Day Zero Notes for Digital Design Class

Digital Design class builds on topics from Digital Logic class. Because it likely has been 9 months since you thought about designing logic circuits it is important to do a little review. Class will begin with a **quiz** the **first day** of class that covers topics that should have been learned in digital logic class. Thus you should review the topics listed below before coming to class.

The first part of digital logic class covered combinational logic which includes boolean algebra, logic functions, minimizing logic expressions for efficient circuit implementation, etc. Various combinational logic circuits were then designed and constructed using basic logic gates. The later part of digital logic class introduced sequential logic circuits, i.e. state machines, and how to design them.

Here are topics that need review. Make sure you understand each.

> Boolean algebra axioms, theorems, and properties [1]:

Axioms		Theorems	
1*a*.	$0 \cdot 0 = 0$	5*a*.	$x \cdot 0 = 0$
1*b*.	$1 + 1 = 1$	5*b*.	$x + 1 = 1$
2*a*.	$1 \cdot 1 = 1$	6*a*.	$x \cdot 1 = x$
2*b*.	$0 + 0 = 0$	6*b*.	$x + 0 = x$
3*a*.	$0 \cdot 1 = 1 \cdot 0 = 0$	7*a*.	$x \cdot x = x$
3*b*.	$1 + 0 = 0 + 1 = 1$	7*b*.	$x + x = x$
4*a*.	If $x = 0$, then $\bar{x} = 1$	8*a*.	$x \cdot \bar{x} = 0$
4*b*.	If $x = 1$, then $\bar{x} = 0$	8*b*.	$x + \bar{x} = 1$
		9.	$\bar{\bar{x}} = x$

`Properties

10*a*.	$x \cdot y = y \cdot x$	*Commutative*
10*b*.	$x + y = y + x$	
11*a*.	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$	*Associative*
11*b*.	$x + (y + z) = (x + y) + z$	
12*a*.	$x \cdot (y + z) = x \cdot y + x \cdot z$	*Distributive*
12*b*.	$x + y \cdot z = (x + y) \cdot (x + z)$	
13*a*.	$x + x \cdot y = x$	*Absorption*

Properties continued

13*b*.	$x \cdot (x + y) = x$	
14*a*.	$x \cdot y + x \cdot \overline{y} = x$	*Combining*
14*b*.	$(x + y) \cdot (x + \overline{y}) = x$	
15*a*.	$\overline{x \cdot y} = \overline{x} + \overline{y}$	*DeMorgan's theorem*
15*b*.	$\overline{x + y} = \overline{x} \cdot \overline{y}$	
16*a*.	$x + \overline{x} \cdot y = x + y$	
16*b*.	$x \cdot (\overline{x} + y) = x \cdot y$	
17*a*.	$x \cdot y + y \cdot z + \overline{x} \cdot z = x \cdot y + \overline{x} \cdot z$	*Consensus*
17*b*.	$(x + y) \cdot (y + z) \cdot (\overline{x} + z) = (x + y) \cdot (\overline{x} + z)$	

> Boolean expressions; SOP (sum of products); POS (product of sums)

$F = B\overline{C} + \overline{A}C + \overline{B}\overline{C} = (B + C)(\overline{A} + \overline{B} + \overline{C})$

sop pos

Prove that these two expressions, SOP and POS, are equivalent using Boolean properties and theorems. Both expressions should produce the same function.

> Canonical Boolean expressions

Canonical expressions have all variables in each term. For example a three variable canonical function could be: $F = \overline{A}\overline{B}C + \overline{A}BC + A\overline{B}C + AB\overline{C}$

> Basic logic gates: AND, OR, NAND, NOR, XOR, NOT. Know the truth tables for these.

A B	F
0 0	0
0 1	0
1 0	0
1 1	1

AND

A B	F
0 0	0
0 1	1
1 0	1
1 1	1

OR

A B	F
0 0	1
0 1	1
1 0	1
1 1	0

NAND

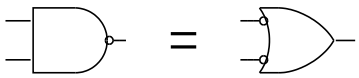
A B	F
0 0	1
0 1	0
1 0	0
1 1	0

NOR

A B	F
0 0	0
0 1	1
1 0	1
1 1	0

XOR

> DeMorgan's theorem and its application to drawing logic diagrams:



These two gates are identical hardware but are performing different logic functions. Be able to convert gates as needed using DeMorgan's theorem. Recall that when drawing a logic diagram the body of the symbol must show the logic function being performed by that gate. Thus use DeMorgan to achieve this..

> Function minimization using regular K-Maps for functions with two, three or four variables. (We will not use regular K-maps to minimize functions with more than four variables.)

Example K-map. What is the minimized function found using this 3-variable map?

a \ b c	00	01	11	10
0	0	0	1	0
1	1	0	1	1

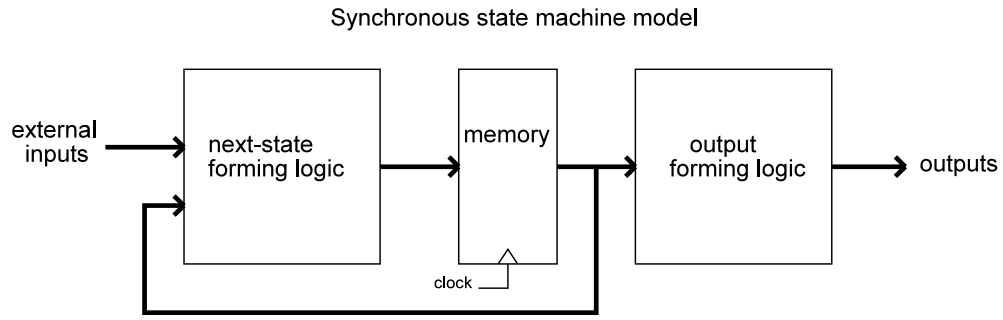
F = ?

> Function minimization using Entered Variable maps (i.e. EV maps)

Example entered variable K-map. What is the minimized function from this map?

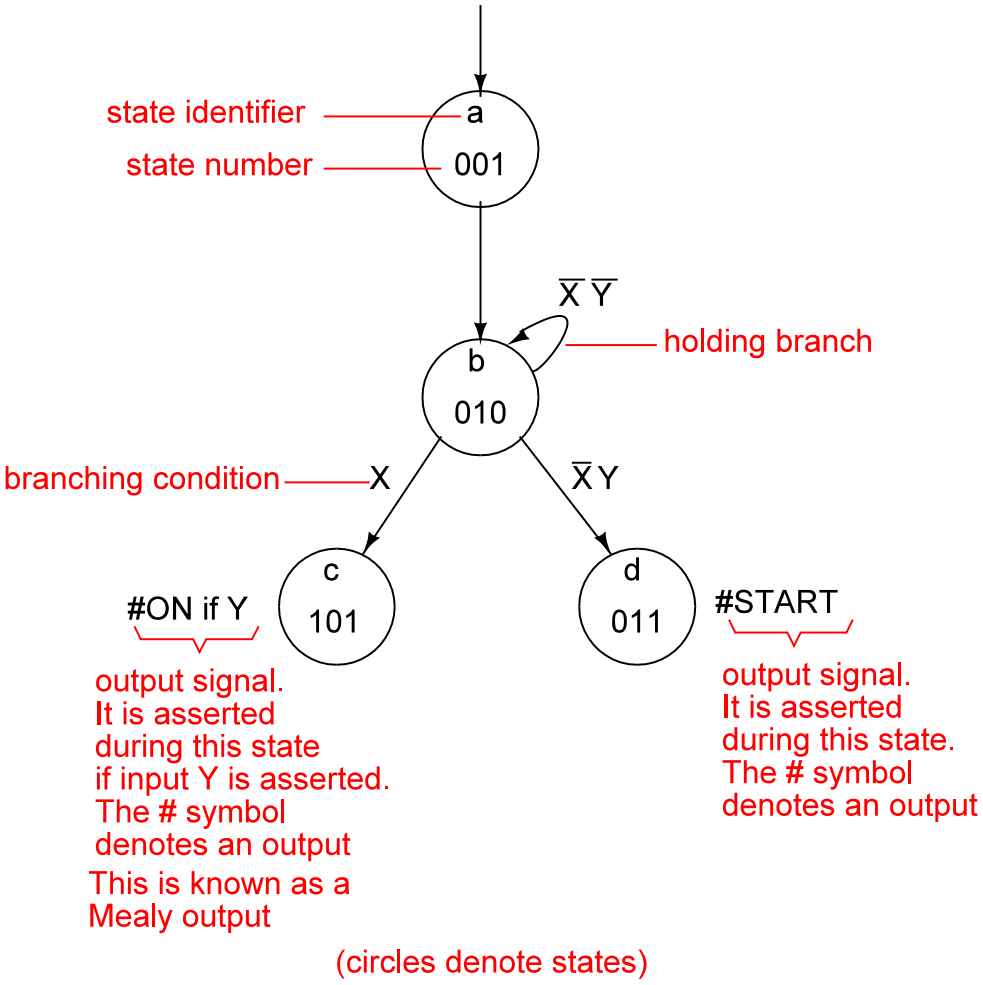
a \ b	0	1
0	0	X
1	1	1

> To understand how a state machine operates I find a block diagram of a generic state machine helpful. I hope this diagram prompts your memory to recall the functional parts of a state machine:



Recall that the name Synchronous State Machine is used because transitions from one state to another only occur on a clock edge, hence the machine is synchronized with the clock. This model is called a Moore state machine model because the outputs depend only on what state the machine is in. Can you name and describe a state machine whose outputs depend not only on state but also on external inputs?

You should have been introduced to state diagrams which are a prime way of describing the operation of a state machine. There are various ways to draw them. Below is an example section of a state diagram illustrating the way we will draw state diagrams in this class. You must use this style for homework, exams, and lab work. Get use to it.



Templates should be used in this class for hand drawn logic diagrams.

[1] Boolean laws scanned from *Fundamentals of Digital Logic with VHDL Design* 3rd ed. by Brown and Vranesic. McGraw Hill.

L.Aamodt