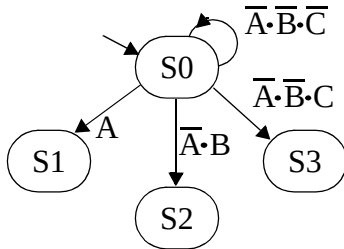
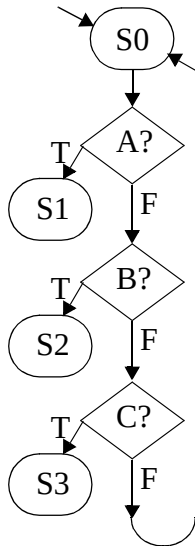


Verilog_Exam_Questions_Solutions_only.pdf

2 (12 + 10 + 24 = 46 points) 30 min.

State diagram coding in Verilog (you may refer to the Cadence (Esperan) Verilog guide):

Consider the following partial flowchart and the corresponding partial state diagram along with the Verilog code segments written by four students.



S0: **#1**
 begin
 if (A)
 state <= S1;
 else
 if (B)
 state <= S2;
 else
 if (C)
 state <= S3;
 else
 state <= S0;
 end
 Nested "if" => Preferred coding method!

S0: **#2**
 begin
 if (!A && !B && !C)
 state <= S0;
 else
 if (!A && !B && C)
 state <= S3;
 else
 if (!A && B)
 state <= S2;
 else // there is no if A
 state <= S1;
 end
 Nested "if" => But unnatural order. Full conditions. Less readable : (

S0: **#3**
 begin
 if (A)
 state <= S1;
 else
 if (!A && B)
 state <= S2;
 else
 if (!A && !B && C)
 state <= S3;
 else
 if (!A && !B && !C)
 state <= S0;
 end
 Nested "if" => But unnecessarily verbose by writing full conditions.

S0: **#4** Separate "if" statements => However, since full conditions were specified, we are not taking advantage of later assignments overriding earlier assignments.
 begin
 if (A)
 state <= S1;

 if (!A && B)
 state <= S2;

 if (!A && !B && C)
 state <= S3;

 if (!A && !B && !C)
 state <= S0;
 end

S0: **#5**
 begin
 if (C) // write A or B or C
 state <= S3;

 if (B) // write A or B or C
 state <= S2;

 if (A) // write A or B or C
 state <= S1;
 end

		B			
		BC			
A	0	S0	S3	S2	S2
	1	S1	S1	S1	S1
		C			

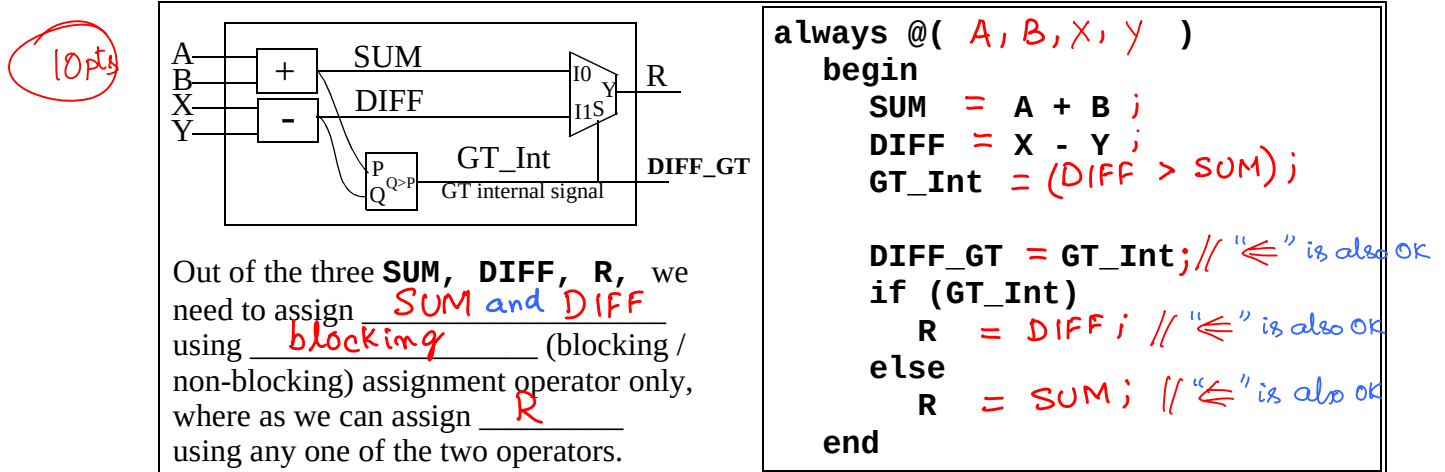
2.1 Notice that code #3 is similar to code #1, except that code #3 is perhaps unnecessarily (but harmlessly) more verbose (like my wife! don't tell her). Code #4 is formed by removing the three occurrences of "else" in code #3. Code #2 is essentially the reverse ordering of code #3. Write "Right" or "Wrong" below for each.
 Code #1 Right ; Code #2 Right ; Code #3 Right ; Code #4 Right ;

2.2 Now consider the incomplete Code #5 on the side along with the Karnaugh map representation of the desired state transitions. If all the three, A, B, C, are true, state gets assigned with S3, gets reassigned with S2 and further reassigned with S1. Since the last assignment prevails over the prior assignments, in this case, state finally goes to S1. Note that there is no if clause leading back to S0. Complete the "if" conditions in code #5. Else state reasons why it can not be completed.

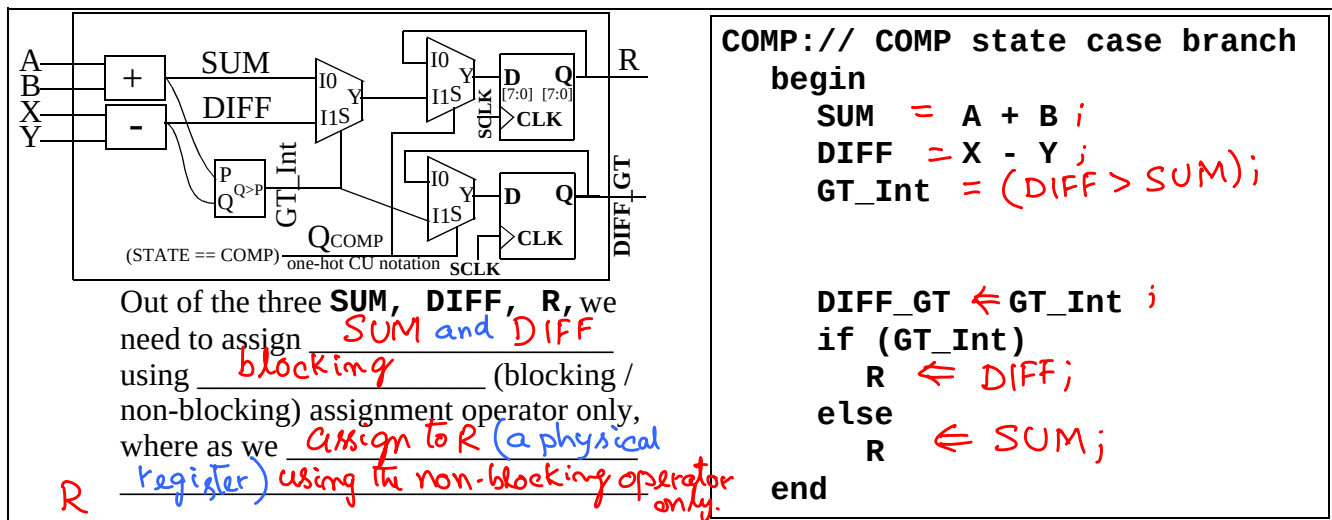
Later please go through the Fall 2018 Midterm Q#1 dealing with the coding of Inches to Feet to Yards to Furlongs to Miles. This makes us understand and appreciate how we used the fact that the later assignments override earlier assignments (to the same variables).

2.3 Combinational logic coding:

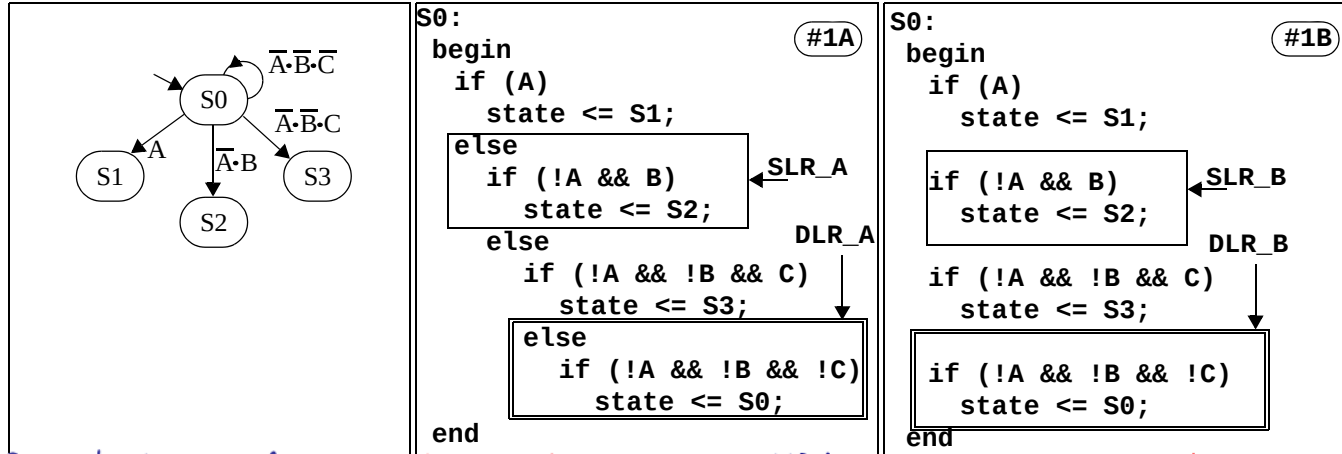
The result **R** is either the sum **SUM** of **A** and **B** (**A+B**) or the difference **DIFF** **X** minus **Y** (**X-Y**), depending on which ever is greater. Assuming that all needed declarations are already made appropriately (as reg or wire), complete the always block below.



Sequential logic coding: The above design is modified to have registered outputs. These registers shall be updated only in COMP state. Complete the RTL (datapath operations) in the COMP state (COMP case branch). The case statement is in an always block(always @ (posedge CLK)).



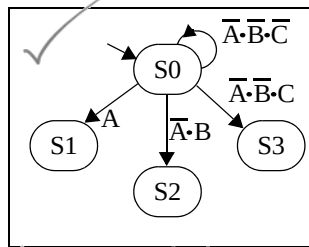
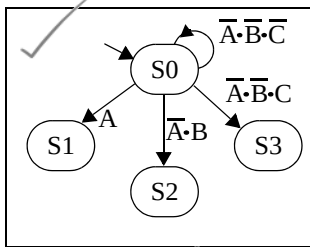
- 3.3** State diagram coding in Verilog: The following is an extract from our Fall2010 quiz exam. And we know that both code snippets (#1A and #1B) (which are parts of their respective clocked always procedural blocks) correctly represent the partial state diagram on the left.



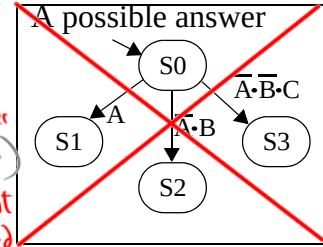
Important point learnt: Coding a state machine in HDL using a case statement will never cause violation of M.E. or A.I rules as HDL can't assign two values and there is always an implicit assignment to itself. If we did not cover all scenarios, the left out scenarios, the state does not change. So removing SLR_A and SLR_B did not cause A.I violation.

4 pts 3.3.1 A later procedural assignment (concurrent assign statement / procedural assignment) overrides an earlier such assignment in a either (either clocked or combinational procedural block) (clocked procedural block / combinational procedural block / either / neither).

8 pts 3.3.2 What happens if we remove the lines in the two double-line rectangles, **DLR_A** and **DLR_B**? A possible answer (removing the loop around S0) is shown on the right. Do you agree with the possible answer? Yes / No. If you disagree, show the correct answer by modifying one or two of the left-side diagrams. Did we violate the A.I or M.E. requirements? Neither (A.I. / M.E. / Both / Neither).



No change to the state diagram is needed, as a register (like the state memory) will retain its content unless it is reassigned.



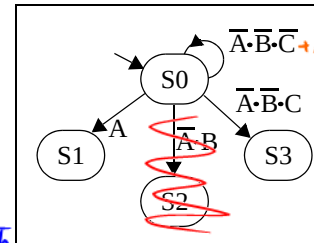
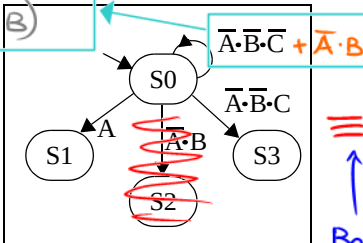
8 pts 3.3.3 What happens if we remove the lines in the two single-line rectangles **SLR_A** and **SLR_B**. A possible answer is shown on the right.

Do you agree with the possible answer? Yes / No.

If you disagree, show the correct answer by modifying one or two of the left-side diagrams.

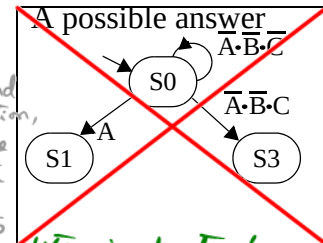
Did we violate the A.I or M.E. requirements? Neither (A.I. / M.E. / Both / Neither).

can be simplified to $\bar{A} \cdot (\bar{C} + B)$



Both Same

While surely the S2 state gets removed, we need to understand that, under $\bar{A}B$ condition, since no new state value is loaded into the state memory, the current state S0 continues to prevail. So the $\bar{A}B$ condition is effectively transferred to the state S0 as shown on the side.



6 pts 3.4 Synchronous counter coding: As shown on the side, many technicians code combinational logic separately and use a clocked always block just to update the registers, whereas engineers code the register together with the upstream combinational logic in one clocked always block.

The 2 lines in EngBox:

OK to leave them commented? Y/N
OK to uncomment them? Y/N

The 2 lines in TechBox:

OK to leave them as is? Y/N
OK to comment them out? Y/N

Engineer's way

```
always @(posedge clk)
begin : COUNT_BLOCK
  if (clr)
    Q <= 3'b000;
  else if (load)
    Q <= BA;
  else if (en)
    Q <= Q + 1;
  // else
  // Q <= Q;
end
```

EngBox

Technician's way

```
always @(posedge clk)
begin
  Q <= Q_next;
end

always @(*)
begin
  if (clr)
    Q_next = 3'b000;
  else if (en)
    Q_next = Q + 1;
  else if (load)
    Q_next = BA;
  else
    Q_next = Q;
end
```

TechBox

In combinational logic coding, we need to make sure that a value is assigned to the output in every pass through the code. Otherwise latches will be inferred.

In some of the RTL designs, we find the use of the following HDL rule produces a convenient understandable design.
Rule: a later assignment overrides an earlier assignment in a procedural block.
 Notice this in the Miles/Furlongs problem in the sequence of problems below.

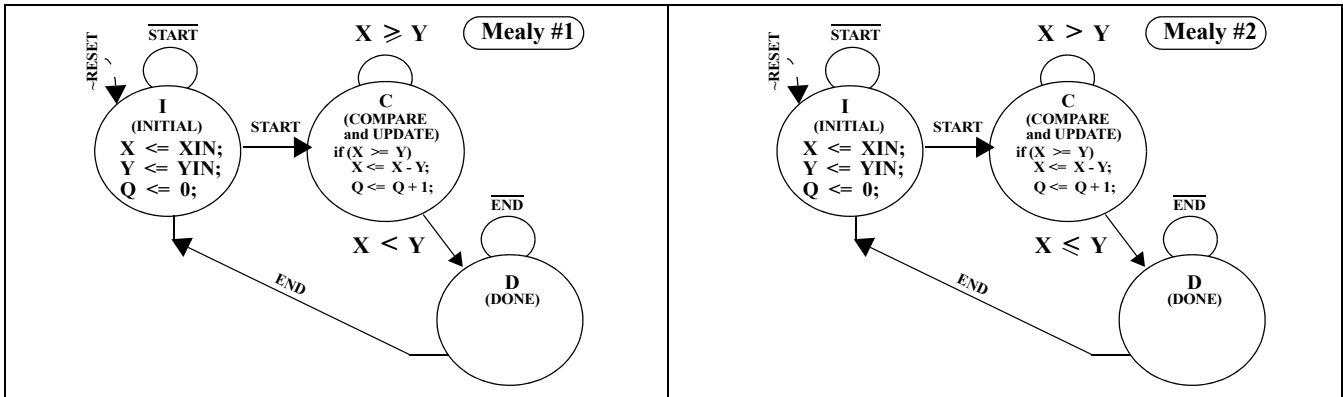
2 (10+20+20+20+30 = 100 points) 60 min. Inches => Feet => Yards conversion

2.1 Given below are two designs of a divider, doing division by repetitive subtraction.

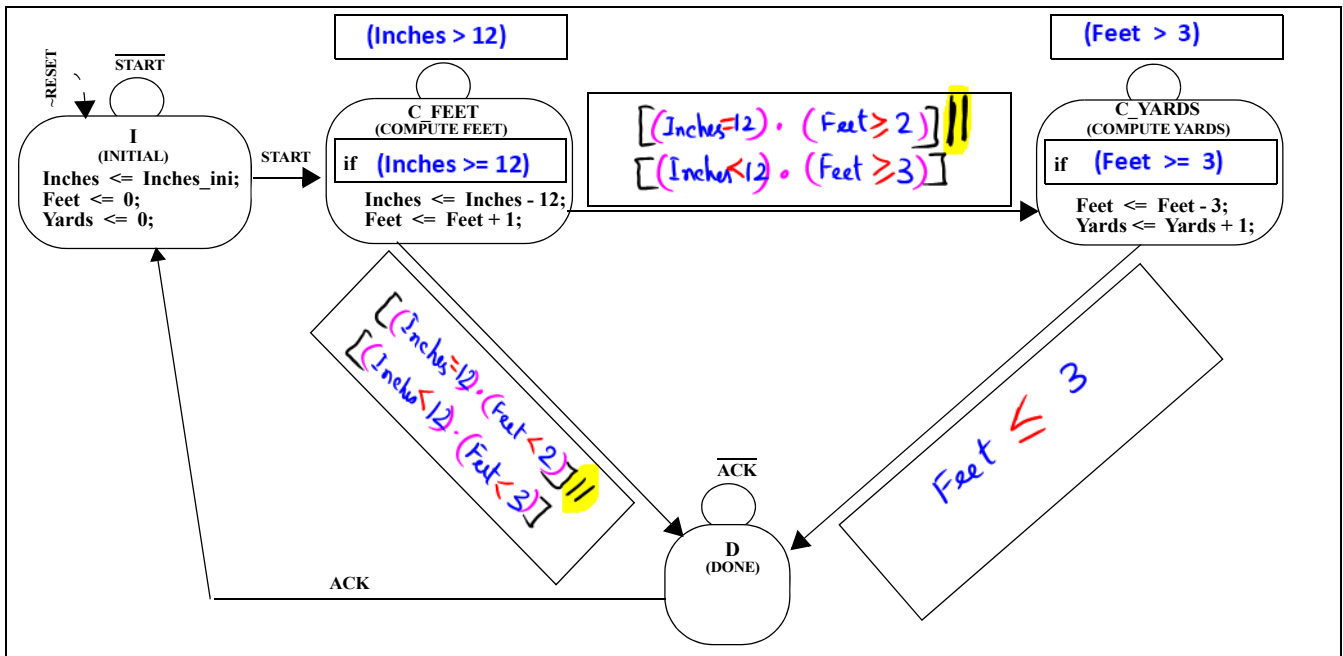
Mealy #1 design produces right (right / wrong) results.

Mealy #2 design produces right (right / wrong) results.

(Mealy #1 / Mealy #2) is less efficient than (Mealy #1 / Mealy #2) because it takes one more (one or several more) clock(s) (always / sometimes). Give an example data and highlight the difference between them. 10 divided by 5 takes 3 clocks in C state in Mealy #1, where as it takes only 2 clocks in Mealy #2 in C state. In all evenly divisible by divider cases, Mealy #1 wastes one clock by going back to the C state when (X == Y).



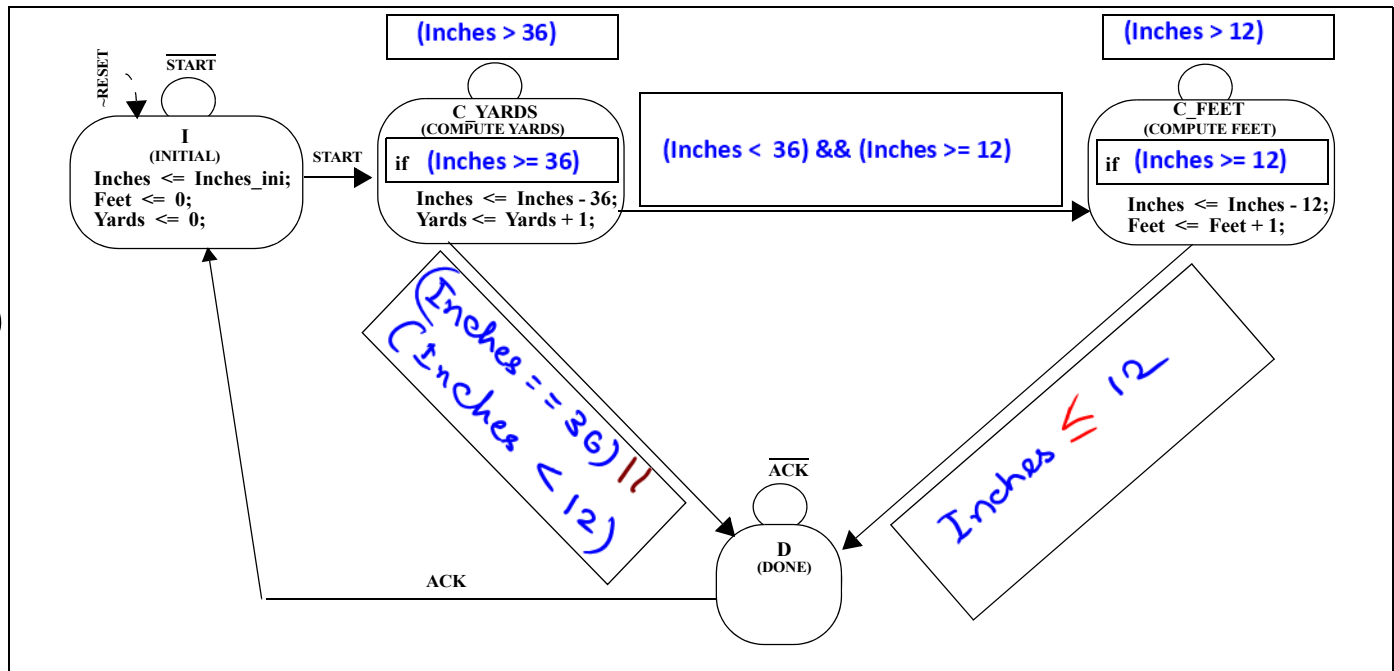
2.2 Using the ~~above~~ efficient Mealy division method above, complete the following inches => feet => yards converter below.



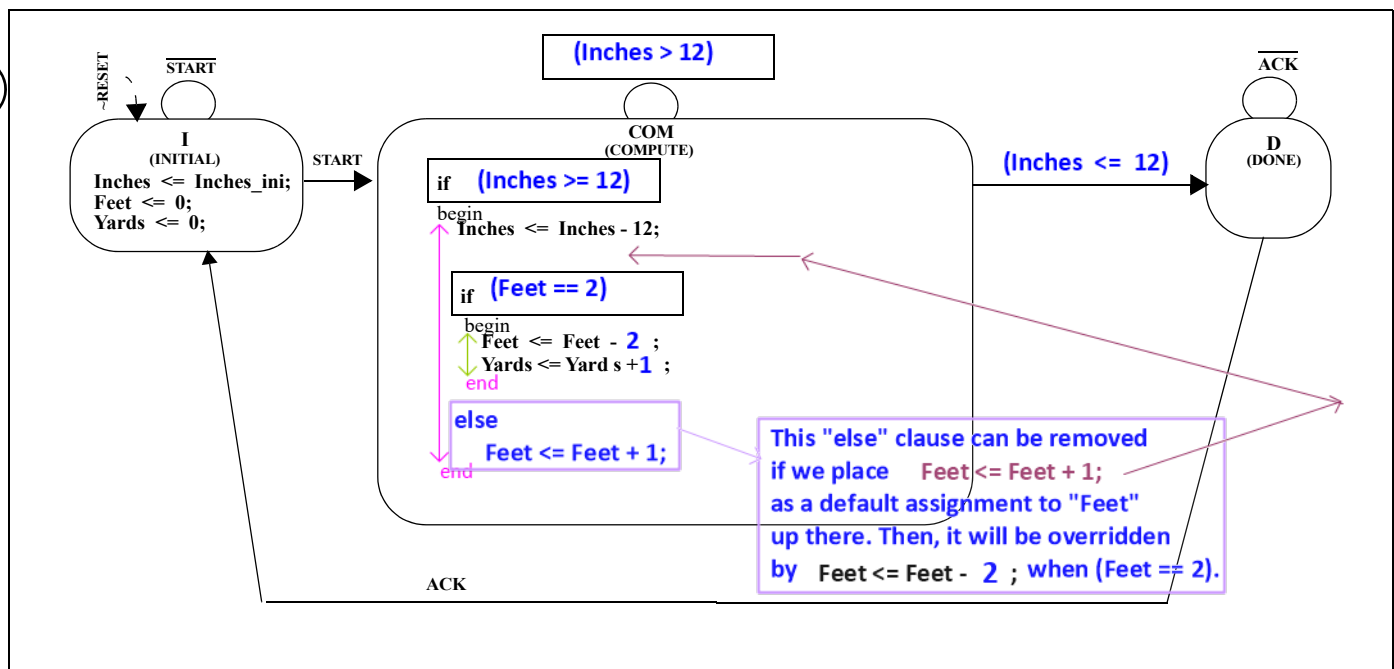
2.3 Do this after you do Q#2.4 and Q#2.5 on the next page. Fill-in the clocks spent in the table below.

	Clocks taken in the design of Q#2.2		Clocks taken in the design of Q#2.4		Clocks taken in the design of Q#2.5
	C_FEET	C_YARDS	C_YARDS	C_FEET	COM
60 inches = 1 Yard 2 feet 0 inches	5	2	2	2	5
61 inches = 1 Yard 2 feet 1 inches	6	2	2	3	6
72 inches = 2 Yard 0 feet 0 inches	6	2	2	0	6
84 inches = 2 Yard 1 foot 0 inches	7	3	3	1	7

2.4 Redesign the state machine of Q#2.2, this time converting inches to yards first and then the leftover inches to feet next



2.5 Our principal designer, Mr. Trojan, observes that the design of Q#2.2 can be improved by merging the C_FEET and CYARDS states into one combined state, called, say, COM (for compute), in which we convert inches to feet (as before in the C_FEET state) and also, whenever enough feet (3 feet) gathered or about to gather (i.e. we have 2 feet and we see that another foot is coming, because currently inches are 12 or more), we convert the feet into an Yard. This way, to convert feet to yards, we incur zero (fewer) extra clocks (extra = over and above the clocks needed to convert inches to feet). Complete the state diagram.



2.5.1 We can apply the above idea to our design in Q#1.1 and save clocks in checking divisibility by 3. T / F

SOLUTION

Fall 2018

EE354L

Instructor: Gandhi Puvvada

Midterm Exam (20% to 25%)

Date: Oct 29, 2018, Friday

Closed-Book Closed-Notes Exam

Time: 9:00-11:50 AM EEB132

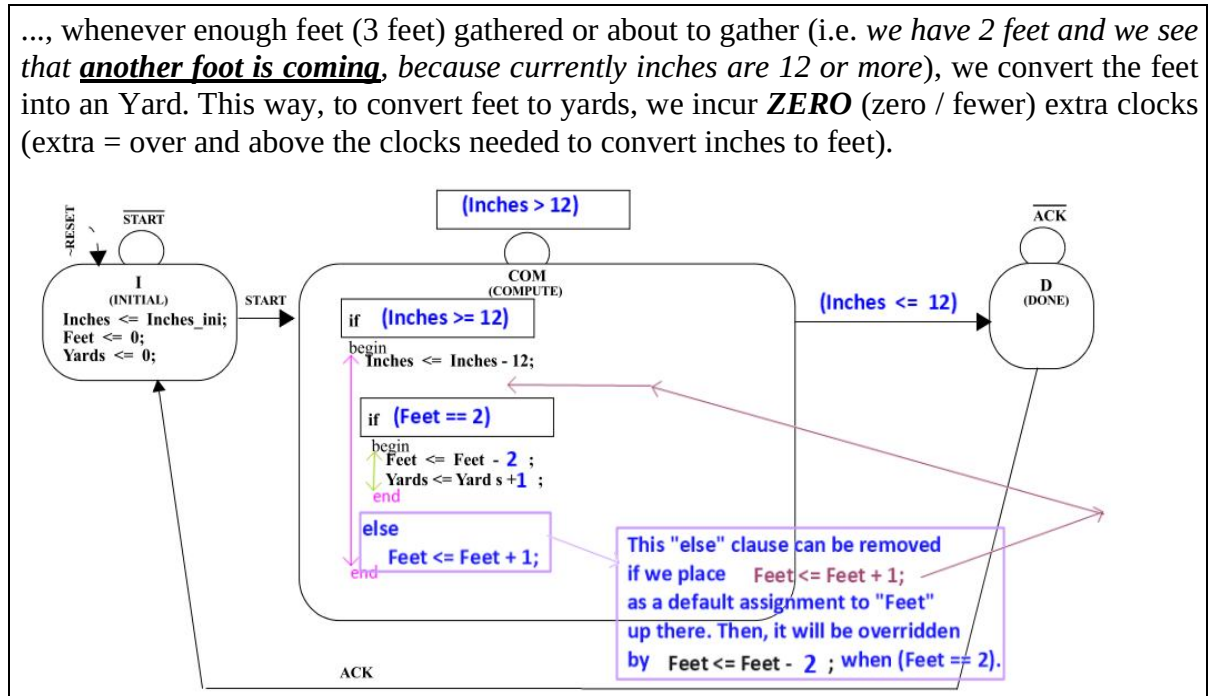
Name:

Total points: 250

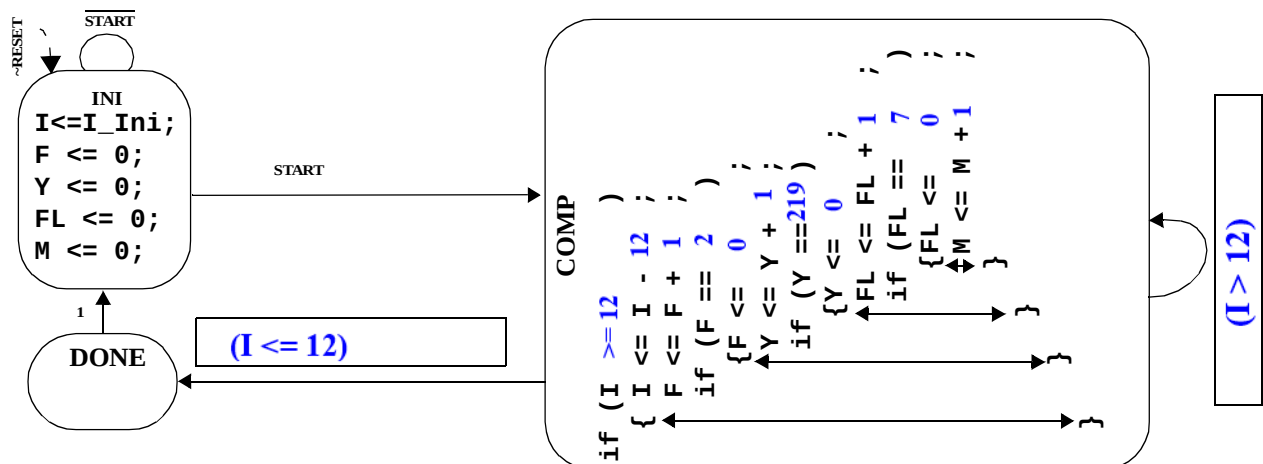
Perfect score: 245 / 250

1 (40 points) 25 min. State Machine design

Given below is solution to Q#2.5 of the midterm of Spring 2018.



Following the above example, let us convert Inches (I) $\Rightarrow$ Feet (F) $\Rightarrow$ Yards (Y) $\Rightarrow$ Furlongs (FL) $\Rightarrow$ Miles (M). One Mile = 8 Furlongs (used in UK and in India, but not so much in the US). One Furlong = 220 Yards. Note that it is convenient to use the above suggestion in the small rectangle regarding taking advantage of later assignments overriding earlier assignment in HDL coding. Try to reduce cost and improve speed by avoiding unnecessary subtractions such as (Feet - 2) under the condition "if (Feet == 2)". You can replace the statement "Feet $\leq$ Feet - 2;" with "Feet $\leq$ 0;". Do not worry about the sizes (bit-widths) of the various registers. Complete the following state diagram.



One engineer (we do not know if he is a Bruin or a Trojan) says the ~~log~~^{long} nested "if" statement means a very slow clock as Miles incrementation takes several additions or subtractions or comparisons to finish before we decide whether to increment miles or not. Derive the condition for Miles incrementation and offer your analysis on the long nested "if" statement. _____

Several comparison conditions are evaluated in parallel, hence we do not expect to see significant impact on clock speed.

Conditions for miles to be incremented:

$(I \geq 12) \ \& \ (F=2) \ \& \ (Y=219) \ \& \ (FL = 7)$

There are no subtractions or additions that the miles incrementation waits on :)

It takes less clocks if we need to convert inches into Yards only instead of up to miles. T / F
 Explain: _____

The clocks spent in COMP states are the clocks it takes to reduce I (Inches) by repetitively subtracting 12. The rest of the items (Feet, Yards, Furlongs, and Miles) increment at appropriate times in the background without taking extra clocks.

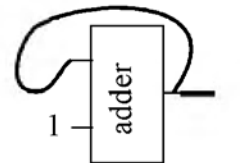
5 (10 points) 10 min.

Combinational feedback (example shown on the right) is

1 pt a SIN, would cost my job.

(a SIN, would cost my job / is correct, my boss would give me a rise).

Read each of the 4 code segments and state if we created a case of combinational feedback. Here i, j, dcount, next_dcount are all 4-bit numbers.



$$X = X + 1$$

#1
assign i = i + 1;

#2
always @(*)
i = i + 1;

#3
always @(*)
i <= i + 1;

#4
always @(*) begin: named_USC
reg [3:0] j; ← local declaration
j = dcount;
j = j + 1;
if (j >= 10)
j = 4'b0000;
else
j = j + 1;
next_dcount = j;
end
always @(posedge clk)
dcount <= next_dcount;

Note: In parity generation code we did ptemp = ptemp ^ x[i];

j can not be seen outside
This is partly to express the relation between

	Combinational feedback exists in
Code #1	Yes / No
Code #2	Yes / No
Code #3	Yes / No
Code #4	Yes / No

Code #4 is trying to design a decade counter, as you would have figured out. This decade counter was purposefully not provided with any reset. We let it come up randomly to any value, including illegal values, but it will soon fall into legal values. Please look at the code for proper usage of blocking and non-blocking assignment operators, and state if there are any errors and also if there is any flexibility anywhere of using either (either blocking or non-blocking).

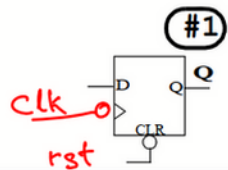
Code is good.

3 pts Flexibility is available for the final assignments within a combinational always procedural block. Here, the last line assigning to next_dcount can use non-blocking: next_dcount <= j;

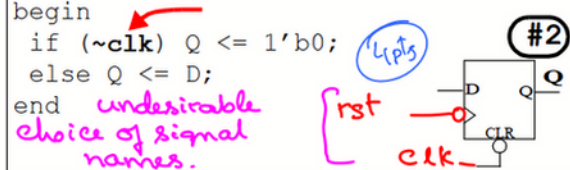
ee201_midterm1_Sp2010_r1.fm

2.6 Draw the flip-flop and any additional logic (such as recirculating mux or some gates) inferred by the following code snippets. Cross-out any bad coding snippets.

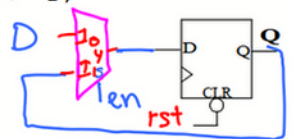
always @(negedge clk, negedge rst)
begin
if (~rst) Q <= 1'b0;
else Q <= D;
end



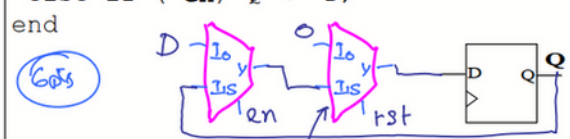
always @(negedge clk, negedge rst)
begin
if (~clk) Q <= 1'b0;
else Q <= D;
end



always @(negedge clk, negedge rst)
begin
if (~rst) Q <= 1'b0;
else if (~en) Q <= D;
end



always @(negedge clk
begin
if (~rst) Q <= 1'b0;
else if (~en) Q <= D;
end

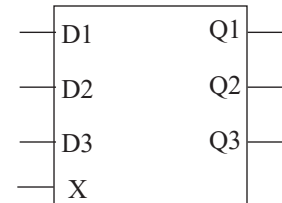


can be simplified into gates

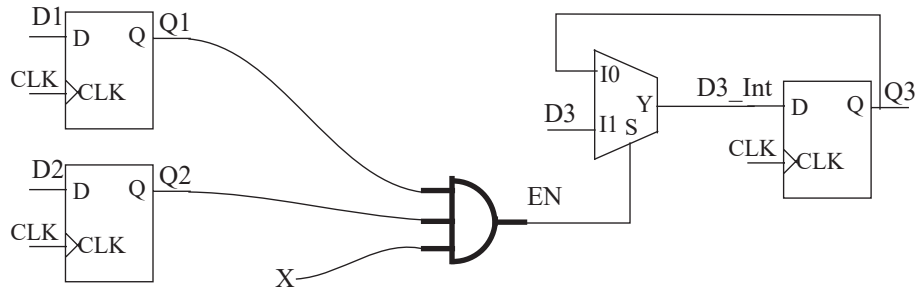
5 (8 + 8 = 16 points) 10 min. Schematic => Verilog Coding

Given below is a small schematic and the associated Verilog code containing two assign statements and three always clocked blocks. Rewrite the code without any assign statements in two ways: Method #1: one clocked always block Method #2: Q1 and Q2 generated in one always clocked block and Q3 generated in a separate clocked block. Do not worry about the data types of signals (wire or reg). Let us assume that they are adjusted accordingly.

module Inputs/Outputs



EN and D3_Int are internal signals (not module port pins)



```

1  always @(posedge CLK)
2      Q1 <= D1;
3
4  always @(posedge CLK)
5      Q2 <= D2;
6
7  assign EN = Q1 & Q2 & X;
8
9  assign D3_Int = EN ? D3 : Q3;
10
11 always @(posedge CLK)
12     Q3 <= D3_Int;

```

Method #1: one clocked always block

```

always @(posedge CLK)
begin
    Q1 <= D1;
    Q2 <= D2;
    Q3 <= (X & Q1 & Q2) ? D3 : Q3;
end

```

8+8
pts

It is not desirable to do low-level coding of the AND gate and the mux by producing the unneeded intermediate signals, EN and D3_Int. But if they are produced, they should be produced using Blocking assignment operator. The three Q's shall always be assigned using non-blocking assignment operator.

Method #2: Q1 and Q2 in one always block
Q3 in another

```

always @(posedge CLK)
begin
    Q1 <= D1;
    Q2 <= D2;
end

always @(posedge CLK)
begin
    Q3 <= (X & Q1 & Q2) ? D3 : Q3;
end

```

It is important that (X & Q1 & Q2), which is the upstream logic of Q3, stays in the always block producing Q3. It is wrong if students put it in the other always block.

We enjoyed teaching this course. Several of you made very interesting projects. Hope to see you in EE457. - Gandhi