BEE 271 Spring 2017
Homework 6 answers

Please answer the following questions. Each is worth 8 points.

1. Create a Verilog module that implements a Moore machine that can report the number of ones on an ***asynchronous*** serial input sampled on the last 128 clocks. Please minimize the critical path from clock to output by avoiding trying to “add up” the ones.

   To solve this, you need to split the problem into two parts, a shift register to capture the 128-bit window of input samples and a separate counter of how many of those samples happen to be ones just this moment, adjusting the number up or down as you shift ones in from the left or out from the right.

```
module MooreCountOnes( input clock, reset, w, output reg [ 7:0 ] count );
   reg [ 0:127 ] Q;
   always @( posedge reset, posedge clock )
      if ( reset )
         { Q, count } <= 0;
      else
         begin
         Q <= { w, Q[ 0:126 ] };
         count <= count + w - Q[ 127 ];
         end
endmodule
```

2. Create a Verilog module for a Mealy version of the same machine that can report the number of ones in the current and last 127 bits on a ***synchronous*** serial input. Please minimize the critical paths from input or clock to output.

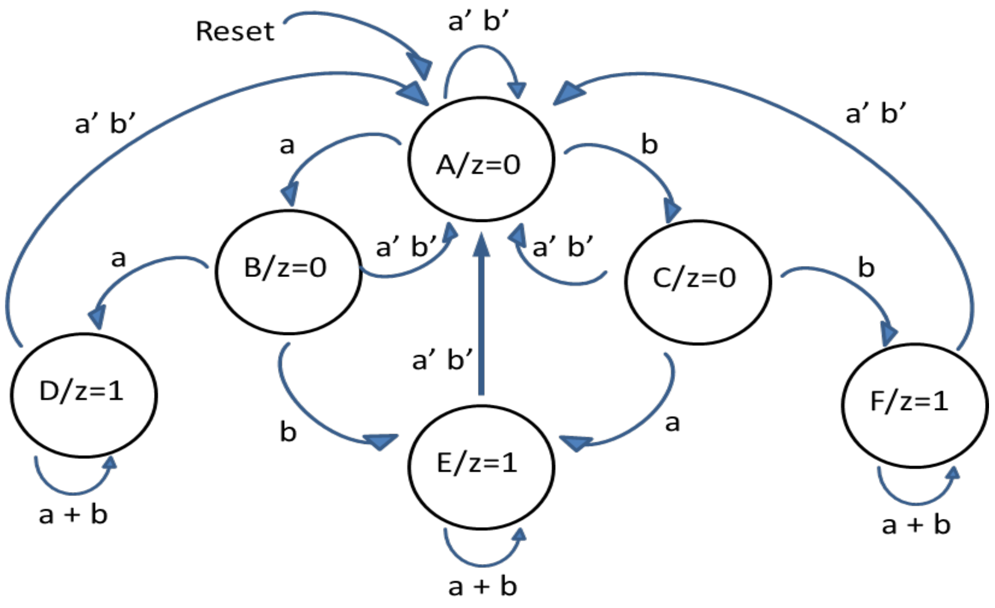
   The way we count how many ones are in the (shortened by 1 bit) shift register is unaffected but we have to add 1 to the output if the current input is a 1, adding additional skew from the clock.

```
module MealyCountOnes( input clock, reset, w, output [ 7:0 ] count );
   reg [ 0:126 ] Q;
   reg [ 7:0 ] srcount;
   assign count = srcount + w;

   always @ ( posedge reset, posedge clock )
      if ( reset )
         { Q, srcount } <= 0;
      else
         begin
         Q <= { w, Q[ 0:125 ] };
         srcount <= srcount + w - Q[ 126 ];
         end
endmodule
```

3. For two states to be equivalent, what has to be true about them?

For every possible input sequence, the same output sequence is produced regardless of which is the initial state.

4. The next several questions relate to this FSM. You may assume a clock and that the reset is synchronous and it's physically impossible for both a and b to be true at once. First, is this a Mealy or a Moore machine?



It's a Moore machine because the outputs do not depend on the inputs, only the state.

5. Create a Verilog module that implements this machine.

Here are a couple of variations, one using parameters, the other using an enum to refer to the states symbolically.

```
module OriginalFSM1( input clock, reset, a, b, output reg z );
   parameter A = 0, B = 1, C = 2, D = 3, E = 4, F = 5;
   reg [ 2:0 ] state;

   always @( * )
      case ( state )
         D, E, F: z = 1;
         default: z = 0;
      endcase

   always @( posedge clock )
      if ( reset )
         state <= A;
      else
         case ( state )
            A: state <= a ? B : b ? C : A;
            B: state <= a ? D : b ? E : A;
            C: state <= a ? E : b ? F : A;
            D: state <= a | b ? D : A;
            E: state <= a | b ? E : A;
            F: state <= a | b ? F : A;
         endcase
endmodule
```

```
module OriginalFSM2( input clock, reset, a, b, output reg z );
   enum { A, B, C, D, E, F } state;

   always @( * )
      case ( state )
         D, E, F: z = 1;
         default: z = 0;
      endcase

   always @( posedge clock )
      if ( reset )
         state <= A;
      else
         case ( state )
            A: state <= a ? B : b ? C : A;
            B: state <= a ? D : b ? E : A;
            C: state <= a ? E : b ? F : A;
            D: state <= a | b ? D : A;
            E: state <= a | b ? E : A;
            F: state <= a | b ? F : A;
         endcase
endmodule
```

6. Create a state table for this FSM.

	ab				
State	00	01	11	10	Z
A	A	C	–	B	0
B	A	E	–	D	0
C	A	F	–	E	0
D	A	D	D	D	1
E	A	E	E	E	1
F	A	F	F	F	1

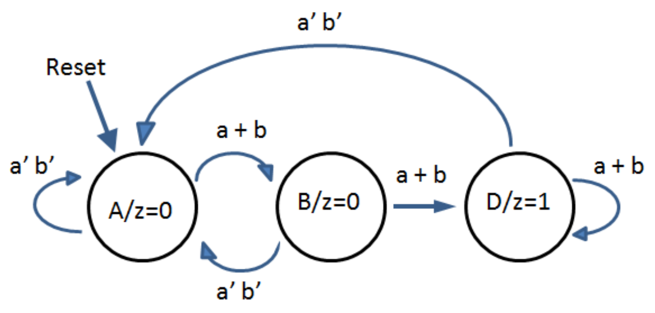
7. Find any equivalent states by partitioning, first by output, then by successors.

By output: (A B C) (D E F)
By 01-successor:
(A B C) 01-successors are (C E E) → split into (A) (B C)
(D E F) 01-successors are (D E F) → no splits
Final partitioning: (A) (B C) (D E F)

8. Create a minimized state table eliminating any equivalent states you discovered.

	ab				
State	00	01	11	10	Z
A	A	B	–	B	0
B	A	D	–	D	0
D	A	D	D	D	1

9. Create a state diagram for your minimized FSM.

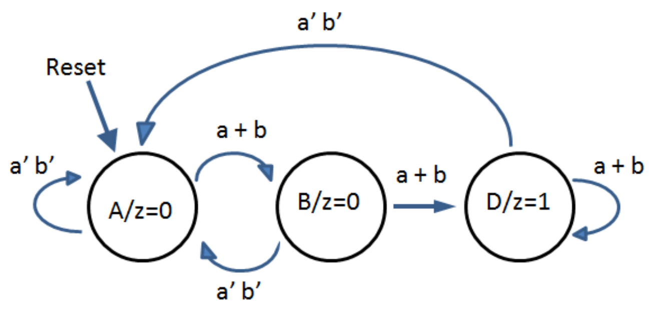


10. Create a Verilog module that implements your minimized FSM.

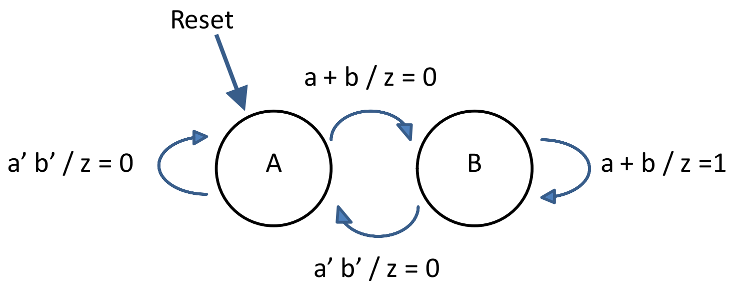
Again, here are a couple of variations, one using parameters, the other using an enum to refer to the states symbolically.

```
module MinimizedFSM1( input clock, reset, a, b, output z );
   parameter A = 0, B = 2, D = 3;
   reg [ 1:0 ] state;
   assign z = state == D;

   always @( posedge clock )
      if ( reset )
         state <= A;
      else
         casex ( { state, a, b } )
            'bxx_00: state <= A;
            'b0x_1x, 'b0x_x1: state <= B;
            'b1x_1x, 'b1x_x1: state <= D;
         endcase
endmodule


module MinimizedFSM2( input clock, reset, a, b, output z );
   enum { A, B, D } state;
   assign z = state == D;

   always @( posedge clock )
      casex ( { reset, a, b } )
         'b1xx, 'bx00: state <= A;
         'b01x, 'b0x1:
            case ( state )
               A: state <= B;
               default: state <= D;
            endcase
      endcase
endmodule
```

11. Create a state diagram for a Mealy version of this FSM.



Here is the Mealy version.



12. Create a Verilog module which implements your Mealy version.

```
module MealyFSM( input clock, reset, a, b, output z );
   reg state;
   parameter A = 0, B = 1;
   assign z = ( state == B ) & ( a | b );

   always @( posedge clock )
      casex ( { reset, a, b } )
         'b1xx, 'bx00: state <= A;
         default: state <= B;
      endcase
endmodule
```

13. Suppose you've been given a proposed, but clearly inefficient state diagram and asked to design the most elegant and efficient implementation you can. List three ways you might try to improve your design.

The first 3 or 4 are the ones I really wanted but I'll accept others.

1. Eliminate equivalent states.
2. Choose a more efficient state assignment.
3. Choose appropriate flip-flops.
4. Choose appropriate interfaces for modules.
5. Write elegant code (or something on that order.)