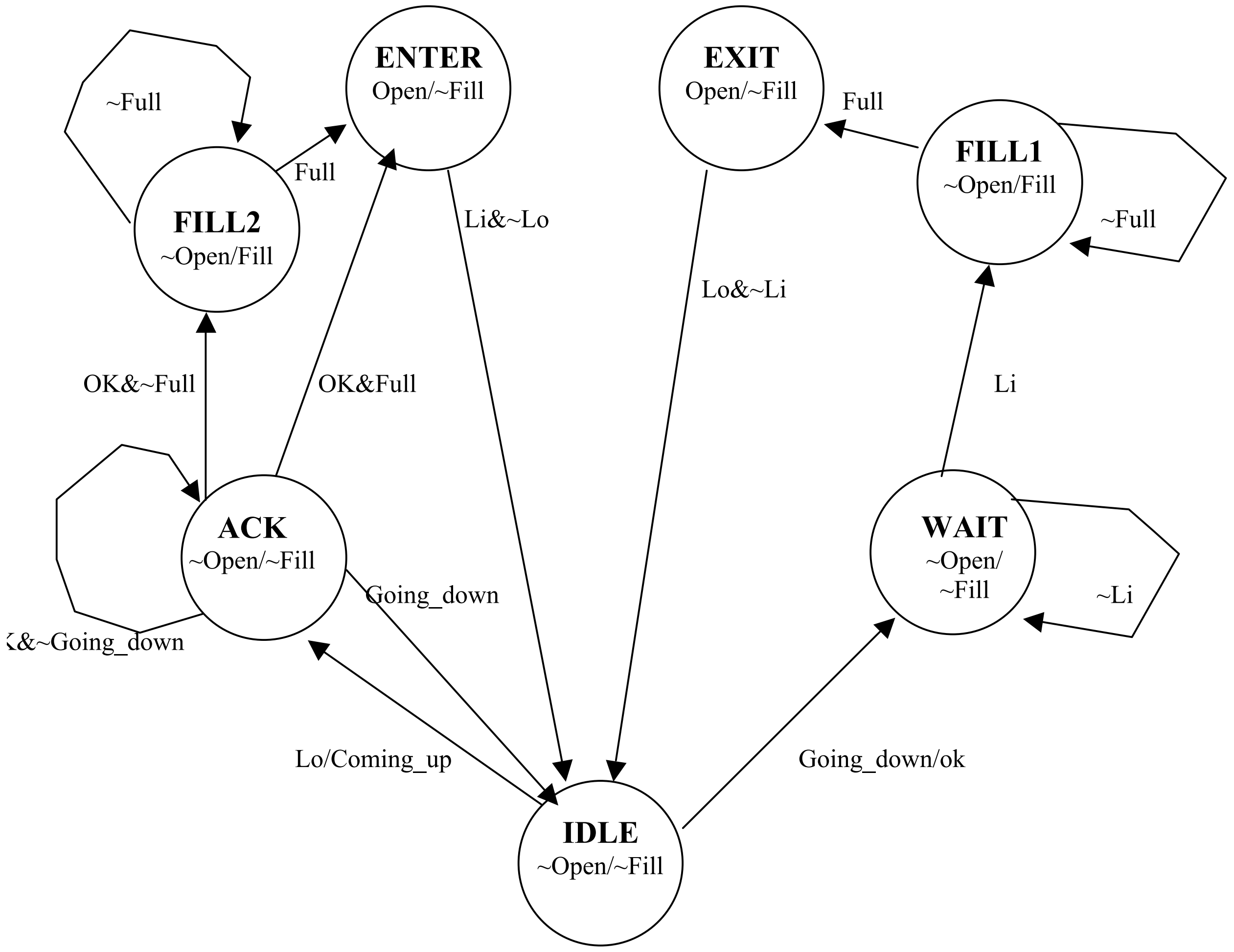
Design the two cooperating finite state machines for the two controllers and give a state transition diagram for each. If ships are present at both gates, the one going down should be permitted to go first.

There are many possible solutions. Please see example on the next page.

Upper Controller



Lower Controller


ENTER
Open/~Fill
EXIT
Open/~Fill
FILL2
~Open/Fill
FILL1
~Open/Fill
ACK
~Open/~Fill
WAIT
~Open/
~Fill
IDLE
~Open/~Fill
~Full
Full
Full
~Full
Li&~Lo
Lo&~Li
OK&~Full
OK&Full
Li
Going_down
K&~Going_down
~Li
Lo/Coming_up
Going_down/ok