BEE 271 Final exam answer key, Spring 2017

Full name: ____

This exam is closed book, closed notes, no cellphones or computers and unproctored. Please try sit next to students you don't normally. You may take this exam to another room if you would be more comfortable. You have 2 hours to answer as many questions as you can. Some questions are easy and some are hard but each is worth the same 5 points, so pick and choose. Most correct answers can be fairly short. If you need more space, additional sheets may be stapled to your exam. I will be outside the room if you need to discuss something.

Please copy the following statement *in your own handwriting* and s*ign your name* below it:

On my honor, I will neither accept nor give unauthorized aid on this exam.

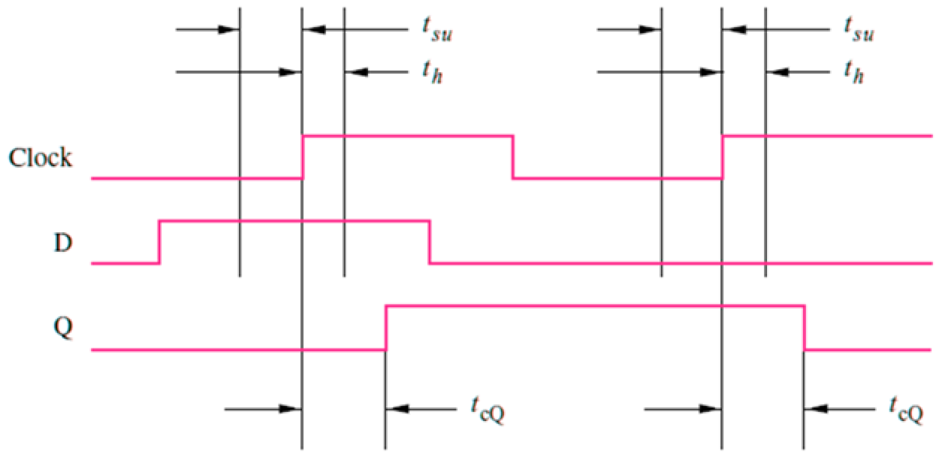
____

____

____
Signed

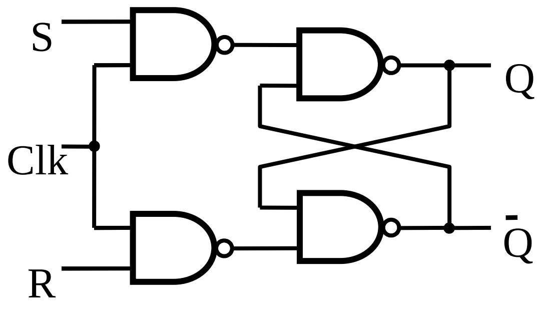
1. What are t_{su}, t_h and t_{cQ}? Draw a diagram showing how they're measured.

t_{su} is the setup time, the time the input must be valid before the active clock edge (shown here for a positive edge-triggered device). t_h is the hold time, the time the input must remain valid after the clock edge. t_{cQ} is the time from clock edge until the outputs of the flip flop become valid. We can assume $t_{cQ} > t_h$ else we'd have problems wiring the output of any flip flop directly to the input of any other.

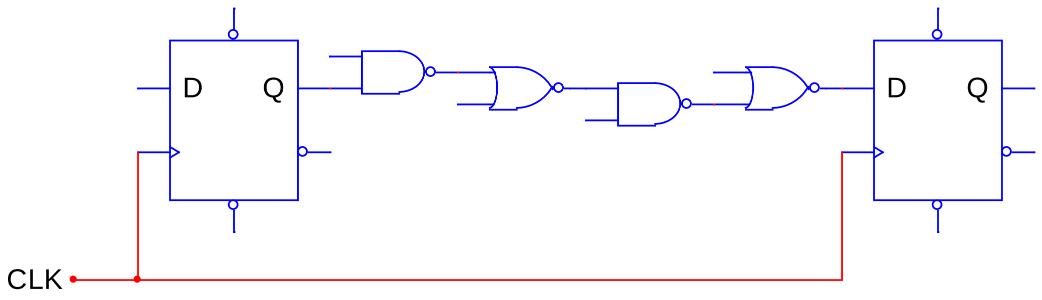


(b) Timing diagram

2. Draw a circuit for a gated latch using NAND gates.



3. Referring to the following circuit fragment, assume this represents the critical path in an actual circuit where the rest of the pins have sensible inputs and that t_{su} = 0.3 ns, t_h = 0.2 ns, 0.5 ns ≤ t_{cQ} ≤ 0.6 ns and t_{gate} = 0.4 ns. What is f_{max}?



T_{min} = t_{su} + t_{cQmax} + 4 * t_{gate} = 0.3 + 0.6 + 4*0.4 = 2.5 ns
f_{max} = 1 / T_{min} = 1 / 2.5e-09 = 0.4e09 = 400 MHz

4. What do the terms, edge-triggered and active edge mean?

Edge-triggered means the outputs of the flip-flop only change at a clock edge. The active edge is the positive or negative edge of the clock that triggers the change.

5. You are to write a Verilog module called Digit that implements one digit decimal counter (one that counts 0 to 9). It should have a synchronous reset and enable plus a special feature: It should tell you when it's about to rollover so that it can be used to build this 4-digit counter:

```
module FourDigits( input clock, reset, enable,
      output [ 3:0 ] bcd[ 3:0 ] );
   wire [ 5:0 ] ro;
   Digit d0( clock, reset, enable,  bcd[ 0 ], ro[ 0 ] );
   Digit d1( clock, reset, ro[ 0 ], bcd[ 1 ], ro[ 1 ] );
   Digit d2( clock, reset, ro[ 1 ], bcd[ 2 ], ro[ 2 ] );
   Digit d3( clock, reset, ro[ 2 ], bcd[ 3 ], ro[ 3 ] );
endmodule
```

Here's the interface to the module you should write. Please write the rest.

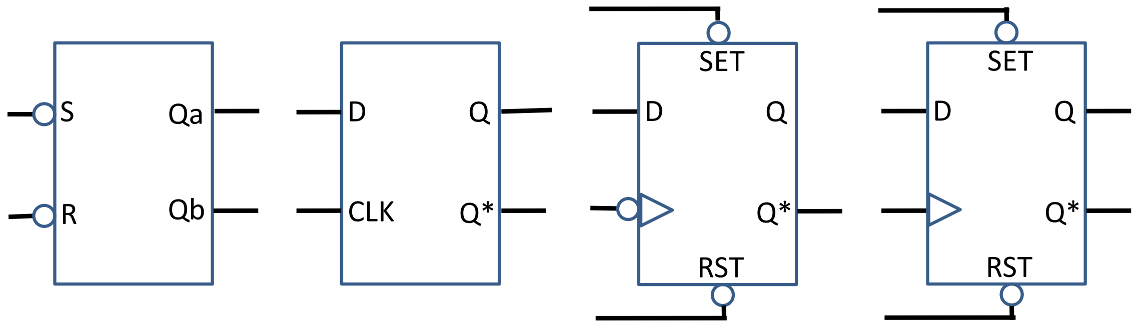
```
module Digit( input clock, reset, enable,
      output reg [ 3:0 ] digit,
      output rollover );

   parameter limit = 9;
   assign rollover = ( digit == limit ) & enable;
   always @( posedge clock )
      if ( reset | rollover )
         digit <= 0;
      else
         if ( enable )
            digit <= digit + 1;
endmodule
```

6. What is the difference between a latch, a gated latch, a master-slave flip-flop and an edge-triggered flip-flop? Draw schematic symbols for each.

A latch has only set or reset inputs. A gated latch adds a clock input and will respond to the set or reset inputs (or a D input) anytime the clock is high. Because there's no isolation between inputs and outputs latches cannot be used in counters and other circuits where feedback is needed.

Master-slave and edge-triggered flip-flops isolate the outputs from the inputs, changing the outputs at a clock edge based on the input values just before the active edge of the clock. The master-slave uses 8 gates in two latch stages, clocking data into the first stage when the clock is high, then into the second when the clock goes low, making it a negative edge-triggered device. A true edge-triggered flip-flop uses 6 gates and has a critical path to disable the input on the rising edge of the clock.



Latch Gated Latch Master-slave Edge-triggered

7. Design a 3-bit one-hot counter in Verilog with an asynchronous reset.

```
module ThreeBitOneHot( input clock, reset, output reg [ 0:2 ] count );
   always @( posedge reset, posedge clock )
      if ( reset )
         count <= 3'b100;
      else
         count <= { count[ 2 ], count[ 0:1 ] };
endmodule
```

8. Write a Verilog module that implements a D flip-flop with a synchronous reset.

```
module Dflipflop( input clock, reset, D, output reg Q );
   always @( posedge clock )
      Q <= reset ? 0 : D;
endmodule
```

9. Write a Verilog module that implements a JK flip-flop with an asynchronous reset.

```
module JK( input clock, J, K, reset,
      output reg Q );
   always @( posedge reset, posedge clock )
      casex ( { reset, J, K } )
         'b1xx: Q <= 0;
         'b000: Q <= Q;
         'b001: Q <= 0;
         'b010: Q <= 1;
         'b011: Q <= ~Q;
      endcase
endmodule
```

10. What does FSM stand for? Why are sequential circuits called FSMs? (What is the significance of the "FS" part?)

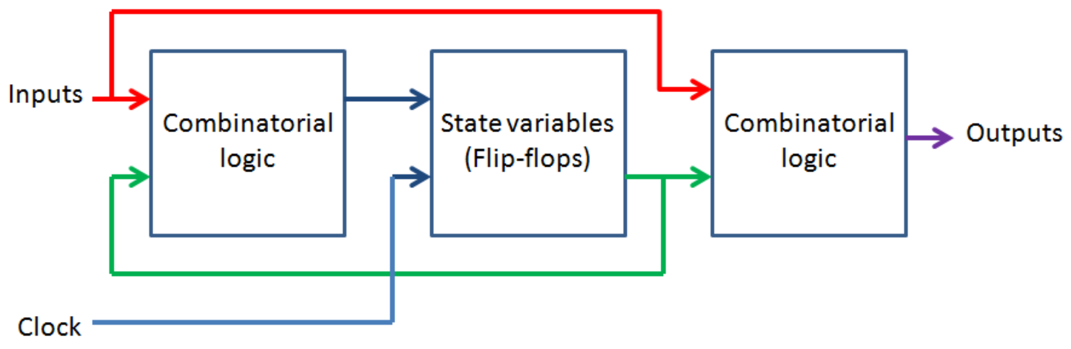
FSM = Finite state machine. The FS part refers to the fact that the circuit has only a finite number of states.

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

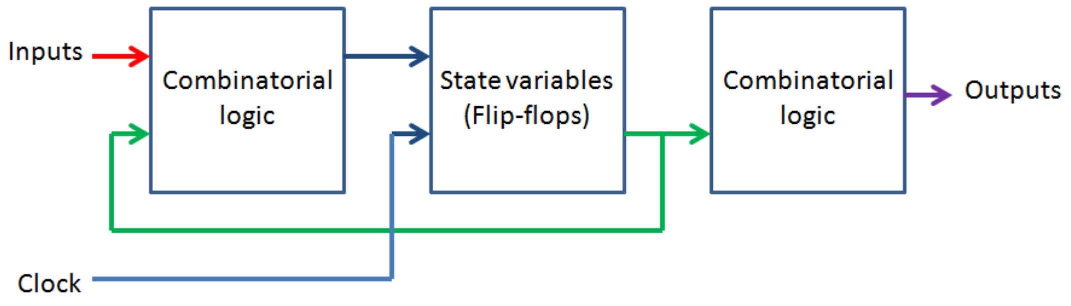
For every possible input sequence, the same output sequence is produced regardless of which is the initial state.

12. What is the difference between a Mealy and a Moore design? Draw a picture of each.

In a Moore design, the outputs depend only on the state variables. In a Mealy design, they also depend on the inputs. This is a Mealy machine.



And this is a Moore machine. ("Moore is less.")



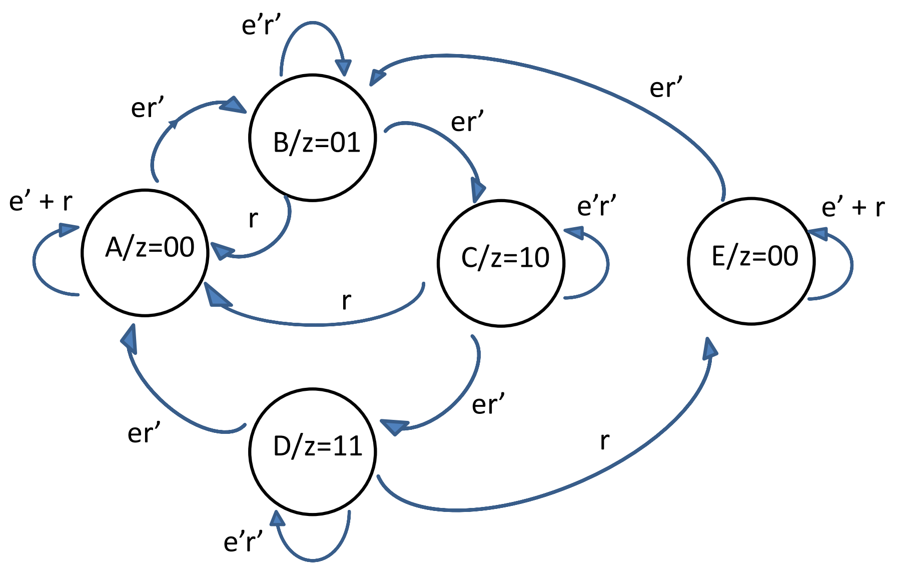
13. How would you decide whether to use a Moore versus a Mealy design? Which one typically produces outputs one clock sooner?

In designing synchronous circuits, we can pretty much take for granted that we will always want synchronous outputs. So the choice between Moore and Mealy is usually decided based solely on whether the inputs are synchronous, changing only with the clock, or asynchronous, changing at other times.

If the inputs are synchronous, we can use a Mealy machine whose outputs depend on both the state variables and the current input. If the inputs are asynchronous, we would probably use a Moore machines whose outputs only depend on the state.

The Mealy machine typically produces outputs one clock sooner.

14. The next several questions relate to this FSM. You may assume a clock. First, is this a Mealy or a Moore machine?



This is a Moore machine because the output, z, only depends on the state.

15. Create a state table for this FSM.

	Next state				
State	er = 00	er = 01	er =11	er =10	Output (z)
A	A	A	A	B	00
B	B	A	A	C	01
C	C	A	A	D	10
D	D	E	E	A	11
E	E	E	E	B	00

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

First, by output:
{ A, E } { B } { C } { D }

Only { A, E } remains that might be split by successors.

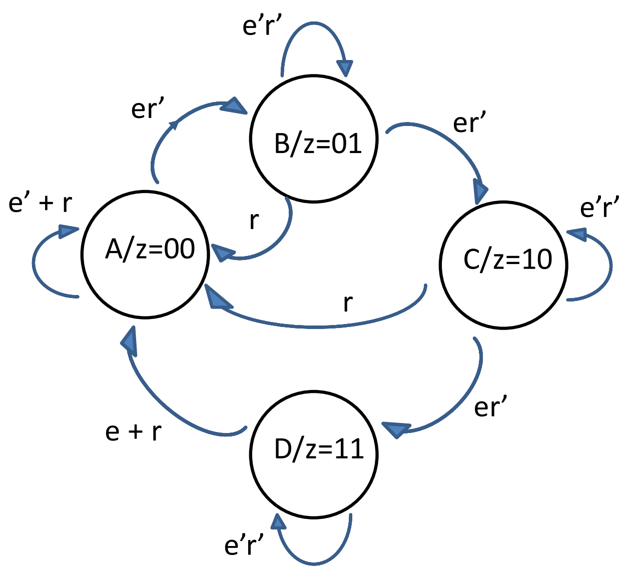
By 00 successor:	{ A, E }
By 01 successor:	{ A, E }
By 10 successor:	{ B, B }
By 11 successor:	{ A, E }

A and E are equivalent.

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

	Next state				
State	er = 00	er = 01	er =11	er =10	Output (z)
A	A	A	A	B	00
B	B	A	A	C	01
C	C	A	A	D	10
D	D	A	A	A	11

18. Create a state diagram for your minimized FSM or mark your changes on the original diagram.



19. Write a Verilog module that implements your minimized FSM.

Here are some examples. Notice the last one: it’s just a 2-bit binary counter with enable and reset inputs.

```
module FsmA( input clock, e, r, output reg [ 1:0 ] z );
   parameter A = 2'b00, B = 2'b01, C = 2'b10, D = 2'b11;
   always @( posedge clock )
      case ( z )
         A: z <= ~e | r ? A : B;
         B: z <= r ? A : e ? C : B;
         C: z <= r ? A : e ? D : C;
         D: z <= r ? A : e ? A : D;
      endcase
endmodule

module FsmB( input clock, e, r, output reg [ 1:0 ] z );
   parameter A = 2'b00, B = 2'b01, C = 2'b10, D = 2'b11;
   always @( posedge clock )
      if ( r )
         z <= A;
      else
         case ( z )
            A: z <= e ? B : A;
            B: z <= e ? C : B;
            C: z <= e ? D : C;
            D: z <= e ? A : D;
         endcase
endmodule

module FsmC( input clock, e, r, output reg [ 1:0 ] z );
   parameter A = 2'b00, B = 2'b01, C = 2'b10, D = 2'b11;
   always @( posedge clock )
      if ( r )
         z <= A;
      else
         if ( e )
            case ( z )
               A: z <= B;
               B: z <= C;
               C: z <= D;
               D: z <= A;
            endcase
endmodule
```

```
module FsmD( input clock, e, r, output reg [ 1:0 ] z );
   parameter A = 2'b00, B = 2'b01, C = 2'b10, D = 2'b11;
   always @( posedge clock )
      if ( r )
         z <= A;
      else
         if ( e )
            z <= z + 1;
endmodule

module FsmE( input clock, e, r, output reg [ 1:0 ] z );
   always @( posedge clock )
      z <= r ? 0 : e ? z + 1 : z;
endmodule
```

20. Create a 3-bit counter in Verilog that cycles through this sequence, 6, 2, 4, 5, 0, 7, 3, 1, with a synchronous reset to 4.

```
module WeirdCount( input clock, reset, output reg [ 2:0 ] Q );
   // 6, 2, 4, 5, 0, 7, 3, 1. Synchronous reset to 4.
   always @( posedge clock )
      if ( reset )
         Q <= 4;
      else
         case( Q )
            6: Q <= 2;
            2: Q <= 4;
            4: Q <= 5;
            5: Q <= 0;
            0: Q <= 7;
            7: Q <= 3;
            3: Q <= 1;
            1: Q <= 6;
         endcase
endmodule
```