EECS 579 Homework No.4 Solutions

Problem 1 (20 points) *Pseudoexhaustive testing*

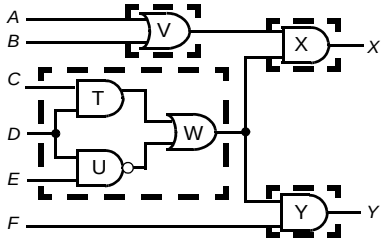


Fig. 1

(**a**) Output *X* is a function of 5 inputs *A,B,C,D,E* and so requires $2^5 = 32$ tests, while output *Y* is a function of 4 inputs *C,D,E,F*, requiring $2^4 = 16$ tests. It's easy to overlap these tests, i.e., apply them in parallel, leading to the result that 32 tests patterns are needed to pseudoexhaustively test the entire circuit.

(**b**) Subcircuit *TUW* has 3 inputs and requires $2^3 = 8$ tests, while all the other subcircuits have just 2 inputs and require 4 tests each. We must also propagate the test responses from the internal subcircuits *TUW* and *V* to at least one of the 2 primary outputs. For example, we can propagate all the responses of *TUW* to primary output *Y* by setting *F* = 1. This will also apply 2 of its 4 tests, namely *WF* = 01 and 11, to subcircuit (AND gate) *Y*. We then require two additional tests that make *WF* = 10 and 10 to complete the testing of subcircuit *Y*. This leads to 10 tests overall. These 10 patterns can easily be overlapped with the tests needed by subcircuits *V* and *X*, which must be propagated to primary output *X*, and so are largely independent of the tests being propagated to output *Y*.

As most of you noted, we can test the entire circuit with just 9 tests if we propagate one of the responses of *TUW* to output *X* and the other responses to output *Y*. This is the minimum possible, since the 8 patterns that exhaustively test *TUW* produce only one 0 on line *W*, so at least one additional test pattern is needed to ensure that subcircuit *Y* has both 00 and 01 applied to it.

Problem 2 (20 points) *Memory tests*

These problems are easy, but they require an understanding of the text's shorthand notation for memory faults and tests.

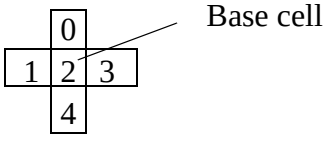
(**a**) *Text, p.308, Prob.9.12.* The MATS++ algorithm is described concisely as follows (see text p. 286):

{⇑⇓(*w*0); ⇑(*r*0,*w*1); ⇓(*r*1,*w*0,*r*0)}

It consists of three "march elements" M0, M1 and M2. MATS++ is exactly the same as MATS+ (which is given in full detail on p. 264) with the addition of an extra *r*0 step in M2.

Proof that MATS++ detects all cell stuck-at-0/1 faults: Element M0 writes 0 into every cell of the memory. If any cell is stuck-at-1, that fault will be detected by *r*0 in M1. Element M1 also writes 1 into every cell, so a stuck-at-0 fault will be detected by *r*1 in M2. It can also be proven that stuck-at faults affecting various lines such as an address or data line are also detected by MATS++.

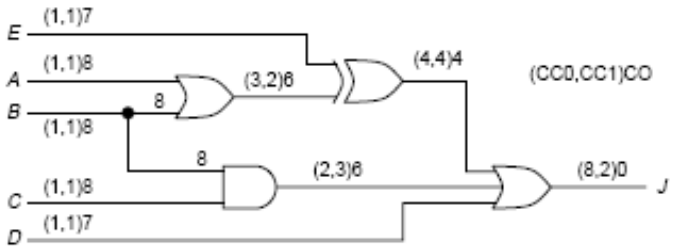
(**b**) *Text, p.308, Prob.9.22*. We need to construct tests for the specific pair of passive neighborhood pattern sensitive faults (PNPSFs) < 1, 0, 1,0; ↑/0 > and < 0, 1, 0, 1; ↓/1 >. Observe that ↑/0 denotes an up transition fault, where the base cell fails to perform the 0-to-1 transition; ↓/1 denotes the corresponding down transition fault (see p. 268). Each such fault requires *two* writes (to create a transition) followed by a read. Some of you used just one write, in effect, testing only for (pattern-sensitive) stuck-at faults.



Any clear, program-like shorthand is acceptable as the pseudocode. The main test steps are listed below. They can easily be put into an iterative (loop) structure to match any given memory array type.

Write 0 to base cell 2;	Write 1 to base cell 2;
Write 1010 to cells 0,1,3,4;	Write 0101 to cells 0,1,3,4;
Write 1 to base cell 2;	Write 0 to base cell 2;
Read base cell (The test fails if 0 is read);	Read base cell (The test fails if 1 is read);

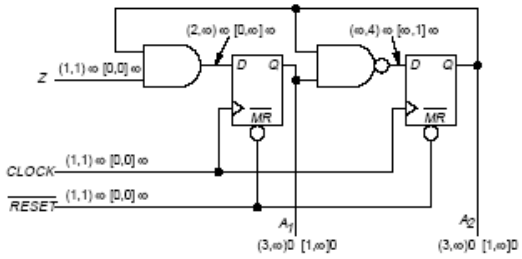
Problem 3 (20 points) *Combinational SCOAP (*Text, p. 151, Prob. 6.5)



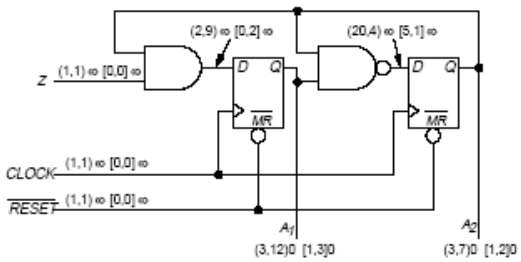
Problem 4 (20 points) *Combinational and sequential SCOAP (*Text, p. 153, Prob.6.13)

As several of you noticed and brought to my attention, the first printing of the book in 2000 has testability formulas on p. 141 which contain many small errors. These errors were corrected in the 2001 printing. Algorithm 6.2 on p. 142 is the same in both printings, but the different controllability/observability formulas lead to two different sets of answers. The authors' errata sheets for the book ignore this major group of errors, which is surprising. It turns out, however, that the "right" and "wrong" answers are fairly close to each other. For example, in the case of line A_2 on the bottom right of the circuit, the "wrong" formulas yield (2,6)0; [1,2]0, whereas the "right" formulas yield (3,7)0; [1,2]0. Either set of formulas would probably be OK to use in practice as a guide in ATPG or DFT procedures. In any case, I accepted either set and graded the problem generously.

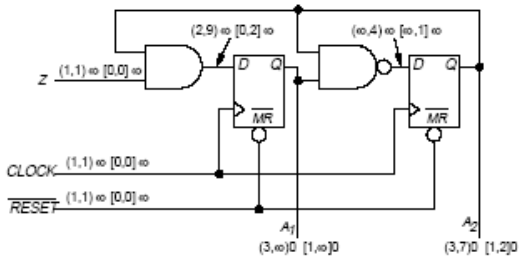
The main steps in calculating the testability measures according to the "right" formulas from the 2001 printing are shown in the four figures below. The combinational measures are written as (CC0,CC1)CO and the sequential measures as [SC0,SC1]SO.



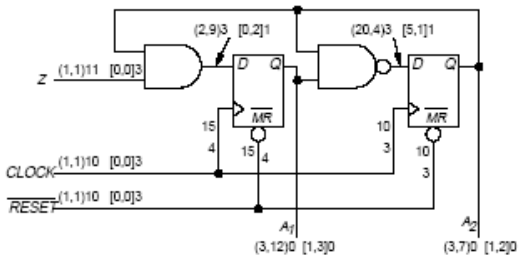
Circuit of Figure 6.31: Initialization and first controllability pass.



Circuit of Figure 6.31: Stabilized controllability values.



Circuit of Figure 6.31: Continuation of controllability calculation.



Circuit of Figure 6.31: All controllability and observability values.

Problem 5 (20 points) *C-testability*

		a=0	a=1
b= 0	S_0	S_0,0	S_0,1
b= 1	S_1	S_0,1	S_1,0

Observe that this ILA is a (ripple-carry) *incrementer*, which computes the arithmetic function $f(A) = A + 1$.

(**a**) The incrementer is *not* C-testable, which is surprising because the more general ripple-carry adder discussed in class, which computes $f(A,B) = A + B$, *is* C-testable. This can be seen from the fact that it is not possible to apply the pattern $ab = 01$ to more than one cell of the ILA at a time. Each of the other three ab patterns can be applied to all cells simultaneously.

Another proof comes from redrawing C's truth table in the sequential, state-table format shown above. We see that S_0 is a "trap state" which C can never exit once it enters it, and S_1 is a transient state to which C can never return once it leaves it. Applying $ab = 01$ to C corresponds to applying $a = 0$ in state S_1 which causes S_1 to be exited, so the pattern $ab = 01$ can only be applied to one cell at a time. See Theorem 8.2 in the Abramovici et al. text for the general conditions for C-testability, which cell C does not meet.

(**b**) From the above discussion, we can easily construct a complete test set for an N-cell ILA. The minimum number of tests required is $N + 3$.

Test			**Comment**
$a_{N-1}a_{N-2}$	…	$a_1a_0b_{in}$	
0 0	…	0 0 0	Applies 00 to every cell simultaneously
1 1	…	1 1 0	Applies 10 to every cell simultaneously
1 1	…	1 1 1	Applies 11 to every cell simultaneously
x x	…	x 0 1	Applies 01 to cell 0
x x	…	0 1 1	Applies 01 to cell 1
	…		
0 1	…	1 1 1	Applies 01 to cell $N-1$