EEDG/CE 6303: Testing and Testable Design (Fall'2021)

Department of Electrical & Computer Engineering

The University of Texas at Dallas

Instructor: Mehrdad Nourani (nourani@utdallas.edu)

Cover Page for All Submissions

(Assignment, Project, Codes/Simulations/CAD, Examinations, etc.)

Last Name (as shown in the official UT Dallas Student ID Card): Feng

First Name: Yu

Submission Materials for (e.g. Homework #, Project #): HW 4

Statement of Academic Honesty

I certify that:

i. the attached report (for assignment, project, codes/simulations/CAD, examinations, etc.) is my own work, based on my personal study and/or research,
ii. I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication,
iii. this report has not previously been submitted for assessment in EEDG/CE 6303 or any other course at UT Dallas or elsewhere,
iv. I have not copied in part or whole or otherwise plagiarized the work of other students and/or persons, and
v. I have read and understood the Department and University policies on scholastic dishonesty as outlined in: http://www.utdallas.edu/deanofstudents/dishonesty/.

Name: Yu Feng Date: 10/31/2021

Signature: Yu Feng

The University of Texas at Dallas
Dept. of Electrical and Computer Engineering

EEDG/CE 6303: Testing and Testable Design
HW # 4: *Due on Tuesday, Nov. 2, 2021 - 11:59 pm (US CST)*

When you submit your homework, to help us grade and identify your work, you need to comply with the following guidelines carefully:

- *Have* **a cover page** *for each document (e.g. homework, project, report, etc.) that you submit. A sample of cover page is provided in the course webpage. This page must include: (1) your name as it appears in your* **student ID card***, (2) course name/number, (3) homework/project number, and (4) the* **Statement of Academic Honesty** *that you sign.*

1. Find the robust and non-robust test patterns for the following **path delay** faults in Figure 8.13: (i) $\uparrow x_3c_2c_3z$, (ii) $\downarrow x_3c_2c_3z$, (iii) $\uparrow x_1c_1c_3z$, (iv) $\downarrow x_1c_1c_3z$.

2. Find the robust and non-robust test patterns for the following **path delay** faults in Figure 8.20: (i) $\uparrow x_3c_2c_3c_4c_6z$, (ii) $\downarrow x_3c_2c_3c_4c_6z$, (iii) $\uparrow x_2c_1c_3c_5z$, (iv) $\downarrow x_2c_1c_3c_5z$.

3. Consider the above two circuits (Figures 8.13 and 8.20 and path delay faults). Use Synopsys toolset to implement the circuits and find robust and non-robust test patterns. A short guide for path delay fault testing is at the end of Test Tutorial in the course webpage. Report and discuss the results.

4. From your text, solve the following problems:
   – Chapter 14: 4, 5, 6, 7 and 8.

5. Determine if: (i) AFs, (ii) SAFs, (iii) TFs, (iv) unlinked CFs (CFin, CFid, CSst), and (v) linked CFids faults can be detected by MARCH C– test shown below.

$$\{M0 :\Updownarrow (w0); M1 :\Uparrow (r0, w1); M2 :\Uparrow (r1, w0); M3 :\Downarrow (r0, w1); M4 :\Downarrow (r1, w0); M5 :\Updownarrow (r0)\}$$

   As you know, in some categories (e.g. coupling faults), there may be multiple faults. You need to justify/prove your answer for each fault. To do this for each case, carefully tabulate which March element(s) (M_i) stimulates it and which element(s) detects it. If there are multiple scenarios to detect a particular fault, list them all.

6. Determine if: (i) AFs, (ii) SAFs, (iii) TFs, (iv) unlinked CFs (CFin, CFid, CSst), and (v) linked CFids faults can be detected by IFA-13 march test shown below.

$$\{M0 :\Uparrow (w0); M1 :\Uparrow (r0, w1, r1); M2 :\Uparrow (r1, w0, r0); M3 :\Downarrow (r0, w1, r1);$$
$$M4 :\Downarrow (r1, w0, r0); Delay; M5 :\Uparrow (r0, w1); Delay; M6 :\Uparrow (r1)\}$$

   As you know, in some categories (e.g. coupling faults), there may be multiple faults. You need to justify/prove your answer for each fault. To do this for each case, carefully tabulate which March element(s) (M_i) stimulates it and which element(s) detects it. If there are multiple scenarios to detect a particular fault, list them all.

7. Read the following paper available in *http://www.utdallas.edu/library/*. Summarize and comment on your understanding of the paper in at most 2 pages.

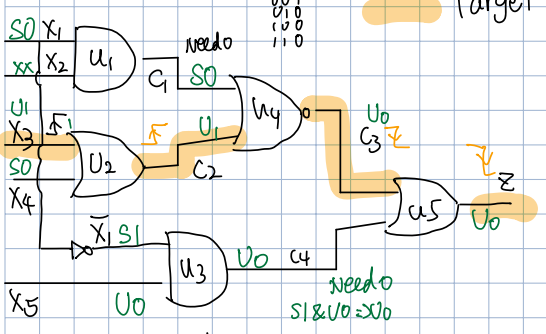
   S. Hamdioui, A. J. van de Goor and M. Rodgers "March SS: A Test for All Static Simple RAM Faults," in *Proceedings of the IEEE International Workshop on Memory Technology, Design and Testing (MTDT 2002)*, pp. 1-6, July 2002.

(f) 1. Find the robust and non-robust test patterns for the following **path delay** faults in Figure 8.13: (i) ↑ $x_3c_2c_3z$, (ii) ↓ $x_3c_2c_3z$, (iii) ↑ $x_1c_1c_3z$, (iv) ↓ $x_1c_1c_3z$.

i) ↑ $x_3 c_2 c_3 z$ Robust

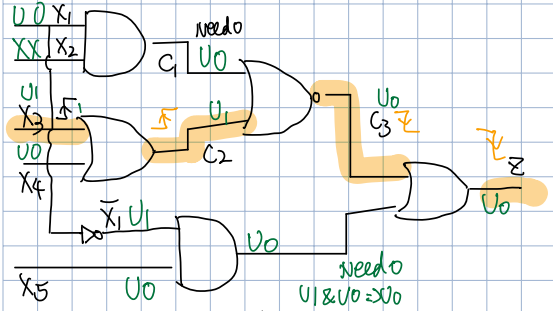
Target path

NOR
0 0 1
0 1 0
1 0 0
1 1 0



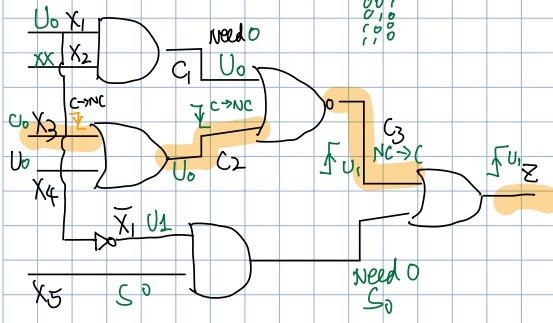
$(X_1, X_2, X_3, X_4, X_5)$
$= (S0, XX, U1, S0, U0)$

Non-Robust test



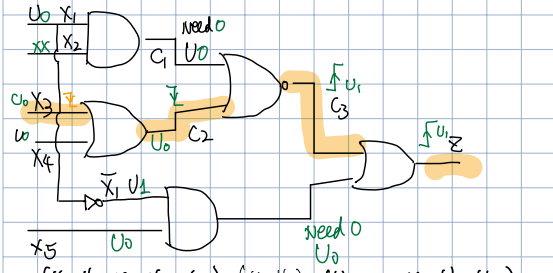
$(X_1, X_2, X_3, X_4, X_5) = (V_1, V_2)$
(U_0, XX, U_1, U_0, U_0)

ii) ↓ $x_3 c_2 c_3 z$ Robust



$(X_1, X_2, X_3, X_4, X_5) = (U_0, XX, U_0, U_0, S_0)$
(V_1, V_2)

Non-Robust



$(X_1, X_2, X_3, X_4, X_5) = (V_1, V_2) = (U_0, XX, U_0, U_0, U_0)$

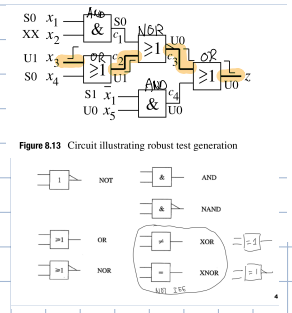


Figure 8.13 Circuit illustrating robust test generation

R

gate type / on-input transition	AND or NAND	OR or NOR
Rising(U1)	U1	S0
Falling(U0)	S1	U0

NC→NC steady
Off-path inputs needed
NC→C steady value

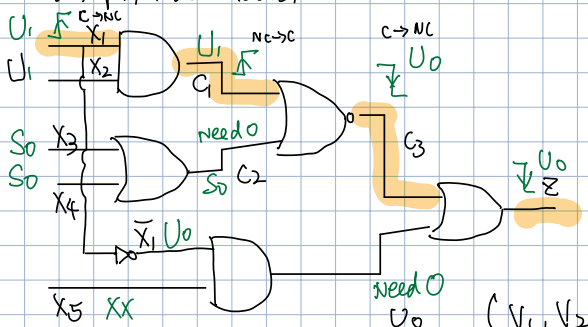
NR

gate type / on-input transition	AND or NAND	OR or NOR
Rising(U1)	U1	U0
Falling(U0)	U1	U0

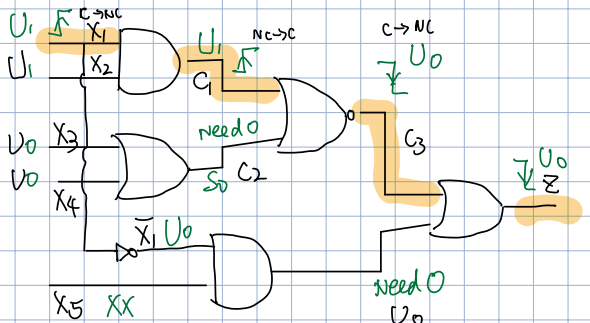
Off-path inputs needed

no reliable hazard

(ii) ↑ $x_1 c_1 c_3 z$ Robust

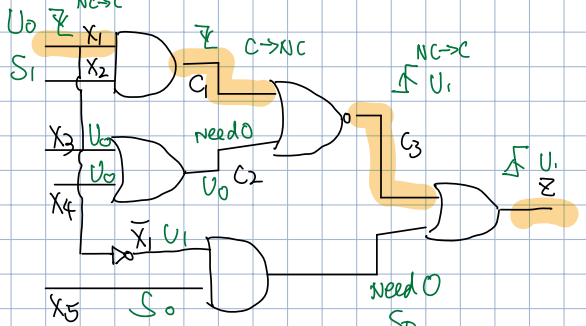


$(X_1, X_2, X_3, X_4, X_5) = (U_1, U_1, S_0, S_0, XX)$
(V_1, V_2)
Non-Robust



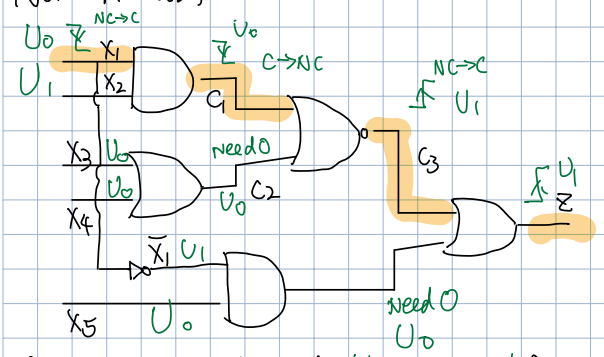
$(X_1 X_2 X_3 X_4 X_5) = (U_1 U_1 U_0 U_0 XX)$
(V_1, V_2)

(iv) ↓ $x_1 c_1 c_3 z$ Robust



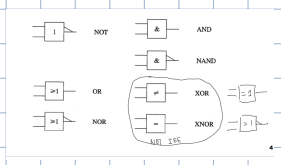
$(X_1, X_2, X_3, X_4, X_5) = (U_0, S_1, U_0, U_0, S_0)$
(V_1, V_2)

Non-Robust



$(X_1, X_2, X_3, X_4, X_5) = (U_0, U_1, U_0, U_0, U_0)$

(V_1, V_2)



2. Find the robust and non-robust test patterns for the following **path delay** faults in Figure 8.20: (i) $\uparrow x_3c_2c_3c_4c_6z$, (ii) $\downarrow x_3c_2c_3c_4c_6z$, (iii) $\uparrow x_2c_1c_3c_5z$, (iv) $\downarrow x_2c_1c_3c_5z$.

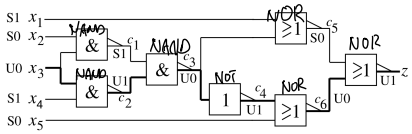
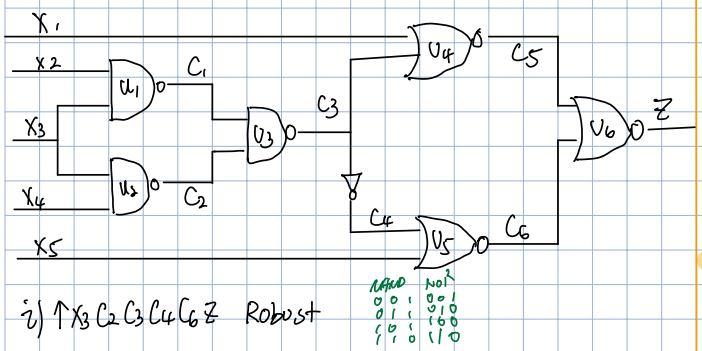
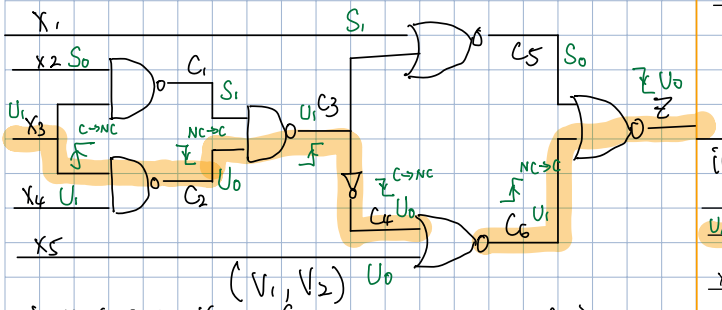


Figure 8.20 S-GDF testing



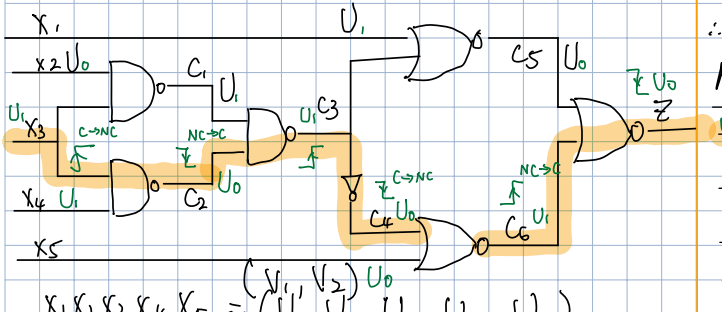
i) $\uparrow X_3C_2C_3C_4C_6Z$ Robust



(V_1, V_2) U_0

$\therefore X_1X_2X_3X_4X_5 = (S_1, S_0, U_1, U_1, U_0)$

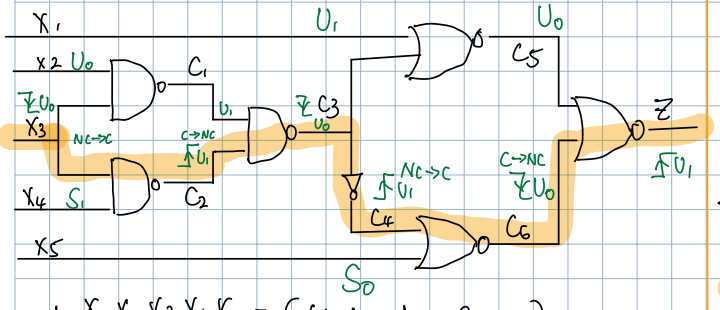
Non-Robust:



(V_1, V_2) U_0

$\therefore X_1X_2X_3X_4X_5 = (U_1, U_0, U_1, U_1, U_0)$

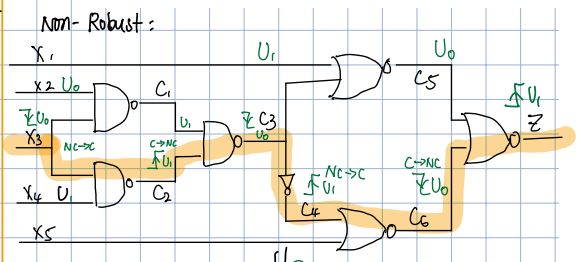
ii) $\downarrow X_3C_2C_3C_4C_6Z$ Robust



$\therefore X_1X_2X_3X_4X_5 = (U_1U_0U_0S_1S_0)$

(V_1, V_2)

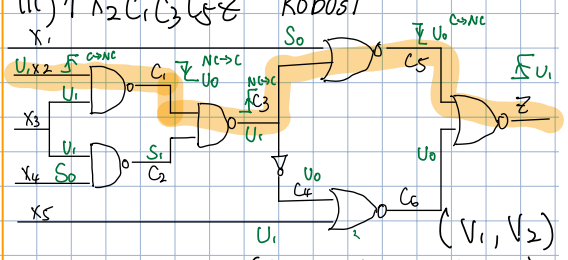
Non-Robust:



$\therefore X_1X_2X_3X_4X_5 = (U_1U_0U_0U_1U_0)$

(V_1, V_2)

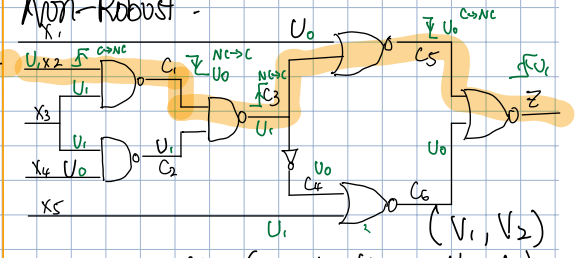
iii) $\uparrow X_2C_1C_3C_5Z$ Robust



(V_1, V_2)

$\therefore (X_1, X_2, X_3, X_4, X_5) = (S_0, U_1, U_1, S_0, U_1)$

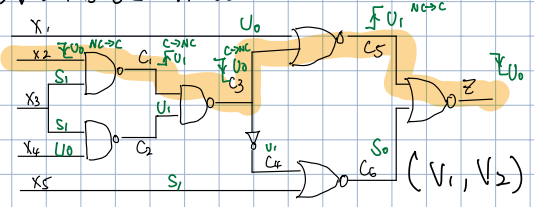
Non-Robust:



(V_1, V_2)

$\therefore (X_1, X_2, X_3, X_4, X_5) = (U_0, U_1, U_1, U_0, U_1)$

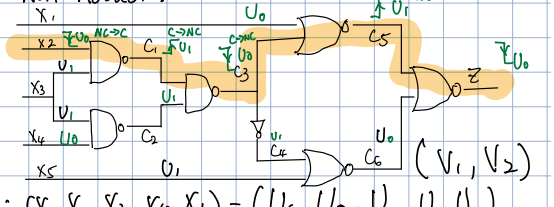
iv) $\downarrow X_2X_1X_3X_5Z$ Robust



(V_1, V_2)

$\therefore (X_1, X_2, X_3, X_4, X_5) = (U_0, U_0, S_1, U_0, S_1)$

Non-Robust:



(V_1, V_2)

$\therefore (X_1, X_2, X_3, X_4, X_5) = (U_0, U_0, U_1, U_0, U_1)$

3.

For the schematic shown in figure 8.13, the schematic in Figure 1 can be generated from VHDL. Its correctness is verified by the waveform as shown in Figure 2.

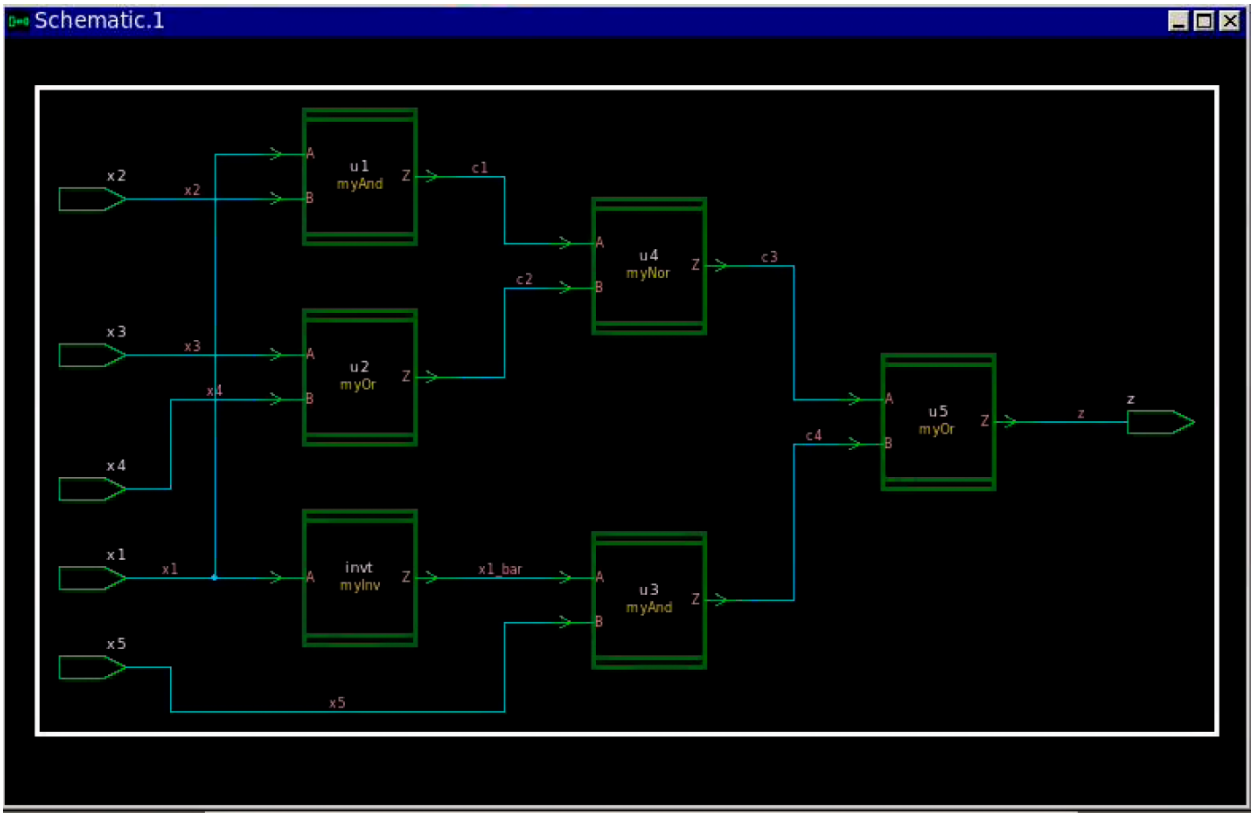


Figure 1 Schematic of Figure 8.13

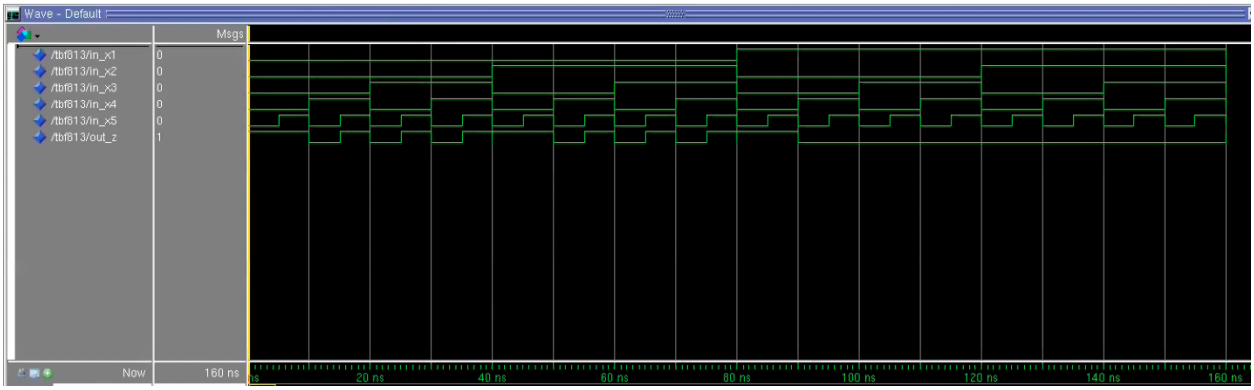


Figure 2 Validation Waveform of Schematic

The delay path of i) and ii) is shown in the following figures. In Figure 4, the delay path analysed by the program have also determined the route through x1_bar and x5 as delay path.

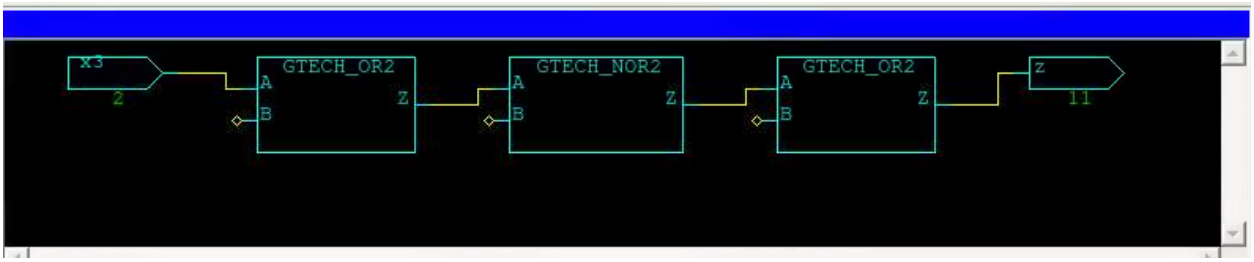


Figure 3 Delay Path of i) and ii) of Q1

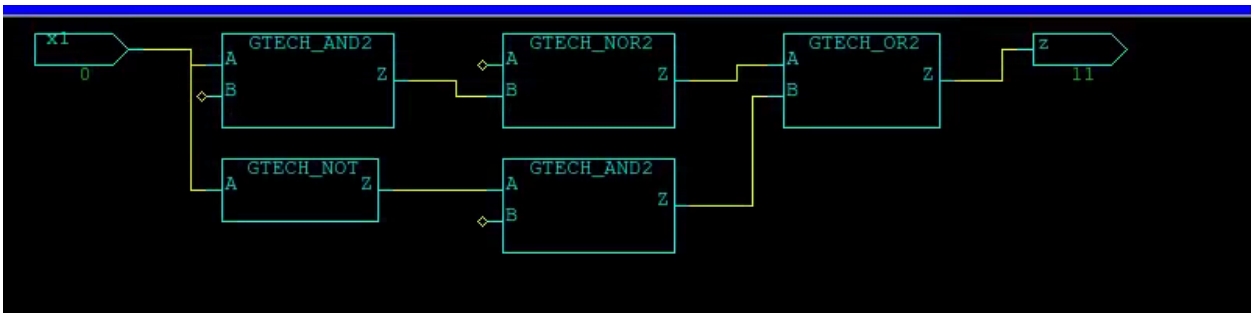


Figure 4 Delay Path of iii) and iv) of Q1

The following figures show the testing patterns for all delay paths. Figure 5 shows Robust test patterns and Figure shows non robust test pattern. Comparing to the manually obtained patterns, patterns generated by the program has no XX (Don't care values) available. For this reason, pattern for i) is different to the result obtained by hand as I treated x2 input as XX instead of setting it to S0.

Report Patterns

Pattern num	Fault name	Fault type	Fault class	Launch clock	Capture clock
0	path1r	str	DR	-	-
1	path2f	stf	DR	-	-
2	path3r	str	DR	-	-
3	path4f	stf	DR	-	-

Loops

Report Patterns

```
Pattern 0 (fast_sequential)
Time 0: force_all_pis =   00001
Time 1: force_all_pis =   10100
Time 2: measure_all_pos = 0
Pattern 1 (fast_sequential)
Time 0: force_all_pis =   10100
Time 1: force_all_pis =   00000
Time 2: measure_all_pos = 1
Pattern 2 (fast_sequential)
Time 0: force_all_pis =   00011
Time 1: force_all_pis =   10111
Time 2: measure_all_pos = 0
Pattern 3 (fast_sequential)
Time 0: force_all_pis =   10101
Time 1: force_all_pis =   00101
Time 2: measure_all_pos = 1
```

Figure 5 Robust Test Patterns

In Figure 6, the same test pattern is used to test i) and iii) in non-robust test scenario. Since setting either gate of AND as don't care results in the same output, the program chose to utilize x2 to save generation of another pattern.

Report Patterns

Pattern num	Fault name	Fault type	Fault class	Launch clock
0	path1r	str	DS	-
0	path3r	str	DS	-
1	path2f	stf	DR	-
2	path4f	stf	DR	-

Report Patterns

```
Pattern 0 (fast_sequential)
Time 0: force_all_pis =   01000
Time 1: force_all_pis =   10101
Time 2: measure_all_pos = 0
Pattern 1 (fast_sequential)
Time 0: force_all_pis =   10110
Time 1: force_all_pis =   00000
Time 2: measure_all_pos = 1
Pattern 2 (fast_sequential)
Time 0: force_all_pis =   11111
Time 1: force_all_pis =   01111
Time 2: measure_all_pos = 1
```

Figure 6 Non-Robust Test Patterns

For figure 8.20, Figure 7 shows the schematic generated from VHDL. Its correctness is shown in the waveform in Figure 8.

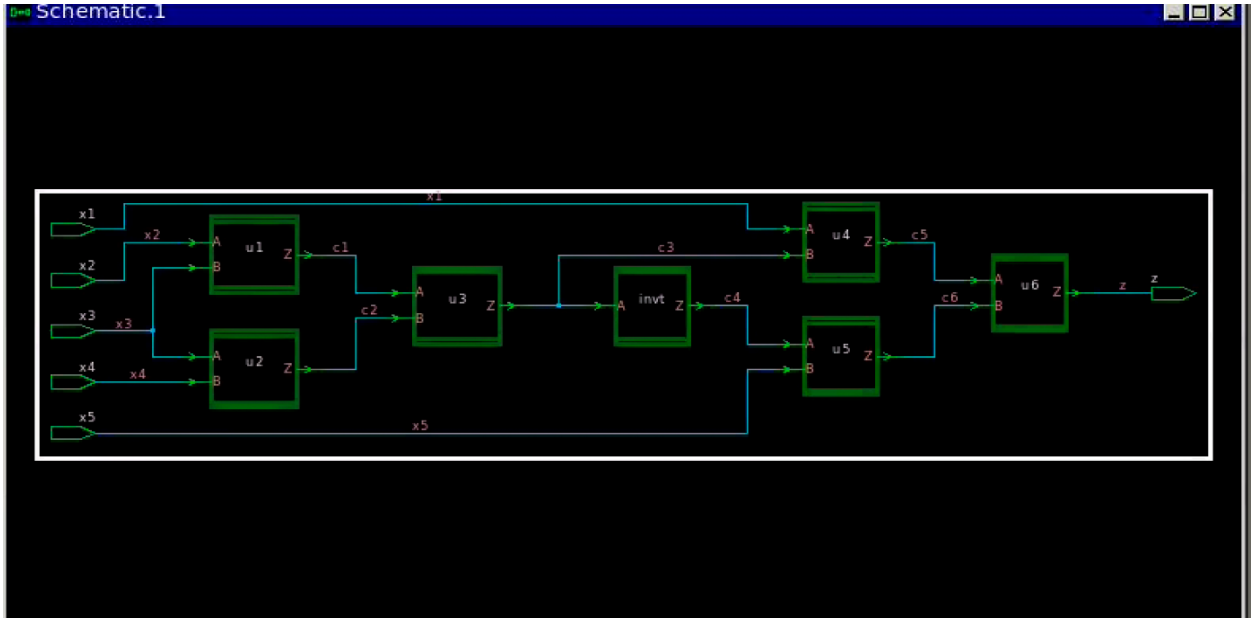


Figure 7 Schematic of Figure 8.20

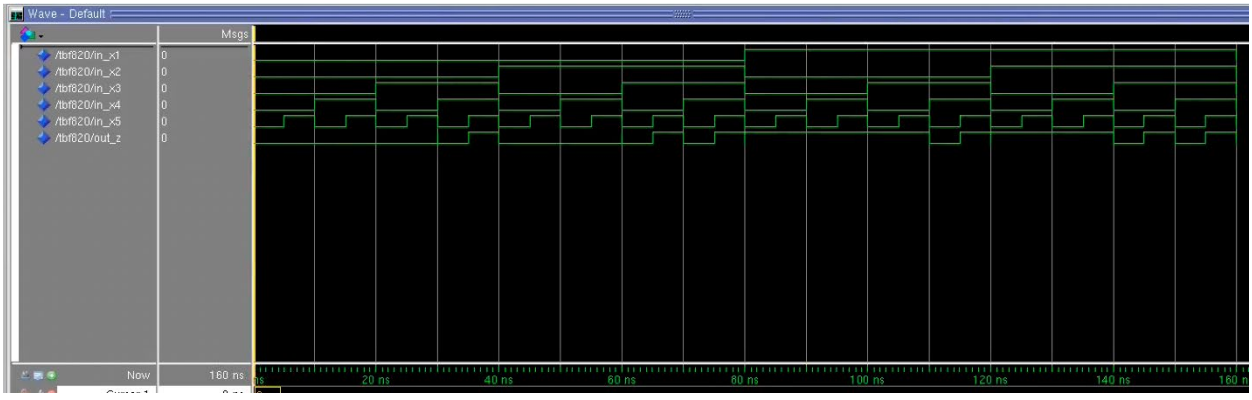


Figure 8 Validation Waveform of Figure 8.20

The delay path from TetraMax is presented in Figure 9 and Figure 10.

Figure 9 Delay path for i) and ii)

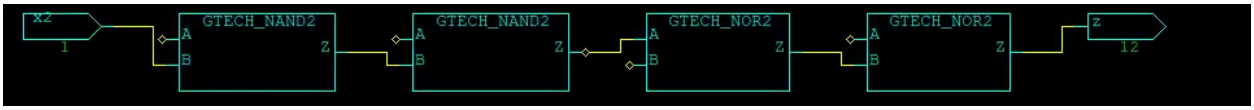


Figure 10 Delay path for iii) and iv)

The result obtained from the program may be slightly different from that of by hand due to different choice of path.

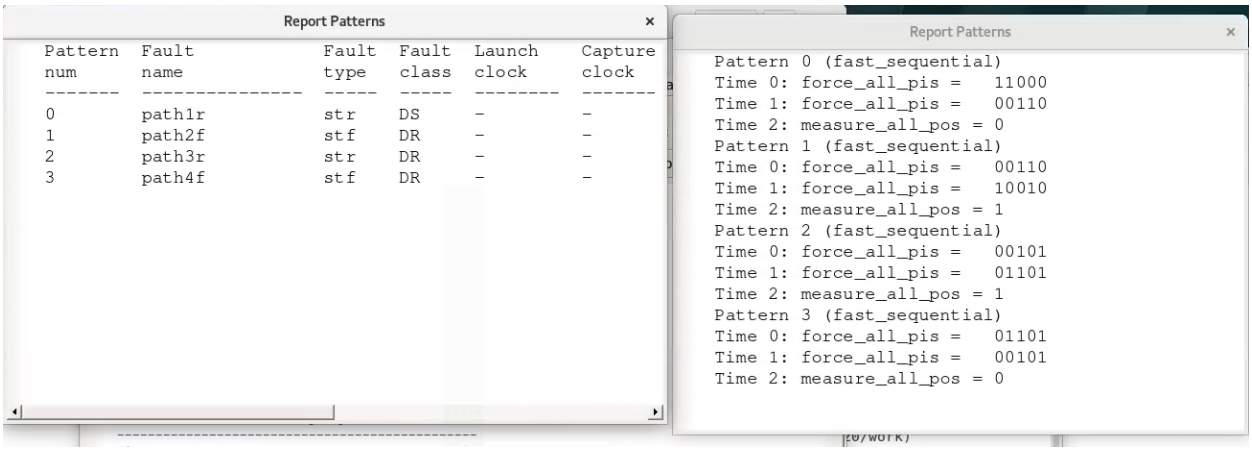


Figure 11 Robust Test of Q2

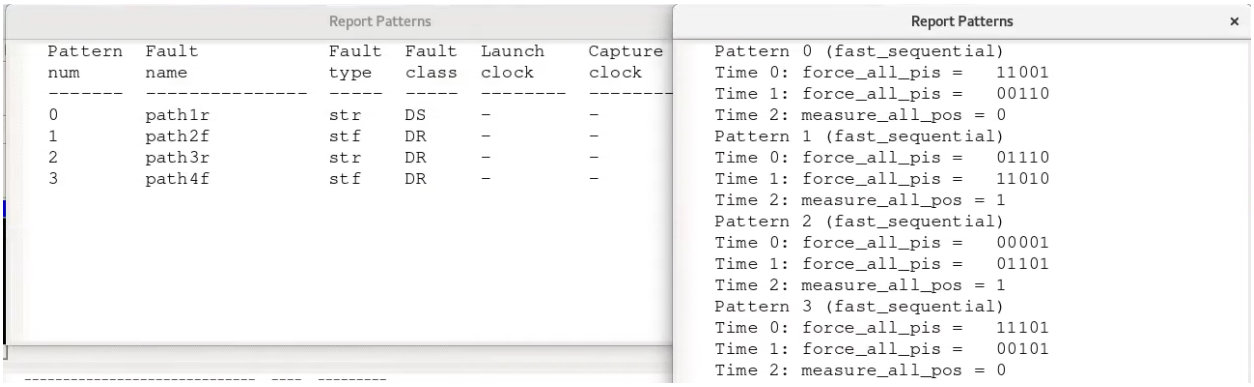


Figure 12 non-Robust Test of Q2

4. From your text, solve the following problems:
– Chapter 14: 4, 5, 6, 7 and 8.

14.4 Suppose we are given a new fault model: disturb coupling fault (CFds), defined as follows:
1 Write operations can sensitize faults (regardless of whether they are transition or non-transition write operations).
2 Read operations also can sensitize faults.
The notation for this fault model is the following: $\langle w0; \downarrow\rangle$, $\langle w0; \uparrow\rangle$, $\langle w1; \downarrow\rangle$, $\langle w1; \uparrow\rangle$, $\langle r0; \downarrow\rangle$, $\langle r0; \uparrow\rangle$, $\langle r1; \downarrow\rangle$, $\langle r1; \uparrow\rangle$.
Prove that March C− detects all unlinked CFdss.

14.5 Assume a memory with B-bit words ($B \geq 2$). Modify the MATS+ algorithm such that it detects all AFs and SAFs in a memory with B-bit words. Give an example of a MATS+ algorithm for a memory with $B = 4$.

14.6 Suppose we are given a first-in first-out (FIFO) memory. FIFOs have separate write and read ports. Each port has its own address register which automatically increments upon completion of a read (write) operation; i.e., the read address (RA) is incremented upon the completion of a read operation, and the write address (WA) upon completion of a write operation. A reset operation resets both RA and WA to 0. What are the restrictions for march tests when applied to FIFOs?

14.7 Someone has designed the following test, called the write-address test (WAT), for a memory consisting of 1024 8-bit words:

For I = 0 to 1023 do A[I] := (I mod 256);
This means that address 0, A[0], gets the value 0 (i.e., A[0]:=0), A[1]:=1, . . . , A[255]:=255, A[256]:=0, A[257]:=1, etc.
For I = 0 to 1023 do { Read A[I]; A[I]:= 255−(I mod 256) };
i.e., Read A[I] and Write the complement of (I mod 256).
For I = 0 to 1023 do Read A[I]
Verify whether this test detects all unlinked AFs, SAFs, TFs, and CFsts. *Hint:* In order to get a quick feel for the fault coverage, assume initially that the memory consists of 1-bit words (i.e., $B = 1$).

14.8 Design a minimal march test which detects the following faults:
(a) Linked CFids of the form: $\langle\uparrow; \downarrow\rangle a_1 v \# \langle\uparrow; \uparrow\rangle a_2 v$.
Note: This fault consists of two CFids, for which $a_1 < v$ and $a_2 < v$; $a_1 < v$ means that the address of a-cell a_1 is lower than the address of the v-cell.
(b) Linked CFids of the form $\langle\uparrow; \downarrow\rangle \# \langle\uparrow; \uparrow\rangle$. The fault consists of two CFids; the a-cells may take on any position relative to the v-cell.

14.4 March C-

$\{M0:\Updownarrow(w0); M1:\Uparrow(r0,w1); M2:\Uparrow(r1,w0); M3:\Downarrow(r0,w1); M4:\Downarrow(r1,w0); M5:\Updownarrow(r0)\}$

Faults	Condition	Sensitizing	Detect
$\langle w0;\downarrow\rangle$		M_2 w_0	M_2, r_1
$\langle w0;\uparrow\rangle$		M_0 w_0	M_1 r_0
$\langle w1;\downarrow\rangle$		M_1 w_1	M_2 r_1
$\langle w1;\uparrow\rangle$		M_3 w_1	M_3 r_0
$\langle r0;\downarrow\rangle$		M_3 r_0	M_4 r_1
$\langle r0;\uparrow\rangle$		M_3 r_0	M_3 r_0
$\langle r1;\downarrow\rangle$		M_4 r_1	M_4 r_1
$\langle r1;\uparrow\rangle$		M_4 r_1	M_5 r_0

14.5

MATS+ algorithm: {M0:⇕(w0); M1:⇑(r0,w1); M2:⇓(r1,w0)}

B=4

M0: ⇕ (w0000); M1: ⇑ (r0000, w1111); M2: ⇓ (r1111, w0000)

14.6

Restriction includes synchronization of both RA and WA at each M element of march test.

Ensure increment of RA / WA at the same time

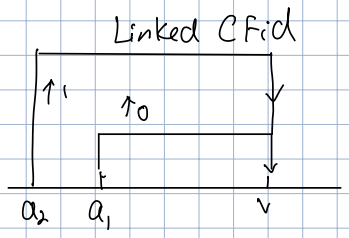
Ensure Reset is done to both signals

RA and WA are accessing the same FIFO memory slot.

14.7

14.8

a) M0: ⇕ (w0)
M1: ⇓ (r0, w1)
M2: ⇓ (r0)

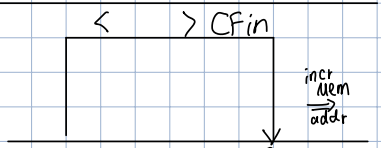
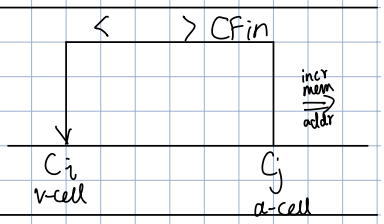
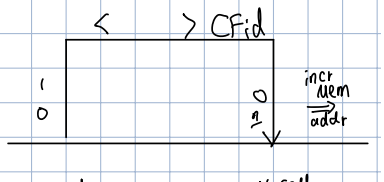
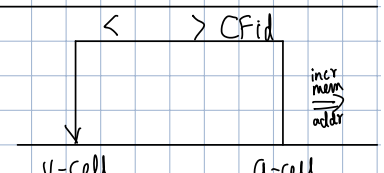


5. Determine if: (i) AFs, (ii) SAFs, (iii) TFs, (iv) unlinked CFs (CFin, CFid, CSst), and (v) linked CFids faults can be detected by MARCH C– test shown below.

$$\{M0:\Updownarrow(w0);M1:\Uparrow(r0,w1);M2:\Uparrow(r1,w0);M3:\Downarrow(r0,w1);M4:\Downarrow(r1,w0);M5:\Updownarrow(r0)\}$$

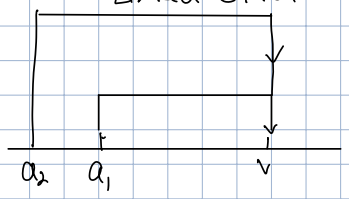
As you know, in some categories (e.g. coupling faults), there may be multiple faults. You need to justify/prove your answer for each fault. To do this for each case, carefully tabulate which March element(s) (M_i) stimulates it and which element(s) detects it. If there are multiple scenarios to detect a particular fault, list them all.

Cond. AF: The march test has to contain the following march elements
1. ⇑(r*x*,...,w*x**) This means either ⇑(r0,...,w1) or ⇑(r1,...,w0)
2. ⇓(r*x**,...,w*x*) Note: '...' means any # of *r* or *w* operations

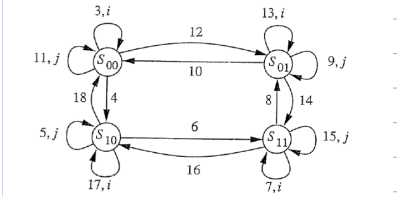
Faults	condition	sensitizing	Detection	comments
AFs				M_1+M_4 together satisfy Condition for detecting AFs. M_1 ⇒ AF① M_4 ⇒ AF②
SAF <∀/0>		M_1 writes 1 attempted from low addr to high addr	M_2 reads and expect 1 (if sa0, 0 is read) error reported	
SAF <∀/1>		M_0 writes 0 attempted	M_1 reads and expect 0 (if sa1, 1 is read, error reported)	
TF <⇑/0>		M_1 provides transition from 0→1	M_2 reads and expect 1 if read 0, error.	
TF <⇓/0>		M_2 provides transition from 1↓0	M_3 reads and expect 0 if read 1, error.	
CFin <⇑;⇕> (j<i)	a-cell (Cj) < v-cell (Ci)	M_1 w1; if fault exists, toggles a value @ higher Mem addr as it marches ⇑	M_1 when try to read 0 as marching ⇑, will detect erroneous 1 set by attempting to write to lower mem value	
CFin <⇓;⇕> (j<i)	a-cell (Cj) < v-cell (Ci)	M_2 w0 toggles a value @ high Mem addr as it march ⇑	M_2 when try to read 1 as marching ⇑, detect erroneous 0 set by early	
CFin <⇑;⇕> (j>i)	a-cell (Cj) > v-cell (Ci)	M_3 w1 toggles a value @ lower mem as it ⇓	M_3 when r0 as marching ⇓ will detect erroneous 1's	
CFin <⇓;⇕> (j>i)	a-cell (Cj) > v-cell (Ci)	M_4 w0 toggles a value @ lower mem as it ⇓	M_4 r_1 as marching ⇓ will detect erroneous 0's	
CFid <⇑;0>	a-cell < v-cell	M_3 writes 1 and, if fault, changes a value @ higher mem to 0 as marching ⇓	M_4 reads, expect 1 but detect erroneous 0 marching ⇓	
CFid <⇑;1>	a-cell < v-cell	M_1 writes 1 and changes a val @ high Mem to 1 as marching ⇑	M_1 reads, expect 0 but (unchanged) detect erroneous 1	
CFid <⇓;0>	a- cell < v-cell	M_2 writes 0 and changes a val @ higher mem to 0 as marching ⇑	M_2 reads, expect 1 but (unchanged) detect 0	
CFid <⇓;1>	a- cell < v-cell	M_4 writes 0 and changes a value @ higher mem to 1 as marching ⇓	M_5 reads expect 0 but detect 1.	
CFid <⇑;0>	a-cell > v-cell	M_1 writes 1 and changes a value @ lower mem to 0 as marching ⇑	M_2 reads expect 1 but detect 0.	
CFid <⇑;1>	a-cell > v-cell	M_3 writes 1 and changes a value @ lower mem to 1 as marching ⇓	M_3 reads expect 0 but detect 1.	

CFid $\langle \downarrow; 0 \rangle$	a - cell > v-cell	M_4 writes 0 and changes a value @ lower mem to 0 as marching $\downarrow$	M_4 reads expect 1 but detect 0.
CFid $\langle \downarrow; 1 \rangle$	a - cell > v-cell	M_0 writes 0 and changes a value @ lower mem to 1	M_1 reads expect 0 but detect 1.
Linked CFid $\langle \uparrow; 0 \rangle$ a_1 linked to $\langle \uparrow; 1 \rangle$ a_2	a_2-cell < a_1-cell < v-cell	M_3 writes 1 to a_1 and changes v-cell to 0, but write 1 to a_2 changes v-cell to 1, making it look normal	None

cannot detect Linked CFid



CFst



All CFst's can be detected as 4 states of 2 cells are reached

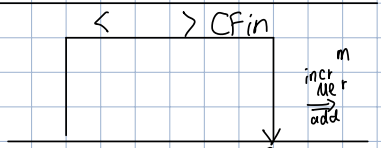
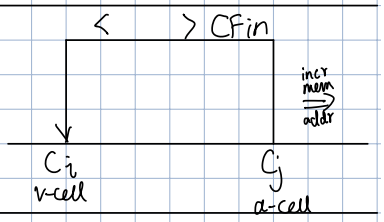
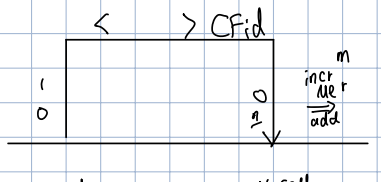
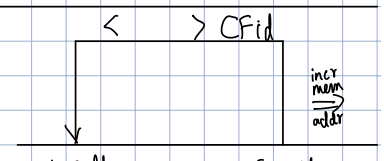
Step	March element	State before operation	operation	State after operation
1	M_0		W_0 in i	
2			W_0 in j	S_{00}
3	M_1	S_{00}	r_0 from i	S_{00}
4		S_{00}	w_1 to i	S_{10}
5		S_{10}	r_0 from j	S_{10}
6		S_{10}	w_1 to j	S_{11}
7	M_2	S_{11}	r_1 from i	S_{11}
8		S_{11}	W_0 to i	S_{01}
9		S_{01}	r_1 from j	S_{01}
10		S_{01}	W_0 to j	S_{10}
11				

6. Determine if: (i) AFs, (ii) SAFs, (iii) TFs, (iv) unlinked CFs (CFin, CFid, CSst), and (v) linked CFids faults can be detected by IFA-13 march test shown below.

$\{M0:\Uparrow(w0); M1:\Uparrow(r0,w1,r1); M2:\Uparrow(r1,w0,r0); M3:\Downarrow(r0,w1,r1);$
$M4:\Downarrow(r1,w0,r0); Delay; M5:\Uparrow(r0,w1); Delay; M6:\Uparrow(r1)\}$

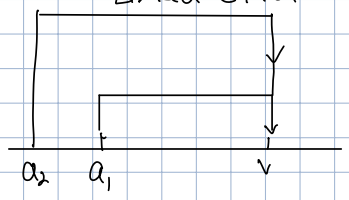
As you know, in some categories (e.g. coupling faults), there may be multiple faults. You need to justify/prove your answer for each fault. To do this for each case, carefully tabulate which March element(s) (M_i) stimulates it and which element(s) detects it. If there are multiple scenarios to detect a particular fault, list them all.

Cond. AF: The march test has to contain the following march elements
1. $\Uparrow(rx,...,wx*)$ This means either ⇑(r0,...,w1) or ⇑(r1,...,w0)
2. $\Downarrow(rx*,...,wx)$ Note: '...' means any # of *r* or *w* operations

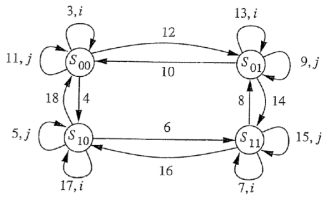
Faults	condition	sensitizing	Detection	comments
AFs				M_1+M_4 together satisfy Condition for detecting AFs. M_1 ⇒ AF① M_4 ⇒ AF②
SAF fault <∀/0>		M_1 writes 1 attempted marching ⇑	M_2 reads and expect 1 Erroneous 0 will be detected	
SAF <∀/1>		M_2 writes 0 attempted marching ⇑	M_3 reads and expect 0 (if sa1, 1 is read, error reported) marching ⇓	
TF Transition <⇑/0>		M_1 provides transition from 0 ↑ 1	M_2 reads and expect 1 if read 0, error.	
TF <⇓/0>		M_2 provides transition from 1 ↓ 0	M_2 reads and expect 0 if read 1, error.	
CFin <⇑;⇕> (j<i)	a-cell (Cj) < v-cell (Ci)	M_1 w_1, if fault exists, toggles a value @ higher Mem addr as it marches ⇑	M_1 when try to read 0 as marching, will detect erroneous 1 set by attempting to write to lower mem value	
CFin <⇓;⇕> (j<i)	a-cell (Cj) < v-cell (Ci)	M_2 w0 toggles a value @ high Mem addr as it march ⇑	M_2 when try to read 1 as marching ⇑, detect erroneous 0 set by early	
CFin <⇑;⇕> (j>i)	a-cell (Cj) > v-cell (Ci)	M_3 w1 toggles a value @ lower mem as it ⇓	M_3 when r0 as marching ⇓ will detect erroneous 1's	
CFin <⇓;⇕> (j>i)	a-cell (Cj) > v-cell (Ci)	M_4 w0 toggles a value @ lower mem as it ⇓	M_4 r_1 as marching ⇓ will detect erroneous 0's	
CFid <⇑;0>	a-cell < v-cell	M_3 writes 1 and, if fault, change v-cell @ high mem to 0 as marching ⇓	M_4 reads, expect 1 but detect erroneous 0 marching ⇓	
CFid <⇑;1>	a-cell < v-cell	M_1 writes 1 and changes a val @ high Mem to 1 as marching ⇑	M_1 reads, expect 0 but (unchanged) detect erroneous 1	
CFid <⇓;0>	a- cell < v-cell	M_2 writes 0 and changes a val @ higher mem to 0 as marching ⇑	M_2 reads, expect 1 but (unchanged) detect 0	
CFid <⇓;1>	a- cell < v-cell	M_4 writes 0 and changes a value @ higher mem to 1 as marching ⇓	M_5 reads expect 0 but detect 1.	
CFid <⇑;0>	a-cell > v-cell	M_1 writes 1 and changes a value @ lower mem to 0 as marching ⇑	M_2 reads expect 1 but detect 0.	
CFid <⇑;1>	a-cell > v-cell	M_3 writes 1 and changes a value @ lower mem to 1 as marching ⇓	M_3 reads expect 0 but detect 1.	

CFid $\langle \downarrow; 0 \rangle$	a- cell > v-cell	$M_{1\downarrow}$ writes 0 and changes a value @ lower mem to 0 as marching $\downarrow$	M_4 reads expect 1 but detect 0.
CFid $\langle \downarrow; 1 \rangle$	a- cell > v-cell	M_0 writes 0 and changes a value @ lower mem to 1 as marching $\uparrow$	M_1 reads expect 0 but detect 1.
Linked CFid $\langle \uparrow; 0 \rangle$ a_1 linked to $\langle \uparrow; 1 \rangle$ a_2	a_2-cell < a_1-cell < v-cell	M_2 writes 1 to a_1 and changes v-cell to 0, but write 1 to a_2 changes v-cell to 1, making it look normal	None

cannot detect Linked CFid



CFst



All CFst's can be detected as 4 states of 2 cells are reached

Step	March element	state before operation	operation	state after operation
1	M_0		W_0 in i	
2			W_0 in j	S_{00}
3	M_1	S_{00}	r_0 from i	S_{00}
4		S_{00}	w_1 to i	S_{10}
5		S_{10}	r_1 from i	S_{10}
6		S_{10}	r_0 from j	S_{10}
7		S_{10}	w_1 to j	S_{11}
8		S_{11}	r_1 from j	S_{11}
9	M_2	S_{11}	r_1 from i	S_{11}
10		S_{11}	W_0 to i	S_{01}
		S_{01}	r_0 from i	S_{01}
		S_{01}	r_1 from j	S_{01}
		S_{01}	W_0 to j	S_{10}
		S_{00}	r_0 from j	S_{00}

7.

All test methods up to current detect faults that are not explainable using known FFMs. As a result, new FFMs need to be created so that it can not only compete in coverage with the industrial march test, but also allow new FFMs to be detectable. Since none of the industrial march test detect all realistic faults, March SS test is introduced in the article to cover all static simple FFMs.

To define “static simple FFM”, the article brought in the concept of fault primitives, which defines the set of targeted FFMs. Under the concept, a set of targeted FFMs is described in <S/F/R>, where S defines sensitizing operation sequence, F defines faulty behaviour, and R is the resultant value. Depending on ways of sensitisation, number of operations, and influence between faults, one can class a set of FFMs into single-port or multi-port, static or dynamic, simple or linked. A set of static simple FFM can then be defined as faults sensitised by performing at most 1 operation, and each fault cannot influence each other.

As the test is designed to test on RAM, functional fault models on RAM can be divided into single-cell FFMs and two-cell FFMs, according to number of cells the fault inpacts. Faults in single-cell FFMs include state fault (SF), transition fault (TF), write disturb faults (WDF), read destructive fault (RDF), deceptive read destructive fault (DRDF), and incorrect read fault (IRF). Faults in two-cell FFMs in clude state coupling fault (CFst), disturb coupling fault (CFds), transition coupling fault (CFtr), write destructive coupling fault (CFwd), read destructive coupling fault (CFrd), deceptive read destructive coupling fault (CFdrd), and incorrect read coupling fault (CFir).

By analysing the common fault primitives (<S/F/R>) across all the faults above, March SS is developed to cover all faults mentioned above to hit multiple birds in one stone. Since other March tests do not have the common fault primitive when designed, they cannot guarantee all faults above are covered. However, while March SS provides the best coverage, its test length of 22n is the longest in the table. It will depend on the application and desired fault coverage to judge whether it’s worthy to trade off long test time with higher coverage.