University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

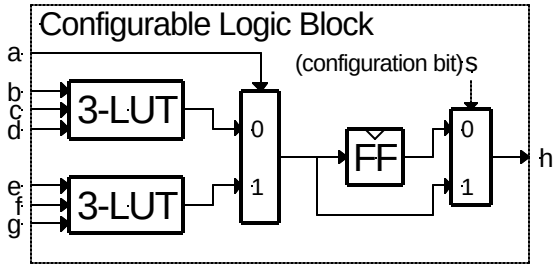
EECS150 J. Wawrzynek
Spring 2000 E. Caspi

Quiz #3 – Solution

There are many possible solutions – we present 3 plausible ones. There are two preliminary points which are important in all solutions:

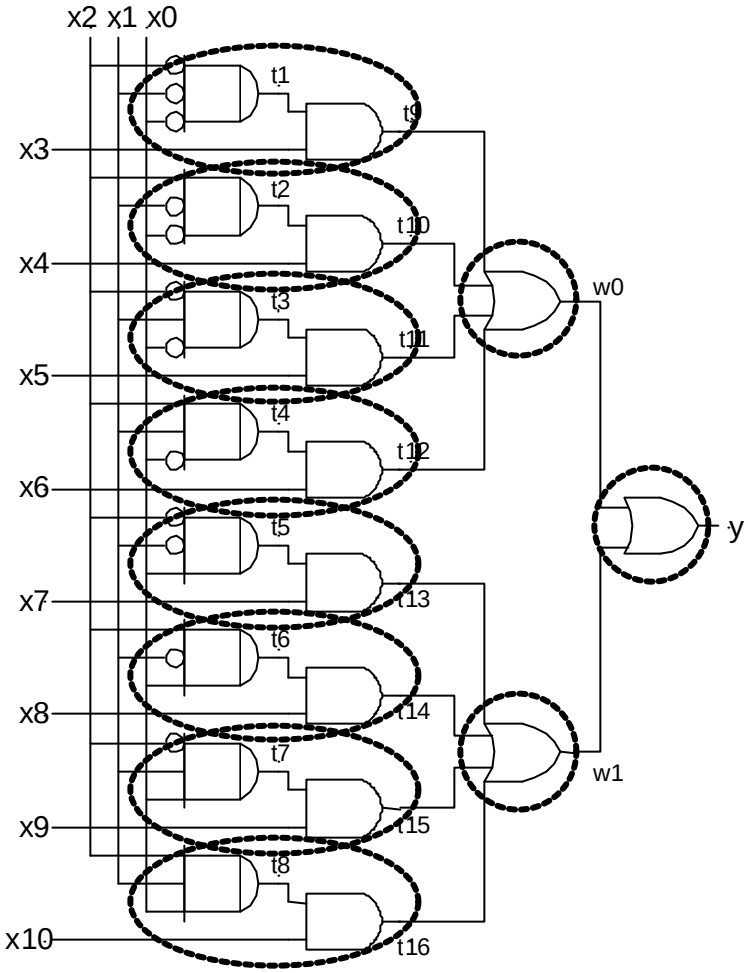
- **s=1.** Since this is a combinational circuit, not a sequential one, we do not need the flip flop. In all cases, the output multiplexor should pass on the signal which bypasses the flip flop (*i.e.* s=1).
- **Decompose the 8-input OR.** A single CLB cannot possibly implement the 8-input OR which generates *y*. You must decompose that OR gate. This is easy, since OR is associative: $a+b+c = (a+b)+c = a+(b+c)$. That means a wide OR can be implemented by a sequence of smaller ORs as well as by a tree (as in Homework #2, problem 2, which did the same for a wide XOR). To minimize total CLB usage, the particular OR decomposition you choose should depend on the partitioning of the rest of the circuit.

Recall that this is our CLB:



The easiest way to use the CLB is as a 4-LUT. We compose the 4-LUT from the pair of 3-LUTs by feeding the same 3 inputs into {*b*,*c*,*d*} as into {*e*,*f*,*g*} and by using *a* for the fourth input. The first 3-LUT implements the 4-LUT function assuming *a*=0 while the second assumes *a*=1. This technique is demonstrated in Homework #3, problem 3.

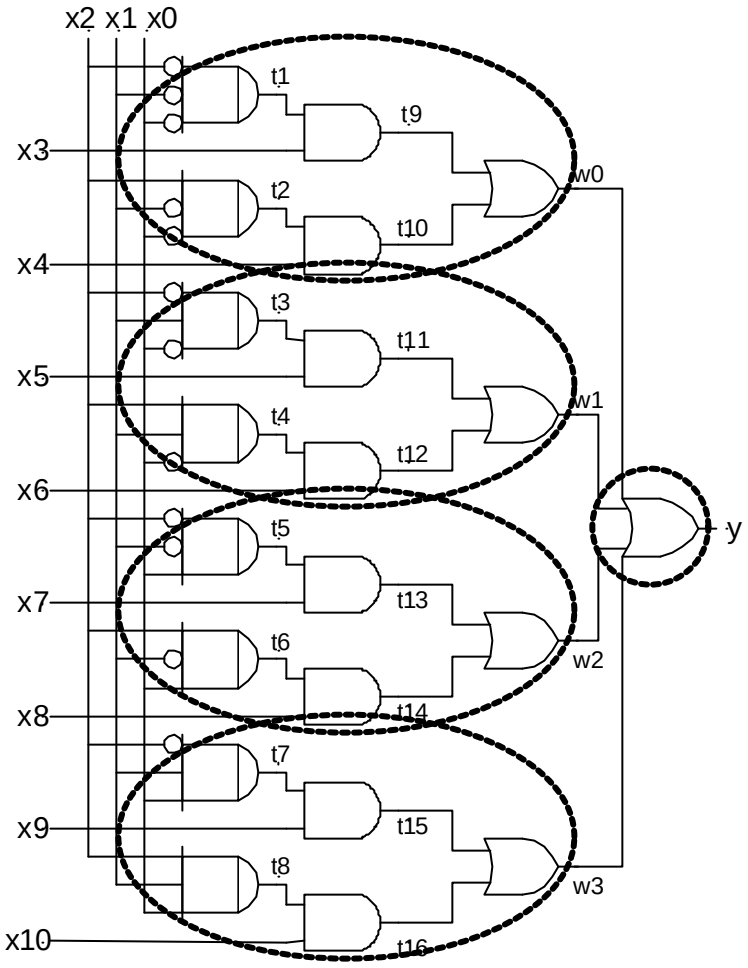
A very simple partitioning can be done using 11 4-LUTs. Each of the 8 AND-AND terms uses one CLB, and the OR is partitioned into a tree of 3 CLBs.



a	b	c	d	e	f	g	s	h
x3	x0	x1	x2	x0	x1	x2	1	t9
x4	x0	x1	x2	x0	x1	x2	1	t10
x5	x0	x1	x2	x0	x1	x2	1	t11
x6	x0	x1	x2	x0	x1	x2	1	t12
x7	x0	x1	x2	x0	x1	x2	1	t13
x8	x0	x1	x2	x0	x1	x2	1	t14
x9	x0	x1	x2	x0	x1	x2	1	t15
x10	x0	x1	x2	x0	x1	x2	1	t16
t12	t9	t10	t11	t9	t10	t11	1	w0
t16	t13	t14	t15	t13	t14	t15	1	w1
0	w0	w1	-	-	-	-	1	y

Implementation in 11 CLBs (4-LUT configuration).

The following partition in 5 CLBs is due to Drew Pertula. The 8 AND-AND terms can be grouped into 4 pairs, where the difference between each pair is the inversion of the x2 input. This guarantees that, in any such pair, one AND-AND term is forced to zero. Since the AND-AND terms are subsequently OR-ed, the value of x2 effectively selects which among each pair of AND-AND terms will pass to the output. Now we can pack each pair of OR-ed AND-AND terms into a single CLB – each AND-AND (now without x2 as an input) gets a 3-LUT, and x2 controls the multiplexer to select one of them. This requires 4 CLBs (2 AND-ANDs in each), and one additional CLB to OR their outputs (using a 4-LUT configuration). Total: 5 CLBs.

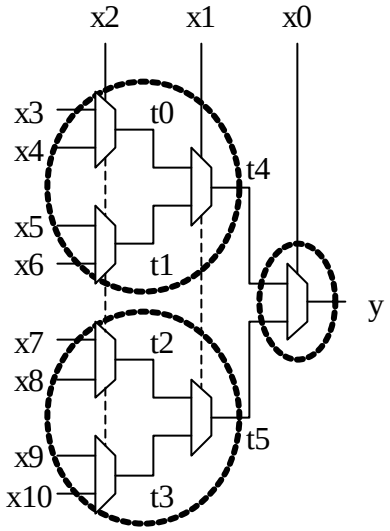


a	b	c	d	e	f	g	s	h
x2	x0	x1	x3	x0	x1	x4	1	w0
x2	x0	x1	x5	x0	x1	x6	1	w1
x2	x0	x1	x7	x0	x1	x8	1	w2
x2	x0	x1	x9	x0	x1	x10	1	w3
w3	w0	w1	w2	w0	w1	w2	1	y

Implementation in 5 CLBs.

We can do one better – 3 CLBs. You must first realize that the circuit is an 8-to-1 multiplexor. It chooses one of {x3,…,x10} according to the select bus {x2,x1,x0}. The first column of ANDs implements a decoder, converting the binary number {x2,x1,x0} into a one-hot representation {t1,…,t8} – only one of {t1,…,t8} will be 1, the rest 0. The second column masks out the unselected inputs from {x3,…,x10} by AND-ing them with 0 – only one of {t9,…,t16} will actually copy an x input, the rest will be 0. The final OR simply passes-on whichever result was selected.

An 8-to-1 multiplexor can be implemented by a tree of 2-to-1 multiplexors. A 2-to-1 multiplexor fits in a 3-LUT. A CLB combines 2 2-to-1 muxes into a 4-to-1 mux. A pair of such CLBs fed through a 2-to-1 multiplexor in a third CLB forms an 8-to-1 mux.



a	b	c	d	e	f	g	s	h
x1	x3	x4	x2	x5	x6	x2	1	t4
x1	x7	x8	x2	x9	x10	x2	1	t5
0	t4	t5	x0	-	-	-	1	y

Implementation in 3 CLBs.