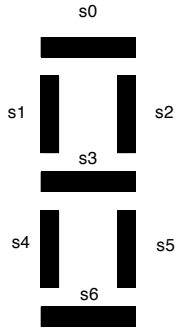
Harvey Mudd College
Computer Science 65
Fall 2008

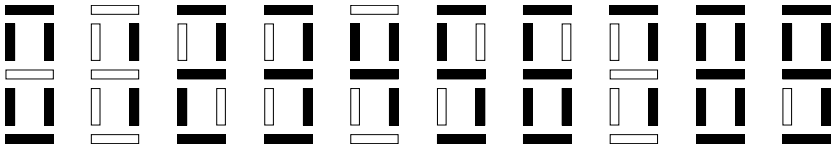
Assignment 5
Logic: Combinational vs. Sequential
Due. 11:59 p.m., Wed., 8 Oct. 2008

Problem 1:

A seven-segment display consists of seven LEDs (light-emitting diodes), numbered s0 through s6, which display the digits from 0 through 9 as shown:



Seven segment LED display



Display of 0 through 9 with a seven-segment display

In the following, we assume that the digits are coded in BCD (see below). This uses ten of sixteen possible combinations of four bits. The remaining combinations are don't cares. The seven segments correspond to seven logic functions, computed by a decoder that you will provide:

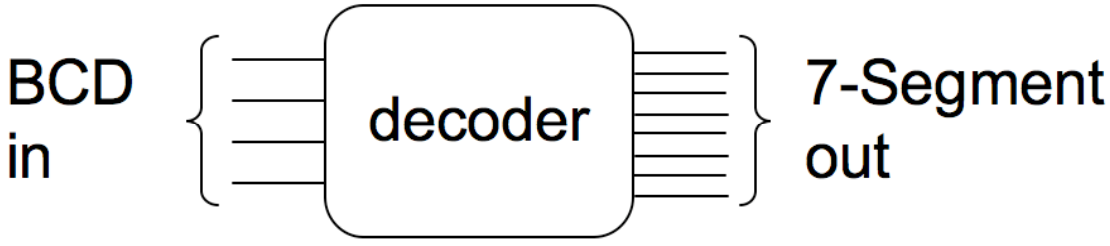


diagram of the decoder

a. Simplify each of the switching functions, using don't cares to advantage, to achieve a 2-level SOP (sum-of-products) realization. This means that each function is represented as the ***or*** of several terms, with each term being an ***and*** of a variable or the ***not*** of a variable. For example, using Scheme syntax, the function s0 can be expressed as follows:

(**or** w y (**and** x z) (**and** (**not** x) (**not** z)))

b. Using the test file provided on the website, transcribe your function definitions to the Scheme function

(**seven-segment** w x y z)

for testing. Use constants #f and #f rather than 0 and 1 for truth values.

Aligned to the BCD encodings are seven segment functions

	BCD	seven-segment							
	wxyz	**s_0**	**s_1**	**s_2**	**s_3**	**s_4**	**s_5**	**s_6**	**s_7**
0	0000	1							
1	0001								
2	0010	1							
3	0011	1							
4	0100				fill				
5	0101	1			in				
6	0110	1			yourself				
7	0111	1							
8	1000	1			or, better, go				
9	1001	1			directly to a Karnaugh map				

Problem 2:

Suppose we wish to build a **counter** device that will count up or down in the range {0, 1, 2, 3}. The output will be displayed using our seven segment display in problem 1, so we will encode these four numbers in BCD, fixing the most-significant two bits to be 0. The counter has two bits of input, with the following interpretation:

00 The value of the counter does not change
01 The counter counts up
10 The counter counts down
11 Assumed never to occur, so we don't care

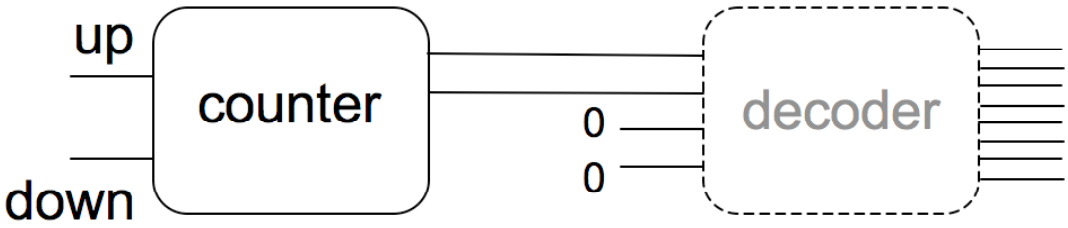


diagram of the counter, in context

a. Show the state diagram for your counter.

b. Transcribe the state diagram to our Scheme finite-state machine simulator in the prescribed format, for testing. Name the machine **'counter**. Make sure that it runs the tests cases.

c. Assuming that the state of the counter is encoded to be identical to the output (i.e. a 2-bit encoding) construct next-state logic functions for your counter, and check them using Scheme.

Problem 3:

Using the format of our finite-state machine simulator, construct machines with the given names that do the following, where in each case the input and output alphabets are {0, 1}. Please make your intial state be the first state listed, so that the test cases can be used.

machine3a: The output is 1 iff the input sequence has no more than two 0's.

machine3b: The output is 1 iff the input sequence has no more than two 0's in a row or nor more than two 1's in a row.

machine3c: The output is 1 iff the input sequence interpreted in binary, *most-significant bit first*, is a multiple of 10 decimal, such as the following:

1010	(10 decimal)
10100	(20 decimal)
11110	(30 decimal)
001010	(10 decimal)
010100	(20 decimal)
0000011110	(30 decimal)
etc.	

The numerals can be arbitrarily large. By convention, we agree that the empty sequence will be treated as a multiple of 10 decimal.