



Universidad
Carlos III de Madrid
www.uc3m.es

TESIS DOCTORAL

Transient error mitigation by means of approximate logic circuits

Autor:

Antonio José Sánchez Clemente

Director y tutor:

Luis Entrena Arrontes

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

Leganés, 19 de julio de 2017

*Dedicado mi familia y amigos,
a todos aquellos que me han apoyado en este periplo
y en especial, a mi madre.*

Contents

List of acronyms	xiii
Resumen	xv
Abstract	xvii
1 Introduction	1
1.1 Outline	1
1.2 Preliminary concepts	2
1.3 Motivation of research	5
1.4 Research goals	6
1.5 Document structure	7
2 Background and previous work on approximate logic circuits	9
2.1 Introduction	9
2.2 Approximate logic fundamentals	10
2.3 Error mitigation by means of approximate logic circuits	12
2.4 Overview of approximate logic techniques	16
2.5 Conclusions	19
3 Circuit approximation method	21
3.1 Introduction	21
3.2 Unidirectional circuits	22
3.3 Unidirectional fault approximation	24
3.4 Unate expansion	26
3.5 Adaptive fault approximation	28
3.6 Conclusions	30
4 Circuit approximation using static testability measures	33
4.1 Introduction	33
4.2 Approximation of combinational circuits	34
4.3 Sequential circuits	36
4.4 Experimental results	47
4.4.1 Experimental set-up	47
4.4.2 Results on combinational circuits	50

4.4.3	Results on sequential circuits	53
4.5	Conclusions	55
5	Circuit approximation using dynamic testability measures	57
5.1	Introduction	57
5.2	Fault probability computation	60
5.3	Approximation generation with dynamic measures	62
5.3.1	Fault implication	62
5.3.1.1	Basic implication mechanism	63
5.3.1.2	Fault implication with multiple outputs	65
5.3.1.3	Inferring approximation conditions	67
5.3.1.4	Approximation conditions with multiple outputs	69
5.3.2	Probability analysis	72
5.3.2.1	Implication-based probability computation	72
5.3.2.2	Probability computation with multiple outputs	74
5.3.2.3	Incremental probability updating	75
5.3.2.4	Probability updating with multiple outputs	78
5.3.3	Estimation of total error probability	81
5.3.4	Approximation generation algorithm	84
5.4	Node substitution	86
5.5	Experimental results	92
5.5.1	Experimental set-up	93
5.5.2	Results on dynamic testability measures	95
5.5.3	Node substitution results	99
5.6	Conclusions	103
6	Applications	107
6.1	Introduction	107
6.2	Extension to FPGA-based circuits	108
6.2.1	Fault mitigation strategies in FPGAs	110
6.2.2	Fault approximation in FPGAs	111
6.2.3	Bidirectional fault approximation	114
6.2.4	Radiation experiments	118
6.2.5	Fault injection experiments	122
6.3	Microprocessor hardened by approximate TMR	126
6.3.1	Specifications	126
6.3.2	ARM Cortex M0 microprocessor	127
6.3.3	Generation of approximate microprocessor versions	129
6.3.4	Experiments	133
6.4	Comparison with evolutionary techniques	135
6.4.1	Introduction to evolutionary logic	136
6.4.2	Cartesian genetic programming	137
6.4.3	Experimental set-up	139
6.4.4	Experimental results	143
6.5	Conclusions	145

7	Conclusions and future work	147
7.1	Conclusions	147
7.2	Future research work	152
	Bibliography	154
A	Circuit approximation examples	163
A.1	Introduction	163
A.2	Example circuit	163
A.3	Approximation by static testability measures	163
A.4	Approximation by dynamic testability measures	172
	Initial probability computation	174
	1st approximation: $PI2 \rightarrow n7/1$	188
	2nd approximation: $n8 \rightarrow n9/1$	192
	3rd approximation: $n8 \rightarrow n11/1$	198
	4th approximation: $PI0/1$	201
	5th approximation: $PI1/1$	206
	6th approximation: $n7/1$	209
	7th approximation: $n11/1$	212
	8th approximation: $PI2 \rightarrow n8/1$	218
	9th approximation: $n9 \rightarrow PO1/1$	222
	10th approximation: $PI3/1$	225
A.5	Approximation with node substitution	230
	Initial substitution candidate search	231
	1st approximation: $PI2 \rightarrow n7/1$	232
	2nd approximation: $n8 \rightarrow n9/1n$	234
	3rd approximation: $n8 \rightarrow n11/1$	237
	4th approximation: $PI0/1$	239
	5th approximation: $PI1/1$	241
	6th approximation: $n7/1$	241
	7th approximation: substitution fault $PO0 \rightarrow PO1_PO/1$	244
	8th approximation: $PI2 \rightarrow n8/1$	246
	9th approximation: $n9 \rightarrow PO0/1$	249
A.6	Approximation for FPGA implementation	250

List of Figures

2.1	Example approximate logic functions	10
2.2	Example over- and under-approximations	11
2.3	Fault tolerant schemes using approximate logic circuits	12
2.4	Approximate TMR with unidirectional functions	14
3.1	Parity computation example	24
3.2	Unidirectional fault approximation examples	25
3.3	Multiple fault approximation	26
3.4	Binare fault approximation	27
3.5	Unate expansion	28
3.6	XNOR equivalent	28
3.7	Instruments for adaptive fault approximation	29
3.8	Adaptive fault approximation example	30
4.1	Implementation of voting logic	35
4.2	Approximate DWC and TMR schemes with sequential circuits	36
4.3	One-shot timer	37
4.4	Implementation of voting logic for sequential circuits	39
4.5	s27 benchmark	40
4.6	Unate version of s27 benchmark	40
4.7	Approximation of faults for a 5% threshold	41
4.8	Approximate circuits for a 5% threshold	42
4.9	Approximate circuits for a 10% threshold	43
4.10	Approximate circuits for a 15% threshold	44
4.11	Approximate circuits for a 20% threshold	44
4.12	Approximate circuits for a 10% threshold - combinational version	45
4.13	Approximate circuits for a 15% threshold - combinational version	46
4.14	Approximate circuits for a 20% threshold - combinational version	46
4.15	Experimental set-up with AMUSE	48
4.16	Scalability of technique for frg2 benchmark	52
4.17	Scalability of technique for i10 benchmark	53
4.18	Comparative between sequential and combinational approximations for s713 benchmark	55

5.1	c17 benchmark	57
5.2	c17 over-approximations	59
5.3	Implication of fault n11/1 in c17 benchmark	64
5.4	Implication of fault PI2→n8/1	65
5.5	Implication of fault PI2→n8/1 through output PO0	66
5.6	Implication of fault PI2→n8/1 through output PO1	67
5.7	Approximation condition of fault PI2→n7/1	68
5.8	Effect of PI2→n7/1 approximation over PI1/1	69
5.9	Approximation condition of fault n8→n9/1 through output PO0	70
5.10	Approximation condition of fault n8→n9/1 through output PO1	70
5.11	Approximation condition of fault PI4/1	71
5.12	Effect of PI4/1 approximation over PI1/1	72
5.13	Implication of fault n7/1	73
5.14	Implication of fault PI3/1 through output PO0	75
5.15	Implication of fault PI3/1 through output PO1	76
5.16	Approximation generation algorithm	85
5.17	Example of node substitution	87
5.18	Example of node substitution	89
5.19	Implication of substitution fault n3→n2/0	89
5.20	Karnaugh maps for output O1 before and after node substitution	90
5.21	Initial probability computation with node substitution	91
5.22	Redundancy checking step with node substitution	91
5.23	Experimental set-up with HOPE	94
5.24	Comparative of error probabilities for c17 benchmark	95
5.25	Comparative of error probabilities for frg2 benchmark	99
5.26	Comparative of approximation mechanisms for c17 benchmark	100
6.1	LUT partitioning of c17 benchmark	113
6.2	Alternative LUT partitioning of c17 benchmark	113
6.3	XNOR gate	115
6.4	Binare fault b stuck-at 1	115
6.5	Approximating the positive part of b/1	116
6.6	Approximating the negative part of b/1	116
6.7	Auto-cancellation example	116
6.8	Effect of binare fault in logic function	117
6.9	Approximation of binare fault d/1	117
6.10	Fault testability analysis of b13 benchmark	118
6.11	Results of radiation experiments	121
6.12	Fault injection results on b13 with accumulation	123
6.13	Fault injection results on b13 without accumulation	124
6.14	Critical bit analysis for b13 benchmark	125
6.15	Cortex M0 architecture [1]	128
6.16	Cortex M0 test-bench [1]	129
6.17	Implemented memory image [2]	131
6.18	Fault testability analysis of ARM Cortex-M0	132
6.19	CGP reconfigurable grid	137

6.20	CGP individual example	138
6.21	Statistical results for b12 approximations evolved by CGP	141
6.22	Statistical results for rd73 approximations evolved by CGP	141
6.23	Statistical results for t481 approximations evolved by CGP	142
6.24	b12 simulation results	143
6.25	rd73 simulation results	144
6.26	t481 simulation results	144
A.1	c17 benchmark	164
A.2	Approximate circuits for a 0 threshold - Pure TMR	165
A.3	Approximation of fault $PI2 \rightarrow n7$ stuck-at 1	166
A.4	Approximate circuits for a 0.12 threshold	166
A.5	Approximate circuits for a 0.135 threshold	167
A.6	Approximate circuits for a 0.14 threshold	168
A.7	Approximate circuits for a 0.175 threshold	169
A.8	Approximate circuits for a 0.185 threshold	170
A.9	Approximate circuits for a 0.19 threshold	171
A.10	Approximate circuits for a 0.197 threshold	172
A.11	Approximate circuits for a 0.2 threshold	173
A.12	Approximate circuits for a 0.25 threshold - Trivial approximation	173
A.13	Implication of fault $PI0/1$	174
A.14	Implication of fault $PI1/1$ through $PO0$	175
A.15	Implication of fault $PI1/1$ through $PO1$	176
A.16	Implication of fault $PI2 \rightarrow n7/1$	177
A.17	Implication of fault $PI2 \rightarrow n8/1$ through $PO0$	177
A.18	Implication of fault $PI2 \rightarrow n8/1$ through $PO1$	178
A.19	Implication of fault $PI3/1$ through $PO0$	178
A.20	Implication of fault $PI3/1$ through $PO1$	179
A.21	Implication of fault $PI4/1$	180
A.22	Implication of fault $n7/1$	180
A.23	Implication of fault $n8 \rightarrow n9/1$ through $PO0$	181
A.24	Implication of fault $n8 \rightarrow n9/1$ through $PO1$	182
A.25	Implication of fault $n8 \rightarrow n11/1$	182
A.26	Implication of fault $n9 \rightarrow PO0/1$	183
A.27	Implication of fault $n9 \rightarrow PO1/1$	183
A.28	Implication of fault $n11/1$	184
A.29	Implication of fault $PO0/0$	184
A.30	Implication of fault $PO0/1$	185
A.31	Implication of fault $PO1/0$	186
A.32	Implication of fault $PO1/1$	186
A.33	Approximation condition of fault $PI2 \rightarrow n7/1$	188
A.34	Effect of fault $PI2 \rightarrow n7/1$ over $PI1/1$	189
A.35	Effect of fault $PI2 \rightarrow n7/1$ over itself	189
A.36	Effect of fault $PI2 \rightarrow n7/1$ over $PO0/1$	190
A.37	Approximation of fault $PI2 \rightarrow n7/1$	191
A.38	Approximation condition of $n8 \rightarrow n9/1$ with respect to $PO0$	192

A.39	Approximation condition of $n8 \rightarrow n9/1$ with respect to PO1	193
A.40	Effect of fault $n8 \rightarrow n9/1$ over PI0/1	194
A.41	Effect of fault $n8 \rightarrow n9/1$ over itself	194
A.42	Effect of fault $n8 \rightarrow n9/1$ over $n8 \rightarrow n11/1$	195
A.43	Effect of fault $n8 \rightarrow n9/1$ over PO0/1	195
A.44	Effect of fault $n8 \rightarrow n9/1$ over PO1/1	196
A.45	Approximation of fault $n8 \rightarrow n9/1$	197
A.46	Approximation condition of $n8 \rightarrow n11/1$	198
A.47	Effect of fault $n8 \rightarrow n11/1$ over itself	199
A.48	Effect of fault $n8 \rightarrow n11/1$ over PO1/1	199
A.49	Approximation of fault $n8 \rightarrow n11/1$	200
A.50	Approximation condition of PI0/1	202
A.51	Effect of fault PI0/1 over itself	202
A.52	Effect of fault PI0/1 over PI1/1	202
A.53	Effect of fault PI0/1 over PO0/1	203
A.54	Approximation of fault PI0/1	204
A.55	Approximation condition of PI1/1	205
A.56	Effect of fault PI1/1 over itself	206
A.57	Effect of fault PI1/1 over PI4/1	207
A.58	Effect of fault PI1/1 over PO1/1	207
A.59	Approximation of fault PI1/1	208
A.60	Approximation condition of $n7/1$	209
A.61	Effect of fault $n7/1$ over itself	210
A.62	Effect of fault $n7/1$ over PO0/0	211
A.63	Approximation of fault $n7/1$	212
A.64	Approximation condition of $n11/1$	213
A.65	Effect of fault $n11/1$ over $PI2 \rightarrow n8/1$	214
A.66	Effect of fault $n11/1$ over PI3/1	215
A.67	Effect of fault $n11/1$ over itself	215
A.68	Effect of fault $n11/1$ over PO1/0	216
A.69	Approximation of fault $n11/1$	217
A.70	Approximation condition of $PI2 \rightarrow n8/1$	218
A.71	Effect of fault $PI2 \rightarrow n8/1$ over itself	218
A.72	Effect of fault $PI2 \rightarrow n8/1$ over $n9 \rightarrow PO0/1$	219
A.73	Effect of fault $PI2 \rightarrow n8/1$ over $n9 \rightarrow PO1/1$	220
A.74	Effect of fault $PI2 \rightarrow n8/1$ over PO0/0	220
A.75	Effect of fault $PI2 \rightarrow n8/1$ over PO1/0	220
A.76	Approximation of fault $PI2 \rightarrow n8/1$	221
A.77	Approximation condition of $n9 \rightarrow PO1/1$	222
A.78	Effect of fault $n9 \rightarrow PO1/1$ over PI3/1	223
A.79	Effect of fault $n9 \rightarrow PO1/1$ over itself	223
A.80	Effect of fault $n9 \rightarrow PO1/1$ over PO1/0	224
A.81	Approximation of fault $n9 \rightarrow PO1/1$	225
A.82	Approximation condition of PI3/1	226
A.83	J-SMA of fault PI3/1 with respect to PO0	226
A.84	Effect of fault PI3/1 over $n9 \rightarrow PO0/1$	227

A.85	Effect of fault PI3/1 over PO0/0	227
A.86	Approximation of fault PI3/1	228
A.87	Implication of substitution candidate PI1→PO1/1	232
A.88	Implication of fault n7/1 in the over-approximate circuit - first approximation	233
A.89	Implication of fault n7/1 in the over-approximate circuit - second approximation	235
A.90	Implication of substitution candidate PI1→PO0/1 - second approximation	235
A.91	Implication of fault n11/1 in the over-approximate circuit - second approximation	236
A.92	Implication of substitution candidate PI1→PO1/1 - second approximation	236
A.93	Implication of fault n11/1 in the over-approximate circuit - third approximation	238
A.94	Implication of substitution candidate PI1→PO1/1 - third approximation	238
A.95	Implication of fault n9→PO1/1 in the over-approximate circuit - fourth approximation	240
A.96	Implication of substitution candidate PI4→PO1/1 - fourth approximation	240
A.97	Implication of fault PO0/1 in the under-approximate circuit - sixth approximation	242
A.98	Implication of fault PO1/1 in the under-approximate circuit - sixth approximation	243
A.99	Implication of substitution candidate PO0→PO1_PO/1 - sixth approximation	243
A.100	Approximation of substitution fault PO0→PO1_PO/1	245
A.101	Implication of fault n8→n9/1 in the under-approximate circuit - seventh approximation	246
A.102	Approximation of fault PI2→n8/1	248
A.103	Implication of fault PO0/1 in the under-approximate circuit - eighth approximation	248
A.104	LUT partitioning of c17 benchmark	251
A.105	Approximate circuits for a 0 threshold - Pure TMR	252
A.106	Approximate circuits for a 0.12 threshold	252
A.107	Approximate circuits for a 0.135 threshold	253
A.108	Approximate circuits for a 0.14 threshold	254
A.109	Approximate circuits for a 0.185 threshold	254
A.110	Approximate circuits for a 0.19 threshold	255
A.111	Approximate circuits for a 0.197 threshold	256
A.112	Approximate circuits for a 0.2 threshold	257
A.113	Approximate circuits for a 0.25 threshold	258
A.114	Approximate circuits for a 0.6 threshold	259

List of Tables

2.1	Summary of the ATMR operation	15
4.1	Results of fault testability analysis for s27	41
4.2	Results of fault testability analysis for combinational part of s27	45
4.3	Experimental results - static testability measures with combinational circuits	51
4.4	Experimental results - static testability measures with sequential circuits	55
5.1	Results of fault testability analysis for c17	58
5.2	Probability computation with COP	61
5.3	Experimental results with dynamic testability measures	98
5.4	Experimental results with node substitution	102
5.5	Comparison between fault selection heuristics	103
6.1	Synthesis results for b13 benchmark	119
6.2	Beam characteristics	120
6.3	Software benchmarks data	130
6.4	Synthesis results	133
6.5	Results of ATMR fault emulation with AMUSE	134
6.6	Synthesis results for selected benchmarks	140
6.7	CGP parameters	140
6.8	Error masking rate versus Hamming distance correlation indexes	142
A.1	Results of fault testability analysis for c17	164
A.2	Summary of initial probability computation	187
A.3	Summary of $PI2 \rightarrow n7/1$ approximation	190
A.4	Fault probabilities after $PI2 \rightarrow n7/1$ approximation	192
A.5	Summary of $n8 \rightarrow n9/1$ approximation	196
A.6	Fault probabilities after $n8 \rightarrow n9/1$ approximation	198
A.7	Summary of $n8 \rightarrow n11/1$ approximation	200
A.8	Fault probabilities after $n8 \rightarrow n11/1$ approximation	201
A.9	Summary of $PI0/1$ approximation	203
A.10	Fault probabilities after $PI0/1$ approximation	205
A.11	Summary of $PI1/1$ approximation	208

A.12	Fault probabilities after PI1/1 approximation	209
A.13	Summary of n7/1 approximation	211
A.14	Fault probabilities after n7/1 approximation	213
A.15	Summary of n11/1 approximation	216
A.16	Fault probabilities after n11/1 approximation	217
A.17	Summary of PI2→n8/1 approximation	221
A.18	Fault probabilities after PI2→n8/1 approximation	222
A.19	Summary of n9→PO1/1 approximation	224
A.20	Fault probabilities after n11/1 approximation	225
A.21	Summary of PI3/1 approximation	228
A.22	Real contribution of each fault to EP	229
A.23	Fault probabilities after PO0→PO1_PO/1 approximation	246
A.24	Summary of PI2→n8/1 approximation	247
A.25	Fault probabilities after PI2→n8/1 approximation	249
A.26	Summary of n9→PO0/1 approximation	249

List of acronyms

ADWC	Approximate DWC
AHB	Advanced High-performance Bus
AIG	And-Inverter Graph
AMBA	Advanced Microcontroller Bus Architecture
AMUSE	Autonomous Multilevel emulation system for Soft-error Evaluation
ASIC	Application Specific Integrated Circuit
ATMR	Approximate TMR
ATPG	Automatic Test Pattern Generation
BDD	Binary Decision Diagram
CGP	Cartesian Genetic Programming
CMOS	Complimentary Metal-Oxide-Semiconductor
CNA	Centro Nacional de Aceleradores
COP	Computation of Probabilities
CRC	Cyclic Redundancy Check
CUT	Circuit Under Test
DICE	Dual Interlocked Storage Cell
DWC	Duplication with Comparison
ECC	Error-Correcting Code
EDAC	Error Detection And Correction
EP	Total Error Probability
FPGA	Field Programmable Gate Array

GP	Genetic Programming
IC	Integrated Circuit
LFSR	Linear Feedback Shift Register
J-SMA	Justified Set of Mandatory Assignments
LUT	Look-Up Table
MA	Mandatory Assignment
MBU	Multiple Bit Upset
MIFTF	Mean Injected Faults To Failure
MTTF	Mean Time To Failure
RAM	Random Access Memory
RO	Read-Only
RTL	Register Transfer Level
RW	Read/Write
SAT	Boolean Satisfiability Problem
SEE	Single Event Effect
SEFI	Single Event Functional Interrupt
SEL	Single Event Latch-up
SEM	Soft Error Mitigation
SET	Single Event Transient
SEU	Single Event Upset
SHE	Single event Hard Error
SMA	Set of Mandatory Assignments
SOP	Sum of Products
SRAM	Static RAM
TAIR	Testability Analysis by Implication Reasoning
TID	Total Ionizing Dose
TMR	Triple Modular Redundancy
ZI	Zero-Initialized

Resumen

Los avances tecnológicos en la fabricación de circuitos electrónicos han permitido mejorar en gran medida sus prestaciones, pero también han incrementado la sensibilidad de los mismos a los errores provocados por la radiación. Entre ellos, los más comunes son los SEEs, perturbaciones eléctricas causadas por el impacto de partículas de alta energía, que entre otros efectos pueden modificar el estado de los elementos de memoria (SEU) o generar pulsos transitorios de valor erróneo (SET). Estos eventos suponen un riesgo para la fiabilidad de los circuitos electrónicos, por lo que deben ser tratados mediante técnicas de tolerancia a fallos.

Las técnicas de tolerancia a fallos más comunes se basan en la replicación completa del circuito (DWC o TMR). Estas técnicas son capaces de cubrir una amplia variedad de modos de fallo presentes en los circuitos electrónicos. Sin embargo, presentan un elevado sobrecoste en área y consumo. Por ello, a menudo se buscan alternativas más ligeras, aunque no tan efectivas, basadas en una replicación parcial. En este contexto surge una nueva filosofía de diseño electrónico, conocida como computación aproximada, basada en mejorar las prestaciones de un diseño a cambio de ligeras modificaciones de la funcionalidad prevista. Es un enfoque atractivo y poco explorado para el diseño de soluciones ligeras de tolerancia a fallos.

El objetivo de esta tesis consiste en desarrollar nuevas técnicas ligeras de tolerancia a fallos por replicación parcial, mediante el uso de circuitos lógicos aproximados. Estos circuitos se pueden diseñar con una gran flexibilidad. De este forma, tanto el nivel de protección como el sobrecoste se pueden regular libremente en función de los requisitos de cada aplicación. Sin embargo, encontrar los circuitos aproximados óptimos para cada aplicación es actualmente un reto.

En la presente tesis se propone un método para generar circuitos aproximados, denominado aproximación de fallos, consistente en asignar constantes lógicas a ciertas líneas del circuito. Por otro lado, se desarrollan varios criterios de selección para, mediante este mecanismo, generar los circuitos aproximados más adecuados para cada aplicación. Estos criterios se basan en la idea de aproximar las secciones menos testables del circuito, lo que permite reducir los sobrecostes minimizando la pérdida de fiabilidad. Por tanto, en esta tesis la selección de aproximaciones se realiza a partir de medidas de testabilidad.

El primer criterio de selección de fallos desarrollado en la presente tesis hace uso de medidas de testabilidad estáticas. Las aproximaciones se generan a partir de los resultados de una simulación de fallos del circuito objetivo, y de un umbral de testabilidad especificado por el usuario. La cantidad de fallos aproximados depende del umbral

escogido, lo que permite generar circuitos aproximados con diferentes prestaciones. Aunque inicialmente este método ha sido concebido para circuitos combinatoriales, también se ha realizado una extensión a circuitos secuenciales, considerando los biestables como entradas y salidas de la parte combinatorial del circuito. Los resultados experimentales demuestran que esta técnica consigue una buena escalabilidad, y unas prestaciones de coste frente a fiabilidad aceptables. Además, tiene un coste computacional muy bajo.

Sin embargo, el criterio de selección basado en medidas estáticas presenta algunos inconvenientes. No resulta intuitivo ajustar las prestaciones de los circuitos aproximados a partir de un umbral de testabilidad, y las medidas estáticas no tienen en cuenta los cambios producidos a medida que se van aproximando fallos. Por ello, se propone un criterio alternativo de selección de fallos, basado en medidas de testabilidad dinámicas. Con este criterio, la testabilidad de cada fallo se calcula mediante un análisis de probabilidades basado en implicaciones. Las probabilidades se actualizan con cada nuevo fallo aproximado, de forma que en cada iteración se elige la aproximación más favorable, es decir, el fallo con menor probabilidad. Además, las probabilidades calculadas permiten estimar la protección frente a fallos que ofrecen los circuitos aproximados generados, por lo que es posible generar circuitos que se ajusten a una tasa de fallos objetivo. Modificando esta tasa se obtienen circuitos aproximados con diferentes prestaciones. Los resultados experimentales muestran que este método es capaz de ajustarse razonablemente bien a la tasa de fallos objetivo. Además, los circuitos generados con esta técnica muestran mejores prestaciones que con el método basado en medidas estáticas. También se han aprovechado las implicaciones de fallos para implementar un nuevo tipo de transformación lógica, consistente en sustituir nodos funcionalmente similares.

Una vez desarrollados los criterios de selección de fallos, se aplican a distintos campos. En primer lugar, se hace una extensión de las técnicas propuestas para FPGAs, teniendo en cuenta las particularidades de este tipo de circuitos. Esta técnica se ha validado mediante experimentos de radiación, los cuales demuestran que una replicación parcial con circuitos aproximados puede ser incluso más robusta que una replicación completa, ya que un área más pequeña reduce la probabilidad de SEEs. Por otro lado, también se han aplicado las técnicas propuestas en esta tesis a un circuito de aplicación real, el microprocesador ARM Cortex M0, utilizando un conjunto de benchmarks software para generar las medidas de testabilidad necesarias. Por último, se realiza un estudio comparativo de las técnicas desarrolladas con la generación de circuitos aproximados mediante técnicas evolutivas. Estas técnicas hacen uso de una gran capacidad de cálculo para generar múltiples circuitos mediante ensayo y error, reduciendo la posibilidad de caer en algún mínimo local. Los resultados confirman que, en efecto, los circuitos generados mediante técnicas evolutivas son ligeramente mejores en prestaciones que con las técnicas aquí propuestas, pero con un coste computacional mucho mayor.

En definitiva, se proponen varias técnicas originales de mitigación de fallos mediante circuitos aproximados. Se demuestra que estas técnicas tienen diversas aplicaciones, haciendo de la flexibilidad y adaptabilidad a los requisitos de cada aplicación sus principales virtudes.

Abstract

The technological advances in the manufacturing of electronic circuits have allowed to greatly improve their performance, but they have also increased the sensitivity of electronic devices to radiation-induced errors. Among them, the most common effects are the SEEs, i.e., electrical perturbations provoked by the strike of high-energy particles, which may modify the internal state of a memory element (SEU) or generate erroneous transient pulses (SET), among other effects. These events pose a threat for the reliability of electronic circuits, and therefore fault-tolerance techniques must be applied to deal with them.

The most common fault-tolerance techniques are based in full replication (DWC or TMR). These techniques are able to cover a wide range of failure mechanisms present in electronic circuits. However, they suffer from high overheads in terms of area and power consumption. For this reason, lighter alternatives are often sought at the expense of slightly reducing reliability for the least critical circuit sections. In this context a new paradigm of electronic design is emerging, known as approximate computing, which is based on improving the circuit performance in change of slight modifications of the intended functionality. This is an interesting approach for the design of lightweight fault-tolerant solutions, which has not been yet studied in depth.

The main goal of this thesis consists in developing new lightweight fault-tolerant techniques with partial replication, by means of approximate logic circuits. These circuits can be designed with great flexibility. This way, the level of protection as well as the overheads can be adjusted at will depending on the necessities of each application. However, finding optimal approximate circuits for a given application is still a challenge.

In this thesis a method for approximate circuit generation is proposed, denoted as fault approximation, which consists in assigning constant logic values to specific circuit lines. On the other hand, several criteria are developed to generate the most suitable approximate circuits for each application, by using this fault approximation mechanism. These criteria are based on the idea of approximating the least testable sections of circuits, which allows reducing overheads while minimising the loss of reliability. Therefore, in this thesis the selection of approximations is linked to testability measures.

The first criterion for fault selection developed in this thesis uses static testability measures. The approximations are generated from the results of a fault simulation of the target circuit, and from a user-specified testability threshold. The amount of approximated faults depends on the chosen threshold, which allows to generate approximate

circuits with different performances. Although this approach was initially intended for combinational circuits, an extension to sequential circuits has been performed as well, by considering the flip-flops as both inputs and outputs of the combinational part of the circuit. The experimental results show that this technique achieves a wide scalability, and an acceptable trade-off between reliability versus overheads. In addition, its computational complexity is very low.

However, the selection criterion based in static testability measures has some drawbacks. Adjusting the performance of the generated approximate circuits by means of the approximation threshold is not intuitive, and the static testability measures do not take into account the changes as long as faults are approximated. Therefore, an alternative criterion is proposed, which is based on dynamic testability measures. With this criterion, the testability of each fault is computed by means of an implication-based probability analysis. The probabilities are updated with each new approximated fault, in such a way that on each iteration the most beneficial approximation is chosen, that is, the fault with the lowest probability. In addition, the computed probabilities allow to estimate the level of protection against faults that the generated approximate circuits provide. Therefore, it is possible to generate circuits which stick to a target error rate. By modifying this target, circuits with different performances can be obtained. The experimental results show that this new approach is able to stick to the target error rate with reasonably good precision. In addition, the approximate circuits generated with this technique show better performance than with the approach based in static testability measures. In addition, the fault implications have been reused too in order to implement a new type of logic transformation, which consists in substituting functionally similar nodes.

Once the fault selection criteria have been developed, they are applied to different scenarios. First, an extension of the proposed techniques to FPGAs is performed, taking into account the particularities of this kind of circuits. This approach has been validated by means of radiation experiments, which show that a partial replication with approximate circuits can be even more robust than a full replication approach, because a smaller area reduces the probability of SEE occurrence. Besides, the proposed techniques have been applied to a real application circuit as well, in particular to the microprocessor ARM Cortex M0. A set of software benchmarks is used to generate the required testability measures. Finally, a comparative study of the proposed approaches with approximate circuit generation by means of evolutive techniques have been performed. These approaches make use of a high computational capacity to generate multiple circuits by trial-and-error, thus reducing the possibility of falling into local minima. The experimental results demonstrate that the circuits generated with evolutive approaches are slightly better in performance than the circuits generated with the techniques here proposed, although with a much higher computational effort.

In summary, several original fault mitigation techniques with approximate logic circuits are proposed. These approaches are demonstrated in various scenarios, showing that the scalability and adaptability to the requirements of each application are their main virtues.

Chapter 1

Introduction

1.1 Outline

The advances on the manufacturing technology of electronic circuits over the years have allowed to improve performance over time. With the shrinking of transistor sizes, electronic devices have gained in integration density, operation frequency and reduction of power consumption. This fact has helped to greatly extend the application field of electronic devices up to levels not seen before. But at the same time, this reduction of transistor sizes is responsible for the growing sensitivity of electronic devices to transient, intermittent and permanent errors [3].

These errors may come from different sources, such as the manufacturing process itself, environmental conditions, and ageing effects. Specially relevant is the kind of radiation-induced error denoted as *Single Event Effect* (SEE). Basically, a SEE is an electric perturbation caused by the strike of an individual high-energy particle. Such perturbation may manifest in multiple ways, being the most common ones the modification of the state of a memory element (known as *Single Event Upset* (SEU)) or the generation of an erroneous transient pulse which may propagate through the circuit (known as *Single Event Transient* (SET)).

The presence of errors represents a risk for the reliability of electronic circuits. For this reason, it is a common practice to implement some mechanisms for either detecting the presence of errors, or minimising their effects when they occur. The first step consist in the manufacturing test, which tries to determine the existence of errors which might be produced during the manufacturing process. Nevertheless, this is not sufficient as errors may be produced during the whole lifetime of the circuit (for example, due to SEEs), and therefore some on-line error mitigation mechanisms should be considered from the design of robust electronic circuits.

Fault-tolerant design has been an active area of research for decades. Many error mitigation techniques have been developed, covering different fault mechanisms and abstraction levels (from transistor level to complete microprocessor systems), and with different cost requirements. Among them, the most popular approaches consist in the full replication of the circuit to protect, combined with logic for either detecting faults

(by comparing the results of two identical instances of the target circuit) or correcting them (by voting the results of three copies, and choosing the majority value). These approaches are known respectively as *Duplication with Comparison* (DWC) and *Triple Modular Redundancy* (TMR), and they have the advantage of covering a wide variety of failure mechanisms which are present in electronic circuits, specially in the modern technologies. However, these techniques suffer from high overhead in terms of area and power consumption (more than 100% and 200% respectively). To alleviate these overheads, alternative techniques have been proposed based on partial fault detection or masking.

In relation with this interest in developing low overhead systems, a branch of electronic circuit design has emerged which is based in the idea of improving the costs or the timing response of a given design by means of slight deviations of the intended functionality. This paradigm is known as *approximate computing*. This strategy has the particularity of potentially generating better implementations with respect to area, power consumption or delay than with conventional synthesis approaches, although this approach is suited mainly for applications which can tolerate some degree of misbehaviour. Applied to the fault-tolerance field, approximate logic may allow to develop flexible and low cost solutions for partial error mitigation, which is the main goal of this thesis.

This introductory chapter is structured as follows. First, section 1.2 provides an introduction about the main sources of errors in electronic circuits, with an emphasis in those causes which are more relevant with respect to this thesis. Then, section 1.3 shows the motivations of the research conducted in this thesis. Later, section 1.4 summarizes the research goals proposed for this thesis. Finally, section 1.5 closes this chapter with a brief summary of the contents of this document, chapter by chapter.

1.2 Preliminary concepts

Errors in electronic circuits are usually classified as *soft errors* and *hard errors*. The first type are errors which originate a temporal misbehaviour without causing any physical harm to the circuit, such as a bit flip in one of the bits in a register, while the latter type include those errors which cause a permanent damage to the physical circuit, for example a burnout in a line or the creation of a parasitic transistor. A special case of errors are *intermittent errors*, a kind of error which appears from time to time. An intermittent error may be caused by either a permanent damage on the circuit which is only excited on particular conditions, or by a soft error which persist over time and therefore may periodically show up. All these errors may come from different sources, including manufacturing process variations, environmental conditions and ageing. Some of this error sources are presented here in more detail.

First, manufacturing process variations are a dominant source of static variability that may significantly affect yield. The results of such variations are defective circuits which deviate from the original intended functionality, and therefore they may present an erroneous response under certain stimuli. In the past, variations were mostly due to imperfect process control, but now intrinsic atomistic effects, such as random dopant fluctuations or line edge roughness have become relevant in technologies under 45

nanometers, as devices of atomic sizes are achieved [4]. Manufacturing tests are intended to detect such variations in advance to normal operation, although due to the increasing difficulty of testing, some defects may escape the manufacturing test and may cause intermittent failures resulting in errors during normal operation.

Environmental conditions may induce errors during the lifetime of circuits. In particular, electronic circuits are sensitive to radiation effects, which may cause errors of various degrees of criticality. There are some environments which are specially exposed to radiation, such as the outer space, but it is present everywhere, including the Earth surface. And although there are some measures which can be implemented in order to shield the circuits against these radiation-induced effects, it is extremely costly to completely get rid of them. In practice, every circuit is exposed to suffer from errors caused by radiation.

Finally, ageing consist in the progressive degradation of the electrical characteristics of electronic circuits with their use over time. This would eventually make the circuit to fail, a process which may be accelerated by some environmental conditions, such as supply voltage or temperature variations, extreme temperatures or high radiation environments.

Let us focus on the radiation effects. Here the radiation is understood as a continuous flow of highly energetic particles, which collide against the electronic devices. On each one of these strikes, the colliding particle may deposit some electric charge inside the semiconductor components of the electronic circuit, affecting its performance in different ways. As said before, the radiation is present everywhere, although there are some environments which are exposed to the radiation with higher intensity than others.

With respect to the nature of radiation-induced errors, there are two different mechanisms. On the one hand, a single particle strike may cause an electric perturbation high enough to immediately generate either a soft or a intermittent error, or even a hard error. Generally speaking, this phenomenon is denoted as SEE. Depending on the circuit elements affected by the strike and its effects, SEEs are classified into several types. If a memory element is affected, causing a bit flip on its content, the effect is denoted as an SEU. On the contrary, if a combinational node receives the strike, a temporary erroneous electric pulse may be originated, which is denoted as an SET. Both SEUs and SETs are soft errors, and they constitute the majority of SEEs which can be originated in electronic circuits. SEUs have traditionally been more studied because they can directly modify the state of any given circuit, while SETs still need to propagate through the circuit in order to cause a noticeable error, either to a primary output or until they are captured by a memory element. The probability of SET occurrence is higher than for SEUs because they can be originated in any combinational node, although a fraction of these errors is naturally filtered out due to several masking effects, known as *electrical masking*, *logical masking* and *temporal masking*.

- Whenever an SET is originated, it propagates through the circuit to either a primary output or to a memory element, possibly traversing several logic gates along its propagation path. The electrical masking effect refers to the case where the faulty transient pulse is progressively attenuated by subsequent logic gates along its propagation path, until it is eventually filtered out.

- Logical masking applies when the transient pulse is originated in a path which is naturally blocked due to the signal assignments at that particular moment. If an SET is generated in a logic path which is not sensitized, it will never be able to manifest.
- Finally, for a transient pulse to be captured by a memory element, the transient must arrive in synchronization with the capturing phase of the memory element. Temporal masking occurs when the transient pulse arrives at the input of the memory element outside of its capturing window time, and therefore it is not registered.

However, in modern technologies both electrical and temporal masking effects are diminishing, and therefore SETs are increasing in relevance, up to the point of being as important as SEUs.

There are other types of SEEs too, although less probable and more specific. In some applications, either an SEU or an SET affecting the control logic may cause the circuit to hang or to enter in an unknown state from where it cannot normally be recovered. In other words, the device functionality is lost. This situation is known as a Single Event Functional Interrupt (SEFI), and due to its nature it is dependant on the functionality of the circuit. A SEFI has to be cleared by either a logic reset or a power reset [5]. Another kind of SEEs imply hard errors, such as the Single event Hard Error (SHE) or the Single Event Latch-up (SEL). A SHE is a kind of SEU with so much energy that the affected memory cell is stuck in a particular value, being unable to change its state [6]. On the other hand, a SEL is the generation of a latch-up condition, this is, a parasitic structure which short circuits the power supply of a transistor, caused by a SEE. This event requires a power reset in order to cancel it. On the contrary, the high currents generated by the short circuit may permanently damage the device [7]. Finally, with the progressive reduction of transistor sizes, electronic components are becoming small enough for a single particle strike to hit multiple elements, thus potentially originating several bit flips in a row. This phenomenon is known as a Multiple Bit Upset (MBU) [8], and it is more characteristic of modern technologies.

On the other hand, even if SEEs do not occur, each one of these particle strikes may deposit a bit of electric charge inside the semiconductor components of an electronic circuit. Over time, with the accumulation of electric charge due to a continuous exposure to radiation, a progressive degradation of the electrical characteristics of electronic components can be observed, affecting the circuit performance. Eventually, the accumulation of electric charge may result in a permanent failure. This effect is known as Total Ionizing Dose (TID), and it is associated with the accelerated ageing of electronic components in the presence of radiation.

Although the effects of TID are generally more harmful in the long term, these can be monitored and foreseen due to its progressive nature. On the contrary, SEEs may happen any time during the lifetime of the circuit without notice. In addition, with the shrinking of transistor sizes, the energy required to change the state of circuits has diminished. But this means too that it is easier for unwanted external sources such as SEEs to generate perturbations within electronic circuits. Therefore, SEEs now more than ever constitute a concern for the reliability of electronic circuits. This is why in this thesis, the main focus is put over the soft errors induced by SEEs.

1.3 Motivation of research

Soft errors have become a significant threat to circuit reliability. Reduction of transistor sizes and supply voltages, and increase of operation frequencies make electronic devices more sensitive to soft errors. It is therefore required to implement solutions aimed to reduce the impact of these effects in order to maintain acceptable reliability levels.

Among the different types of soft errors, SEUs have traditionally been more studied because they can directly modify the state of a circuit. Consequently, it is a well-known problem. Several techniques for SEU mitigation have been developed, some of which are commonly used. For example, hardened memory cells such as Dual Interlocked Storage Cell (DICE) [9], or replication of memory elements with either fault detection or correction, are approaches used today.

However, with technology scaling, SETs are also becoming very relevant [10]. Actually, the probability of SET occurrence is higher than of SEUs because they can be originated in any combinational node. Nevertheless, there are several effects (logical, electrical and temporal masking) that naturally may prevent a SET from provoking an error.

In fact, SET detection and mitigation approaches take advantage on these masking mechanisms as a means to either block the propagation of transient pulses or detecting when they occur. Electrical masking is exploited by using larger transistors or nodal capacitances [11]. Temporal masking is achieved by using time redundancy, i.e., latching data at different times [12]. Finally, logical masking is exploited by introducing some sort of redundancy in the functionality of the target design. This requires implementing additional logic in order to either detect a faulty logic value in certain points of the circuit or block the propagation path of a transient pulse before it reaches a sensitive element of the circuit.

This work is focused on logical masking techniques, which can be classified in three categories: hardware redundancy, time redundancy and information redundancy techniques [13]. The first group consist in replicating the target design, either completely or partially, so the same computation is performed multiple times. This is combined with either an error detection module or a voter, depending on the number of instances. Among them, TMR and DWC are well-known techniques which are widely used for critical applications. It must be noted that they can be used at different levels of abstraction, from system to transistor level. Time redundancy techniques are based on the same principle as hardware redundancy, but without replicating any logic. Instead, required computations are performed multiple times on the same logic, and compared or voted thereafter. Therefore, such techniques present little area overhead at the cost of severe performance penalties. Finally, information redundancy techniques are based on performing some additional computations on data, thus generating extra bits for each value that allows to check their integrity and even correct some faults. Examples of these are parity bits and Error Detection And Correction (EDAC) codes such as the Hamming code. They can be very effective for single or double errors, so they are well suited for memories or communication protocols, but they cannot be applied in the general case because a low multiplicity of errors cannot be guaranteed.

The capability of DWC and TMR techniques to mitigate both transient and perma-

nent errors makes them good approaches to tackle the variety of potential failure mechanisms that must be considered for advanced technologies. However, These techniques suffer from high overhead in terms of area and power (more than 100% and 200% respectively), which might not be acceptable for certain applications. To alleviate this overhead, alternative techniques have been proposed based on partial error detection or masking. An early partial error masking approach is proposed in [14] which consists on triplicating and voting the nodes with the highest soft error susceptibility. Subsequent approaches attempt to insert redundancies that protect against the most common errors or to resynthesize the circuit to improve reliability [15–19]. Within this context, approximate computing has recently emerged as an alternative approach for building partial DWC/TMR solutions.

Approximate computing is based in the idea of optimizing a given circuit in terms of resource utilization, power consumption or delay, by deviating from the intended functionality. For example, if some parts of a given circuit are rarely excited, they could be removed in order to save some resources, with a reduced impact in the overall functionality. The degree of misbehaviour which can be tolerated in order to gain in either resources, power or timing will depend on the specifications of the target application. Logic circuits which are designed by applying this kind of optimizations are denoted as *approximate circuits*.

The reduced overheads associated with approximate circuits makes them good candidates for implementing partial error mitigation solutions. As it is not required to exactly match the original circuit, the approximate circuit can be smaller but it can still be used to detect or correct errors where it overlaps with the original circuit. Approximate logic circuits provide a systematic framework for the implementation of fault tolerant designs. In addition, they provide a flexibility that other partial DWC/TMR approaches do not have. They can be implemented to satisfy a wide range of error protection or area overhead requirements, while at the same time optimizing the trade-off between both parameters.

1.4 Research goals

The goal of this thesis consist in devising novel approaches for soft error mitigation by logic masking, based on the concepts of the approximate computing. The use of approximate logic circuits will serve to a double purpose: reducing the area overheads with respect to the full DWC/TMR solutions, while providing a flexibility in their design which is not possible with the partial DWC/TMR approaches. Therefore, the proposed techniques have to be designed with the ability of tuning the level of redundancy to different application requirements. Of course, each proposed approach has to be validated by adequate experiments.

In addition, it is desirable that the proposed approaches would be predictable, that is, that the characteristics of the generated solutions could be inferred in advance from the tuning parameters. In that way, the process is more user-friendly, and trial and error iterations are avoided.

Finally, once after the error mitigation approaches will be developed and validated, they should be demonstrated on real applications.

1.5 Document structure

The present document is divided in seven chapters, including this introductory chapter. The rest of this document is structured as follows.

Chapter 2, titled Background and previous work on approximate logic circuits, introduces the main concepts about approximate circuits and presents the state of the art in relation with approximate computing and fault tolerance with approximate logic circuits.

Chapter 3, Circuit approximation method, introduces the fundamental mechanisms which are proposed in this thesis in order to generate approximate circuits. The proposed logic transformations constitute the base on which the heuristics for generation of approximate circuits are built.

Chapter 4, Circuit approximation using static testability measures, deals with the first approximation generation heuristic developed in this thesis, which is based on static testability measures. This approach is initially developed for combinational circuits, although an extension to sequential circuits is later presented.

Chapter 5, titled Circuit approximation using dynamic testability measures, introduces an alternative approximation generation heuristic, based on dynamic testability computations. This change is motivated by the limitations of the previous approach, which are here exposed. This new approach makes use of fault probability computations by means of implications, and therefore an introduction to this field can be found in this chapter. The fault implications are later exploited to introduce a new type of logic transformation for the generation of approximate logic circuits, which is denoted as node substitution.

Chapter 6, Applications, exposes the different applications of the previously presented approximation generation heuristics which have been developed within the scope of this thesis. Namely, the extension of the techniques developed in this thesis for Field Programmable Gate Arrays (FPGAs), the development of a fault tolerant version of a real application microprocessor using approximate logic circuits, and a comparative study of the techniques presented here with evolutionary approaches.

Chapter 7, Conclusions and future work, closes this document. Here, the main contributions and conclusions of this thesis are summarized, and some future research lines which could extend this work are proposed.

Chapter 2

Background and previous work on approximate logic circuits

2.1 Introduction

This chapter serves as an introduction to the concepts of the approximate computing. The main idea behind the approximate computing consist in generating circuits that performs a possibly different but closely related logic function to the correspondent original circuit. Such circuits are denoted as approximate logic circuits. A classification of the different types of approximate logic circuits according to their characteristics is done. This classification determines how a particular approximate circuit can be used to either detect or correct faults.

Fault mitigation with approximate logic circuits is based in the classic DWC and TMR approaches, but replacing the additional copies of the target circuit with approximate versions. In that way, the area and power consumption overheads are reduced at the expense of leaving some faults unprotected. This trade-off between the overheads and the error protection level can be tuned to satisfy the requirements of each particular application. In addition, a careful design of the error mitigation scheme is required in order to ensure a correct functionality, because when approximate circuits are used, differences may be observed even in the absence of faults.

Finally, a review of the state of the art on the approximate computing field is performed, with special emphasis in those techniques which make use of approximate logic circuits to implement fault mitigation mechanisms.

With respect to the rest of the chapter, this is structured as follows. Section 2.2 introduces the main concepts about approximate logic functions and gives a classification of them according to their characteristics. Then section 2.3 explains the general schemes of fault tolerance with approximate logic circuits, and section 2.4 presents some techniques already developed about this topic. Finally, section 2.5 concludes the chapter.

2.2 Approximate logic fundamentals

In this work, the use of approximate logic circuits for partial logical masking is proposed. Therefore, first of all it is required to give a definition of this concept.

Given a target logic function G , an *approximate logic function* with respect to G is another logic function \hat{G} which partially implements G . That is, function \hat{G} gives the same result as G for a fraction of its input space [20]. An approximate logic circuit is simply any circuit that implements an approximate logic function \hat{G} .

It must be noted that the concept of approximate logic refers to functional similarity between logic functions or circuits instead of functional equivalence, in which both TMR and partial TMR techniques are based. In addition, the degree of similarity has no restrictions, and so it can be as high or low as required. The proportion of the input space in which the approximate logic function \hat{G} correctly predicts the result of G gives the quality of the approximation. The comparison between target G and any approximate logic function \hat{G} divides the input space in two parts. The set of input vectors for which both G and \hat{G} give the same result is denoted as the *correct subspace*, while the remaining input vectors (for which G and \hat{G} differ) conform the *incorrect subspace*. If the relative size of the correct subspace is high, then the approximate logic function \hat{G} is close to G . On the contrary, if the relative size of the correct subspace is low, \hat{G} is a poor approximation with respect to G . As an example, consider the target logic function $G = a\bar{b} + bcd$, whose Karnaugh map appears on Figure 2.1a. Now consider the function $\hat{G}_1 = a\bar{b} + cd$, represented on Figure 2.1b. With respect to G , \hat{G}_1 serves as an approximate logic function, as both functions partially overlap. Moreover, \hat{G}_1 is a very close approximation, because there is only one input vector for which the results are different, which is marked as the red cell on Figure 2.1b, that is, it has a big correct subspace. On the other hand, consider the function $\hat{G}_2 = \bar{a} + b$, represented on Figure 2.1c. This is also an approximate logic function with respect to G , but a very poor one. \hat{G}_2 correctly predicts the result of G just for two input vectors, therefore its correct subspace is very small.

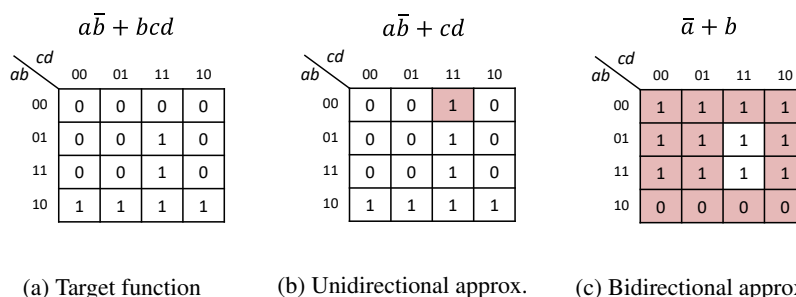


Figure 2.1: Example approximate logic functions

Another interesting conclusion is that the domain of any approximate function \hat{G} is a subset of the domain of G . In other words, function \hat{G} can be implemented with a reduced amount of inputs with respect to the original function G . Considering the

previous example, $\hat{G}_2 = \bar{a} + b$ is implemented with just two of the outputs from target function G . This fact can be exploited to build cost efficient approximate logic functions, which is a key point for the application of this kind of circuits to the field of fault tolerance.

Approximate logic functions can be classified regarding to the characteristics of its correct and incorrect subspaces. If the incorrect subspace of a given approximate function \hat{G} is a subset of either the onset or offset of the target function G , then \hat{G} is an *unidirectional approximation*. On the contrary, \hat{G} is considered a *bidirectional approximation* if its incorrect subspace overlaps with both the onset and offset of function G [20]. For an unidirectional approximation, every difference with respect the target logic function has the same direction, either $0 \rightarrow 1$ or $1 \rightarrow 0$. Function $\hat{G}_1 = \bar{a}\bar{b} + cd$ on Figure 2.1b is an example of an unidirectional approximation, as its incorrect subspace is a subset of the offset of function G . On the other hand, a bidirectional approximation has differences of both types with respect to its target function, such as the case of $\hat{G}_2 = \bar{a} + b$ on Figure 2.1c. This classification determines how approximate functions can be used for fault tolerance, as it is explained on section 2.3.

By definition, an unidirectional approximation satisfies an implication relationship. Given a target logic function G , an unidirectional approximation F which fulfils $F \Rightarrow G$ is denoted as an *under-approximation* with respect to G . This is equivalent to say that $F \subset G$. On the other hand, an approximation H is called an *over-approximation* of G if it satisfies $\bar{H} \Rightarrow \bar{G}$, i.e., $G \Rightarrow H$. Which is equivalent to $H \supset G$ [20]. As an example, consider again the target logic function $G = \bar{a}\bar{b} + bcd$ and its approximation $\hat{G}_1 = \bar{a}\bar{b} + cd$, whose Karnaugh maps are represented on Figures 2.2a and 2.2b respectively. According to previous definitions, $\hat{G}_1 = H$ is an over-approximation of G , because it expands onset of G , that is, the incorrect subspace of H belongs to the offset of function G . Now consider the logic function $F = \bar{a}\bar{b}$ represented on Figure 2.2c. This function works as an under-approximation of G , because it has a reduced onset with respect to function G , and therefore the incorrect subspace of F is a subset of the onset of G .

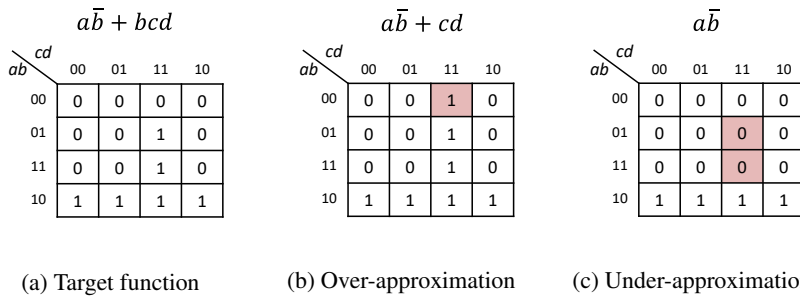
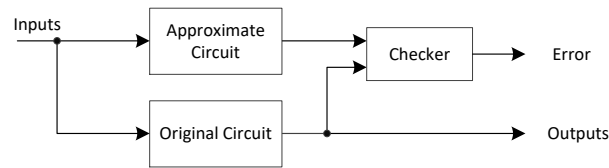


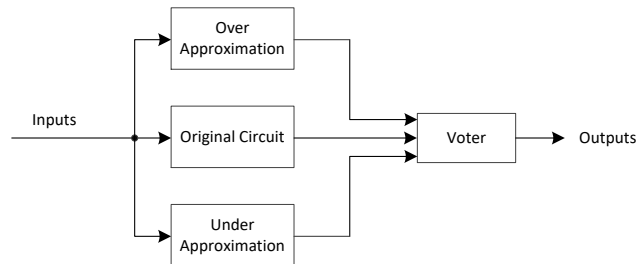
Figure 2.2: Example over- and under-approximations

2.3 Error mitigation by means of approximate logic circuits

The application of approximate logic circuits to the field of fault tolerance is based on the full DWC/TMR approach, but using approximate versions of the target circuit instead of exact copies of it to either detect errors or masking them [21,22]. The interest of this approach lies in implementing approximate circuits which require less resources than the target circuit, so some savings in terms of area and power consumption are obtained at the expense of slightly reduce error mitigation capabilities of the whole system.



(a) Detection scheme - Approximate DWC



(b) Correction scheme - Approximate TMR

Figure 2.3: Fault tolerant schemes using approximate logic circuits

Depending on how many approximate instances are implemented, we talk about either an error detection or correction scheme, which are shown in Figure 2.3. An error detection scheme is built with just one approximate version of the target circuit and the target circuit itself in a DWC-like scheme as shown in Figure 2.3a [22]. Therefore, this approach is called *Approximate DWC* (ADWC) as well. A checking module is included with the purpose of comparing the outputs of both circuits, signalling an error in the case a fault is observed. On the other hand, an error correction scheme is built by using several approximate versions of target circuit. The simplest way of implementing this approach makes use of the target circuit along with two different approximate versions of it in a TMR-fashion [21] (see Figure 2.3b), this approach also receives the

name of *Approximate TMR* (ATMR). A voter is used to select the majority output, thus being able to mask errors produced in any of the three circuits under certain conditions. Of course, additional instances can be implemented, thus having a n-Modular Redundancy built with approximate logic circuits. But all these solutions require a careful design due to the fact that, as approximate circuits are not functionally equivalent to target circuit, outputs may differ even in the absence of faults.

In the approximate DWC approach, the target circuit works in parallel with one of its approximations, and outputs or both circuits are compared by a checking module, which activates an error signal when it observes a mismatch [22]. Because outputs may differ in the absence of faults, the checker has to work only when the outputs of both circuits are expected to coincide. In the general case, this requires an additional logic function which explicitly indicates the overlapping cases. This extra circuit, referred to as indicator function, enables the checker module just in the correct cases. This predictor-indicator approach has been proposed in [20]. As a major drawback, this approach cannot detect faults produced in either the indicator function or the approximate circuit, so they have to be intrinsically robust by design. Moreover, the additional indicator function imposes an extra overhead, thus reducing the benefits of using approximate logic circuits. Alternatively, ADWC can take advantage on unidirectional approximations in the following way. As said in section 2.2, every unidirectional approximation satisfies an implication relationship with respect to the target circuit. Therefore, if an unidirectional approximation is used in an ADWC, the checker only has to check for violations of that implication relationship, thus greatly simplifying the logic required to perform the checking [22]. If an under-approximation F is implemented along with target circuit G , then implication $F \Rightarrow G$ has to be met. Any situation that produces $F = 1$ and $G = 0$ is identified as an error, because by definition such output combination is not possible unless there is a fault. Therefore, the checking module is simply the logic gate $F \cdot \bar{G}$. With this approach, faults of type $1 \rightarrow 0$ on the output of G can be detected when $F = 1$, as well as faults $0 \rightarrow 1$ on F in the case of $G = 0$. Alternatively, the ADWC can be implemented with an over-approximation H . In this case the implication relationship that applies is $\bar{H} \Rightarrow \bar{G}$, and the output combination that violates this rule is $H = 0$ and $G = 1$. Whenever this output combination is observed, it indicates the presence of a fault. In consequence, the checker has to implement just the logic function $\bar{H} \cdot G$. This implementation is able to detect both faults of type $0 \rightarrow 1$ on G when $H = 0$ and faults $1 \rightarrow 0$ on H for $G = 1$. Because the ADWC with unidirectional approximations is able to detect errors occurring in the target circuit in just one direction, this approach is well suited for logic functions with either a large onset or offset.

With respect to the approximate TMR approach, it implements the target circuit along with two different approximate versions of it, and the outputs of all three instances are voted to select the majority value for each output. As the ATMR approach includes two approximate circuits, a careful design is required in order to, at least, ensure a correct functionality in the absence of faults. This condition is met when, for every input vector, at least one of the approximate circuits agrees with the target circuit. In other words, it is required that incorrect subspaces of both involved approximations do not overlap. Otherwise, if the two approximate circuits agree in an incorrect value, the ATMR will produce a wrong result. There are many types of approximate logic

circuits that can satisfy this requirement. Among them, the simplest way of assuring this condition is by means of unidirectional approximations. As said in section 2.2, for an under-approximation F with respect to target circuit G , $F \subset G$ holds, that is, the set of input vectors for which both functions differ is a subset of the onset of G . On the other hand, the incorrect subspace of an over-approximation H with respect to G fully belongs to the offset of G , or in other words, $G \subset H$. Therefore, by implementing an ATMR with target circuit G , an under-approximation F and an over-approximation H as represented on Figure 2.4a, at least one of the approximate instances agrees with G for every input vector, thus ensuring a correct functionality in the absence of faults by construction.

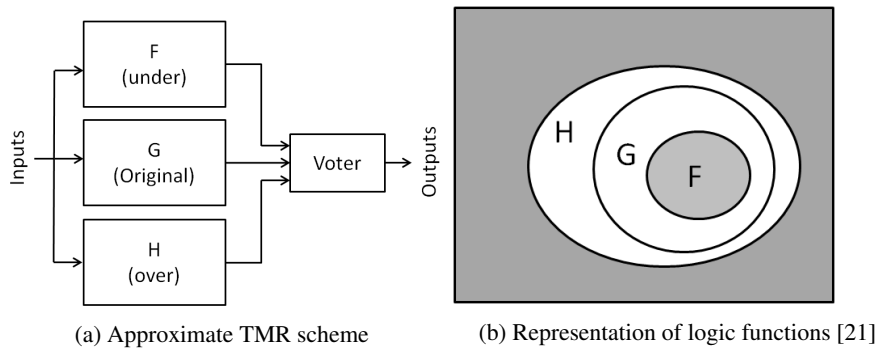


Figure 2.4: Approximate TMR with unidirectional functions

How this scheme works can be better explained with the diagram of Figure 2.4b, where the on-sets of all three logic functions are represented, and the Table 2.1, which summarizes the possible output combinations of the ATMR scheme under normal operation (that is, in the absence of faults). From both it is clear that the ATMR scheme satisfies the relationship $F \subset G \subset H$ [21]. In the offset of G , both G and F outputs are set to 0, so the voter provides the correct result independently from the value of H . On the other hand, in the onset of G , both G and H functions generate a logic 1, thus the correct value is driven to the output no matter what the result of function F is. In conclusion, the relationship $F \subset G \subset H$ ensures that this scheme is functionally equivalent to target function G . But in addition, it has fault masking capabilities. In the onset of function F , all three circuits are set to 1. Because the three instances are implemented separately, a single fault can only affect one circuit and therefore its effect will be masked in any case. The same applies in the offset of H , where all circuits give a logic 0. In any of these situations, the scheme behaves like a full TMR. Therefore, total protection is provided for the set of input vectors included in $F \cup \bar{H}$, which corresponds with the shadowed area in Figure 2.4b. On the contrary, in the area in between one of approximate functions already produces an incorrect result. By default, this wrong value is masked by the voter, but a fault in any of the other two instances may cause a second wrong value which would propagate to circuit outputs. In the incorrect subspace of F , $F = 0$ while both G and H provide a logic 1. Therefore, any fault which causes a $1 \rightarrow 0$ change in either G or H outputs will not be masked, thus

provoking an error. It must be noted that in this situation, circuit F is still protected against faults, because any fault which causes a bit flip on its output simply corrects the discrepancy caused by the approximation. Similarly, in the incorrect subspace of H , circuit outputs are $F = 0$, $G = 0$ and $H = 1$. Faults of type $0 \rightarrow 1$ in either F or G will arise an error, while any fault affecting H will be masked. Therefore, the level of protection provided by this approach is related to the combined probability of these incorrect subspaces. In conclusion, the challenge of the ATMR approach consist in finding approximations of target circuit with an optimal trade-off between the probability of their incorrect subspaces and the overhead they impose.

F	G	H	Output	Notes
0	0	0	0	All errors masked
0	0	1	0	$0 \rightarrow 1$ errors in F or G are not masked
0	1	1	1	$1 \rightarrow 0$ errors in G or H are not masked
1	1	1	1	All errors masked

Table 2.1: Summary of the ATMR operation

The approach based on unidirectional approximations is the preferred way of implementing an ATMR scheme in this work, because it is straightforward and unidirectional approximations can be automatically generated by means of simple transformations of the original circuit under certain conditions that depend on structural properties, as it will be explained in chapter 3. But there are alternative configurations with bidirectional approximations that can also be implemented. Moreover, even the target circuit could be replaced with another approximate circuit in the ATMR, thus having a full ATMR scheme built with approximate circuits exclusively, as proposed in [23]. The only requisite is that the incorrect subspaces of the different approximate circuits do not overlap. However, all these alternative schemes require a careful design in order to ensure a correct functionality in the absence of faults, even more than with the conventional ATMR.

In any case, it must be noted that approximate versions of target circuit are susceptible to faults, and any fault affecting any of approximations may propagate to an error in case that the other approximate circuit already generates an incorrect result. However, this situation can be detected because, considering the restrictions imposed for the selection of approximations, it is not possible that both approximate circuits disagree with the target circuit unless there is an error. Thus, all faults generated in the approximate logic circuits can be either masked or detected. In general, it is expected that the contribution to the global error rate due to the additional approximate circuits is compensated by the amount of errors masked in the target circuit. However, if that is not the case, the voter can be complemented with an error detector, thus ensuring that global error rate always diminishes as long as the quality of approximations increases (i.e., incorrect subspaces of approximate circuits reduce, thus becoming more functionally similar to the target circuit).

All these approaches rely on an additional module which is in charge of either detecting errors (a checker module) or masking them (a voter), exactly as with full DWC/TMR approaches. It is well known that these modules constitute a vulnerable

point in all these techniques. A fault affecting the voter will not be masked and will cause an error, while a fault in the checking module will cause a false positive in the error detection. This problem is typically solved by means of fault tolerant versions of these modules. With that goal, the simple voter of the TMR approaches can be replaced by a triple voter, in such a way that any fault originated in the voting logic will affect just one of the three instances. Similarly, the checker module of DWC approaches can be replaced by a totally self-checking checker with two-rail error coding.

2.4 Overview of approximate logic techniques

Up to date, several techniques aimed at applying approximate logic circuits to the field of fault tolerance have already been proposed. While the general schemes for implementing error mitigation with approximate logic circuits are clear and well documented (see section 2.3 for details), generation of optimal approximate versions of a target design constitutes an open question with many alternatives. Here is where the different approaches widely differ.

When generating approximations of a target design, two major questions arise: 1) how are approximate circuits designed? and 2) which are the requirements of target application to be satisfied? For the second question, several constraints in terms of area overhead, power consumption, delay, reliability and different error metrics can be used, depending on the particular requirements of each application. With respect to the first question, several approaches can be followed. Although it is possible to manually generate approximations for a given design, thus having full control of the characteristics of approximations, the limited applicability of this approach renders it less interesting than the automated generation of approximate circuits. In the field of fault tolerance, the latter is the preferred approach for approximate circuit design. When opting to implement an automated design method, the previous questions can be reformulated as: 1) which mechanism is applied to generate approximations? and 2) which criteria are used to selectively apply this mechanism? Each one of the different existing techniques propose its own answer to these questions. It must be noted that these decisions are influenced by the preferred circuit description format.

The first technique in the list, proposed in [21], generates approximations by means of transformations applied over the Binary Decision Diagram (BDD) representing target logic function. By selectively redirecting some paths within the BDD it can be ensured that either an over- or under- approximation is generated. The authors focus on fault masking, so both over- and under-approximate circuits are used. Then each pair of over- and under-approximations is evaluated in terms of soft error rate reduction, which is computed as the difference between the proportion of masked input vectors and the estimated area overhead. In order to implement fault tolerance, the authors propose a modified version of the ATMR scheme, where the majority voter is replaced with the masking function $F + G \cdot H$. While this masking function is simpler, it makes also the whole system more vulnerable to errors, because $0 \rightarrow 1$ errors in F cannot be masked in any case.

The technique proposed in [22] focuses on fault detection with unidirectional approximations. In that work, the preferred circuit description format is a multilevel

technology-independent network. This consists in a directed acyclic graph where each node constitutes a partial logic function, which is represented as a Sum of Products (SOP) expression. The approximation process is performed in two stages. First, each node in the list is assigned a type according to both the type of immediately subsequent nodes and the local observability of the node. Although the authors use a particular terminology, the node types are closely related to the bidirectional, over- and under-approximation types. Thereafter, local functions of certain nodes, according to their types, are simplified by selecting a subset of cubes or terms. Two different methods for cube selection are proposed. In the first one, cubes are selected directly from the SOP expression of the node. While this first approach guarantees the generation of unidirectional approximations, it considerably limits the solution space. Alternatively, authors propose a second cube selection method that takes advantage of observability don't-cares to expand the range of feasible solutions. However, this alternative method does not guarantee the generation of unidirectional approximations, thus requiring a Boolean Satisfiability Problem (SAT) solver in order to ensure correctness. The generated ADWC schemes are subsequently evaluated with respect to concurrent error detection rate. This approach is extended in [20] by considering predictor-indicator bidirectional approximations. In addition to the previous procedure, these functions can be further optimized by taking advantage on the don't-care interdependencies existing between them. In particular, all those input vectors where the indicator function evaluates to 0 (that is, when target and approximate functions are expected to differ) can be considered as don't cares for the predictor (i.e. approximate) function. Similarly, all those input vectors where the predictor agrees with target logic function can be used as don't-cares for optimizing the indicator function. Additionally, the work in [20] proposes the implementation of fault masking by using just one approximation. This can be seen as a borderline case of an ATMR where one of the approximations has been reduced to a logic constant. This particular topic will be discussed in section 4.2.

In a similar way, the technique in [24] builds approximations by selectively choosing cubes from the target logic function expressed as a SOP. But in this case, the SOP expression is computed for the whole logic function, without splitting it into pieces. In addition, this approach is intended for an ATMR scheme, so both under- and over-approximations are considered. The first is generated by selecting cubes among the logic function minterms, while the latter is generated by choosing among the cubes representing the function maxterms. In this approach don't-cares are not considered, thus meaning that target logic function is completely defined. Cubes are iteratively selected according to a fault reduction metric, which is a combination of the number of terms covered by the cube (i.e., the number of input vectors that will be protected by the cube, which is related to the gate size) and the fault propagation probability through that cube, estimated by statistical fault injection. On each iteration, the best cube according to this metric is selected and included as a part of its correspondent approximate function, and the fault reduction value of all remaining cubes is recomputed, updating also the coverage of each cube in order to reflect the overlap with respect to all previously selected cubes.

Finally, the approach proposed in [25] performs an analytical decomposition of the target logic function into their simplest literals, which are then recombined according to

certain rules in order to generate valid unidirectional approximations. Besides, once the over- and under-approximate functions have been generated, some layout techniques at transistor level are applied with the goal of minimising the SET sensitivity of each individual logic gate specifically for the input vectors belonging to the incorrect subspaces of the generated approximate circuits. Due to the characteristics of this approach, only small logic functions can be processed with it.

Approximate logic circuits have other applications in addition to error mitigation. Their potentially superior performance in terms of area, power consumption and speed with respect to their original circuits make approximate logics well suited for those applications which can tolerate some degree of misbehaviour. In these cases, the original circuit can be replaced by an approximate version of it, which is less accurate than the original circuit, but offers lower overheads and reduced computation time. This is denoted *approximate computing*. In this paradigm, the same questions previously considered about the design of approximate versions of target circuit and its relationship with the expected requirements of target application still apply. But in addition, there is the issue of performing functional verification of the final application, because now incorrect outputs can be generated even in the absence of faults.

With respect to the design of approximations in this field, there are several works focused on generating efficient approximations for specific common designs, such as approximate adders [26,27] and multipliers [28]. But, because this kind of approaches are limited to specific designs, research has been done on design automation methods as well, so the benefits of approximation can be applied to any design. With this goal, several synthesis approaches have been proposed, including SOP reduction [29], redundancy propagation [30], don't-care-based simplifications [31], formal verification techniques [32], dedicated three-level circuit construction heuristics [33] and And-Inverter Graph (AIG) rewriting [34].

Finally, another interesting approach consist on evolutionary generation of approximate circuits. Evolutionary algorithms are used in many applications to solve hard optimization and design problems, making use of the high processing capabilities of latest technologies in order to explore a wide range of solutions by trial and error. Since the beginning of research in evolutionary computation, this techniques has been applied for purposes of hardware optimization. Several monographs [35,36] summarize the applications from the field of electronic design, diagnostics, and testing. Later, evolutionary algorithms were applied not only to optimize parameters of existing circuits, but also to generate complex circuit structures and dynamically adapt them [37]. Recently, evolutionary algorithms have been applied to the generation of approximate circuits [38,39], a field which suits well for this kind of techniques. As the method is intrinsically based on the trial and error approach, it is usually very time consuming, but, on the other hand, capable of discovering solutions hard to reach by other automated design methods. A more detailed explanation of this kind of techniques is addressed in Section 6.4.1.

2.5 Conclusions

This chapter has presented the main concepts on approximate logic circuits, and how they can be used for error mitigation.

The approximate computing provides a systematic framework for the implementation of partial error mitigation solutions. Approximate logic circuits can be used for either error detection or correction in those situations where the approximate logic function overlaps with the target function. The degree of functional similarity can be tuned for different trade-offs between error coverage and overheads. However, generating approximate circuits with an optimal trade-off for a particular application is still a challenging issue.

A review of the state of the art in relation with fault mitigation approaches based in approximate computing has been done. The techniques presented are quite recent, so this research topic is still young. Besides, most of the existing techniques are synthesis oriented and depart from a conceptual description of the target circuit. In such conditions, estimating the effect of approximations in terms of either overheads or error protection level is a really difficult task.

This thesis is devoted to develop novel fault mitigation techniques based in approximate logic circuits which overcome the limitations of these previous approaches. On the one hand, in this thesis the generation of approximations is based in structural properties of circuits. Therefore, departing from a circuit implementation instead of a higher abstraction level allows estimating area overheads with more accuracy. On the other hand, the approximation generation will be driven by testability measures, making use of some heuristics to guide the approximation process. In that way, approximate circuits which minimize the impact on the error coverage may be favoured.

Chapter 3

Circuit approximation method

3.1 Introduction

In this thesis, the preferred way of implementing error mitigation with approximate circuits is by an ATMR scheme. In this scheme, target circuit is grouped with two different approximate circuits, and the output is selected by voting among the three instances, which allows masking errors. The ATMR has been selected because it can potentially protect against every fault affecting the circuit while it assumes that additional logic can be sensitive to errors as well. However, a careful design of the approximate circuits is required to both ensure the correctness of the ATMR in the absence of errors and optimize the trade-off between error protection and overheads.

When designing approximate logic circuits, two main aspects have to be considered: which mechanism is used to generate approximations and which criteria are used to selectively apply them. This chapter addresses the first question.

The circuit approximation approach adopted in this thesis is based on assigning logic constants to circuit lines. This mechanism is called *fault approximation*, because it is equivalent to force stuck-at faults, or line approximation. But in order to ensure a correct operation of the ATMR in the absence of faults, not every line can be approximated. Specifically, only by approximating faults in unate lines the unidirectional approximations required to ensure the correctness of the ATMR can be generated. Fault approximations in the same direction can be combined for further savings.

However, there are multiple other ways to generate approximations, such as BDD simplification, cube addition or removal, subcircuit resynthesis, etc. The reader may consult section 2.4 for details. The method applied is usually influenced by the preferred circuit description format, which in this thesis is a gate level netlist.

Approximating a binate fault generates a bidirectional approximation, which is not valid for an ATMR. In order to approximate a fault in a binate line, first it is necessary to perform some logic transformations, jointly denoted as *unate expansion*. The unate expansion consist in duplicating all binate nodes, and separating their connections according to the parity of each connection. In that way, any circuit can be transformed into a fully unate circuit.

The penalty of the unate expansion is an increased circuit area. As a way of minimising this effect, an adaptive fault approximation mechanism is proposed. With this approach when a fault is approximated, instead of just forcing a constant logic value on a line, an additional gate with a don't-care input is introduced, in such a way that the approximation can be taken or discarded depending on the logic value assigned to the don't-care input. An external synthesis software would be in charge of assigning the values of all those inputs for an optimal solution.

With respect to the approximation selection criteria, several heuristics have been developed over the whole thesis, which are properly described in chapters 4 and 5. All these heuristics are based in the fault approximation mechanism presented here.

The structure of this chapter is as follows. Section 3.2 introduces some relevant concepts which are necessary in order to understand the proposed technique, such as the concepts of unidirectional and unate circuits. Next, section 3.3 explains the basic fault approximation approach and how it can be applied to generate approximate logic circuits. Section 3.4 describes the unate expansion procedure. Then, section 3.5 addresses the adaptive fault approximation approach. Finally, section 3.6 concludes the chapter.

3.2 Unidirectional circuits

The approximation generation mechanism applied in this thesis consist on forcing faults within the circuit. In particular, the interest of this approach resides in forcing unidirectional faults, as it will be explained in section 3.3. An *unidirectional fault* is a fault which appears in such a way that, for any input vector, all erroneous outputs are affected in the same direction, either $0 \rightarrow 1$ or $1 \rightarrow 0$. From here it can be deduced that any fault which propagates to just one output is unidirectional too, as well as any fault which cannot propagate to any output. A *circuit* is denoted as *unidirectional* with respect to a given set of faults Φ if every fault $f \in \Phi$ is unidirectional [40]. From now on, it will be assumed that Φ is the set of all stuck-at faults within the target circuit.

According with this previous definition, checking whether a circuit is unidirectional or not would require a fault simulation in the general case. In order to avoid this, an alternative property is used instead.

A logic function $F(x_1, x_2, \dots, x_n)$ is said to be *positive* in x_i if, expressing F as a sum of products, x_i does not appear complemented. Analogously, F is *negative* in x_i if x_i only appears complemented in the sum of products. This holds not only for the function inputs, but also for any intermediate result by performing a variable change. Therefore, a positive function F in x_i can be expressed as

$$F = x_i F_1(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) + F_0(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) \quad (3.1)$$

while a negative function can be expressed as

$$F = \overline{x_i} F_1(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) + F_0(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) \quad (3.2)$$

A logic function F is said to be *unate* in x_i if F is either positive or negative in x_i , otherwise it is *binate* in x_i [41]. When considering multiple output functions it is said

that F is unate in x_i if either every partial logic function F_j corresponding to output O_j is positive in x_i , or negative instead. Otherwise it is binate in x_i . It must be noted that if some outputs are positive in x_i while the rest are negative, then the whole function is binate in x_i .

From these definitions it can be inferred that if a logic function is unate in x_i , then the faults x_i stuck-at 0 and x_i stuck-at 1 are both unidirectional. Unate condition is more restrictive but it ensures unidirectionality while it is a property which can be easily checked, because it depends on structural characteristics as explained below. In practice, only a very small proportion of binate lines may present unidirectional faults [40].

A logic circuit is the logical representation of a multiple output logic function. It is said that a node or line x_i within a logic circuit has *even parity* if every logical path from x_i to the outputs of the circuit has an even number of inversions. Analogously, x_i has *odd parity* if all propagation paths from x_i to the outputs have an odd number of inversions [42]. If node or line x_i within logic circuit C has even or odd parity, then C is positive or negative in x_i respectively, which implies that C is unate in x_i . Otherwise, x_i has no parity, and therefore C is binate in x_i .

Parities in a circuit can be computed by traversing the network from primary outputs to primary inputs using the following algorithm [43]. Here, the parity of a line is considered as a variable with three possible values: 0 corresponds to even parity, 1 means odd parity, and x denotes no parity.

1. For a primary output O, $parity(O) = 0$.
2. For an output w_i of a gate G with output w_o , $parity(w_i) = parity(w_o) \oplus Inv(G)$. Here, $Inv(G)$ denotes the inversion value of gate G. This value equals to 0 for a non-inverting gate -AND, OR- and has value 1 for an inverting gate -NOT, NAND, NOR. If w_o has no parity or gate G has no inversion value -case of XOR and XNOR gates-, then the inputs of that gate have no parity either.
3. On a multiple fanout point, the stem wire parity equals the parity of the branches if every branch has the same parity. Otherwise, the stem line has no parity.

Figure 3.1 shows an example of how parities are computed with the previous algorithm. First of all, even parity is assigned to primary outputs, and then the circuit is traversed towards the inputs, analysing each gate. The first node, n4, is non-inverting, so its inputs maintain the output parity. The same applies with gate n3. With respect to node n2, it has one non-inverting input, corresponding to primary input a, which preserves even parity, while the other input has an inverter, and therefore its parity changes to value 1. Now, the output of gate n1 is a stem line, and therefore its parity depends on the parities of the branches, which have different values. Therefore, the output of n1 has no parity, and consequently neither inputs c and d have parity.

Even if a logic circuit has binate lines, it can be modified through logic transformations in order to become fully unate. This topic is discussed in section 3.4.

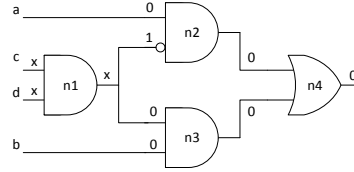


Figure 3.1: Parity computation example

3.3 Unidirectional fault approximation

Fault approximation is the approximation generation method applied in this work. Given a target logic circuit, approximations are generated by simply assigning a constant value to any of the lines of the circuit, which is equivalent to forcing a stuck-at fault on that particular line. In principle, any stuck-at fault could be forced in order to generate an approximation. However, this thesis work is focused on the generation of unidirectional approximations, because they ensure the correctness of the ATMR scheme presented in section 2.3 for error correction. In that case, fault approximation must be limited to unate lines. The justification of this statement is presented below.

Let be a logic function G unate in x . Then the cofactors of G obtained by assigning $x = 0$ and $x = 1$ are both unidirectional approximations with respect to G . If G is positive in x , then $G(x = 0)$ is an under-approximation of G and $G(x = 1)$ is an over-approximation. On the contrary, if G is negative in x , then $G(x = 0)$ is an over-approximation, while $G(x = 1)$ is an under-approximation with respect to F . This is demonstrated by realizing that, if function G is positive in x , then $G = xG_1 + G_0$ according with equation 3.1, where logic functions G_0 and G_1 do not depend on x . Therefore the cofactors of G with respect to x would be $G(x = 0) = G_0$ and $G(x = 1) = G_1 + G_0$. It can be verified that $G(x = 0) = G_0 \Rightarrow \overline{G}$, i.e., $F = G(x = 0)$ is an under-approximation of G . In the other hand, $\overline{G(x = 1)} = \overline{G_0 \cdot G_1} \Rightarrow \overline{G}$, which means that $H = G(x = 1)$ is an over-approximation with respect to G . Demonstration for a negative function is analogous by simply considering that $G = \overline{x}G_1 + G_0$, according with equation 3.2 [43].

In summary, if a line x has even parity, assigning a logic 0 or 1 on x (i.e. forcing x stuck-at 0 or 1) will result in an under- and over-approximation respectively. On the contrary, if x has odd parity, then assigning a logic 0 generates an over-approximation, and a logic 1 generates an under approximation. To illustrate this, consider the example circuit of Figure 3.2a, along with its Karnaugh map. It can be verified that every line within this circuit has even parity with the only exception of input b, whose branch to node n1 has odd parity. Within this circuit, consider a line with even parity, such as input d. If a logic 1 is assigned to that input, then the circuit of Figure 3.2b is obtained, and it can be verified by means of Karnaugh maps that the circuit is an over-approximation with respect to the original circuit. On the contrary, if a logic 0 is assigned on input d, circuit of Figure 3.2d is obtained, which is an under-approximation

compared with the circuit in Figure 3.2a. Now, let us focus on a line with odd parity, i.e., the wire from input b to node n1. By forcing a stuck-at 0 at that line, circuit of Figure 3.2c is obtained, while circuit in Figure 3.2e is generated by forcing stuck-at 1. It can be verified that Figure 3.2c is an over-approximation of Figure 3.2a and 3.2e is an under-approximation with respect to the original circuit.

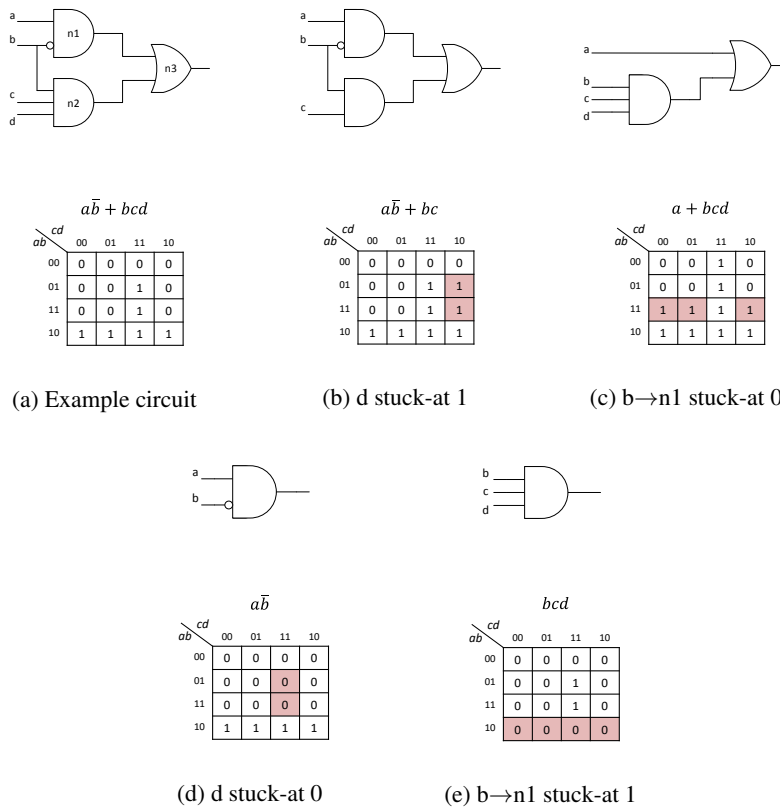


Figure 3.2: Unidirectional fault approximation examples

Approximations can also be generated by combining several assignments on the same circuit as long as the final result is either an under- or an over-approximation. To this purpose, the transitive property applies to unidirectional approximations. Given the logic functions G, F_1 and F_2 , if F_1 is an under-approximation of G and F_2 is an under-approximation of F_1 , then F_2 is an under-approximation with respect to G as well. The demonstration is straightforward by taking into account that every unidirectional approximation satisfies an implication relationship. In other words, if $F_1 \Rightarrow G$ and $F_2 \Rightarrow F_1$, then $F_2 \Rightarrow G$. The same property applies in case of over-approximations. Given logic functions G, H_1 and H_2 , if H_1 is an over-approximation of G and H_2 is an over-approximation of H_1 , then H_2 is also an over-approximation with respect to

G . Or, expressing it with implications, if $\overline{H_1} \Rightarrow \overline{G}$ and $\overline{H_2} \Rightarrow \overline{H_1}$, then $\overline{H_2} \Rightarrow \overline{G}$. In summary, if several assignments are combined, the resulting circuit will be an under-approximation if each individual assignment leads to an under-approximation. On the contrary, if each individual assignment generates an over-approximation, the result of combining them will be an over-approximation. To illustrate this, let us take again the previous example from Figure 3.2. There, faults d stuck-at 1 and $b \rightarrow n1$ stuck at 0 both produce over-approximations when forced (Figures 3.2b and 3.2c respectively). Therefore, both transformations can be combined to create another circuit, which is shown on Figure 3.3 and works as an over-approximation of the original circuit on Figure 3.2a. As long as logic transformations are applied on unate lines, the final result will not depend on the order in which faults are approximated, because parities do not change due to such transformations.

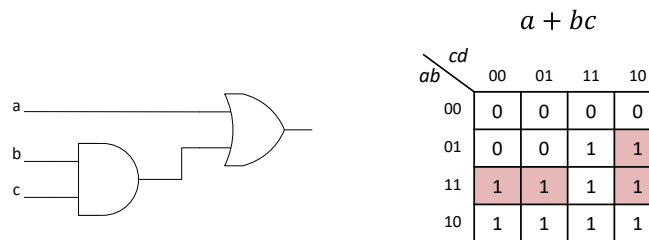


Figure 3.3: Multiple fault approximation

From now on, approximating a particular fault is said to denote that the fault is forced. In addition, line approximation is used to denote that a constant is assigned on a particular line of the circuit, i.e., that a fault is approximated on that line.

3.4 Unate expansion

The fault approximation method described in the previous section can be applied to any fault within the target circuit, but to ensure that the result will be either an under- or an over-approximation, the approximated fault must be unidirectional. Otherwise, the resulting circuit would not meet the characteristics required in order to implement an ATMR scheme. To illustrate this, consider the logic circuit represented in Figure 3.4a. In this circuit, node $n1$ is binate because it has one path to the output with one inversion (through $n2$) and another with no inversion (through $n3$). Now suppose an approximation is generated by approximating (i.e. forcing) fault d stuck-at 1. The resulting approximate circuit after proper simplifications is shown in Figure 3.4b. By comparing the Karnaugh maps corresponding to both circuits it can be seen that approximation is a bidirectional one, because neither the on-set or the off-set of original circuit are preserved.

Fortunately, any circuit can be made fully unate by means of logic transformations,

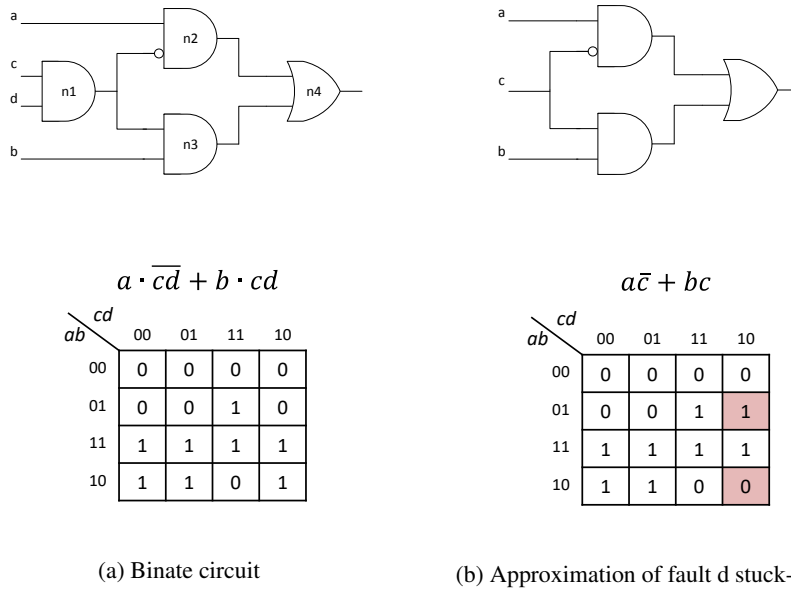


Figure 3.4: Binate fault approximation

with the only exception of primary inputs. This is achieved by traversing the circuit from outputs to inputs while computing the parity of each node. If a binate node is found, it is duplicated and the paths from that node are split between the two copies. Paths with even parity are assigned to one replica, and paths with odd parity to the other replica. This process is repeated until primary inputs are reached on every propagation path [44]. Applied to the previous example (circuit of Figure 3.4a), it requires to duplicate the binate node n1 as shown in Figure 3.5. Then the original node is connected to gate n2, while the replica n1' is linked to gate n3. If there were additional nodes in the transitive fanin of n1, this process should be repeated until primary inputs were reached. The expanded circuit is completely unate, excluding primary inputs c and d which cannot be directly approximated. However, it must be noted that individual lines from c and d to n1 and n1' do have parity and therefore can be approximated, despite the stem line is still binate.

XOR and XNOR gates interfere with computation of parities. As previously said in section 3.2, these gates have no inversion value, because the output inverts inputs or not depending on its input values. For the sake of approximation, XOR and XNOR gates are temporarily substituted by their equivalent with AND, OR and NOT gates. Figure 3.6 shows the equivalent for a XNOR gate. Equivalent for XOR is analogous.

Due to these transformations the circuit size may increase, but this phenomenon is expected to be compensated by fault approximation, or by undoing the transformations performed in case no faults are approximated. In those cases where the penalties of applying the unate expansion are greater than their benefits, the adaptive fault approx-

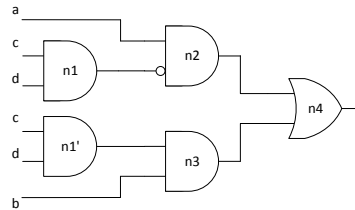


Figure 3.5: Unate expansion

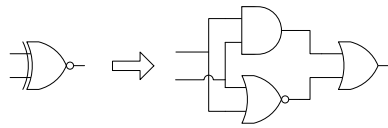


Figure 3.6: XNOR equivalent

imation approach presented in the next section can be applied, which is intended to minimise the overheads due to the unate expansion. On the other hand, these transformations are not mandatory, but helpful. In a fully unate circuit every fault can be approximated, which results in more opportunities for resource savings. But there is no problem in working with a partially binate circuit, as long as faults on binate lines are never approximated.

3.5 Adaptive fault approximation

The unate expansion introduced in the previous section allows approximating the circuit sections which could not be normally approximated. However, this has the cost of increasing the area of the target circuit, and by extension the area of the approximate circuits can be affected too, because they are initially created as exact replicas of the target circuit. In the general case it is expected that the area of the approximate circuits will decrease because of the approximations performed over them. But it may happen too that the area of the resulting approximate circuits is greater than the original circuit itself. For example, this can be the case of a fully binate circuit where, after performing the unate expansion, only faults in one of the two unate branches are approximated, leaving the other branch unmodified, in such a way that both branches cannot be later recombined through synthesis. Such situation is not desirable at all, because it does not improve the area overhead with respect to the traditional DWC or TMR solutions.

Therefore, when the unate expansion is applied, it would be desirable to have a

mechanism which would decide if approximating a given set (or subset) of faults is worth or not. However, the area of the approximate circuits cannot be accurately estimated until the whole approximation process has been completed. Because of this, the fault approximations should be annotated somehow in the circuit during the approximation process, and in a later step each one of the suggested fault approximations should be taken or not in order to optimize the area. Here a method for inserting fault approximations which can be optionally undone is proposed, which is denoted as *adaptive fault approximation*. The subsequent optimization step can be performed by a synthesis tool, for example.

In the proposed adaptive fault approximation approach, a fault is approximated by inserting an auxiliary 2-input logic gate in the corresponding line, instead of just assigning a constant logic value. The type of the logic gate depends just on the logic value of the approximated fault: AND for stuck-at 0 faults, and OR for stuck-at 1. The particularity of the approach is that the side input of the auxiliary gate has a don't-care logic value. In that way, depending on the logic value finally assigned to the don't care input, the auxiliary gate may force the line to a logic constant (forcing the intended stuck-at fault on the line) or alternatively behave as a buffer (thus discarding the logic transformation).



Figure 3.7: Instruments for adaptive fault approximation

Figure 3.7 shows the logic gates which are inserted in order to approximate faults with the adaptive fault approximation approach. A stuck-at 0 fault is approximated by inserting an AND gate with a don't-care input (Figure 3.7a). The other input (labelled with a) is connected to the line where is located the fault being approximated. If the don't-care input is assigned to 0, it sets the output of the AND gate to a logic 0, effectively forcing the fault stuck-at 0 on line a. On the other hand, if the don't-care input is set to 1, then the output of the AND gate is dominated by the logic value of the line a, which leaves the approximated fault without effect. On the contrary, a stuck-at 1 is approximated by inserting an OR gate (Figure 3.7b). In this case, setting the don't-care input to 1 forces the same value at the output (thus taking the approximation), while setting a 0 allows propagating the logic value of the original circuit line (discarding the approximation).

An application example of the adaptive fault approximation approach is explained here, using the circuit of Figure 3.2a as the target circuit. From that circuit, two faults have been selected for approximation: d stuck-at 1 and $b \rightarrow n1$ stuck-at 0. The fault in the input d is approximated by inserting an OR gate, as it can be seen in the Figure

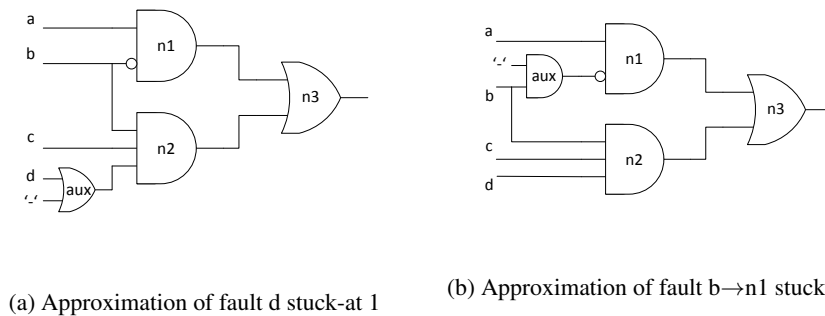


Figure 3.8: Adaptive fault approximation example

3.8a. If the don't-care input is set to 1, then the approximation is taken, resulting in the approximate circuit of Figure 3.2b. On the contrary, by setting a logic 0 the approximation is discarded, obtaining the target circuit itself. On the other hand, the stuck-at 0 fault from the input b to the node n1 is approximated by inserting an AND gate (Figure 3.8b). In this case, by assigning the don't-care input to 0 the fault is forced, generating the circuit of Figure 3.2c, while the approximation is undone by setting a logic 1 on the don't-care input. These both faults are of the over-approximate type, and therefore they could be grouped together by inserting both auxiliary logic gates at the same time.

Once all the selected approximate faults have been inserted in the circuit, all the don't-care inputs have to be set to a value, either a logic 0 or 1. This decision is performed individually for each don't-care input with the goal of optimising the resulting circuit area. For each approximated fault, it can be decided either to take it, reducing area from the fault approximation, or to discard it and try to reduce area by merging the duplicated logic (assuming that in the first place an unate expansion was performed). The complexity of this optimization problem rapidly grows with the number of don't-care inputs and it is not addressed here. Instead, it is proposed that an external synthesis tool performs the optimization step. However, the synthesis tool has to be instructed to deal with this kind of instruments. In some cases, synthesis tools struggle when processing don't-care inputs, which is an inconvenient for the application of the adaptive fault approximation approach.

3.6 Conclusions

This chapter introduces the basic method for the generation of approximate circuits applied within this thesis, which is denoted as fault approximation. In summary, approximations are generated by assigning logic constants to any circuit line, which is equivalent to forcing stuck-at faults. Any fault can be forced in order to generate an approximation, but to ensure that the logic transformation leads to either an under- or an over-approximation with respect to the original circuit it is necessary that the forced

fault is unidirectional, i.e., the circuit has to be unate with respect to the fault injection point. Moreover, approximations can be generated by combining several assignments as long as each approximated fault is unidirectional and produce a change in the same direction (either to an under- or to an over-approximation).

In order to approximate a fault in a binate line, first an unate expansion is required, which consist on duplicating every binate node and separating its outputs according to their parities. The penalty of this approach is an increased circuit area. As a way of minimising such penalties, an adaptive fault approximation approach has been proposed, based in don't-care inputs. Then, the approximations are taken or discarded depending on the logic values assigned to the don't-care inputs.

In addition, several improvements of the basic fault approximation approach have been studied and implemented along this thesis. Such improvements are introduced here, and are explained in detail in their corresponding sections.

- Complimentary to assigning logic constants to lines, the possibility of substituting a line with any other line within the circuit has been studied. In this way the range of possible logic transformations is expanded, allowing to reach solutions out of the scope of the basic fault approximation method. To ensure that some circuit properties are preserved, there are some constraints that lines have to meet in order to become candidates for substituting a given line. This technique is denoted as *node substitution*, and is explained in depth in section 5.4.
- Bidirectional faults cannot be directly approximated, even after unate expansion is performed. In fact, bidirectional faults are split into a pair of complimentary unidirectional faults. Therefore, approximation of a bidirectional fault requires approximating two faults in opposite directions simultaneously, which is not straightforward. This procedure intends to minimize the area overhead due to unate expansion, as it is explained in section 6.2.3. *Bidirectional fault approximation* is described in detail in that section.

Chapter 4

Circuit approximation using static testability measures

4.1 Introduction

When using approximate logic circuits for fault tolerance, fault sensitivity plays an important role. Whenever a fault is approximated, a difference is generated with respect to the original circuit, which leaves a fraction of it unprotected in case the approximate circuit is used for error detection or masking. In practice, fault testabilities can be used in order to measure fault sensitivity. The impact of approximating a particular fault on the global protection level depends somehow on the testability of that fault. If a given fault has low testability, then there are few test vectors capable of testing that fault, and therefore approximating it has a low impact on global protection against faults. To sum up, the main interest for fault tolerance with approximate logic circuits consist in approximating faults with the lowest testability. In that way, resources are saved while maximizing the error mitigation capabilities.

To achieve this goal, the first solution adopted within this thesis is based on *static testability measures*. In other words, the testability of each fault is obtained by means of an initial analysis of the original circuit, such as fault simulation or fault injection. Then, the results obtained are used to guide the approximation generation. An arbitrary *testability threshold* is applied to discriminate which faults should be approximated first.

Therefore, in this approach the approximation transformations are linked to fault testabilities. This way, it is possible to quantify the impact of each approximation, and to select the most beneficial logic transformations. This is a distinctive key feature over alternative state-of-the-art approaches.

Initially, this heuristic has been used for combinational circuits, for the sake of simplicity. But in practice circuits do have sequential elements. Hence, an extension for sequential circuits has been developed. It must be noted that accurate testability analysis is much more difficult for sequential circuits compared with combinational ones because its response depends on its internal state in addition to the input vectors. This

technique has been validated with experiments for both combinational and sequential circuits.

The structure of the chapter is as follows. Section 4.2 explains how static testability measures are applied in order to generate approximations for combinational circuits. Section 4.3 extends the same procedure to sequential circuits, taking into account the particularities of such kind of circuits. Section 4.4 shows the setup of experiments performed as well as the experimental results. Finally, section 4.5 presents the conclusions of this chapter.

4.2 Approximation of combinational circuits

A first version of the fault selection heuristic has been developed for combinational circuits, for the sake of simplicity. In this kind of circuits, the outputs depends only on the input vectors, which makes testability analysis easier than with sequential circuits.

In this work, static testability measures are obtained by performing a fault simulation on the original circuit, after unate expansion is completed. Stuck-at fault model is applied on this process, which is carried out by means of a parallel simulator. In particular, we have used the fault simulation tool HOPE [45]. Without loss of generality, randomly generated input vectors are used, although it would be useful to apply typical workloads in those cases where they are known beforehand. In addition, the amount of tested input vectors has to be high enough for results to become representative.

Fault sensitivity is linked to the proportion of input vectors capable of testing each particular fault. In this way, HOPE provides information about the faults which are detected for each input vector, as well as the list of undetected faults. All this information is processed and a list of faults within the circuit is generated, along with the number of occurrences for each fault. It has to be noted that HOPE automatically generates a collapsed fault list, i.e., at gate inputs only those faults corresponding to the sensitization value of that gate are tested. This means, as an example, that a stuck-at 0 fault at an AND gate input is never tested, because it is equivalent to a stuck-at 0 fault at the output of that gate.

Once testability measures are obtained, approximations are generated in the following way. First, two replicas of the unate original circuit are generated, and faults within the circuit are split into two groups, according to whether they produce under- or over-approximations. Bidirectional faults are not taken into account. Then a testability threshold is set, which is an arbitrary value between 0 and 1, and every fault whose testability lies under the threshold value is approximated. Faults on stem lines are excluded, only individual branches become approximated. Faults which produce an under-approximation are assigned to one of the replicas of the original circuit, and faults that generate an over-approximation go to the other replica. In the present work, this process is performed by a custom made software program. Finally, both approximate circuits are simplified by means of a synthesis tool, with the aim of removing all logic constants, thus reducing area overhead. In this way, a pair of complimentary unidirectional approximate circuits are generated as result, which can be used to build a masking scheme along with the original circuit or to detect errors that violate the implication relationships among circuits.

The testability threshold provides the required flexibility to the method. Setting a proper testability threshold according to the requirements of the target application is crucial, because it determines the actual trade-off between resources consumption and protection against faults. The lower the testability threshold, the fewer faults are approximated, which implies that approximate circuits are more similar to the original one and therefore the error masking rate is higher. In the extreme case where the testability threshold is set to 0, no faults are approximated and a pure TMR is obtained. On the other hand, a higher testability threshold allows approximating more faults, which results in greater savings in terms of area and power consumption at the expense of reducing robustness. In the extreme case where every fault becomes approximated, approximate circuits are reduced to just logic constants and the ATMR reduces to the original circuit. This is denoted as the *trivial approximation*.

As a consequence of fault approximations, it may happen that some of the approximate circuits outputs are tied to logic constants. In this case, further savings can be obtained by simplifying the voting logic. Figure 4.1a shows an implementation of the majority voter for a single output. Let us assume that the under-approximation output becomes approximated, which implies that one of the three voter inputs is a constant logic 0. Therefore, the voter can be simplified to just an AND gate, as illustrated in Figure 4.1b. Likewise, if the over-approximation output is approximated, the majority voter can be replaced by an OR gate (see Figure 4.1c). Finally, in the case of trivial approximation, 2 out of 3 output signals always present opposite values. This implies that voting logic can be completely removed, as in Figure 4.1d

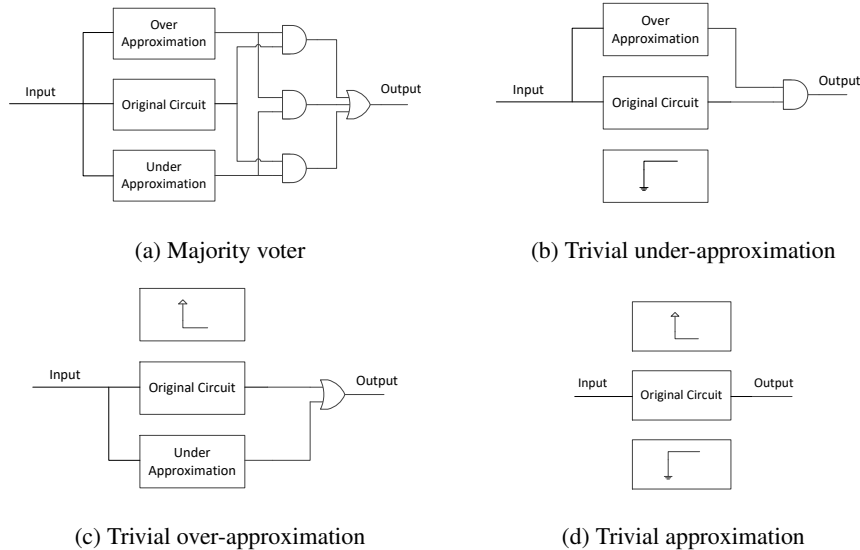
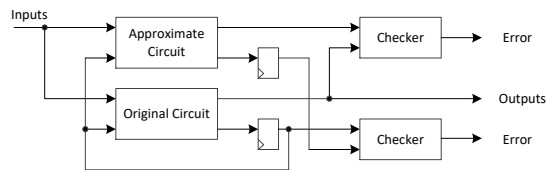


Figure 4.1: Implementation of voting logic

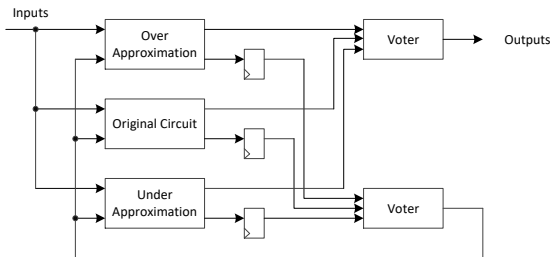
A detailed example of how this approximation generation method is applied is provided in appendix A.3.

4.3 Sequential circuits

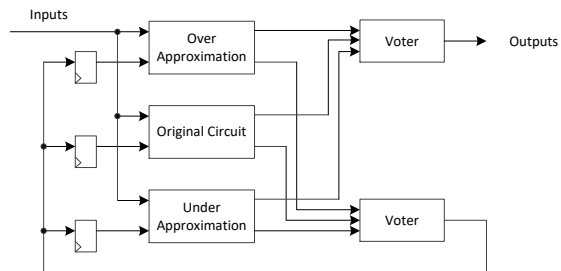
In practice, real application circuits usually have sequential elements. Therefore, it is important to consider the application of the fault approximation criteria to sequential circuits. In principle, extension to sequential circuits is very straightforward. It consist in approximating the combinational part of the circuit, considering the sequential elements as inputs and outputs of the combinational logic. However, the consideration of the sequential nature of the circuit may lead to improved results.



(a) ADWC scheme



(b) ATMR scheme - voter after flip-flops



(c) ATMR scheme - voter before flip-flops

Figure 4.2: Approximate DWC and TMR schemes with sequential circuits

In the first place, approximate TMR and DWC schemes now include sequential el-

ements, and therefore they require some small changes with respect to basic schemes introduced in section 2.3. In the ADWC scheme in Figure 4.2a there is an additional checker to compare flip-flop contents. Besides, the original circuit flip-flops are fed back to both instances, because approximate circuit flip-flops may hold incorrect values. On the other hand, the ATMR scheme includes additional voters for flip-flops, whose results are fed back to all three circuit versions. Voters can be placed either after or before flip-flops (Figures 4.2b and 4.2c respectively). In the last case (with the voter being placed before the flip-flops) each triad of flip-flops could be replaced by just one flip-flop in order to save resources, as all three flip-flops always receive the same assignment, but at the expense of being vulnerable to SEUs. Therefore, this choice requires SEU-resilient flip-flops, such as DICE [9].

Another relevant point is related to how testability measures are obtained. Test of sequential circuits is much more difficult than combinational test, due to the complexity of reaching all possible states. This is often alleviated by means of scan techniques, which consists in making all sequential elements completely controllable and observable. Scan techniques transform the sequential test problem into a combinational one, which is significantly easier. But when the goal is to obtain representative testability measures, combinational testability approaches may lead to inaccurate results. As a matter of fact, it has been proved that certain faults may be testable from a combinational point of view, but sequentially redundant at the same time. Such kind of faults can be classified into three different categories, which are the following: 1) faults that cause the interchange or creation of equivalent states, 2) faults which affect only the transitive fanout of an invalid state and therefore they cannot be reached during normal operation, and 3) faults that just modify the encoding of states. The opposite may also happen, that is, a line may be tested by few combinational input vectors, but these are highly probable in the sequential circuit. The following example illustrates this fact.

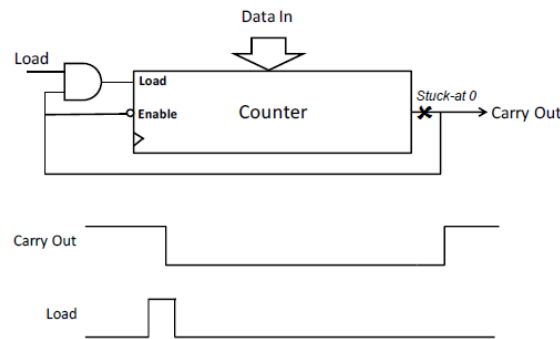


Figure 4.3: One-shot timer

Consider the circuit in Figure 4.3, which represents a one-shot timer. It consists of a counter with Enable and Load inputs. The circuit has only one stable state, which corresponds to the maximum value of the counter. In this state, the Carry Out output

signal is high and the counter is disabled. When the load signal is asserted, a new value is loaded into the counter, the output will be low and the counter is enabled until the maximum value is reached again. Consider the stuck-at 0 fault at the Carry Out output. From a combinational point of view, this fault has low testability, because there is only one single counter state that can activate this fault. Considering the inputs and outputs of the flip-flops as primary outputs and inputs of the circuit, respectively, this fault can be tested only by a single input vector, which corresponds to the maximum value of the counter. Thus, this line could be a good candidate for approximation. The resulting approximate circuit is the trivial one (Carry Out = 0). However, from a sequential point of view, the state required to activate this fault is highly reachable, because it is the only stable state. Actually, the circuit will evolve to the stable state from any initial state and considering a random usage of the circuit, this is expected to be in the stable state more than in any other one. Therefore, the approximation at the output line is a poor approximation when sequential behaviour is taken into account.

In summary, there are two different approaches to compute testability measures for a sequential circuit, each one with pros and cons. Testability in a sequential context is more accurate, but harder to measure. On the contrary, combinational testability measures are easier to obtain, but more imprecise.

In addition, sequential circuits usually have two recognisable parts: the control logic and the data processing unit. The first part manages the circuit execution flow, while the second one is in charge of computing the proper data according with the actual phase within the execution flow. This distinction is relevant here because of the different criticality of errors in both parts. While faults affecting data computation may temporarily affect circuit outputs, faults in the control logic are potentially much more dangerous. They may affect the execution flow, causing either a temporary or a permanent misbehaviour. Therefore, testability of faults in control logic tends to be higher than in data flow. Although approximate logic is not intended as a control flow technique, when applying it with a sequential circuit it is much more likely that control logic is preserved.

As with combinational circuits, resource savings can be achieved when some approximate outputs are tied to logic constants by simplifying the voting logic. But in addition, with sequential circuits further savings can be obtained when flip-flops inputs or outputs become approximated. To this purpose it must be considered that every flip-flop is going to be replaced with a modified flip-flop including a majority voter, which can be placed either before or after the flip-flop, as shown in Figures 4.4a and 4.4b respectively. The actual savings will depend on the position of the majority voter with respect to the flip-flops. Therefore if a flip-flop input is tied to a logic constant, for the scheme of Figure 4.4a it means that majority voter can be simplified as in the left part of Figure 4.4c, as one of its inputs receives a constant value. In this case no flip-flops are removed as their values still depend on the remaining versions of the circuit. However if the voter is placed after flip-flops not only the voter can be reduced, but the correspondent flip-flop can be removed as well, because it always holds the same value (see right side of Figure 4.4c). On the other hand, if a flip-flop output becomes approximated, in the voter-before-flip-flops scheme the flip-flop can be removed, because its value is no longer necessary. However, for the voter after flip-flops scheme no savings are possible, as the flip-flop content is still used to compute the result of

remaining circuits and the voter cannot be simplified. These cases are shown in Figure 4.4d.

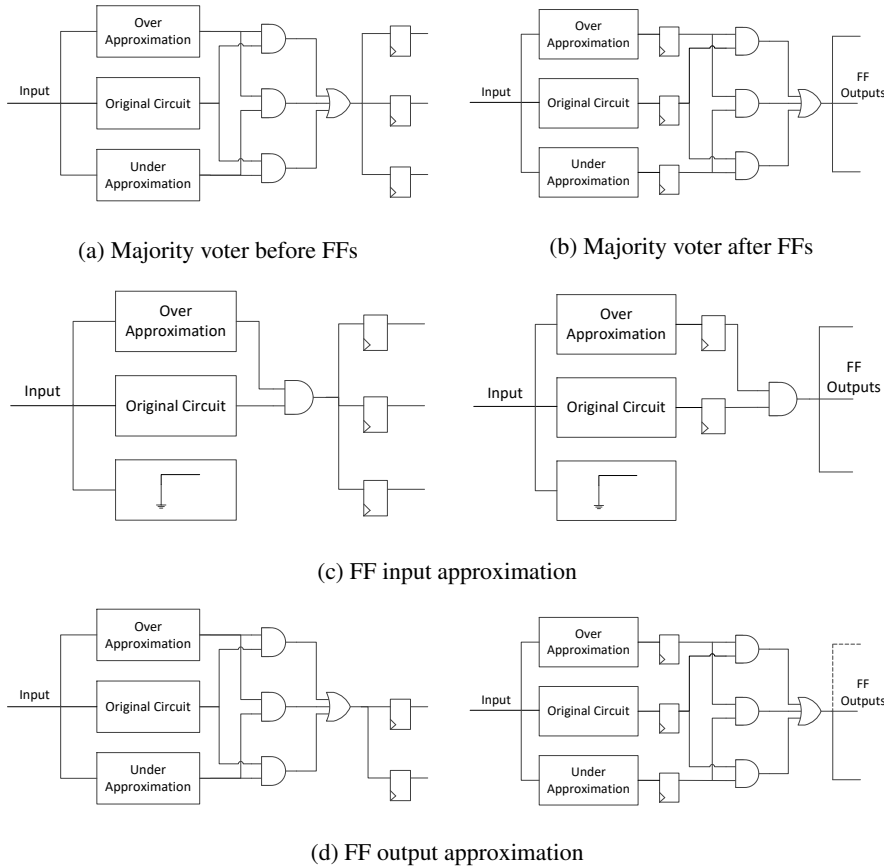


Figure 4.4: Implementation of voting logic for sequential circuits

The generation of approximation for sequential circuits will be illustrated with an example. Benchmark s27 from LGSynth93 set has been selected to conduct this example, which is represented in Figure 4.5. In addition, the figure shows the parity computed for each line within the circuit, according to the method explained in section 3.2. It must be taken into account that the flip-flops in the circuit are considered as inputs and outputs for the purposes of parity computation and approximation generation. Therefore, the circuit can be considered fully unate with the only exception of flip-flop G6, whose input should have a parity value of 0 as it is considered as a circuit output. This would normally require to duplicate some nodes as explained in the unate expansion method (see section 3.4), but in this case a more intelligent solution can be adopted by inserting inverters in G6 input and output as shown in Figure 4.6. In this way, the parity mismatch is solved while the circuit functionality is preserved,

all with a minimum cost. It must be noted that this is an ad hoc solution that can be applied only if the conflicting output or flip-flop is unate on its own but it interferes in the global computation of parities.

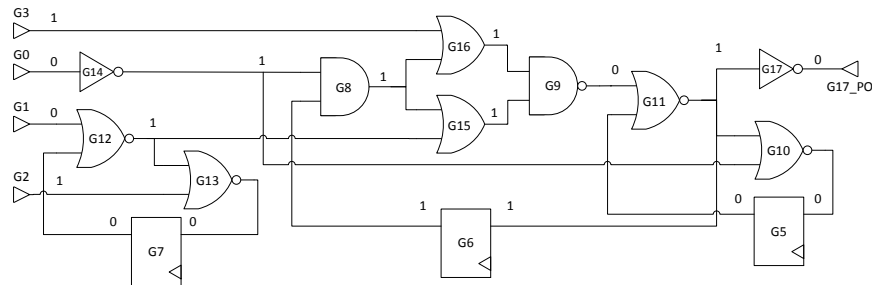


Figure 4.5: s27 benchmark

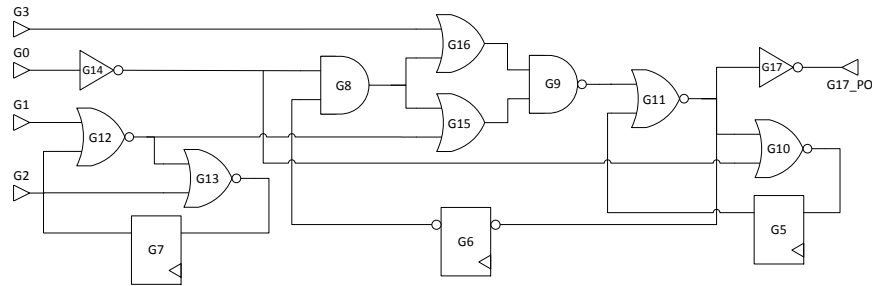


Figure 4.6: Unate version of s27 benchmark

Later, testability measures are obtained. In this example this is performed by a stuck-at fault simulation of the full sequential circuit with parallel simulator HOPE. Table 4.1 shows the sensitivity results for the collapsed fault list of the benchmark by using a sample of 1000 randomly generated input vectors. Fault testabilities are computed as the proportion of input vectors that test each fault. Testabilities are ordered in descending value.

Next, the faults are separated into two groups: those that result in under- approximations and those that result in over-approximations, respectively. In this example, faults are classified as follows:

- Under-approximation faults: G1/0, G5/0, G6/1, G7/0, G9/0, G10/0, G11→G6/1, G13/0, G14→G8/1, G15/1, G16/1 and G17/0.
- Over-approximation faults: G2/0, G3/0, G8→G15/0, G8→G16/0, G11→G6/0, G11→G10/0, G10/1, G12→G13/0, G12→G15/0, G13/1, G14→G10/0 and G17/1.

Fault	Prob.	Fault	Prob.
G17 /0	0.860	G10 /1	0.140
G11 /1	0.860	G14→G10 /0	0.140
G9 /0	0.860	G14 /0	0.140
G8 /1	0.860	G12 /0	0.140
G14 /1	0.857	G2 /0	0.140
G14→G8 /1	0.857	G13 /1	0.140
G15 /1	0.291	G5 /0	0.121
G12 /1	0.291	G10 /0	0.121
G1 /0	0.291	G7 /0	0.070
G6 /1	0.168	G13 /0	0.070
G11→G6 /1	0.167	G11→G6 /0	0.057
G16 /1	0.163	G8 /0	0.057
G17 /1	0.140	G8→G15 /0	0.053
G12→G15 /0	0.140	G8→G16 /0	0.043
G3 /0	0.140	G11→G10 /0	0.039
G11 /0	0.140	G12→G13 /0	0.026

Table 4.1: Results of fault testability analysis for s27

Faults G8/0, G8/1, G11/0, G11/1, G12/0, G12/1, G14/0 and G14/1 correspond to stem lines, and therefore they are not taken into account. Under-approximation faults are assigned to one of the circuit replicas when approximated, while over-approximation faults correspond to the other replica. Therefore two copies of s27 benchmark are initially generated. This way, if no faults were approximated a pure TMR scheme with three identical circuits would be obtained.

Then approximations are generated by assigning an arbitrary testability threshold. Every fault whose testability is below the selected threshold becomes approximated. This result in several lines forced to logic 0 or 1 in the approximate circuits. Finally, constants are removed by simplifying the resulting approximate circuits. In this example four different solutions are shown with threshold values of 5%, 10%, 15% and 20%.

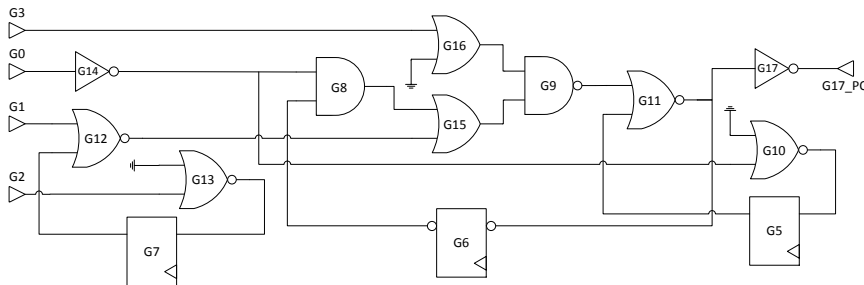
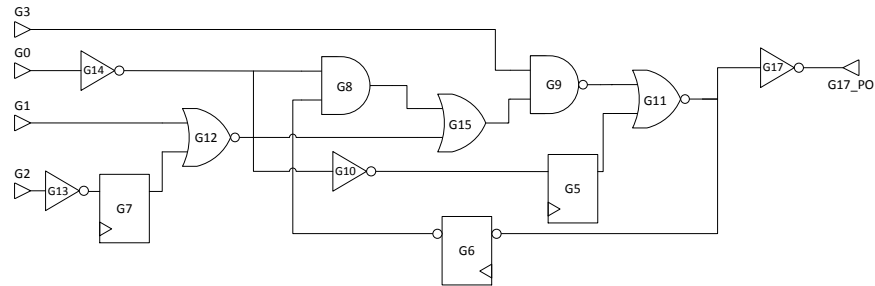
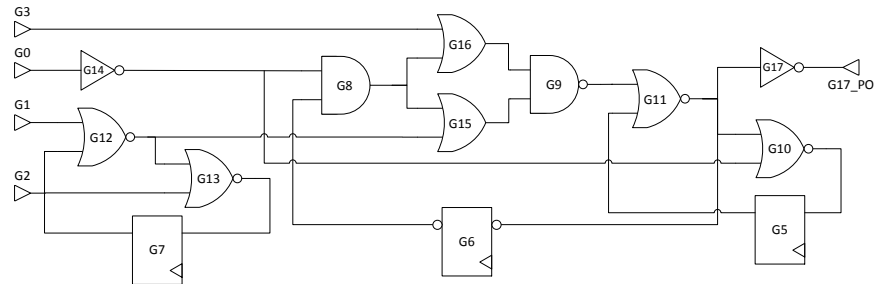


Figure 4.7: Approximation of faults for a 5% threshold



(a) Over-approximation



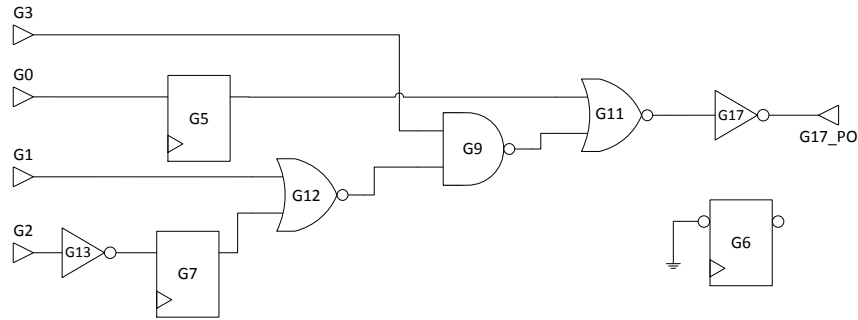
(b) Under-approximation

Figure 4.8: Approximate circuits for a 5% threshold

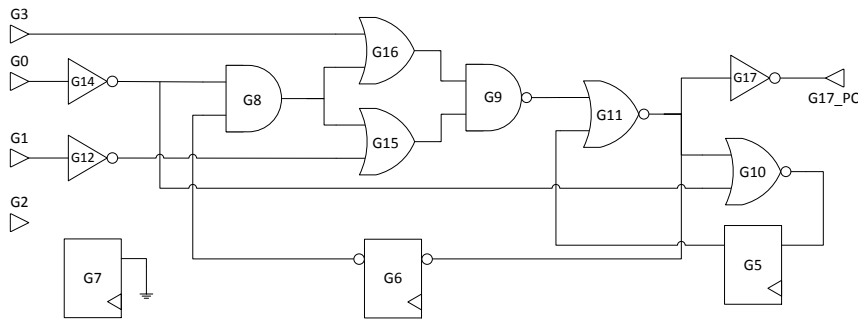
First, for a threshold of 5% faults $G8 \rightarrow G16/0$, $G11 \rightarrow G10/0$ and $G12 \rightarrow G13/0$ are approximated. They all are over-approximation faults and therefore are forced in the same circuit (see Figure 4.7). This circuit should be eventually optimised, thus obtaining the final over-approximation circuit of Figure 4.8a. In this case no under-approximation faults have a testability value under the selected threshold. For this reason, the under-approximate circuit is an exact replica of s27 benchmark as shown in Figure 4.8b. Circuits in Figure 4.8 could be used to build a masking scheme in conjunction with the initial s27 benchmark.

The following threshold value is 10%. Under this level lay faults $G7/0$ and $G13/0$ from the under-approximation side, and faults $G8 \rightarrow G15/0$, $G8 \rightarrow G16/0$, $G11 \rightarrow G6/0$, $G11 \rightarrow G10/0$ and $G12 \rightarrow G13/0$ from the over-approximation side. All these faults are approximated in their respective circuit replicas. After simplification, circuits of Figure 4.9 are obtained. It can be seen that, as a consequence of fault approximations, some flip-flops have been isolated, that is, their inputs are tied to logic constants and their outputs are left open. This is the case of G6 in the over-approximate circuit (Figure 4.9a) and G7 in the under-approximation of Figure 4.9b. This allows to remove both flip-flops in the final implementation of the masking scheme as well as simplifying the voting logic.

Now let us assume a testability threshold of 15%. From the under-approximation



(a) Over-approximation



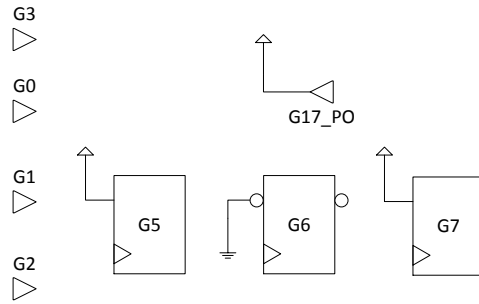
(b) Under-approximation

Figure 4.9: Approximate circuits for a 10% threshold

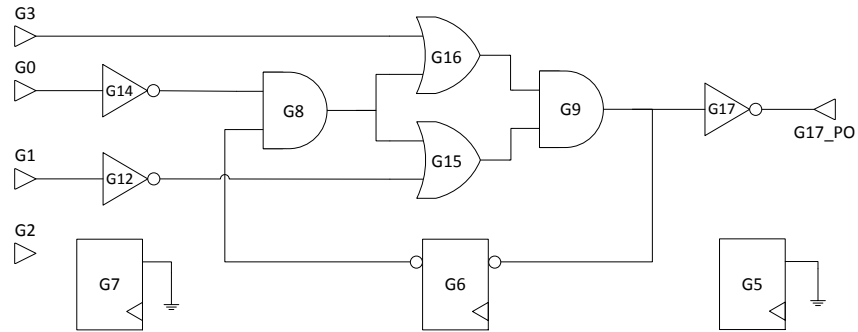
side, faults G5/0, G7/0, G10/0 and G13/0 become approximated. Figure 4.10b shows the resulting under-approximation after simplification of logic constants. Here flip-flops G5 and G6 have been completely approximated. On the other side, fault testability of every over-approximation fault is below the selected threshold. Hence, the over-approximate circuit is reduced to the trivial approximation as shown in Figure 4.10a, where all outputs and flip-flops have been reduced to logic constants.

Finally, consider a testability threshold of 20%. Under-approximation faults which lie under that value are G5/0, G6/0, G7/0, G10/0, G11→G6/0 and G13/0. After simplifying logic constants, circuit of Figure 4.11a is obtained, where all flip-flops have been reduced to logic constants. On the other hand, the trivial over-approximation is generated (see Figure 4.11b).

The results shown previously were obtained by using sequential testability measures. Would there be any differences if combinational measures were used instead of sequential ones? Table 4.2 collects the sensitivity results for the combinational part of s27 benchmark. These results have been obtained with exactly the same sample of 1000

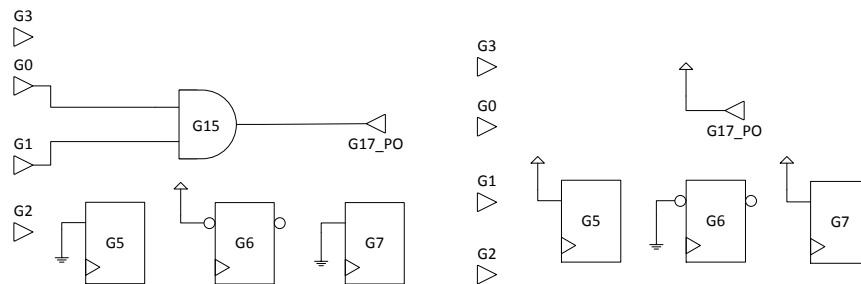


(a) Over-approximation

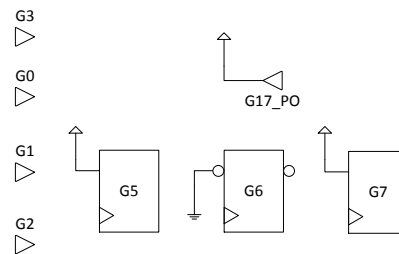


(b) Under-approximation

Figure 4.10: Approximate circuits for a 15% threshold



(a) Under-approximation



(b) Over-approximation

Figure 4.11: Approximate circuits for a 20% threshold

input vectors used for the sequential measures. By comparing these results with those of Table 4.1, noticeable differences can be observed. For example, faults G11→G6/1, G13/1, G10/1 and G10/0 have significantly higher sensitivity with combinational measures. On the other hand, faults G9/0 and G14→G8/1 present much higher testability when sequential measures are computed. As a consequence of these differences, it is expected that the results of logic approximation will somehow be different with respect to the previous ones.

Fault	Prob.	Fault	Prob.
G17 /0	0.860	G6 /1	0.168
G11 /1	0.860	G12→G13 /0	0.157
G11→G6 /1	0.860	G15 /1	0.156
G13 /1	0.662	G17 /1	0.140
G10 /1	0.523	G11 /0	0.140
G14 /1	0.477	G11→G6 /0	0.140
G10 /0	0.477	G7 /0	0.107
G14 /0	0.440	G16 /1	0.078
G12 /1	0.421	G5 /0	0.073
G9 /0	0.383	G12→G15 /0	0.070
G8 /1	0.383	G3 /0	0.070
G14→G10 /0	0.383	G8 /0	0.057
G2 /0	0.343	G14→G8 /1	0.057
G13 /0	0.338	G8→G15 /0	0.051
G1 /0	0.206	G11→G10 /0	0.039
G12 /0	0.193	G8→G16 /0	0.030

Table 4.2: Results of fault testability analysis for combinational part of s27

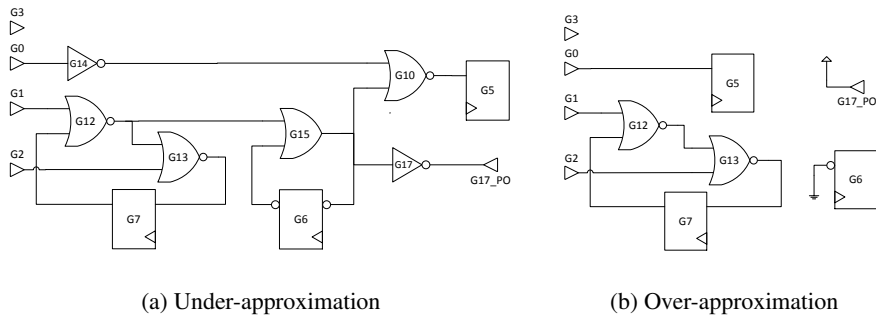


Figure 4.12: Approximate circuits for a 10% threshold - combinational version

For example, consider a testability threshold of 10%. By using the results of Table 4.2, under-approximate faults G5/0, G14→G8/1 and G16/1 become approximated, as well as over-approximation faults G3/0, G8→G15/0, G8→G16/0, G11→G10/0 and G12→G15/0. These transformations eventually lead to circuits of Figure 4.12, which

are rather different from those obtained with sequential testability measures (see Figure 4.9). It is particularly remarkable that now the output of the over-approximation is tied to a logic constant while that is not the case for the circuit on Figure 4.9a. Similarly, with sequential testability measures flip-flop G7 was isolated in the under-approximate circuit (see Figure 4.9b), while it is not now. In addition, it must be noted that output of flip-flop G5 is unconnected in both the under- and over-approximate circuits. Notwithstanding, such flip-flops cannot be removed, as their inputs still affect to the value of the corresponding flip-flop in the original circuit.

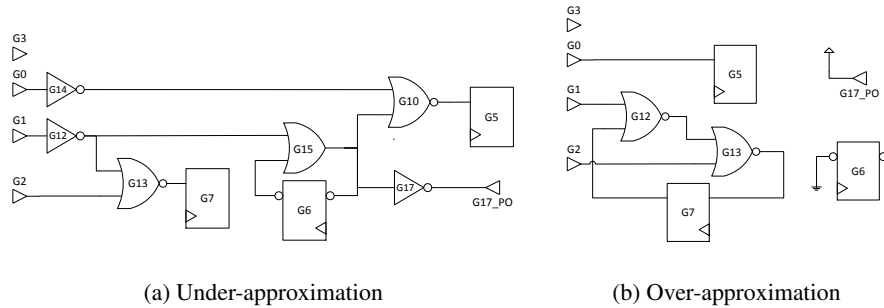


Figure 4.13: Approximate circuits for a 15% threshold - combinational version

In the same way, by applying a testability threshold of 15% circuits of Figure 4.13 are obtained. Over-approximate circuit is identical with respect to the previous step (Figure 4.12b) and different to the trivial approximation obtained with sequential measures for the same threshold (Figure 4.10a). On the other hand, under-approximation has also differences with respect to the sequential counterpart in Figure 4.10b.

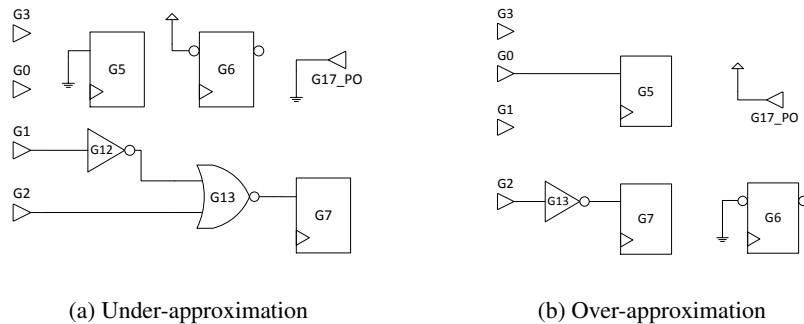


Figure 4.14: Approximate circuits for a 20% threshold - combinational version

Finally, with a threshold value of 20%, circuits of Figure 4.14 are generated. Differences between these circuits and those obtained with sequential testability measures (see Figure 4.11) can be appreciated.

4.4 Experimental results

This section details the experiments performed with the technique described in this chapter, both for combinational and sequential circuits. In addition, results of these experiments are presented and subsequently discussed.

The section is organised as follows. Subsection 4.4.1 explains the experimental set-up, which includes benchmark selection, masking scheme building and fault emulator configuration. Later, subsection 4.4.2 shows the results of experiments performed on combinational circuits and discuss them. Finally, subsection 4.4.3 does likewise for sequential circuits.

4.4.1 Experimental set-up

Experiments with combinational circuits have been conducted with a group of 19 benchmarks from LGSynth93 set. The selected combinational benchmarks have been the following: alu1, alu2, c17, c432, c880, c5315, c7552, cmb, cordic, dalu, des, frg2, i2, i8, i10, s444, term1, unreg and x1. For each one of these circuits, several solutions have been generated with different degrees of approximation. A fault masking scheme has been implemented for this group of experiments, that is, using both over- and under-approximations and a majority voter. Experiments have consisted in two fault emulation campaigns for each solution, namely injecting transient pulses of critical path duration and fixed 300ps lengths.

Later, experiments with sequential circuits have been performed. Up to 11 benchmarks from ISCAS89 set have been selected to run the experiments, which are: s27, s298, s344, s349, s444, s510, s641, s713, s832, s938, s1494 and s3330. Again, several testability thresholds have been applied to each benchmark, by using both combinational and sequential testability measures in order to compare both approaches. The preferred configuration for these experiments has been an error masking scheme with voter placed after flip-flops, such as in Figure 4.2b. Fault emulation campaigns with transient pulses of just the critical path duration have been performed as the experiments.

Fault emulation campaigns have been carried out by means of the Autonomous Multilevel emulation system for Soft-error Evaluation (AMUSE) tool [46], developed by Microelectronics and Applications Group of Electronic Technology department from Universidad Carlos III de Madrid. AMUSE is an emulation-based fault injection system which provides a very high performance with an accuracy close to electric simulation by means of a quantized representation of time, voltage and delays. It consists in a group of hardware modules capable of fully controlling the fault injection campaign over a properly instrumented design, thus speeding up the whole process with respect to software based simulations in several orders of magnitude. AMUSE is intended to be implemented in a configurable logic device, such as an FPGA.

The experimental set-up process for both combinational and sequential circuits is summarized in the diagram of Figure 4.15. The starting point is a description of the target circuit in BENCH format. First, parities within the target circuit are computed according to the algorithm detailed in section 3.2 and, in the case of binate circuits, the unate expansion procedure described in section 3.4 is applied in order to generate a

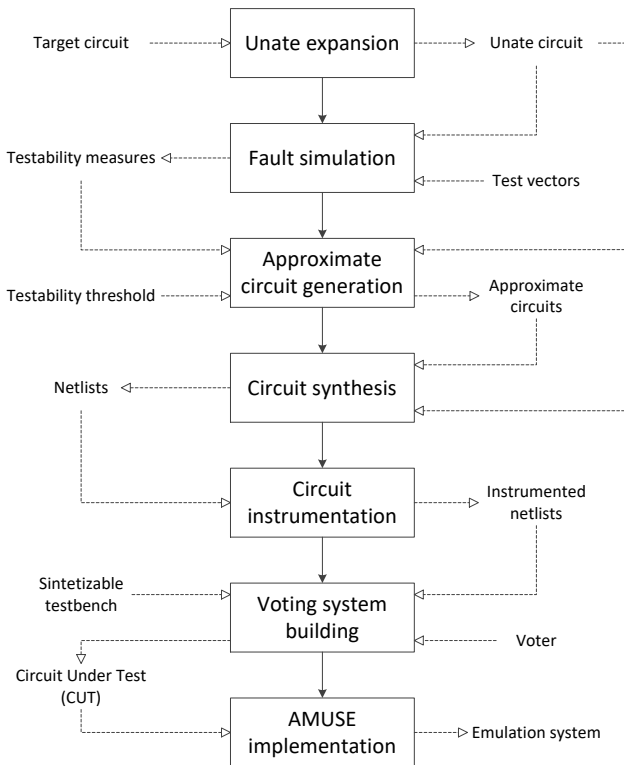


Figure 4.15: Experimental set-up with AMUSE

fully unate circuit in BENCH format. Remember that, in the case of sequential circuits, parities are computed for the combinational part of the circuit, considering the inputs and outputs of flip-flops as respectively the outputs and inputs of the circuit. This step is performed by a custom made BENCH parser.

Next, a fault simulation is performed on the unate version of the target circuit, by means of parallel simulator HOPE. During the simulation 10000 input vectors are applied, which are pseudo randomly generated by a Linear Feedback Shift Register (LFSR) whose size depends on the number of circuit inputs. As a result a file is generated which contains the faults detected by each input vector as well as the list of undetected faults. This file is post-processed by a custom made software program in order to obtain the required testability measures: a list with the number of occurrences for every fault. In the case of sequential circuits, these testability measures are obtained in two different ways by simulating both the whole circuit and its combinational part itself. Thus, testability measures in a sequential and combinational context are respectively obtained.

After fault simulation is performed, testability measures and the unate version of

the target circuit are both used to generate the under- and over-approximate circuits by means of the same custom made bench parser. The user must set a testability threshold that determines which faults are approximated, and subsequently which approximations are generated. Resulting circuits are described in BENCH format. This format does not allow logic constants, and therefore approximation of faults are handled in the following way. Two additional inputs, VCC and GND, are created for the approximate circuits, which corresponds to logic values 1 and 0 respectively. Any time a fault is approximated the corresponding line is then substituted by a connection with either VCC or GND, depending on the fault's value.

After approximations are generated, the approximate circuits along with the unate version of the original circuit are translated into Verilog format by means of ABC synthesizer [47]. In addition, in the case of sequential circuits the combinational part is extracted. The resulting modules are individually encapsulated into a VHDL design which in the case of approximate circuits assigns the proper logic values to VCC and GND inputs, thus making the process user-friendly. This capsule also reintroduces the extracted flip-flops in the case of sequential circuits, and makes them accessible from outside, allowing the insertion of voters for the flip-flops afterwards. After this step, each design is individually synthesized in order to avoid logic sharing. Synthesis are performed with Synopsys for the logic cell library SAED90nm [48]. In addition, voters of proper size and a testbench are synthesized. The testbench is required for AMUSE emulation, and it consists on a LFSR for random number generation, similar to the one employed for simulation with HOPE.

After synthesis, each one of the target circuit versions as well as the synthesized benchmark and voters must be instrumented for AMUSE implementation. Instrumentation consists on substituting circuit components by modified versions that preserve the functionality and in addition support fault injection. This step is performed by means of VIOLIN software, a companion tool of the AMUSE system. It must be noted that faults are only injected in target circuit and its approximations, and therefore voters are instrumented just with the error detection capabilities.

Later, all instrumented designs (testbench, original circuit, approximations and voters) are properly interconnected in one high-level entity known as Circuit Under Test (CUT). Finally, the CUT is implemented in AMUSE system, being ready for the fault injection campaign. This can be performed in two different ways: emulation through implementation in a configurable logic device, or simulation by means of a digital circuit simulator software. The latter option is much slower than the first, but it does not require a final synthesis step which can be time consuming. Therefore it is feasible for small circuits. In fact, in these set of experiments the method has been chosen depending on the target circuit size. Up to a size of 300 logic gates, AMUSE simulation with Modelsim has been performed, while for larger circuits the emulation system has been implemented in a Xilinx Virtex5 XC5VLX110T FPGA by means of Synplify and Xilinx ISE software tools.

On each fault emulation campaign, 10000 randomly generated input vectors have been applied. Faults have been injected in the target circuit as well as its approximations, considering transient pulses with a length equal to the critical path. This can be considered as a non-persistent stuck-at fault or a worst-case SET. These results can be scaled for SETs of a particular pulse width. For combinational benchmarks, an

additional campaign with 300ps transient pulses has been performed. The high performance of the AMUSE system allowed an exhaustive analysis, injecting faults in every gate for every input vector applied.

4.4.2 Results on combinational circuits

Bmk.	Gates	Threshold (%)	Area overhead (%)		Error masking rate (%)	
			Over	Under	Stuck-at	SET
alu1	31	100	0	0	42.98	42.98
		12.5	16.13	51.61	76.03	76.03
		7	61.29	61.29	91.35	91.35
		6.4	87.10	74.19	94.57	94.57
alu2	267	100	0	0	68.75	93.29
		1	19.48	36.33	79.78	95.66
		0.5	47.19	79.03	83.68	96.49
c17	6	100	0	0	18.67	18.67
		13	33.33	100	86.67	86.67
c432	133	100	0	0	69.70	93.62
		8	36.84	40.60	82.98	96.42
		3	56.39	75.94	87.81	97.43
c880	213	100	0	0	56.44	90.45
		6	12.21	22.07	77.07	94.97
		2	13.62	53.05	81.17	95.87
		1	84.51	65.73	93.62	98.60
c5315	1097	100	0	0	56.18	92.90
		10	21.15	36.28	79.12	96.61
		3	74.38	73.93	94.98	99.19
c7552	1007	100	0	0	51.51	93.24
		10	52.83	89.77	94.56	99.30
cmb	31	100	0	0	83.05	88.85
		0.1	0	41.94	93.85	95.95
		0.06	22.58	67.74	97.82	98.56
cordic	141	100	0	0	93.81	96.38
		0.1	3.55	7.80	93.56	96.24
		0.05	37.59	10.64	95.37	97.57
		0.03	60.28	18.44	96.51	98.53
dalu	577	100	0	0	77.24	95.01
		3	0.17	3.81	80.72	95.77
		0.5	62.39	62.05	94.82	98.91
des	2826	100	0	0	30.53	90.28
		3	27.18	24.38	63.60	96.22
		1	70.42	68.22	82.94	98.09
		0.7	93.38	89.77	91.30	98.98

Bmk.	Gates	Threshold (%)	Area overhead (%)		Error masking rate (%)	
			Over	Under	Stuck-at	SET
frg2	666	100	0	0	55.08	87.28
		5	11.56	11.86	74.35	92.13
		1	19.07	14.11	85.70	95.58
		0.5	55.41	18.77	91.18	97.29
		0.3	60.06	56.91	97.26	99.16
		0.1	73.87	70.12	99.23	99.77
i2	89	100	0	0	95.89	97.84
		1	0	4.49	96.26	98.03
		0.01	0	25.84	96.80	98.32
i8	656	100	0	0	43.38	86.97
		0.5	67.07	67.23	81.74	95.80
		0.3	84.45	96.65	84.92	96.66
i10	1346	100	0	0	45.39	92.66
		11	27.56	25.19	77.87	96.46
		7	35.22	30.61	83.77	97.43
		3	41.68	38.34	89.01	98.24
		2	47.10	44.73	90.21	98.42
		1	55.13	59.14	92.05	98.75
		0.5	63.45	62.70	93.10	98.93
		0.2	75.41	72.36	95.29	99.27
		0.1	83.43	83.66	96.75	99.51
s444	96	100	0	0	38.10	65.71
		13	44.79	26.04	75.31	87.01
		3	66.67	67.71	92.31	96.15
term1	208	100	0	0	87.11	94.57
		0.3	19.71	21.15	97.09	98.78
		0.1	38.46	49.52	98.80	99.49
unreg	83	100	0	0	21.12	30.81
		15	40.96	19.28	85.37	87.16
		6.3	75.90	42.17	91.33	93.91
x1	381	100	0	0	70.29	88.42
		1	15.49	20.21	93.90	97.17
		0.1	41.73	41.73	95.01	97.69

Table 4.3: Experimental results - static testability measures with combinational circuits

Results of fault injection campaign on combinational circuit are collected in Table 4.3, grouped by benchmark. The first two columns show respectively the name of each circuit and its size, measured in number of logic gates. The third column contains the different testability thresholds applied on each benchmark. Next, results of experiments are given. First, area overheads due to each approximate circuit are shown, which are referred to the area of the target circuit's unate version. Last, rate of masked faults for both pulses of critical path duration and SETs of 300ps is reported. In those cases where critical path is shorter than 300ps both results coincide.

These results show that error masking with approximate circuits is an interesting technique, capable of reaching good enough protection level against faults with reasonable area overheads. For example, x1 benchmark with a 1% testability threshold masks 94% of stuck-at faults and 97% of SETs with an area overhead smaller than 36%. Another relevant cases are cmb at 0.1% (with 98% of masked stuck-at faults and 99% of SETs for a 42% area overhead), frg2 at 0.5% (error masking rates of 91% and 97% respectively with an additional area of 74%), or term1 at 0.3% (protection of 97% and 99% against stuck-at faults and SETs respectively with an area overhead of 41%). Apart from that, it can be appreciated that error masking rate for SETs is noticeable higher than for faults with the duration of the critical path, in general. More over, these differences tend to grow for high testability thresholds, and with large circuits, which usually have longer critical paths than smaller circuits. This is due to the fact that, the shorter the transient pulse duration is with respect to the critical path, the more relevance have temporal and electrical masking effects. Therefore, results with stuck-at faults can be considered as the worst case.

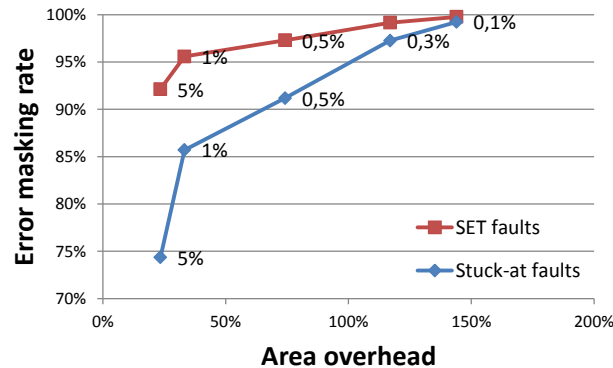


Figure 4.16: Scalability of technique for frg2 benchmark

In order to show the scalability of the technique, two benchmarks have been selected to graphically represent the error masking rate against the area overhead. In particular, experimental results of frg2 and i10 benchmarks are shown in Figures 4.16 and 4.17 respectively. Area overhead in these charts is the sum of over- and under-approximate circuits. The marks denote the testability threshold applied on each point. In both cases, as long as the testability threshold goes down, the error masking rate increases, as well as the size of approximate circuits. Therefore, there is a correlation between the level of protection against faults and overheads. In addition, it can be appreciated that the error masking rate is higher when pulses are shorter, because it is more difficult that the transient pulse effectively propagates through the circuit to the outputs.

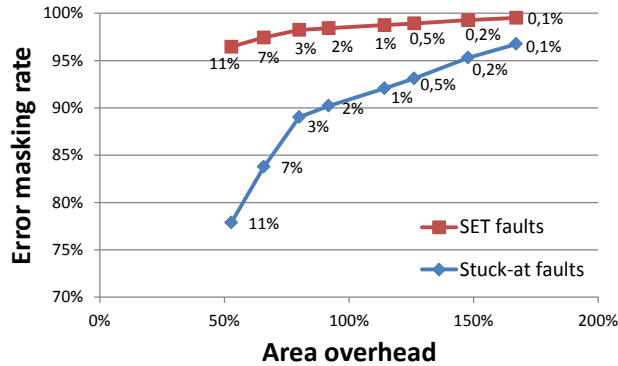


Figure 4.17: Scalability of technique for i10 benchmark

4.4.3 Results on sequential circuits

Table 4.4 contains the results of experiments on sequential circuits. The first three columns indicate, from left to right, the name of each benchmark, and its size expressed in number of logic gates and flip-flops. Next, the testability threshold applied on each experiment is given. Later, experimental results with approximate circuits generated by means of testability measures in a sequential context are shown, including the area overhead due to both over- and under-approximations, as well as the error masking rate against non-persistent stuck-at faults. Area overheads are computed with respect to the unate version of the target circuit. Finally, the same results for the solutions generated with combinational testability measures are shown.

From these data it can be appreciated that different results are obtained depending on which kind of testability measures are used in order to generate approximate circuits. In general, area overhead in sequential mode is smaller than in combinational mode for the same threshold level. Such is the case of s641 benchmark approximated at 0.1% (164% area overhead against 176%) or s3330 benchmark with a threshold of 3% (42% area overhead against 56%). Some cases can be found where both approaches give similar protection against faults, but the sequential approach usually generates smaller circuits than the combinational one. Such is the case of s444 benchmark approximated at 13%, which presents 4% against 71% area overhead. Even there are some results in which sequential approach presents higher protection against faults with greater overheads, as for example s344 benchmark with 8% threshold (89% error masking rate and 82% area overhead against 83% and 58%). On the contrary, there are cases where combinational approach gives better results, such as s510 at 10% with a 116% area overhead against 168% for a similar error masking rate. These results show that when applying fault mitigation techniques in a sequential circuit considering just the combinational behaviour may produce suboptimal results.

Bmk.	Gates	FFs	Th. (%)	Sequential measures			Combinational measures		
				Area ov. (%)		EMR (%)	Area ov. (%)		EMR (%)
				Over	Under		Over	Under	
s27	10	3	100	0	0	52,12	0	0	31,34
			17	30	0	69,87	20	10	64,55
			10	40	20	76,67	30	90	84,55
			5	90	50	89,91	50	100	88,33
s298	72	14	100	0	0	56,14	0	0	47,81
			10	16,67	2,78	70,16	13,51	6,76	64,18
			4	27,78	11,11	93,01	33,78	22,97	74,57
			0,2	47,22	33,33	97,99	100	100	100
s344	105	15	100	0	0	49,39	0	0	41,43
			20	23,81	27,62	77,18	8,57	8,57	57,45
			8	41,90	40	89,46	30,48	27,62	83,13
			3	61,90	56,19	93,98	76,19	83,81	96,61
s349	105	15	100	0	0	50,18	0	0	41,02
			20	23,81	28,57	77,66	8,65	7,69	56,42
			6	44,76	47,62	92,03	55,77	46,15	88,10
			2	63,81	80,95	97,06	96,15	96,15	98,39
s444	103	21	100	0	0	68,43	0	0	38,10
			13	2,91	0,97	75,93	44,79	26,04	75,31
			3	4,85	0,97	76,42	66,67	67,71	92,31
			0,01	9,71	1,94	81,19	100	100	98,24
s510	174	6	100	0	0	52,30	0	0	53,55
			50	58,05	63,22	87,95	8,94	2,23	62,67
			30	71,26	75,29	91,07	34,64	23,46	79,71
			10	85,63	82,18	95,46	68,72	47,49	95
s641	114	19	100	0	0	54,67	0	0	32,50
			10	11,40	14,91	82,53	31,82	30	75,07
			3	35,96	25,44	95,02	59,09	59,09	90,84
			0,1	87,72	76,32	99,31	90,91	85,45	99,17
s713	115	19	100	0	0	54,55	0	0	32,31
			15	0,87	13,91	74,04	13,04	14,78	54,09
			10	9,57	14,78	81,03	30,43	26,09	71,09
			2	44,35	18,26	94,52	64,35	57,39	89,65
0,01	90,43	100	99,67	92,17	101,74	98,42			
s832	191	5	100	0	0	67,01	0	0	65,94
			5	2,09	5,24	77,08	3,66	2,62	69,53
			1	4,19	19,90	79,58	30,89	26,18	88,44
			0,1	14,14	26,18	88,07	88,48	76,44	99,30
s938	210	32	100	0	0	56,11	0	0	55,76
			5	0,48	3,33	58,19	21,05	10,05	75,37
			1	2,38	9,52	63,18	42,58	23,44	96,52
			0,01	11,90	30,48	77,78	68,90	89	99,98

Bmk.	Gates	FFs	Th. (%)	Sequential measures			Combinational measures		
				Area ov. (%)		EMR (%)	Area ov. (%)		EMR (%)
				Over	Under		Over	Under	
s1494	430	6	100	0	0	47	0	0	63,77
			10	1,16	5,58	54,66	1,38	0,92	65,88
			3	5,35	7,21	63,67	13,79	7,36	75,21
			0,1	5,35	10	63,89	91,72	93,10	99,63
s3330	697	132	100	0	0	62,96	0	0	53,46
			10	10,90	12,20	81,01	13,13	14,14	71,84
			3	21,66	20,37	85,69	31,75	24,24	81,64
			0,1	42,04	45,91	92,79	72,73	73,45	99,07

Table 4.4: Experimental results - static testability measures with sequential circuits

Figure 4.18 graphically shows the difference between the sequential and combinational approaches for s713 benchmark. The chart plots the error masking rate with respect to the sum of the area overheads of both approximate circuits. The marks denote the testability threshold applied on each case. It can be observed that for the same threshold value the sequential approach reaches higher protection against faults at lower cost. Therefore, for this particular case approximation generation by means of sequential testability measures is more efficient than the combinational approach.

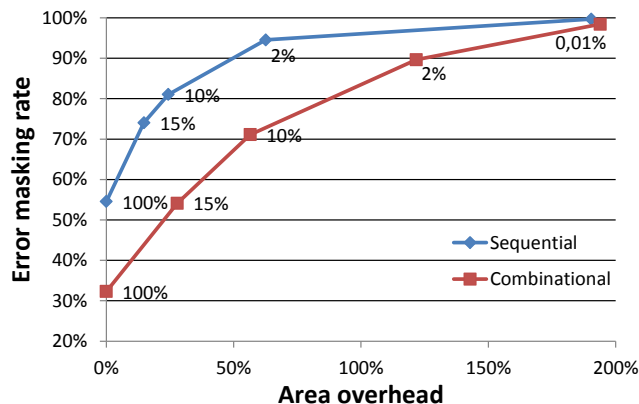


Figure 4.18: Comparative between sequential and combinational approximations for s713 benchmark

4.5 Conclusions

This chapter presents the first fault selection criteria developed within this thesis for approximate logic circuit generation applied to fault mitigation. This approach is based

on static testability measures, and it consists in performing an initial fault simulation of the target circuit's unate version with the aim of obtaining representative data about sensitivity of faults within the circuit. Later, a testability threshold is set and every fault whose sensitivity lies below that value becomes approximated, thus obtaining a pair of complementary approximate logic circuits. By varying the testability threshold a wide range of solutions can be obtained between pure TMR and trivial approximation, with different trade-offs between fault mitigation capabilities and overheads. Initially, this approach has been developed for combinational circuits.

The technique has been validated through fault emulation experiments. Results are reasonably good, and they show that this approach is widely scalable, and therefore capable of adapting to different requirements in terms of reliability or costs. Apart from that, the technique can be implemented with a low computational cost.

However, the fault selection criteria based in static testability measures has some drawbacks. The first one is that the range of feasible solutions depends on the initial structure of the target circuit, as it influences the testability measures and the set of possible logic transformations. This issue can be addressed by means of an initial synthesis and logic optimization step. Another question refers to the static nature of testability measures. Whenever a fault is approximated, testability of the rest of faults may change, and therefore simulation results become less and less representative as more logic transformations are performed, which may subsequently lead to suboptimal solutions. This issue is discussed in detail in the next chapter. Finally, experimental results show that there is little correspondence between applied testability thresholds and effective costs or fault mitigation capabilities of generated solutions. Of course there is a qualitative correlation between them, but the testability threshold is not a good estimator of any relevant metric, and therefore achieving an specific error masking rate or area overhead can only be done by trial-and-error.

An extension for sequential circuits has later been developed, motivated by the fact that real application circuits usually have sequential elements. This has been done in a straightforward manner, by approximating the combinational part of the circuit and considering flip-flops as inputs and outputs of the combinational logic. In addition, the implementation of fault detection and correction schemes suffer small changes compared with combinational version, allowing the comparison or voting the contents of flip-flops in addition to outputs. A major concern about this approach is how testability measures are obtained. It is widely known that testability analysis of sequential circuits is a difficult task, due to the complexity of reaching certain states. On the contrary, testing just the combinational part of a sequential circuit is much easier, but the results may be inaccurate. Experimental data show that there are noticeable differences between the results obtained with both approaches.

With the idea of addressing some of the issues of the fault selection method based in static testability measures, an improved criteria has been developed which makes use of probability analysis in order to estimate fault sensitivities. In this way, testability of faults can be recomputed each time a logic transformation is performed. In addition, probability computations can be used to keep an estimation of the final error protection level. This improved approach is described in chapter 5.

Chapter 5

Circuit approximation using dynamic testability measures

5.1 Introduction

The approximation generation method based on static testability measures introduced in chapter 4, although it is widely scalable and has low computational cost, presents several drawbacks that should be addressed. First, the testability threshold, which is the metric used to discriminate which faults are approximated, is not a good estimator of the final error protection level. Moreover, the most important flaw is the fact that, whenever a fault is approximated, the testability of the rest of the faults may change, thus making static measures less accurate as more faults become approximated. The following example illustrates this.

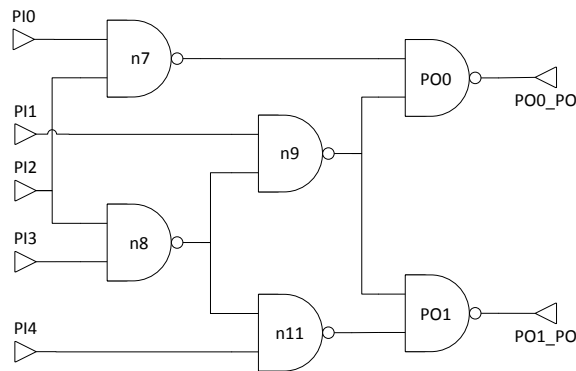


Figure 5.1: c17 benchmark

Consider the circuit of Figure 5.1, which represents the c17 benchmark. Table 5.1 shows the results of testability analysis under several conditions. Columns labelled with "Initial prob." contains the sensitivity results over the target circuit, which coincides with the static testability measures. Those measures have been obtained through fault simulation with 1000 randomly generated input vectors. Let us assume that the first approximated fault is PI2→n7/1, the least testable fault. The resulting circuit is shown in Figure 5.2a. After performing this transformation, fault testabilities are computed again in the new circuit by means of fault simulation with the same 1000 input vectors, and their values are presented in the columns denoted as "1st approx". By comparing these results with those corresponding to the initial circuit, several changes can be observed. For example, faults PO0/0, n7/1 and PI0/1 now have higher testability, while faults PO0/1, n9→PO0/1 and PI1/1 are less testable. Now suppose that the next fault with the lowest testability is approximated, which is the fault n8→n11/1 according to static testability measures. This transformation produces an over-approximation exactly as the previous one, so both can be paired together, resulting in the circuit of Figure 5.2b. If a fault simulation with the same 1000 input vectors is performed in this new approximate circuit, results of columns "2nd approx" are obtained, which are completely different to the initial testability measures.

Fault	Initial prob.	1 st approx	2 nd approx	Fault	Initial prob.	1 st approx	2 nd approx
PO0 /0	0.590	0.704	0.704	PI2→n8 /1	0.200	0.200	0.105
n9 /0	0.572	0.293	0.446	n9→PO1 /1	0.198	0.198	0.198
n8 /0	0.566	0.566	0.293	n8 /1	0.198	0.198	0.098
PO1 /0	0.566	0.566	0.700	n7 /1	0.196	0.310	0.620
PO1 /1	0.434	0.434	0.300	PI3 /1	0.196	0.196	0.103
PO0 /1	0.410	0.296	0.296	PI0 /1	0.188	0.296	0.296
n9 /1	0.361	0.293	0.293	PI4 /1	0.180	0.180	0.300
n9→PO0 /1	0.318	0.181	0.181	n11 /1	0.172	0.172	0.612
PI1 /1	0.318	0.261	0.261	n8→n9 /1	0.135	0.135	0.098
PI2 /0	0.295	0.198	0.098	n8→n11 /1	0.134	0.134	-
PI2 /1	0.286	0.200	0.105	PI2→n7 /1	0.114	-	-

Table 5.1: Results of fault testability analysis for c17

This phenomenon may lead to suboptimal solutions in case static testability measures are applied to generate approximate logic circuits. It may happen that a fault initially has a low testability, but it starts to grow as long as faults are approximated, up to the point that it no longer is a good candidate for approximation for a given testability threshold. Conversely, it may happen that a fault is not approximated due to its high initial testability, but in the end its value lies under the selected testability threshold. In order to solve this problem an alternative method has been developed, which consists in approximating faults one by one, and every time a fault is approximated, testability of the remaining faults are recomputed. Thus, every sensitivity change due to logic transformations is taken into account, and the most promising candidate can be selected on every step. This approach is said to be based on *dynamic testability measures*.

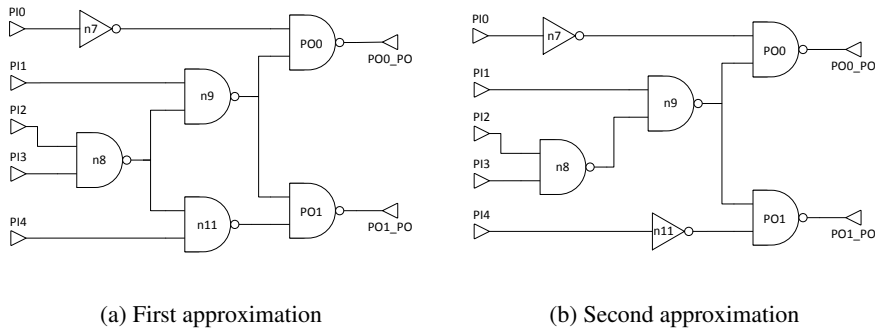


Figure 5.2: c17 over-approximations

In this new approach, testability measures can still be computed by means of fault simulation. But performing a fault simulation per approximated fault imposes a very high computational cost, specially in the case of large circuits. In order to alleviate this cost, testability measures could be recomputed after a given number of faults are approximated, for example every 5 or 10 faults, thus reducing accuracy in favour of computational cost.

However, there are alternative methods for obtaining testability measures. In this chapter an approach is proposed, that is based on implication reasoning and probability computation. First, each fault is properly justified with the aim of finding the necessary set of assignments that allows the propagation of the fault. Then signal probabilities are computed based on the deduced assignments, which are finally combined in order to obtain the correspondent fault probability. With this approach, the sensitivity of each fault is linked to the analytical probability of testing that fault. In addition, sensitivity results obtained with this method are deterministic, as opposed to static testability measures obtained by simulation, where it may be slight deviations caused by random generation of input vectors.

Therefore, an iterative process is adopted here. In each iteration, fault probabilities are recomputed and the best candidate is selected and approximated, if any. As an advantage, this method allows to partially reuse computations from one step to another and incrementally update the testability measures, as the difference between two steps is assumed to be small, thus reducing computational cost.

Moreover, these probability computations can be used to keep an estimation of the effect of approximations on the global error rate. In practice, this estimation serves as the ending condition of the iterative process. In other words, the user now specifies a certain *error target*, and the algorithm starts approximating faults until the estimated error rate meets that value. In theory, the resulting approximate circuits should provide an error rate similar to the specified target value when implemented in an error masking scheme along with the target circuit. This approach has been validated through fault simulation experiments.

Additionally, advantage can be taken on the fault justification process. The set of deduced assignments for each fault can be used to implement a new type of logic trans-

formation, which consists on replacing lines with others within the same circuit instead of assigning logic constants. This technique receives the name of *node substitution*, and allows to generate solutions which are not reachable by means of fault approximation. In practice both approaches can be combined to widen the range of feasible solutions. This approach has also been validated through fault simulation experiments.

The rest of this chapter is organized as follows. Section 5.2 introduces the main concepts and techniques about computation of fault probabilities. Later, section 5.3 deals with how this probability computation is applied to approximate circuit generation. Next, node substitution technique is explained in section 5.4, which is an improvement over the approach based in dynamic testability measures. Section 5.5 describes the experiments performed in relation with the techniques described in this chapter and presents its results. Finally, section 5.6 summarizes the main conclusions of this chapter.

5.2 Fault probability computation

The basic notions about signal probability computation were established by Parker and McCluskey in [49]. Signal probability $P(s = v)$ refers to the probability of evaluating signal s to the value v . If that condition is denoted as the assignment $a = \{s = v\}$, then we can likewise speak in terms of probability of assignment $P(a)$. The joint probability of a set of assignments $A = \{a_1, a_2, \dots, a_i, \dots\}$, $P(A)$, is the probability of simultaneously satisfying every assignment in A . This can be computed by using the probability chain rule:

$$P(A) = P(\cap_{i=1}^n a_i) = \prod_{i=1}^n p(a_i | \cap_{j=1}^{i-1} a_j) \quad (5.1)$$

In addition, if every assignment a_i is not dependent from the others, the probability of the whole set of assignments can be computed as the product of probabilities of each assignment, thus making:

$$P(A) = P(\cap_{i=1}^n a_i) = \prod_{i=1}^n p(a_i) \quad (5.2)$$

Signal probabilities in a combinational network can be computed by traversing the circuit from inputs to outputs [49]. Initial values are assigned to input probabilities, typically either 0.5 or a value which models a typical workload, in case it is known beforehand. Then at each node, signal probability at the output is computed as a function of the probability of input signals and the node type. Later, testability of each fault is computed by combining certain signal probabilities according to the controllability and observability conditions of the fault [50].

Testability analysis of circuits has been widely studied, and multiple techniques have been developed within this subject. Among them, the simplest approach for fault probability computation is the Computation of Probabilities (COP) algorithm [50]. This technique computes signal probability at the output of a node by direct application of intersection, union and complementary properties, assuming that inputs are

independent. Table 5.2 shows the formulas for signal probability computation according to COP algorithm for some basic logic gates. All probabilities in the table are referred to the correspondent signal being evaluated to value 1. Formulas for NAND, NOR, XNOR and more complex gates can be deduced from these ones.

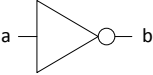
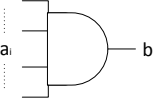
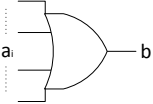
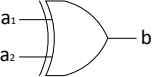
Logic gate	Scheme	Formula
NOT		$P(b) = 1 - P(a)$
AND		$P(b) = \prod_{i=1}^n P(a_i)$
OR		$P(b) = 1 - \prod_{i=1}^n (1 - P(a_i))$
XOR		$P(b) = P(a_1)(1 - P(a_2)) + (1 - P(a_1))P(a_2)$

Table 5.2: Probability computation with COP

COP algorithm has the advantage of being simple and intuitive, and it requires very low computational cost. Nevertheless, probabilities computed by means of COP algorithm are generally inaccurate. This is due to the reconvergent fanout problem, which contradicts the assumption of every signal being independent from the others.

Other approaches try to solve this problem. For instance, the PREDICT algorithm [51] introduces the concept of super gates, thus achieving more precise estimations at the expense of a higher complexity and computational effort. The testability analysis tool SCOAP [52] does not make use of probability estimations, but instead computes the costs of controlling and observing each node within a given circuit, which are related with intrinsic testability. The cutting algorithm [53] computes the lower and upper bounds for each signal probability -instead of exact values- by cutting all multiple fanout branches in the circuit, converting it in a tree network where probabilities are far easier to calculate. Parker and McCluskey propose an algorithm [49] to compute signal probabilities by analysing the logic equations that a given logic circuit implements, although its application may soon become infeasible for large circuits. The improved cutting algorithm [54] combines the cutting and Parker-McCluskey algorithms in order to reduce the computed probability intervals with respect to the original cutting algo-

rithm. The algorithm chooses a subset of fanout branches for cutting and the others for formal analysis.

Among all existing testability analysis techniques, it is worth mentioning the Testability Analysis by Implication Reasoning (TAIR) algorithm [55] as the base of the dynamic probability computation developed within this work. It makes use of implication reasoning to deduce interdependencies between signals. The TAIR algorithm departs from signal probabilities computed with COP and then calculates fault probabilities, refining them by applying correlating factors according to the assignments deduced during implication of each fault.

5.3 Approximation generation with dynamic measures

This section explains how dynamic testability measures are computed and how they are applied to approximation generation. Similarly to the TAIR algorithm [55], this approach makes use of implication reasoning to deduce interdependencies between signals. The main difference is that here signal probabilities are computed after implication of each fault, taking into account the set of deduced assignments. Therefore, there is no need of applying correlation factors to derive fault testabilities from signal probabilities. The different concepts and procedures developed for this approach are progressively introduced in the following subsections. Subsection 5.3.1 tells how faults are justified by using implications. Later, subsection 5.3.2 deals with how fault probabilities are computed based on deduced assignments during justification, and how probabilities are updated every time a fault is approximated. Next, subsection 5.3.3 explains how global error rate is estimated thanks to computed fault probabilities. Finally, subsection 5.3.4 shows the whole algorithm designed to generate approximate circuits with dynamic testability measures.

It is important to remark that this approach is intended for unate circuits. If the target circuit is binate, logic transformations introduced in section 3.4 have to be applied in order to obtain a fully unate initial circuit.

5.3.1 Fault implication

In this approach, the first step towards the computation of probability of a fault consists in its implication, i.e., deducing its Mandatory Assignments (MAs) and checking their consistency [56]. The *mandatory assignments* of a given fault are the set of assignments required to test that fault. Any input vector must satisfy the mandatory assignments of a fault in order to be able to test it.

Several concepts must be introduced before properly explaining the implication method. The *transitive fanout* of a given fault f is the set of all reachable points for fault f , or in other words, the set of all propagation paths from the fault injection site to the primary outputs of the circuit. A node n is a *dominator* of fault f if every propagation path in the transitive fanout of f goes through n , i.e., n is common to all those propagation paths. There are two types of dominators. *Static dominators* are those which can be simply deduced from the topology of the circuit. On the contrary,

dynamic dominators are those which appear during the implication process, as a consequence of some propagation paths being blocked due to deduced assignments. Given a node n belonging to the transitive fanout of a fault f , the *side inputs* of n with respect to f are all the immediate inputs to n which are not in the transitive fanout of f . The *controlling value* of a logic gate is the logic value which, when assigned to one of its inputs, uniquely determines the output value regardless of the values of the remaining inputs. From this definition it can be deduced that for an OR gate the controlling value is 1, and 0 for an AND gate. The inverse of the controlling value is denoted as the *sensitizing value*.

The rest of this section is divided in several subsections, explaining the functioning of the proposed fault implication method over cases with increasing complexity. Starting with the case of faults which propagate through just one output (subsection 5.3.1.1), then we move to the situation with multiple outputs (subsection 5.3.1.2). Later, the conditions of previously approximated faults are introduced in the implication (subsection 5.3.1.3), and finally the proper is done in the case of multiple outputs (subsection 5.3.1.4).

5.3.1.1 Basic implication mechanism

The fault implication process is based on the implication engine developed in [57] for redundancy identification. Basically, implication of a fault consists in applying the controllability and observability conditions necessary to excite the fault and propagate it to any circuit output, respectively. The *controllability condition* is the MA which assigns the *activating value* in the fault injection point, that is, the logic value opposed to the faulty one. On the other hand, the *observability conditions* are the MAs which assign the respective sensitizing values to the side inputs of all static dominators of the fault. Once these assignments have been deduced, they are propagated through the circuit by *direct implication*, i.e., either deducing the output value of a gate from its inputs values (which is known as *propagation*) or vice-versa, deducing input assignments from its output value (denoted as *justification*). It may happen that, as a consequence of the implication process, some propagation paths become blocked. As a result, new dynamic dominators may appear for that fault. Whenever a dynamic dominator is found, observability conditions are immediately applied to it, and then the implication process is resumed. This is performed until no more assignments can be deduced. Additional mandatory assignments can be found by using more sophisticated methods such as recursive learning [58], although they require a considerable computational cost.

As previously said, this procedure was originally intended in [57] for redundancy identification. If there is an inconsistency detected during the implication of a certain fault, then that fault is redundant. On the contrary, the main objective in this work is estimating fault probabilities. Therefore, the fault implication process is intended here to find the simplest Set of Mandatory Assignments (SMA) that satisfy the fault controllability and observability conditions with the minimum number of signal dependencies, that is, the *justification frontier* of the fault. The initial SMA results from the direct application of controllability and observability conditions. Whenever a mandatory assignment is justified backwards in the circuit, that assignment is removed from the SMA and replaced with the new deduced assignments, if any. If a dynamic dominator

is found, its observability conditions are added to the SMA. When the implication process is complete, the final SMA will contain those assignments which cannot be further justified, which becomes the justification frontier of the fault, also known as Justified Set of Mandatory Assignments (J-SMA).

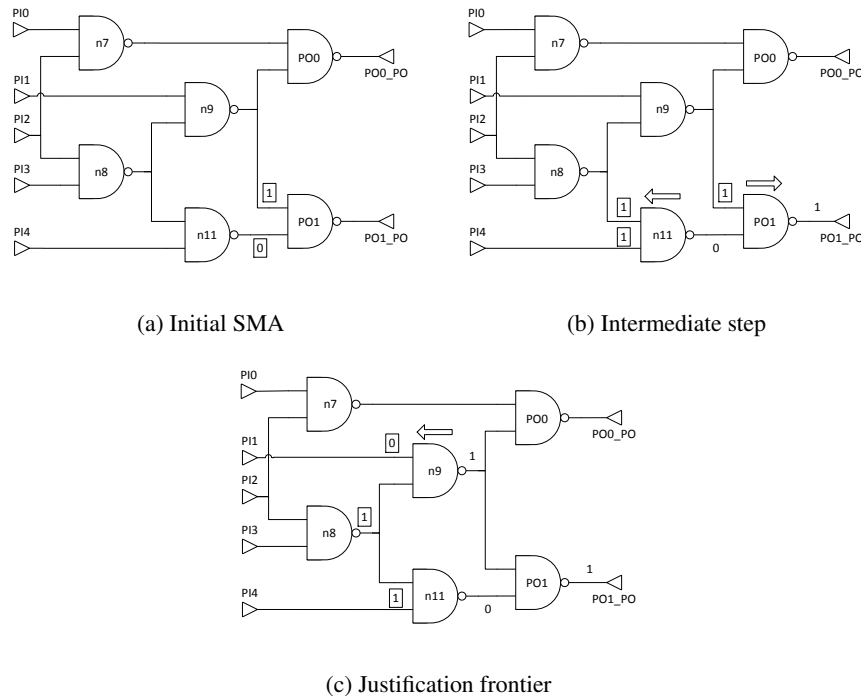


Figure 5.3: Implication of fault $n11/1$ in $c17$ benchmark

To show an application example of the implication process, let us imply fault $n11/1$ from $c17$ benchmark (Figure 5.1). First of all, controllability and observability conditions are applied. On one hand, the controllability condition is the one that excites the fault, in this case the MA $n11=0$. On the other hand, observability conditions are those which ensure fault propagation to circuit outputs, which are linked to fault dominators. As fault $n11/1$ has only one possible propagation path, all nodes in its transitive fanout (node $PO1$ and output $PO1_PO$) automatically become static dominators of it. Among them, only node $PO1$ has one side input ($n9$) which is set to the sensitizing value, thus having the MA $n9=1$. Both conditions ($n9=1$ and $n11=0$) form the initial SMA as shown in Figure 5.3a. Now, these assignments are propagated through the circuit by direct implication. Assignment $n11=0$ can be justified, as at node $n11$ there is only one combination of inputs which can set its output to 0: $PI4=1$ and $n8=1$. These assignments replace the former one in the SMA of the fault. In addition, the value of output $PO1$ can be deduced from $n9$ and $n11$ values. These steps are reflected in Figure 5.3b, where the highlighted assignments correspond to those which currently belong to

the SMA of the fault. Finally, assignment $n9=1$ implies that at least one of the inputs of node $n9$ must be set to 0. Because $n8=1$, that means the other input ($PI1$) must hold value 0. Therefore, assignment $n9=1$ is justified and removed from the SMA, being replaced by the new assignment $PI1=0$, as shown in Figure 5.3c. The implication process stops here, as no more assignments can be deduced, and the SMA $PI1=0$, $PI4=1$ and $n8=1$ becomes the justification frontier of fault $n11/1$. It can be verified that these assignments do not have interdependencies, which simplifies subsequent probability computations.

Once the justification frontier is obtained for a given fault, its probability can be computed as the joint probability of assignments in the J-SMA, as explained in section 5.3.2. For instance, in the previous example the probability of the fault $n11/1$ would be computed as $P(n11/1) = P(\overline{PI1} \cdot PI4 \cdot n8) = 1/2 \cdot 1/2 \cdot 3/4 = 3/16$. In addition, the justification frontier serves to deduce additional mandatory assignments for circuits with multiple outputs, which contributes to refine probability estimations. Finally, justification frontiers of approximate faults can be taken into account when updating probabilities for the remaining faults, thus obtaining the incremental probability of approximating a new fault. All these cases are covered in the next subsections

5.3.1.2 Fault implication with multiple outputs

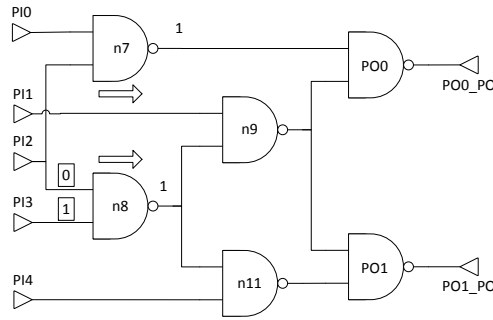


Figure 5.4: Implication of fault $PI2 \rightarrow n8/1$

In circuits with multiple outputs it may be harder to adequately estimate fault probabilities. As faults may propagate through several outputs, it is less likely to find fault dominators, and the set of mandatory assignments deduced may not accurately represent the testability conditions. Consider as an example the fault $PI2 \rightarrow n8/1$ in c17 benchmark. It must be noted that, similarly to the approximation method with static testability measures, only individual branches may be approximated, not the stem lines. The controllability condition of fault $PI2 \rightarrow n8/1$ generates the MA $PI2=0$. On the other hand, that fault has only node $n8$ as dominator, because it has propagation paths to both outputs. Therefore, only the MA $PI3=1$ can be deduced as observability condition. These assignments can just be propagated to nodes $n7$ and $n8$, as shown in Figure

5.4. In conclusion, the set of mandatory assignments for that fault is insufficient for an accurate probability estimation, because it does not include the requirement that the fault must be propagated to at least one primary output. To solve this problem, the propagation of faults is implied independently for each output, thus obtaining for each fault a justification frontier per output. This requires to compute dominators independently for each output. In this way more assignments can be deduced, at the expense of a greater computational cost. Then, the complete fault probability is computed from the union of the partial J-SMAs for each output. The possible interdependencies of these J-SMAs are solved by using the probability chain rule, as explained right after.

Let us denote the J-SMA of the propagation of fault f through node output O_i as J_{f_i} . For the first output O_1 , J_{f_1} is computed by directly applying the implication reasoning procedure explained above. J_{f_2} for output O_2 is then computed in the same way, but with the addition of checking that J_{f_1} is not fulfilled. This may generate additional MAs for the partial fault or make it redundant, or on the contrary it allows deducing a probability correlation coefficient as explained in section 5.3.2. J_{f_3} for output O_3 is computed in the same way, with the additional conditions $\overline{J_{f_1}}$ and $\overline{J_{f_2}}$, and so on. In summary, when computing J-SMA for output O_i the input vectors which propagate the fault to any of the previous outputs are excluded. This way the different J_{f_i} are guaranteed to be mutually exclusive in theory, thus simplifying computation of the total fault probability (see section 5.3.2). If a given fault f cannot propagate through output O_i or it becomes redundant, then its correspondent J_{f_i} will be null.

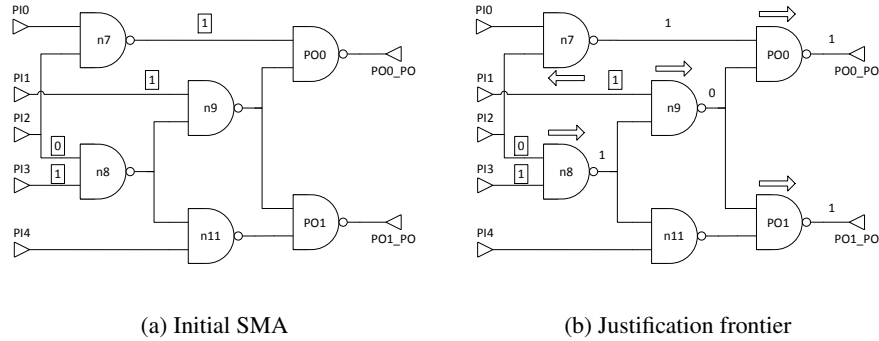
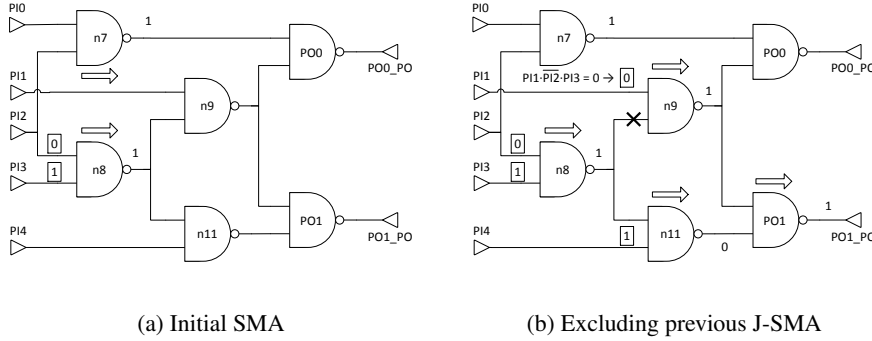


Figure 5.5: Implication of fault $PI2 \rightarrow n8/1$ through output PO0

Let us see how this procedure is applied to previous example. Instead of computing J-SMA of fault $PI2 \rightarrow n8/1$ for all possible propagation paths simultaneously, this is performed in an output by output basis. First, propagation through output PO0 is implied. According to the circuit topology, the initial SMA for this partial fault is formed by the assignments $PI1=1, PI2=0, PI3=1$ and $n7=1$, as shown in the Figure 5.5a. Now they are propagated by direct implication as depicted in the Figure 5.5b. Finally, the partial J-SMA is extracted, which contains the assignments $PI1=1, PI2=0$ and $PI3=1$. In other words, $J_{PI2 \rightarrow n8/1_0} = PI1 \cdot \overline{PI2} \cdot PI3$.

For the second output (PO1) the same process is performed, but with the additional restriction $\overline{J_{PI2 \rightarrow n8/1_0}}$. The idea is to find those input vectors which propagate the

Figure 5.6: Implication of fault $PI2 \rightarrow n8/1$ through output PO1

fault through PO1 and not through PO0. Initially, controllability and observability conditions are inferred as usual. In this case the initial SMA is just formed by MAs $PI2=0$ and $PI3=1$, and these values can be propagated as shown in the Figure 5.6a. Without any additional information, the implication process would stop here. But when excluding input vectors belonging to the previous J-SMA, additional MAs can be inferred. In particular, the condition $\overline{J_{PI2 \rightarrow n8/1_0}}$ implies $PI1 \cdot \overline{PI2} \cdot PI3 = 0$. As $PI2$ and $PI3$ already have values 0 and 1 respectively, the only way this restriction can be fulfilled is assigning $PI1=0$, which is automatically included in the SMA. As a consequence of this new implied MA, the propagation path through $n9$ becomes blocked, causing node $n11$ to become a dynamic dominator of the fault. Observability conditions are immediately applied to the new dominator, which results in the new MA $PI4=1$, which is included in the SMA too. The mandatory assignment $PI1=0$ ensures that the dynamic side input of node PO1 ($n9$) is assigned to its sensitizing value, thus ensuring at least one propagation path to output PO1. All this process is depicted in Figure 5.6b. The partial J-SMA for the second output finally includes the assignments $PI1=0$, $PI2=0$, $PI3=1$ and $PI4=1$, i.e., $J_{PI2 \rightarrow n8/1_1} = \overline{PI1} \cdot \overline{PI2} \cdot PI3 \cdot PI4$. It can be verified that both justification frontiers $J_{PI2 \rightarrow n8/1_0}$ and $J_{PI2 \rightarrow n8/1_1}$ are mutually exclusive, while at the same time both together contain the whole set of input vectors which allows propagation of fault $PI2 \rightarrow n8/1$ to any circuit output.

5.3.1.3 Inferring approximation conditions

Whenever a fault is approximated, the probabilities of the remaining faults may vary. This is a consequence of the effects that an approximation causes on the controllability and observability conditions of the remaining faults. The justification frontier of a fault can be used to estimate the impact over every other fault probability when it is approximated, as described below.

When a fault is selected to be approximated it is implied again in order to extract its J-SMA. But this time implication is performed on its correspondent approximation instead of the original circuit, thus taking into account the effect of previous approximated faults. The J-SMA of a fault f computed on the approximate circuit, i.e., the

approximation condition, is denoted as A_f . This represents the set of input vectors which differ between the original and approximate circuits because of the approximation of the fault f and therefore are susceptible to errors. Given any other fault g , its unmasked area as a consequence of fault f approximation equals the intersection $J_g \cap A_f$. Therefore, whenever a fault is approximated all remaining faults must be implied again, forcing the additional condition A_f as part of its J-SMA. This procedure provides information about which faults become affected, which allows updating fault probabilities as explained in section 5.3.2.

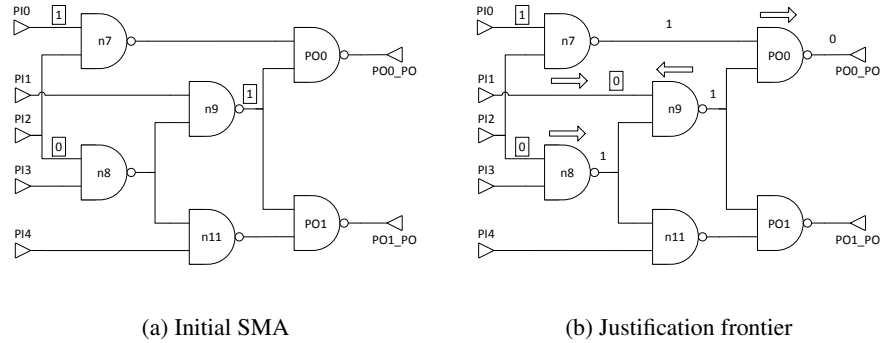
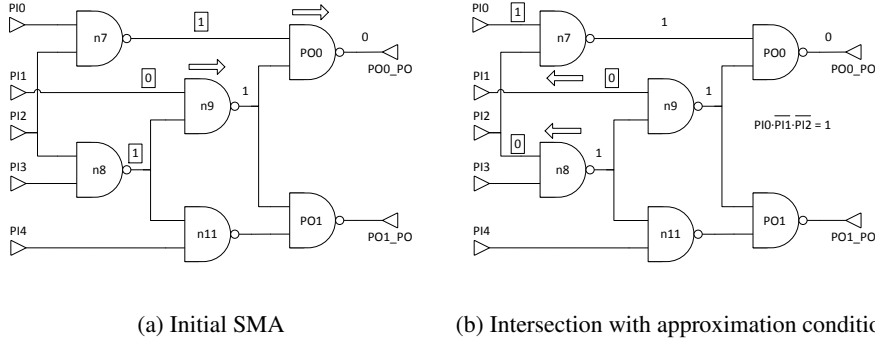


Figure 5.7: Approximation condition of fault $PI2 \rightarrow n7/1$

For example, consider $PI2 \rightarrow n7/1$ as the first fault selected for approximation in c17 benchmark. In order to estimate the effect of its approximation over remaining faults, first the approximation conditions of that fault have to be obtained. This is achieved by implying the selected fault in the correspondent approximate circuit (over-approximation in this case). This is done before the fault is effectively approximated. Therefore, as this is the first approximated fault, the approximate circuit is still an exact copy of the original. The initial MAs for this fault are $PI0=1$, $PI2=0$ and $n9=1$, as shown in the Figure 5.7a. Then, these conditions are propagated through the circuit by direct implication, as indicated in the Figure 5.7b. Finally the J-SMA $A_{PI2 \rightarrow n7/1} = PI0 \cdot \overline{PI1} \cdot \overline{PI2}$ is inferred, which becomes the approximation condition of the fault. When the fault $PI2 \rightarrow n7/1$ is approximated, the circuit is unprotected for the set of input vectors which satisfy this condition.

Once $A_{PI2 \rightarrow n7/1}$ is obtained, the effect over remaining faults can be estimated. Let us consider fault $PI1/1$, for example. In order to know which test vectors from this fault become unmasked, we first imply it in the original circuit as usual, departing from the initial SMA $PI1=0$, $n7=1$ and $n8=1$ as in Figure 5.8a. It must be noted that only propagation through output PO0 is considered here, because the approximated fault does not affect the output PO1. But in addition, the J-SMA of fault $PI1/1$ is intersected with the approximation condition $A_{PI2 \rightarrow n7/1} = PI0 \cdot \overline{PI1} \cdot \overline{PI2}$. This generates the new MAs $PI0=1$ and $PI2=0$, which are propagated as usual. The process is shown in the Figure 5.8b. Finally, the J-SMA $PI0 \cdot \overline{PI1} \cdot \overline{PI2}$ is inferred, that indicates which test vectors from fault $PI1/1$ are unmasked when fault $PI2 \rightarrow n7/1$ is approximated. The

Figure 5.8: Effect of $PI2 \rightarrow n7/1$ approximation over $PI1/1$

same procedure is applied to the rest of faults within the circuit.

As the objective consists in computing the incremental effect of approximating a new fault, intersection with all previous approximated faults should be excluded when forcing a set of faults in sequence. The goal is to know which input vectors are newly unmasked for each fault which were not already unmasked with previous approximated faults. Fortunately, for single output circuits it is guaranteed that the different A_f conditions resulting with any sequence of approximate faults are mutually exclusive. This applies to faults which propagate to just one output as well.

5.3.1.4 Approximation conditions with multiple outputs

In the case of faults which propagate through several outputs, however, the incremental effect of approximating a new fault is a bit harder to compute. Whenever a fault f is selected to be approximated, an approximation condition A_{f_i} is computed for each output O_i , that indicates the whole set of input vectors which allow propagation of fault f through output O_i in its correspondent approximate circuit. If any given fault cannot propagate through a certain output, then the correspondent A_{f_i} will be null. Approximation conditions of different faults are still mutually exclusive as long as they refer to the same circuit output, but this may not hold when comparing different outputs. In other words, consider faults f and g . A_{f_i} and A_{g_i} are guaranteed to be disjoint whatever output O_i is considered. But when considering different outputs, A_{f_i} and A_{g_j} with $i \neq j$ they may not be mutually exclusive.

For example, suppose that fault $n8 \rightarrow n9/1$ is selected for approximation in second place after the fault $PI2 \rightarrow n7/1$ is approximated. Then the approximation conditions for that fault are computed in the correspondent approximate circuit, which is the over-approximation, where $PI2 \rightarrow n7/1$ has already been forced. This means that the line connecting $PI2$ and $n7$ has been removed, and node $n7$ has been replaced with an inverter (see Figure 5.9a). As fault $n8 \rightarrow n9/1$ can propagate to both outputs, one approximation condition is derived for each primary output. This results in the approximation conditions $A_{n8 \rightarrow n9/1,0} = \overline{PI0} \cdot PI1 \cdot PI2 \cdot PI3$ and $A_{n8 \rightarrow n9/1,1} = PI1 \cdot PI2 \cdot PI3$ for outputs $PO0$ and $PO1$ respectively. The full implication process followed to ob-

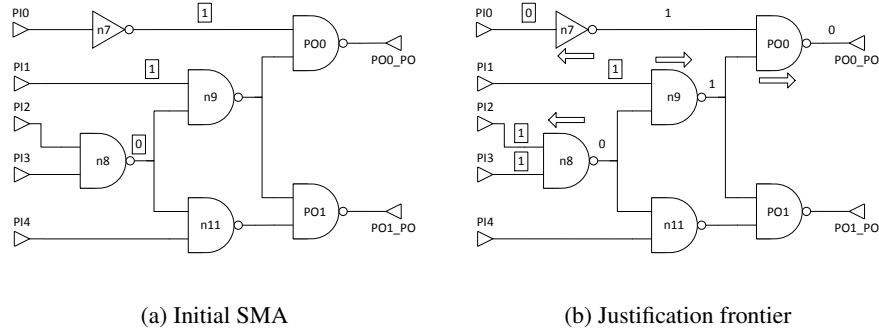


Figure 5.9: Approximation condition of fault $n8 \rightarrow n9/1$ through output PO0

tain these conditions is summarized in the Figures 5.9 for output PO0, and 5.10 for output PO1. It must be noted that when deducing the approximation conditions of any given fault through several outputs, the J-SMAs corresponding to previous outputs are not taken into account. By comparing $A_{n8 \rightarrow n9/1,0}$ with the approximation condition of the previously approximated fault ($A_{PI2 \rightarrow n7/1} = PI0 \cdot \overline{PI1} \cdot \overline{PI2}$) it can be verified that both sets of assignments are disjoint. This is due to the fact that both approximation conditions are referred to the same primary output, PO0. With respect to

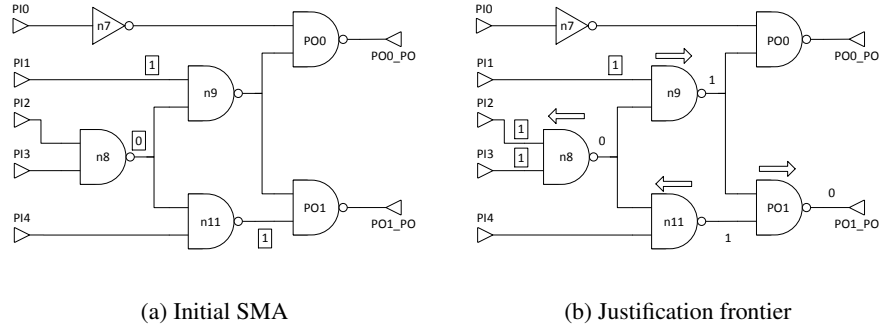


Figure 5.10: Approximation condition of fault $n8 \rightarrow n9/1$ through output PO1

the approximation condition through output PO1, $A_{n8 \rightarrow n9/1,1}$, there is no overlapping with $A_{PI2 \rightarrow n7/1}$ even though they correspond to different outputs. Therefore, effect of approximating this fault over any other fault can be computed incrementally by simply intersecting with approximation conditions of fault $n8 \rightarrow n9/1$. Notwithstanding, the contrary may also happen.

Now consider that fault PI4/1 is approximated instead of $n8 \rightarrow n9/1$. This fault corresponds to the over-approximate part too, but it only can propagate through output PO1. In order to obtain the approximation condition for this fault, it is implied in the proper approximate circuit as usual, as shown in Figure 5.11. At the end of this process,

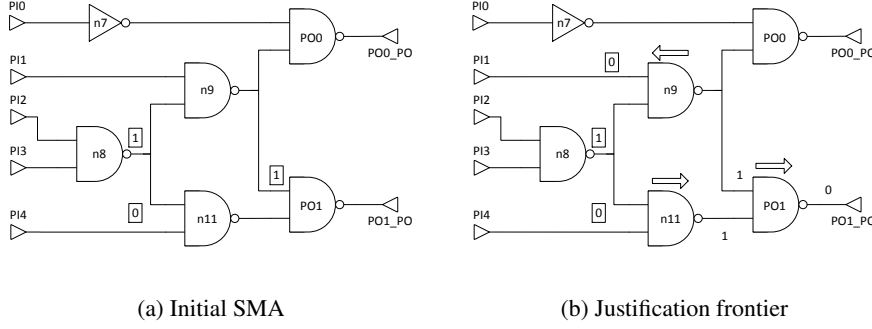


Figure 5.11: Approximation condition of fault PI4/1

the approximation condition $A_{PI4/1} = \overline{PI1} \cdot \overline{PI4} \cdot n8$ is obtained. It can be verified that $A_{PI4/1}$ overlaps with the approximation condition for the first approximated fault, $A_{PI2 \rightarrow n7/1} = PI0 \cdot \overline{PI1} \cdot \overline{PI2}$. Therefore, when incrementally computing the effect of approximation of this fault over any other fault in the same circuit it may happen that some test vectors which were already unmasked with fault $PI2 \rightarrow n7/1$ are counted again.

This fact obliges to explicitly exclude faulty input vectors already counted with all previously approximated faults which refer to different outputs. In the general case, it will be necessary to compute the effect of the last approximated fault, g , over a remaining fault, h , while having a set of f_i already approximated faults. This is computed in an output per output basis, exactly as when implicating simple faults. For the first output, O_0 , the set of conditions that we are looking for correspond to the formula

$$J_{h_0} = h_0 \cap A_{g_0} \bigcap_{i,j \neq 0} (\overline{h_0 \cap A_{f_{i,j}}}) \quad (5.3)$$

where h_0 is the set of conditions which allow propagation of the fault h through the output O_0 , A_{g_0} is the approximation condition of fault g over the output O_0 , and $h_0 \cap A_{f_{i,j}}$ is part of the set of input vectors already unmasked with the approximation of fault f_i . In the case that fault h propagated just to one output, this last term would not be necessary, because all approximation conditions referring to the same output are necessarily disjoint. However, in the general case the circuit will have multiple outputs. For each one of the remaining O_k outputs, the incremental effect of approximating g will be computed by discounting the sets of input vectors already considered in all the previous O_k , according with the formula

$$J_{h_k} = h_k \cap A_{g_k} \bigcap_{i,j \neq k} (\overline{h_k \cap A_{f_{i,j}}}) \bigcap_{j < k} \overline{J_{h_j}} \quad (5.4)$$

To see how these formulas work, let us compute the effect of $PI4/1$ approximation over fault $PI1/1$, once $PI2 \rightarrow n7/1$ has been approximated. Controllability condition $PI1=0$ and observability conditions $n8=1$ and $n11=1$ form the initial set of mandatory

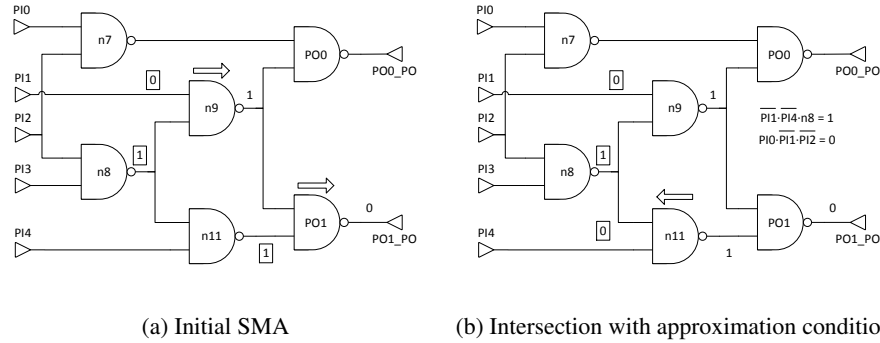


Figure 5.12: Effect of PI4/1 approximation over PI1/1

assignments of fault PI1/1 as shown in Figure 5.12a. Then this SMA is intersected with approximation condition of fault PI4/1, which is $A_{PI4/1} = \overline{PI1} \cdot \overline{PI4} \cdot n8$ as deduced in previous example. This leads to the additional MA PI4=0, which automatically justifies n1=1 (see Figure 5.12b). It must be noted that this J-SMA is computed with respect to output PO1, because the approximation condition of fault PI4/1 for output PO0 is null. As PI4/1 is not the first approximated fault, test vectors already unmasked with previous faults (PI2→n7/1 in this case) should be properly discounted. In a previous example has already been computed that approximation of fault PI2→n7/1 unmarks those test vectors from the J-SMA of fault PI1/1 which fulfil the condition $PI0 \cdot \overline{PI1} \cdot \overline{PI2}$, which is associated to output PO0. Therefore, these test vectors have to be explicitly excluded, what is done by imposing the additional restriction $PI0 \cdot \overline{PI1} \cdot \overline{PI2} = 0$. This condition does not allow to deduce any additional MA, but it is still used to derive a probability correlation coefficient, as explained in section 5.3.2.

5.3.2 Probability analysis

Once a fault has been implied and its J-SMA has been obtained, the fault probability can be derived from the assignments belonging to the J-SMA. This section explains how is this computation performed, considering all possible cases, which appear in increasing complexity order. Therefore, the first subsection (5.3.2.1) explains how fault probabilities are computed from the implied J-SMAs for single output cases, which is then extended to multiple outputs in subsection 5.3.2.2. Later, by introducing approximation conditions, the incremental update of these fault probabilities is addressed. Subsection 5.3.2.3 deals with probability updating for single output cases, while subsection 5.3.2.4 does the same for multiple outputs.

5.3.2.1 Implication-based probability computation

After a fault has been adequately justified by implication, its probability can be computed as the joint probability of assignments belonging to its justification frontier. If

the J-SMA of a given fault is null (for having no propagation path) or inconsistent (for being redundant), then it is assigned a probability equal to 0. The probability of each individual assignment can be computed by means of the COP algorithm. By default it is assumed that every input has a 50% chance of holding a logic 1, and consequently another 50% of having a logic 0. Alternatively, signal probability of inputs can be set to other values in order to model a specific workload.

As an example, consider fault n11/1 in c17 benchmark. Implication of this fault has already been performed in section 5.3.1.1 (see Figure 5.3) resulting in the J-SMA $\overline{PI1} \cdot PI4 \cdot n8$. Therefore the probability of fault n11/1 can be computed as the product of the probabilities of assignments $PI1=0$, $PI4=1$ and $n8=1$, as they are all independent. Input probabilities are assumed to be 0.5, while probability of assignment $n8=1$ is computed applying COP, thus having $P(n8) = P(\overline{PI2} \cdot \overline{PI3}) = 3/4$. As result, $P(n11/1) = P(\overline{PI1} \cdot PI4 \cdot n8) = 1/2 \cdot 1/2 \cdot 3/4 = 3/16$.

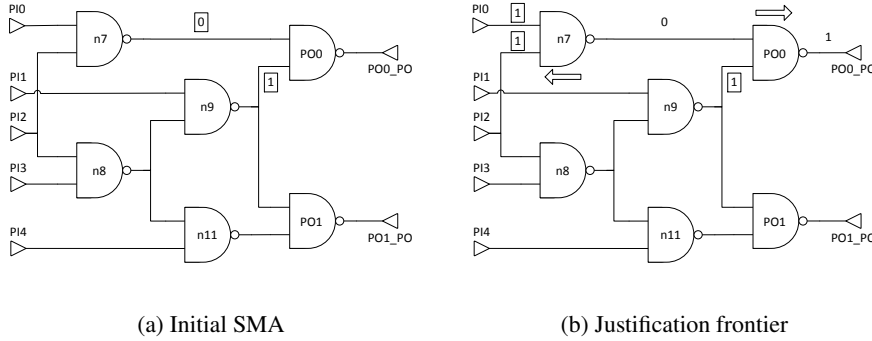


Figure 5.13: Implication of fault n7/1

However, the COP algorithm may not be accurate when there is reconvergent fanout. This may lead to inaccurate probability estimations in case not all interdependencies between mandatory assignments have been resolved. Consider as an example the fault n7/1 in c17 benchmark. Initially MAs $n7=0$ and $n9=1$ conform the SMA for this fault (see Figure 5.13a). Then these MAs are propagated as in Figure 5.13b. In the end, the J-SMA of fault n7/1 becomes $PI0 \cdot PI2 \cdot n9$. By using simple COP, probability of assignment $n9=1$ is computed as

$$P(n9) = P(\overline{PI1} \cdot n8) = 1 - P(PI1) \cdot P(n8) = 1 - \frac{1}{2} \cdot \frac{3}{4} = \frac{5}{8}$$

and therefore probability of fault n7/1 results $P(n7/1) = P(PI0 \cdot PI2 \cdot n9) = 1/2 \cdot 1/2 \cdot 5/8 = 5/32$. However, the correct result is 3/16. This is due to the fact that the implication process has not been able to resolve all signal dependencies. In particular, the mandatory assignment $n9=1$ is correlated with $PI2=1$. To solve this, signal probabilities are computed with a modified version of the COP algorithm where all assignments inferred during implication are taken into account. Actually, this means computing signal probabilities conditioned to the whole set of already deduced assignments. Applied to previous example, probability of assignment $n9=1$ is now computed

as

$$P(n9|PI2) = 1 - P(PI1) \cdot P(n8|PI2) = 1 - P(PI1) \cdot P(\overline{PI3}) = 1 - \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}$$

and fault probability becomes $P(n7/1) = 1/2 \cdot 1/2 \cdot 3/4 = 3/16$, which is the correct result.

5.3.2.2 Probability computation with multiple outputs

In the case of faults which propagate through multiple outputs, implication is made individually for each output as explained in section 5.3.1.2. Therefore, fault probabilities are computed as the union of all partial justification frontiers. Assuming the simplest case of having two outputs, the probability of any fault f would be computed with the formula

$$P(f) = P(J_{f_1} \cup J_{f_2}) = P(J_{f_1}) + P(J_{f_2}) - P(J_{f_1} \cap J_{f_2}) \quad (5.5)$$

In the general case, this means computing the intersection of the different J-SMAs. But the implication process takes advantage on previous justification frontiers in such a way that all J-SMAs inferred are mutually exclusive. This greatly simplifies probability computations, as there is no intersection between different justification frontiers, and the probability of any fault can be computed as just the sum of probabilities all partial J-SMAs.

For example, let us compute probability of fault $PI2 \rightarrow n8/1$ in c17 benchmark. This fault has already been justified in a previous example in section 5.3.1.2, obtaining $J_0 = PI1 \cdot \overline{PI2} \cdot PI3$ for output PO0 and $J_1 = \overline{PI1} \cdot \overline{PI2} \cdot PI3 \cdot PI4$ for output PO1 as J-SMAs, depicted in Figures 5.5 and 5.6 respectively. As these conditions are disjoint, the total fault probability is computed as the sum of probabilities of both J-SMAs, thus having $P(PI2 \rightarrow n8/1) = P(J_0) + P(J_1) = 1/8 + 1/16 = 3/16$.

With faults which propagate through multiple outputs it may happen that, when implicating the fault through a particular output, some of the conditions associated to previous J-SMAs are not properly justified. In that case, they are still taken into account in order to compute fault probabilities. From unjustified restrictions a probability correlation coefficient k_J is derived in the following way. After fault implication, a probability is computed for each unjustified J_{f_i} condition as if they were part of the circuit, taking into account the set of already deduced assignments. Then the correlation coefficient is computed as $k_J = 1 - \sum_i P(J_{f_i})$, assuming that the different J_{f_i} conditions are mutually exclusive. This coefficient is multiplied with the fault probability of the corresponding output.

Consider as example the fault $PI3/1$ in c17 benchmark. At first place it is implied through output PO0. Controllability and observability conditions of the fault result in the SMA $PI1=1, PI2=1, PI3=0$ and $n7=1$, as shown in Figure 5.14a. Then these assignments are propagated through the circuit by direct implication as depicted in Figure 5.14b. It is relevant to mention that assignment $n7=1$ becomes justified by $PI0=0$, the only possibility after $PI2$ is already assigned to logic 1. Implication finishes and the J-SMA $J_{PI3/1_0} = \overline{PI0} \cdot PI1 \cdot PI2 \cdot \overline{PI3}$ is extracted, with a probability of $1/16$.

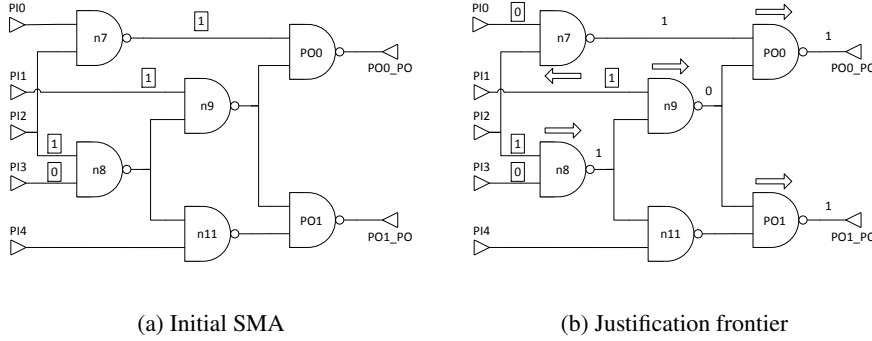


Figure 5.14: Implication of fault PI3/1 through output PO0

Then propagation of fault PI3/1 through output PO1 is implied. Because there are several propagation paths for output PO1, only MAs PI2=1 and PI3=0 can be inferred as SMA, as shown in Figure 5.15. In this case even imposing the restriction $\overline{J_{PI3/1_0}}$ does not allow deducing additional MAs, so implication stops here. The J-SMA for output PO1 is then $J_{PI3/1_1} = PI2 \cdot \overline{PI3}$, which is corresponded a probability of 1/4. This is clearly not the real probability for this partial fault, but a correlation coefficient can still be applied because $\overline{J_{PI3/1_0}}$ condition has not been completely justified. Considering the already deduced assignments for this fault, the probability of the J-SMA for output PO0 is $P(J_{PI3/1_0} | J_{PI3/1_1}) = P(\overline{PI0}) \cdot P(PI1) = 1/4$. The correlation coefficient is then $k_J = 1 - 1/4 = 3/4$, and the probability of fault through PO1 is estimated as $P(J_{PI3/1_1}) \cdot k_J = 3/16$. This is still not the real probability of propagating fault PI3/1 through PO1 and not through PO0, which would be equal to 1/8, but it is closer than if correlation coefficient were not applied. This discrepancy is due to an incomplete fault justification, which suppose a source of inaccuracy in this method. As previously said, with more sophisticated implication methods more MAs can be found, which results in better probability estimations at the expense of a higher computational cost.

5.3.2.3 Incremental probability updating

Once the probability of every fault has been computed, the best candidate among faults in the original circuit is approximated. In this approach a greedy heuristic is employed, selecting on each iteration the fault with lowest probability. The idea behind this criteria consist in minimizing the global impact of approximating a fault, as the lower the probability, the smaller the difference from the original circuit when that fault is approximated. If there is more than one fault with minimum probability value, the one with produces the greatest area savings is approximated. Area savings are estimated as the size of the transitive fanin until multiple fanout points.

Whenever any fault is approximated, this may have an impact over remaining faults. Faults in the same approximate circuit may change its probability, while in the other instances faults may be partially or totally unmasked. Therefore, whenever a fault is

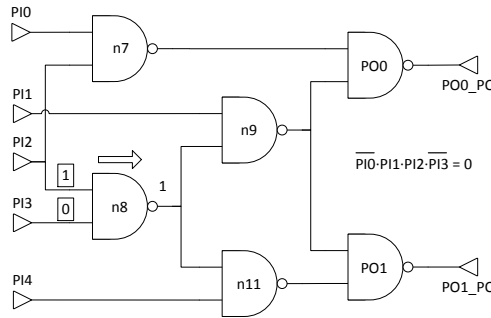


Figure 5.15: Implication of fault PI3/1 through output PO1

selected for approximation fault probabilities have to be updated. In theory, probability of every fault in the correspondent approximate circuit should be recomputed, but several considerations allow to adopt a different approach to this problem.

On the one hand, each approximate circuit is constrained to approximate faults in one particular direction. In other words, in the under-approximate circuit only faults with direction $1 \rightarrow 0$ are approximated, while the over-approximation is reserved for faults of type $0 \rightarrow 1$. Thus, for the sake of selecting the next best approximation candidate, faults $0 \rightarrow 1$ in the under-approximate circuit are irrelevant, as well as faults $1 \rightarrow 0$ in the over-approximation. In conclusion, whenever a fault is approximated only probability of faults in the same direction need to be recomputed.

On the other hand, let us consider the effect of approximating a fault over remaining faults in the same circuit and direction. Faults in the same direction may have their probabilities decreased due to the offset/onset reduction caused by the logic transformation. But they may also increase as a consequence of the appearance of new valid test vectors in the reduced offset/onset, favoured by the fault approximation. The first effect can be computed by means of intersection between approximation conditions of last approximated fault and justification frontier of the considered fault in the original circuit, which is equivalent to affected test vectors in the approximate circuit. As advantages, this computation can be incrementally performed and in addition it can be reused to estimate the global effect of fault approximation over any other fault, as explained in section 5.3.3. However, the latter contribution, which tends to increase fault probabilities, cannot be observed in the original circuit and cannot be incrementally computed. But this effect is less relevant than the first one because, as the goal is selecting the fault with lowest probability, it is more likely that the next best candidate will be one whose probability diminishes instead of increasing. Neglecting this effect may result in some fault probabilities being underestimated, which is acceptable as long as the least testable fault is correctly identified. Nevertheless, this effect could become more relevant the more faults were approximated, selecting a suboptimal fault as consequence. Besides, in any case it may happen that any fault becomes redundant cause of every propagation path being blocked. This phenomenon can only be

appreciated by implying faults in the correspondent approximate circuit.

Taking all this into account, the following method for fault probability recomputing is proposed. First, the approximation condition A_f of the selected fault f is deduced, i.e., its justification frontier computed in the correspondent approximate circuit, before that fault is approximated. This represents the input vectors for which now one of the instances hold an incorrect value. Then A_f is intersected with the justification frontiers of certain faults g , thus having $A_f \cap g$. This intersection is computed in the original circuit, and it is extended to all faults in the same direction as f , including f itself. The probability of these intersections is then computed, applying the same method as with J-SMAs. Finally, the probability of each $A_f \cap g$ condition is subtracted to the former probability value of the considered fault g . In other words, probabilities are updated by discounting those input vectors which become newly unmasked after fault approximation.

In order to see how this method works, consider fault PI1/1 in benchmark c17. Let us assume that probability of this fault has already been computed with a result of $P(PI1/1) = 11/32$, which is in fact the real probability value. Suppose now that fault PI2→n7/1 is approximated in the first place. Approximation condition for this fault has already been computed in a previous example in section 5.3.1.3, resulting in $A_{PI2 \rightarrow n7/1} = PI0 \cdot \overline{PI1} \cdot \overline{PI2}$ with respect to output PO0 (see Figure 5.7) and the null set for output PO1. Intersection between $A_{PI2 \rightarrow n7/1}$ and J-SMA of fault PI1/1 has already been computed too in other example, obtaining the subset $J'_{PI1/1} = PI0 \cdot \overline{PI1} \cdot \overline{PI2}$ (as shown in Figure 5.8). This is the set of input vectors which now may generate an error if fault PI1/1 appears in one of the two correct instances, which has associated a probability of $P(J'_{PI1/1}) = 1/8$. Finally, the probability of fault PI1/1 can be updated by subtracting the probability of unmasked test vectors, thus obtaining $P'(PI1/1) = P(PI1/1) - P(J'_{PI1/1}) = 11/32 - 1/8 = 7/32$.

When approximating several faults in sequence it is necessary to update fault probabilities on each step. As it has already been explained, this is done by computing the probability of the conditions which allow fault propagation with the last approximated fault but not with previous faults. For single output circuits this incremental computing can be implemented in an easy manner, because it is guaranteed that the different approximation conditions are mutually exclusive. The only necessary steps in order to update the probability of fault g when a new fault f is approximated are:

1. Compute the approximation condition of fault f , A_f .
2. Compute the intersection of A_f with the set of conditions which allow the propagation of the fault g , $A_f \cap g$, and the probability of this intersection, $P(A_f \cap g)$.
3. Subtract $P(A_f \cap g)$ to the former probability value of fault g .

This applies to faults which propagate to just a single output as well. In the case of faults reaching multiple outputs, however, probability updating is a bit harder to compute, as it is explained in the next section.

5.3.2.4 Probability updating with multiple outputs

As previously said, when approximating several faults in sequence it is necessary to update fault probabilities on each step. In the case of faults reaching multiple outputs, the approximation conditions associated to different faults and outputs may intersect as explained in section 5.3.1.4, and these intersections have to be properly discounted. This is performed by imposing an additional restriction $\overline{h} \cap \overline{A_{f_i}}$ for each already approximated fault f and each output O_i when re-implying any given fault h , as told in section 5.3.1.4. And as explained in section 5.3.2.2, if any of these conditions is not properly justified a probability correlation coefficient k_A is then derived.

For the sake of clarifying, let us consider first the case of just two outputs, namely O_1 and O_2 . Later the study will be extrapolated to circuits with more than two outputs. Consider fault g as the next candidate to be approximated, fault h as one of the remaining faults, and a set of already approximated f_i faults. In this case we have to compute the effect of the new fault approximation over h with respect to O_1 and O_2 independently. According to the formulas 5.3 and 5.4, to perform this computation the set of conditions $h_1 \cap A_{g_1} \bigcap_i (\overline{h_1} \cap \overline{A_{f_{i2}}})$ has to be imposed with respect to output O_1 (obtaining a J-SMA J'_{h_1}), and $h_2 \cap A_{g_2} \bigcap_i (\overline{J'_{h_1}} \cap \overline{h_2} \cap \overline{A_{f_{i1}}})$ with respect to O_2 (resulting in a J-SMA J'_{h_2}). These sets of conditions are first resolved by implication. If all the conditions are adequately justified, then their probabilities are simply computed with the modified COP algorithm explained in section 5.3.2.1, and we can directly update the probability of fault h by subtracting $P(J'_{h_1})$ and $P(J'_{h_2})$. But in the opposite case we will have an incomplete J-SMA, either J'_{h_1} or J'_{h_2} , and a set of unjustified conditions $\overline{h_1} \cap \overline{A_{f_{i2}}}$ or $\overline{h_2} \cap \overline{A_{f_{i1}}}$. In such situation, the probabilities inferred from the J-SMAs J'_{h_1} or J'_{h_2} will be inaccurate, but the unjustified conditions can be used to derive correlation coefficients (k_{A_1} or k_{A_2}) to adjust these probabilities as explained below. These correlation coefficients will then modify the probability of the J-SMAs J'_{h_1} and J'_{h_2} , thus obtaining a more realistic probability estimation of the full set of conditions. In the end, the probability of fault h will be updated as

$$\Delta P(h) = -P(J'_{h_1}) \cdot k_{A_1} - P(J'_{h_2}) \cdot k_{A_2}$$

Now let us see how these correlation coefficients are computed, starting with k_{A_1} . By applying the conditional probability theorem, the probability of the full set of conditions with respect to the output O_1 can be expressed as

$$P(h_1 \cap A_{g_1} \bigcap_i (\overline{h_1} \cap \overline{A_{f_{i2}}})) = P(J'_{h_1}) \cdot P(\bigcap_i (\overline{h_1} \cap \overline{A_{f_{i2}}}) | J'_{h_1}) = P(J'_{h_1}) \cdot k_{A_1}$$

From here, the correlation coefficient k_{A_1} can be easily computed when realizing that the different $A_{f_{i2}}$ conditions are disjoint, because all of them are referred to the same output. Thus,

$$k_{A_1} = P(\bigcap_i (\overline{h_1} \cap \overline{A_{f_{i2}}}) | J'_{h_1}) = 1 - \sum_i P((h_1 \cap A_{f_{i2}}) | J'_{h_1})$$

In order to compute k_{A_1} , the probability of each $h_1 \cap A_{f_{i2}}$ condition is computed conditioned to the set of assignments deduced in J'_{h_1} . It must be noted that if any $\overline{h_1} \cap \overline{A_{f_{i2}}}$

condition is justified during the implication process its contribution to the correlation coefficient will be null. In the case that all the conditions are properly justified, then $k_{A_1} = 1$, and the probability $P(J'_{h_1})$ remains unchanged.

The correlation coefficient k_{A_2} is computed in the same way, with the additional restriction $\overline{J'_{h_1}}$, which ensures that contributions associated to each output do not overlap. Analogously to the previous case, now we have

$$P(h_2 \cap A_{g_2} \cap \overline{J'_{h_1}} \bigcap_i (\overline{h_2 \cap A_{f_{i1}}})) = P(J'_{h_2}) \cdot P((\overline{J'_{h_1}} \bigcap_i (\overline{h_2 \cap A_{f_{i1}}})) | J'_{h_2}) = P(J'_{h_2}) \cdot k_{A_2}$$

In this case the correlation coefficient includes the term $\overline{J'_{h_1}}$. But this supposes no problem when realizing that J'_{h_1} is a subset of A_{g_1} , which by construction does not overlap with any other $A_{f_{i1}}$ condition. Therefore, all terms within k_{A_2} are mutually exclusive, and it can be computed simply as

$$k_{A_2} = 1 - P(J'_{h_1} | J'_{h_2}) - \sum_i P(h_2 \cap A_{f_{i1}} | J'_{h_2})$$

where again, the probability of each term is calculated conditioned to the set of deduced assignments in the J-SMA J'_{h_2} .

As an example, consider again the fault PI1/1. After the approximation of fault PI2→n7/1, the subset $J'_{PI1/1} = PI0 \cdot \overline{PI1} \cdot \overline{PI2}$ becomes unmasked, so the probability of fault PI1/1 is reduced to $P'(PI1/1) = 7/32$, already computed in previous examples. Now suppose that the fault PI4/1 is approximated. Its approximation condition has already been inferred in an example in section 5.3.1.4, and it is equal to $A_{PI4/1} = \overline{PI1} \cdot \overline{PI4} \cdot n8$ (see Figure 5.11 for details). It must be noted that $A_{PI4/1}$ is referred to output PO1, while $J'_{PI1/1}$ has been computed with respect to output PO0. Therefore, the effect of approximating the fault PI4/1 over PI1/1 has to be computed by imposing the conditions $A_{PI4/1}$ and $\overline{J'_{PI1/1}}$ over the J-SMA of PI1/1 with respect to output PO1. This leads to the J-SMA $J''_{PI1/1} = \overline{PI1} \cdot \overline{PI4} \cdot n8$ as indicated in the Figure 5.12, which has a probability equal to $3/16$. But the implication process is not able to fully justify the restriction $\overline{J'_{PI1/1}}$, and consequently a correlation coefficient k_A is derived. This is computed from the probability of the unjustified restriction, conditioned to $J''_{PI1/1}$. That is, $k_A = 1 - P(J'_{PI1/1} | J''_{PI1/1}) = 1 - P(PI0 \cdot \overline{PI2}) = 3/4$. Finally, the probability of fault PI1/1 is updated, thus having $P''(PI1/1) = P'(PI1/1) - P(J''_{PI1/1}) \cdot k_A = 7/32 - 3/16 \cdot 3/4 = 5/64$. There is a little imprecision in the result, because the implication method is not able to find the dependence between input PI2 and node n8, being $3/32$ the real probability result. Again, a more sophisticated implication method would be able to find more dependencies and consequently probability estimations would be more accurate.

In the case of circuits with more than two outputs, the probability updating is done in a similar way. In the general case, for each output O_k the probability of the set of

conditions defined in the formula 5.4 have to be computed, which can be expressed as

$$\begin{aligned} & P(h_k \cap A_{g_k} \bigcap_{j < k} \overline{J'_{h_j}} \bigcap_{i, j \neq k} (\overline{h_k \cap A_{f_{ij}}})) = \\ & = P(J'_{h_k}) \cdot P\left(\left(\bigcap_{j < k} (\overline{J'_{h_j}} | J'_{h_k}) \bigcap_{i, j \neq k} (\overline{h_k \cap A_{f_{ij}}})\right) | J'_{h_k}\right) \approx P(J'_{h_k}) \cdot \prod_{j \neq k} k_{A_{kj}} \end{aligned}$$

It must be noted that, in order to imply the J-SMA J'_{h_k} , all J-SMAs corresponding to previous outputs (J'_{h_j} with $j < k$) should have been already computed. Again, if some of these conditions are not properly justified, the implied J-SMA (J'_{h_k}) will be incomplete, and its probability can be adjusted by a factor derived from these unjustified conditions. In this case, this factor is obtained as a product of several partial correlation coefficients $k_{A_{kj}}$, one per output O_j . This is a simplification intended to reduce computation complexity, as explained below. Once all the partial probabilities have been computed, the probability of fault h is updated as

$$\Delta P(h) = - \sum_k (P(J'_{h_k}) \cdot \prod_{j \neq k} k_{A_{kj}})$$

As previously said, the probability of the J-SMA J'_{h_k} is adjusted by a factor derived from all the conditions not justified during implication. In theory, this factor should be computed as the probability of the intersection of all unjustified conditions, conditioned to the set of assignments J'_{h_k} . But such computation presents some drawbacks, due to the fact that the approximation conditions from different outputs may overlap. Therefore, in order to accurately adjust the probability of the J-SMA, it would be required to compute all cross terms, i.e., the joint probability that test vectors from h_k are simultaneously unmasked by approximation conditions from two different outputs, and three, and so on. But the computation complexity of such procedure grows exponentially with the number of outputs, which is not practical. Alternatively, a simplification is proposed to alleviate the computational cost. A partial correlation coefficient, $k_{A_{kj}}$, is computed per output by grouping all the unjustified conditions which refer to the same output O_j , according to the formula

$$k_{A_{kj}} = \begin{cases} 1 - \sum_i P(h_h \cap A_{f_{ij}} | J'_{h_k}) & \text{if } j > k \\ 1 - P(J'_{h_j} | J'_{h_k}) - \sum_i P(h_h \cap A_{f_{ij}} | J'_{h_k}) & \text{if } j < k \end{cases}$$

and the adjusting factor is obtained as the product of every $k_{A_{kj}}$.

The computation of each correlation coefficient $k_{A_{kj}}$ is mathematically exact, because the approximation conditions referring to the same output are always disjoint. But the interdependencies between the conditions from different outputs are neglected. In practice, it is assumed that approximation conditions with respect to different outputs are independent between them. Although this assumption may not be true, and therefore some little imprecisions can be made, it greatly simplifies probability computations.

5.3.3 Estimation of total error probability

The methodology for probability computation exposed in section 5.3.2 is intended to guide an optimal selection of approximation candidates. On each iteration, the best candidate is approximated, and probability of each remaining fault is incrementally updated. In addition, this procedure allows to estimate the impact of approximating a fault in the overall protection level of the circuit. This is achieved by aggregating all incremental probabilities over every fault from each approximation made.

With this goal in mind, the concept of *Total Error Probability* (EP) is introduced. Given a set Φ of faults f_i of size n , the EP is the average probability that any fault in Φ propagates to circuit outputs and therefore generates an error. This definition corresponds to the formula

$$EP = \frac{\sum_i P(f_i)}{n} \quad (5.6)$$

where $P(f_i)$ denotes the probability of testing fault f_i . Here it is assumed for simplicity that all faults have the same probability of occurrence, which may not be true. This average can be weighted by probability of each fault occurrence, if such information can be estimated. Do not confuse probability of fault occurrence (i.e., the probability that a certain fault is generated within the circuit due to external factors) with probability $P(f_i)$ of testing that fault (which is the probability that, assuming a certain fault appears in the circuit, this is propagated to primary outputs). In addition, only stuck-at faults are considered in this work for the sake of simplicity. However, this average can be extended to SETs by applying electrical and timing de-rating factors.

The previous definition can be used as a metric of error masking capabilities of any given circuit. For a completely unprotected circuit, the contribution of each fault to EP is equal to its fault probability. On the other hand, consider the same circuit is implemented with a TMR scheme. In that case, the total error probability for the set of faults from all three circuit instances is null because all of them are masked, supposing that each fault can only affect a single instance. In an intermediate point, a partial TMR scheme will have some faults completely masked which do not contribute to EP, and others completely unprotected whose contribution to EP is equal to its fault probability. From a global perspective, EP for a partial TMR scheme will be greater than pure TMR, but lower than the same circuit without protection mechanisms.

The same concept can be applied to error masking schemes with approximate logic circuits. In this case, the contribution of each fault to EP is not so simple as all or nothing. Each fault may be totally or partially unmasked depending on which input vectors are unprotected, i.e., which differ between target circuit and its approximate versions. First, let us consider only faults within target circuit. If the set of unprotected input vectors is denoted as A , then formula 5.6 transforms into

$$EP = \frac{\sum_i P(f_i \cap A)}{n} \quad (5.7)$$

For a fault becoming an error it is not only necessary to propagate the fault to any primary output, but in addition faulty outputs must hold a wrong value in any of other two circuit instances, so the voting scheme gives a wrong result. Because of that the contribution of any fault f_i to EP is now computed as $P(f_i \cap A)$, at least for faults from

target circuit. Faults belonging to approximate instances require a special consideration which is discussed later.

A represents the union of approximation conditions from every approximated fault. As more faults become forced, more input vectors are included in A and therefore EP grows. This is consistent with the previous reasoning. In a masking scheme with no approximated faults all three instances are identical, which is equivalent to pure TMR, and therefore A is the null set and EP=0. On the other hand, in a trivial approximation all faults have been approximated, so A is the whole set of possible input vectors and all faults can normally propagate to primary outputs. In this situation the contribution of each fault to EP is maximum and equation 5.7 transforms into 5.6, exactly as in the fully unprotected circuit. As EP tends to grow with the number of approximated faults, it can be used as a metric of the approximation process overall progress. In practice, it is used as the ending condition of the algorithm, as explained in section 5.3.4

As the approximation generation process consist in approximating faults one by one, computing the incremental EP for each latest approximated fault is the preferred way of keeping an estimation of total error probability. Consider a masking scheme with already k approximated faults. In this point, EP has a value of EP_k and the union of approximation conditions of all k faults is A_K . Now suppose an additional fault $k+1$ is approximated, whose approximation condition corresponds to A_{k+1} . Then the total error probability for this new set of approximated faults can be computed as

$$EP_{k+1} = \frac{\sum_i P(f_i \cap (A_K \cup A_{k+1}))}{n} = \frac{\sum_i (P(f_i \cap A_K) + P(f_i \cap A_{k+1} \cap \overline{f_i \cap A_K}))}{n}$$

$$EP_{k+1} = EP_k + \frac{\sum_i P(f_i \cap A_{k+1} \cap \overline{f_i \cap A_K})}{n} \quad (5.8)$$

In other words, the incremental contribution of each fault to EP as a consequence of the last approximated fault is the probability of those input vectors which newly allow propagation for the considered fault. This is exactly the same value which is discounted from each fault during the probability updating phase. Therefore, EP can be estimated by simply aggregating all incremental probabilities computed for each approximated fault during the whole process.

However, as faults may originate in any of the three instances, in theory EP should be computed with respect to the whole set of faults within approximate circuits F and H as well as the original circuit G. As previously said, a fault in one of the instances may become an error only if correspondent output belonging to any of the other two instances already holds a wrong value. It must be noted that, in absence of faults, only approximate circuits may hold incorrect values, while target circuit always gives the right result. Therefore, a fault in an approximate circuit may generate an error only for those input vectors which test any of the approximated faults in the opposite approximation. Let us split the set A of approximation conditions in two groups: A_F and A_H , which represents the union of all approximated faults in the under- and over-approximation respectively. In that way, faults in the under-approximation F may generate an error just in A_H , the set of approximation conditions belonging to over-approximate circuit. Similarly, faults in the over-approximation H may induce an error only in A_F . But there are some faults in the approximate circuits that cannot produce

an error in any case, and therefore its contribution to EP is always null. This is due to the implication relationships existing between involved logic functions. Let us consider a certain primary output O in all the three instances. In the absence of faults, possible output combinations $O_f O_g O_h$ are the following: "000", "001", "011" and "111". The first and last cases correspond to a TMR situation where all circuits give the same result and every fault is masked. Case "001" implies that over-approximate circuit holds an incorrect value, which happens with those input vectors belonging to A_H . In this situation, any 0→1 fault in either F or G which successfully propagates to primary outputs will not be masked and therefore causes an error. Faults of type 1→0 in H may propagate to primary outputs as well, but as their effect consist in correcting the mismatch caused by fault approximation they cannot be observed, and therefore this kind of faults do not contribute to EP in any case. Similarly, for those input vectors in A_F output "011" is obtained, which implies that under-approximation disagrees with the other two instances. In this situation, faults of type 1→0 in either G or H which propagates to primary outputs generate an error. On the contrary, faults 0→1 in F cannot be observed in any case, and therefore they never contribute to EP. In conclusion, the error probability extended to all instances is computed as follows

$$EP = \frac{\sum_G P(f_g^{0 \rightarrow 1} \cap A_H) + \sum_G P(f_g^{1 \rightarrow 0} \cap A_F)}{n} + \frac{\sum_F P(f_f^{0 \rightarrow 1} \cap A_H) + \sum_H P(f_h^{1 \rightarrow 0} \cap A_F)}{n} \quad (5.9)$$

But incremental computation of this formula presents some issues. On first place, here n represents the size of considered fault list, taking into account the three instances. As faults are approximated, sections of approximate circuits are removed. Therefore, some faults may no longer appear, effectively reducing the number n of possible faults as approximation process progresses. But even more important is the fact that testing conditions of faults in the approximate circuit may change during the approximation process. The EP can be incrementally computed just for the set of faults in the original circuit G because the fault list always has the same size n and testing conditions of every fault in G remain always constant. In that situation, the contribution to EP of each approximated fault (which has the effect of gradually expanding A_F and A_H sets) can be easily computed in an incremental way. When faults in approximate circuits are included into the equation, it is no longer true that testing conditions of all faults remain constant. If a fault corresponding to the under-approximation F is forced, then A_F expands and the set of conditions that test each fault in F may either enlarge or reduce. Contributions to EP of faults in both G and H can be incrementally computed, because its testing conditions do not change, but that is not the case for faults in F. Some of the input vectors included in A_H may be no longer valid test vectors for any of the faults in F, and therefore the intersection $f_f^{0 \rightarrow 1} \cap A_H$ should be recomputed again for each fault in F. On the contrary, if a fault in H is approximated, the contribution to EP of faults in either F or G can be incrementally computed, but not in the case of circuit H. For this reason, it has been decided to implement EP computation just for the set of faults in the target circuit.

It must be noted that EP is not a reliability measure on itself. It gives the average probability of, assuming that any fault has appeared in the circuit, this generates an

error. But in addition reliability takes into account the probability of fault occurrence, which is correlated with circuit area. The larger a circuit is, the greater the probability for a highly energetic particle to collide and cause a faulty transient pulse. Therefore, fault approximation presents two opposed effects. On one hand, approximating any fault tends to degrade fault masking capabilities of circuit thus increasing EP, which is detrimental with respect to reliability. But on the other hand, it also tends to reduce the size of approximate instances and subsequently the whole system, which is beneficial for reliability as it diminishes the probability of fault occurrence. Which of both effects prevails cannot be easily determined during approximation generation process, but after circuit synthesis, and it depends on both the target circuit and the degree of approximation. In an extreme case, it may happen that certain solutions have even worse reliability than the original unmitigated circuit. This is more likely to happen with heavily approximated instances and significant area overheads. In such cases, reliability can be improved by means of a combined detection and masking scheme. In that way, all faults originated in any of approximate instances are either detected or masked.

5.3.4 Approximation generation algorithm

Now all the different aspects about fault justification and probability computation have been addressed, the full process for approximation generation can be explained. Figure 5.16 shows the general scheme of the approximation generation algorithm as a pseudo-code. The process needs as inputs the target design G to be protected and an error probability target EP_T , and it generates a pair of approximate circuits that adjust to given EP target. Following paragraphs will explain the different steps that the algorithm contains.

First of all it is required to analyse unateness of target circuit, and perform any necessary logic transformations in order to obtain a fully unate initial design. Right after, two replicas of the unate version of target circuit are generated, which will become the resulting approximate circuits. At this point there are no approximated faults yet. Therefore, $EP = 0$ and the sets of approximation conditions A_F and A_H are null.

Later, the list of candidate faults is generated based on the structure of unate version of target circuit. By default, only the compact fault list is considered, i.e., only the faults which forces the sensitizing value at the inputs of each gate are considered, plus the faults at primary outputs. The reason for this is that the opposite fault, which forces the controlling value of the gate, is equivalent to some other fault located at output of the gate. Nevertheless, the procedure could be easily adapted to the whole set of faults within target design. And as usual, faults in stem lines are not considered, just individual branches.

Next, fault probabilities are computed for the first time. This is performed by implying each fault as explained in section 5.3.1 to obtain its justification frontier, and then computing the probability of that J-SMA as told in section 5.3.2. These probability values are supposed to be the superior limit of each fault, and they are stored for comparison and updating.

Then, the main loop in the algorithm is entered. In each iteration, the best candidate fault is selected and approximated, and probability of every remaining fault is updated,

Input: Target design G , error probability target EP_T
Output: Under-approximation F , over-approximation H

if G is *binate* **then**
 | Make G unate;
 $F \leftarrow G$, $H \leftarrow G$, $EP \leftarrow 0$;
 $A_F \leftarrow \emptyset$, $A_H \leftarrow \emptyset$;
 Create list of faults;
foreach fault f_i in G **do** //Initial probability computation
 | Imply J-SMA(f_i) in G ;
 | Compute probability $P(f_i)$;
while $EP < EP_T$ and non-approximated faults remain **do**
 | Select best fault f_A ;
 if f_A is an under-approximate fault **then**
 | $A_{f_A} \leftarrow$ imply f_A in F ;
 foreach under-approximate fault f_i in G **do** //Probability updating
 | Imply J-SMA($(f_i \cap A_{f_A}) \cap \overline{(f_i \cap A_F)}$) in G ;
 | $P'(f_i) \leftarrow P((f_i \cap A_{f_A}) \cap \overline{(f_i \cap A_F)})$;
 | $P(f_i) \leftarrow P(f_i) - P'(f_i)$;
 | Store J-SMA;
 | Update EP ;
 | Approximate f_A in F ;
 | $A_F \leftarrow A_F \cup A_{f_A}$;
 foreach under-approximate fault f_i in F **do** //Redundancy check
 | Imply J-SMA(f_i) in F ;
 | **if** f_i is redundant **then**
 | | Approximate f_i ;
 else // f_A is an over-approximate fault
 | $A_{f_A} \leftarrow$ imply f_A in H ;
 foreach over-approximate fault f_i in G **do** //Probability updating
 | Imply J-SMA($(f_i \cap A_{f_A}) \cap \overline{(f_i \cap A_H)}$) in G ;
 | $P'(f_i) \leftarrow P((f_i \cap A_{f_A}) \cap \overline{(f_i \cap A_H)})$;
 | $P(f_i) \leftarrow P(f_i) - P'(f_i)$;
 | Store J-SMA;
 | Update EP ;
 | Approximate f_A in H ;
 | $A_H \leftarrow A_H \cup A_{f_A}$;
 foreach over-approximate fault f_i in H **do** //Redundancy check
 | Imply J-SMA(f_i) in H ;
 | **if** f_i is redundant **then**
 | | Approximate f_i ;

Figure 5.16: Approximation generation algorithm

as well as the error probability. The loop is repeated as long as estimated EP lies below the target EP, or until every fault has been approximated. Following paragraphs explain the steps of this loop in more detail.

First step in the main loop consists in selecting the best candidate fault f_A among the remaining faults, i.e., those which have not been yet approximated. The criteria for fault selection is based in fault testability. In each iteration, the fault with minimum probability value is selected. In case of several faults sharing the least probability value, then the one among them which produces the highest area reduction is selected. This is measured as the number of wires that would be removed in case of that fault being approximated, i.e., the transitive fanin until multiple fanout points.

Once a fault has been selected for approximation, this is implied in the correspondent approximate circuit in order to obtain the approximation conditions A_{f_A} for that fault. This is required to update fault probabilities, which is the next step in the loop. Only probability of faults of the same type than selected fault f_A need to be recomputed, while faults associated with the opposite approximation are not affected by approximation of fault f_A . For each fault f_i , first the set of newly unmasked input vectors is deduced, i.e., the set of conditions $(f_i \cap A_{f_A}) \cap (\overline{f_i} \cap \overline{A_F})$ is implied. Once the justification frontier for this set of conditions is deduced, its probability is computed as explained in section 5.3.2, thus obtaining a partial probability $P'(f_i)$. Then probability of fault f_i is updated by subtracting $P'(f_i)$ from the former value. And finally the implied J-SMA is stored in order to facilitate future probability updatings as part of the $\overline{f_i} \cap \overline{A_F}$ term. After updating probabilities of all faults, the EP is updated too by adding all the $P'(f_i)$ probabilities computed for each fault.

After probability updating, selected fault f_A can be finally approximated in its correspondent instance by replacing correspondent line with a logic constant. From this point, f_A belongs to the set of approximated faults A_F if this is an under-approximate fault or A_H if it belongs to the over-approximation. But this logic transformation may cause some other faults to be redundant. So at last, all other faults of the same type as f_A are re-implied in the correspondent approximate circuit, and those faults which become redundant are marked as approximated as well. Approximated faults are excluded from selection, implication and probability updating processes. And then another iteration of the loop starts.

The ending condition of the loop is associated with EP estimation. When this is equal or greater than the target EP, the algorithm ends. The current state of both under- and over-approximate circuits constitute the result of the algorithm. Resulting circuits have to be then synthesized in order to remove generated logic constants.

5.4 Node substitution

Up to this point, the approximation generation mechanism has been based on assigning logic constants to certain lines within the replicas of the target circuit, which is equivalent to force some stuck-at faults. In conjunction with a good fault selection heuristic, this procedure has been proved to be useful with respect to the goal of achieving reasonable protection levels against faults with reduced costs. Notwithstanding, with this mechanism the set of reachable logic transformations is limited to the set of stuck-at

faults within the target circuit, while other valid solutions are left.

With the aim of widening the set of reachable solutions, a new type of logic transformation is proposed: replacing a line with a functionally similar signal. This mechanism is denoted as *node substitution* and its interest relies on finding logic transformations that cannot be obtained by fault approximation. The idea is to use both mechanisms together to generate approximate circuits. It must be noted that substitution is limited to the approximate circuit itself. In other words, it does not imply logic sharing with the target circuit or between approximate instances.

Identifying functionally similar nodes may seem a difficult task, but the approximation generation method based on dynamic testability measures provides some useful tools. In particular, advantage can be taken on fault implications performed during the generation of approximations. When a given fault is implied, a set of assignments is inferred that allow activation and propagation of that fault. Each pair of such assignments represents in fact an implication relationship and therefore a possible substitution. For the sake of simplicity, let us just consider substitutions of the fault injection point with any other line which receives an assignment. If it is assumed that the line where a given fault f is located is identified by its source node s and its destination node d and v denotes the activating value of fault f , then it can be referred to as $f : s \rightarrow d/\bar{v}$. Implication of such fault will generate assignments in certain circuit nodes n_i of value v_i . Each one of those assignments denotes a potential substitution candidate. If $v_i = v$, then source node s could be replaced with n_i . In the opposite case of $v_i = \bar{v}$, then s could be substituted with \bar{n}_i , that is, inserting an inverter. These kinds of substitutions are denoted respectively as *direct* and *inverting*. As an example, consider the circuit in

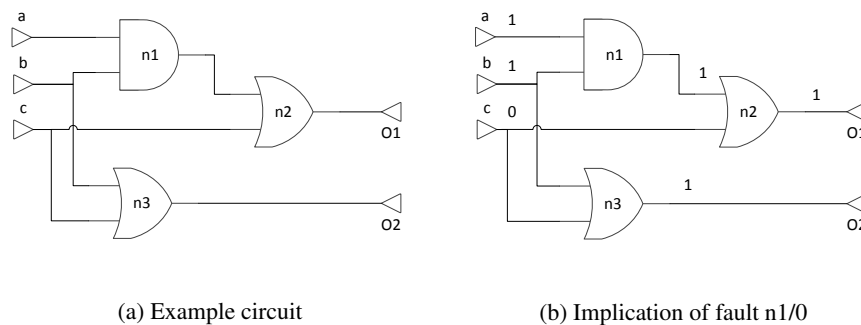


Figure 5.17: Example of node substitution

Figure 5.17a and let us suppose that fault n1 stuck-at 0 is implied. According to the procedure described in section 5.3.1, first controllability and observability conditions are identified ($n1=1$ and $c=0$). These values are then justified, thus obtaining the set of assignments in Figure 5.17b. Every node which has received an assignment indicates a candidate substitution with respect to node n1, which has been set to 1. Therefore, every other line with the same logic value (a, b, n2 and n3) would replace n1 with a direct substitution. On the contrary, input c is assigned to the opposite logic value, so in this case an inverting substitution would be performed. In the case of faults propagating to

multiple outputs, a particular node n_i will be a suitable candidate for substitution only if the same value v_i is inferred with respect to every relevant output. In other words, the intersection of the sets of assignments resulting from implication of fault f with respect to each individual output has to be performed first.

Not all potential substitutions are valid candidates, however. There are several restrictions which must be taken into account in order to preserve certain circuit properties. These are now listed, and applied to current example:

- First of all, it must be ensured that the result of any logic transformation is either an under- or an over-approximation with respect to target circuit, similarly to the line approximation approach. For this reason, the parities of involved nodes must be coherent. In a direct substitution, that is, $v_i = v$, the outputs of both replaced node s and replacing node n_i must have the same parity. On the other hand, in an inverting substitution with $v_i = \bar{v}$, both node outputs must have different parity values due to the addition of an inverter. Primary inputs are an exception to this rule, because they are not required to be unate. In the example of Figure 5.17 it can be seen that all potential substitutions are valid according to this rule. All lines have the same parity because there are no inverting gates in the whole circuit and all lines receive the same value with the only exception of primary input c . If node c had not been a circuit input, then substituting $n1$ with c would not meet the parity constraint and therefore it would not be valid.
- Asynchronous combinational feedback loops are forbidden. For this reason, all nodes in the output cone of implied fault f are excluded. In current example, only output $O1$ is affected.
- Substitutions have to be performed in such a way that critical the path should not increase with respect to the target circuit. Although delays can only be precisely computed by means of logic synthesis, an estimation based in circuit topology is used instead. A level is assigned to each node in such a way that this is always greater than any other node in its input cone. Only substitutions where the destination node d has greater or equal level than the source node s are allowed. According to this rule, the output $O1$ must be excluded in the current example.
- To avoid substitutions which are equivalent to a line approximation, all side inputs of every node in the fanout cone of the considered fault f are excluded. In the current example, primary inputs a , b and c are affected. It is true that node $n1$ could be replaced with any of these signals to obtain a valid approximate circuit, but the same result can be obtained by simply approximating a certain fault in the target circuit.
- Finally, area overhead must be taken into account. A node substitution is only interesting if the logic transformation effectively reduces circuit area, the contrary will result in a degradation of functionality without obtaining any benefits in terms of resource utilization. For this reason, the replaced node s should not be a multiple fanout point.

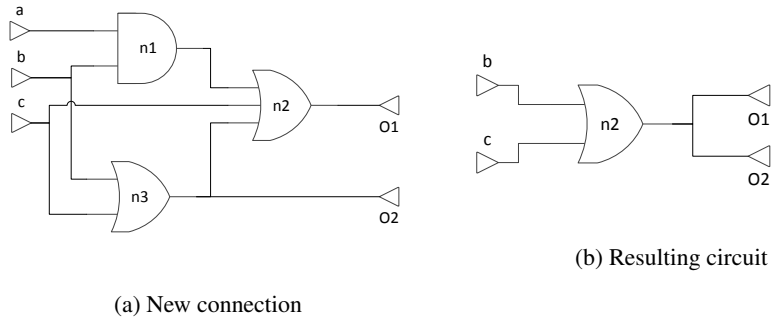


Figure 5.18: Example of node substitution

In summary, all potential substitutions originated from fault $n1/0$ are not suitable, with the only exception of node $n3$. That is, $n3$ is the only valid substitution discovered from $n1/0$ implication in order to replace node $n1$. Substitution is performed by connecting node $n3$ to $n2$ (see Figure 5.18a) and removing the line from $n1$ to $n2$. Note that the connection from c to $n2$ becomes redundant because of the new connection. Therefore, both lines can be removed, thus obtaining the circuit in Figure 5.18b. In this example the same result could have been obtained by approximating fault a suck-at 1 and following a logic optimization step, but this demonstrates that the node substitution technique is compatible with line approximation approach.

When substituting a circuit node with another, a discrepancy is originated with respect to the target circuit. In order to decide whether performing the substitution is adequate or not, the difference due to logic transformation has to be somehow evaluated. In practice, it can be quantified by computing the probability of the virtual fault $f_{subs} : n_i \rightarrow d/\bar{v}$. This is the fault associated to node substitution, or in other words, the *substitution fault*. It must be noted that this probability is computed before performing the correspondent logic transformation. Therefore, the implication mechanism has to be able to test a virtual fault, that is, a fault located in a line that does not really exist.

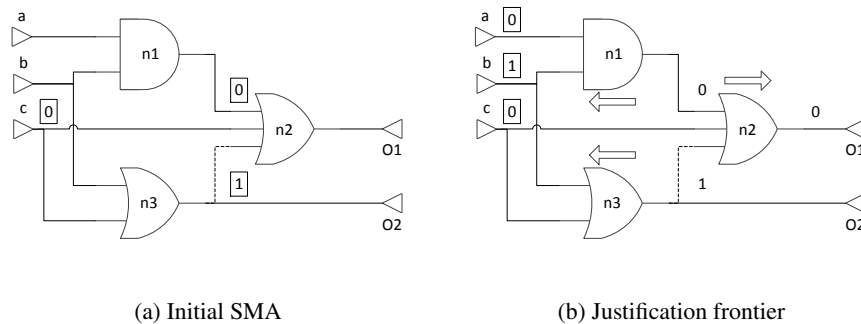


Figure 5.19: Implication of substitution fault $n3 \rightarrow n2/0$

In the current example, differences between the target circuit of Figure 5.17a and the approximation resulting from substituting node $n1$ with $n3$ (see Figure 5.18b) are computed as the probability of substitution fault $n3 \rightarrow n2/0$. This is located at the additional connection of Figure 5.18a but, because probability computation is performed prior to node substitution, such line is virtual, and therefore it is represented in the Figure 5.19 as a discontinuous line. Notwithstanding, implication is made as usual. Controllability condition $n3=0$ is assigned to the source node of the fault injection point, while observability conditions are applied to fault dominators, resulting in assignments $c=0$ and $n1=0$. This initial set of mandatory assignments is shown in Figure 5.19a. Then they are propagated through the circuit by direct implication. Assignment $n3=1$ is justified by $b=1$ because input c is already set to 0. At the same time, this causes assignment $a=0$ to be inferred as the only way of justifying $n1=0$. Finally, the output $O1$ receives a value based on the real inputs of node $n2$ (the connection from $n3$ to $n2$ is virtual and does not really exist). This process is shown in Figure 5.19b. In the end, the justification frontier $a=0, b=1, c=0$ is obtained, with a probability of $1/8$. This can be verified by comparing the logic functions for output $O1$ in both

		$O1$						$\widehat{O1}$			
		ab						ab			
c		00	01	11	10	c		00	01	11	10
0		0	0	1	0	0		0	1	1	0
1		1	1	1	1	1		1	1	1	1

(a) Target function (b) Approximate function

Figure 5.20: Karnaugh maps for output $O1$ before and after node substitution

target circuit (see Figure 5.20a) and approximation with node substitution (in Figure 5.20b). Among the 8 possible input combinations, differences can be observed in just one of them. In addition, this demonstrates that the resulting circuit of Figure 5.18b is an over-approximation with respect to target circuit because it expands its on-set. In fact, this logic transformation is based on the implication relationship $n3=0 \Rightarrow n1=0$, i.e., $\overline{O1} \Rightarrow \widehat{O1}$, which is the definition of over-approximation given in section 2.2. It must be noted that substitution of node $n1$ with $n3$ produces an approximation of the opposite type to the elimination fault $n1/0$. That is, fault $n1/0$ produces an under-approximation. In conclusion it can be inferred that, given an unidirectional fault f , any approximation by substitution deduced from it according to the rules defined previously and which preserves the unateness of target circuit would be of the opposite type to f . This fact requires some modifications to the approximation generation algorithm as it is next explained.

Integration in the main approximation algorithm introduced in section 5.3.4 is based on reusing implication of basic stuck-at faults in order to infer potential substi-


```

foreach fault  $f_i$  in  $G$  do
  | Imply J-SMA( $f_i$ ) in  $G$ ;
  | Compute probability  $P(f_i)$ ;
  | Identify valid substitution faults  $f_{sub,i}$ ;
  | foreach substitution fault  $f_{sub,i}$  do
  | | Imply J-SMA( $f_{sub,i}$ ) in  $G$ ;
  | | Compute probability  $P(f_{sub,i})$ ;
  | | Include  $f_{sub,i}$  in the list of faults;

```

Figure 5.21: Initial probability computation with node substitution

tutions. Figure 5.21 shows the modifications over the initial step of the approximation generation algorithm in 5.16 required to implement node substitution. After each circuit fault is implied, potential substitutions are identified from it. Whenever a valid node substitution is discovered, it is implied and its probability is computed as usual. Finally, the correspondent substitution fault is included in the list of candidate faults. If a node substitution is selected as the next fault to be approximated, its approximation conditions are inferred and applied to all the remaining circuit faults as usual. It must be noted that inferred potential substitutions are only valid until the approximate circuit is modified. When a new approximation candidate is selected, all previous substitution faults associated to that approximate circuit are removed from the fault list and new candidates have to be inferred.

```

foreach fault  $f_i$  in  $F$  do
  | Imply J-SMA( $f_i$ ) in  $F$ ;
  | if  $f_i$  is an under-approximate fault then
  | | if  $f_i$  is redundant then
  | | | Approximate  $f_i$ 
  | | else //  $f_i$  is an over-approximate fault
  | | Identify valid substitution faults  $f_{sub,i}$ ;
  | | foreach substitution fault  $f_{sub,i}$  do
  | | | Imply J-SMA( $f_{sub,i}$ ) in  $G$ ;
  | | | Compute probability  $P(f_{sub,i})$ ;
  | | | Include  $f_{sub,i}$  in the list of faults;

```

Figure 5.22: Redundancy checking step with node substitution

Therefore, on each iteration of the algorithm the process of identifying potential substitutions and evaluating its probabilities has to be repeated. In these cases implications performed with the aim of updating fault probabilities are not useful due to the intersection with approximation conditions of current approximated fault, which limits the scope of each fault. In addition, logic functions of target and approximate circuits differ, so it is better to identify potential substitutions in the same circuits where such transformations are performed. In conclusion, new substitutions have to be inferred by implying faults in the approximate circuit. In the approximation generation algorithm

there is a step where all faults are re-implied in the correspondent approximate circuit right after approximating a fault with the aim of detecting redundancies. Therefore, this redundancy check can be reused to discover potential substitutions. But it was previously explained that all substitutions inferred from a particular fault f generate an approximation of the opposite type to f . That is, in the under-approximation it is required to imply over-approximate faults with the goal of inferring logic transformations which agree with considered approximate circuit and vice-versa. In conclusion, in the redundancy check step implication have to be extended to all faults independently from their types. Figure 5.22 shows the changes in the redundancy check step required with respect to the under-approximate circuit. For the over-approximation changes are analogous. Implication is extended to faults of all types. For those faults of the same type than considered circuit, redundancy check is performed as usual. Faults of opposite type are implied to identify potential substitutions, and any valid transformation is included in the list of faults and its probability is evaluated as usual.

With respect to the fault selection heuristic, two of them are proposed to use in conjunction with the node substitution approach:

- Heuristic 1. This is the classical heuristic proposed for the basic method with dynamic testability measures. It is based on the criteria of minimum fault probability. If there are several candidates with the lowest value, the one which produces the greatest reduction in area is selected. Area savings are quantified as the net reduction in the number of connections if a particular logic transformation is performed. It must be noted that in the case of substitution faults an additional connection is generated. This criteria favours logic transformations with minimal impact on reliability.
- Heuristic 2. An alternative criteria is proposed apart from the classical one. This new criteria is based in the ratio between fault probability and area savings. The lowest values in this ratio are selected, with the idea of approximating the most cost-effective faults.

The node substitution technique is presented as a complement of the basic approximation generation approach based on dynamic testability measures. It has the capacity of greatly expanding the range of possible solutions. But it is also a novel idea which needs further development in order to exploit all its potential. Some possible improvements consist in expanding the scope of candidate substitutions apart from those where the fault injection point is replaced with any other line. Besides, faults in the additional lines generated due to node substitutions should be later included in the list of faults with the idea of allowing their approximation with the classic approach or inferring additional substitutions.

5.5 Experimental results

This section deals with experiments performed for both techniques explained in this chapter. Experimental results are presented and subsequently discussed.

The section is structured as follows. First, subsection 5.5.1 summarizes the experimental set-up, including benchmark selection, approximation generation process and

configuration of tests. Subsection 5.5.2 presents the results of experiments performed for approximation generation with dynamic testability measures and discuss them. Finally, subsection 5.5.3 does likewise in the case of the node substitution technique.

5.5.1 Experimental set-up

Experiments have been conducted with a group of benchmarks from LGSynth93 set. For the approximation method based on dynamic testability measures, up to 23 circuits have been chosen: alu1, alu2, b1, b9, b12, c17, c432, c880, c1908, c3540, cmb, cordic, dalu, frg1, frg2, i2, m4, rd73, rd84, s444, term1, unreg and x1. For each benchmark, approximate circuits with several error targets have been generated, thus resulting in different error masking solutions. With respect to the experiments on the node substitution technique, these have been performed with a subset of these benchmarks: alu1, alu2, b1, b9, b12, c17, c432, c880, cmb, frg1, i2, rd73, rd84, s444, term1 and unreg. Several error masking solutions with different error targets have been generated too and, in addition, for some of these circuits approximations have been generated with the alternative fault selection criteria introduced in section 5.4. Experiments have consisted in stuck-at fault simulation campaigns by means of parallel simulator HOPE with random input vectors. In this case stuck-at faults have been preferred as the test vehicle instead of SETs in order to evaluate the accuracy of probability estimations in the approximation generation algorithm. This way, logical masking is the only allowed mechanism for fault mitigation. Notwithstanding, the results can be easily extended to SETs by applying appropriate de-rating factors.

The experimental set-up for both proposed techniques is summarized in the diagram of Figure 5.23. The process starts with a description of target circuit in BENCH format. First of all, this circuit is optimized by synthesis. To this purpose, the target circuit is first translated to verilog format by means of ABC synthesizer [47], then it is synthesized with the aid of Synopsys software against the logic cell library SAED90nm [48] and finally the resulting netlist is translated again to BENCH format by means of a custom made VHDL parser. The idea behind this initial step is to start the approximation process on a simplified circuit, as it is used in practice, and in the format required by our tools.

Later, a parity analysis according to the algorithm detailed in section 3.2 is performed over the optimized target design. At the same time, unate expansion is performed in the case of binate circuits as explained in section 3.4, including expansion of XOR and XNOR gates. Then the list of faults within target unate circuit is generated, which is later used in experiments. Next, approximate instances of target circuit are generated based on its unate version and the error target specified by the user, according to the algorithms previously described either in section 5.3.4 or 5.4. One under- and over-approximate circuits are generated simultaneously on each execution of the approximation algorithm, which are described in BENCH format. All this steps are performed by means of a custom made BENCH parser. This format does not allow logic constants, and therefore approximation of faults are handled in the following way. Two additional inputs, VCC and GND, are created for the approximate circuits, which corresponds to logic values 1 and 0 respectively. Whenever a fault is approximated the corresponding line is then substituted by a connection with either VCC or

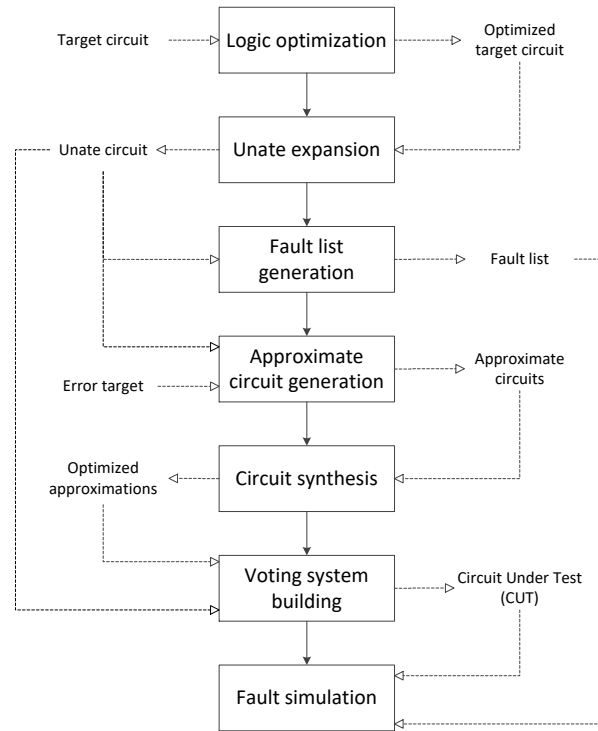


Figure 5.23: Experimental set-up with HOPE

GND, depending on the fault value.

After approximate circuits have been generated, they have to be synthesized in order to remove logic constants. This is done in the following way. First, approximate circuits are translated into Verilog format by means of ABC synthesizer. Resulting modules are individually encapsulated into a VHDL design which assigns the proper logic values to VCC and GND inputs, thus making the process user-friendly. After this step, each design is individually synthesized in order to avoid logic sharing. Synthesis are performed by means of Synopsys software with the logic cell library SAED90nm. Finally, resulting netlists are translated again to BENCH format by using the same VHDL parser as before.

The final step consist in generating the error masking system by grouping the unate version of target circuit and both optimized approximations in a TMR-like scheme. This step is performed by the BENCH parser, which is in charge of generating the proper voting logic for each triad of corresponding primary outputs, taking into account if any of them has been approximated in order to optimize the additional logic. As a result, the final circuit under test is obtained. This CUT is then tested by means of HOPE simulator, using the list of faults previously generated.

As said before, fault simulation campaigns have been performed with HOPE simulator. On each one of these tests, 50000 randomly generated input vectors have been applied. For each input vector, every fault within the initially generated list has been simulated. With respect to these experiments, the collapsed list of faults in the target circuit has been generated. Faults in the approximate circuits are not considered, with the aim of validating the assumptions for which EP is estimated in the approximation generation algorithm. The goal is therefore evaluating the accuracy of probability estimations. At the end of simulation, HOPE returns the number of faults detected by each input vector, among those included in the given fault list. The experimental error probability has been calculated as the average number of faults detected per input vector, divided by the size of the fault list, according to equation 5.6. For simplicity, all faults were considered equally likely.

5.5.2 Results on dynamic testability measures

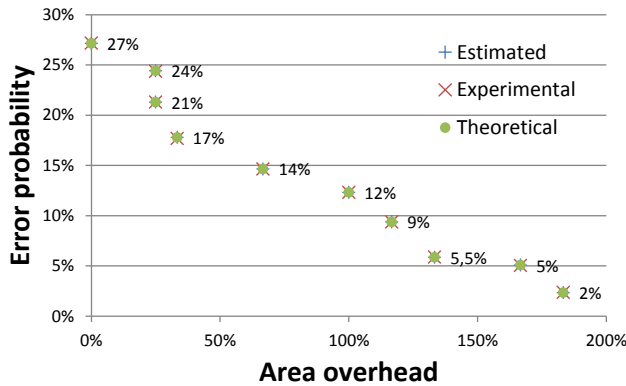


Figure 5.24: Comparative of error probabilities for c17 benchmark

To show the error probability prediction capabilities of the approximation generation algorithm with dynamic testability measures, c17 benchmark has been analysed in depth as a case study. The full example is described in detail in appendix A.4, where every step of the algorithm applied to this example is explained. Only the results of all generated solutions are commented here. Figure 5.24 shows the correlation between the area overhead due to each pair of generated approximate circuits with respect to the original design and the error probability obtained when implementing an error masking scheme with them. The labels report the error target specified to obtain correspondent points. The reported area overhead is based on the number of cells of each circuit. On the other hand, EP measures obtained from three different ways are reported: probability values predicted by the approximation generation algorithm (Estimated EP), those obtained through fault simulation with random input vectors (Experimental EP) and

Bmk.	Gates	Error probability (%)				Area overhead (%)	
		Target	Estimated	Real	Error	Over	Under
alu1	31	100	21.65	21.31	-0.33	0	0
		10	10.29	10.28	0.02	15.67	30.20
		5	5.15	5.14	0.01	66.87	53.07
		1	1.19	1.18	0.01	100	80.24
alu2	264	100	7.14	4.35	2.79	0	0
		4	4.06	2.62	1.44	31.27	77.06
		2	2.01	1.55	0.46	67.27	77.06
		1	1.07	0.94	0.13	93.25	84.34
b1	4	100	26.79	26.85	-0.06	0	0
		20	21.43	21.40	0.03	14.29	14.29
		10	10.71	10.74	-0.03	50	50
		5	5.02	5.00	0.02	50	100
b9	88	100	15.08	15.05	0.03	0	0
		10	10.09	10.09	0	5.16	7.75
		5	5.02	5.00	0.02	28.20	21.28
		1	1.02	1.12	-0.10	77.23	78.42
b12	66	100	12.36	13.86	-1.50	0	0
		6	6.55	7.91	-1.36	8.33	15.99
		3	3.11	4.13	-1.02	30.99	23.35
		1	1.03	1.19	-0.17	55.62	70.32
c432	131	100	10.82	6.63	4.19	0	0
		6	6.23	4.91	1.32	94.29	16.16
		4	4.06	3.32	0.74	102.71	82.21
		2	2.02	1.85	0.17	104.10	96.65
		1	1.02	1.14	-0.12	97.63	100.20
c880	216	100	10.27	10.60	-0.33	0	0
		5	5.01	4.51	0.50	22.54	41
		2	2.00	1.38	0.62	66.14	89.96
		1	1.00	0.64	0.36	94.74	118.26
c1908	201	100	15.10	11.42	3.67	0	0
		10	10.38	9.77	0.61	6.58	71.34
		5	5.00	6.28	-1.28	39.46	69.17
		1	1.01	1.23	-0.22	78.23	76.83
c3540	717	100	19.65	6.52	13.13	0	0
		15	15.01	3.00	12.01	100.60	3.16
		10	10.84	2.27	8.57	100.60	117.39
		5	5.01	1.32	3.69	100.60	128.15
		1	1.02	0.39	0.63	100.60	124.30
cmb	31	100	6.47	6.67	-0.20	0	0
		2	2.03	2.03	0	0	44
		1	1.36	1.36	0	18.87	45.35
		0.03	0.03	0.03	0	30.52	45.35

Bmk.	Gates	Error probability (%)				Area overhead (%)	
		Target	Estimated	Real	Error	Over	Under
cordic	79	100	2.16	1.75	0.41	0	0
		1	1.14	0.81	0.33	16	25.43
		0.5	0.51	0.24	0.27	38.08	38.81
		0.1	0.10	0.08	0.02	80.16	69.37
dalu	523	100	14.88	5.49	9.40	0	0
		10	10.17	3.09	7.08	71.24	77.14
		2	2.05	0.87	1.18	95.65	77.14
		0.2	0.20	0.06	0.14	102.13	95.81
frg1	177	100	2.94	2.73	0.21	0	0
		1.5	1.50	1.26	0.24	3.70	14.36
		1	1.00	0.94	0.06	42.07	51.07
		0.1	0.11	0.17	-0.06	61.77	51.07
frg2	629	100	15.76	14.94	0.81	0	0
		10	10.03	9.72	0.31	12.99	13.83
		6	6.40	6.27	0.14	42.49	23.79
		1	1.09	1.55	-0.46	52.20	39.66
		0.5	0.50	0.71	-0.21	52.87	57.96
		0.2	0.25	0.32	-0.07	59.28	56.23
		0.1	0.10	0.20	-0.10	79.47	79
i2	91	100	0.33	0.40	-0.07	0	0
		0.1	0.123	0.180	-0.057	0	3.96
		0.01	0.019	0.032	-0.013	0	24.49
		0.001	0.001	0.005	-0.004	41.20	43.80
m4	335	100	5.73	6.54	-0.81	0	0
		5	5.11	6.02	-0.91	2.75	2.45
		3	3.00	4.22	-1.22	13.03	22.93
		1	1.01	1.87	-0.86	64.53	55.35
rd73	20	100	14.73	20.49	-5.76	0	0
		10	10.32	16.56	-6.24	88	0
		6	6.21	10.22	-4.01	109.85	71.48
		1	1.02	3.37	-2.35	109.85	94.76
rd84	26	100	19.62	20.45	-0.82	0	0
		10	10.36	14.23	-3.87	109.47	0
		6	6.01	9.51	-3.50	114.66	72.09
		1	1.05	2.55	-1.50	108.98	110.34
s444	97	100	19.22	17.98	1.25	0	0
		10	10.11	9.62	0.49	19.12	7.64
		5	5.04	5.34	-0.30	48.03	19.85
		1	1.03	1.20	-0.16	93.11	84.92
term1	177	100	4.79	4.33	0.46	0	0
		2	2.00	1.84	0.17	4.25	2.96
		1	1.01	1.08	-0.07	10.80	6.66
		0.1	0.12	0.12	-0.01	52.61	59.81

Bmk.	Gates	Error probability (%)				Area overhead (%)	
		Target	Estimated	Real	Error	Over	Under
unreg	83	100	22.40	22.26	0.14	0	0
		18	18.00	17.85	0.15	0	17.99
		5	5.63	5.75	-0.12	43.93	17.99
		1	1.03	0.95	0.08	100	64.42
x1	329	100	8.96	8.56	0.40	0	0
		5	5.02	4.65	0.37	5.57	12.87
		2	2.09	1.90	0.19	23.61	23.57
		1	1.02	1.00	0.02	27.96	26.15
		0.1	0.10	0.11	0	58.02	58.73

Table 5.3: Experimental results with dynamic testability measures

the real probability values resulting from exhaustive simulation (Theoretical EP). It can be seen that all three methods give practically identical results. The estimated EP values coincide exactly with the theoretical ones, while the experimental EP presents a maximum deviation of 0.1% due to the random selection of test vectors.

Table 5.3 contains the results of fault simulation experiments with dynamic generation of approximate logic circuits, grouped by benchmark. First columns on the left shows the name of each benchmark and its size in number of logic gates. Then the different error targets applied on each benchmark appear, followed by the error probability estimated in each case by the approximation generation algorithm. It can be appreciated that the estimated EP is always equal or greater than target EP, except in those cases where the trivial approximation is obtained. Next the experimental error probability obtained through fault simulation is shown, along with the error in the estimation of EP. Minor discrepancies in the EP deviation can be observed due to rounding errors. The last two columns show the area overhead due to each one of the approximate versions of the circuit with respect to the combinational area of target design prior to unate expansion. These data are obtained from the synthesis steps performed for logic optimization.

In general, the approximation generation algorithm estimates accumulated error probabilities with a reasonably good accuracy, although noticeable differences can be appreciated depending on each circuit. For example, in benchmarks alu1, b1, b9, cmb, i2 and unreg probabilities are estimated with high accuracy, with absolute error margins lower than 0.33%. In particular, alu1 and cmb benchmarks are predicted almost perfectly, with the only exception of the trivial approximation. On the other hand, in benchmarks such as c432, c3540, dalu and rd73 probability estimations tend to be very poor, specially with high error targets, close to the trivial approximation. In conclusion, accuracy of EP estimations are strongly influenced by circuit structure. From data in Table 5.3 it can be seen that probability estimations in small circuits (for example alu1, b1, b9, cmb, cordic, i2 or unreg) tend to be more precise than in the case of large circuits (such as c1908, c3540 or dalu). Despite of this, there are large circuits which are predicted with reasonably good accuracy, such as c880, x1 or frg2. The last benchmark is represented in Figure 5.25 to illustrate this. The graphic shows the trade-off between error probability and the combined area overhead due to both approximations

for the whole set of solutions tested in the experiments for benchmark frg2. Both estimated and experimental EP are shown for each error target, and it can be seen that predicted values are very close to the real ones, with a maximum deviation of 0.81%. With respect to area, the results show that low error rates can be achieved with limited

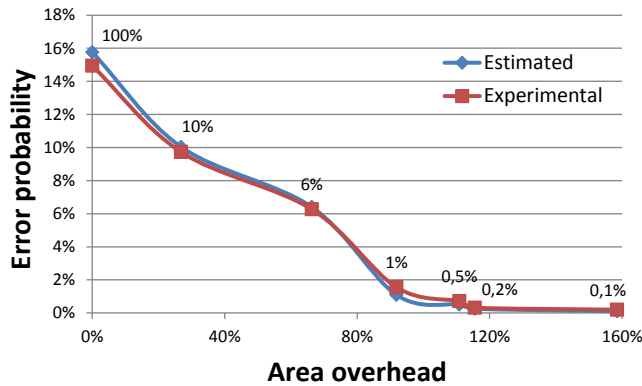


Figure 5.25: Comparative of error probabilities for frg2 benchmark

resources. For example, in b9 benchmark a 10% error rate can be obtained (i.e., 90% of errors are masked) with an area overhead smaller than 13%. If larger circuits are considered, a 6% error probability is obtained for m4 benchmark with around 5% area overhead, 1.5% EP with 18% additional area in the case of frg1, or less than 2% error rate with around 7% extra area for term1. On the other hand, there are some approximate circuits with an area overhead greater than 100%. This is the case of c432, c880 at 1% error target, c3540, dalu at 0.2%, rd73 and rd84. This phenomenon is due to the unate expansion. In circuits with a high degree of binateness many nodes have to be duplicated, and it may happen that even after approximating several faults resulting approximations are bigger than the original circuit, specially with low error targets. In such cases it is recommended to discard approximate circuits and use a classic TMR solution instead. In addition, it can be seen that in this kind of circuits probabilities tend to be more difficult to estimate. In conclusion, probability estimations work better in unate circuits, or in those with low degree of binateness.

5.5.3 Node substitution results

First, in order to show the potential benefits of the node substitution technique with respect to the simple fault approximation approach, a detailed comparison of the sequence of approximations produced with and without substitution for c17 benchmark is presented. Results are graphically shown in Figure 5.26, depicting the different points in the design space reached along the approximation process for each approximation mechanism, considering the experimental error probability and the combined

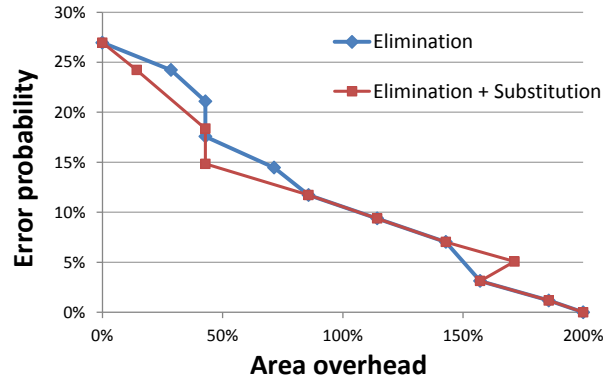


Figure 5.26: Comparative of approximation mechanisms for c17 benchmark

area overhead of both generated approximate circuits. In the cases where node substitution is allowed, heuristic number 1 has been applied, i.e., preference is given to faults with lowest probability. Initially, approximate circuits are exact replicas of target design with 0% EP and 200% area overhead, and as approximation progresses error probability increases while area overhead diminishes. Therefore, the approximation process proceed from right to left. In this case, it can be observed that including node substitution along with fault approximation generally produces equal or better results than just the basic approximation mechanism. In some points node substitution can reach a better solution in terms of either area overhead or error probability, showing that more optimal points in the solution space can be reached thanks to the node substitution. Nevertheless, eventually both approaches converge. It can be seen as well that node approximation produces at a certain point a suboptimal solution where both EP and area overhead increase due to a particular logic transformation, although this is immediately corrected in the next step.

Table 5.4 shows the results of simulations performed with approximate logic circuits generated with node substitution by applying heuristic 1. In this heuristic faults with minimum probability are approximated, and in case of a draw the fault with produces the greatest area saving is selected. The table shows for each benchmark and error target combination the experimental error probability and the combined area overhead of each pair of generated approximate circuits. These data are compared with those correspondent to the approximation generation method without substitution, inferred from table 5.3. The results show that node substitution with the classical fault selection criteria has a marginal improvement over the original approach with dynamic testability measures. There are some concrete cases where better results in terms of area or error probability are obtained, such is the case of b9 with an error target of 1% (where with a similar EP of about 1.13%, area overhead diminishes in around 14% with node substitution) or c432 approximated at 6% (2.67% EP with node substitution against 4.91% without it for an area overhead a 2% smaller in case substitution is al-

Bmk.	Gates	Error Target (%)	Node substitution		Fault approximation	
			EP (%)	Area ov. (%)	EP (%)	Area ov. (%)
alu1	31	100	21.31	0	21.31	0
		10	10.74	52.13	10.28	45.87
		5	5.26	112.30	5.14	119.94
		1	1.08	188.46	1.18	180.24
alu2	264	100	4.34	0	4.35	0
		4	2.54	124.11	2.62	108.33
		2	1.79	161.66	1.55	144.33
		1	0.63	225.95	0.94	177.59
b1	4	100	26.75	0	26.85	0
		20	21.43	28.57	21.40	28.57
		10	10.67	100	10.74	100
		5	6.25	150	6.26	150
b9	88	100	15.06	0	15.05	0
		10	10.10	12.92	10.09	12.92
		5	5.07	50.93	5	49.48
		1	1.14	141.30	1.12	155.65
b12	66	100	13.85	0	13.86	0
		6	7.52	27.53	7.91	24.32
		3	3.87	58.03	4.13	54.34
		1	1.19	125.94	1.19	125.94
c432	131	100	6.61	0	6.63	0
		6	2.67	108.03	4.91	110.45
		4	1.68	208.44	3.32	184.93
		2	0.67	226.88	1.85	200.75
c880	216	5	3.76	103.15	4.51	63.54
		2	1.63	192.54	1.38	156.09
		1	0.95	226.01	0.64	213.01
cmb	31	100	6.68	0	6.67	0
		2	2.03	44	2.03	44
		1	1.36	64.22	1.36	64.22
		0.03	0.03	75.87	0.03	75.87
frg1	177	1.5	1.21	62.26	1.26	18.05
		1	0.06	130.83	0.94	93.14
		0.1	0.04	131.47	0.17	112.84
i2	91	0.01	0.031	24.49	0.032	24.49
		0.001	0.005	85	0.05	85
rd73	20	6	11.69	187.57	10.22	181.33
		1	3.38	204.61	3.37	204.61
rd84	26	10	9.87	177.13	14.23	109.47
		1	1.59	208.74	2.55	219.31

Bmk.	Gates	Error Target (%)	Node substitution		Fault approximation	
			EP (%)	Area ov. (%)	EP (%)	Area ov. (%)
s444	97	100	17.96	0	17.98	0
		10	9.15	43.84	9.62	26.76
		5	5.13	107.39	5.34	67.88
		1	4.34	132.46	1.20	178.03
term1	177	100	4.33	0	4.33	0
		2	1.93	7.29	1.84	7.20
		1	1	21.60	1.08	17.46
		0.1	0.10	114.57	0.12	122.42
unreg	83	100	22.26	0	22.26	0
		18	17.91	17.99	17.85	17.99
		5	5.70	61.92	5.75	61.92
		1	0.97	164.42	0.95	164.42

Table 5.4: Experimental results with node substitution

lowed). But in the general case the results obtained with both approaches are pretty similar, or even identical in the case no node substitutions are performed. This indicates that in order to appreciate clear differences, an alternative fault selection heuristic which favours node substitutions should be applied. On the other hand, there are some cases where node substitution achieves worse results than with the classic approach, for example with *alu2* approximated at 4%, or *frg1* with an error target of 1.5%. That can be originated by either the local minima problem or the inability of approximating additional faults that appear due to node substitutions, which is pendant to be implemented. Finally, similarly to the approach without substitution, there are some solutions that exceed the limit of 200% area overhead due to the unate expansion. When this happens implementing a classic TMR instead is preferable.

The results of experimental error probability and combined area overhead for node substitution with heuristic number 2 are shown in Table 5.5. Correspondent data from heuristic 1 in Table 5.4 and approximation generation method without substitution from Table 5.3 are included for the sake of comparison. This shows no clear winner. For instance, heuristic 1 is clearly better in the case of *c17* at 14% target, producing both less error and less overhead, whereas heuristic 2 is clearly better in the case of *b9* at 5% target (with reduced area overhead with respect other two solutions), or *b12* with 6% error target (where both EP and area overhead are the lowest). It must be noted that in some cases one the two aspects, either the error probability or the area overhead, improves at the expense of the other. For some cases, all three approaches give exactly the same results (for example, *cmb* at any error target or *unreg* at 1%). The reason is that no interesting substitutions are found in those cases, or that there are several approximations which finally converge to the same result.

Bmk.	Error target (%)	Node substitution 1		Node substitution 2		Fault approximation	
		EP (%)	Area ov. (%)	EP (%)	Area ov. (%)	EP (%)	Area ov. (%)
alu1	100	21.31	0	21.32	0	21.31	0
	10	10.74	52.13	10.28	45.87	10.28	45.87
	5	5.26	112.30	5.05	115.50	5.14	119.94
	1	1.08	188.46	1.09	180.15	1.18	180.24
b9	100	15.06	0	15.05	0	15.05	0
	10	10.10	12.92	10.18	19.89	10.09	12.92
	5	5.07	50.93	5	41.35	5	49.48
	1	1.14	141.30	1.09	159.84	1.12	155.65
b12	100	13.85	0	13.86	0	13.86	0
	6	7.52	27.53	6.11	22.63	7.91	24.32
	3	3.87	58.03	3.12	77.07	4.13	54.34
	1	1.19	125.94	1.17	119.36	1.19	125.94
c17	100	26.97	0	26.97	0	26.93	0
	20	24.25	14.19	22.25	28.83	21.13	53.23
	14	14.73	47.78	16.43	67.46	14.49	77.07
	7	7.06	148.24	8.59	115.70	7.02	148.24
cmb	100	6.68	0	6.67	0	6.67	0
	2	2.03	44	2.04	44	2.03	44
	1	1.36	64.22	1.35	64.22	1.36	64.22
	0.03	0.03	75.87	0.03	75.87	0.03	75.87
s444	100	17.96	0	17.94	0	17.98	0
	10	9.15	43.84	8.62	29.35	9.62	26.76
	5	5.13	107.39	5.87	58.98	5.34	67.88
	1	4.34	132.46	1.13	181.21	1.20	178.03
term1	100	4.33	0	4.33	0	4.33	0
	2	1.93	7.29	1.81	14.53	1.84	7.20
	1	1	21.60	0.97	21.90	1.08	17.46
	0.1	0.10	114.57	0.07	118.77	0.12	122.42
unreg	100	22.26	0	22.26	0	22.26	0
	18	17.91	17.99	22.27	0	17.85	17.99
	5	5.70	61.92	5.17	105.62	5.75	61.92
	1	0.97	164.42	0.96	164.42	0.95	164.42

Table 5.5: Comparison between fault selection heuristics

5.6 Conclusions

In this chapter an alternative fault selection method for approximate logic circuit generation has been presented, which is based in dynamic testability measures. In this approach fault testabilities are analytically computed, with the aid of an implication motor in order to overcome the reconvergent fanout problem. The best fault according to the criteria of minimum probability and maximum area saving is selected. Whenever a fault is approximated, fault probabilities are dynamically updated, thus taking

into account the effect of last approximated fault, and the next best candidate fault is selected in an iterative process. In that way, testability measures are always representative of the current state of the circuit and no suboptimal transformations are performed, except for the local minima problem. These are relevant improvements with respect to the approach based in static testability measures. In addition, variations in computed fault probabilities allow estimating the total error rate for the whole set of faults considered, which eventually serves as the ending condition of the approximation generation process. Therefore, with this approach approximate circuits which roughly meet a specified target error rate can be generated. This is another improvement over the previous approach, where trial and error was required. Error target is now the parameter which provides flexibility to the method, allowing to generate solutions with different trade-offs between error rate and overheads, from pure TMR to the trivial approximation.

With respect to the solution based in static testability measures, the dynamic approach generates solutions with a finer granularity at the expense of a higher computational effort. And by making use of dynamic testability measures, the approximation threshold (i.e., the error target) is more representative of achieved error protection level, which is useful for the final user.

Possible improvements of the proposed approach include avoiding the problem of local minima and improving the accuracy of probability estimations in the most difficult cases. Both issues can be addressed by implementing recursive learning at different levels in the approximation generation algorithm, at a cost of an increased execution time. It is possible to develop alternative fault selection heuristics as well, in order to favour other criteria. Finally, the global error probability estimation can be improved to include faults originated in all three instances and not just in the original circuit. Mathematical computations required to do so have been presented, although it has been proved that such computation cannot be done incrementally, which further increases computational costs.

It must be noted that this technique has been developed just for combinational circuits. Its application to sequential circuits is really difficult, due to the fact that signal probabilities depends not only on primary inputs, but also on current and past circuit states. Therefore, accurate computation of signal probabilities in a sequential circuit is considered a irresolvable problem. Alternatively, it is suggested to consider sequential elements as inputs and outputs with respect to circuit combinational logic, although this is not fully representative of the real behaviour of the circuit.

In addition, over this method it has been implemented the possibility of substituting nodes as a way of generating approximations, complementary to logic constant assignment. This is denoted as node substitution, and it allows reaching solutions out of the scope with the classic line approximation approach. It makes use of the implications inferred during probability computations in the approximation generation method with dynamic testability measures in order to deduce candidate substitutions. A probability is associated to each discovered substitution, so it is evaluated along with all other circuit faults. In order to favour node substitutions over fault approximations, an alternative fault selection heuristic is proposed, consisting in optimizing the ratio between the fault probability and the area savings. Experimental results show a marginal benefit of this approach with respect to the classic fault approximation method, and no clear

winner between both proposed fault selection heuristics. Further development would be required in order to exploit the full potential of this node substitution mechanism.

Chapter 6

Applications

6.1 Introduction

In this thesis a new method for approximate logic circuit generation in the context of error mitigation has been developed. This technique consist, on the one hand, in an approximate generation mechanism based on stuck-at fault forcing, which has been introduced in chapter 3, and on the other hand, in a fault selection criteria which selectively applies this mechanism to generate approximate versions of a given target circuit. Two different heuristics are proposed in this work, based respectively on static and dynamic testability measures, which has been described in chapters 4 and 5 respectively.

Once the proposed method has been developed, it has been applied to several contexts, taking into account its potential benefits. In particular, the following applications have been proposed:

- Generation of approximations with the aim of implementing approximate TMR in Field Programmable Gate Arrays (FPGAs). Circuits are typically synthesized in an FPGA in the form of a structure of Look-Up Tables, as opposed to the logic gate structure used in the proposed technique, which is better suited for Application Specific Integrated Circuits (ASICs). Therefore, in the context of circuit design for FPGAs, several considerations have to be taken in order to benefit from the approximation generation method proposed in this thesis.
- Implementation of an approximate TMR for a real application circuit. In particular, the ARM Cortex M0 microprocessor has been selected as the target circuit for this application.
- Comparison of the approximation generation method with alternative novel approaches. In particular, the proposed approach based on dynamic testability measures, which follows a greedy heuristic to decide among circuit manipulations, has been compared against evolutive techniques for generation of approximate circuits, which can randomly generate multiple approximations by trial and error.

In the first two applications the approximation generation method based on static testability measures has been used, while for the comparison with evolutive approaches the dynamic approach has been preferred,

Some of these applications have been motivated by the collaboration with external institutions from industry as well as from academia. In particular, the approximate TMR for the Cortex M0 microprocessor is the product of a collaboration with CISCO (USA) IROC Technologies (France) and University of Saskatchewan (Canada). The group *Concepção de Circuitos Integrados* from Universidade Federal do Rio Grande do Sul (Brasil) has collaborated with the tests of ATMR in FPGAs. Finally, the comparison with evolutive techniques was possible by means of a collaboration with Brno University of Technology (Czech Republic).

The following sections address each one of these topics in detail. Thus, section 6.2 deals with the application of the approximate TMR to FPGAs and presents the results of the tests performed in relation with it. Then, section 6.3 is about the implementation of an ARM Cortex M0 microprocessor hardened by approximate TMR. Next, section 6.4 introduces the evolutionary generation of approximate circuits and presents the results of the comparison between that approach and the one based in dynamic testability measures. And finally, section 6.5 concludes the chapter.

6.2 Extension to FPGA-based circuits

An FPGA is a logic device whose functionality can be externally programmed. It contains a fixed amount of logic blocks which can be configured and interconnected as desired in order to implement any target design, up to the point its resources allow. Their flexibility and relatively low cost make FPGAs well suited for prototyping tasks and applications with low volume of production, where manufacturing a dedicated ASIC would be too expensive. In recent years, their performance and power consumption have improved as well, motivating their use in space applications, among others. However, FPGAs are susceptible to errors due to radiation effects, and therefore any design implemented on them would require some mitigation mechanisms when operating in radiation environments.

Currently there are three different technologies that allow the configuration of those programmable logic devices

- In Static RAM (SRAM)-based FPGAs, the configuration elements consists in static Random Access Memory (RAM) memory cells. Since SRAM is volatile and cannot keep data without power source, the FPGA must be configured upon start, either by an external device or by means of a non-volatile memory implemented along with the FPGA, which stores the FPGA configuration. Although they demand higher power with respect to other FPGA technologies, the majority of commercial FPGAs are SRAM-based due to its reduced transistor sizes, which allow higher integration densities.
- Flash-based FPGAs make use of non-volatile memory elements (floating gate switches) in order to store the FPGA configuration. Therefore, configuration is

not lost when the power is disconnected. They consume less power in comparison with SRAM-based FPGAs. However, they require a flash-based manufacturing process that makes them less competitive.

- In antifuse FPGAs, antifuses are used as configuration elements. These are elements that by default work as an open circuit, but after "burning" they conduct current (as opposed to a fuse). This process is not reversible, hence this type of FPGAs can only be configured once.

These different technologies are affected by radiation effects in different ways. In the antifuse technologies, the main effects are SETs in the combinational logic or SEUs in sequential elements, like an ASIC. The SRAM-based FPGAs, however, are mainly susceptible to SEUs in the configuration memory, which may modify the programmed functionality almost permanently (at least until power is removed or the device is re-configured). With respect to flash-based FPGAs, their configuration bits require much more energy to be modified, so they are less sensitive to SEUs in the configuration memory. But with technology shrinking, the sensitivity of flash memory cells will gradually increase.

The approximate TMR technique can be implemented in an FPGA for sure. Usually, any given design will not make use of all resources available inside the FPGA, meaning that additional logic can be used to implement approximate versions of such target design in a TMR fashion. This allows a wide range of solutions with different tradeoffs between error mitigation capabilities and overheads for those applications that may allow some temporary misbehaviour, in contrast with the partial TMR approach where the options are limited to replicate or not each one of the components of given design. This approach can be combined with scrubbing in those cases where the configuration memory is susceptible to SEUs, typically in SRAM-based FPGAs.

In an FPGA, the combinational part of a circuit is implemented as a set of Look-Up Tables (LUTs) properly configured and routed. A LUT is a small memory that implements a logic function of its inputs, typically up to 6 inputs in modern FPGA technologies. These LUTs work as partial truth tables of the logic design. In contrast, the approximation technique proposed in this work is based in a logic gate structure. A single LUT can implement a logic function equivalent to a group of logic gates, which means that some logic transformations which were valid in a logic gate circuit may not produce any resource savings on its LUT counterpart. Therefore, it is required to know the correspondence between both circuit structures in order to generate adequate approximations in this new context.

In practice, real application circuits tend to have sequential elements, and in the case of circuits implemented in an FPGA this is no exception. Indeed, the programmable logic of an FPGA includes configurable flip-flops among its resources, and even in some technologies the LUTs may be configured to work as shift registers. Therefore, in order to be able to process sequential circuits, the approximation generation method based on static testability (described in chapter 4) measures has been preferred for this extension.

The approximation generation method proposed in this thesis is based on the properties of unate functions. This requires to perform a unate expansion for binate circuits, as explained in section 3.4. This expansion poses two significant problems. On the one

hand, the area of the target circuit increases. A larger area means a higher chance for a high energetic particle to impact the circuit and provoke a soft error, which is not desirable at all. This issue can be solved by using the binate version of the target circuit instead of the unate one in the ATMR implementation, although approximate circuits still have to make use of the unate expansion. On the other hand, it is assumed that the area increase in the approximate circuits will be compensated by performing fault approximations. But this is not necessarily true, specially when a low amount of faults is approximated. Moreover, synthesis tools may not be able to undo the logic transformations due to the unate expansion. In order to compensate the overhead due to the unate expansion, a new approach for approximation of faults in binate lines is proposed. Although a binate fault cannot be directly approximated, it can be decomposed in two complementary unidirectional faults, one belonging to the under-approximate circuit and the other to the over-approximate circuit. Therefore, approximating a binate fault reduces the logic in both approximate instances. Moreover, with this approach the testability measures can be directly obtained from the target circuit itself, instead of its unate version.

Finally, radiation experiments of an ATMR scheme have been performed for the first time with this FPGA approach. These experiments have been complemented with some fault injection tests for a better comprehension. Some of these tests have been performed in collaboration with the group *Concepção de Circuitos Integrados* from Universidade Federal do Rio Grande do Sul.

This part of the chapter is structured as follows. Subsection 6.2.1 introduces some existing solutions to implement fault mitigation on SRAM based FPGAs. Then, subsection 6.2.2 explains how the approximation generation method proposed in this thesis is applied to circuits implemented on a FPGA, and 6.2.3 presents a new method for approximating binate faults. Finally, subsections 6.2.4 and 6.2.5 show, respectively, the radiation and fault injection experiments performed with this approach, explaining both the experimental setup and the results of such experiments.

6.2.1 Fault mitigation strategies in FPGAs

FPGAs are becoming increasingly attractive for space applications. In comparison with ASICs, they provide higher flexibility and lower cost, particularly for the low volume production which is characteristic of space applications. As technology progresses, new devices with increased resources and performance are becoming available. Unfortunately, FPGAs are susceptible to radiation effects. Thus, fault tolerance is generally required for applications that operate in a radiation environment.

FPGAs programmed by antifuse topology are more like standard cell ASICs, as the customization cells (antifuse) are not susceptible to radiation effects. For this reason, techniques used in ASICs can be easily applied to the high-level description, such as DWC, TMR or EDAC. At the architectural level, for instance, it is simple to replace all the flip-flops with hardened memory cells [59]. These can be applied to flash-based FPGAs too, due to the high energy required to modify the state of the configuration bits.

With respect to SRAM-based FPGAs, they can be affected by SEEs in the programmed (functional) logic as well. But the main source of errors are the SEUs in the

configuration memory, which can modify the functionality of programmed logic until reconfiguration or power off. For that very reason, any high-level fault-tolerant techniques applied to the target design (such as DWC or TMR) are not enough to ensure reliability, as they do not prevent from error accumulation in the configuration memory, eventually making the design to fail. Therefore, such techniques have to be combined with a periodic refresh of the configuration memory in order to clean any possible bit upsets, which is known as *scrubbing* [59]. The refreshing rate should be high enough to prevent the failure of the design due to SEU accumulation. But on the other hand, one reconfiguration cycle takes a significant amount of time. For that reason, alternative approaches based in partial reconfiguration of the configuration memory, i.e. partial scrubbing, have been proposed. In addition, scrubbing has significant error detection latency and therefore it cannot prevent temporary erroneous behaviour. So, designs hardened by TMR or any other fault-tolerant technique are still required to protect against transient effects in the programmed logic.

Although TMR is a very effective mitigation technique, it is often expensive in terms of FPGA resource utilization and power consumption [60]. For applications that can tolerate some temporary misbehaviour, Partial TMR can be used to trade off the reliability with the cost of mitigation. In [61] and [60] an automatic solution for Partial TMR is proposed that is based on the concept of persistence. A persistent configuration bit is a sensitive configuration bit that will cause an error when upset that cannot be recovered by scrubbing, so that even after repairing persistent configuration bits through configuration scrubbing, the FPGA circuit does not return to normal operation. On the contrary, non-persistent bits imply some data loss, but the design returns to normal operation when the error is repaired through configuration scrubbing. Persistent bits can be found by topological analysis, looking for feedback structures. If the feedback logic is affected by a fault on the correspondent configuration bits, it may cause the design flip-flops to store a wrong state, which cannot be repaired by just scrubbing. These feedback structures are associated to persistent bits and thus must be triplicated first. If resources allow, mitigation is applied to the non-persistent circuit structures to reduce the remaining design sensitivity.

6.2.2 Fault approximation in FPGAs

The fault approximation techniques developed up to this point in this thesis and explained in chapters 3, 4 and 5 were intended for ASICs. In this context, a logic circuit is typically represented in a logic gate structure. On the contrary, in an FPGA a logic circuit is represented in the form of a group of LUTs, where a single LUT may be functionally equivalent to several logic gates. The application of the fault approximation technique on FPGAs has been performed by reusing the tools and techniques already developed for a logic gate structure, with the minimal modifications required to adapt to the new circuit structure.

While in a logic gate structure any approximated fault will produce resource savings, this may not be true for a LUT structure. In the latter case, approximating a fault may cause the logic function of a LUT to be modified but not simplified, and therefore it would deviate from the intended functionality without any real benefits. This requires imposing some restrictions on the faults which can be approximated. In particular, the

most interesting logic transformations are those that effectively reduce the number of LUTs in the circuit, either by eliminating LUTs or merging contiguous LUTs. The contrary will result in a degradation of the logic function of the circuit without achieving any benefits in terms of resource savings. Reducing the size of a LUT, by removing some of its inputs, may also contribute to reduce the interconnection needs and the associated configuration bits, to a small extent. Eventually, when the input or the output of a flip-flop is substituted by a constant, the flip-flop can be removed and the voting logic can also be simplified, as explained in section 4.3.

With this idea, the classical logic gate structure used in this thesis for approximation generation is complemented with a LUT superstructure. Here, a LUT is considered as a logic block with a maximum of 6 inputs and just one output, and which may encapsulate any number of logic gates. When using this additional structure, the logic gate structure of any given circuit is partitioned into LUTs. Thus, every logic gate within the circuit has to be contained in a LUT, and LUTs cannot overlap. As a simplified example consider the Figure 6.1, where the c17 benchmark circuit has been partitioned into LUTs of 2 or 3 inputs. Due to the existing multiple fanouts, some of the LUTs contain just one logic gate. After the circuit has been partitioned into LUTs, fault approximations can be performed as usual, with the additional restriction that they can only be applied on the inputs and outputs of the LUTs. That means, for example, that the connection between gates n7 and PO0 cannot be forced to a constant logic value, because such connection is located inside of a LUT. On the contrary, fault n8→n11/1 could be approximated, allowing the LUTs number 2 and 3 to be merged, and at the same time reducing the number of inputs of LUT4 from 3 to 2. Another fault which could be approximated is PO0/0, thus suppressing LUT1 and merging LUTs number 2, 3 and 4 into just one LUT. It must be noted that, in modern FPGA technologies, the LUTs typically have a size of 6 inputs, so the mapping proposed in Figure 6.1 would probably be suboptimal. This has been generated directly from the logic gate structure, avoiding any replication of logic gates.

When synthesizing a circuit to be implemented in an FPGA, it is transformed into LUTs. Therefore, departing from the logic gate structure and partitioning it into LUTs may be suboptimal, which is the case of the example in Figure 6.1. In practice, the synthesized circuit, which is already mapped into LUTs, is taken as the starting point for this new approach. And from there, the internal logic gate structure of every LUT is regenerated based on its truth table by means of a custom made parser. This, applied to the case of c17 benchmark, would produce a result more similar to the mapping of Figure 6.2, where just two LUTs are required by replicating nodes n8 and n9. From the point of view of an FPGA resource utilization, this would be a more efficient implementation than the mapping of Figure 6.1 which requires 4 LUTs. On the other side, the number of faults which can be approximated in the 2 LUT implementation is smaller, restricted just to primary inputs and outputs.

As explained in chapter 3, in order to ensure that a given approximated fault generates a unidirectional approximation, this has to be applied on an unate line. For that reason, faults on binate lines cannot be approximated unless an unate expansion is performed (explained in section 3.4), which introduces some area penalties. In the classic fault approximation approach, the original circuit was substituted by its unate version in the final ATMR implementation, and the testability measures were obtained from

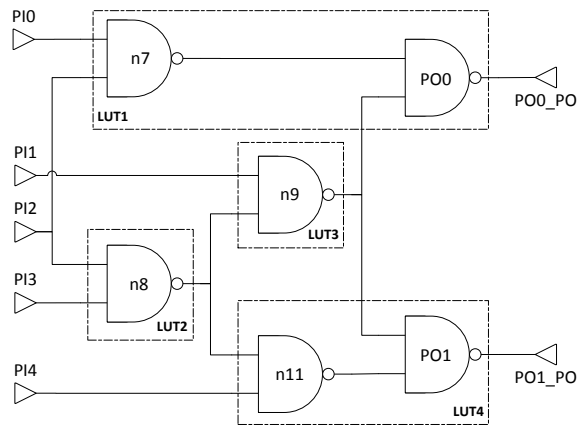


Figure 6.1: LUT partitioning of c17 benchmark

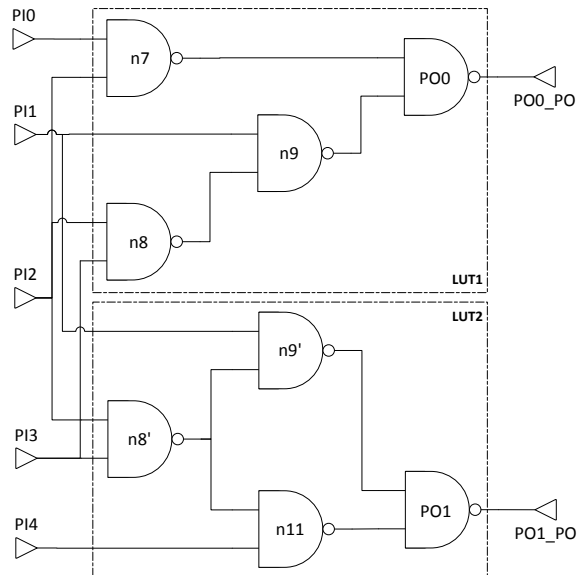


Figure 6.2: Alternative LUT partitioning of c17 benchmark

that version. Now, in the FPGA implementation, this is intended to be avoided, thus reducing the penalties due to approximating binate faults. Therefore, the testability measures are now obtained from the original circuit itself, and the ATMR is built with that same circuit. This change requires devising a method for approximating binate faults, which is explained in the next section.

6.2.3 Bidirectional fault approximation

If a logic function G is unate with respect to a certain signal x , then G can be expressed as either $G = xG_1 + G_0$ or $G = \bar{x}G_1 + G_0$, depending on the parity of signal x , where G_0 and G_1 are logic functions independent from x . In that case, the logic functions $G_x = G(x = 1)$ and $G_{\bar{x}} = G(x = 0)$ will serve as unidirectional approximations of logic function G . The demonstration is explained in section 3.3. G_x and $G_{\bar{x}}$ are denoted as the cofactors of G with respect to x .

On the contrary, this no longer holds in the case of a binate circuit. According to Shannon's expansion formula, a logic function G which is binate with respect to a certain signal x can be expressed as $G = xG_x + \bar{x}G_{\bar{x}}$. In Boolean Algebra, the classical way to reduce a logic function is to use the consensus ($F = G_x \cdot G_{\bar{x}}$) and smoothing functions ($H = G_x + G_{\bar{x}}$). Such functions can work in order to approximate binate lines, although its generation is not straightforward. On the other hand, splitting the effects of a binate fault into its unidirectional components is far simpler. Whenever a binate fault is approximated, there is a set of input vectors whose result change from 0 to 1, and another set of input vectors which produce changes in the opposite direction. This is obvious by taking into account the Shannon's expansion formula. Therefore, what can be done in order to approximate a binate fault is approximating the positive and negative parts of the selected binate signal in the different approximate circuits. For example, consider that a binate fault $x/1$ can be approximated. According to Shannon's expansion formula, approximating the negative part will generate the logic function $F = xG_x$, and approximating the positive part, $H = G_x + \bar{x}G_{\bar{x}}$ is obtained. From here it is obvious that $F = xG_x \Rightarrow G = xG_x + \bar{x}G_{\bar{x}}$, i.e., F is an under-approximation of G . On the other hand,

$$\bar{H} = \overline{G_x + \bar{x}G_{\bar{x}}} = \bar{G}_x \cdot \overline{\bar{x}G_{\bar{x}}} = \bar{G}_x \cdot (x + \bar{G}_{\bar{x}})$$

$$\bar{G} = \overline{xG_x \cdot \bar{x}G_{\bar{x}}} = (\bar{x} + \bar{G}_x) \cdot (x + \bar{G}_{\bar{x}}) = \bar{x}(x + G_{\bar{x}}) + \bar{G}_x(x + \bar{G}_{\bar{x}}) = \bar{x}G_{\bar{x}} + \bar{G}_x(x + \bar{G}_{\bar{x}})$$

which means that $\bar{H} \Rightarrow \bar{G}$, i.e., H is an over-approximation with respect to G . The procedure is analogous in the case of fault $x/0$. In this situation, approximating the positive part will generate the under-approximation $F = \bar{x}G_{\bar{x}}$, while the over-approximation $H = xG_x + G_{\bar{x}}$ is obtained when the negative part is approximated.

In practice, approximation of binate faults can be done in the following way. The unate expansion already allows splitting any binate line into two unate lines which correspond to the positive and negative parts of the original line, respectively. The approximation of a binate fault is then performed by approximating the fault for just one of the two unate lines in each one of the approximate circuits, depending on the fault value and the parity of each line. This is better explained with the aid of an example. Consider an XNOR gate, an archetypical case of a binate circuit, represented

in Figure 6.3 along with its unate expansion. Consider the fault stuck-at 1 located at input b, as in Figure 6.4a. According to Figure 6.4b, it is clear that this is a binate fault, because it produces differences in both the onset and offset of the logic function of the gate. Approximating this fault requires then approximating one of the two lines in

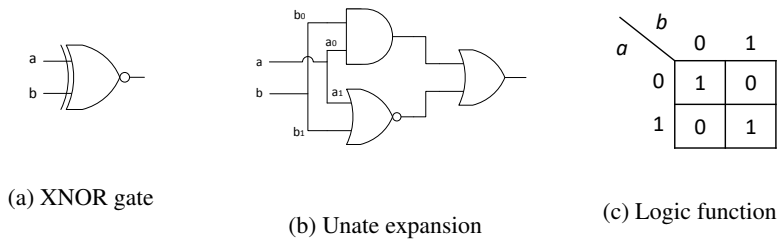


Figure 6.3: XNOR gate

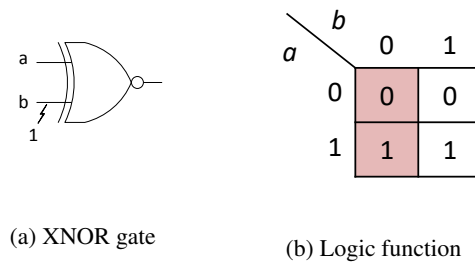


Figure 6.4: Binate fault b stuck-at 1

which input b is split during unate expansion (either b_0 or b_1) in one approximation, and the other line in the opposite approximation. If the positive line is approximated (b_0), the circuit of Figure 6.5 is obtained, which turns out to be an over-approximation with respect to the original XNOR gate. Conversely, approximating the negative part of input b (b_1) an under-approximation is generated, as shown in Figure 6.6. This procedure requires a mechanism to identify the two lines resulting from the splitting of the original binate line.

It may happen in some cases that approximating a binate fault by splitting it into two complementary unidirectional faults leaves more input vectors unprotected than the original fault. This phenomenon is related to the auto-cancellation effect, that is, when a fault propagates simultaneously through several paths with reconvergent fanout, in such a way that the fault propagation is eventually blocked. If the paths involved in the auto-cancellation have opposite parities, then approximating that fault with the proposed method cannot take advantage of the auto-cancellation effect. As an example, consider the binate circuit of Figure 6.7a. In this circuit, inputs c and d are binate, as well as the output of node n1. Any fault in any of these lines will affect both the

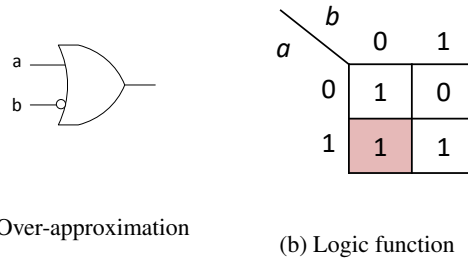


Figure 6.5: Approximating the positive part of $b/1$

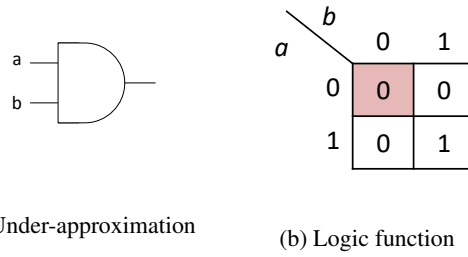


Figure 6.6: Approximating the negative part of $b/1$

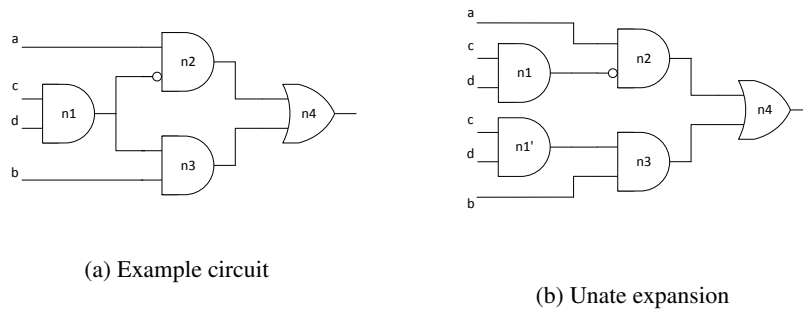


Figure 6.7: Auto-cancellation example

circuit onset and offset, as shown in Figure 6.8 for the case of fault d stuck-at 1. In order to approximate this fault, first the unate expansion has to be performed in the target circuit, resulting in the unate circuit of Figure 6.7b where both inputs c and d have been duplicated. Then each one of the two instances of input d is independently approximated. By approximating the positive part, that is, the fault $d \rightarrow n1'/1$, the resulting circuit will implement the logic function shown in Figure 6.9a, which is clearly an over-approximation with respect to the original circuit. On the other hand, approximation of the negative part (fault $d \rightarrow n1/1$) will generate the logic function of Figure 6.9b, which is an under-approximation. It can be checked that the under-approximate

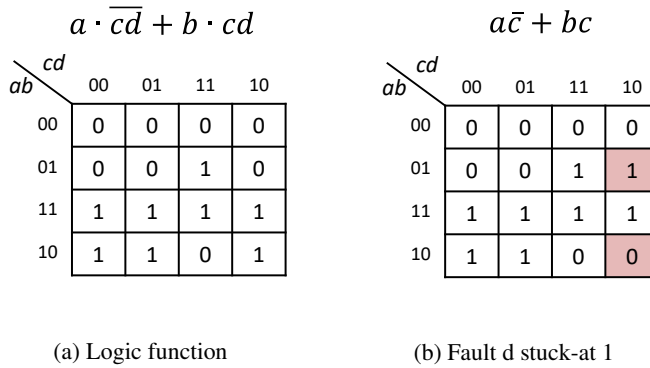


Figure 6.8: Effect of binate fault in logic function

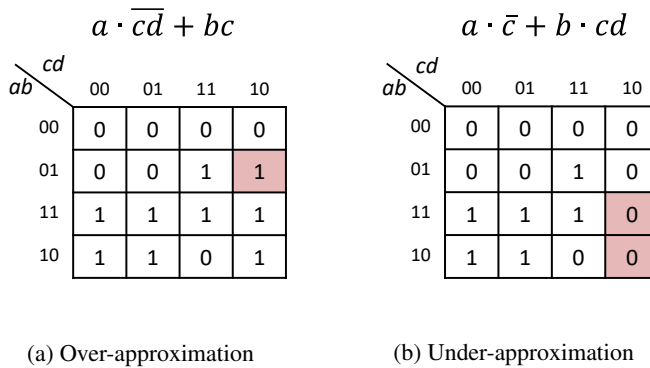


Figure 6.9: Approximation of binate fault $d/1$

logic function uncovers an additional discrepant input vector which was not considered for the binate fault $d/1$ (compare with Figure 6.8b). In fact, the input vector $abcd = 1110$ allows the propagation of fault $d/1$ through both existing paths, causing to be cancelled itself. This effect is suppressed when the two unidirectional components of

the fault are split in different approximate instances, and so an additional input vector is unprotected. The consequence of this phenomenon consist in a slight underestimation of some binate fault sensitivities.

With this approach the unate expansion is just necessary as an intermediate step to generate the approximate circuits. Therefore, the use of the unate version of target circuit for either obtaining the testability measures or working as one of the ATMR instances is no longer necessary. In addition, approximating a binate fault reduces area in both approximate circuits, thus minimising the overheads due to the unate expansion.

6.2.4 Radiation experiments

The radiation experiments for the proposed approach were conducted with the b13 benchmark from the ITC'99 set. This sequential benchmark was selected for compliance with current efforts towards a common set of benchmarks that can be used for comparison among different experiments [62]. It also includes a set of pre-generated input vectors that were designed by the ATPG community to fully cover the functionality of the circuit.

The criticality of the different faults within the circuit is estimated by fault simulation, as explained in chapter 4. The estimations have been performed by means of parallel fault simulator HOPE [45] against the set of pre-generated input vectors for the selected benchmark. The testability of each fault is estimated by counting the number of clock cycles for which the faulty circuit response differs from the correct one. Then, a fault is considered as critical if it produces a large amount of differences. The rationale of this criticality metric is that a fault that produces a very different output response implies a large data loss and most likely means the circuit functionality cannot be recovered. Conversely, an error that produces an almost correct response involves some data loss but the circuit can be considered as being operational.

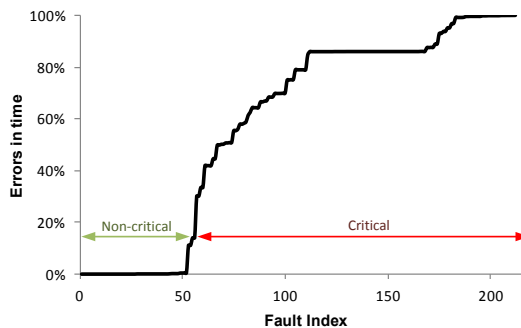


Figure 6.10: Fault testability analysis of b13 benchmark

Figure 6.10 shows the results of this testability analysis for b13 benchmark. Faults are ordered according to their testabilities, in an increasing order. It can be seen that

there is an abrupt change around 25% of the faults. The faults on the left can be considered as non-critical and the faults on the right as critical. In order to establish a threshold, a fault is considered as critical if it produces an erroneous response for more than 15% of the clock cycles in a representative testbench. This threshold can be changed according to the reliability requirements of the application. For the particular threshold used, it has been verified that the non-critical faults actually belong to feed-forward logic. Therefore, this criterion coincides with the feedback criterion proposed in [60] for the studied circuit.

Based on the testability analysis results, several ATMR schemes have been implemented by using approximate logic circuits generated with different testability thresholds. In total, 4 different ATMR schemes were used, from A1 to A4, where A1 has the lowest proportion of approximated faults and A4 the highest. As a general target, only approximations that produce less than 15% erroneous responses in the execution of the full set of input stimuli were considered. Then, for each design, faults with a percentage of errors below a selected threshold were forced. Namely, thresholds of 0.05%, 0.2%, 2% and 15% have been used for A1 to A4 designs, respectively. For the sake of comparison, a full TMR version of b13 benchmark has been included, in which the whole logic has been triplicated. All these designs (A1 to A4 and TMR) have been implemented with triple voters at the output of every flip-flop. The outputs of the circuit were also voted using single voters. Inputs and clock lines were not triplicated for the sake of simplicity. The set of implemented designs is completed with a full TMR design using just single voters (SV) and the original unmitigated benchmark (ORIG). The synthesis results for all these designs, as given by Xilinx Vivado tool, are shown in Table 6.1. The designs are ordered from the most protected (TMR) to the less protected (ORIG). The last two columns show the relative use of resources with respect to the full TMR version with triple voters.

Design	#LUTs	#FFs	%LUTs	%FFs
TMR	135	298	100%	100%
A1	127	287	94%	96%
A2	119	276	88%	93%
A3	117	265	87%	89%
A4	115	261	85%	88%
SV	135	213	100%	71%
ORIG	45	53	33%	18%

Table 6.1: Synthesis results for b13 benchmark

The experiments were run on an Artix7 XC7A100T FPGA from Xilinx. As the b13 benchmark is rather small, 24 instances of each design were included in the same FPGA. Synthesis options were used to ensure instances of each circuit version were not optimized away. However, the placement was not constrained. All designs run concurrently using the same input stimuli, which are the ITC99 proposed input stimuli. Also a small checker circuit was included to detect if any of the design copies produces an error or if the percentage of errors in a single execution of the full set of input stimuli is greater than the selected target of 15%. The checkers and the interface to the host are

tripled to reduce the impact of errors in these modules on the measures. The complete circuit, including all copies of all designs, used 66% of the LUTs and 15% of the ip-ops of the FPGA device.

Radiation ground testing has been carried out in CNA facilities (Centro Nacional de Aceleradores, Sevilla, Spain). A cyclotron, capable of accelerating protons and deuterons up to 18 MeV, has been used for the experiments. Although it was originally intended for radioisotope production in medical applications, an external beam allows testing electronic circuits in either vacuum or open air. FPGA test has been performed with protons in open air, with 18 MeV energy and flux range between 10^8 to $10^9 p/cm^2 s$. In total, three runs have been performed with different beam characteristics, as shown in Table 6.2. A single device has been exposed, which was allowed to run without scrubbing or reconfiguration it until it was observed most of the design versions had a critical error. Then it was fully reconfigured and checked again. During the run number 3, the one with the largest fluence, up to 120 reconfiguration cycles were completed.

Run	Energy (MeV)	Flux ($p/cm^2 s$)	Fluence (p/cm^2)
1	18	10^8	$1.23 \cdot 10^{11}$
2	18	$2.5 \cdot 10^8$	$11 \cdot 10^{11}$
3	18	10^9	$50.3 \cdot 10^{11}$
Total			$62.53 \cdot 10^{11}$

Table 6.2: Beam characteristics

The results of the radiation test with the largest fluence are shown in Figure 6.11, in terms of Mean Time To Failure (MTTF) for each one of the designs. The results for lower fluences are similar, but they are not reported because the number of collected errors is small and the confidence intervals are wide. Cross-section can be obtained as the inverse of the product of MTTF times the flux ($10^9 p/cm^2 s$). For each version, the non-critical MTTF was measured as the mean time until the first error is detected in any of the 24 design copies. The critical MTTF was measured as the mean time until the first critical error is detected, i.e., until the percentage of erroneous responses in a single execution of the full set of input stimuli is greater than the critical threshold of 15%. The condence intervals for 95% condence level are also displayed.

As expected, the original unmitigated design shows a low MTTF in comparison with the mitigated versions. The critical and non-critical MTTF are very close, because most errors produce a critical effect. The version using single voters improves MTTF with respect to the original version by a factor of approximately 2.4. For the full TMR and all the ATMR versions, the non-critical MTTF is similar and between 6 to 8 times greater than that of the original design. Even though some degradation of the non-critical MTTF can be expected for the approximate logic circuits, the differences in the results are small and some ATMR versions can even be better than full TMR. On the other hand, the results clearly show that the MTTF for critical errors is significantly improved in the ATMR versions. There are two explanations for this behaviour:

- On the one hand, the reduction in size produced by the use of approximate logic

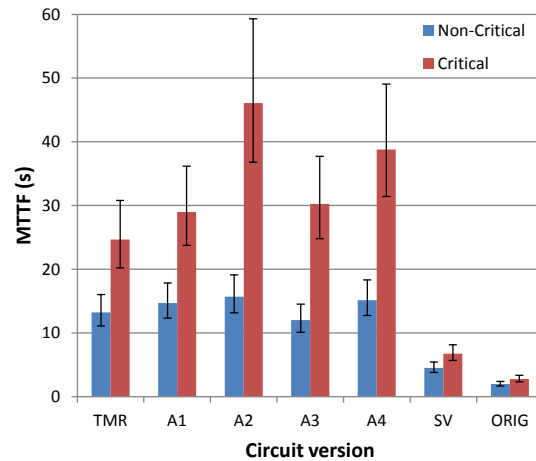


Figure 6.11: Results of radiation experiments

circuits makes the ATMR scheme less sensitive. A smaller size implies a smaller probability of receiving a particle strike, and therefore a smaller probability of suffering from SEEs.

- On the other hand, the approximation of unidirectional faults may cause a shift of the critical cross-section to the non-critical cross-section. To illustrate this, consider a line within a circuit which is triplicated in a TMR fashion. For instance, let us assume that the fault stuck-at 1 on that line is critical while the opposite is not. In such a case, the fault stuck-at 0 can be approximated, so the line can be replaced by a constant 0 in just one of the approximate circuits. The overall effect is equivalent to having a non-critical unidirectional hardwired error in one of the three subcircuits that are voted. The ATMR still works correctly in the absence of SEUs because there are always two correct copies for voting. The circuit is unprotected for errors in the same direction as the non-critical fault, because it is enough to have such an error in any of the two correct instances for the circuit to fail. However, the circuit is still protected for critical errors in the opposite direction. As a matter of fact, for such errors to be unmasked it is necessary that the two correct copies simultaneously have errors in the same direction as the critical fault. Critical errors are less likely to happen than in the TMR circuit, because in the TMR circuit an error is observed when two out of the three copies fail. Thus, the overall effect is a shift of the critical cross-section to the non-critical cross-section with respect to the TMR circuit.

For a better comprehension of these results, the radiation experiment has been complemented with several fault injection tests in FPGA. These are addressed in the next section.

6.2.5 Fault injection experiments

To complement the analysis, several fault injection campaigns have also been performed, using the same benchmark and the same designs than in the radiation experiments. First, a pair of fault injection campaigns with exactly the same FPGA mapping have been performed, using the Soft Error Mitigation (SEM) Core from Xilinx [63] for random fault injection both with and without accumulation. Later, additional experiments have been performed, using in this case the FPGA fault injection tool developed at the group *Concepção de Circuitos Integrados* from Universidade Federal do Rio Grande do Sul [64]. This tool allowed performing an in-depth study of the sensitive configuration bits in the design, i.e., the critical bits.

It must be noted that fault injection results may not accurately match radiation results due to several reasons [65–67]. First of all, fault injection must rely on the mechanisms provided by the manufacturer for the access to the configuration memory. The documentation about such mechanisms is generally very limited, so that a fair fault injection campaign cannot be guaranteed. In particular, fault injection cannot emulate changes in the internal proprietary state of an FPGA (i.e., internal registers or global logic for managing the device). On the other hand, only single bit errors in the configuration memory have been injected and it is assumed that all configuration bits are equally vulnerable, which may not be true in real devices. In spite of these limitations, fault injection plays a very important part in evaluating and validating FPGA designs for use in radiation environments [19]. It is shown here that fault injection results for the proposed approach follow a pattern similar to the one of radiation test results.

To implement fault injection, the SEM Controller has been added to the same FPGA design used for radiation testing. Fault injection was monitored through the serial interface provided by the SEM Controller. This interface allowed to observe the internal state of the SEM Controller and inject faults. The controller can also correct the internal configuration memory errors. This means that the SEM controller may be affected by faults, as faults are injected randomly in the configuration memory of the FPGA. However, this situation can be detected through the serial interface. When an error cannot be corrected or abnormal operation of the SEM Controller is observed through the serial interface, the FPGA is fully reconfigured and the fault injection process is resumed.

In the first fault injection campaign, faults were injected in random addresses of the configuration memory at regular time intervals. A sufficiently large time interval was selected to allow for the complete execution of the full set of input stimuli. Following the same approach as in the radiation testing experiment, the device was allowed to run without scrubbing or reconfiguration until it was observed that most of the design versions had a critical error, or the SEM Controller failed. Then the device was fully reconfigured and checked again. In this campaign, 215 reconfiguration cycles were run with a total of 389,764 injected faults.

Figure 6.12 summarizes the results obtained with this fault injection campaign. As a similar metric to MTTF, the ratio of injected faults to failures has been used, called here Mean Injected Faults To Failure (MIFTF). As in the radiation test results, the difference between the non-critical MIFTF and the critical MIFTF is small for the original unprotected design and for the design using single voters, while the protected versions

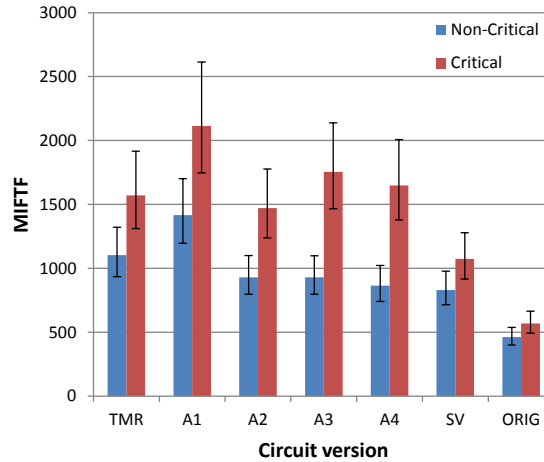


Figure 6.12: Fault injection results on b13 with accumulation

(TMR and A1 to A4) clearly improve the critical MIFTF. The Partial TMR versions show similar non-critical MIFTF to the full TMR version but the critical MIFTF is generally improved.

In a second fault injection campaign the error correction capabilities of the SEM Controller have been used. In this case, faults were injected in random addresses of the configuration memory one by one, waiting for sufficient time between injections to make sure the full set of input stimuli has been executed and then letting the SEM Controller attempt to correct the injected fault. This procedure is repeated until the SEM Controller reports an uncorrectable error or the SEM Controller itself fails. Then, the FPGA is fully reconfigured before continuing the fault injection campaign. Among the three different error correction modes the SEM Controller provides [63], in these experiments the Enhanced Repair mode has been used, which is based in both Error-Correcting Code (ECC) and Cyclic Redundancy Check (CRC) algorithms, because it provides higher error correction capabilities, namely correction of configuration memory frames with single-bit errors or double-bit adjacent errors. In this campaign 75,620 faults have been injected in a total of 86 reconfiguration cycles, allowing a time interval of 37.4 ms between faults.

The results for the fault injection campaign using error correction are shown in Figure 6.13. In this case it can be seen that the non-critical MIFTF decreases as the protection for non-critical errors decreases from TMR to A4. On the other hand, the critical MIFTF is similar in all these circuit versions. This is the expected behaviour, because these versions differ in the protection for non-critical errors but keep a similar protection for critical errors.

Comparing the results with and without error accumulation, shown in Figures 6.12

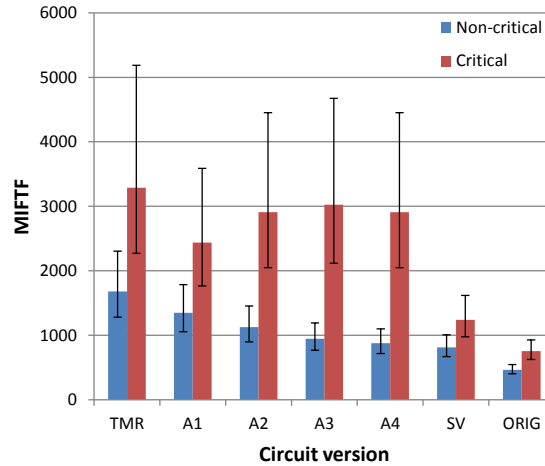


Figure 6.13: Fault injection results on b13 without accumulation

and 6.13, respectively, it can be seen that the full TMR version produces the best MIFTF when accumulation is prevented by error correction (Figure 6.13), while this may not be true with accumulation (Figure 6.12). The full TMR version has higher protection for non-critical errors at the expense of increasing the size of the circuit. The additional size increases the chances of error accumulation, for which the circuit is not protected, resulting in a smaller critical MIFTF. Thus, size plays a significant role when error accumulation is possible. By reducing the protection for non-critical errors, the approximate circuit versions can reduce the size and improve the critical MIFTF.

In order to explain these results, some other aspects must be considered. It is taken for granted that TMR is the best choice when full protection against faults is required, under the assumption that any individual fault will only affect just one of the circuit instances. While this statement holds for ASICs, it is not necessarily true in the case of FPGAs, specially for SRAM-based FPGAs. In fact, FPGAs are really complex devices, and a single fault in a configuration element may cause multiple errors. For example, a configuration bit may control the routing of a group of lines, or affecting a common mode signal such as the system clock. Therefore, in an FPGA, a single fault may potentially affect several instances of the TMR, a situation where TMR does not work. In this case, ATMR could be more beneficial in avoiding these common-mode errors, due to a lower resource utilization.

In order to prove this, additional tests have been performed by using a FPGA fault injection tool developed by the group Concepção de Circuitos Integrados from Universidade Federal do Rio Grande do Sul [64]. Similar to the Xilinx SEM Core, this is a tool for fault injection in FPGA through reconfiguration, that is, by modifying bits in the configuration memory. But unlike the SEM Core, this tool allows to define the area

were faults are going to be injected, thus ensuring that faults are injected exclusively in the design under test. In addition, this tool supports a mode for exhaustive evaluation of critical bits, apart from random injection of faults with or without accumulation. Critical bits are those configuration bits which can themselves cause an error when affected by a fault, without the need of fault accumulation. In other words, a single fault in any of these critical bits will cause a complete TMR to fail. The number and the position of the critical bits may vary depending on the particular design implemented in the FPGA.

A critical bit analysis for b13 benchmark has been performed with the tool mentioned above. For this purpose, the same designs tested in the radiation experiment as well as in the fault injection tests with the SEM Core have been used, namely: a full TMR with triple voters, four ATMR schemes built with different testability thresholds, a full TMR with single voters, and the original unmitigated design. In this case, each design has been independently tested and only one copy per design has been implemented, as there is full control of the area where faults are injected. In addition, each one of the components of the TMR-like schemes (i.e., each one of the three circuit instances plus the voter) have been independently mapped in adjacent sections of the FPGA, in order to distinguish the contribution of each module to the overall number of critical bits. For each design under test, the fault injection tool has sequentially injected faults in all bits in the configuration memory of the predefined area, repairing each fault prior to injecting the next one. The area for fault injection has been always the same independently of the size of the design under test. A configuration bit is identified as critical if it causes an erroneous output of the circuit at any time during the execution of a predefined testbench.

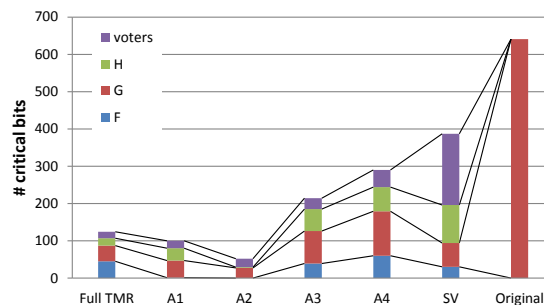


Figure 6.14: Critical bit analysis for b13 benchmark

Figure 6.14 shows the results of the critical bit analysis. For each design, the total number of identified critical bits is shown, but at the same time they are classified according to the module where they were detected: the original circuit (G), its under-approximation (F), its over-approximation (H) and the voters. In the case of the full TMR versions (labelled as TMR and SV), the circuits F, G and H are identical. The results show that the full TMR with triple voters presents much less critical bits than

the original unmitigated design (124 versus 641), but surprisingly this is not the most robust design. In fact, small approximations effectively reduce the overall number of critical bits up to a minimum of 52 for design A2, resulting from a testability threshold of 0.2%. In this case, the circuit approximations present almost no critical bits. But if additional approximations are performed, it grows again up to levels higher than the full TMR. It is worth to mention that the use of single voters instead of triple voters has a significant negative effect on circuit reliability, as it was expected. The presence of single points of failure in the design rises the number of critical bits, especially in the voter module, which passes from 20 critical bits on average to 191.

In conclusion, ATMR in FPGAs maybe more tolerant to faults than full TMR for two reasons. ATMR does not only occupy less area, thus reducing the probability for an energetic particle to collide and provoke a fault, but also the reduced number of configuration bits required may make the design more robust against common mode errors.

6.3 Microprocessor hardened by approximate TMR

Since the beginning on the research for this thesis, there always was the intention of applying the techniques developed on some real application circuits. The opportunity came thanks to a collaboration project with CISCO (USA), IROC Technologies (France) and the University of Saskatchewan (Canada) to fabricate a chip implementing a microprocessor hardened by ATMR.

The ARM Cortex M0 core has been the microprocessor selected in this project. This is a relatively small 32-bit microprocessor, optimized for small silicon die size, which is commonly used for university research projects. The Register Transfer Level (RTL) source code of the ARM Cortex M0 is made freely available, although in an obfuscated form.

Due to the complexity of the target design, the approximation generation technique based on static testability measures was preferred. In this case, the testbench used to obtain the testability measures consist in up to three software benchmarks.

The generated ATMR versions of ARM Cortex M0 core have been evaluated by means of AMUSE tool before fabricating the chip. Eventually, the objective of the project consists in evaluating and validating the ATMR approach for a real application circuit by means of laser and heavy-ion radiation.

This section is divided as follows. Subsection 6.3.1 describes the main objectives and specifications of the collaboration project, as well as the contribution of this thesis to the collaboration project. Then, section 6.3.2 describes the characteristics of the ARM Cortex M0 microprocessor. Next, section 6.3.3 explains how the approximate versions have been generated, including the software benchmark selection. Finally, section 6.3.4 introduces the experiments performed and the results obtained.

6.3.1 Specifications

The main objective of the project is the study of techniques for protecting combinational logic, considering a broad fault model. This fault model may include effects

such as radiation induced transients (SEUs and SETs), and manufacturing and degradation faults. This is also a good opportunity to demonstrate if experimental techniques can scale to large real application circuits, such as a microprocessor.

In order to make a reasonable comparison, several versions of the core would be implemented on the same chip. All versions would use DICE flip-flops to mitigate SEUs, so that the remaining soft errors would come purely from SETs, except for the reference version, which would have combinational logic produced using a standard synthesis flow. The protected versions would take the reference design as a starting point, and would then apply approximate logic techniques to protect the combinational logic. The chip would contain a small (4 KB) on-chip memory implemented using hardened memory cells to hold both code and data. The memory would be shared by the processors, as only one processor would be active at a time. This is a very reduced memory capacity, which requires a careful selection of software benchmarks and an intensive code and memory usage optimization.

The chip will be designed such that the different cores can be powered independently so that the power overhead of the approximate logic can be accurately quantified. All circuits will be tested using a two-photon pulsed laser in order to identify their relative sensitivities and to identify the most sensitive regions. The circuits would also be tested under heavy-ion radiation. It is noted that this study can include running multiple workloads on the processor, to investigate the effect of the workload.

The contribution of this thesis on such project consisted on the whole process of generation of approximate versions of the ARM Cortex M0 microprocessor, including the selection of software benchmarks and their optimization for the reduced memory requirements, the sensitivity analysis of the microprocessor for the set of selected software benchmarks, the generation of approximate versions according to the combined sensitivity results, and the preliminary reliability study of the resulting ATMR schemes.

6.3.2 ARM Cortex M0 microprocessor

The Cortex-M0 processor is a very low gate-count, highly energy-efficient processor that is intended for microcontroller and deeply embedded applications that require an area optimized processor. It is built on a high-performance processor core, with a 3-stage pipeline von Neumann architecture. The Cortex-M0 processor implements the ARMv6-M architecture, which implements the ARMv6-M Thumb instruction set, including Thumb-2 technology [1]. It includes a 32 bit multiplier, among two options: either a high performance single-cycle multiplier, or a 32 cycle multiplier, optimized for low area.

The Figure 6.15 shows the main components of the Cortex-M0 implementation. Apart from the microprocessor itself, it contains an interface to the Advanced Microcontroller Bus Architecture (AMBA) Advanced High-performance Bus (AHB) communication bus and a configurable interrupt controller. In addition, debug hardware can optionally be implemented.

In particular, the version of the microprocessor used in this project is the ARM Cortex-M0 DesignStart processor at revision r0p0. This is a fixed configuration of the Cortex M0 processor, enabling low cost access to Cortex-M0 processor technology by offering a subset of the full product. This version is delivered as a pre-configured

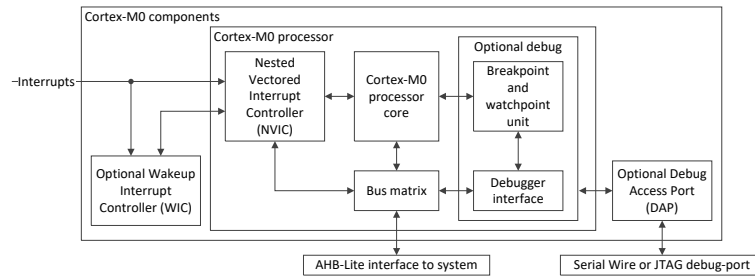


Figure 6.15: Cortex M0 architecture [1]

and obfuscated, but synthesizable, Verilog description of the full Cortex-M0 processor. These are some of the features of this version, as they are described in [68]:

- Master interface to AMBA AHB communication bus. No slave interfaces included.
- Small 32-cycle multiplier.
- Fixed number of interrupt inputs: 16.
- Absence of hardware debug support.

The latest revision of the ARM Cortex M0 DesignStart processor can be freely obtained from [69].

The Cortex-M0 DesignStart processor is provided along with a lightweight simulation test-bench, which includes: the Verilog test-bench itself, an example program and corresponding memory image, and a simple Makefile.

The test-bench, shown in Figure 6.16, instantiates the microprocessor and connects it in a minimal way to a memory model and clock and reset generators. It also provides a means of outputting information from the processor to the console output of the Verilog simulator. It must be noted that the components within the test-bench are intended for simulation purposes only, and should be replaced with synthesizable counterparts before performing any synthesis [1].

An example memory image for the processor is provided by the *ram.bin* file. This is loaded by the memory model in the Verilog test-bench at the beginning of every simulation. The provided memory image is for a simple hello world program, which uses the test-bench to write a message to the simulator console, and then to terminate the simulation. The C source code of the program is provided too. In order to simulate an alternative program, it is required to modify the source code of the program, and later recompile it by means of an appropriate tool-chain, thus generating a new valid memory image [1]. In this project, the Keil MDK tools have been used to compile the different selected software benchmarks.

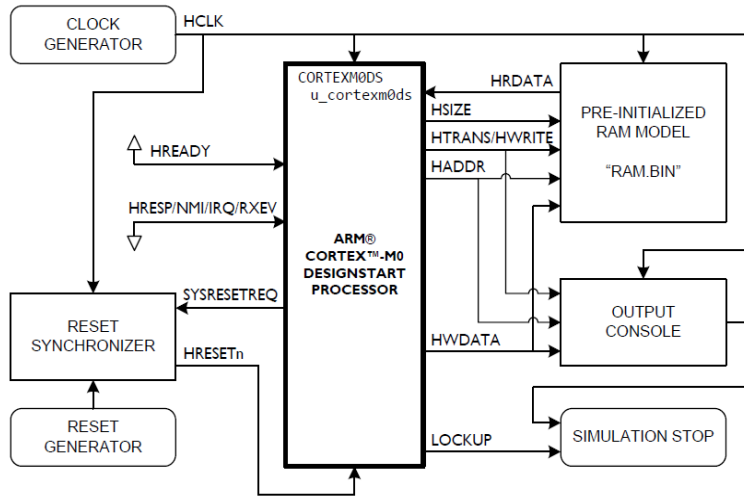


Figure 6.16: Cortex M0 test-bench [1]

6.3.3 Generation of approximate microprocessor versions

The first step in the generation of approximate versions of the microprocessor consist in selecting the software applications which will be used to obtain the testability measures. For this purpose, several programs from the MiBench benchmark suite [70] were selected. MiBench is a freely available set of commercially representative software benchmarks for embedded applications. It is divided in six categories, thus reflecting the huge diversity of the embedded applications market: automotive and industrial control, network, security, consumer devices, office automation, and telecommunications. Among all the 36 benchmarks which compound MiBench, a small group has been selected to perform the sensitivity analysis. With the idea of covering a wide enough set of microprocessor functionalities, one software benchmark per group was initially chosen, except for the consumer devices group. None of the benchmarks in this group has been selected because they are multimedia applications and therefore its memory requirements are expected to exceed the tight memory constraints. The initial benchmark selection was composed of the following applications:

- qsort (from the automotive and industrial control group), which implements the quick sort algorithm for sorting an array of elements in ascending order.
- dijkstra (network), an algorithm for finding the shortest path between two nodes.
- sha (security), the security hash algorithm, which produces a 160-bit message digest for a given input.
- stringsearch (office automation). This application searches for a chain of text into another chain of text using a case insensitive comparison algorithm.

- CRC32 (telecommunications), which performs a 32-bit Cyclic Redundancy Check for a given input.

The initial selection of benchmarks were tested in the simulation test-bench provided with the Cortex-M0 DesignStart processor. The test-bench was modified to implement a reduced memory of just 4 KB (by default, the test-bench creates a memory model with 256 KB capacity). Similarly, all selected software benchmarks were configured with reduced program data, and re-compiled with the Keil MDK compiler. Among the tested software benchmarks, djikstra and sha were not able to successfully run with the limited memory size, and therefore they were excluded. In summary, the final selection of benchmarks was reduced to qsort, stringsearch and CRC32 applications. Table 6.3 contains the memory sizes and execution times of each one of these benchmarks. The size of the different memory regions were provided by the Keil MDK compiler, while the execution time has been measured by simulating the microprocessor test-bench with the appropriate memory image.

Benchmark	Memory size				Total (KB)	Execution time (#clock cycles)
	Code (Bytes)	RO data (Bytes)	RW data (Bytes)	ZI data (Bytes)		
qsort	1512	328	16	1120	2.91	8743
stringsearch	848	360	24	2144	3.30	21203
CRC32	580	380	1040	1120	3.05	4322

Table 6.3: Software benchmarks data

The memory image generated by the Keil MDK compiler is divided into sections. After code compilation, the linker allocates memory for the different sections required for program execution. There are different types of sections, depending on the type of data contained within them. Each section type has a different behaviour in the memory image [2]:

- Read-Only (RO) section, which contains the program code and read-only data. The location of an RO section in the execution view is the same as in the load view.
- Read/Write (RW) section, which contains pre-initialized variable program data. Its location in load and execution views can be different.
- Zero-Initialized (ZI) section, which allocates non-initialized data used during program execution, including stack and heap. This section is loaded with zeros at the beginning of execution, and therefore only exists in the execution view.

A memory image has two different views. First, when the code program and input data are initially loaded into memory (load view), and later, the set of memory addresses which are expected to be accessed during program execution (execution view). There are several ways in which the different sections can be allocated in the memory image. In order to optimize the limited memory space, it has been adopted a memory image

structure similar to which appears in Figure 6.17, where the different regions are contiguous and the RW section in both load and execution views share the same range of addresses [2]. The base address for the RO section has been set to 0X0000.

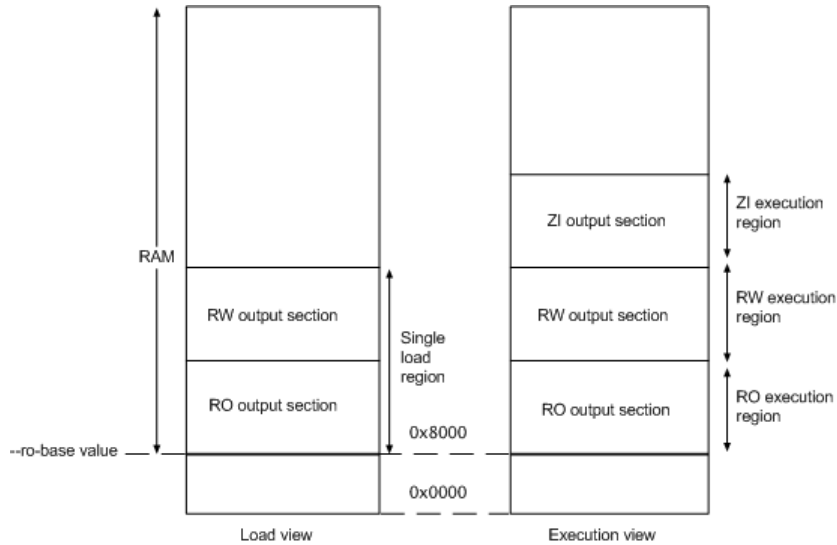


Figure 6.17: Implemented memory image [2]

The total memory size indicated in Table 6.3 is referred to the execution view, in other words, the ZI section is already taken into account. Therefore, it is ensured that these three software benchmarks are capable of being successfully executed with a small memory of 4 KB.

The next step consists in obtaining the testability measures by means of fault simulation. This is performed with the aid of the fault simulator HOPE [45]. But this software requires, on the one hand, the description of the simulated circuit in BENCH format, and on the other hand, a set of input vectors which represent the software benchmarks intended to be executed. The input vectors are obtained by simulating again the DesignStart test-bench. For this task, the simulation test-bench has been modified to read the inputs of the microprocessor on every clock cycle, and writing them in a text file. Three different sets of input vectors have been generated, corresponding to each selected software benchmark. The description of the Cortex-M0 processor in the appropriate format is obtained in two steps. First, the obfuscated description of the microprocessor is synthesized by means of the Synopsys software against a Complimentary Metal-Oxide-Semiconductor (CMOS) 28nm synthesis library, thus obtaining a synthesized netlist. Later, the netlist is translated into BENCH format by means of a custom made parser.

Once the preparations are complete, the fault simulations can be performed. Each input vector set has been independently simulated, thus obtaining three different fault sensitivity reports. The results of the fault simulation step are summarized in the Figure 6.18, with the faults ordered according to their testabilities in an increasing order.

Each line represents the results of a particular software benchmark, while the "Total" line represents the combination of all of them. It must be noted that each software benchmark has a different duration, and because of this the number of observed errors is referred as a fraction of the whole input vector set rather than in absolute numbers. It is interesting to note that around half of the faults within the processor produce a very low number of errors. This is beneficial for the logic approximation approach, as great savings can be obtained with very low impact in fault mitigation capabilities.

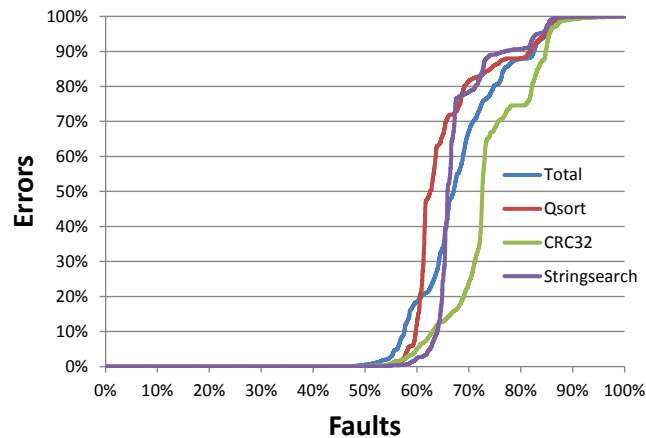


Figure 6.18: Fault testability analysis of ARM Cortex-M0

After obtaining the fault testability measures, the unate version of the microprocessor is generated, as well as its approximate versions. The full unate expansion has been employed for generating the unate version, including replacing the XOR and XNOR gates with their AND-OR equivalent (readers are referred to Section 3.4 for details). The approximate versions have been generated from the unate version of the microprocessor and the fault testability reports by means of the same custom-made BENCH parser used through the whole thesis. This tool has been modified in order to accept multiple testability reports as inputs, combining the results of all of them to decide which faults become approximated for a given threshold, which is specified by the user. According to the results obtained from the fault simulations (Figure 6.18), thresholds with the values of 0.1%, 1%, 20% and 50% have been set, thus generating approximate circuits with different trade-offs between area savings and protection against faults. All the resulting circuits (in BENCH format) were translated to Verilog by means of the ABC synthesizer [47], and then synthesized with Synopsys against the CMOS 28nm synthesis library. In this way, the logic constants introduced in the approximate version of the target processor are simplified. Finally, voters for both the flip-flops and primary outputs were properly placed in order to build an ATMR scheme for each pair of generated approximate circuits.

Th.	Number of cells				Combinational area			
	Under	Over	Total	Ov.(%)	Under	Over	Total	Ov.(%)
50%	687	795	1482	23.00	471.5	531.5	1003.0	21.49
20%	1318	1179	2497	38.76	865.3	794.5	1659.7	35.55
1%	2076	1915	3991	61.94	1397.2	1309.8	2707.0	57.99
0.1%	2270	2305	4575	71.01	1549.4	1606.6	3155.5	67.59

Table 6.4: Synthesis results

Table 6.4 shows the synthesis results for the approximate circuit versions. It contains the area of both approximations for each testability threshold in both the number of cells and the combinational area, as well as the combined area of both circuits and the total area overhead with respect to the synthesized ARM Cortex-M0. These results do not take into account the area overhead due to the voters, which represent an additional 13.58% in terms of number of cells, or a 21.41% in terms of combinational area. Due to the use of DICE flip-flops, which are immune to SEUs, it was decided not to replicate the flip-flops in order to save some area. For the same reason, the ATMR has been implemented with single voters. This solution presents the problem of making the voters a single point of failure: any fault affecting the voters cannot be masked in any case, thus generating an error.

Approximations of 0.1%, 20% and 50% have been chosen to be implemented in the final design, as there was not enough place for allocating all the ATMR versions. In addition, a full TMR version with DICE flip-flops and a plain unmitigated ARM Cortex-M0 have been implemented, for the sake of comparison. All the microprocessor versions share a small RAM memory of 4 KB integrated in the design, which is triplicated as a way of fault mitigation. Only one of the 5 ARM Cortex-M0 versions is allowed to be active at any given time, thus ensuring a correct memory usage. There is a control logic implemented to choose which microprocessor is active on each moment. The design is complemented with two flip-flop chains, one of them made of DICE flip-flops, intended to eventually measure the cross-section due to SEUs. The expected operation frequency for the whole design is about 550 MHz. The layout of the design has been performed at the University of Saskatchewan. Finally, it has been sent to fabrication.

6.3.4 Experiments

Once the ATMR versions of the ARM Cortex-M0 were generated, they were tested by means of the AMUSE tool [46] as a way of evaluating their fault mitigation capabilities in advance. Later, these results would be compared with those obtained from the final radiation experiments.

The setup for the tests with AMUSE is similar to the procedure already explained in section 4.4.1. Once the generated circuits have been synthesized into a netlist format, they are instrumented by means of VIOLIN software, a companion tool of the AMUSE system. In that way, the components in the synthesized netlists are replaced with alternative versions which support fault injection and evaluation. In this group of

tests, faults are injected in all the elements of the ATMR scheme: the target circuit, its approximate versions and the voters. Additionally, a synthesizable testbench has been implemented, consisting in a memory which is loaded with the different sets of input vectors corresponding to each software benchmark, the same sets which have already been obtained for performing the testability analysis. Then, the CUT is implemented by interconnecting all the instrumented circuits including the synthesizable testbench, and integrated in the AMUSE tool.

To perform the tests, the AMUSE tool has been implemented in a Xilinx Virtex5 XC5VLX110T FPGA by means of Synplify and Xilinx ISE software tools. For each one of the generated ATMR schemes, all the three software benchmarks have been tested. Two fault emulation campaigns have been performed on each case: one injecting transient pulses with the complete duration of the system clock cycle, and another injecting pulses with a duration of a 10% of the full system clock cycle. An exhaustive analysis has been performed on each test, injecting faults in every gate output for every input vector applied.

Th.	Benchmark	Full clock cycle			10% clock cycle		
		Silent	Failure	Latent	Silent	Failure	Latent
50%	qsort	68.25%	19.97%	11.78%	96.74%	2.12%	1.14%
	CRC32	67.23%	21.15%	11.62%	96.67%	2.20%	1.13%
	stringsearch	67.63%	20.60%	11.77%	96.72%	2.14%	1.14%
	Total	67.74%	20.51%	11.75%	96.72%	2.14%	1.14%
20%	qsort	72.09%	17.36%	10.55%	97.13%	1.84%	1.02%
	CRC32	71.33%	18.29%	10.38%	97.09%	1.90%	1.01%
	stringsearch	71.54%	17.91%	10.55%	97.11%	1.86%	1.02%
	Total	71.66%	17.82%	10.53%	97.12%	1.86%	1.02%
1%	qsort	77.05%	13.85%	9.10%	97.65%	1.47%	0.88%
	CRC32	76.28%	14.76%	8.96%	97.60%	1.53%	0.87%
	stringsearch	76.18%	14.68%	9.14%	97.59%	1.52%	0.89%
	Total	76.42%	14.48%	9.11%	97.60%	1.51%	0.88%
0.1%	qsort	80.46%	11.67%	7.87%	98.00%	1.24%	0.76%
	CRC32	80.14%	12.07%	7.78%	98.00%	1.24%	0.76%
	stringsearch	80.19%	11.95%	7.86%	98.00%	1.24%	0.76%
	Total	80.26%	11.89%	7.85%	98.00%	1.24%	0.76%

Table 6.5: Results of ATMR fault emulation with AMUSE

Table 6.5 contains the results of the fault emulation campaign. The table shows, for each ATMR scheme, the fault classification for each pulse length and each software benchmark, as well as the combination of the three. The AMUSE tool classifies the faults in three categories: no effect (silent), fault propagated to circuit outputs (failure) and faults which cause a long-lasting change in the internal state of the circuit, so it might produce an error in the future even though the test execution is complete (latent). It can be seen that the results are practically the same independently from the executed software benchmark. On the other hand, it is clear that the lower the testability threshold, the higher number of faults that are masked. This makes sense, because

a lower testability threshold implies a higher degree of similarity between the target circuit and its approximate versions. The percentage of masked errors for pulses with the full duration of the clock cycle may seem not so good (around 80% in the best case), but this has to be considered as a worst case scenario. Usually transient pulses are much shorter than this, which are far easier to mask. This is reflected in the results with transient pulses with a duration of 10% of the clock cycle, where the proportion of faults masked is much better (between 96 % and 98%).

The final radiation experiments are still pending.

6.4 Comparison with evolutionary techniques

Evolutionary algorithms are used in many applications to solve hard optimization and design problems. They take advantage on the high computation capacity of latest technologies in order to explore a wide range of solutions by trial and error. Because of this, evolutionary algorithms are usually very time consuming, but, on the other hand, capable of discovering solutions that are hard to reach by other methods. In the context of this thesis, evolutionary algorithms have been applied as an alternative way of generating approximate versions for a given target circuit, with the ultimate goal of mitigating faults in an ATMR scheme. This has been done in collaboration with Brno University of Technology (Czech Republic), where there is a research group expert in evolvable hardware.

Among the different existing evolutionary algorithms, the chosen algorithm for the generation of approximate logic circuits is known as *Cartesian Genetic Programming* (CGP). Basically, it consist in a fixed-size array of reconfigurable logic where circuits are generated. Taking the target circuit as the initial seed, random *mutations* (i.e. modifications) are performed, thus obtaining a group of mutated circuits, typically small, which is called a *generation*. Every circuit in the generation is evaluated with respect to a predefined *fitness function*, and the best candidate is chosen as the seed for the next generation. This process can be iteratively repeated as long as desired.

One of the major advantages of evolutionary algorithms in general, and CGP in particular, is the ability to get out from local minima and increase the chances to reach global minima, achieving better trade-offs between overheads and fault masking capabilities. Thus, CGP can provide radically different solutions from the circuit approximation approaches previously proposed in this thesis. A comparison with the approximation generation method based on dynamic testability measures (without node substitution) is carried out in this work in order to contrast their respective capabilities in terms of computational time and characteristics of generated solutions. Because it is expected that approximations generated with CGP will be closer to the optimal solution, this analysis is also a way to measure how well the deterministic dynamic approach works.

The remaining of this section is structured as follows. Subsection 6.4.1 first briefly introduces the state of the art related with evolutionary algorithms. Then, subsection 6.4.2 explains how works the Cartesian Genetic Programming algorithm used to generate approximate circuits. To conclude, subsection 6.4.3 summarizes the setup for the experiments performed, and subsection 6.4.4 presents their results.

6.4.1 Introduction to evolutionary logic

Since the very beginning of the research in evolutionary computation, evolutionary algorithms have been applied for purposes of hardware optimization. Several monographs [35, 36] summarize the applications from the field of electronic circuits design, diagnostics, and testing. Later, evolutionary algorithms were applied to generate complete circuit structures (i.e., not only to optimize parameters of existing circuits) and dynamically adapt circuit structures [37]. For example, in the area of dependability, an evolutionary repair method was proposed for TMR implemented into FPGAs [71]. It employs an evolutionary algorithm to repair one damaged module of TMR by using the two healthy modules as sources of golden data. An analysis has shown a significant improvement of reliability for small benchmark circuits.

The evolutionary design of combinational circuits has been well established in the past. The majority of designs in this area is conducted by CGP or methods similar to it. CGP is a branch of Genetic Programming (GP) introduced by Miller and Thomson [72]. Unlike GP, which uses a tree representation of circuits, an individual in CGP is represented by a directed acyclic graph of a fixed size. The candidate circuits can have multiple outputs and intermediate results can be reused. CGP can be used to design various types of circuits as surveyed in [73]. Basically, CGP is an iterative algorithm where, on each iteration, several candidate circuits are generated by means of random modifications of a given seed. Later, those generated circuits are evaluated with respect to a predefined fitness function, and the best candidate with respect to that criteria is taken as the seed for the next iteration (also called generation) of the algorithm. The number of iterations can be as high as desired, although there are some recommended parameters in the literature, depending on the characteristics of the input circuit [73].

One of the most important components of the evolutionary circuit design consist in formulating the fitness function. It usually contains several objectives (functionality, area, delay etc.) where the functionality is typically understood as the quality of the candidate circuit measured as the number of correct output bits compared to a specified truth table (i.e. the Hamming distance). In order to obtain a fully working circuit, all combinations of input values have to be evaluated. For a circuit with n_i inputs and n_o outputs, 2^{n_i} test vectors need to be fetched to the primary inputs and $n_o \cdot 2^{n_i}$ output bits have to be verified so as to compute the fitness value. The fitness calculation is computationally very intensive, since the number of test vectors grows exponentially with the number of primary inputs. Recently, it has been sped up by applying parallelism at various levels (data, thread, process) [74] or by introducing formal methods based on, for example, SAT solving [75].

When designing digital circuits with respect to multiple secondary objectives, such as area, latency or power consumption, or with the goal to approximate circuit behaviour, one can make use of several approaches. The single-objective approach can be extended to deal with multiple objectives either by combining the objectives in a single fitness function just by summing the particular fitnesses weighted with a constant or, in a more sophisticated way, by introducing a multi-stage fitness function activating the particular objectives step by step. Thanks to the fixed size of the CGP genotype, resources can be constrained in order to find circuits with smaller area or power consumption [38]. Recently, a truly multi-objective approach applied to the de-

sign of (approximate) digital circuits has been proposed [39]. None of these methods, however, had been used before to approximate circuits in a TMR fashion.

6.4.2 Cartesian genetic programming

The proposed evolutionary method used to generate approximate circuits is based on CGP, and therefore it is worth to mention how CGP works.

The CGP approach defines a reconfigurable cartesian grid of nodes with a fixed size of n_r rows and n_c columns (shown in Figure 6.19), where all the candidate circuits are going to be generated. Every node in the grid has a fixed number of n_a inputs (typically

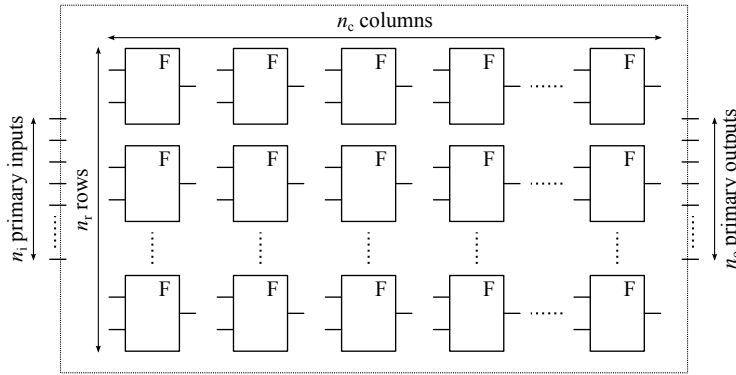


Figure 6.19: CGP reconfigurable grid

equal to 2) and can perform any of the combinational logic functions from a predefined set Γ . Nodes are interconnected by a feed-forward network, where node inputs can be connected to either one of the n_i primary inputs or to an output of any node in the preceding L columns. In that way, it is ensured that combinational loops are avoided. Each one of the n_o primary outputs has to be connected to either a primary input or to the output of any node in the grid. Thus, The area and delay of the circuit can be constrained by changing the grid size and the L -back parameter.

Every circuit implemented in this grid can be represented as an array of integers denoted as *genotype*. In the genotype, each two-input node in the reconfigurable grid is encoded using three integers: an address for the first input, an address for the second input, and an index representing the logic function of the node. Connection addresses are encoded in the following way: first, primary inputs are numbered from 0 to $n_i - 1$, and then, output nodes are labelled from n_i to $n_i + n_c \cdot n_r - 1$. Finally, for each primary output, the genotype contains one integer specifying the connection address. Thus, the genotype size is $(n_a + 1) \cdot n_r \cdot n_c + n_o$ genes (integers). Nevertheless, a circuit does not necessarily have to use all available nodes. Although the generation of candidate circuits is conducted at the level of genotypes, the fitness function evaluates *phenotypes* (actual circuits established according to the genotypes). While the genotype is of fixed length, the size of the phenotype depends on the number of inactive nodes, i.e. nodes whose output is not used by any other node or primary output. Since the inactive nodes

have no influence on the phenotype, there may be individuals with different genotypes but the same phenotypes.

An example of a CGP individual with its chromosome can be seen in Figure 6.20. It has three inputs, one output and three active nodes. The cartesian reconfigurable grid in this example has a size of 1x4 nodes, so this genotype presents one inactive node.

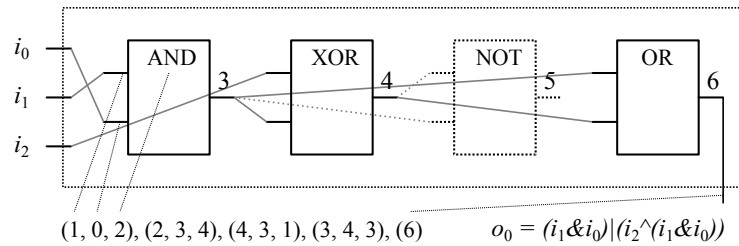


Figure 6.20: CGP individual example

The initial population, i.e. the circuit which serves as the seed for the first generation of circuits, can be constructed either randomly (in the case of evolutionary design) or by mapping of a known solution to the CGP chromosome (in the case of evolutionary optimization). Because the application of CGP in this thesis consists in generating approximate versions of a given target circuit, such target circuit is used as the initial population.

CGP uses a simple mutation based $(1 + \lambda)$ evolutionary strategy as a search mechanism. The population size on each generation, $1 + \lambda$, is mostly very small, typically, $\lambda = 4$. Each one of the individuals in the generation is obtained by means of random modifications of the seed. The mutation rate m is usually set to modify up to 5% randomly selected genes. In each generation, the best individual is passed to the next generation unmodified along with its λ offspring individuals created by means of a point mutation operator. In case more individuals with the best fitness exist, a randomly selected one is chosen. The maximum number of generations created in a single run is N_g . The role of mutation is significant in CGP (detailed analysis can be found in [76, 77]).

The fitness function is in charge of evaluating which is the best candidate on each generation, which will be the seed for the next generation of circuits. Therefore, defining a proper fitness function constitutes a critical task, as it may favour some circuit characteristics over others. According to the experience of the partners from Brno University, in order to generate approximate versions of a given circuit it has been decided to use a multistage single-objective approach with constrained resources. The fitness function f_{under} used to find under-approximations has been defined as follows:

$$f_{under} := \begin{cases} f_{hamm}^{max} + (f_{area}^{max} - f_{area}) & \text{if } f_{hamm} = 0, \\ f_{hamm}^{max} - f_{hamm} & \text{if } f_{off} = 0, \\ f_{off}^{max} - f_{off} & \text{otherwise,} \end{cases}$$

where f_{hamm} is the total Hamming distance between the outputs generated by the candidate solution and the original circuit for all possible input combinations, and f_{hamm}^{max}

is the size of the whole set of input vectors, $f_{hamm}^{max} = n_o 2^{n_i}$. f_{area} is the actual area of the circuit, i.e. the number of active nodes, while f_{area}^{max} is the maximum area according to chosen number of rows n_r and columns n_c . f_{off} is the number of $0 \rightarrow 1$ errors for all possible input combinations, i.e. the number of input vectors which have changed from 0 to 1 with respect to the original logic function. For any generated circuit to be a valid under-approximation, f_{off} should be equal to 0. Finally, f_{off}^{max} is the number of minterms (zeros) in the truth table of the original circuit.

How this fitness function works is summarized as follows. First, f_{under} evaluates whether the candidate circuit fulfils the condition of an under-approximation ($f_{off} = 0$) or not. If true, the fitness function then attempts to minimize the hamming distance between the candidate and target circuits, because the more functionally similar the approximate and target circuits are, the higher protection against faults is achieved. Finally, if both circuits are functionally equivalent, f_{under} tends to minimize the circuit area. All candidates with fitness $f_{under} \geq f_{off}^{max}$ represent a valid under-approximation.

Analogously, the fitness function f_{over} used to find over-approximations has been defined as follows:

$$f_{over} := \begin{cases} f_{hamm}^{max} + (f_{area}^{max} - f_{area}) & \text{if } f_{hamm} = 0, \\ f_{hamm}^{max} - f_{hamm} & \text{if } f_{on} = 0, \\ f_{on}^{max} - f_{on} & \text{otherwise,} \end{cases}$$

where f_{on} represents the number of input vectors which produces a 1 in the original circuit and 0 in the approximate circuit, and f_{on}^{max} is the number of maxterms (ones) in the truth table of the original circuit. For a circuit to be considered a valid over-approximation, $f_{on} = 0$. In other words, all candidate circuits with $f_{over} \geq f_{on}^{max}$ represent a valid over-approximation.

One can observe that since all possible input vectors have to be generated in order to evaluate the fitness function, the approach is not scalable. In order to speed up the design, parallelism at various levels (data, thread, process) can be introduced [74]. In practice, it is applicable for circuits containing less than approximately 20 inputs and 200 gates. More complex circuits can be optimized by introducing formal methods, e.g., SAT solvers or binary decision diagrams (BDD), however, an initial fully working solution is needed in this case [78].

6.4.3 Experimental set-up

The experiments have been conducted with a small selection of benchmarks from LGSynth93 set. Due to the constraints from the evolutionary algorithm side, the selection has been limited to circuits with a maximum of 20 primary inputs and 200 gates. Benchmarks b12, rd73 and t481 have been finally selected to perform the tests. The original version of each benchmark was obtained by synthesizing the circuit with Synopsys using the Nangate15nm synthesis library [79]. Table 6.6 shows the size of each benchmark as provided by the synthesis tool both in the number of cells and the area, as well as the number of PIs and POs.

Benchmark	#PIs	#POs	#cells	area
b12	15	9	58	12.93
rd73	7	3	20	5.90
t481	16	1	32	6.98

Table 6.6: Synthesis results for selected benchmarks

The generation of approximate logic circuits for the experiments was done in the following way. For the probabilistic approach based on dynamic testability measures, the same procedure described in subsection 5.5.1 has been applied, with the particularity of using the Nangate15nm synthesis library instead of the SAED90nm [48]. A set of arbitrary error targets was set for each circuit, covering a significant range of cases between the conventional TMR (full protection) and trivial approximation (no redundant logic). Each error target generated a pair of approximate circuits (over- and under-approximation), which were resynthesized in order to remove logic constants. Each pair of approximate circuits was combined with the original circuit to build a valid ATMR scheme.

With respect to the evolutionary approach, a large set of approximate circuits was generated for each benchmark by means of CGP. The parameters introduced in subsection 6.4.2 were set up to the values recommended in the literature [73], which are summarized in Table 6.7.

Parameter	b12	rd73	t481
n_i	15	7	16
n_o	9	3	1
n_c	6	5	5
n_r		1...10	
L	6	5	5
Γ		all 2-input gates	
λ		4	
m		5%	
N_g	2000000	2000000	1000000

Table 6.7: CGP parameters

For each configuration of the CGP grid (as n_r varies from 1 to 10), a total of 100 over-approximations and 100 under-approximations were generated for each benchmark circuit. Figures 6.21, 6.22 and 6.23 show, in the form of box plots, a statistical analysis of multiple CGP runs for the selected circuits evolved as the underapproximations and overapproximations. The box plots give the Hamming distances obtained for increased amount of resources (i.e., the number of rows in the CGP configuration matrix) available for the implementation. A clear tradeoff between the Hamming distance (quality) and the area can be observed. In addition, it can be observed that, in general, a bigger CGP configuration matrix allows generating approximate circuits closer to the target logic function. The case of t481 benchmark in Figure 6.23 has the particularity

of being a circuit with just one primary output with a high probability of onset. Under such conditions, the evolutionary approach tends to generate onset circuits that behave like logic constants due to the limited size of the configurable logic array.

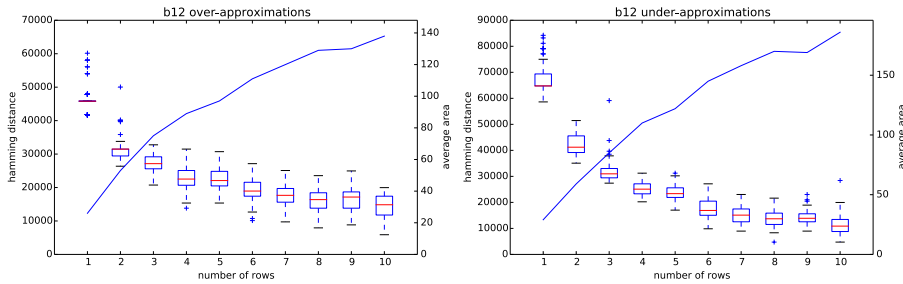


Figure 6.21: Statistical results for b12 approximations evolved by CGP

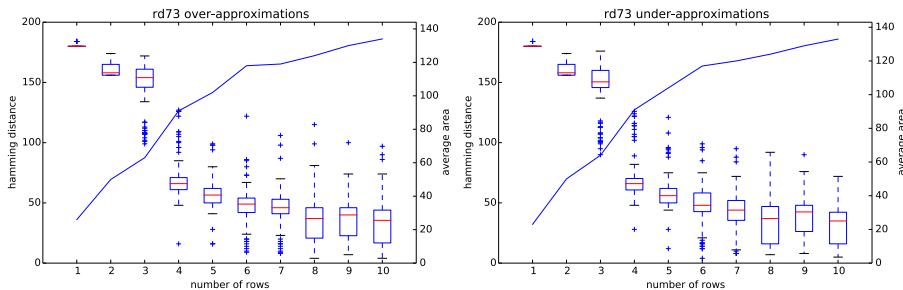


Figure 6.22: Statistical results for rd73 approximations evolved by CGP

In total, 1000 approximate circuits of each type were generated for each benchmark. According to the fundamentals of fault masking with approximate logic circuits, any over-approximation can be combined with any under-approximation to conform a valid ATMR scheme. Therefore, there are 10^6 possible solutions to test for each benchmark. Testing the whole set of solutions would take too much time, and therefore, it was studied if there was any representative data that allowed to select the best solutions in terms of error masking capabilities. For the first benchmark, b12, an exhaustive analysis was performed, i.e., all the 10^6 over- and under-approximation pairs were tested. The whole process is very time consuming, therefore, data collected for b12 were studied in order to properly select the most promising candidates for the rest of the benchmarks. The goal consists in building the ATMR scheme with the highest error masking rate, given fixed CGP configuration grid sizes for the generation of over- and under-approximate circuits. It is clear that the more functionally similar are the approximate circuits with respect to the original circuit, the more protection against faults is achieved. Therefore, approximate circuits with low Hamming distance compared with the original circuit are good candidates, in principle. To validate this hypothesis, the correlation between the sum of Hamming distances of both approximate circuits and the experimental error probability was computed. The results are shown in Table 6.8,

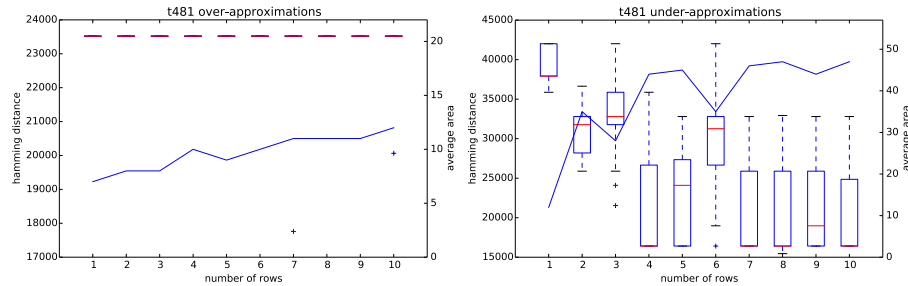


Figure 6.23: Statistical results for t481 approximations evolved by CGP

grouped according to the size of the configuration matrix for both under- and over-approximate circuits. The results show that there is a significant correlation between both metrics, with an average correlation index equal to 0.831. In conclusion, for the remaining benchmarks only the circuits with a Hamming distance below the average of each group were selected for experiments.

Over/Under	1	2	3	4	5	6	7	8	9	10
1	0.700	0.686	0.739	0.754	0.708	0.707	0.674	0.646	0.688	0.667
2	0.792	0.772	0.803	0.843	0.818	0.840	0.838	0.820	0.845	0.839
3	0.717	0.626	0.653	0.732	0.704	0.766	0.770	0.763	0.786	0.797
4	0.802	0.734	0.820	0.875	0.833	0.867	0.890	0.879	0.887	0.875
5	0.806	0.737	0.799	0.847	0.815	0.849	0.867	0.857	0.868	0.861
6	0.836	0.818	0.871	0.903	0.885	0.900	0.916	0.910	0.914	0.913
7	0.823	0.814	0.861	0.897	0.882	0.897	0.913	0.905	0.911	0.912
8	0.811	0.805	0.853	0.893	0.880	0.896	0.912	0.902	0.911	0.914
9	0.795	0.751	0.795	0.854	0.834	0.867	0.887	0.875	0.888	0.893
10	0.815	0.840	0.865	0.901	0.900	0.913	0.929	0.923	0.930	0.938

Table 6.8: Error masking rate versus Hamming distance correlation indexes

Once approximate circuits were generated, ATMR schemas were built for testing. Voters were placed at the output of circuits, and the list of stuck-at faults was generated for each circuit. This list included all faults on every input of each gate in the circuit, plus the faults on the outputs of the circuit before the voter. This allowed to introduce simple voters, as there is full control of fault injection points.

For each ATMR schema under test, a fault simulation with random input vectors was performed by means of the parallel simulator HOPE [45]. A total of 50,000 randomly generated input vectors were applied for each design under test, and all faults in the list previously generated were tested for each input vector. The total error probability was computed as the average number of faults detected per input vector, divided by the size of the fault list. For simplicity, all faults were considered equally likely.

6.4.4 Experimental results

Figure 6.24 graphically represents the trade-off between error masking rate and area overhead for several ATMR solutions found by using either the probabilistic or the evolutionary approach for circuit b12. For the latter technique, only the cases with the best error masking rate for each possible combination in the sizes of under-approximate and over-approximate circuits are represented. The same applies for Figures 6.25 and 6.26 for rd73 and t481 benchmarks, respectively.

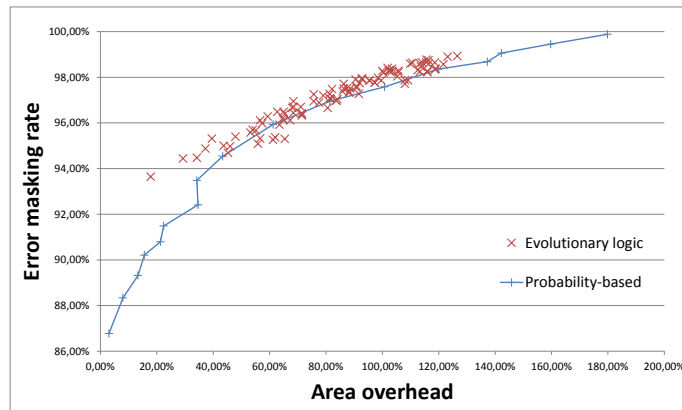


Figure 6.24: b12 simulation results

Analysing the results in Figure 6.24, it can be seen that the evolutionary approach achieves in general slightly better results than the probabilistic one for b12 benchmark. This is reasonable, because evolutionary approach can explore a much larger range of solutions, although at a much larger computational cost. However, the probabilistic approach can still obtain good solutions, close to the evolutionary approach. On average, the error masking rate achieved by the probabilistic approach is less than 2% lower than the best cases generated by the evolutionary algorithm for the same area overhead.

Results for rd73 benchmark are shown on Figure 6.25. This is an example of a circuit with a high degree of binateness, which means that small expansions on either the on-set or the off-set with respect to the original logic function have a high cost in terms of resources. In the case of the probabilistic approach, this is due to the necessity of performing the unate expansion, but the same tendency can be observed in the evolutionary side. This leads to suboptimal solutions with overheads greater than 200% in both approaches, which are uninteresting. As long as the combined area overhead of both approximate circuits is greater or equal than 200%, a pure TMR is preferred instead. On the other hand, under the 200% overhead limit the same tendency as with b12 circuit is observed. The evolutionary approach produces slightly better solutions than the probabilistic approach, but with much more computational effort.

With respect to t481 benchmark (see Figure 6.26), it can be observed that the probabilistic approach presents more scalability than the evolutionary one. This is due to

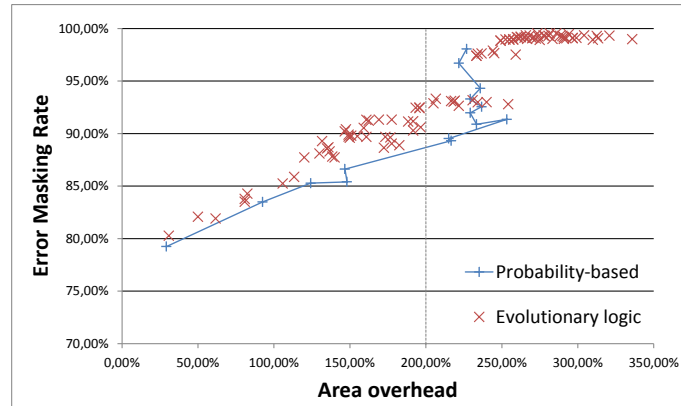


Figure 6.25: rd73 simulation results

the fact that, as already mentioned in subsection 6.4.3, t481 is a circuit with just one output with high onset probability. Under such conditions, the evolutionary approach tends to generate onset circuits that behave like logic constants due to the limited size of the configurable logic array, thus limiting both area overhead and error masking rate. On the other hand, the probabilistic approach is based on gradually degrading the logic function of the circuit, which allows to reach more robust solutions in the region close to the conventional TMR.

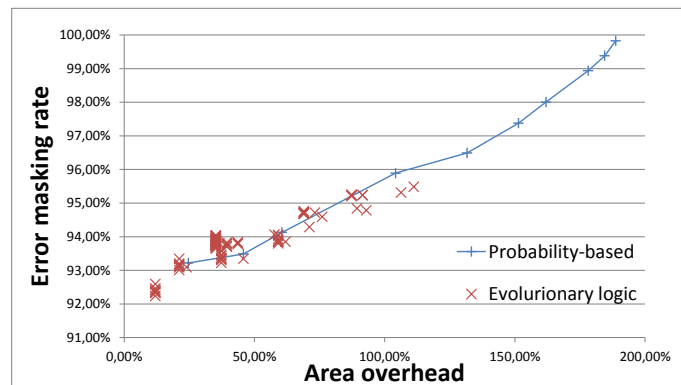


Figure 6.26: t481 simulation results

6.5 Conclusions

In this chapter, several applications of the Approximate TMR have been presented, based on the approximate circuit generation techniques developed through this thesis. First, an extension to ATMR implemented in FPGA has been addressed. Then, a collaboration project intended to design and test an ATMR version of a commercial microprocessor has been presented. Finally, a comparison with an alternative method for approximate circuit generation based on evolutionary algorithms has been performed.

ATMR implementation in FPGAs can be done at a minimum cost by using the FPGA resources that not used by the target design, with the ability of adapting to a specific area overhead. However, in an FPGA the combinational logic is implemented in the form of LUTs, in contrast to the logic gate structure in which the approximate circuit generation approaches proposed in this thesis are based, which are better suited for ASICs. Therefore, in order to adapt the techniques presented in chapters 3, 4 and 5 to the new LUT structure, some restrictions are imposed to the faults which can be approximated. In that way, only logic transformations that effectively reduce the number or the size of LUTs of approximate circuits are allowed. This approach is combined with a novel way of fault approximation for binate faults, where the both components of the binate fault are simultaneously approximated, each one in its correspondent approximate circuit. This approach allows reducing the costs of the unate expansion for binate circuits.

Thanks to the implementation on FPGAs, for the first time error injection experiments by means of radiation have been performed. The results show that ATMR schemes can be more robust than pure TMR, specially when faults are allowed to accumulate in the FPGA, due to a lower area overhead.

The design of ATMR versions of a commercial microprocessor has been done within the scope of a collaboration between academia and industry, which proves that the proposed techniques are interesting for real applications. In particular, the ARM Cortex M0 microprocessor has been selected as the target design. A preliminary study has been performed in order to obtain representative testability measures for the microprocessor, consisting in the selection and simulation of several software benchmarks, which had to be adapted to the limited memory availability inside the final design. This final design containing a few ATMR design as well as a golden reference has been manufactured in an Integrated Circuit (IC), with the goal of testing its performance in a radiation experiment, still pending.

Finally, approximations generated with the method proposed in this thesis, based on testability measures, have been compared with those generated by means of evolutionary algorithms, CGP in particular. This kind of approaches are able to generate approximations by means of random modifications of the target circuit. A user defined fitness function selects the most promising modified circuits in an iterative process. This process is time consuming, but it has the advantage of avoiding the local minima problem associated with greedy algorithms. The results show that, although CGP achieves better solutions in terms of area overhead and/or protection against faults, the probabilistic approach developed in this thesis is good enough considering the reduced computational time required. In addition, the probabilistic approach presents no restrictions on the size of the circuits that can be processed.

Chapter 7

Conclusions and future work

7.1 Conclusions

The technological advances in the digital electronics field allows manufacturing electronic circuits with smaller transistor sizes. This fact has several benefits, namely a lower power consumption, a higher integration density and the possibility of operating at higher clock frequencies. But as a consequence of these advances, in turn, electronic circuits are becoming more and more susceptible to transient faults, because the energy required to perturb the internal logic has decreased as well. The main cause of such faults is the existence of highly energetic particles which can strike the circuit. This phenomenon is more intense in extreme radiation environments such as the outer space, but it is present everywhere, including the Earth surface.

Among the different types of radiation induced errors, SEUs have traditionally been more studied because they can directly modify the state of any given circuit. However, in modern technologies, SETs are also becoming more relevant. Actually, the probability of SET occurrence is higher than for SEUs because they can be originated in any combinational node, although due to the logical, electrical and temporal masking effects there is a fraction of SETs which are filtered out. However, with the reduction of transistor sizes and the increment of clock frequencies, both electrical and temporal masking effects have less and less relevance. In conclusion, currently there is the need of implementing fault tolerant solutions able to deal with both SEUs and SETs in an efficient manner.

The most common solutions to protect against transient faults are Duplication With Comparison and Triple Modular Redundancy. These approaches implement several instances of the target circuit on parallel, along with additional logic which is in charge of respectively detecting or correcting errors produced in any of the circuit instances. The capability of DWC and TMR techniques to mitigate both transient and permanent errors makes them good techniques to tackle the variety of potential failure mechanisms that must be considered for advanced technologies. However, these techniques suffer from high overhead in terms of area and power consumption (more than 100% and 200% respectively). To alleviate these overheads, alternative techniques have been

proposed based on partial fault detection or masking.

A new paradigm for the design of efficient circuits is emerging, which is known as approximate computing. The idea consist in slightly deviating from the intended functionality in order to achieve some benefits in either the circuit size, the power consumption or the critical path duration. There are several approaches based on this idea, which in any case is intended for those applications which can tolerate a certain degree of misbehaviour.

This thesis is focused on applying the approximate computing paradigm to the fault tolerance field. The goal consists in generating simplified versions of a given target circuit, not necessarily equivalent, in order to build a TMR-like scheme (also known as ATMR) for partial error correction with reduced overheads. These alternative versions of the target circuit are denoted as approximate circuits. A main advantage of this approach is the great flexibility it provides. For a given target circuit, there are multiple approximate circuits which can be chosen, potentially providing a continuous trade-off between fault masking capabilities and overheads which is not possible with conventional TMR. Nevertheless, finding approximate circuits with the optimal trade-off for a given application is a challenging problem.

Because approximate circuits are not required to be functionally identical to the target circuit, there may be some discrepancies between the outputs of the different circuits involved in error correction. Then, in order to ensure a correct result in the absence of faults, one of the approximations is built by just expanding the onset of the target circuit, and the other one just reducing it. This kind of approximate circuits are respectively denoted as over-approximation and under-approximation. Based on this premise, this thesis presents several original methods for generating approximations, departing from a gate level description of the target circuit.

All the approaches for generation of approximate logic circuits proposed in this thesis are based on the same mechanism, denoted as fault approximation, which is the first fundamental contribution of this thesis. Basically, approximations are generated by assigning constant logic values to some circuit lines, which is equivalent to forcing some stuck-at faults. If the logic assignment is performed on a unate line, then either an under- or an over-approximation is generated depending on the parity of the line and the forced logic value. In addition, multiple fault approximations of the same type can be combined in a circuit for further savings. On the contrary, approximating a fault on a binate line does not guarantee a correct behaviour of the masking scheme. Therefore, binate faults cannot be directly approximated, being necessary at first making the whole circuit unate. This is achieved by means of the unate expansion procedure, which consists in performing a duplication of all the binate nodes of the circuit, and grouping their connections to one replica or the other according to the parity of each individual connection. This process is optional, but if it is not applied, then there will be some areas of the circuit which could not be approximated. The application of the unate expansion has the drawback of temporary increasing circuit area, although it is expected to be compensated when fault approximations are performed. As a way of reducing the negative effects associated to the unate expansion, an alternative adaptive fault approximation approach has been proposed. In this approach, faults are approximated by inserting additional logic gates with don't care side inputs, in such a way that the approximations can be taken or discarded depending on the logic values assigned to

the don't-care inputs. An external synthesis tool would be in charge of deciding these values for an optimal solution in terms of area.

The second major contribution of this thesis are the methods to select which faults are approximated. Several fault selection heuristics have been developed, although all of them are based on the same principle: the selection of approximations has to be driven by testability measures, being the least testable faults the best candidates for approximation. The rationale of this statement lies in that approximating the least testable faults provides resource savings while minimizing the impact on the error correction capabilities of the final ATMR scheme. Moreover, the fault approximation approach facilitates the estimations of testability. This is a distinctive key feature with respect to most of the state-of-the-art approaches, which are typically synthesis-oriented and, therefore, they pay much more attention to resource savings than to fault tolerance. A drawback of the synthesis-oriented methods is that they cannot estimate the impact of approximations on the fault-tolerance and as a result, they offer limited scalability.

The first developed version of the fault selection heuristic is based in static testability measures. In other words, fault testabilities are computed on the original circuit, before any logic transformations are applied, and they remain unchanged during the whole approximation process. Here it is proposed to perform a fault simulation in order to estimate the testability of every fault. Then, a testability threshold is specified by the final user, and every fault whose testability lies below that threshold becomes approximated. This process generates a pair of complementary (over and under) approximate circuits. A variety of approximate circuits with different trade-offs in terms of robustness versus overheads can be obtained simply by modifying the testability threshold. This approach has been validated through experiments, showing a wide scalability and reasonably good results, all with a low computational cost.

In addition, an extension for sequential circuits has been devised. Such an extension has been performed in a very straightforward manner, by considering the sequential elements as inputs and outputs of the combinational part of the circuit. With respect to testability measures, they are obtained in the same way as for combinational circuits, by means of an initial stuck-at fault simulation. However, sequential circuit testing is a hard task, which is usually alleviated by transforming it into a combinational problem. Both approaches have pros and cons. While combinational testing is much easier, the testability measures obtained in this way may be inaccurate, which may lead to suboptimal approximations.

The approximation generation method based in static testability measures is simple and flexible, but it has some limitations. First, obtaining a specific area overhead or fault masking rate with this method can only be achieved by trial and error because the testability threshold, which is the parameter used to tune the trade-off of the generated approximate circuits, is not representative of any of these metrics. But even more relevant is the fact that, whenever a fault is approximated, the testabilities of the remaining faults in the circuit may change. Therefore, the static testability measures are less representative as more faults are approximated, which may result in suboptimal logic transformations. In order to solve these limitations, an alternative fault selection criteria based on dynamic testability measures has been developed.

The second fault selection heuristic developed in this thesis makes use of dynamic testability measures. In this case, the fault testabilities are iteratively computed and

updated after each new logic transformation. This way, the effect of every fault approximation can be accurately estimated and the most beneficial logic transformations can be selected at each step. The iterative computation of fault testabilities is performed by means of a fault probability analysis based in fault implications, intended to overcome the reconvergent fanout problem and solve most of the signal interdependencies. Therefore, this approach is only suitable for combinational circuits, due to the high difficulty of computing fault probabilities in sequential circuits. Once the probability of every fault in the circuit has been estimated, the best candidate according to the criteria of (first) minimum fault probability and (second) maximum area savings is selected and approximated. The fault probabilities of the remaining faults are then dynamically updated and the next best candidate is selected according to the updated probabilities in an iterative process. In addition, the total accumulated variation of fault probabilities, denoted as total error probability, serves as an estimation of the average error rate for the whole set of faults considered, which provides an ending condition to the iterative algorithm. By using this approach it is possible to generate a pair of complementary approximate logic circuits that meet a required error target when implemented in an ATMR scheme along with the original circuit. Here the error target is what provides the required flexibility, because setting different error targets modifies the characteristics of the generated circuits. The experimental results shows that the proposed approach is able to stick to a given error target with reasonably good precision and has wide scalability and flexibility.

The approximation generation approach based in dynamic testability measures overcomes the main limitations of the static version, with the cost of an increased complexity and computational effort. On the one hand, the total error probability target is more comprehensive and useful for the final user than the former testability threshold, because it is an estimator of the global error masking rate. On the other hand, the iterative computation of fault probabilities allows selecting the best candidate at each step, thus avoiding suboptimal logic transformations. However, the proposed approach makes use of a greedy heuristic for choosing the sequence of approximations, and therefore it is susceptible to fall into local minima.

In addition, a new type of logic transformation has been proposed, which consists in substituting functionally similar nodes within the circuit. This transformation receives the name of node substitution, and it can be combined with the classical fault approximation approach in order to widen the range of feasible solutions for approximate circuit generation. Node substitution candidates are found departing from the implications made in order to compute the probability of each fault. For each identified node substitution candidate, a representative probability of the logic transformation is computed. In that way, the iterative search can be conducted jointly for node substitutions and fault approximations. An alternative heuristic has been developed in order to favour node substitutions over fault approximations, consisting in selecting the logic transformation which maximizes the ratio probability with respect to estimated area savings. Experimental results shows marginal benefits with respect to the dynamic approach with just fault approximations, and no clear winner among the proposed criteria for the selection of logic transformations. However, this is a promising preliminary concept which needs to be further developed to reach its true potential.

Finally, several applications of the approximation approaches proposed in this the-

sis have been demonstrated. Namely, they are three: an extension to FPGA-based circuits, an implementation of a fault tolerant version of a real application circuit (specifically an ARM Cortex M0) with approximate logic circuits, and finally a comparison against logic optimization approaches based in evolutionary algorithms.

The extension of the approximation generation techniques to FPGAs has been performed considering the particular features of these devices. In an FPGA, the combinational part of logic circuits is implemented as a group of interconnected LUTs. A single LUT may be equivalent to several logic gates. Therefore, when departing from the logic gate structure, it is necessary to impose some restrictions over the faults which can be approximated to ensure that every logic transformation effectively saves resources (by either suppressing LUTs or merging adjacent LUTs). In addition, a new procedure for direct approximation of binate faults has been developed, consisting in splitting the binate fault into two complementary unidirectional faults, which are simultaneously approximated in the opposite approximate circuits. This approach is intended to reduce the penalties due to the unate expansion for binate circuits. These techniques have been validated by fault injection and radiation experiments, with surprising and interesting results. In particular, the experiments reveal that an ATMR scheme can be even more robust against SEUs in the configuration memory of an FPGA than a pure TMR scheme, because a smaller area overhead implies a lower probability of SEEs and it may compensate a partial degradation of the masking properties. This is specially relevant when faults are allowed to accumulate in the configuration memory of the FPGA.

The development of ATMR versions of the ARM Cortex M0, a real application microprocessor, has been accomplished in the context of a collaboration project between academia and industry, which illustrates the existing interest in this topic. Due to the complexity of the target circuit, the approximation generation approach based in static testability measures has been preferred. In order to obtain these measures, first a preliminary analysis of software benchmarks has been performed, selecting a representative group of programs which were able to run within a very reduced memory, due to the limitations of the final design. A few ATMR designs with different approximation levels and a golden reference were implemented. The final design has been manufactured in a 28 nanometers ASIC with the aim of evaluating its performance in a radiation experiment, still pending.

Finally, a comparison of the approximation generation method proposed in this thesis (on its dynamic version) against evolutionary algorithms (based in CGP) has been performed. Evolutionary algorithms use a random mutation mechanism to generate logic circuits departing from a seed. A fitness function evaluates the generated circuits, selecting on each generation the circuit with the better fitness in an iterative process. The fitness function can be designed to meet any desired criteria, in this case the generation of optimal approximate circuits. Evolutionary algorithms are time consuming, but they can presumably generate solutions that cannot be reached by greedy approaches, such as the dynamic approach proposed in this thesis. Therefore, it could be expected that the evolutionary algorithm could generate more optimal approximate circuits than the dynamic approach and the interest relies in knowing how far the results of the dynamic approach are from the CGP results. The experiments show that the approach proposed in this thesis is able to generate approximate circuits that are close

to the evolutionary approach, although a bit less optimal. In addition, the proposed approach has the advantages of a reduced computational effort, and no restrictions in the size of the circuits that it is able to process.

In summary, this thesis has proposed an original method for the generation of approximate logic circuits that has been studied in depth. It has given way to different approaches and has been demonstrated for different application scenarios with relevant results. As main advantages over the state of the art, the proposed method is very flexible and scalable. Furthermore, it allows to estimate the impact of approximation transformations on the error mitigation, which is essential in order to achieve a good balance with the area overhead.

7.2 Future research work

Several topics are proposed which may be part of future research works, with the aim of improving the error mitigation techniques presented in this thesis.

First of all, every approximation generation method proposed in this thesis departs from an initial implementation of the target circuit. As this fact might influence the final result, it would be interesting to study different optimizations of the initial circuit, with the aim of discovering if there is any particular approach which favours the generation of more optimal approximate circuits.

In relation with the fault approximation mechanism, it has been introduced the idea of performing an adaptive fault approximation alternatively to simply forcing constant logic values to some circuit lines. This mechanism would generate don't-care inputs in the approximated lines in such a way that, depending on the value that is later given to such input, the approximation would be taken or not. This decision would be taken by a synthesis software, although some synthesizers struggle when handling this kind of instrumentation, and therefore choosing an adequate synthesis software would be critical. This alternative approximation generation mechanism can be studied to be implemented, with the idea of minimizing the costs associated to the unate expansion in binate circuits.

With respect to the approximation generation method based on dynamic testability measures, there are several possible improvements. First, alternative fault selection heuristics can be implemented in order to favour other kinds of solutions which might be more beneficial for certain applications. In addition, recursive learning can be implemented at different levels through the algorithm. On the one hand, applying recursive learning during the fault implication process may allow resolving the signal interdependencies that are not discovered by just direct implication, which results in more precise fault probability estimations. On the other hand, recursive learning can be additionally applied to the fault selection heuristic itself, which may help in avoiding the local minima problem, increasing the chances of obtaining an optimal solution. In any case, recursive learning imposes a great computational effort. Finally, there is a set of possible improvements with respect to the computation of the total error probability. In the first place, the computation of the EP should be extended to the three circuit instances for a better predictability of the approximation generation algorithm. The mathematical reasoning to do so has been presented, although it has been proved

that EP cannot be incrementally computed in that case, thus imposing an additional complexity. Besides, additional terms can be introduced in the computation of the EP in order to obtain more realistic estimations. For example, each fault probability could be weighted by the probability that each particular fault was originated by external factors, or the estimation of EP could be performed taking into account fault models alternative to the stuck-at, such as bit flips (multiplying each fault probability by the corresponding signal probability) or SETs (applying the corresponding electrical and temporal de-rating factors).

Regarding the node substitution approach, in addition to the topics already discussed for the dynamic approach, its performance can be improved by considering dynamic fault lists. That is, that the node substitutions performed introduce new connections to the approximate circuits, which have associated new faults. These new faults could be approximated in later steps, or be replaced by another node substitutions. Currently, these transformations can never take place because these new faults are not dynamically introduced in the fault list.

With respect to the approximation generation for circuits implemented in an FPGA, more opportunities for the optimization of approximate circuits can be generated by implementing a dynamic LUT superstructure, where LUTs can be merged and re-defined as long as the approximation process progresses. In such a way, the set of possible logic transformations can be enlarged, because the dynamic configuration of the LUT frontiers may allow approximating some faults which initially were located inside a LUT, and therefore forbidden. In addition, concerning the direct approximation of binate faults, a small improvement can be achieved by detecting the autocancellation cases, and subsequently performing the approximation of each complementary unate fault taking into account this fact.

Finally, regarding the ATMR version of the ARM Cortex M0, radiation results were still pending at the moment of writing this document. Depending on the final results, additional studies on ATMR applied at different scopes, from system level to individual submodules, could be performed.

Bibliography

- [1] ARM limited. *ARM Cortex-M0 Technical Reference Manual*, 2010. <http://infocenter.arm.com>.
- [2] ARM limited. *ARM Compiler v5.06 for Vision armlink User Guide*. www.keil.com/support/man/docs/armlink/.
- [3] Robert C Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *Device and Materials Reliability, IEEE Transactions on*, 5(3):305–316, 2005.
- [4] Gustavo Neuberger, Gilson Wirth, and SpringerLink (Online service). *Protecting Chips Against Hold Time Violations Due to Variability*. Springer Netherlands, Dordrecht, 2014.
- [5] R. Koga, S. H. Penzin, K. B. Crawford, and W. R. Crain. Single event functional interrupt (sefi) sensitivity in microcircuits. In *RADECS 97. Fourth European Conference on Radiation and its Effects on Components and Systems*, pages 311–318, Sep 1997.
- [6] A. Haran, J. Barak, D. David, E. Keren, N. Refaeli, and S. Rapaport. Single event hard errors in sram under heavy ion irradiation. *IEEE Transactions on Nuclear Science*, 61(5):2702–2710, Oct 2014.
- [7] W. A. Kolasinski, J. B. Blake, J. K. Anthony, W. E. Price, and E. C. Smith. Simulation of cosmic-ray induced soft errors and latchup in integrated-circuit computer memories. *IEEE Transactions on Nuclear Science*, 26(6):5087–5091, Dec 1979.
- [8] R. C. Martin, N. M. Ghoniem, Y. Song, and J. S. Cable. The size effect of ion charge tracks on single event multiple-bit upset. *IEEE Transactions on Nuclear Science*, 34(6):1305–1309, Dec 1987.
- [9] M. Nicolaidis T.Calin and R. Velazco. Upset hardened memory design for sub-micron cmos technology. *Nuclear Science, IEEE Transactions on*, 43(6):2874–2878, Dec 1996.
- [10] Veronique Ferlet-Cavrois, Lloyd W Massengill, and Pascale Gouker. Single event transients in digital cmosa review. *Nuclear Science, IEEE Transactions on*, 60(3):1767–1790, 2013.

- [11] Q. Zhou and K. Mohanram. Gate sizing to radiation harden combinational logic. *Computer-Aided Design, IEEE Transactions on*, 25:155–166, Jan 2006.
- [12] M. Nicolaidis. Time redundancy based soft-error tolerance to rescue nanometer technologies. In *Proc. VLSI Test Symposium*, pages 86–94, 1999.
- [13] Barry W Johnson. *Design & analysis of fault tolerant digital systems*. Addison-Wesley Longman Publishing Co., Inc., 1988.
- [14] Kartik Mohanram, Nur Touba, et al. Partial error masking to reduce soft error failure rate in logic circuits. In *Defect and Fault Tolerance in VLSI Systems, 2003. Proceedings. 18th IEEE International Symposium on*, pages 433–440. IEEE, 2003.
- [15] Aiman H El-Maleh and Feras Chikh Oughali. A generalized modular redundancy scheme for enhancing fault tolerance of combinational circuits. *Microelectronics Reliability*, 54(1):316–326, 2014.
- [16] Kundan Nepal, Nuno Alves, Jennifer Dworak, and R Iris Bahar. Using implications for online error detection. In *Test Conference, 2008. ITC 2008. IEEE International*, pages 1–10. IEEE, 2008.
- [17] A.H. El-Maleh and K.A.K. Daud. Simulation-based method for synthesizing soft error tolerant combinational circuits. *Reliability, IEEE Transactions on*, PP(99):1–14, 2015.
- [18] Smita Krishnaswamy, Stephen M Plaza, Igor L Markov, and John P Hayes. Enhancing design robustness with reliability-aware resynthesis and logic simulation. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 149–154. IEEE, 2007.
- [19] Sobeeh Almkhaizim and Yiorgos Makris. Soft error mitigation through selective addition of functionally redundant wires. *Reliability, IEEE Transactions on*, 57(1):23–31, 2008.
- [20] M.R. Choudhury and K. Mohanram. Low cost concurrent error masking using approximate logic circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(8):1163–1176, Aug 2013.
- [21] Brian D Sierawski, Bharat L Bhuvva, and Lloyd W Massengill. Reducing soft error rate in logic circuits through approximate logic functions. *Nuclear Science, IEEE Transactions on*, 53(6):3417–3421, 2006.
- [22] Mihir R Choudhury and Kartik Mohanram. Approximate logic circuits for low overhead, non-intrusive concurrent error detection. In *Design, Automation and Test in Europe, 2008. DATE'08*, pages 903–908. IEEE, 2008.
- [23] I. A. C. Gomes, M. Martins, A. Reis, and F. L. Kastensmidt. Using only redundant modules with approximate logic to reduce drastically area overhead in tmr. In *2015 16th Latin-American Test Symposium (LATS)*, pages 1–6, March 2015.

- [24] Hao Xie, Li Chen, Rui Liu, A. Evans, D. Alexandrescu, Shi-Jie Wen, and R. Wong. New approaches for synthesis of redundant combinatorial logic for selective fault tolerance. In *On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International*, pages 62–68, July 2014.
- [25] I.A.C. Gomes, M. Martins, F. Lima Kastensmidt, A. Reis, R. Ribas, and S.P. Novales. Methodology for achieving best trade-off of area and fault masking coverage in atm. In *Test Workshop - LATW, 2014 15th Latin American*, pages 1–6, March 2014.
- [26] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy. Impact. Impact: Imprecise adders for low-power approximate computing. In *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, pages 409–414, Aug 2011.
- [27] A. Kahng and S. Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Design Automation Conf.*, pages 820–825, June 2012.
- [28] P. Kulkarni, P. Gupta, and M. Ercegovic. Trading accuracy for power with an underdesigned multiplier architecture. In *VLSI Design, 24th International Conference on*, pages 346–351, Jan 2011.
- [29] D. Shin and S. Gupta. Approximate logic synthesis for error tolerant applications. In *Design, Automation and Test in Europe, 2010. DATE'10*, pages 957–960. IEEE, March 2010.
- [30] D. Shin and S. Gupta. A new circuit simplification method for error tolerant applications. In *Design, Automation and Test in Europe, 2011. DATE'11*, pages 1–6. IEEE, 2011.
- [31] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. Salsa: Systematic logic synthesis of approximate circuits. In *Design Automation Conf.*, pages 796–801, June 2012.
- [32] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan. Aslan: Synthesis of approximate sequential circuits. In *Design, Automation and Test in Europe, 2014. DATE'14*, pages 1–6. IEEE, March 2014.
- [33] A. Bernasconi and V. Ciriani. 2-spp approximate synthesis for error tolerant applications. In *EUROMICRO Symposium on Digital System Design*, pages 411–418, Aug 2014.
- [34] Arun Chandrasekharan, Mathias Soeken, Daniel Groe, and Rolf Drechsler. Approximation-aware rewriting of aigs for error tolerant applications. In *Computer-Aided Design, 2016. ICCAD 2016. IEEE/ACM International Conference on*. IEEE, 2016.
- [35] Rolf Drechsler. *Evolutionary Algorithms for VLSI CAD*. Kluwer Academic Publishers, Boston, 1998.

- [36] Erik Larsson. *Introduction to Advanced System-on-Chip Test Design and Optimization*. Springer, 2005.
- [37] Martin A. Trefzer and Andy M. Tyrrell. *Evolvable Hardware: From Practice to Application*. Springer, 2015.
- [38] Zdenek Vasicek and Lukas Sekanina. Evolutionary approach to approximate digital circuits design. *IEEE Transactions on Evolutionary Computation*, 19(3), 2015.
- [39] Radek Hrbacek. Parallel multi-objective evolutionary design of approximate circuits. In *GECCO '15 Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 687–694. Association for Computing Machinery, 2015.
- [40] Va. V. Saposhnikov, A. Morosov, Vl. V. Saposhnikov, and Michael Gössel. A new design method for self-checking unidirectional combinational circuits. In *On-line testing for VLSI*, pages 41–53. Springer, 1998.
- [41] Z. Kohavi. *Switching and Finite Automata Theory*. Computer Science Series. McGraw-Hill, 1978.
- [42] L. Entrena, C. Lopez, E. Olias, E. San Millan, and J.A. Espejo. Logic optimization of unidirectional circuits with structural methods. In *On-Line Testing Workshop, 2001. Proceedings. Seventh International*, pages 43–47, 2001.
- [43] A. Sanchez-Clemente, L. Entrena, M. Garcia-Valderas, and C. Lopez-Ongil. Logic masking for set mitigation using approximate logic circuits. In *On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International*, pages 176–181, June 2012.
- [44] Hyungwon Kim and J.P. Hayes. Realization-independent atpg for designs with unimplemented blocks. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 20(2):290–306, Feb 2001.
- [45] Hyung Ki Lee and Dong Sam Ha. Hope: an efficient parallel fault simulator for synchronous sequential circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 15(9):1048–1058, Sep 1996.
- [46] L. Entrena, M. Garcia-Valderas, R. Fernandez-Cardenal, A. Lindoso, M. Portela, and C. Lopez-Ongil. Soft error sensitivity evaluation of microprocessors by multilevel emulation-based fault injection. *Computers, IEEE Transactions on*, 61(3):313–322, Jan 2012.
- [47] Berkeley Logic Synthesis and Verification Group. Abc: A system for sequential synthesis and verification. Release 2013-01-08. <http://www.eecs.berkeley.edu/~alanmi/abc/>.
- [48] Synopsys Armenia Educational Department. Saed 90nm generic library. <http://www.synopsys.com/Community/UniversityProgram>.

- [49] Kenneth P Parker and Edward J McCluskey. Probabilistic treatment of general combinational networks. *Computers, IEEE Transactions on*, 100(6):668–670, 1975.
- [50] F. Brglez. On testability of combinational networks. In *IEEE International Symposium on Circuits and Systems*, 1984.
- [51] S. C. Seth, L. Pan, and V. D. Agrawal. PREDICT-probabilistic estimation of digital circuit testability. In *Proceeding of International Symposium on Fault-Tolerant Computing*, pages 220–225, June 1985.
- [52] L. Goldstein. Controllability/observability analysis of digital circuits. *IEEE Transactions on Circuits and Systems*, 26(9):685–693, Sep 1979.
- [53] Jacob Savir, Gary S Ditlow, and Paul H Bardell. Random pattern testability. *Computers, IEEE Transactions on*, 100(1):79–90, 1984.
- [54] Savir. Improved cutting algorithm. *IBM Journal of Research and Development*, 34(2.3):381–388, March 1990.
- [55] Shih-Chieh Chang, Wen-Ben Jone, and Shi-Sen Chang. Tair: Testability analysis by implication reasoning. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 19(1):152–160, 2000.
- [56] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital systems testing and testable design*. Wiley-IEEE Press, 1994.
- [57] Luis Entrena, Kwang-Ting Cheng, et al. Combinational and sequential logic optimization by redundancy addition and removal. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 14(7):909–916, 1995.
- [58] W. Kunz and D. K. Pradhan. Recursive learning: a new implication technique for efficient solutions to cad problems-test, verification and optimization. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 13(9):1143–1158, Aug 1994.
- [59] Fernanda Lima Kastensmidt, Luigi Carro, and Ricardo Reis. *Fault-tolerance techniques for SRAM-based FPGAs*. Springer International Publishing, 2006.
- [60] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin. Improving fpga design robustness with partial tmr. In *2006 IEEE International Reliability Physics Symposium Proceedings*, pages 226–232, March 2006.
- [61] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin. Seu-induced persistent error propagation in fpgas. *IEEE Transactions on Nuclear Science*, 52(6):2438–2445, Dec 2005.
- [62] H. Quinn et al. Using benchmarks for radiation testing of microprocessors and fpgas. *Nuclear Science, IEEE Transactions on*, 62(6):2547–2554, Dec 2015.

- [63] Xilinx corporation. *Soft Error Mitigation Controller v4.1. LogiCORE IP Product Guide*, Sept 2015.
- [64] Jorge Tonfat, Lucas Tambara, André Santos, and Fernanda Kastensmidt. *Method to Analyze the Susceptibility of HLS Designs in SRAM-Based FPGAs Under Soft Errors*, pages 132–143. Springer International Publishing, Cham, 2016.
- [65] Brian Pratt, Michael Caffrey, James F. Carroll, Paul Graham, Keith Morgan, and Michael Wirthlin. Fine-grain seu mitigation for fpgas using partial tmr. *Nuclear Science, IEEE Transactions on*, 55(4):2274–2280, Sept. 2008.
- [66] N. A. Harward, M. R. Gardiner, L. W. Hsiao, and M. J. Wirthlin. Estimating soft processor soft error sensitivity through fault injection. In *Proc. IEEE 23rd Annu. Int. Symp. Field-Programmable Custom Computing Machines*, pages 143–150, May 2015.
- [67] Jimmy Tarrillo, Jorge Tonfat, Lucas Tambara, Fernanda Lima Kastensmidt, and Ricardo Reis. Multiple fault injection platform for sram-based fpga based on ground-level radiation experiments. In *Proc. 16th Latin-American Test Symposium (LATS)*, pages 1–6, March 2015.
- [68] ARM limited. *ARM Cortex-M0 DesignStart r0p0-00rel0 Release Note*, Aug 2010.
- [69] Cortex-m0 processor designstart. <https://www.arm.com/products/designstart/index.php>.
- [70] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the 4th Annual IEEE International Workshop on Workload Characterization (WWC-4)*, pages 3–14, Dec 2001. www.eecs.umich.edu/mibench/.
- [71] M. Garvie and A. Thompson. Scrubbing away transients and jiggling around the permanent: long survival of fpga systems through evolutionary self-repair. In *Proc of the 10th IEEE International On-Line Testing Symposium IOLTS 2004*, pages 155–160, 2004.
- [72] JulianF. Miller and Peter Thomson. Cartesian genetic programming. In *Genetic Programming*, volume 1802 of *Lecture Notes in Computer Science*, pages 121–132. Springer Berlin Heidelberg, 2000.
- [73] Julian F. Miller, editor. *Cartesian Genetic Programming*. Natural Computing Series. Springer Verlag, 2011.
- [74] Radek Hrbacek and Lukas Sekanina. Towards highly optimized cartesian genetic programming: From sequential via simd and thread to massive parallel implementation. In *GECCO '14 Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 1015–1022. Association for Computing Machinery, 2014.

- [75] Zdenek Vasicek and Lukas Sekanina. Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. *Genetic Programming and Evolvable Machines*, 12(3):305–327, 2011.
- [76] Julian F. Miller and Stephen L. Smith. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Trans. Evolutionary Computation*, 10(2):167–174, 2006.
- [77] B. W. Goldman and W. F. Punch. Analysis of cartesian genetic programming’s evolutionary mechanisms. *IEEE Transactions on Evolutionary Computation*, 19(3):359–373, 2015.
- [78] Zdenek Vasicek and Lukas Sekanina. Evolutionary design of approximate multipliers under different error metrics. In *17th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 135–140. IEEE Computer Society, 2014.
- [79] Nangate freepdk15 open cell library, 2014. http://www.nangate.com/?page_id=2328.

Appendix A

Circuit approximation examples

A.1 Introduction

For a better comprehension of the techniques developed within this thesis, several examples are explained here in detail. Each example corresponds to one of the different approaches proposed in this document. In addition, all examples depart from the same initial circuit for the sake of comparison between techniques.

The rest of the appendix is organised as follows. Section A.2 introduces the circuit in which all examples in this chapter are based. Section A.3 applies the approximation method based on static testability measures to our example circuit. Section A.4 makes use of dynamic testability measures. Example in section A.5 employs the node substitution technique. Finally, section A.6 performs an approximation of the example circuit intended for FPGA implementation.

A.2 Example circuit

C17 benchmark from LGSynth93 set has been the selected circuit to conduct the examples included here. Figure A.1 shows the gate level structure of the circuit. It can be seen that c17 is already a fully unate circuit, with the only exception of PI2 primary input, and therefore unate expansion is not required. This benchmark has been chosen for being a multiple output circuit, complex enough to appreciate differences between different approximation methods but at the same time small enough to be manageable. This circuit is used as the starting point for all examples within this section, with the aim of comparing between different approaches.

A.3 Approximation by static testability measures

The first example corresponds to approximation generation with static testability measures, which has already been explained in chapter 4. Although this method is quite

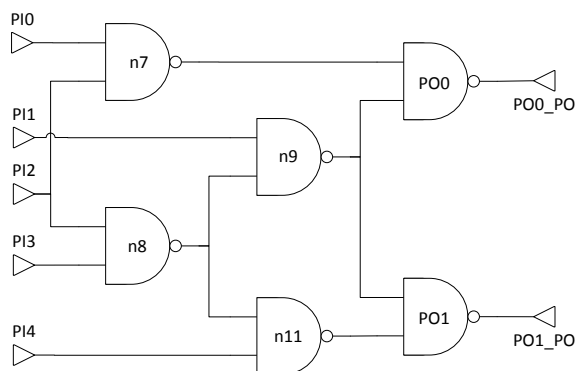


Figure A.1: c17 benchmark

straightforward, it is shown here with the purpose of comparing results with the rest of techniques addressed within this thesis.

In the first place, testability measures are obtained by means of stuck-at fault simulation on c17 benchmark with the aid of parallel simulator HOPE. Up to 1000 randomly generated input vectors have been applied on this example. Results of fault simulation are processed, thus obtaining the data contained in Table A.1. The table shows the collapsed list of faults for c17 benchmark along with the probability of each fault, in descending order. Fault probabilities have been computed as the number of occurrences of that fault, divided by the total number of applied input vectors -1000 in this case. This means, as an example, that fault PO0/1 has been detected 410 times during the simulation, while for fault n8→n9 there have been 135 input vectors which allowed its propagation to the outputs.

Fault	Prob.	Fault	Prob.
PO0 /0	0.590	PI2→n8 /1	0.200
n9 /0	0.572	n9→PO1 /1	0.198
n8 /0	0.566	n8 /1	0.198
PO1 /0	0.566	n7 /1	0.196
PO1 /1	0.434	PI3 /1	0.196
PO0 /1	0.410	PI0 /1	0.188
n9 /1	0.361	PI4 /1	0.180
n9→PO0 /1	0.318	n11 /1	0.172
PI1 /1	0.318	n8→n9 /1	0.135
PI2 /0	0.295	n8→n11 /1	0.134
PI2 /1	0.286	PI2→n7 /1	0.114

Table A.1: Results of fault testability analysis for c17

Now, faults within the circuit are classified according whether they produce an under- or an over-approximation. As explained in section 3.3, this depends only on the fault value and the parity of the line where that fault is applied, provided that circuit is unate. In the case of this example, faults are classified as follows:

- Under-approximation faults: $PI2 \rightarrow n8/1$, $PI3/1$, $n7/1$, $n9 \rightarrow PO0/1$, $n9 \rightarrow PO1/1$, $n11/1$, $PO0/0$ and $PO1/0$.
- Over-approximation faults: $PI0/1$, $PI1/1$, $PI2 \rightarrow n7/1$, $PI4/1$, $n8 \rightarrow n9/1$, $n8 \rightarrow n11/1$, $PO0/1$ and $PO0/1$.

Faults $PI2/0$, $PI2/1$, $n8/0$, $n8/1$, $n9/0$ and $n9/1$ correspond to stem lines, and therefore they are not taken into account. This classification is useful when generating approximations. Faults of the same type can be approximated together, i.e., in the same circuit. Faults of different types cannot, because the resulting circuit would be a bidirectional approximation.

After all the preparations have been completed, approximation generation can be performed. With this purpose, two replicas of the c17 benchmark are generated, an arbitrary testability threshold is set and every fault whose testability lies under the threshold becomes approximated. In this example a wide range of thresholds have been used with the aim of covering the whole range of reachable solutions between pure TMR and trivial approximation.

The less testable fault in the example is $PI2 \rightarrow n7$ stuck-at 1, whose probability is 0.114. Any testability threshold equal or lower than this value would produce a pure TMR scheme, as no faults become approximated and therefore approximate circuits are exact copies of c17 benchmark, as shown in Figure A.2.

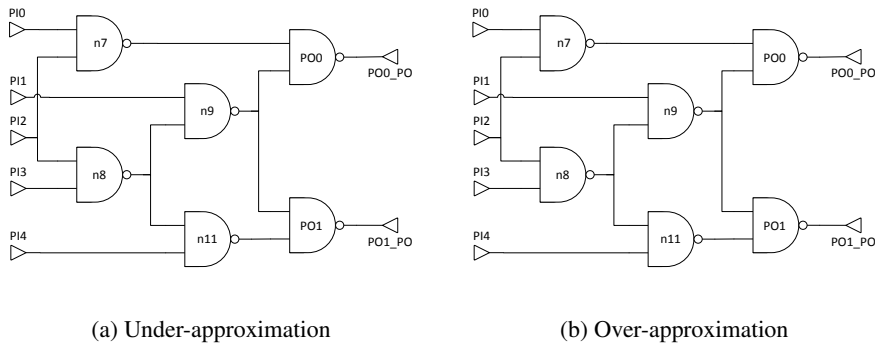


Figure A.2: Approximate circuits for a 0 threshold - Pure TMR

Let us assume a threshold of 0.12. In this case, only the fault $PI2 \rightarrow n7$ stuck-at 1 is approximated, because this is the only fault whose testability is lower than 0.12. The fault is approximated by substituting the corresponding connection - from input $PI2$ to node $n7$ - with a constant logic 1 - see Figure A.3a. This circuit can be simplified by replacing node $n7$ - a two input NAND gate with a constant input- with an inverter, thus obtaining the equivalent circuit of Figure A.3b, which is slightly smaller than original

circuit. According to the previous classification, fault $PI2 \rightarrow n7$ stuck-at 1 produces an over-approximation, and therefore the same applies for this circuit. On the other hand, an exact replica of c17 is used as the under-approximation. Figure A.4 shows the resulting approximate circuits in this case.

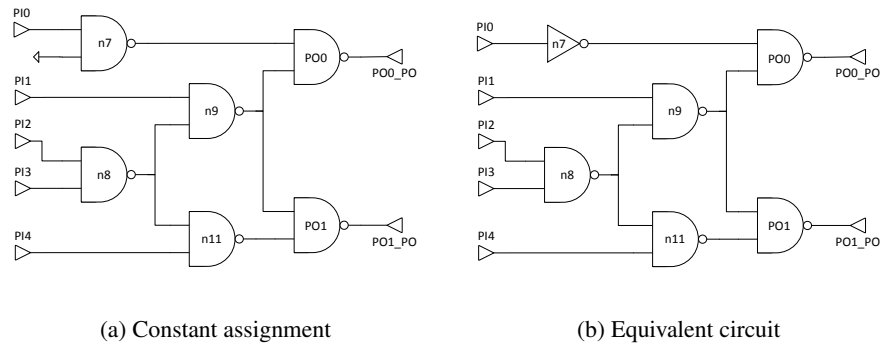


Figure A.3: Approximation of fault $PI2 \rightarrow n7$ stuck-at 1

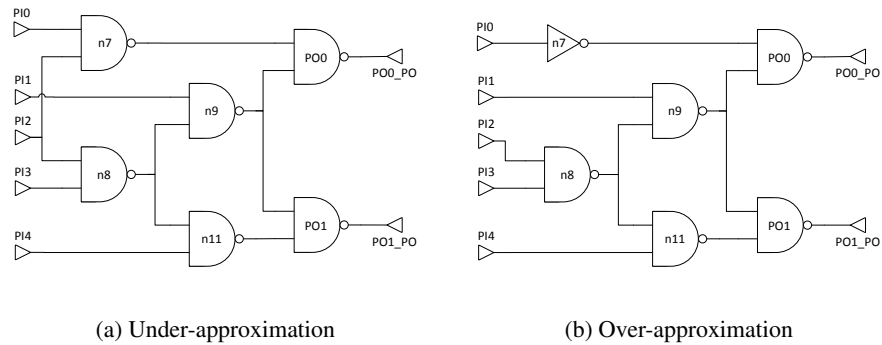


Figure A.4: Approximate circuits for a 0.12 threshold

The next selected threshold is 0.135. For this value, faults $PI2 \rightarrow n7/1$ and $n8 \rightarrow n11/1$ become approximated. It must be noted that faults whose testability equals the threshold value are not approximated, and therefore fault $n8 \rightarrow n9/1$ is not taken into account yet. Both faults are forced together as they are of the same type, and then the circuit is simplified, removing logic constants - see Figure A.5b. The resulting circuit serves as an over-approximation of the c17 benchmark, while the under-approximation is still a replica of c17. Figure A.5 illustrates the resulting approximate circuits for the selected testability threshold.

Consider now a threshold of 0.14. Three faults lie under this value: $PI2 \rightarrow n7/1$, $n8 \rightarrow n9/1$ and $n8 \rightarrow n11/1$. All this faults produce over-approximations, and therefore they are jointly forced in the same circuit, which is subsequently simplified as shown

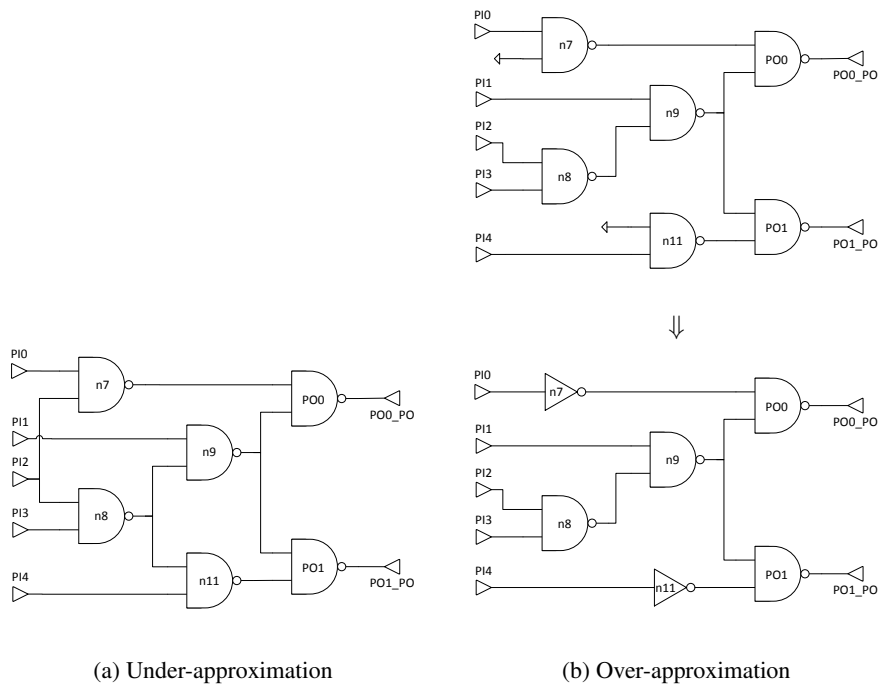


Figure A.5: Approximate circuits for a 0.135 threshold

in Figure A.6b. Dangling node n8 is suppressed, and the remaining nodes can be combined in just two OR gates. Here the potential benefits of approximate logic can be appreciated, as the over-approximation area has been reduced to about 1/3 of c17 area. Meanwhile, the under-approximation is again a copy of the original benchmark, as reflected in Figure A.6a.

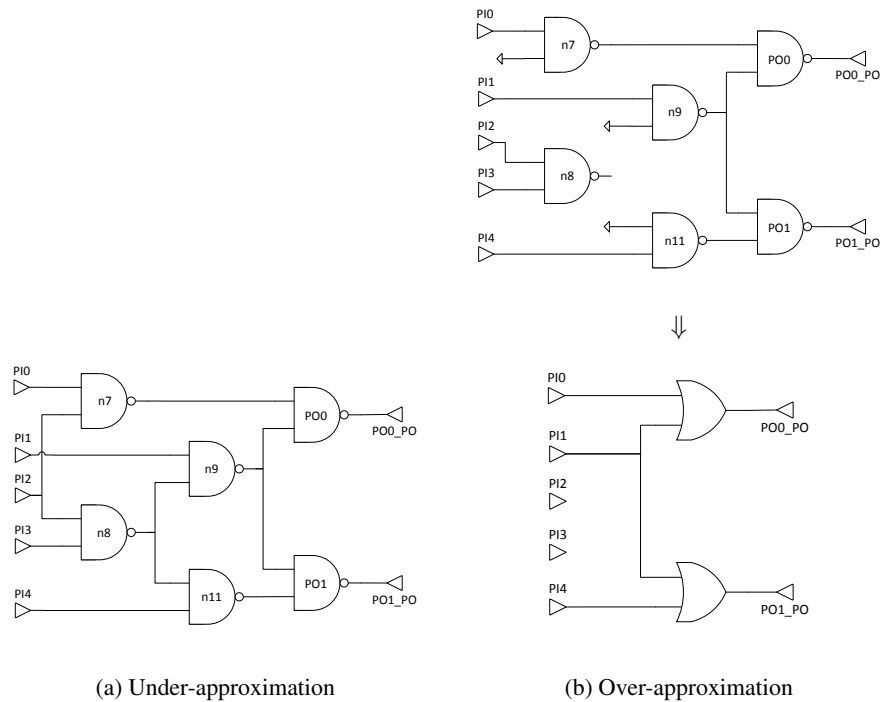


Figure A.6: Approximate circuits for a 0.14 threshold

The following value is 0.175. Below this threshold are over-approximation faults $PI2 \rightarrow n7/1$, $n8 \rightarrow n9/1$ and $n8 \rightarrow n11/1$ and the under-approximation fault $n11$ stuck-at 1. The three first faults are approximated in one of the c17 replicas, and the last fault in the other one, as shown in Figure A.7. Later, both circuits are simplified, obtaining the final solutions in order to build an error masking scheme. With respect to the under-approximation - Figure A.7a -, approximation of fault $n11/1$ allows to remove node $n11$, thus reducing the area overhead.

Let us establish a threshold of 0.185. In addition to the previous faults, $PI4$ stuck-at 1 become approximated. According with the fault classification, this is done in conjunction with $PI2 \rightarrow n7/1$, $n8 \rightarrow n9/1$ and $n8 \rightarrow n11/1$ faults, generating an over-approximation - see Figure A.8b. When simplifying this circuit, it can be appreciated that both inputs of node $n11$ are tied to logic constant 1, and therefore that gate can be suppressed by propagating the logic values forward. This eventually turns out that a constant logic 1 is assigned to primary output $PO1$. This contributes to reduce area overhead not only

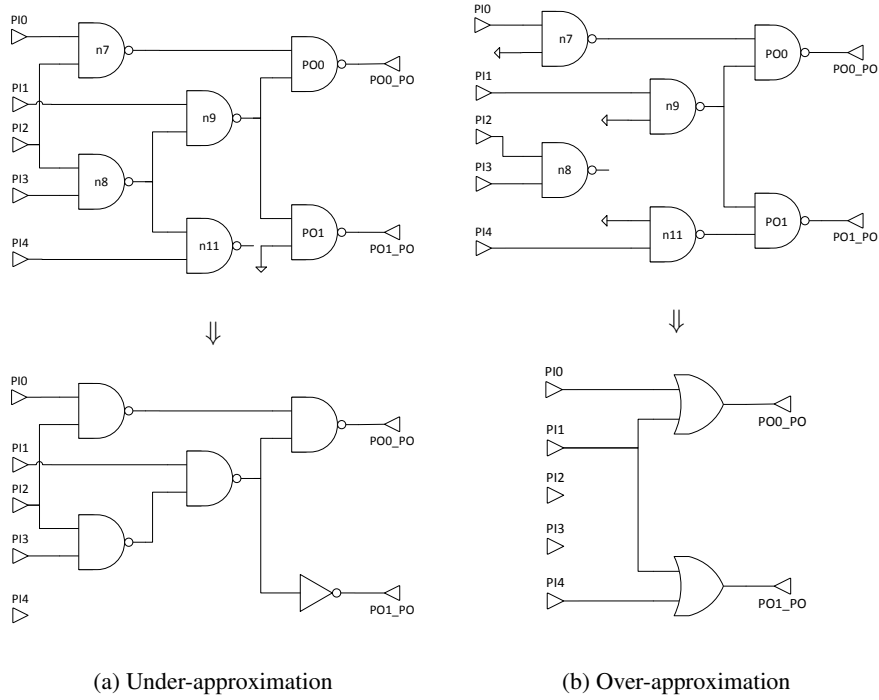


Figure A.7: Approximate circuits for a 0.175 threshold

by removing one part of the logic function, but it also allows to simplify the voting logic, because one of the three instances of primary output PO1 being voted is a logic constant. On the other hand, the under-approximation presents no changes with respect to the previous step. Resulting circuits for the considered testability threshold can be seen in Figure A.8.

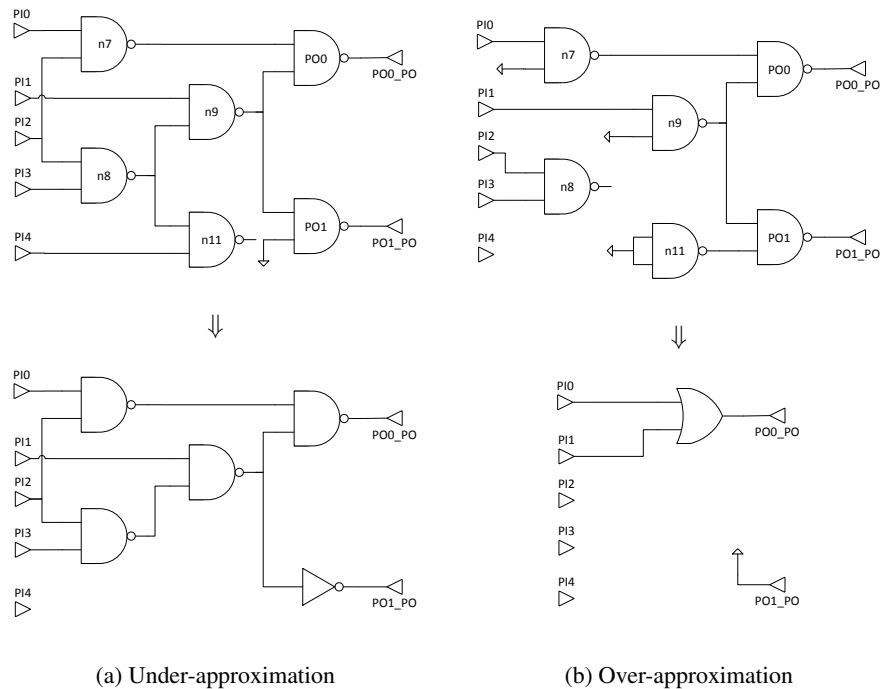


Figure A.8: Approximate circuits for a 0.185 threshold

The next value in the list is 0.19. From the part of under-approximation, only the fault n11 stuck-at 1 is approximated yet, while from the over-approximation side, faults PI0/1, PI2→n7/1, PI4/1, n8→n9/1 and n8→n11/1 are forced, as illustrated in Figure A.9. As usual, those circuits are simplified in order to remove logic constants. Due to simplifications and constant propagations, in this case all primary outputs in the over-approximate circuit receive a constant assignment - see Figure A.9b. In other words, over-approximation has been reduced to the trivial approximation, which introduces no area overhead. This holds for all the remaining cases. In addition, voting logic can be simplified for both primary outputs as one of the signals voted is constant, further reducing area overheads.

Following with a 0.197 threshold. Faults PI3/1 and n7/1 are included in the list of approximated faults, along with all the previous ones. These new faults are forced in the under-approximate version of the circuit, same as n11 stuck-at 1, and subsequently simplified. As result, both primary outputs implement the same logic function, as shown in Figure A.10a. Besides, the over-approximate circuit is the trivial approxi-

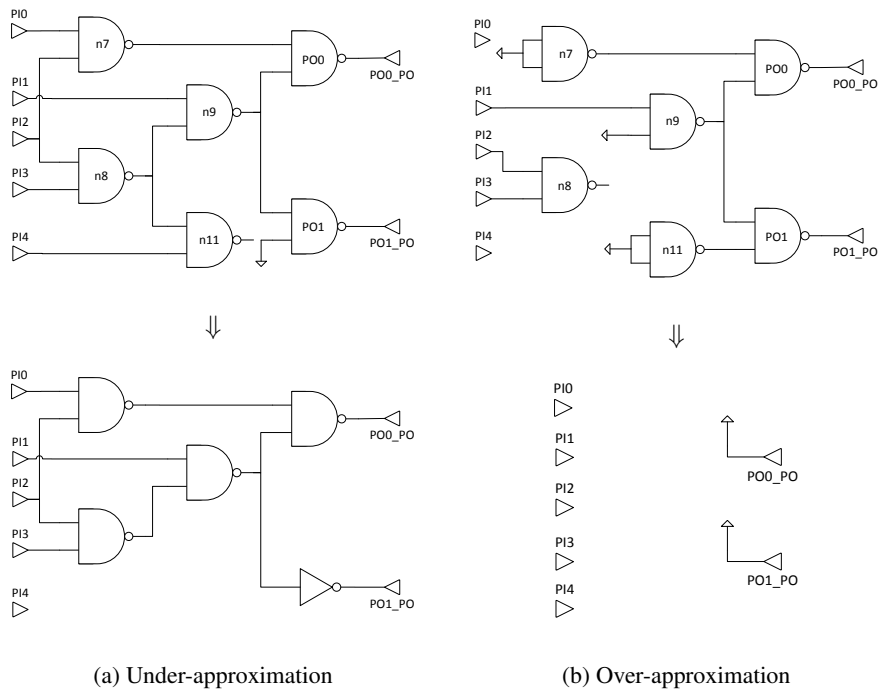


Figure A.9: Approximate circuits for a 0.19 threshold

mation, illustrated in Figure A.10b.

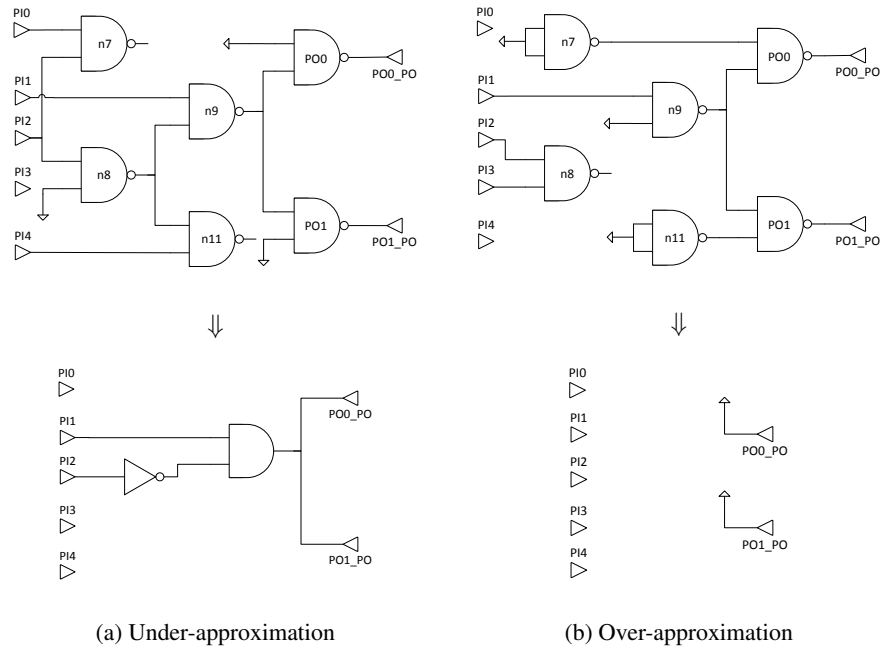


Figure A.10: Approximate circuits for a 0.197 threshold

Now let us assume a testability threshold of 0.2. In addition to the previous faults, $n9 \Rightarrow PO1$ stuck-at 1 becomes approximated, which belongs to the under-approximation side. Fault $n8$ stuck-at 1 is not taken into account as it is applied on a stem line. Resulting approximate circuits are shown in Figure A.11. It can be appreciated that primary output PO1 is constant in both approximate circuits. This allows to completely remove voter for that particular output, as both replicas will always provide opposite values, thus leaving output PO1 unprotected.

Finally, consider a threshold of 0.25. Fault $PI2 \Rightarrow n8$ stuck-at 1 is added to the list of approximated faults. After performing proper simplifications, the full trivial approximation is obtained, as it is illustrated in Figure A.12, where all the primary outputs in both circuits are tied to logic constants. As a consequence, voters are not required at all, leaving just the unprotected original circuit.

A.4 Approximation by dynamic testability measures

For a better comprehension of the approximation generation method by means of dynamic testability measures introduced in chapter 5, the whole process for c17 benchmark is detailed in this section, which is divided in several subsections. The first one corresponds to the initialization of the algorithm, and then there is one subsection per

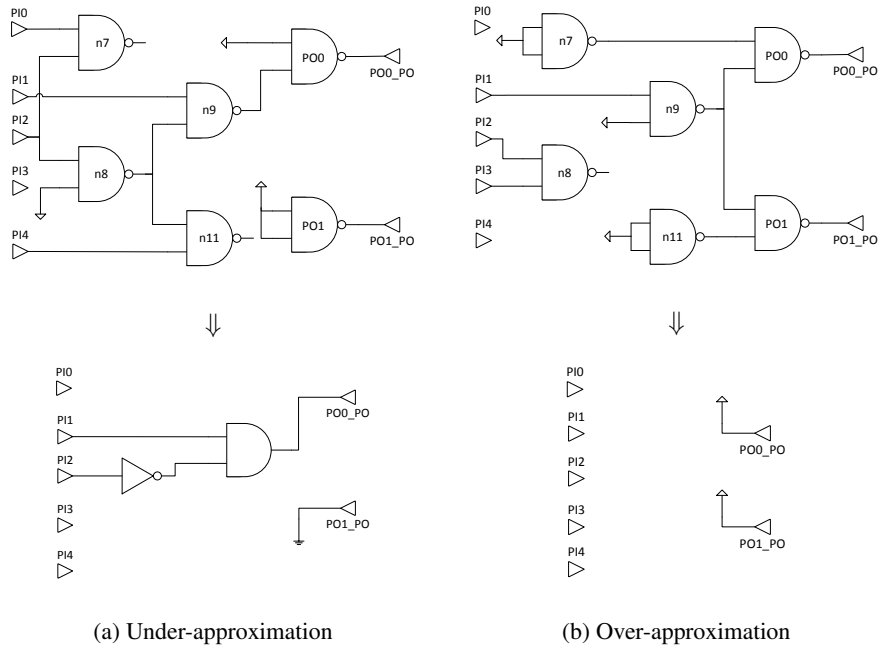


Figure A.11: Approximate circuits for a 0.2 threshold

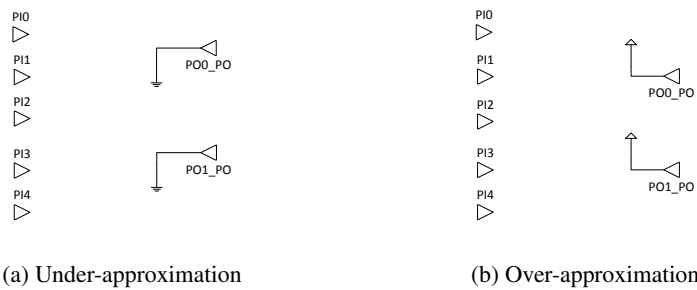


Figure A.12: Approximate circuits for a 0.25 threshold - Trivial approximation

iteration of the algorithm, that is, for each approximated fault. A total of 10 iterations are performed, from the full TMR to the trivial approximation. In the end, a comparison between estimated and real fault probabilities is presented, in order to show the accuracy of proposed approach.

Initial probability computation

The first step in the initialization phase of approximation generation algorithm is a unateness check of the target design by means of parity computation. As said in section A.2, c17 is already a fully unate circuit, and therefore no logic transformations are required. If the target design would be binate, then it should be transformed into a unate circuit.

Next, the list of candidate stuck-at faults is generated. In this example only the compact list of faults is considered, with the idea of having a small and manageable fault list. And as usual, faults in stem lines are excluded, just individual lines are considered. Faults within the list are classified according to whether they produce an over- or under-approximation, which depends just in the fault value and the parity of the line where the fault is applied, as explained in section 3.3. The candidate faults are next listed, already split in two groups according to their classification:

- Under-approximation faults: $PI2 \rightarrow n8/1$, $PI3/1$, $n7/1$, $n9 \rightarrow PO0/1$, $n9 \rightarrow PO1/1$, $n11/1$, $PO0/0$ and $PO1/0$.
- Over-approximation faults: $PI0/1$, $PI1/1$, $PI2 \rightarrow n7/1$, $PI4/1$, $n8 \rightarrow n9/1$, $n8 \rightarrow n11/1$, $PO0/1$ and $PO0/1$.

Now, fault probabilities are initialized. This is performed by implying each fault, that is, applying the corresponding controllability and observability conditions and deducing its justification frontier, and finally computing the probability value according to the J-SMA as explained in sections 5.3.1 and 5.3.2. Next this process is applied to each fault in the list.

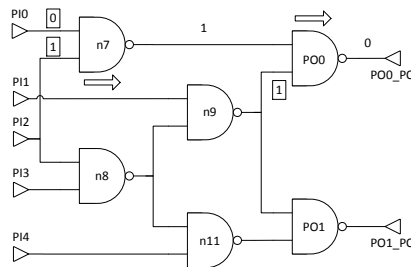


Figure A.13: Implication of fault $PI0/1$

First fault is $PI0/1$, which can only propagate through output $PO0$. Controllability condition of this fault corresponds to assignment $PI0=0$. In addition, nodes $n7$ and

PO0 are dominators of this fault, and therefore assignments on its side inputs PI2=1 and n9=1 become observability conditions of the fault. This assignments are propagated through the circuit by direct implication as shown in Figure A.13. Assignment PI0=0 implies n7=1, and this along with n9=1 imply PO0=0. But the initial set of mandatory assignments is not justified, and therefore it becomes the justification frontier of fault PI0/1. In conclusion, probability of this fault is computed as the product of probabilities of assignments PI0=0, PI2=1 and n9=1. But in order to resolve all possible dependencies, each probability is computed conditioned to the set of deduced assignments for the current J-SMA. This means that probability of assignment n9=1 is conditioned to PI2=1. Assuming a probability of 0.5 for every input, then

$$\begin{aligned} P(n9|PI2) &= 1 - (P(PI1) \cdot P(n8|PI2)) = \\ &= 1 - (P(PI1) \cdot (1 - P(PI3))) = 1 - (0.5 \cdot 0.5) = 0.75 \end{aligned}$$

And the probability of fault PI0/1 is then

$$P(PI0/1) = P(\overline{PI0}) \cdot P(PI2) \cdot P(n9|PI2) = 0.5 \cdot 0.5 \cdot 0.75 = 0.1875$$

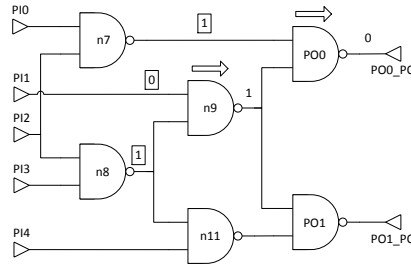


Figure A.14: Implication of fault PI1/1 through PO0

Next fault in the list is PI1/1. This fault can propagate through both outputs, and therefore one justification frontier is implied per output. Let us start with output PO0. Assignment PI1=0 is identified as the fault's controllability condition. Nodes n9 and PO0 are the dominators of this fault with respect to output PO0, so the observability conditions are the assignments that put the sensitizing value to this dominator nodes. In other words, assignments n7=1 and n8=1 are the observability conditions. This assignments are propagated through the circuit as in Figure A.14, implying the assignments n9=1 and PO0=0. No more signal values can be deduced, so the initial SMA becomes the justification frontier $J_{PI1/1_0} = \overline{PI1} \cdot n7 \cdot n8$. Then probability of this J-SMA is computed as

$$P(PI1/1_0) = P(\overline{PI1}) \cdot P(n7) \cdot P(n8) = 0.5 \cdot 0.75 \cdot 0.75 = 0.28125$$

It can be appreciated that both signals n7 and n8 depend on PI2, but this dependence cannot be identified by the implication algorithm. As result there is a slight deviation

from the real probability value. By using a more sophisticated implication method more dependencies could be found and therefore probability estimations would be more precise, but at a cost of a higher computational cost.

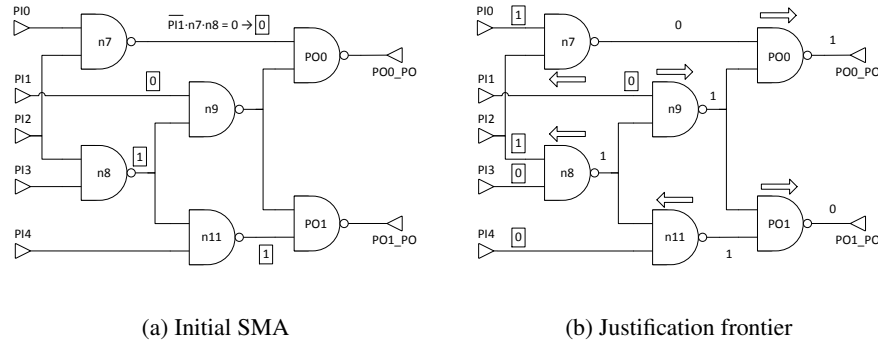


Figure A.15: Implication of fault PI1/1 through PO1

Later implication of the fault PI1/1 is performed with respect to output PO1. Here assignments $n8=1$ and $n11=1$ are inferred as observability conditions, and along with controllability condition $PI1=0$ they are all possible mandatory assignments deduced from the topology of the circuit. But in order to easily compute fault probabilities, input vectors already taken into account with output PO0 have to be excluded. In other words, $J_{PI1/1_0} = \overline{PI1} \cdot n7 \cdot n8$ is included here as an additional restriction that must not be fulfilled. Lines PI1 and n8 have already assignments which are compatible with $J_{PI1/1_0}$, so line n7 must be assigned to 0. These four assignments form the initial SMA for this step as depicted in Figure A.15a. Then they are propagated through the circuit. $n7=0$ is justified by assigning 1 to both of its inputs and, because $PI2=1$, assigning $PI3=0$ is the only way that $n8=1$, which becomes justified. Therefore, lines n7 and n8 are removed from the SMA in favour of newly deduced assignments. Finally, $n11=1$ is justified by $PI4=0$, because n8 already has a value 1, so this is removed from the SMA as well, and replaced with $PI4=0$. Apart from this, these assignments are propagated towards circuit outputs. All this process is summarized in Figure A.15b. In the end, J-SMA is composed by assignments $PI0=1$, $PI1=0$, $PI2=1$, $PI3=0$ and $PI4=0$. Assigning a probability of 0.5 to each input, then this J-SMA has a probability of $P(PI1/1_1) = 0.5^5 = 0.03125$.

Finally, probability of the whole fault PI1/1 is computed as the sum of both partial values, because they have been computed in such a way that intersection between both J-SMAs is null. Therefore,

$$P(PI1/1) = P(PI1/1_0) + P(PI1/1_1) = 0.28125 + 0.03125 = 0.3125$$

Fault $PI2 \rightarrow n7/1$ comes next. Controllability condition $PI2=0$ and observability conditions $PI0=1$ and $n9=1$ form the initial set of mandatory assignments for this fault, as shown in Figure A.16a. Then these values are propagated through the circuit. $PI2=0$ implies $n8=1$, which in turn forces the assignment $PI1=0$ in order to justify $n9=1$. n7

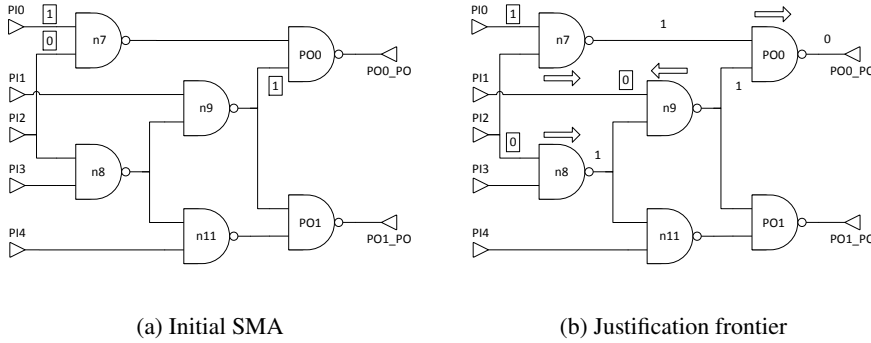


Figure A.16: Implication of fault $PI2 \rightarrow n7/1$

and PO0 receive assignments as well by propagation. All this implication process is shown in Figure A.16b. No more assignments can be inferred, and the final J-SMA is the set of conditions $PI0=1, PI1=0, PI2=0$. The probability of current fault is then $P(PI2 \rightarrow n7/1) = 0.5^3 = 0.125$, taking into account that each input has a probability equal to 0.5.

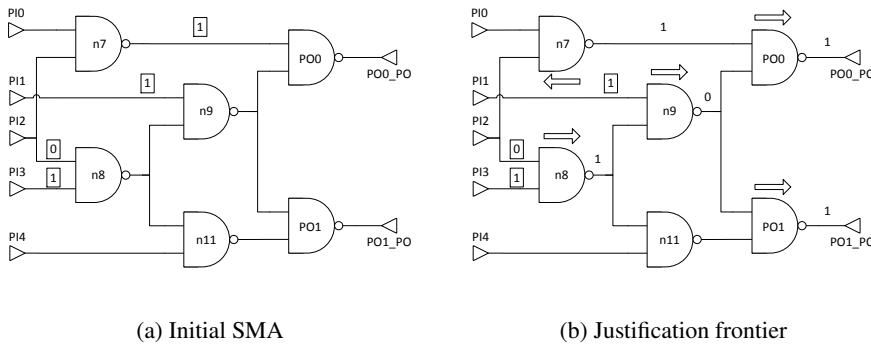


Figure A.17: Implication of fault $PI2 \rightarrow n8/1$ through PO0

The following fault is $PI2 \rightarrow n8/1$. This fault can propagate to both outputs, and therefore one implication per output is performed as usual. With respect to PO0, the initial SMA includes assignments $PI2=0, PI3=1, PI1=1$ and $n7=1$ as depicted in Figure A.17a. During implication process, $n7=1$ is justified by $PI2=0$, and initial conditions are propagated until primary outputs as shown in Figure A.17b. At the end of the implication process, the SMA $J_{PI2 \rightarrow n8/1_0}$ is formed by $PI1=1, PI2=0$ and $PI3=1$, which become the J-SMA, with a probability of $P(PI2 \rightarrow n8/1_0) = 0.5^3 = 0.125$

In the case of output PO1, only $PI2=0$ and $PI3=1$ can be initially inferred as mandatory assignments. Only nodes n8 and PO1 are dominators of this fault, and PO1 does not have side inputs because the fault has two different propagation paths. But by adding the restriction $\overline{J_{PI2 \rightarrow n8/1_0}}$ intended to exclude those input vectors already con-

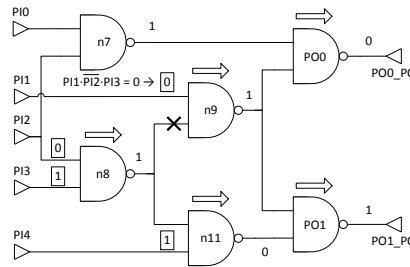


Figure A.18: Implication of fault PI2→n8/1 through PO1

sidered for output PO0, additional MAs can be inferred. In particular, PI1 must be assigned to 0 because PI2 and PI3 already have values compatible with $J_{PI2 \rightarrow n8/1_0}$. This new MA causes that propagation path through node n9 to be blocked. As a consequence, node n11 becomes a dynamic dominator of the fault and new observability conditions can be inferred. Assignment PI4=1 is included in the SMA, and all these values are propagated through the circuit as shown in Figure A.15. At the end of implication process, justification frontier is formed by assignments PI1=0, PI2=0, PI3=1 and PI4=1. From this J-SMA a probability is computed as usual, being $P(PI2 \rightarrow n8/1_1) = 0.5^4 = 0.0625$.

Finally, as both partial J-SMAs are mutually exclusive, the whole probability of fault PI2→n8/1 can be computed by simply adding probabilities, thus having

$$P(PI2 \rightarrow n8/1) = 0.125 + 0.0625 = 0.1875$$

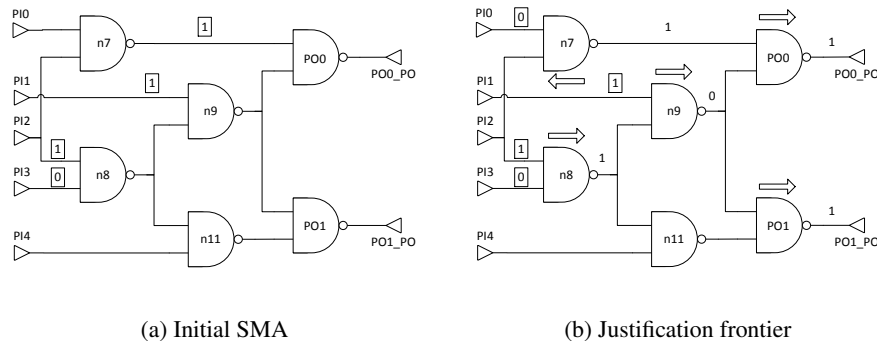


Figure A.19: Implication of fault PI3/1 through PO0

Now it is the turn of fault PI3/1. This one propagates to both outputs, so it has to be implied in an output by output basis. First propagation through PO0 is addressed. Controllability and observability conditions are inferred first, thus generating the initial SMA PI1=1, PI2=1, PI3=0 and n7=1, as shown in Figure A.19a. Assignment n7=1 is

justified by $PI0=0$, because $PI2$ is already set to 1. Like this, other assignments are implied from the initial SMA as shown in Figure A.19b. Implication ends, and J-SMA $J_{PI3_0} = \overline{PI0} \cdot PI1 \cdot PI2 \cdot \overline{PI3}$ is extracted. This J-SMA has associated a probability of $P(PI3/1_0 = 0.5^4) = 0.0625$.

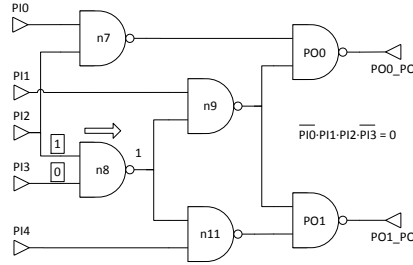


Figure A.20: Implication of fault $PI3/1$ through $PO1$

Then, propagation of fault $PI3/1$ through output $PO1$ is implied. The same problem as with fault $PI2 \rightarrow n8/1$ arises here. As there are several propagation paths, just assignments $PI2=1$ and $PI3=0$ can be initially inferred as in Figure A.20. But in this case, applying the additional restriction $\overline{J_{PI3_0}}$ does not allow to deduce additional MAs. So in this case the J-SMA remains as $J_{PI3/1_1} = PI2 \cdot \overline{PI3}$, which corresponds a probability of 0.25. Notwithstanding, the unjustified restriction still serves to infer a probability correlation coefficient which leads to a more realistic probability estimation. This coefficient k_J is derived from the probability of J_{PI3_0} , conditioned to the set of assignments in J_{PI3_1} as follows

$$k_J = 1 - P(J_{PI3_0} | J_{PI3_1}) = 1 - P(\overline{PI0} \cdot P(PI1)) = 1 - 0.25 = 0.75$$

and partial probability of $PI3/1$ through $PO1$ is then

$$P(PI3/1) = P(J_{PI3_1}) \cdot k_J = 0.25 \cdot 0.75 = 0.1875$$

which is a value closer to the real probability -0.125- than if correlation coefficient were not applied.

Total probability of fault $PI3/1$ is finally computed by adding both partial probabilities, assuming that they are independent between them. Therefore,

$$P(PI3/1) = P(PI3/1_0) + P(PI3/1_1) = 0.0625 + 0.1875 = 0.25$$

Next fault in the list is $PI4/1$. The initial SMA for this fault is formed by controllability condition $PI4=0$ and observability conditions $n8=1$ and $n9=1$, as shown in Figure A.21a. Later these values are propagated through the circuit as it can be seen in Figure A.21b. It is relevant to notice that assignment $PI1=0$ is required to justify $n9=1$, because the other input to node $n9$ is already set to 1. Finally, J-SMA of the fault contains the assignments $PI1=0$, $PI4=0$ and $n8=1$, which corresponds a probability of

$$P(PI4/1) = P(\overline{PI1}) \cdot P(\overline{PI4}) \cdot P(n8) = 0.5 \cdot 0.5 \cdot 0.75 = 0.1875$$

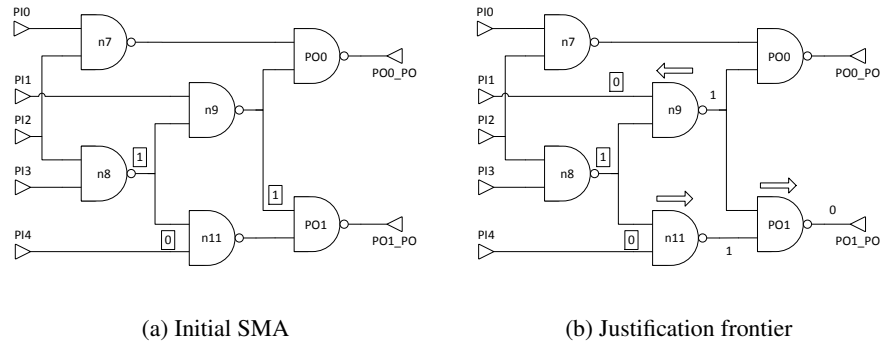


Figure A.21: Implication of fault PI4/1

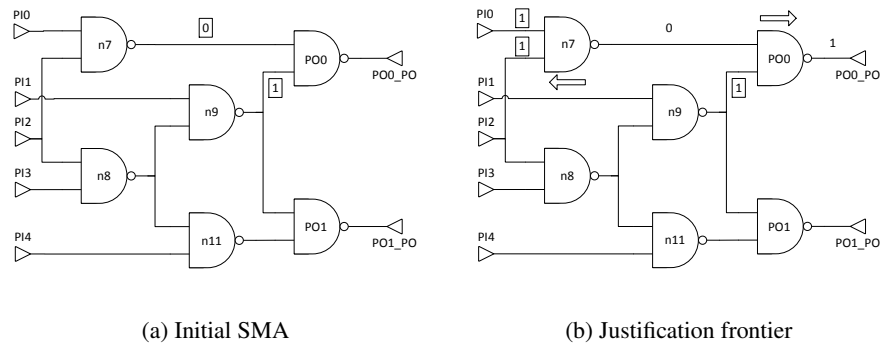


Figure A.22: Implication of fault n7/1

Following with fault $n7/1$. The controllability and observability conditions of this fault are $n7=0$ and $n9=1$ respectively, which are represented in Figure A.22a. This is the initial set of mandatory assignments, which are then propagated as shown in Figure A.22b. Assignment $n7=0$ implies $PO0=1$, and at the same time it is justified by $PI0=1$ and $PI2=1$. In the end, justification frontier is $J_{n7/1} = PI0 \cdot PI2 \cdot n9$. Probability of fault $n7/1$ is then derived from assignments belonging to J-SMA as usual, computing the probability of each assignment conditioned to the whole set of deduced values. This allows to resolve the dependence between signals $PI2$ and $n9$ present in this case by operating with probability $P(n9|PI2)$. This value has already been computed during implication of fault $PI0/1$ with a result of 0.75. In conclusion, probability of fault $n7/1$ is

$$P(n7/1) = P(PI0) \cdot P(PI2) \cdot P(n9|PI2) = 0.5 \cdot 0.5 \cdot 0.75 = 0.1875$$

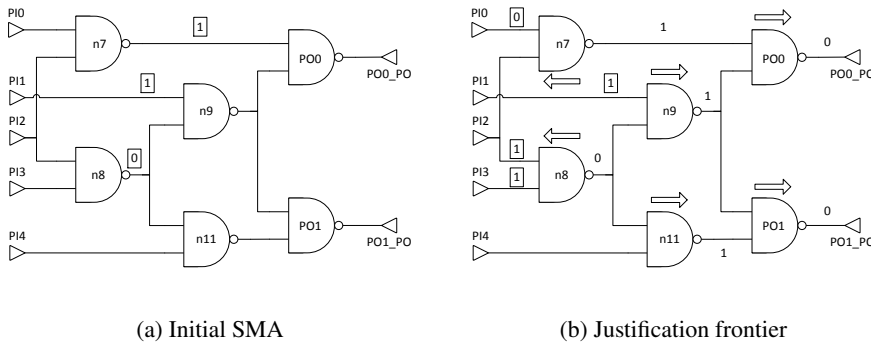
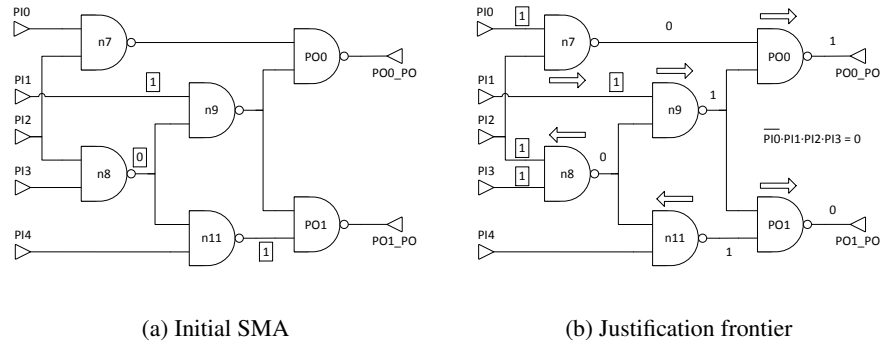


Figure A.23: Implication of fault $n8 \rightarrow n9/1$ through $PO0$

Later fault $n8 \rightarrow n9/1$ follows. This fault may propagate through both outputs, so it is implied with respect to each output independently as usual. Let us start with output $PO0$, which has nodes $n9$ and $PO0$ as dominators. The initial SMA is formed by controllability condition $n8=0$ and observability conditions $PI1=1$ and $n7=1$ as shown in Figure A.23a. Then these values are implied through the circuit. Assignment $n8=0$ is justified by assigning $PI2=1$ and $PI3=1$, and because of this $n7=1$ has to be justified by $PI0=0$. In addition, these values are propagated to circuit outputs -see Figure A.23b for details. In the end, J-SMA $J_{n8 \rightarrow n9/1_0} = \overline{PI0} \cdot PI1 \cdot PI2 \cdot PI3$ is obtained. As all assignments in the justification frontier correspond to primary inputs, probability of this set of conditions is easily computed as $P(n8 \rightarrow n9/1_0) = 0.5^4 = 0.0625$

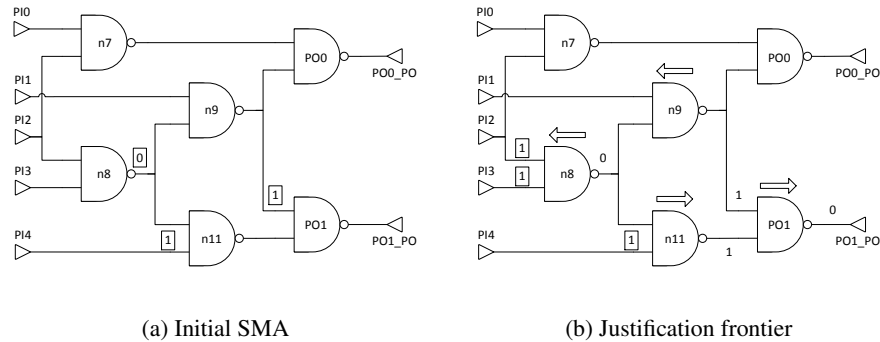
With respect to output $PO1$, nodes $n9$ and $PO1$ are dominators of the fault. So assignments $n8=0$, $PI1=1$ and $n11=1$ initially belong to the SMA as shown in Figure A.24a. In addition, restriction $\overline{J_{n8 \rightarrow n9/1_0}}$ is imposed in order to exclude those input vectors already detected with previous implication. Then the initial conditions are propagated through the circuit by direct implication. Again, $n8=0$ forces both inputs $PI2=1$ and $PI3=1$, thus becoming justified. Being already $PI1$, $PI2$ and $PI3$ set to 1, it is inferred that $PI0$ must hold value 1 as well to fulfill the restriction $\overline{J_{n8 \rightarrow n9/1_0}}$. Finally, these values are propagated to $n7$, $n9$, $n11$ and both outputs as represented in Figure

Figure A.24: Implication of fault $n8 \rightarrow n9/1$ through PO1

A.24b. Finally, assignments $PI0=1$, $PI1=1$, $PI2=1$ and $PI3=1$ form the justification frontier of the fault, which has associated a probability $P(n8 \rightarrow n9/1_1) = 0.5^4 = 0.0625$.

Once both partial probabilities have been computed, probability of the complete fault is obtained. This can be done by simply addition because both J-SMAs are mutually exclusive. Therefore

$$P(n8 \rightarrow n9/1) = P(n8 \rightarrow n9/1_0) + P(n8 \rightarrow n9/1_1) = 0.0625 + 0.0625 = 0.125$$

Figure A.25: Implication of fault $n8 \rightarrow n11/1$

Next fault is $n8 \rightarrow n11/1$. Controllability and observability conditions are applied, resulting in the initial MAs $n8=0$, $PI4=1$ and $n9=1$ as it can be seen in Figure A.25a. During implication process, assignment $n9=1$ becomes justified by $n8=0$, which at the same time is justified by $PI2=1$ and $PI3=1$ as shown in Figure A.25b. In addition, values are inferred for nodes $n11$ and $PO1$ by propagation. After implication, justification frontier contains the assignments $PI2=1$, $PI3=1$ and $PI4=1$. From here fault probability is computed, thus having $P(n8 \rightarrow n11/1) = 0.5^3 = 0.125$.

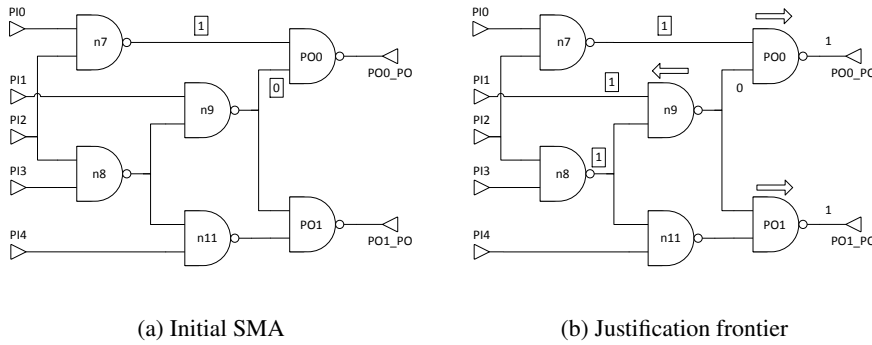


Figure A.26: Implication of fault $n9 \rightarrow PO0/1$

Fault $n9 \rightarrow PO0/1$ comes next. Controllability condition $n9=0$ and observability condition $n7=1$ form the initial SMA as shown in Figure A.26a. Then assignment $n9=0$ propagates to $PO0=1$, and it becomes justified by $PI1=1$ and $n8=1$, which is represented in Figure A.26b. No more assignments can be inferred, so implication finishes and justification frontier $PI1=1$, $n7=1$ and $n8=1$ is extracted. Probability of this fault is then computed as the product of probabilities of every assignment in the J-SMA as follows

$$P(n9 \rightarrow PO0/1) = P(PI1) \cdot P(n7) \cdot P(n8) = 0.5 \cdot 0.75 \cdot 0.75 = 0.28125$$

Again, there is a slight deviation from the real probability value because implication method is not able to find the existing interdependence between $n7$ and $n8$.

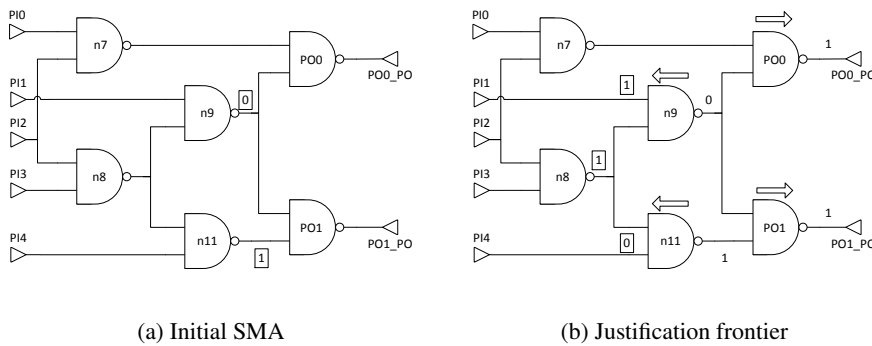


Figure A.27: Implication of fault $n9 \rightarrow PO1/1$

Following with fault $n9 \rightarrow PO1/1$. $n9=0$ and $n11=1$ are identified as controllability and observability conditions respectively - see Figure A.27a. From here, assignment $n9=0$ is justified by $PI1=1$ and $n8=1$, and it is propagated to $PO1=1$ as well. At the same time, MA $PI4=0$ is inferred as the only way of justifying $n11=1$ once $n8$ has

been set to 1. All these steps are depicted in Figure A.27b. Implication stops here, and assignments $PI1=1$, $PI4=0$ and $n8=1$ are identified as belonging to J-SMA. Fault probability is finally computed as follows

$$P(n9 \rightarrow PO1/1) = P(PI1) \cdot P(\overline{PI4}) \cdot P(n8) = 0.5 \cdot 0.5 \cdot 0.75 = 0.1875$$

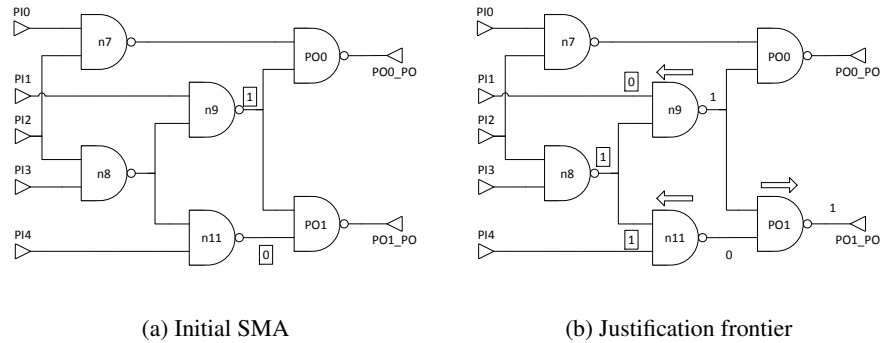


Figure A.28: Implication of fault $n11/1$

Now it is the turn of fault $n11/1$, which is pretty similar to the previous one. The initial set of mandatory assignments contains $n9=1$ and $n11=0$ as shown in Figure A.28a. These assignments are subsequently implied. $n11=0$ propagates to output $PO1$, while at the same time it is justified by assignments $PI4=1$ and $n8=1$. A consequence of this, $PI1$ must be set to 0 in order to justify $n9=1$. All this can be seen in Figure A.28b. When implication process finishes, J-SMA $J_{n11/1} = \overline{PI1} \cdot PI4 \cdot n8$ is extracted. Probability of fault $n11/1$ is inferred from justification frontier, resulting in

$$P(n11/1) = P(\overline{PI1}) \cdot P(PI4) \cdot P(n8) = 0.5 \cdot 0.5 \cdot 0.75 = 0.1875$$

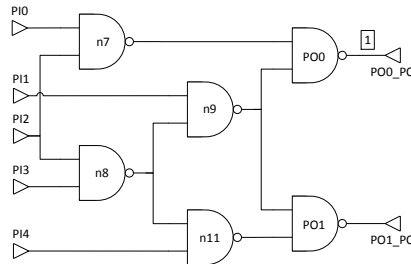


Figure A.29: Implication of fault $PO0/0$

Next fault in the list is $PO0/0$, the first of the four faults located at primary outputs. Only controllability condition $PO0=1$ can be inferred as initial MA, and no additional

assignments are deduced during implication, as shown in Figure A.29. Therefore, justification frontier for this fault only contains assignment $PO0=1$, whose probability is considered equal to that of current fault. Therefore, probability of fault $PO0/0$ is, according with COP

$$\begin{aligned} P(PO0/0) &= 1 - P(n7) \cdot P(n9) = 1 - P(n7) \cdot (1 - P(PI1) \cdot P(n8)) = \\ &= 1 - 0.75 \cdot (1 - 0.5 \cdot 0.75) = 0.53125 \end{aligned}$$

It is clear that this is not the real probability value. There are implications that obviously have not been properly inferred.

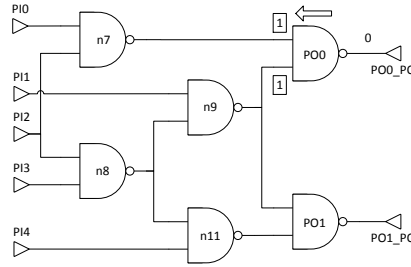


Figure A.30: Implication of fault $PO0/1$

Later fault $PO0/1$ comes. Here the initial SMA is the controllability condition $PO0=0$, which is justified by assignments $n7=1$ and $n9=1$ as shown in Figure A.30. No more implications can be inferred, and J-SMA is then $J_{PO0/1} = n7 \cdot n9$. Fault probability is finally computed as

$$\begin{aligned} P(PO0/1) &= P(n7) \cdot P(n9) = P(n7) \cdot (1 - P(PI1) \cdot P(n8)) = \\ &= 0.75 \cdot (1 - 0.5 \cdot 0.75) = 0.46875 \end{aligned}$$

It can be appreciated that this value is the complimentary of the probability of previous fault $PO0/0$. This makes sense, because both faults are opposites and they are located at primary outputs, so there is no possibility of error masking.

Then fault $PO1/0$ follows. Similarly to $PO0/0$, $PO1=1$ is the only assignment that can be inferred during implication, as shown in Figure A.31. Therefore fault probability is computed by means of simple COP as

$$\begin{aligned} P(PO1/0) &= 1 - P(n9) \cdot P(n11) = 1 - (1 - P(PI1) \cdot P(n8)) \cdot \\ &\cdot (1 - P(PI4) \cdot P(n8)) = 1 - (1 - 0.5 \cdot 0.75) \cdot (1 - 0.5 \cdot 0.75) = 0.609375 \end{aligned}$$

Again, there is a deviation in probability estimations due to an incomplete justification. Probability of assignment $n8=1$ is counted up to two times.

Finally, fault $PO1/1$ is implied. In this case controllability condition $PO1=0$ is applied, which becomes justified by assignments $n9=1$ and $n11=1$, as it can be seen in

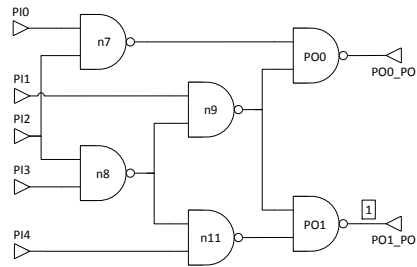


Figure A.31: Implication of fault PO1/0

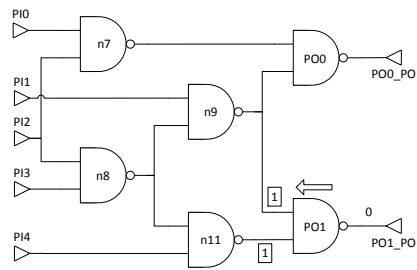


Figure A.32: Implication of fault PO1/1

Figure A.32. Implication stops here, and justification frontier $J_{PO1/1} = n9 \cdot n11$ is extracted, from which probability of fault PO1/1 is inferred, thus having

$$\begin{aligned} P(PO1/1) &= P(n9) \cdot P(n11) = (1 - P(PI1) \cdot P(n8)) \cdot (1 - P(PI4) \cdot P(n8)) = \\ &= (1 - 0.5 \cdot 0.75) \cdot (1 - 0.5 \cdot 0.75) = 0.390625 \end{aligned}$$

Again, this result is the complimentary of previous fault, PO1/0.

Fault	Justification frontier		Probability		
	PO0	PO1	PO0	PO1	Total
PI0/1	$\overline{PI0} PI2 n9$	-	0.1875	0	0.1875
PI1/1	$\overline{PI1} n7 n8$	$PI0 \overline{PI1} PI2 \overline{PI3} \overline{PI4}$	0.2812	0.0313	0.3125
PI2→n7/1	$PI0 \overline{PI1} \overline{PI2}$	-	0.125	0	0.125
PI2→n8/1	$PI1 \overline{PI2} PI3$	$\overline{PI1} \overline{PI2} PI3 PI4$	0.125	0.0625	0.1875
PI3/1	$\overline{PI0} PI1 PI2 PI3$	$PI2 \overline{PI3}$	0.0625	0.1875	0.25
PI4/1	-	$\overline{PI1} \overline{PI4} n8$	0	0.1875	0.1875
n7/1	$PI0 PI2 n9$	-	0.1875	0	0.1875
n8→n9/1	$\overline{PI0} PI1 PI2 PI3$	$PI0 PI1 PI2 PI3$	0.0625	0.0625	0.125
n8→n11/1	-	$PI2 PI3 PI4$	0	0.125	0.125
n9→PO0/1	$PI1 n7 n8$	-	0.2812	0	0.2812
n9→PO1/1	-	$PI1 \overline{PI4} n8$	0	0.1875	0.1875
n11/1	-	$\overline{PI1} PI4 n8$	0	0.1875	0.1875
PO0/0	$PO0$	-	0.5312	0	0.5312
PO0/1	$n7 n9$	-	0.4688	0	0.4688
PO1/0	-	$PO1$	0	0.6094	0.6094
PO1/1	-	$n9 n11$	0	0.3906	0.3906

Table A.2: Summary of initial probability computation

Table A.2 collects the results of this initial phase in the approximation generation algorithm. It contains the justification frontiers obtained for each fault and its probabilities, both per output and combined. These results determine the first candidate to be approximated. According to the proposed criteria, it would be selected the fault with the lowest probability value. It can be seen that there are several candidate faults: PI2→n7/1, n8→n9/1 and n8→n11/1, all of them with a probability equal to 0.125. In this case a second criteria is used to discriminate which fault is selected: the fault, among those with minimum probability, which produces the highest area savings is approximated. Real area savings can only be measured after performing logic synthesis, but a quick estimation can be obtained by counting the number of lines backward between fault injection site and multiple fanout points, which can be seen as the size of the immediate transitive fanin. But in this case such criteria does not help in selecting the best candidate. All three faults only remove one line if approximated, because both PI2 and n8 are multiple fanout nodes. In conclusion, any of these faults can be the first candidate, which has to be arbitrarily chosen. Let us assume that fault PI2→n7/1 is first selected, which is an over-approximate fault.

In this point, two exact replicas of target circuit have already been generated, which will serve as under- and over-approximations respectively. Because no faults have been approximated, the sets of approximation conditions A_F and A_H are null and the

estimated error probability is equal to 0. This EP is the finishing condition of the algorithm: faults are iteratively selected and approximated as long as EP is under given error target, or until all faults have been approximated.

1st approximation: $PI2 \rightarrow n7/1$

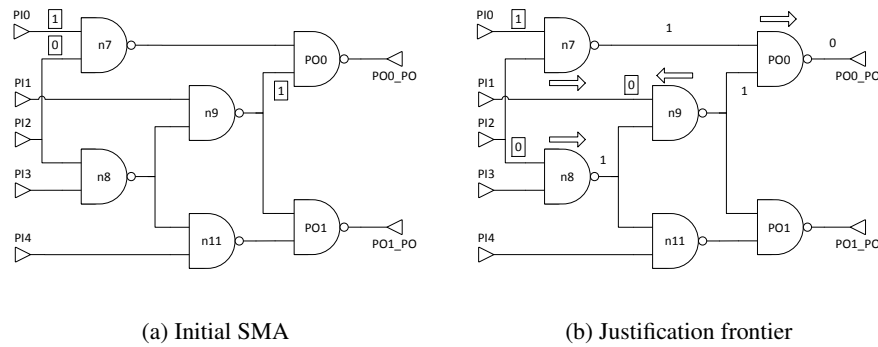


Figure A.33: Approximation condition of fault $PI2 \rightarrow n7/1$

First fault selected to be approximated is $PI2 \rightarrow n7/1$. But before that, approximation conditions of the fault have to be obtained. This is performed by implying the fault on its correspondent approximation, the over-approximate circuit in this case. Because there are no faults approximated yet, over-approximation is an exact replica of the original circuit. Therefore, implication of fault $PI2 \rightarrow n7/1$ in order to obtain its approximation condition is identical to that performed in the initial probability computation step, as it can be seen in Figure A.33. Resulting J-SMA is identified as the approximation condition of the fault, being $A_{PI2 \rightarrow n7/1} = PI0 \cdot \overline{PI1} \cdot \overline{PI2}$ with respect to output PO0, and null for PO1.

With this approximation condition, the effect of $PI2 \rightarrow n7/1$ approximation over remaining faults can be estimated, which serves to update fault probabilities and global EP. This is performed by re-implying each fault intersected with $A_{PI2 \rightarrow n7/1}$. Because this is the first approximated fault, the sets of approximation conditions A_F and A_H are still null, and therefore no input vectors have to be excluded yet. Not all faults have to be re-implicated, only those of the same type than $PI2 \rightarrow n7/1$, i.e., over-approximate faults. And among them, not every fault is affected. Justification frontiers of faults $PI0/1$ and $n8 \rightarrow n9/1$ are not compatible with approximation condition $A_{PI2 \rightarrow n7/1}$, and faults $PI4/1$, $n8 \rightarrow n11/1$ and $PO1/1$ do not propagate through the same output as fault $PI2 \rightarrow n7/1$. For all these faults, result of re-implication is the null set, and therefore approximation has no impact on their probabilities.

Fault $PI1/1$ is one of those affected faults. Figure A.34a shows the set of assignments deduced during initial fault implication. These are now intersected with the approximation condition $A_{PI2 \rightarrow n7/1}$ and re-implicated. The new MA $PI2=0$ automatically justifies both $n7=1$ and $n8=1$ assignments, as it can be seen in Figure A.34b. Finally, J-SMA for this set of conditions is $J_{PI1/1}^1 = PI0 \cdot \overline{PI1} \cdot \overline{PI2}$, which corresponds a

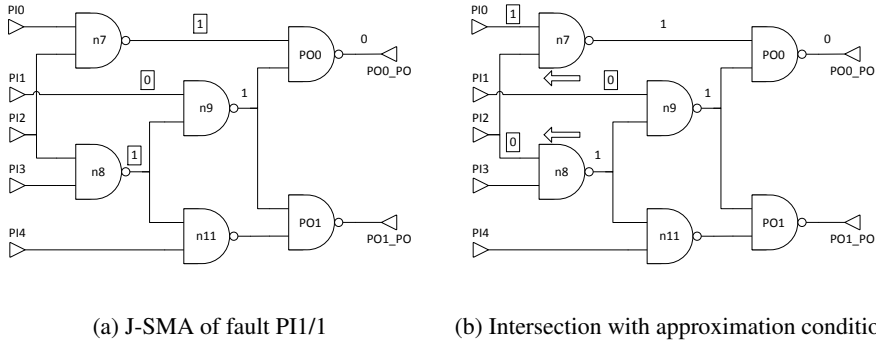


Figure A.34: Effect of fault $PI2 \rightarrow n7/1$ over $PI1/1$

probability of 0.125. This value is discounted from the original fault probability, which was equal to 0.3125, in order to reflect probability changes due to fault approximation, resulting in a value 0.1875. In addition, the inferred justification frontier is stored in order to exclude associated input vectors in future implications of fault $PI1/1$.

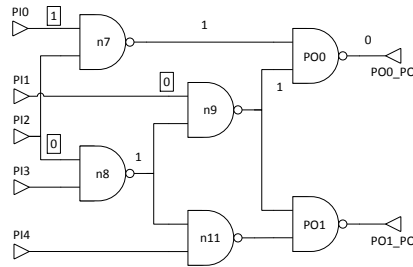
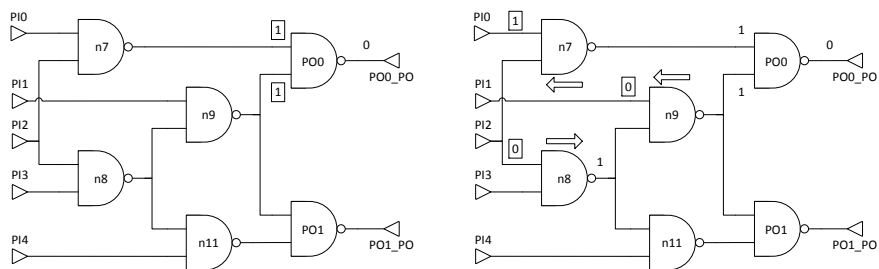


Figure A.35: Effect of fault $PI2 \rightarrow n7/1$ over itself

Fault $PI2 \rightarrow n7/1$ itself also has to be re-implied, even if this example is completely straightforward. Figure A.35 shows the result of initial implication for this fault. It can be seen that J-SMA coincides with its own approximation condition, so justification frontier suffers no changes and its probability is the same as that initially computed, 0.125. Therefore, fault probability drops to 0 when updated, reflecting that this fault has been completely unmasked.

$PO0/1$ is the last fault being affected by approximation of $PI2 \rightarrow n7/1$. Assignments deduced from implication of this fault are shown in Figure A.36a. Over this, $A_{PI2 \rightarrow n7/1}$ is applied. After introducing the MAs corresponding to the approximation condition, assignments $n7=1$ and $n9=1$ become justified by $PI2=0$ and $PI1=0$ respectively, as it can be seen in Figure A.36b. Finally, J-SMA $J_{PO0/1}^1 = PI0 \cdot \overline{PI1} \cdot \overline{PI2}$ is inferred. Assuming a probability of 0.5 at each primary input, this set of conditions has a probability of 0.125. Therefore, after updating fault probability drops to 0.3438.



(a) J-SMA of fault PO0/1

(b) Intersection with approximation condition

Figure A.36: Effect of fault $PI2 \rightarrow n7/1$ over PO0/1

Fault	Justification frontier (PO0)	Probability		
		Former	Unmasked	Updated
PI0/0	-	0.1875	0	0.1875
PI1/1	$PI0 \overline{PI1} \overline{PI2}$	0.3125	0.125	0.1875
$PI2 \rightarrow n7/1$	$PI0 \overline{PI1} \overline{PI2}$	0.125	0.125	0
PI4/1	-	0.1875	0	0.1875
$n8 \rightarrow n9/1$	-	0.125	0	0.125
$n8 \rightarrow n11/1$	-	0.125	0	0.125
PO0/1	$PI0 \overline{PI1} \overline{PI2}$	0.4688	0.125	0.3438
PO1/1	-	0.3906	0	0.3906

Table A.3: Summary of $PI2 \rightarrow n7/1$ approximation

Table A.3 summarizes the results of probability computations in the first iteration of the approximation generation algorithm. For each over-approximate fault, the table shows the justification frontier resulting from the intersection of each fault with $A_{PI2 \rightarrow n7/1}$, along with three probability values. From left to right, first it comes the probability initially computed, then the incremental one that corresponds to unmasked input vectors due to approximation of fault $PI2 \rightarrow n7/1$ -i.e., the probability of inferred justification frontier- and finally the updated fault probability, resulting from subtracting previous two values. Updated values represent the remaining probability of protected input vectors for that fault, or in other words, how many input combinations will become unmasked if such fault is next approximated.

Updating probabilities also serves to keep an estimation of global EP. Once all probabilities have been recomputed, EP is increased by the incremental probability of every fault, averaged with respect to the total number of faults. Thus,

$$EP_1 = \frac{\sum_i P(f_i \cap A_{PI2 \rightarrow n7/1})}{n} = \frac{3 \cdot 0.125}{16} = 2.3438\%$$

In other words, after $PI2 \rightarrow n7/1$ approximation, there is a 2.3438% probability that any fault in target circuit produces an error, considering all faults are equally probable. If a target EP lower than 2.3438% had been set, approximation generation algorithm would have stopped right after fault $PI2 \rightarrow n7/1$ would have been approximated.

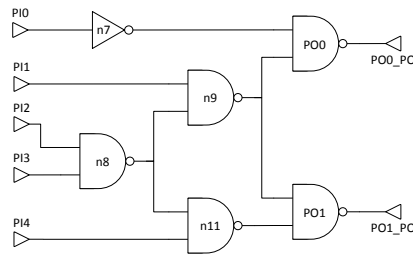


Figure A.37: Approximation of fault $PI2 \rightarrow n7/1$

Now approximation of candidate fault $PI2 \rightarrow n7/1$ can finally be performed by substituting the wire $PI2 \rightarrow n7$ with a logic 1, which is the correspondent faulty value. Because one of the inputs of node n7 is tied to a logic constant, this can be simplified with an inverter as shown in Figure A.37. This is now the over-approximation of c17 benchmark, while the under-approximation is still an exact replica. From now, fault $PI2 \rightarrow n7/1$ is officially approximated, and therefore excluded from candidate selection and implications.

After fault approximation, all remaining over-approximate faults are implied in the correspondent approximation in order to detect whether additional faults become accidentally approximated or not. In this iteration, none of the remaining faults is considered redundant, and therefore there are no additional approximate faults.

Under-approx fault	Probability	Over-approx fault	Probability
PI2→n8/1	0.1875	PI0/0	0.1875
PI3/1	0.25	PI1/1	0.1875
n7/1	0.1875	PI2→n7/1	-
n9→PO0/1	0.2812	PI4/1	0.1875
n9→PO1/1	0.1875	n8→n9/1	0.125
n11/1	0.1875	n8→n11/1	0.125
PO0/0	0.5312	PO0/1	0.3438
PO1/0	0.6094	PO1/1	0.3906

Table A.4: Fault probabilities after PI2→n7/1 approximation

Finally, next candidate is selected. Table A.4 contains the current value of all fault probabilities. The dash close to PI2→n7/1 fault indicates that it has already been approximated. Among the remaining faults, the best candidate is selected according to the already known criteria of minimum probability and maximum area savings. At this point there are two possible faults that can be selected according with this criteria: n8→n9/1 and n8→n11/1, both of them over-approximate faults with a probability of 0.125 and the same size of transitive fanin. Among them, n8→n9/1 is arbitrarily chosen as the next approximation candidate, and so the next iteration begins.

2nd approximation: n8→n9/1

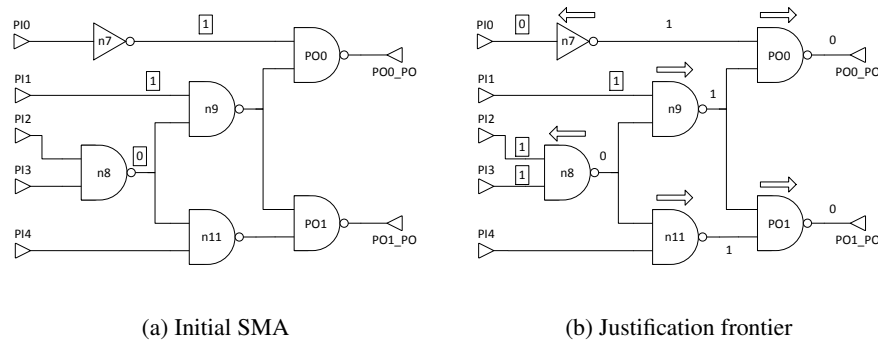


Figure A.38: Approximation condition of n8→n9/1 with respect to PO0

Once the next candidate fault has been selected, its approximation conditions have to be obtained. Fault n8→n9/1 may propagate through both outputs, and therefore one approximation condition is implied for each output. These are obtained by implying the fault in over-approximate circuit, which is no longer an exact replica of c17 benchmark. First n8→n9/1 is implied with respect to output PO0. The initial set of mandatory assignments is shown in Figure A.38a, which consist in the controllability condition

$n8=0$ and the observability conditions $PI1=1$ and $n7=1$. Then these are propagated through the circuit. $n7=1$ implies $PI0=0$, which replaces the previous assignment in the SMA. At the same time, $n8=0$ is justified by $PI2=1$ and $PI3=1$. And nodes $n9$, $n11$, $PO0$ and $PO1$ also receive values based on previous assignments. See Figure A.38b for details. At the end of the implication process, J-SMA $\overline{PI0} \cdot PI1 \cdot PI2 \cdot PI3$ is extracted, which becomes the approximation condition of fault $n8 \rightarrow n9/1$ with respect to output $PO0$, $A_{n8 \rightarrow n9/1_0}$.

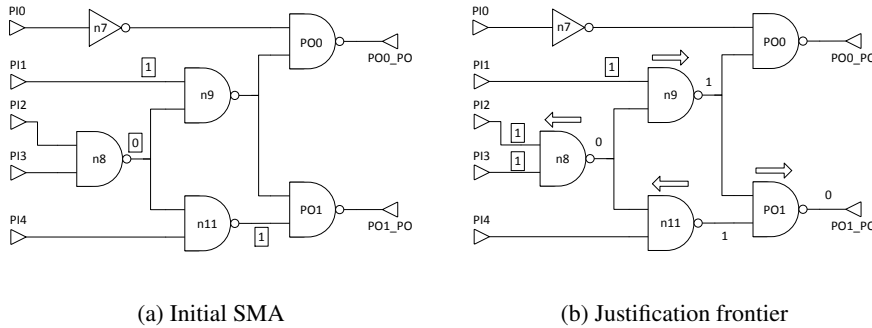
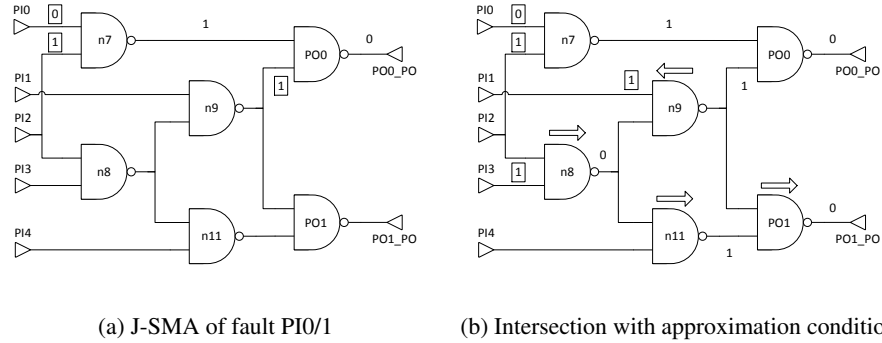


Figure A.39: Approximation condition of $n8 \rightarrow n9/1$ with respect to $PO1$

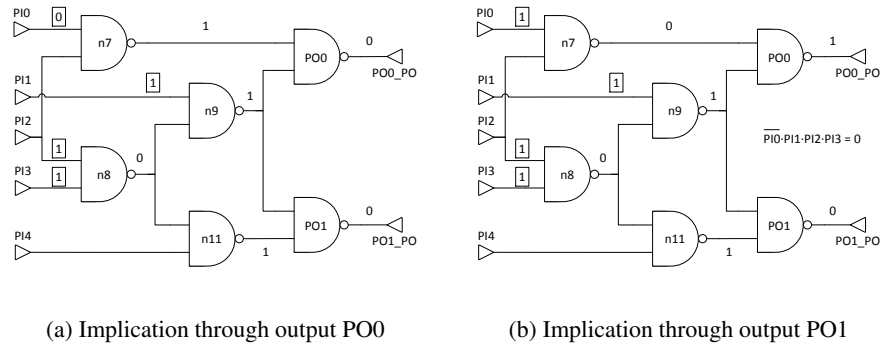
Then the same procedure is performed for output $PO1$. Now the initial set of MAs is formed by assignments $n8=0$, $PI1=1$ and $n11=1$, as shown in Figure A.39a, which are subsequently propagated by direct implication. Assignment $n11=1$ is automatically justified by $n8=0$, which at the same time infers $PI2=1$ and $PI3=1$, thus replacing the previous assignments in the SMA. This is represented in Figure A.39b, along with the propagation of $n9=1$ and $n11=1$ to $PO1$. Implication stops here, and approximation condition with respect to output $PO1$ is obtained, being $A_{n8 \rightarrow n9/1_0} = PI1 \cdot PI2 \cdot PI3$. It must be noted that in order to obtain approximation conditions every output is independent from each other, J-SMA inferred with respect a certain output do not affect in any way implication for other outputs, as opposed with computation of fault probabilities.

Inferred approximation conditions are later applied to each over-approximate fault in order to compute probability of unmasked input vectors, with the exception of $PI2 \rightarrow n7/1$ which has already been approximated. Implication is performed in an output by output basis as usual. In addition, input vectors which were unmasked with previous approximation have to be properly discounted for each fault. But there are some over-approximate faults which are not affected in this iteration. Both $PI1/1$ and $PI4/1$ are not compatible with approximation conditions of fault $n8 \rightarrow n9/1$, so their implications fail in finding a valid J-SMA and their probabilities remain unchanged.

$PI0/1$ is one of the faults affected by $n8 \rightarrow n9/1$ approximation. This fault only can propagate to output $PO0$, and therefore its set of assignments -which can be seen in Figure A.40a- has to be intersected just with $A_{n8 \rightarrow n9/1_0}$, the approximation condition of fault $n8 \rightarrow n9/1$ corresponding to output $PO0$. This generates the additional MAs $PI1=1$ and $PI3=1$. The SMA is then further implied. $PI2=1$ and $PI3=1$ imply $n8=0$,

Figure A.40: Effect of fault $n8 \rightarrow n9/1$ over PI0/1

which allows justifying assignment $n9=1$ and at the same time implies $n11=1$, as it can be seen in Figure A.40b. In the end, J-SMA $J_{PI0/1}^2 = \overline{PI0} \cdot PI1 \cdot PI2 \cdot PI3$ is obtained, which corresponds a probability of 0.0625. This value is deducted from the original probability of this fault, so it has a new value of 0.125. Intersection with $A_{n8 \rightarrow n9/1_1}$ is null, because PI0/1 does not propagate to that output. Therefore, there is no need of applying approximation condition corresponding to output PO1.

Figure A.41: Effect of fault $n8 \rightarrow n9/1$ over itself

Next fault $n8 \rightarrow n9/1$ comes, which is the approximation candidate itself. First it is re-implied with respect to output PO0. Figure A.41a shows the assignments already inferred for this fault during initial implication. It can be seen that J-SMA coincides with approximation condition $A_{n8 \rightarrow n9/1_0}$, and therefore its intersection does not alter the set of assignments. As result, J-SMA $J_{n8 \rightarrow n9/1_0}^2 = \overline{PI0} \cdot PI1 \cdot PI2 \cdot PI3$ is obtained, which corresponds a probability value of 0.0625. Input vectors included in $J_{n8 \rightarrow n9/1_0}^2$ have to be discounted when implying the same fault with respect to output PO1. The result can be seen in Figure A.41b, where the additional condition $\overline{J_{n8 \rightarrow n9/1_0}^2}$ implies the MA $PI0=1$, cause of PI1, PI2 and PI3 being already assigned

to value 1. This set of assignments makes the intersection with $A_{n8 \rightarrow n9/1_1}$ redundant, so no additional MA are inferred. The J-SMA with respect to output PO1 is then $J_{n8 \rightarrow n9/1_1}^2 = PI0 \cdot PI1 \cdot PI2 \cdot PI3$. This justification frontier presents a probability of 0.0625 too. The sum of both partial probabilities equals the original fault probability, so it drops to 0, reflecting the fact that fault $n8 \rightarrow n9/1$ is going to be approximated.

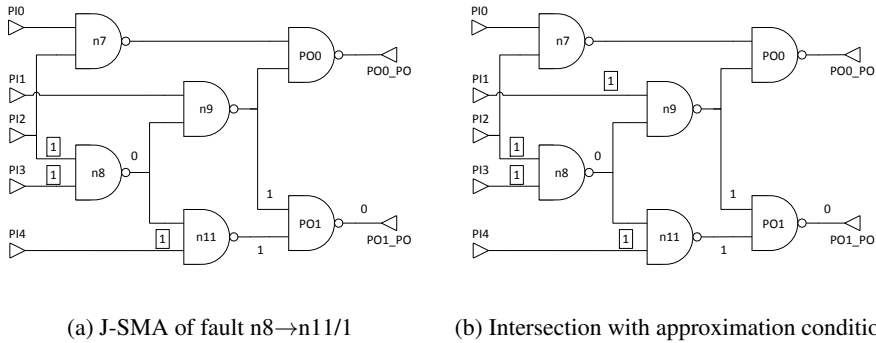


Figure A.42: Effect of fault $n8 \rightarrow n9/1$ over $n8 \rightarrow n11/1$

Then fault $n8 \rightarrow n11/1$ is re-implied. Figure A.42a shows the set of assignments previously inferred during initial implication. This fault only can propagate through output PO1, and because of that it has to be intersected just with approximation condition $A_{n8 \rightarrow n9/1_1}$. As a result, new MA $PI1=1$ appears, although no additional assignments can be inferred. J-SMA $J_{n8 \rightarrow n11/1}^2 = PI1 \cdot PI2 \cdot PI3 \cdot PI4$ is obtained. This justification frontier has associated a probability equal to 0.0625, which is subtracted from former value resulting in a probability of 0.0625 for fault $n8 \rightarrow n11/1$.

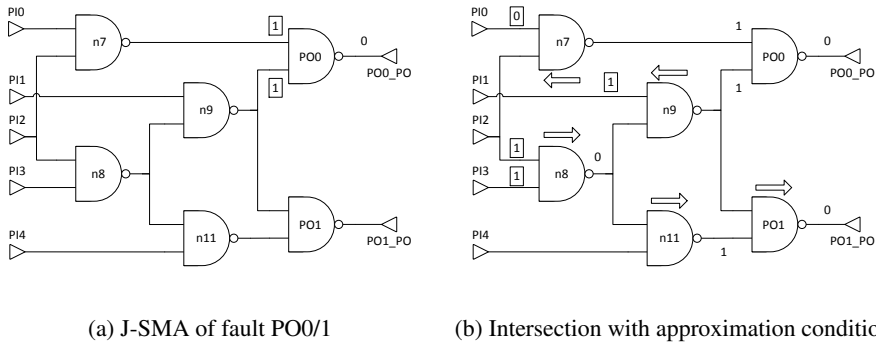
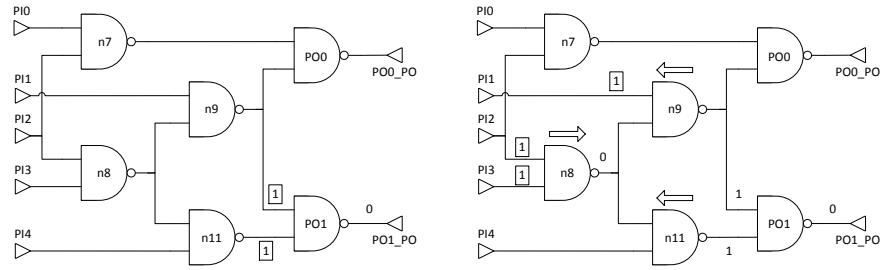


Figure A.43: Effect of fault $n8 \rightarrow n9/1$ over PO0/1

Later effect of approximation over fault PO0/1 is analysed. Figure A.43a shows again the set of assignments inferred for this fault, which are then intersected with $A_{n8 \rightarrow n9/1_0}$, the approximation condition with respect to output PO0, which is the only

one that current fault can be propagated through. As a result, additional MAs $PI0=0$, $PI1=1$, $PI2=1$ and $PI3=1$ are discovered, which are subsequently propagated. MA $n7=1$ is justified by $PI0=0$, while the combination of $PI2=1$ and $PI3=1$ implies $n8=0$, which in turn makes $n9=1$ to become justified. The whole implication process is shown in Figure A.43b. Finally, J-SMA $J_{PO0/1}^2 = \overline{PI0} \cdot PI1 \cdot PI2 \cdot PI3$ is obtained. This justification frontier has a probability of 0.0625, which reduces probability of fault $PO0/1$ to 0.2813. It must be noted that, although this fault was re-implicit in the first iteration as well, J-SMA obtained during that implication, $J_{PO0/1}^1 = PI0 \cdot \overline{PI1} \cdot \overline{PI2}$, does not have to be explicitly excluded in this iteration. As both justification frontiers are referred to the same output, they are mutually exclusive by construction.



(a) J-SMA of fault PO1/1

(b) Intersection with approximation condition

Figure A.44: Effect of fault $n8 \rightarrow n9/1$ over $PO1/1$

Finally, fault $PO1/1$ is re-implicit. The set of assignments deduced during initial step of the algorithm is shown in Figure A.44a, over which approximation condition $A_{n8 \rightarrow n9/1}$ is applied because fault $PO1/1$ only propagates through output $PO1$. This causes the appearance of MAs $PI1=1$, $PI2=1$ and $PI3=1$, which are further propagated. In particular, $PI2=1$ and $PI3=1$ imply $n8=0$, which justifies both mandatory assignments $n9=1$ and $n11=1$ as shown in Figure A.44b. At the end of implication, justification frontier $J_{PO1/1}^2 = PI1 \cdot PI2 \cdot PI3$ is deduced, corresponding a probability of 0.125. This value is deducted from original probability value of the fault, resulting in 0.2656.

Fault	Justification frontier		Former	Probability Unmasked		Updated
	PO0	PO1		PO0	PO1	
$PI0/1$	$\overline{PI0} PI1 PI2 PI3$	-	0.1875	0.0625	0	0.125
$PI1/1$	-	-	0.1875	0	0	0.1875
$PI4/1$	-	-	0.1875	0	0	0.1875
$n8 \rightarrow n9/1$	$\overline{PI0} PI1 PI2 PI3$	$PI0 PI1 PI2 PI3$	0.125	0.0625	0.0625	0
$n8 \rightarrow n11/1$	-	$PI1 PI2 PI3 PI4$	0.125	0	0.0625	0.0625
$PO0/1$	$\overline{PI0} PI1 PI2 PI3$	-	0.3438	0.0625	0	0.2813
$PO1/1$	-	$PI1 PI2 PI3$	0.3906	0	0.125	0.2656

Table A.5: Summary of $n8 \rightarrow n9/1$ approximation

Table A.5 collects the results of implication for all involved faults. It contains,

from left to right, the justification frontiers resulting from re-implying each fault with the approximation conditions of fault $n8 \rightarrow n9/1$ with respect to both outputs, the fault probability at the beginning of the iteration, the probability of unmasked input vectors for each fault with respect to each output, and finally the updated probability values. Unmasked probability values are obtained from the J-SMAs inferred during current iteration, which are in turn deducted from previous values in order to update fault probabilities. At the same time, EP is updated by adding the probability of newly unmasked input vectors corresponding to each fault, in the following way

$$EP_2 = EP_1 + \frac{\sum_i P(f_i \cap A_{n8 \rightarrow n9/1} \cap \overline{f_i \cap A_H})}{n} =$$

$$= 2.3438\% + \frac{(5 \cdot 0.0625) + 0.125}{16} = 5.0481\%$$

where $f_i \cap A_H$ represents the input vectors already unmasked with previous approximated faults - $PI2 \rightarrow n7/1$ in this case. If a target EP lower than 5.0481% had been set, $n8 \rightarrow n9/1$ would have been the last approximated fault, and the approximation algorithm would have finished.

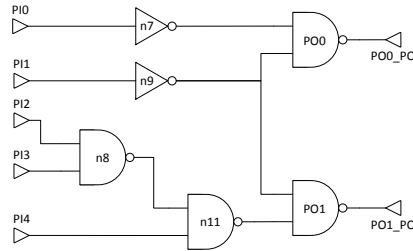


Figure A.45: Approximation of fault $n8 \rightarrow n9/1$

After probability updating, fault $n8 \rightarrow n9/1$ can be finally approximated. This is performed by replacing the correspondent line with a logic constant 1 in the over-approximate circuit. As result, node $n9$ has one of its inputs tied to the sensitizing value, and therefore this node can be replaced with a NOT gate as shown in Figure A.45. This circuit is now the over-approximation of $c17$, while the under-approximation is still an exact copy of it. Fault $n8 \rightarrow n9/1$ is excluded from implication process and candidate selection from now.

Again, all the remaining over-approximate faults are implied in this new over-approximation in order to detect additional faults which become redundant. In this iteration no one is detected, and therefore not additional faults become approximated.

Next candidate is selected among the faults which have not been approximated yet. Table A.6 shows the current probability of every fault. According with the proposed criteria, over-approximate fault $n8 \rightarrow n11/1$ is the next approximation candidate because it has the minimum probability value, 0.0625. With this, the next iteration of the algorithm starts.

Under-approx fault	Probability	Over-approx fault	Probability
PI2→n8/1	0.1875	PI0/0	0.125
PI3/1	0.25	PI1/1	0.1875
n7/1	0.1875	PI2→n7/1	-
n9→PO0/1	0.2812	PI4/1	0.1875
n9→PO1/1	0.1875	n8→n9/1	-
n11/1	0.1875	n8→n11/1	0.0625
PO0/0	0.5312	PO0/1	0.2813
PO1/0	0.6094	PO1/1	0.2656

Table A.6: Fault probabilities after n8→n9/1 approximation

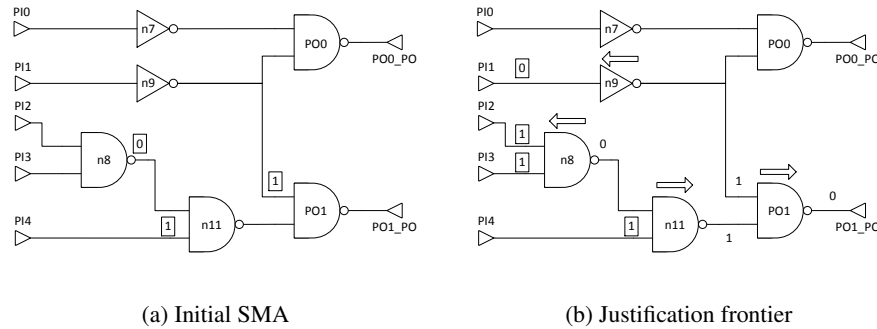
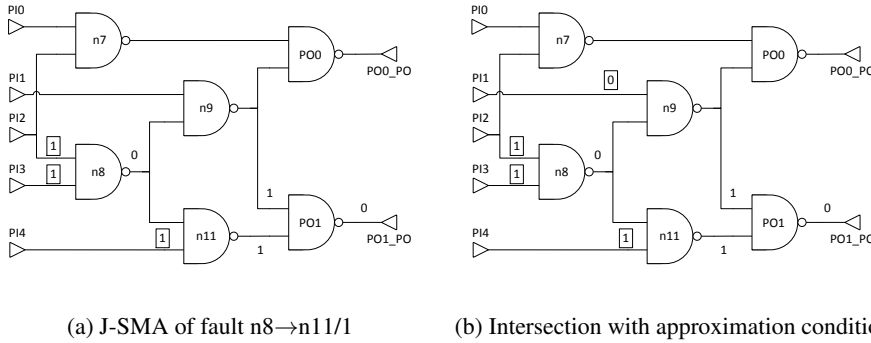
3rd approximation: n8→n11/1

Figure A.46: Approximation condition of n8→n11/1

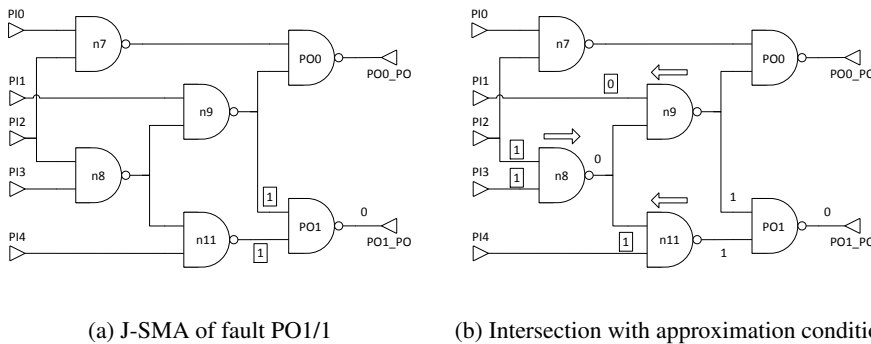
First step in the new iteration is to find the approximation condition of the new candidate fault, n8→n11/1. This fault only propagates through output PO1, so with respect to PO0 its approximation condition is directly null. $A_{n8 \rightarrow n11/1}$ is obtained by implying the fault in the over-approximate circuit generated during previous iteration of the algorithm. The initial SMA can be seen in Figure A.46a, which consists in controllability condition n8=0 and observability conditions PI4=1 and n9=1. These are propagated through the circuit, resulting in assignments n8=0 and n9=1 being justified by PI1=0, PI2=1 and PI3=1 as shown in Figure A.46b. When implication ends, current J-SMA becomes the approximation condition of candidate fault, $A_{n8 \rightarrow n11/1} = \overline{PI1} \cdot PI2 \cdot PI3 \cdot PI4$. This J-SMA is then intersected with every remaining over-approximate fault in order to compute which input vectors are left unmasked due to n8→n11/1 approximation. This procedure reveals that faults PI0/1, PI1/1, PI4/1 and PO0/1 either are not compatible with $A_{n8 \rightarrow n11/1}$ or they do not propagate through the same output as n8→n11/1. For these faults, implication infers no valid J-SMA and their probabilities remain without changes. Only faults n8→n11/1 and PO1/1 are affected, which is following discussed.



(a) J-SMA of fault $n8 \rightarrow n11/1$ (b) Intersection with approximation condition

Figure A.47: Effect of fault $n8 \rightarrow n11/1$ over itself

At first, fault $n8 \rightarrow n11/1$ is implied, which corresponds with the fault selected for approximation. The set of assignments resulting from implication of this fault can be seen in Figure A.47a, and over it the approximation condition $A_{n8 \rightarrow n11/1}$ is applied. This intersection generates the MA $PI1=0$ as shown in Figure A.47b, but it does not allow deducing additional assignments. J-SMA $J_{n8 \rightarrow n11/1}^3 = \overline{PI1} \cdot PI2 \cdot PI3 \cdot PI4$ is extracted, which has associated a probability of 0.0625. When subtracted to previous value of fault $n8 \rightarrow n11/1$, this drops to 0, indicating that the fault has been completely unmasked.



(a) J-SMA of fault $PO1/1$ (b) Intersection with approximation condition

Figure A.48: Effect of fault $n8 \rightarrow n11/1$ over $PO1/1$

The other affected fault is $PO1/1$. Figure A.48a shows the set of assignments inferred during implication of this fault, which approximation condition $A_{n8 \rightarrow n11/1}$ is intersected with. As result, MAs $PI1=0$, $PI2=1$, $PI3=1$ and $PI4=1$ are applied and subsequently propagated. Assignment $n9=1$ is justified by $PI1=0$, while $PI2=1$ and $PI3=1$ imply $n8=0$, which further justifies $n11=1$. All this process is represented in Figure A.48b. At the end of implication process, justification frontier $J_{PO1/1}^3 = \overline{PI1} \cdot PI2 \cdot PI3 \cdot PI4$ is inferred, which again implies a probability of 0.0625. This value is deducted from probability of fault $PO1/1$, resulting in 0.2031.

Fault	Justification frontier (PO1)	Probability		
		Former	Unmasked	Updated
PI0/1	-	0.125	0	0.125
PI1/1	-	0.1875	0	0.1875
PI4/1	-	0.1875	0	0.1875
n8→n11/1	$\overline{PI1} PI2 PI3 PI4$	0.0625	0.0625	0
PO0/1	-	0.2813	0	0.2813
PO1/1	$\overline{PI1} PI2 PI3 PI4$	0.2656	0.0625	0.2031

Table A.7: Summary of n8→n11/1 approximation

Although these faults were already implied in the second iteration of the algorithm it is not required to explicitly exclude input vectors detected then and contained in $J_{n8 \rightarrow n11}^2$ and $J_{PO1/1}^2$ respectively. Because these both faults can propagate just to one output, justification frontiers corresponding to different approximated faults are disjoint by construction.

Table A.7 summarizes updating of fault probabilities in the current iteration. From left to right, it contains the J-SMA resulting from the intersection of each over-approximate fault with $A_{n8 \rightarrow n11/1}$, fault probabilities at the end of previous iteration, probability of justification frontiers lastly inferred, and finally the updated fault probabilities. Based on these results EP is increased in the following way

$$\begin{aligned}
 EP_3 &= EP_2 + \frac{\sum_i P(f_i \cap A_{n8 \rightarrow n11/1} \cap \overline{f_i} \cap A_H)}{n} = \\
 &= 5.0481\% + \frac{2 \cdot 0.0625}{16} = 5.8594\%
 \end{aligned}$$

At this point, any EP target below the 5.8594% threshold would force the algorithm to stop, being n8→n11/1 the last approximated fault. The term $f_i \cap A_H$ represents the unmasked input vectors for previous approximated faults, both PI2→n7/1 and n8→n9/1.

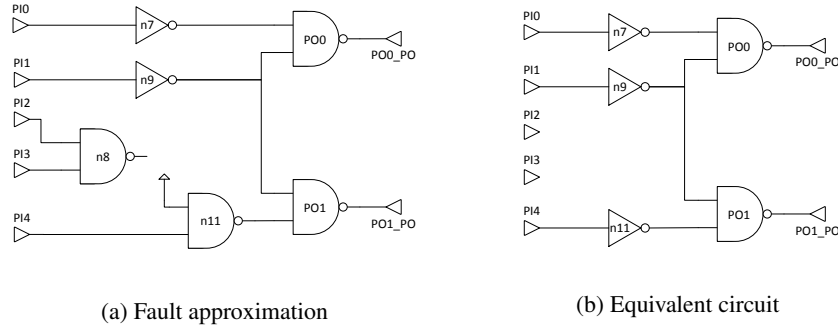


Figure A.49: Approximation of fault n8→n11/1

Now candidate fault n8→n11/1 can be finally approximated. In the over-approximate circuit, line from node n8 to n11 is replaced with a connection with VCC as shown in

Under-approx fault	Probability	Over-approx fault	Probability
PI2→n8/1	0.1875	PI0/0	0.125
PI3/1	0.25	PI1/1	0.1875
n7/1	0.1875	PI2→n7/1	-
n9→PO0/1	0.2812	PI4/1	0.1875
n9→PO1/1	0.1875	n8→n9/1	-
n11/1	0.1875	n8→n11/1	-
PO0/0	0.5312	PO0/1	0.2813
PO1/0	0.6094	PO1/1	0.2031

Table A.8: Fault probabilities after n8→n11/1 approximation

Figure A.49a, which has two effects. On one hand, node n8 becomes a dangling node, and therefore it can be removed without affecting circuit functionality. On the other hand, one of the inputs of node n11 is tied to a logic constant, so it can be simplified, replacing the NAND gate with an inverter. After performing proper simplifications, circuit of Figure A.49b is obtained, which is used at this point as the over-approximation of c17 benchmark. No faults have been yet approximated in the other instance, so the under-approximate circuit is still an exact copy of target design. Again, the remaining over-approximate faults are implied in the new over-approximation in order to detect redundant faults. And again, no additional faults become approximated.

After approximating fault n8→n11/1, the next candidate is selected. Current fault probabilities are collected in Table A.8. Excluding already approximated faults, the one with the lowest probability is fault PI0/1 with 0.125. Once again, an over-approximate fault becomes the next approximation candidate. Then following iteration of the algorithm begins.

4th approximation: PI0/1

A new iteration starts with the implication in the correspondent approximate circuit of the selected fault PI0/1 in order to obtain its approximation condition. Assignments PI0=0 and n9=1 are respectively the controllability and observability condition, and they form the initial SMA as depicted in Figure A.50a. These assignments are then propagated by direct implication as shown in Figure A.50b. J-SMA $A_{PI0} = \overline{PI0} \cdot \overline{PI1}$ is finally inferred, which becomes the approximation condition of fault PI0/1. This has been computed with respect to PO0, and therefore those faults which do not propagate through that output are not affected by PI0/1 approximation, in particular PI4/1 and PO1/1.

First fault re-implied is PI0/1, the approximation candidate. Figure A.51a shows the assignments inferred from its implication, which are intersected with approximation condition $A_{PI0} = \overline{PI0} \cdot \overline{PI1}$. Cause of this, additional MA PI1=0 appears, which justifies assignment n9=1 as shown in Figure A.51b. At the end of implication process J-SMA $J_{PI0/1}^4 = \overline{PI0} \cdot \overline{PI1} \cdot PI2$ is inferred, which corresponds a probability of 0.125. This makes that PI0/1 probability drops to 0.

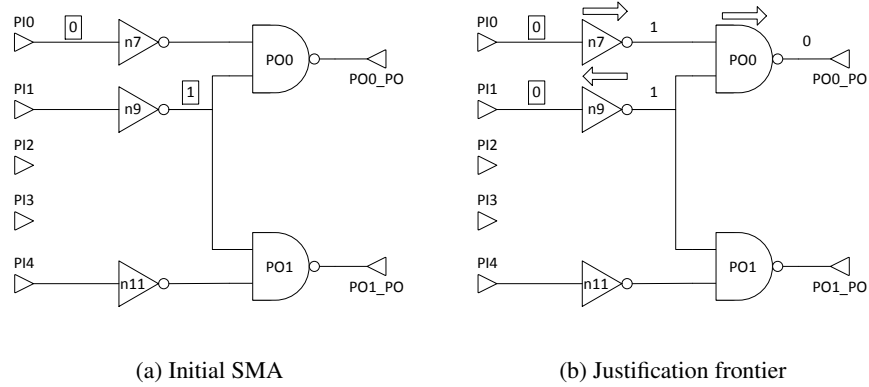


Figure A.50: Approximation condition of PI0/1

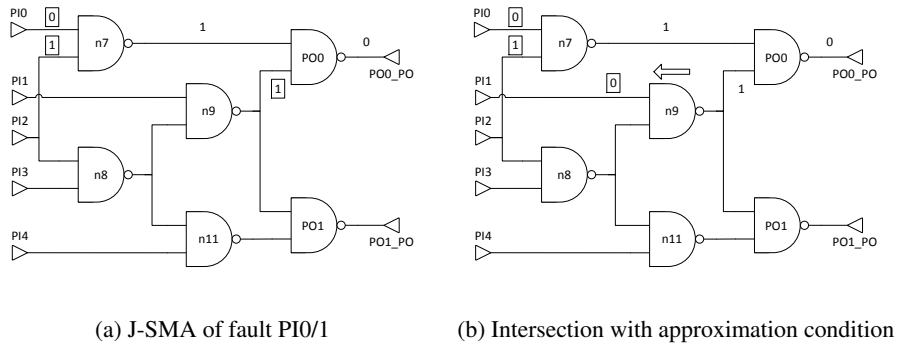


Figure A.51: Effect of fault PI0/1 over itself

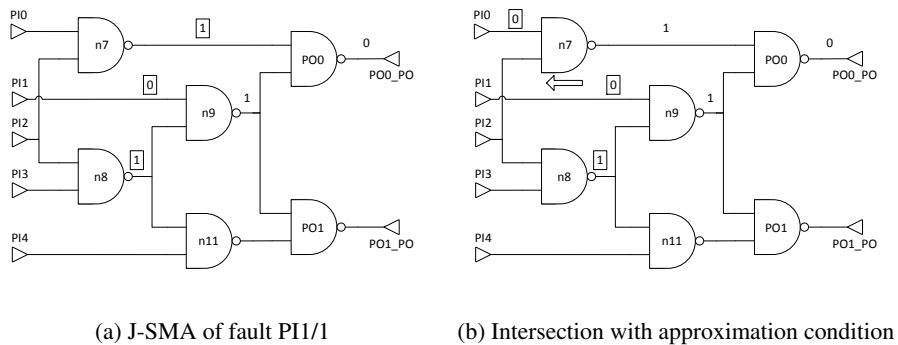
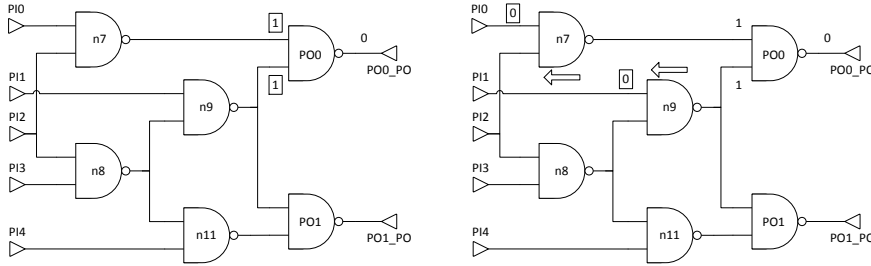


Figure A.52: Effect of fault PI0/1 over PI1/1

Fault	Justification frontier (PO0)	Probability		
		Former	Unmasked	Updated
PI0/1	$\overline{PI0} \overline{PI1} PI2$	0.125	0.125	0
PI1/1	$\overline{PI0} \overline{PI1} n8$	0.1875	0.1875	0
PI4/1	-	0.1875	0	0.1875
PO0/1	$\overline{PI0} \overline{PI1}$	0.2813	0.25	0.0313
PO1/1	-	0.2031	0	0.2031

Table A.9: Summary of PI0/1 approximation

Next fault PI1/1 comes. Although it can propagate through both outputs, just PO0 is considered here because is the only output affected by the current candidate fault. The set of assignments resulting from implication of PI1/1 with respect to PO0 are represented in Figure A.52a. Approximation condition A_{PI0} is applied over this, generating the additional mandatory assignment PI0=0. This new MA justifies n7=1 as shown in Figure A.52b. No more assignments can be deduced, so J-SMA $J_{PI1/1}^A = \overline{PI0} \cdot \overline{PI1} \cdot n8$ is extracted. By using simple COP, a probability equal to 0.1875 is inferred from this implication. Such value is deducted from current probability of fault PI1/1, which makes a total of 0. In this case, that not necessarily means the fault has been completely approximated. Faults are still considered until they are marked as approximated, even if they have a null or negative probability value.



(a) J-SMA of fault PO0/1

(b) Intersection with approximation condition

Figure A.53: Effect of fault PI0/1 over PO0/1

Finally, fault PO0/1 is updated. Departing from the set of assignments inferred during implication of this fault - see Figure A.53a -, intersection with A_{PI0} is performed. This generates the assignments PI0=0 and PI1=0, which allows justifying n7=1 and n9=1 respectively as it is indicated in Figure A.53b. In the end, justification frontier $J_{PO0/1}^A = \overline{PI0} \cdot \overline{PI1}$ is obtained, which corresponds a probability of 0.25. This result is subtracted from PO0/1 probability, which diminishes to 0.0313.

Table A.9 collects the computed justification frontiers and probability changes during this iteration, as usual. And once all fault probabilities have been updated, global EP is increased in an amount proportional to the sum of all computed probability vari-

ations as follows

$$\begin{aligned}
 EP_4 &= EP_3 + \frac{\sum_i P(f_i \cap A_{PI0/1} \cap \overline{f_i \cap A_H})}{n} = \\
 &= 5.8594\% + \frac{0.125 + 0.1875 + 0.25}{16} = 9.3750\%
 \end{aligned}$$

Estimate EP is now equal to 9.3750%. If target EP had been set in a value lower than this, current approximation candidate would have been the last approximate fault, and the algorithm would have finished.

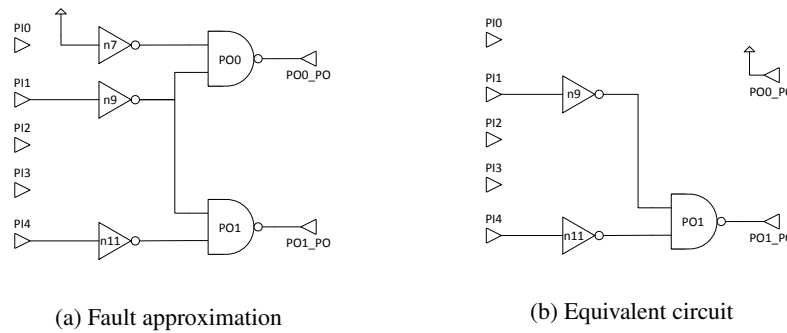


Figure A.54: Approximation of fault PI0/1

Finally fault PO0/1 can be approximated and marked. This is an over-approximate fault, and therefore it is approximated in the correspondent instance. Input PI0 is replaced with a logic constant 1 as shown in Figure A.54a. This can be propagated to $n7=0$ due to the inverter n7, which in turn forces output PO0 to be tied to 1. Therefore PO0 can be substituted by a logic constant, thus simplifying the over-approximation as shown in Figure A.54b. Meanwhile, under-approximate circuit is still a replica of c17 benchmark. Later, the remaining over-approximate faults are re-implied in the over-approximation in order to detect additional faults which has been approximated. For the first time, this step becomes relevant because, as it can be seen in Figure A.54b, fault PO0/1 cannot longer propagate in the over-approximate circuit. In fact, implication of this fault reveals that it is now redundant, and therefore it is marked as an approximated fault in addition to PI0/1, even though current probability of fault PO0/1 is not zero. This fact only means that initial fault probability was overestimated.

Table A.10 contains the current value of all fault probabilities, both from under and over-approximate circuits. Among them the next approximated fault is selected, excluding those faults which has already been identified as approximated, PO0/1 included. According to the current values, over-approximate fault PI1/1 is selected because it is the one with the lowest probability. Another iteration is then performed as usual, even though the selected fault has a probability equal to 0.

Under-approx fault	Probability	Over-approx fault	Probability
PI2→n8/1	0.1875	PI0/0	-
PI3/1	0.25	PI1/1	0
n7/1	0.1875	PI2→n7/1	-
n9→PO0/1	0.2812	PI4/1	0.1875
n9→PO1/1	0.1875	n8→n9/1	-
n11/1	0.1875	n8→n11/1	-
PO0/0	0.5312	PO0/1	-
PO1/0	0.6094	PO1/1	0.2031

Table A.10: Fault probabilities after PI0/1 approximation

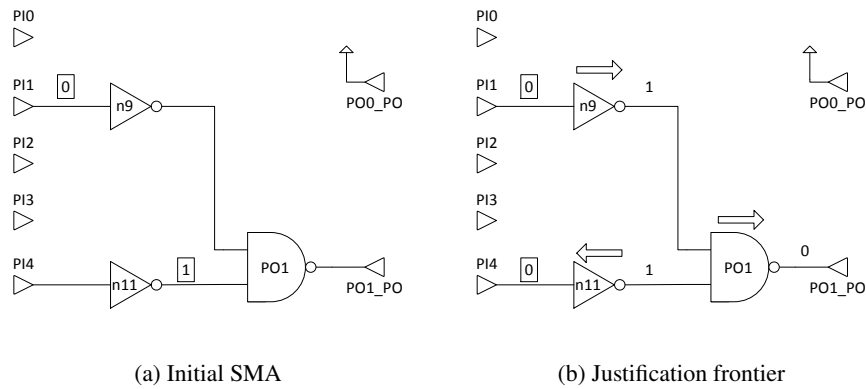
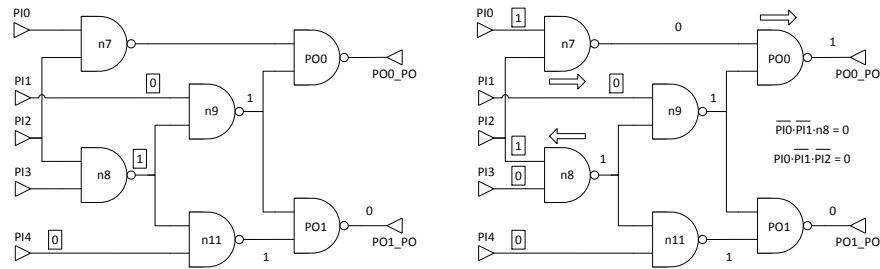


Figure A.55: Approximation condition of PI1/1

5th approximation: PI1/1

Once a new approximation candidate has been selected, it is implied in the correspondent approximate circuit with the goal of obtaining the approximation conditions of that fault, as usual. In theory, one implication has to be performed with respect to each output but, in the current state of the over-approximate circuit, output PO0 is tied to a logic constant - see Figure A.54b. Therefore, there is no valid set of conditions that allow propagation of fault PI1/1 thorough PO0. On the other hand, candidate fault has to be implied with respect to output PO1. Controllability condition $PI1=0$ and observability condition $n11=1$ form the initial SMA for this fault as shown in Figure A.55a. $PI1=0$ is propagated to n9 and subsequently to PO1, while n11 is justified by $PI4=0$. All this process is represented in Figure A.55b. Finally, approximation condition $A_{PI1/1} = \overline{PI1} \cdot \overline{PI4}$ is obtained. It must be noted that this is referred just to output PO1. Then all remaining over-approximate faults are re-implied with respect to PO1, including $A_{PI1/1}$ as an additional condition.



(a) J-SMA of fault PI1/1

(b) Intersection with approximation condition

Figure A.56: Effect of fault PI1/1 over itself

First fault is the approximation candidate, PI1/1. This is implied again with respect to output PO1, generating the set of assignments which appear in Figure A.56a. Then approximation condition $A_{PI1/1}$ is applied, but this is compatible with the set of assignments of fault PI1/1 and therefore it cannot infer additional MAs. But this fault has been already implied in previous iterations, resulting in justification frontiers $J_{PI1/1}^1 = PI0 \cdot \overline{PI1} \cdot \overline{PI2}$ and $J_{PI1/1}^4 = \overline{PI0} \cdot \overline{PI1} \cdot n8$. These previous J-SMAs have been computed with respect to PO0 and therefore they have to be explicitly excluded now, because they correspond to a different output. Condition $\overline{J_{PI1/1}^4}$ implies the MA $PI0=1$, because PI1 and n8 have been already set to 0 and 1 respectively. Thanks to this new MA, $PI2=1$ can be deduced cause of $\overline{J_{PI1/1}^1}$ condition, which in turn causes MA $PI3=0$ to appear as the only way of justifying assignment $n8=1$. In addition, $PI0=1$ and $PI2=1$ are propagated to n7 and PO0. All this process is shown in Figure A.56b, resulting in the J-SMA $J_{PI1/1}^5 = PO0 \cdot \overline{PI1} \cdot PO2 \cdot \overline{PI3} \cdot \overline{PI4}$. Probability associated to this justification frontier is approximately 0.0313, and therefore probability of fault PI1/1 is updated to a value of -0.0313. This means that fault probability were initially underestimated.

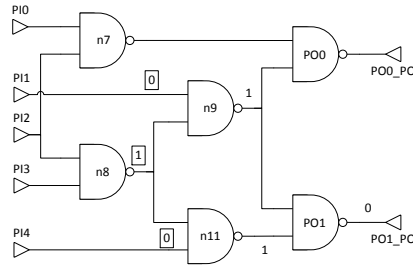


Figure A.57: Effect of fault PI1/1 over PI4/1

Then fault PI4/1 is processed. Figure A.57 shows the result of the implication of this fault, already computed during the initial probability computation step. Approximation condition $A_{PI1/1} = \overline{PI1} \cdot \overline{PI4}$ is then intersected with this set of assignments. But these two sets completely overlap, and therefore no additional assignments can be inferred. Justification frontier $J_{PI4/1}^5 = \overline{PI1} \cdot \overline{PI4} \cdot n8$ is obtained, with a probability of 0.1875. This makes that PI4/1 probability drops to zero when updated.

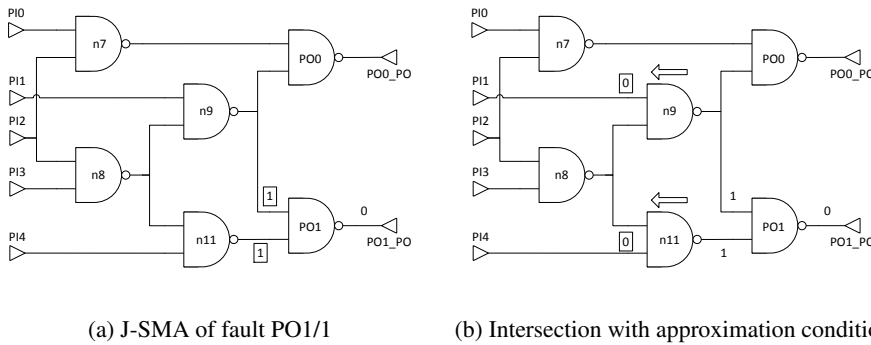


Figure A.58: Effect of fault PI1/1 over PO1/1

Finally, fault PO1/1 is re-implied. The set of assignments inferred by implicating this fault can be seen in Figure A.58a. Over this, approximation condition $A_{PI1/1}$ is applied, generating the additional MAs PI1=0 and PI4=0. These new mandatory assignments justify both n9=1 and n11=1, which are removed from the SMA as shown in Figure A.58b. In the end, justification frontier $J_{PO1/1}^5 = \overline{PI1} \cdot \overline{PI4}$ is deduced, which has associated a probability of 0.25. This value is deducted from PO1/1 probability, resulting in an updated value of -0.0469, which indicates that initial probability was underestimated.

The results of probability updating step for PI1/1 approximation are summarized in Table A.11, including the inferred J-SMAs. From here, total EP is increased by the probability of intersections with $A_{PI1/1}$ computed in the current iteration, weighted

Fault	Justification frontier (PO1)	Probability		
		Former	Unmasked	Updated
PI1/1	$\overline{PI0} \overline{PI1} \overline{PI2} \overline{PI3} \overline{PI4}$	0	0.0313	-0.0313
PI4/1	$\overline{PI1} \overline{PI4} n8$	0.1875	0.1875	0
PO1/1	$\overline{PI1} \overline{PI4}$	0.2031	0.25	-0.0469

Table A.11: Summary of PI1/1 approximation

with the size of the fault list as follows

$$\begin{aligned}
 EP_5 &= EP_4 + \frac{\sum_i P(f_i \cap A_{PI1/1} \cap \overline{f_i} \cap \overline{A_H})}{n} = \\
 &= 9.3750\% + \frac{0.0313 + 0.1875 + 0.25}{16} = 12.3047\%
 \end{aligned}$$

The new estimated EP is 12.3047%. Any target EP under this value would cause the algorithm to finish right after approximation of fault PI1/1.

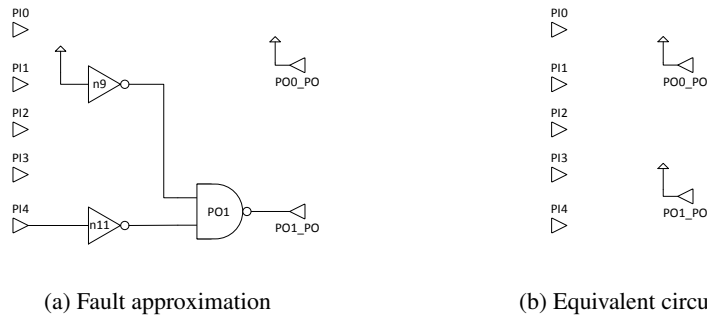


Figure A.59: Approximation of fault PI1/1

After all relevant faults have been re-implied and their probabilities have been updated, fault PI1/1 can be finally approximated. Input PI1 is substituted with a logic constant 1 as it can be seen in Figure A.59a. This assignment can be propagated to $n9=0$ due to the inverter n9, and finally to $PO1=1$ through node PO1. In other words, this logic transformation is equivalent to tie PO1 to logic 1, resulting in the trivial over-approximate circuit as shown in Figure A.59b. The under-approximation is still an exact copy of target circuit. Now all primary outputs are tied to logic constants there is no valid propagation path for any of the faults in the over-approximation. Implication of remaining over-approximate faults PI4/1 and PO1/1 indicates that both are redundant, which means that they have been approximated too. At this point, all over-approximate faults have become approximated.

The next candidate is selected as usual. Now all over-approximate faults have been approximated, only under-approximate ones can be chosen. Their probabilities are listed in the Table A.12. According with these values, there are four possible candidates with the minimum probability value: $PI2 \rightarrow n8/1$, $n7/1$, $n9 \rightarrow PO1/1$ and $n11$.

Under-approx fault	Probability	Over-approx fault	Probability
PI2→n8/1	0.1875	PI0/0	-
PI3/1	0.25	PI1/1	-
n7/1	0.1875	PI2→n7/1	-
n9→PO0/1	0.2812	PI4/1	-
n9→PO1/1	0.1875	n8→n9/1	-
n11/1	0.1875	n8→n11/1	-
PO0/0	0.5312	PO0/1	-
PO1/0	0.6094	PO1/1	-

Table A.12: Fault probabilities after PI1/1 approximation

Among them, the fault which produces the largest area savings is selected. This is measured as the number of lines removed if the correspondent fault would be approximated. PI2→n8/1 only affects one line before reaching a primary input, n9→PO1/1 only removes one line too because node n9 is a multiple fanout point but approximation of both faults n7/1 and n11/1 affects up to 3 lines. Any of these two faults can be arbitrarily chosen. Let us assume that n7/1 is the next approximated fault.

6th approximation: n7/1

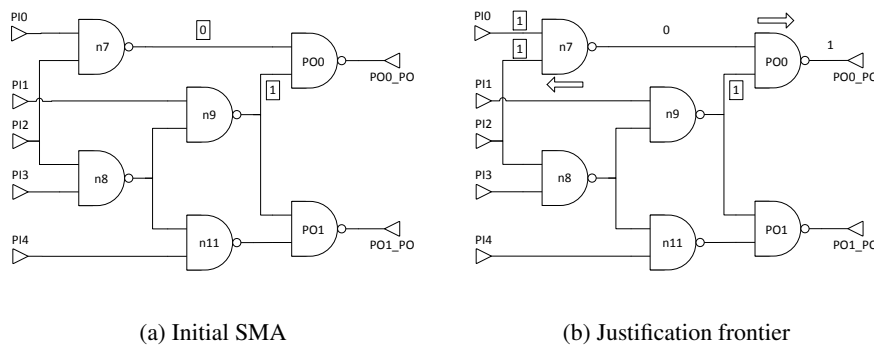


Figure A.60: Approximation condition of n7/1

First step in a new iteration consist in obtaining the approximation conditions of selected fault as usual. Being n7/1 an under-approximate fault, it is implied in the correspondent approximate circuit. There are no faults approximated in this instance yet, so the under-approximation is an exact copy of c17 benchmark. The initial set of mandatory assignments is formed with n7=0 and n9=1 as it can be seen in Figure A.60a which are subsequently propagated through the circuit. Assignment n7=0 implies PO0=1 while at the same time it is justified by PI0=1 and PI2=1 as shown in Figure A.60b. No more assignments can be deduced and J-SMA $A_{n7/1} = PI0 \cdot PI2 \cdot n9$ is

extracted, which becomes the approximation condition of fault $n7/1$. It must be noted that this fault propagates just to output PO0 and consequently $A_{n7/1}$ is referred to that output, being the approximation condition with respect to PO1 null. Now all under-approximate faults are re-implied with the additional condition $A_{n7/1}$. But not all faults are affected by $n7/1$ approximation. $n9 \rightarrow PO1/1$, $n11/1$ and $PO1/0$ do not propagate through the same output as $n7/1$, and faults $PI2 \rightarrow n8/1$, $PI3/1$ and $n9 \rightarrow PO0/1$ are not compatible with $A_{n7/1}$. For all these faults the result of implication is the null set, and therefore their probabilities remain unchanged.

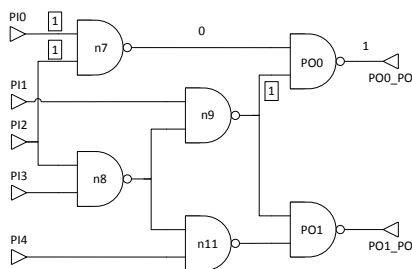


Figure A.61: Effect of fault $n7/1$ over itself

Among the faults affected by $n7/1$ approximation, the candidate fault itself is the first one. Figure A.61 shows the set of assignments inferred when implicating this fault, which is then intersected with $A_{n7/1}$. But the approximation condition of $n7/1$ completely overlaps with it, deducing no additional MAs. Justification frontier $J_{n7/1}^6 = PI0 \cdot PI2 \cdot n9$ is obtained. Probability of this J-SMA is then computed based on the assignments it contains. Probability of each assignment is conditioned to the whole set of inferred assignments, thus having

$$P(J_{n7/1}^6) = P(PI0) \cdot P(PI2) \cdot P(n9|PI2) = 0.5 \cdot 0.5 \cdot 0.75 = 0.1875$$

This is the same value as the one computed in the algorithm initialization. Therefore, updated value of fault $n7/1$ is 0.

$PO0/0$ is the other fault affected by $n7/1$ approximation. This is re-implied by intersecting approximation condition $A_{n7/1} = PI0 \cdot PI2 \cdot n9$ with $PO0=1$, the unique assignment that can be inferred from implication of fault $PO0/0$. $PO0=1$ is justified by $n7=0$, the only possibility because $n9$ is already set to 1. At the same time, $n7=0$ is justified by $PI0=1$ and $PI2=1$. This implication process is shown in Figure A.62. Finally, J-SMA $J_{PO0/0}^6 = PI0 \cdot PI2 \cdot n9$ is obtained. This is the same set of conditions than in the case of fault $n7/1$, so the associated probability is the same, 0.1875. This value is subtracted from the initial probability of fault $PO0/0$, resulting in 0.3437.

Table A.13 summarizes the effect over every under-approximate fault of $n7/1$ approximation. It contains, from left to right, the name of each fault, the justification frontier resulting from the intersection with $A_{n7/1}$, the initial probability of each fault, the probability of inferred J-SMA, and finally the updated probability of each fault.

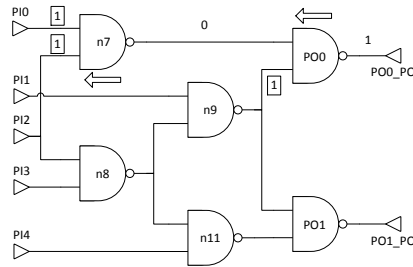


Figure A.62: Effect of fault n7/1 over PO0/0

Fault	Justification frontier (PO0)	Probability		
		Former	Unmasked	Updated
PI2→n8/1	-	0.1875	0	0.1875
PI3/1	-	0.25	0	0.25
n7/1	PI0 PI2 n9	0.1875	0.1875	0
n9→PO0/1	-	0.2812	0	0.2812
n9→PO1/1	-	0.1875	0	0.1875
n11/1	-	0.1875	0	0.1875
PO0/0	PI0 PI2 n9	0.5312	0.1875	0.3437
PO1/0	-	0.6094	0	0.6094

Table A.13: Summary of n7/1 approximation

Once all faults have been re-implied, total EP can be updated too by means of the usual procedure:

$$EP_6 = EP_5 + \frac{\sum_i P(f_i \cap A_{n7/1})}{n} = 12.3047\% + \frac{2 \cdot 0.1875}{16} = 14.6484\%$$

Any target EP below this value would interrupt the approximation algorithm right after n7/1 was approximated.

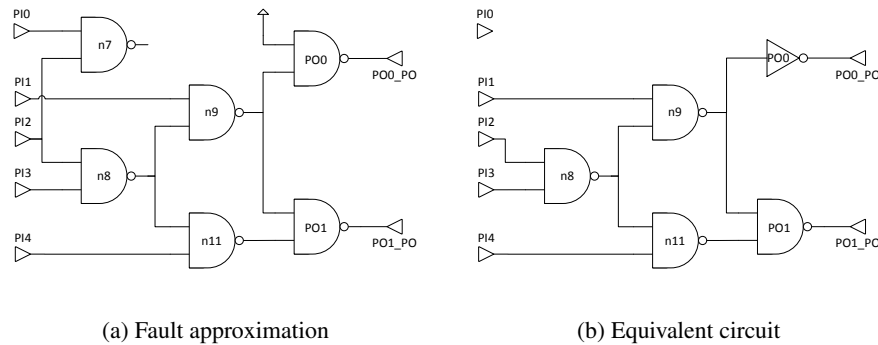


Figure A.63: Approximation of fault n7/1

The selected fault can now be approximated, connecting the line from node n7 to PO0 with VCC as shown in Figure A.63a. This transformation has two effects. On one hand, node n7 becomes dangling and therefore it can be removed without affecting circuit functionality. On the other hand, node PO0 can be simplified cause of having one of its inputs tied to a logic constant, being replaced with an inverter. Figure A.63b shows the final under-approximation after performing proper simplifications. On the other side, the over-approximate circuit is the trivial approximation. Now the rest of under-approximate faults are implied in this new circuit with the goal of detecting additional approximated faults. But none of them becomes redundant, so there are no additional excluded faults.

Finally the next approximated fault is selected among the remaining faults. Table A.14 collects all fault probabilities. According to this values there are three possible candidates: PI2→n8/1, n9→PO1/1 and n11, all of them with the lowest probability value. Among them, n11/1 is finally chosen because is the fault which produces the largest area savings.

7th approximation: n11/1

Once a new approximation candidate has been chosen, it is implied in the correspondent approximate circuit in order to obtain its approximation condition. Controllability condition of n11/1 generates the assignment n11=0, while n9=1 becomes its observability condition. These assignments form the initial SMA as it can be seen in Figure A.64a. Then they are propagated through the circuit. Assignment n11=0 is propagated to PO1=1 and at the same time is justified by PI4=1 and n8=1. Because of this

Under-approx fault	Probability	Over-approx fault	Probability
PI2→n8/1	0.1875	PI0/0	-
PI3/1	0.25	PI1/1	-
n7/1	-	PI2→n7/1	-
n9→PO0/1	0.2812	PI4/1	-
n9→PO1/1	0.1875	n8→n9/1	-
n11/1	0.1875	n8→n11/1	-
PO0/0	0.3437	PO0/1	-
PO1/0	0.6094	PO1/1	-

Table A.14: Fault probabilities after n7/1 approximation

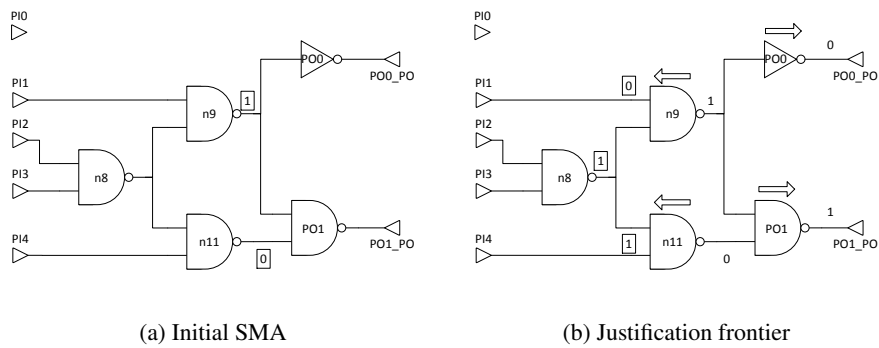


Figure A.64: Approximation condition of n11/1

assignment $PI1=0$ is inferred as the only way of justifying $n9=1$, while this value is propagated to $PO0=0$. All this process is shown in Figure A.64b. No more assignments can be deduced, and approximation condition $A_{n11/1} = \overline{PI1} \cdot PI4 \cdot n8$ is obtained. It must be noted that fault $n11/1$ can only propagate through output $PO1$. Therefore, under-approximate faults which only propagate through output $PO0$ are not affected by approximation of $n11/1$, in particular $n9 \rightarrow PO0/1$ and $PO0/0$. In addition, fault $n9 \rightarrow PO1/1$ is not compatible with the set of conditions from $A_{n11/1}$. For all these faults there are no unmasked input vectors, so their probabilities do not change. The rest of under-approximate faults are next re-implied.

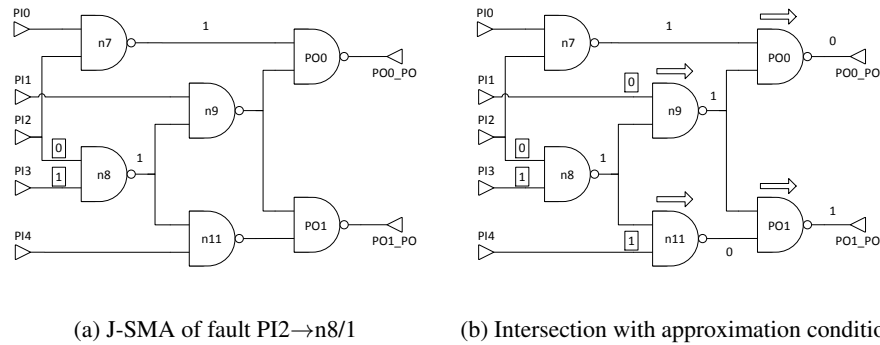


Figure A.65: Effect of fault $n11/1$ over $PI2 \rightarrow n8/1$

The effect over fault $PI2 \rightarrow n8/1$ is computed at first place. Although this fault can propagate through both outputs, now it is re-implied just with respect to $PO1$. Due to the existence of several propagation paths for this fault through $PO1$ there are few assignments that can be inferred, as it can be seen in Figure A.65a. Then this set of assignments is intersected with $A_{n11/1}$, generating the MAs $PI1=0$ and $PI4=1$ which are then propagated through the circuit as in Figure A.65b. J-SMA $J_{PI2 \rightarrow n8/1}^7 = \overline{PI1} \cdot \overline{PI2} \cdot PI3 \cdot PI4$ is then extracted, which is stored for future implications. Probability of this justification frontier is computed, which is equal to 0.0625. And finally this value is deducted from the initial value of $PI2 \rightarrow n8/1$ probability, resulting in 0.125

Next fault $PI3/1$ is re-implied, which is very similar to the previous one. Figure A.66a contains the set of assignments deduced from implication of fault $PI3/1$ with respect to output $PO1$. Over this, approximation condition $A_{n11/1}$ is applied, generating the additional MAs $PI1=0$ and $PI4=1$, which are subsequently propagated as shown in Figure A.66b. At the end of implication process, justification frontier $J_{PI3/1}^7 = \overline{PI1} \cdot PI2 \cdot \overline{PI3} \cdot PI4$ is extracted and stored for future implications. Probability of this J-SMA is 0.0625, which subtracted from original probability of fault $PI3/1$ results in 0.1875.

Then fault $n11/1$ comes, the approximation candidate itself. This fault is implied again, generating the set of assignments shown in Figure A.67, which is then intersected with $A_{n11/1}$. But in this case the justification condition of fault $n11/1$ perfectly overlaps with the current state of the circuit, and therefore no additional assignments

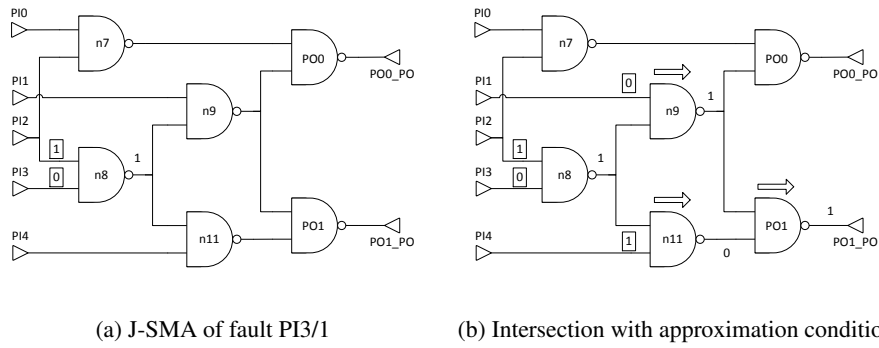


Figure A.66: Effect of fault n11/1 over PI3/1

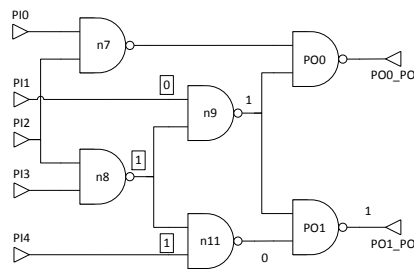


Figure A.67: Effect of fault n11/1 over itself

Fault	Justification frontier (PO1)	Probability		
		Former	Unmasked	Updated
PI2→n8/1	$\overline{PI1} \overline{PI2} PI3 PI4$	0.1875	0.0625	0.125
PI3/1	$\overline{PI1} PI2 \overline{PI3} PI4$	0.25	0.0625	0.1875
n9→PO0/1	-	0.2812	0	0.2812
n9→PO1/1	-	0.1875	0	0.1875
n11/1	$\overline{PI1} PI4 n8$	0.1875	0.1875	0
PO0/0	-	0.3437	0	0.3437
PO1/0	$\overline{PI1} PI4 n8$	0.6094	0.1875	0.4219

Table A.15: Summary of n11/1 approximation

can be inferred. J-SMA $J_{n11/1}^7 = \overline{PI1} \cdot PI4 \cdot n8$ is obtained. This justification frontier has a probability of 0.1875, which makes that probability of fault n11/1 drops to 0.

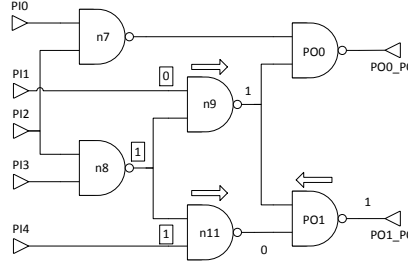


Figure A.68: Effect of fault n11/1 over PO1/0

Finally, fault PO1/0 is re-implied. Approximation condition of fault n11/1 is intersected with PO1=1, which is the only assignment inferred from implication of PO1/0. This generates the MAs PI1=0, PI4=1 and n8=1, which are propagated to assignments n9=1 and n11=0. As a result, PO1=1 becomes justified as it can be seen in Figure A.68. The final J-SMA is $J_{PO1/0}^7 = \overline{PO1} \cdot PI4 \cdot n8$, which has associated a probability of 0.1875. This value is then subtracted from initial probability of fault PO1/0, being 0.4219 the updated value.

Table A.15 summarizes the effects of n11/1 approximation, including the set of unmasked input vectors and probability changes for each fault. Then total EP is increased in an amount proportional to the sum of all variations in fault probabilities, that is,

$$\begin{aligned}
 EP_7 &= EP_6 + \frac{\sum_i P(f_i \cap A_{n11/1} \cap \overline{f_i} \cap \overline{A_F})}{n} = \\
 &= 14.6484\% + \frac{2 \cdot 0.0625 + 2 \cdot 0.1875}{16} = 17.7734\%
 \end{aligned}$$

If a target EP had been set with a value below 17.7734%, approximation algorithm would have finished at this point.

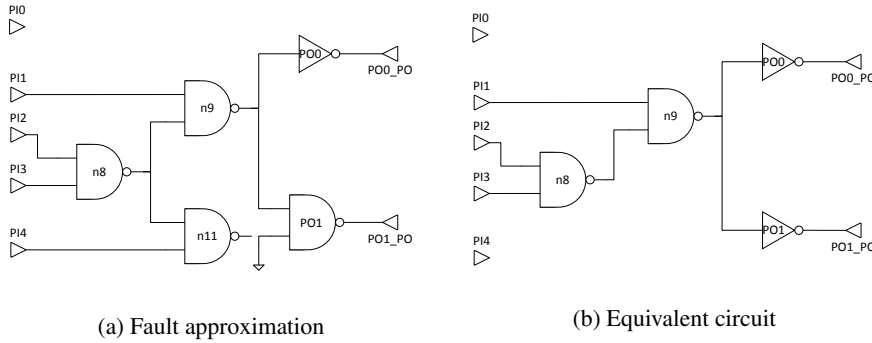


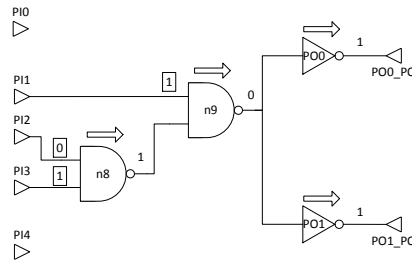
Figure A.69: Approximation of fault n11/1

Under-approx fault	Probability	Over-approx fault	Probability
PI2→n8/1	0.125	PI0/0	-
PI3/1	0.1875	PI1/1	-
n7/1	-	PI2→n7/1	-
n9→PO0/1	0.2812	PI4/1	-
n9→PO1/1	0.1875	n8→n9/1	-
n11/1	-	n8→n11/1	-
PO0/0	0.3437	PO0/1	-
PO1/0	0.4219	PO1/1	-

Table A.16: Fault probabilities after n11/1 approximation

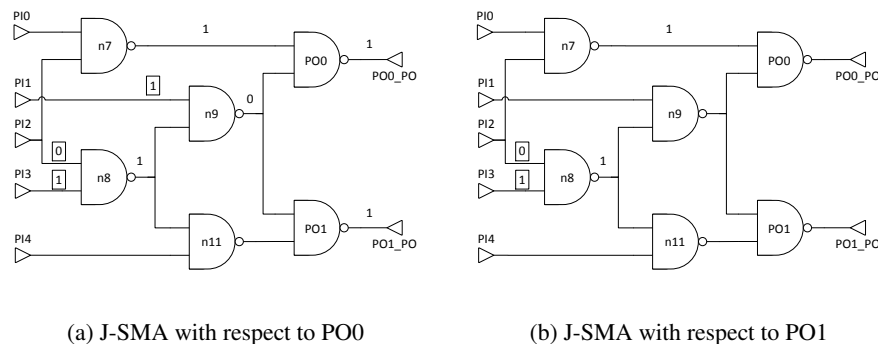
Now candidate fault n11/1 can finally be approximated by forcing the line from n11 to PO1 with constant logic 1 as it can be seen in Figure A.69a. Cause of this, node n11 is left dangling and therefore it can be removed from the circuit, saving area without further degradation of functionality. On the other hand, this logic transformation causes node PO1 to be functionally equivalent to an inverter. After performing proper simplifications, circuit of Figure A.69b is generated, which is the current under-approximation. This is paired with the trivial approximation of the over-approximate circuit to form the error masking scheme corresponding to current state. After fault approximation, the remaining under-approximate faults are implied again in this new circuit in order to detect additional redundancies. The result of this step is that there are no more approximated faults as consequence of n11/1 approximation.

Next candidate fault to be approximated is selected among the remaining under-approximate faults. Table A.16 collects the current probability values of all faults. Among them, the fault with the lowest probability is selected, which corresponds with PI2→n8/1. A new iteration of the approximation algorithm then begins.

Figure A.70: Approximation condition of $PI2 \rightarrow n8/1$

8th approximation: $PI2 \rightarrow n8/1$

First step in a new iteration consist in obtaining the approximation conditions of selected fault, as usual. This is achieved in this case by implying fault $PI2 \rightarrow n8/1$ in the under-approximate circuit. Controllability condition $PI2=0$ and observability conditions $PI1=1$ and $PI3=1$ form the initial set of mandatory assignments for this fault, which are then propagated through the circuit by direct implication as shown in Figure A.70. Justification frontier for this implication is then formed by assignments $PI1=1$, $PI2=0$ and $PI3=1$. It must be noted that this fault can propagate through both outputs, and the set of conditions that allow its propagation is the same with respect to both outputs. Therefore, $A_{PI2 \rightarrow n8/1_0} = A_{PI2 \rightarrow n8/1_1} = PI1 \cdot \overline{PI2} \cdot PI3$ are inferred as approximation conditions. All remaining under-approximate faults are then intersected with these approximation conditions in order to determine which input vectors become unmasked for each fault. Among them, $PI3/1$ is the only fault for which intersection with either $A_{PI2 \rightarrow n8/1_0}$ or $A_{PI2 \rightarrow n8/1_1}$ is null, and therefore its probability does not change during current iteration.

Figure A.71: Effect of fault $PI2 \rightarrow n8/1$ over itself

First fault is the approximation candidate itself, $PI2 \rightarrow n8/1$. This is implied first with respect to output $PO0$. The J-SMA of this fault -which can be seen in Figure

A.71a- is identical to the approximation condition $A_{PI2 \rightarrow n8/1_0}$. Therefore no additional assignments are inferred and justification frontier $J_{PI2 \rightarrow n8/1_0}^8 = PI1 \cdot \overline{PI2} \cdot PI3$ is obtained, with a probability of 0.125. Then the same fault is implied with respect to PO1. Figure A.71b shows the assignments resulting from it. Over this, approximation condition $A_{PI2 \rightarrow n8/1_1}$ is applied and, at the same time, input vectors already included in $J_{PI2 \rightarrow n8/1_0}^8$ have to be excluded. It can be seen that both conditions are identical, and therefore it is not possible to include one of them and exclude the other simultaneously. Because of this, the result of implication with respect to PO1 is the null set. Finally, probability of fault $PI2 \rightarrow n8/1$ is updated by subtracting the value of inferred J-SMAs. As a result, fault probability drops to 0.

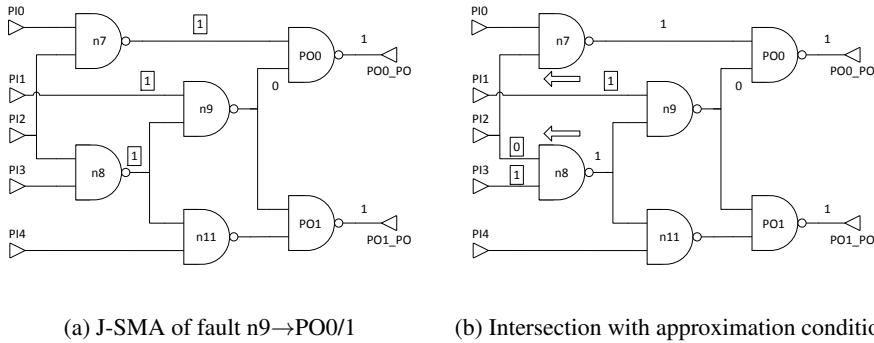
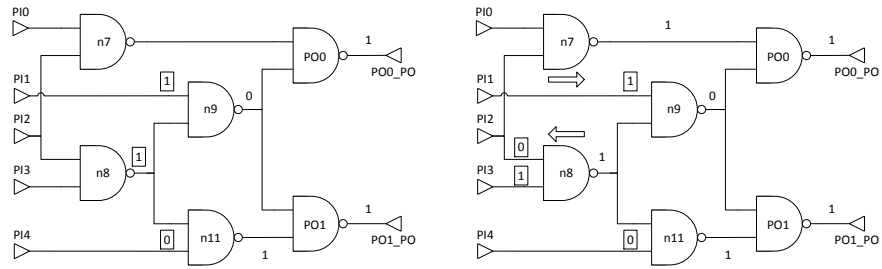


Figure A.72: Effect of fault $PI2 \rightarrow n8/1$ over $n9 \rightarrow PO0/1$

Then fault $n9 \rightarrow PO0/1$ comes. The result of its implication is shown in Figure A.72a. This fault only propagates through output PO0, so it is intersected with approximation condition $A_{PI2 \rightarrow n8/1_0}$. This generates the additional MAs $PI2=0$ and $PI3=1$. Assignments $n7=1$ and $n8=1$ become justified by $PI2=0$ as it can be seen in Figure A.72b. At the end of implication process, $J_{n9 \rightarrow PO0/1}^8 = PI1 \cdot \overline{PI2} \cdot PI3$ is obtained as justification frontier, with a probability equal to 0.125. Based on this value, probability of fault $n9 \rightarrow PO0/1$ is updated, resulting in 0.1562.

Next fault is $n9 \rightarrow PO1/1$. The set of assignments obtained during its initial implications is shown in Figure A.73a. Then approximation condition $A_{PI2 \rightarrow n8/1_1}$ is applied, because this faults only propagates through output PO1. This makes MAs $PI2=0$ and $PI3=1$ to appear, justifying assignment $n8=1$ as it can be seen in Figure A.73b. Then J-SMA $J_{n9 \rightarrow PO1/1}^8 = PI1 \cdot \overline{PI2} \cdot PI3 \cdot \overline{PI4}$ is extracted, which has associated a probability equal to 0.0625. Finally this value is deducted from initial probability of fault $n9 \rightarrow PO1/1$, with a result of 0.125.

Later fault $PO0/0$ is addressed. The set of assignments for this fault is reduced to $PO0=1$, which is intersected with the approximation condition corresponding to output PO0, $A_{PI2 \rightarrow n8/1_0}$ as it can be seen in Figure A.74. These new MAs allow justifying the original J-SMA, so the new justification frontier is $J_{PO0/0}^8 = PI1 \cdot \overline{PI2} \cdot PI3$. This has associated a probability equal to 0.125, which in turn reduces the value of $PO0/0$ probability to 0.2187.



(a) J-SMA of fault $n9 \rightarrow PO1/1$ (b) Intersection with approximation condition

Figure A.73: Effect of fault $PI2 \rightarrow n8/1$ over $n9 \rightarrow PO1/1$

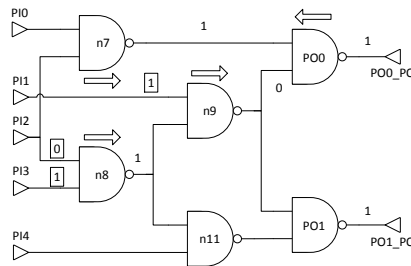


Figure A.74: Effect of fault $PI2 \rightarrow n8/1$ over $PO0/0$

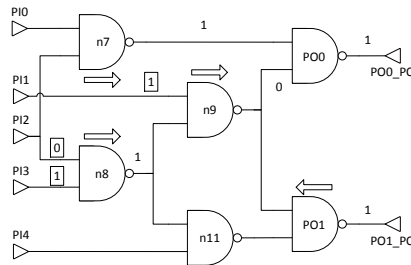


Figure A.75: Effect of fault $PI2 \rightarrow n8/1$ over $PO1/0$

In the last place, fault PO1/0 is re-implied. This fault obviously propagates through output PO1, so the approximation condition applied is $A_{PI2 \rightarrow n8/1}$. This is intersected with the unique assignment that can be inferred from that fault, PO1=1, which is justified by MAs derived from the approximation condition as shown in Figure A.75. Eventually J-SMA $J_{PO1/0}^8 = PI1 \cdot \overline{PI2} \cdot PI3$ is inferred, with a probability of 0.125. Then probability of fault PO1/0 is updated, which descends to 0.2969.

Fault	Justification frontier		Probability			
			Former	Unmasked		Updated
	PO0	PO1		PO0	PO1	
PI2→n8/1	$PI1 \overline{PI2} PI3$	-	0.125	0.125	0	0
PI3/1	-	-	0.1875	0	0	0.1875
n9→PO0/1	$PI1 \overline{PI2} PI3$	-	0.2812	0.125	0	0.1562
n9→PO1/1	-	$PI1 \overline{PI2} PI3 \overline{PI4}$	0.1875	0	0.0625	0.125
PO0/0	$PI1 \overline{PI2} PI3$	-	0.3437	0.125	0	0.2187
PO1/0	-	$PI1 \overline{PI2} PI3$	0.4219	0	0.125	0.2969

Table A.17: Summary of PI2→n8/1 approximation

Table A.17 contains the results of re-implication and probability updating in the usual format. With these results, total EP is then updated as follows

$$\begin{aligned}
 EP_8 &= EP_7 + \frac{\sum_i P(f_i \cap A_{PI2 \rightarrow n8/1} \cap \overline{f_i} \cap \overline{A_F})}{n} = \\
 &= 17.7734\% + \frac{0.0625 + 4 \cdot 0.125}{16} = 21.2891\%
 \end{aligned}$$

If this new value had surpassed the target EP, approximation algorithm would have finished in the current iteration.

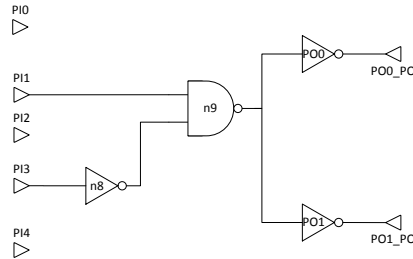


Figure A.76: Approximation of fault PI2→n8/1

After probability updating has been performed, fault PI2→n8/1 can be approximated by forcing primary input PI2 to logic 1. As a result, node n8 can be simplified with an inverter because one of its inputs is now tied to a logic constant. The resulting under-approximated circuit appears in Figure A.76, which would be part of the error masking scheme in conjunction with the trivial over-approximate circuit. Then all re-

Under-approx fault	Probability	Over-approx fault	Probability
PI2→n8/1	-	PI0/0	-
PI3/1	0.1875	PI1/1	-
n7/1	-	PI2→n7/1	-
n9→PO0/1	0.1562	PI4/1	-
n9→PO1/1	0.125	n8→n9/1	-
n11/1	-	n8→n11/1	-
PO0/0	0.2187	PO0/1	-
PO1/0	0.2969	PO1/1	-

Table A.18: Fault probabilities after PI2→n8/1 approximation

maining under-approximate faults are implied in the resulting approximate circuit. In this step there are no redundant faults, so there are no additional approximated faults.

Finally, the next fault which is going to be approximated is selected according to their probabilities, which are collected in Table A.18. Among them, the fault with the lowest probability value is n9→PO1/1, and therefore is the chosen fault.

9th approximation: n9→PO1/1

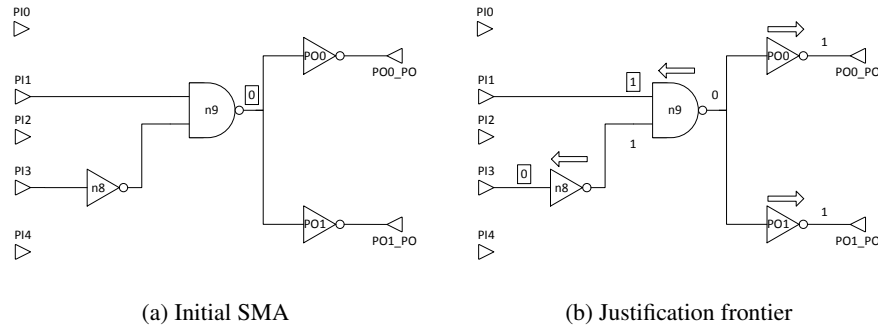


Figure A.77: Approximation condition of n9→PO1/1

Once a new fault has been selected, it is implied in the under-approximate circuit with the aim of obtaining its approximation condition. Figure A.77a shows the initial SMA of the fault, which is just the controllability condition n9=0. This is propagated to both primary outputs, and at the same time justified by PI1=1 and PI3=0, as shown in Figure A.77b. After implication, J-SMA $A_{n9 \rightarrow PO1/1} = PI1 \cdot \overline{PI3}$ is extracted. It must be noted that fault n9→PO1/1 only propagates through output PO1. Then all remaining under-approximate faults are re-implied with this new condition. But those which do not propagate through the same output -n9→PO0/1 and PO0/0- are not affected by approximation of n9→PO1/1, the result of implication for these faults is

the null set and their probabilities are not modified. The rest of under-approximated faults are following addressed.

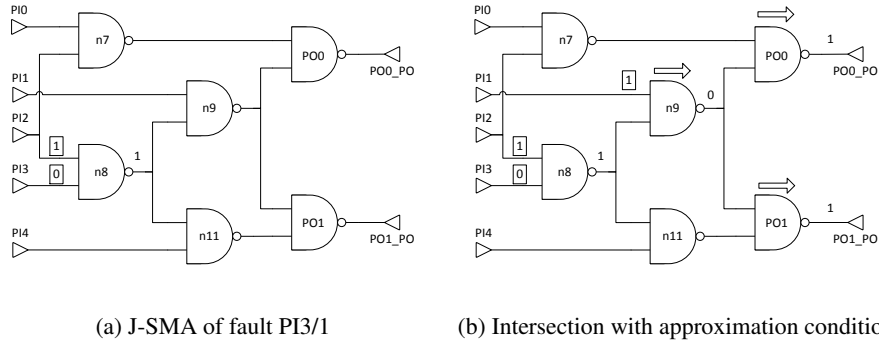


Figure A.78: Effect of fault $n9 \rightarrow PO1/1$ over PI3/1

First fault PI3/1 is analysed. Over the set of assignments inferred for this fault - see Figure A.78a- approximation condition $A_{n9 \rightarrow PO1/1}$ is applied, thus generating the additional MA PI1=1. These assignments are then propagated by direct implication as shown in Figure A.78b. After all possible assignments have been inferred, J-SMA $J_{PI3/1}^9 = PI1 \cdot PI2 \cdot \overline{PI3}$ is extracted. This condition occurs with a probability of 0.125, a value that is deducted from PI3/1 probability which goes down to 0.0625.

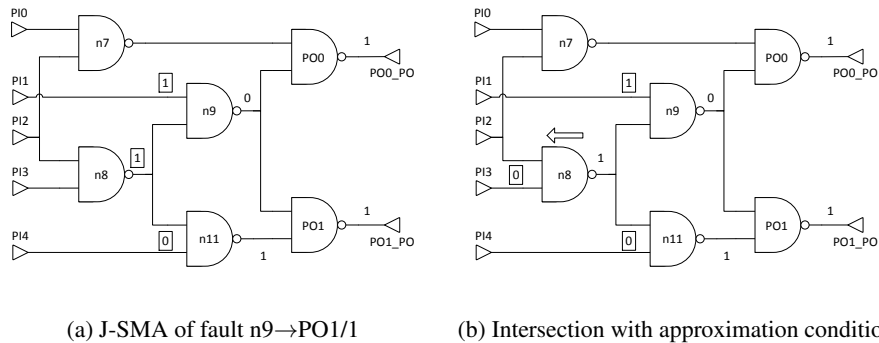
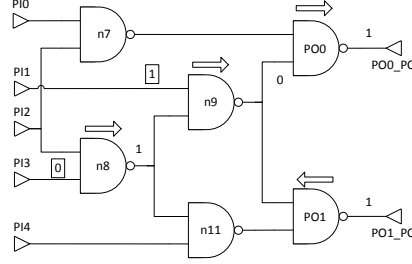


Figure A.79: Effect of fault $n9 \rightarrow PO1/1$ over itself

Next the approximation candidate itself is considered. The set of assignments resulting from the implication of fault $n9 \rightarrow PO1/1$ is shown in Figure A.79a. Then approximation condition $A_{n9 \rightarrow PO1/1}$ is overlapped. This generates the additional MA PI3=0, which justifies $n8=1$ as it can be seen in Figure A.79b. Justification frontier $J_{n9 \rightarrow PO1/1}^9 = PI1 \cdot \overline{PI3} \cdot \overline{PI4}$ is obtained, which has associated a probability equal to 0.125. In conclusion, probability of fault $n9 \rightarrow PO1/1$ drops to 0.

Finally fault PO1/0 is re-implied. $A_{n9 \rightarrow PO1/1} = PI1 \cdot \overline{PI3}$ is the approximation condition applied in conjunction with original assignment PO1=1, which eventually

Figure A.80: Effect of fault $n9 \rightarrow PO1/1$ over $PO1/0$

Fault	Justification frontier (PO1)	Probability		
		Former	Unmasked	Updated
PI3/1	$PI1 PI2 \overline{PI3}$	0.1875	0.125	0.0625
$n9 \rightarrow PO0/1$	-	0.1562	0	0.1562
$n9 \rightarrow PO1/1$	$PI1 \overline{PI3} \overline{PI4}$	0.125	0.125	0
PO0/0	-	0.2187	0	0.2187
PO1/0	$PI1 \overline{PI3}$	0.2969	0.25	0.0469

Table A.19: Summary of $n9 \rightarrow PO1/1$ approximation

becomes justified as it can be seen in Figure A.80. After implication, it is obtained the J-SMA $J_{PO1/0}^9 = PI1 \cdot \overline{PI3}$, with a probability of 0.25. This value is subtracted from $PO1/0$ probability, resulting in 0.0469.

The results of intersection with $A_{n9 \rightarrow PO1/1}$ with every remaining under-approximate fault are collected in Table A.19, which includes the inferred J-SMAs and its associated probability, and the changes in fault probabilities. From this data, total EP is increased in an amount proportional to the accumulated probability of unmasked input vectors as follows

$$\begin{aligned}
 EP_9 &= EP_8 + \frac{\sum_i P(f_i \cap A_{n9 \rightarrow PO1/1} \cap \overline{f_i} \cap \overline{A_F})}{n} = \\
 &= 21.2891\% + \frac{2 \cdot 0.125 + 0.25}{16} = 24.4141\%
 \end{aligned}$$

This means that if any fault appears in the circuit within the masking system, it has around 24.4% chance on average that it is propagated to outputs. If a target EP below that value had been set, the algorithm would have stop in current iteration.

After probability updating phase, fault $n9 \rightarrow PO1/1$ can finally be approximated. Input of node PO1 is tied to logic constant 1, which is equivalent to set primary output PO1 to 0. Figure A.81 shows the simplified under-approximate circuit, which is paired with the trivial over-approximation to form an error masking scheme. Later, the remaining under-approximate faults are implied in this new circuit. Because output PO1 is now tied to a logic constant, all faults which only propagate through it are left

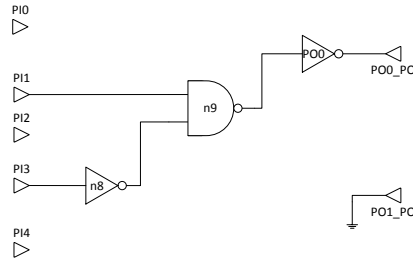


Figure A.81: Approximation of fault n9→PO1/1

Under-approx fault	Probability	Over-approx fault	Probability
PI2→n8/1	-	PI0/0	-
PI3/1	0.0625	PI1/1	-
n7/1	-	PI2→n7/1	-
n9→PO0/1	0.1562	PI4/1	-
n9→PO1/1	-	n8→n9/1	-
n11/1	-	n8→n11/1	-
PO0/0	0.2187	PO0/1	-
PO1/0	-	PO1/1	-

Table A.20: Fault probabilities after n11/1 approximation

without a valid propagation path. This is the case of PO1/0 fault, whose implication determines that it becomes redundant and therefore it is marked as approximated as well. It must be noted that its probability is still 0.0469, but that only means the initial probability value for this fault was overestimated.

Only three under-approximate faults remain that have not been approximated yet, and among them the next approximation candidate is selected according to their probabilities. Based on data from Table A.20, PI3/1 is the fault with the lowest probability, and therefore it is selected for the next iteration.

10th approximation: PI3/1

After selecting a new fault to be approximated, it is implied in the under-approximate circuit in order to obtain its approximation condition. Controllability and observability conditions are applied, resulting in MA $PI1=1$ and $PI3=0$, which are subsequently propagated as shown in Figure A.82. Justification frontier $A_{PI3} = PI1 \cdot \overline{PI3}$ is obtained, which becomes the approximation condition of fault PI3/1. This is identical to that obtained in previous iteration with fault n9→PO1/1, but in this case it is computed with respect to output PO0 instead of PO1. Then all remaining faults are re-implied with this new condition.

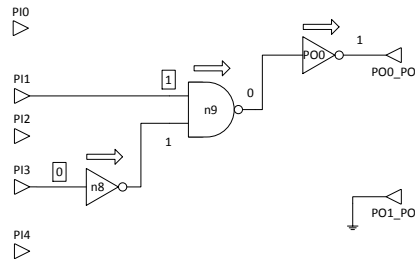


Figure A.82: Approximation condition of PI3/1

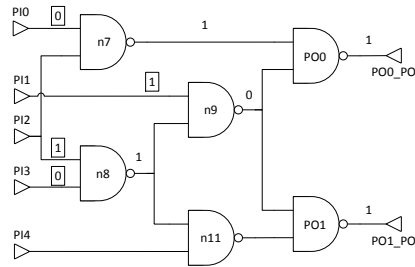


Figure A.83: J-SMA of fault PI3/1 with respect to PO0

First fault PI3/1 is analysed, the approximation candidate. This time it is implied with respect to output PO0, generating the set of assignments in Figure A.83. This is intersected with A_{PI3} , and at the same time the sets of assignments inferred in previous implications of fault PI3/1 with respect to output PO1 have to be excluded, in particular $J_{PI3}^7 = \overline{PI1} \cdot PI2 \cdot \overline{PI3} \cdot PI4$ and $J_{PI3/1}^9 = PI1 \cdot PI2 \cdot \overline{PI3}$. It can be appreciated that condition $\overline{J_{PI3/1}^9}$ is not compatible with assignments of Figure A.83. Therefore, the result of current implication is the null set and probability of fault PI3/1 is not altered. This fault has a remaining probability of 0.0625, but it is going to be approximated in current iteration. Therefore, it can be concluded that initial probability of this fault was overestimated.

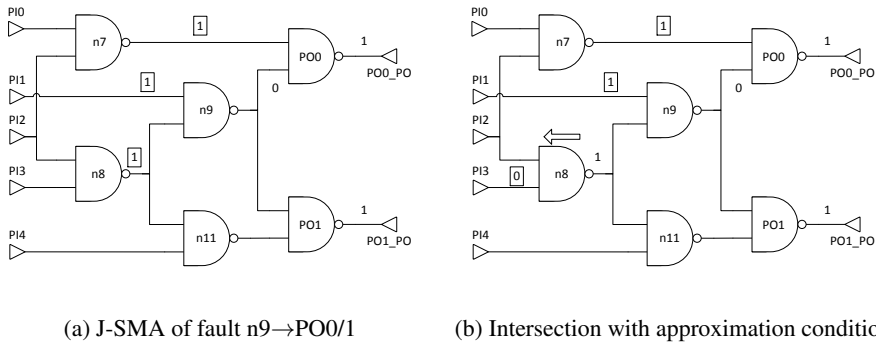


Figure A.84: Effect of fault PI3/1 over $n9 \rightarrow PO0/1$

Next fault is $n9 \rightarrow PO0/1$. Figure A.84a shows the assignments inferred from implication of this fault, which are then intersected with A_{PI3} . This generates the additional MA $PI3=0$, which justifies $n8=1$ as it can be seen in Figure A.84b. Implication ends and justification frontier $J_{n9 \rightarrow PO0/1}^{10} = PI1 \cdot \overline{PI3} \cdot n7$ is obtained. This J-SMA has a probability equal to 0.1875. After deducting this value, probability of fault $n9 \rightarrow PO0/1$ descends to -0.0313, indicating that the initial probability value of this fault was underestimated.

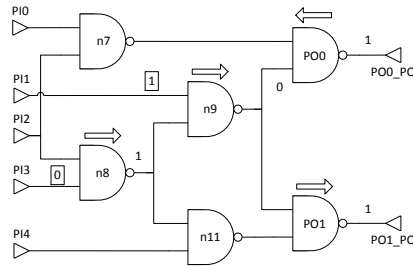


Figure A.85: Effect of fault PI3/1 over PO0/0

Fault	Justification frontier (PO0)	Probability		
		Former	Unmasked	Updated
PI3/1	-	0.0625	0	0.0625
n9→PO0/1	$PI1 \overline{PI3} n7$	0.1562	0.1875	-0.0313
PO0/0	$PI1 \overline{PI3}$	0.2187	0.25	-0.0313

Table A.21: Summary of PI3/1 approximation

PO0/0 is the last fault in the list. Approximation condition $A_{PI3} = PI1 \cdot \overline{PI3}$ is implied along with assignment PO0=1, which eventually becomes justified as shown in Figure A.85. J-SMA $J_{PO0/0}^{10} = PI1 \cdot \overline{PI3}$ is obtained with a probability of 0.25. As a result, probability of fault PO0/0 drops to -0.0313. It can be concluded that this value was initially underestimated too.

Table A.21 summarizes the results of implication with respect to PI3/1 approximation, including inferred justification frontiers and all probability adjustments. According with these results, total EP is updated in the following way

$$\begin{aligned}
 EP_{10} &= EP_9 + \frac{\sum_i P(f_i \cap A_{PI3/1} \cap \overline{f_i} \cap A_F)}{n} = \\
 &= 24.4141\% + \frac{0.1875 + 0.25}{16} = 27.1484\%
 \end{aligned}$$

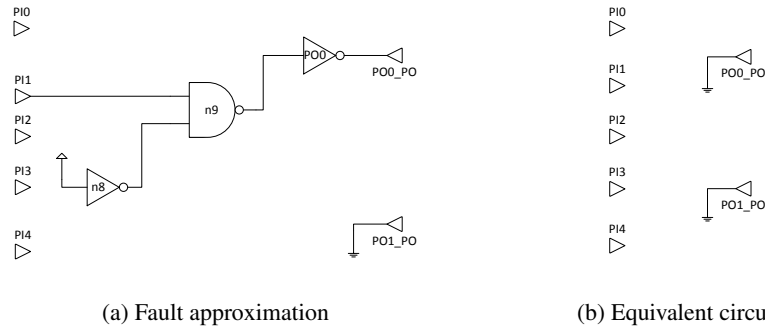


Figure A.86: Approximation of fault PI3/1

Then the fault PI3/1 can be approximated by assigning a logic 1 to input PI3, as shown in Figure A.86a. This logic constant can be propagated through the circuit, eventually resulting in output PO0 tied to logic 0. Therefore, the result of this logic transformation is the trivial approximation (see Figure A.86b). As a result all the propagation paths become blocked, so all the remaining faults are automatically approximated. The complete trivial approximation is obtained, where both approximate circuits are reduced to logic constants. In this situation there is no additional logic, and the target circuit is completely unprotected. There are no faults left, so the approximation algorithm finishes here anyway.

Fault	Probabilities				
	1 st approx.	2 nd approx.	3 rd approx.	4 th approx.	5 th approx.
PI0/1	0	0.0625	0.0625	0.1875	0.1875
PI1/1	0.125	0.125	0.125	0.3125	0.3438
PI2→n7/1	0.125	0.125	0.125	0.125	0.125
PI2→n8/1	0	0	0	0	0
PI3/1	0	0	0	0	0
PI4/1	0	0	0	0	0.1875
n7/1	0	0	0	0	0
n8→n9/1	0	0.125	0.125	0.125	0.125
n8→n11/1	0	0.0625	0.125	0.125	0.125
n9→PO0/1	0	0	0	0	0
n9→PO1/1	0	0	0	0	0
n11/1	0	0	0	0	0
PO0/0	0	0	0	0	0
PO0/1	0.125	0.1875	0.1875	0.4375	0.4375
PO1/0	0	0	0	0	0
PO1/1	0	0.125	0.1875	0.4375	0.4375
EP	2.3438%	5.0781%	5.8594%	9.3750%	12.3047%

Fault	Probabilities				
	6 th approx.	7 th approx.	8 th approx.	9 th approx.	10 th approx.
PI0/1	0.1875	0.1875	0.1875	0.1875	0.1875
PI1/1	0.3438	0.3438	0.3438	0.3438	0.3438
PI2→n7/1	0.125	0.125	0.125	0.125	0.125
PI2→n8/1	0	0.0625	0.1875	0.1875	0.1875
PI3/1	0	0.0625	0.0625	0.1875	0.1875
PI4/1	0.1875	0.1875	0.1875	0.1875	0.1875
n7/1	0.1875	0.1875	0.1875	0.1875	0.1875
n8→n9/1	0.125	0.125	0.125	0.125	0.125
n8→n11/1	0.125	0.125	0.125	0.125	0.125
n9→PO0/1	0	0	0.125	0.125	0.3125
n9→PO1/1	0	0	0.0625	0.1875	0.1875
n11/1	0	0.1875	0.1875	0.1875	0.1875
PO0/0	0.1875	0.1875	0.3125	0.3125	0.5625
PO0/1	0.4375	0.4375	0.4375	0.4375	0.4375
PO1/0	0	0.1875	0.3125	0.5625	0.5625
PO1/1	0.4375	0.4375	0.4375	0.4375	0.4375
EP	14.6484%	17.7734%	21.2891%	24.4141%	27.1484%

Table A.22: Real contribution of each fault to EP

Finally, in order to show the accuracy of probability estimations performed throughout the whole algorithm, the real fault probabilities have been computed for each one of the error masking schemes resulting from each iteration in the algorithm. These results have been obtained by performing an exhaustive simulation of the faults within the target circuit for each pair of approximations generated. Table A.22 contains the real fault probabilities with respect to each iteration. In addition, total EP has been computed according to these real values. By comparing these results with those obtained throughout the approximation generation algorithm, it can be seen that estimated EP coincides with the real value in every iteration. Furthermore, the accumulated probability of unmasked input vectors computed for each fault is identical to real fault probability on each iteration. In conclusion, although initial computation of probabilities can be a little imprecise, the estimation of effects of successive approximated faults are much more accurate.

A.5 Approximation with node substitution

This section shows an application example of the approximation method by node substitution over the c17 benchmark. This approach is an extension of the method based in dynamic testability measures, which takes advantage on the fault implications required to compute fault probabilities in order to deduce and apply a new kind of logic transformations. The approach is detailed in section 5.4. Therefore, most of the computations performed in the example with dynamic testability measures are reused here. For the sake of brevity, the reader is then referred to the section A.4 for all those computations which are omitted in this example.

The same fault classification as in the example with dynamic testability measures is applied here. However, for some faults in that list substitution candidates can never be found, due to the constraints imposed to the substitution candidates. Such is the case of the faults located at the primary inputs. By substituting a primary input with any other node within the circuit the critical path can never improve, and only a substitution with another primary input preserves the critical path duration. But this transformation is uninteresting, because there is no benefit in terms of area. Therefore, when looking for candidates for node substitution, the search is conducted just for the following faults (as the replaced node. This restriction does not apply to the replacing nodes)

- Under-approximation faults with substitution candidates: $n9 \rightarrow PO0/1$, $n9 \rightarrow PO1/1$, $n7/1$, $n11/1$, $PO0/0$ and $PO1/0$.
- Over-approximation faults with substitution candidates: $n8 \rightarrow n9/1$, $n8 \rightarrow n11/1$, $PO0/1$ and $PO1/1$.

This example is structured in several subsections for a better understanding. The first subsection corresponds to the initial search of substitution candidates, which is performed right after computing fault probabilities for the first time, before applying any approximation. Then there is one subsection per iteration of the approximation algorithm. Each iteration corresponds to one logic transformation, either a fault approximation or a node substitution. In this example there is a total of 9 iterations, from the full TMR to the trivial solution.

Initial substitution candidate search

The approximation generation algorithm initiates with a parity check of every line in the circuit. The c17 benchmark is already a fully unate circuit, and therefore applying the unate expansion is not required.

Then, fault probabilities are computed for the first time. This step is exactly the same than the initial probability computation step from the example with dynamic testability measures of section A.4, and the results obtained (both the inferred assignments and the probability value of each fault) are identical.

Once the probability of any fault is computed, the set of inferred assignments for that particular fault is analysed in order to identify substitution candidates. In principle, any line which has received an assignment during the fault implication is a potential candidate in order to replace the line where the fault is located. However, there are some constraints which must be met, thus reducing the set of potential candidates. Namely they are 5: the logic transformation must be coherent with the computed line parities, avoid asynchronous combinational feedback loops, the critical path must not increase, exclude substitutions which are equivalent to a fault approximation, and ensure that the logic transformation effectively reduces the circuit area (by removing at least one node).

Because of these constraints, it is not possible to find substitution candidates for the faults located at the primary inputs, which are automatically excluded from the search. The same applies to the faults located at multiple fanout points, although in this case those faults are still taken into account because the multiple fanout point could disappear in subsequent stages of the approximation process.

Taking into account the previous constraints, the set of substitution candidates is very limited in this example. Faults $n8 \rightarrow n9/1$, $n8 \rightarrow n11/1$, $n9 \rightarrow PO0/1$ and $n9 \rightarrow PO1/1$ correspond to multiple fanout points, and therefore they have no valid candidates as long as the multiple fanout points persist. With respect to the fault $n7/1$, all of its inferred assignments (which can be seen in the Figure A.22b) correspond to either the fanout cone of the fault (assignments $n7=0$ and $PO0=1$), which would produce a combinational feedback loop, or to an immediate input to any of the nodes in the fanout cone of the fault (assignments $PI0=1$, $PI2=1$ and $n9=1$), which would be equivalent to a fault approximation. Therefore, there are no candidates for the fault $n7/1$ either. The same applies to faults $PO0/0$, $PO1/0$, $PO0/1$ and $PO1/1$: just the fault location and at most its immediate inputs (in the cases of $PO0/1$ and $PO1/1$) receive an assignment.

The only fault from which substitution candidates are identified is $n11/1$. Figure A.28b summarizes the set of inferred assignments for this fault. Among them, nodes $n11$ and $PO1$ belong to the fanout cone of the fault, and nodes $PI4$, $n8$ and $n9$ are immediate inputs of either $n11$ or $PO1$. Therefore, all these nodes are excluded. Only the assignment $PI1=0$ remains as a valid substitution candidate, and because the value of this assignment is the same than the assignment on the fault location ($n11=0$), it is identified as a potential direct substitution from $PI1$ to $PO1$. In this case, because the replacing node is a primary input, the logic transformation would preserve the circuit parities independently from the parity of the node $n11$. It must be noted that the fault $n11/1$ propagates just to output $PO1$, and therefore the implication with respect to $PO0$ is not considered. If a fault may propagate through several outputs, it has to be at first

implied independently for each output, thus obtaining a particular set of assignments per output, and then all of those sets of assignments have to be properly combined.

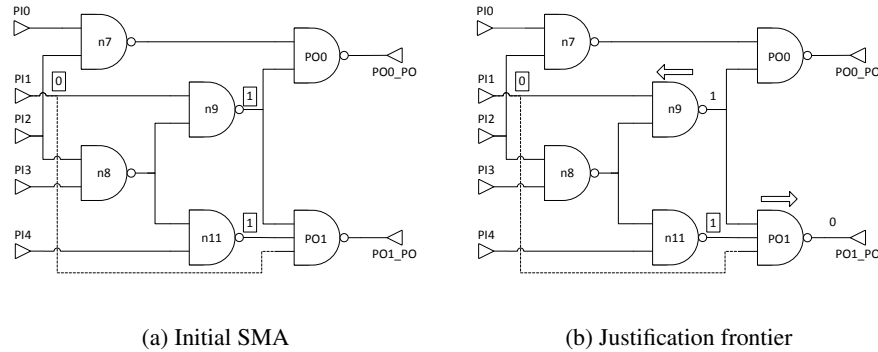


Figure A.87: Implication of substitution candidate $PI1 \rightarrow PO1/1$

The identified substitution candidate has associated the virtual fault $PI1 \rightarrow PO1/1$. This fault represents the set of input vectors which will be unmasked if this node substitution is finally performed. In order to decide whether it is worth performing this logic transformation or not, the probability of the associated fault is computed. The fault $PI1 \rightarrow PO1/1$ has the controllability condition $PI1=0$ and the observability conditions $n9=1$ and $n11=1$, as shown in the Figure A.87a. Then these assignments are propagated by direct implication as usual. $PI1=0$ justifies $n9=1$, and the value of $PO1$ is derived from $n9$ and $n11$ (it must be noted that the discontinuous connection from $PI1$ to $PO1$ does not really exist), as shown in the Figure A.87b. Finally, the J-SMA $PI1=0, n11=1$ is obtained, and the probability of the fault $PI1 \rightarrow PO1/1$ is inferred from the J-SMA as $P(PI1 \rightarrow PO1/1) = P(\overline{PI1}) \cdot P(n11) = 0.5 \cdot 0.625 = 0.3125$.

Once the probability of the fault associated to the candidate node substitution has been computed, it is included in the set of approximation candidates along with all the fault approximations (whose probabilities are detailed in the Table A.2), and the best logic transformation is selected. In this example, the selection criteria is the classical heuristic of the dynamic approach: the fault with the lowest probability is selected and, in the case of a draw, the one which produces the highest area savings. The probability of the substitution candidate (0.3125) is worse than the probability of the best fault approximation ($PI2 \rightarrow n7/1$, $n8 \rightarrow n9/1$ or $n8 \rightarrow n11/1$, all of them with a probability of 0.125). Therefore, the candidate node substitution is discarded in favour of one of these fault approximations. Because all of them produces the same area savings (estimated as the size of the transitive fanin), the over-approximate fault $PI2 \rightarrow n7/1$ is arbitrarily chosen as the first approximated fault.

1st approximation: $PI2 \rightarrow n7/1$

After selecting the first fault to be approximated ($PI2 \rightarrow n7/1$), but before approximating it, the approximation condition of the fault has to be inferred, the probability of all the

over-approximate faults has to be updated (including $PI2 \rightarrow n7/1$ itself) and finally the global EP has to be updated as well. All these steps are identical to the first iteration of the example with dynamic testability measures, and they provide exactly the same results.

Next, the selected fault is approximated in the correspondent over-approximate circuit, which up to this point was an exact replica of the c17 benchmark. The approximation of the fault $PI2 \rightarrow n7/1$ generates the logic circuit of Figure A.37, where the node $n7$ has been replaced by an inverter. From now, the fault $PI2 \rightarrow n7/1$ is marked as approximated, and therefore it is not taken into account in future iterations.

Now, all the remaining faults are implied in the new over-approximate circuit. For the over-approximate faults, the goal is discovering additional faults which may have become approximated due to the approximation of the fault $PI2 \rightarrow n7/1$ (with negative result in the current iteration). While for the under-approximate faults, the goal consists in discovering node new substitution candidates. It must be noted that the node substitutions inferred from under-approximate faults during the initial stage of the example (particularly $PI1 \rightarrow PO1/1$) are discarded because the over-approximate circuit has changed.

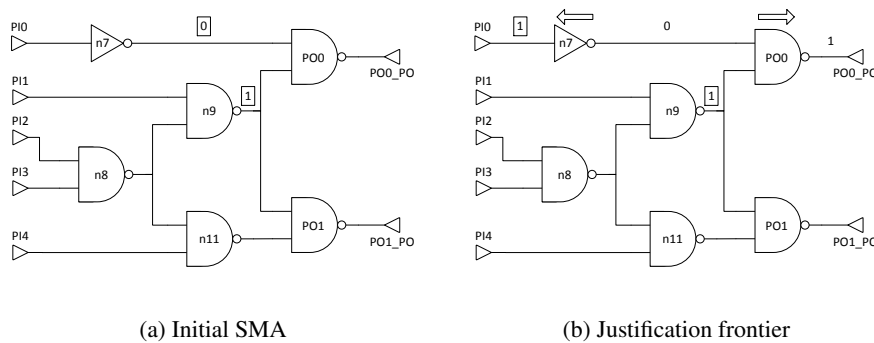


Figure A.88: Implication of fault $n7/1$ in the over-approximate circuit - first approximation

Let us start with the fault $n7/1$. The initial SMA of this fault is constituted by the assignments $n7=0$ and $n9=1$, as shown in the Figure A.88a. These assignments are then propagated by direct implication, as shown in the Figure A.88b. None of these assignments points to a valid substitution candidate. Nodes $n7$ and $PO0$ belong to the output cone of the fault $n7/1$, while nodes $PI0$ and $n9$ are immediate inputs of respectively $n7$ and $PO0$. Therefore, no substitution candidates are inferred for this fault.

The remaining under-approximate faults do not suffer changes with respect to the initial stage. Faults $n9 \rightarrow PO0/1$ and $n9 \rightarrow PO1/1$ are still located at a multiple fanout point, and faults $PO0/0$ and $PO1/0$ only receive one single assignment at the fault location. For all these faults, no substitution candidates are found. With respect to the fault $n11/1$, replacing the node $n7$ with an inverter does not modify the set of deduced assignments (shown in the Figure A.28). Therefore, the direct substitution from $PI1$ to

PO0 is again identified as a candidate substitution. The probability of the associated fault, $PI1 \rightarrow PO1/1$, is again computed with identical result (a probability of 0.3125).

In summary, only the substitution fault $PI1 \rightarrow PO1/1$ is added to the list of candidate approximations along with all the remaining circuit faults, which are listed in the Table A.4. Among them, the fault with the lowest probability is selected as the next fault to be approximated. In this case, there are two possible candidates: $n8 \rightarrow n9/1$ and $n8 \rightarrow n11/1$, both of them with a probability of 0.125 and the same transitive fanin size. Among them, the over-approximate fault $n8 \rightarrow n9/1$ is arbitrarily chosen, and the next iteration of the algorithm begins.

2nd approximation: $n8 \rightarrow n9/1$

Once the fault $n8 \rightarrow n9/1$ has been selected, its approximation condition is inferred on the over-approximate circuit. Then, the probability of every over-approximate fault in the circuit is updated by subtracting the probability of the intersection with the approximation condition of the fault $n8 \rightarrow n9/1$. And finally, the global EP is updated as the average variation of all fault probabilities. All these steps coincide with the second iteration of the example with dynamic testability measures, and they provide exactly the same results.

Then, the fault $n8 \rightarrow n9/1$ is approximated in the over-approximation, resulting in the circuit of the Figure A.45. As a result of the fault approximation, the node $n9$ has been replaced with an inverter. The fault $n8 \rightarrow n9/1$ is now marked as approximated.

The remaining faults are implied again in this new approximate circuit. On the one hand, the over-approximate faults are implied to detect if any of them becomes redundant with the last approximation, which turns out to be false. On the other hand, the under-approximate faults are implied to identify substitution candidates in the new approximate circuit. The substitution candidates discovered in the previous step of the example are discarded.

Among all the considered under-approximate faults, $n9 \rightarrow PO0/1$, $n9 \rightarrow PO1/1$, $PO0/0$ and $PO1/0$ do not generate substitution candidates for different reasons. Faults $n9 \rightarrow PO0/1$ and $n9 \rightarrow PO1/1$ are located at a multiple fanout point, while $PO0/0$ and $PO1/0$ do not infer assignments further from the fault location itself.

With respect to the fault $n7/1$, it is implied with the initial SMA $n7=0$ and $n9=1$, as shown in the Figure A.89a. Then these assignments are propagated by direct implication, resulting in the final set of assignments from the Figure A.89b. Among them, the nodes $PI0$, $n7$, $n9$ and $PO0$ are no valid candidates due to the constraints imposed to the substitution candidates. But the assignment $PI1=0$ is identified as a valid substitution candidate. The assignment at the fault location is $n7=0$, and therefore the identified candidate is a direct substitution from $PI1$ to $PO0$.

This substitution candidate has associated the virtual fault $PI1 \rightarrow PO0/1$, which is now computed as follows. First, the initial SMA is deduced, which is formed by the assignments $PI1=0$, $n7=1$ and $n9=1$, as shown in the Figure A.90a. Then these assignments are propagated by direct implication, as shown in the Figure A.90b. $n7=1$ and $n9=1$ are justified by $PI0=0$ and $PI1=0$ respectively, while the value of $PO0$ is derived from the nodes $n7$ and $n9$ (the discontinuous connection between $PI1$ and $PO0$ does not really exist). Finally, the J-SMA $PI0=0$, $PI1=0$ is inferred, from which the probability

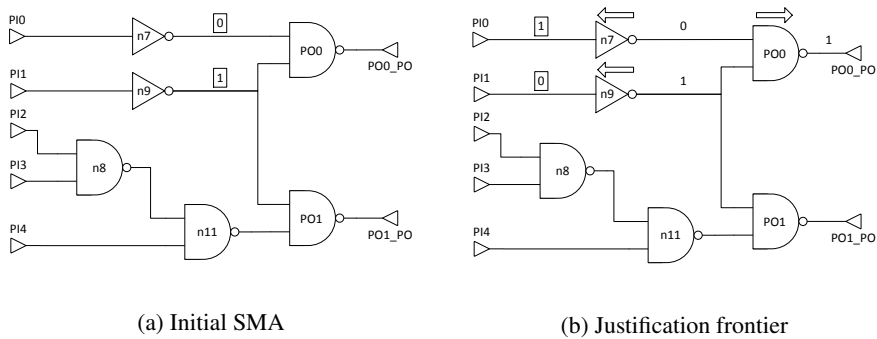


Figure A.89: Implication of fault n7/1 in the over-approximate circuit - second approximation

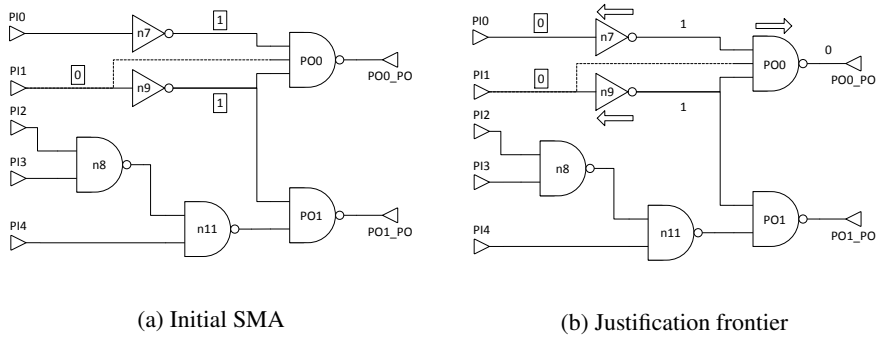


Figure A.90: Implication of substitution candidate $PI1 \rightarrow PO0/1$ - second approximation

of the substitution fault is derived as $P(PI1 \rightarrow PO0/1) = P(\overline{PI0}) \cdot P(\overline{PI1}) = 0.5^2 = 0.25$.

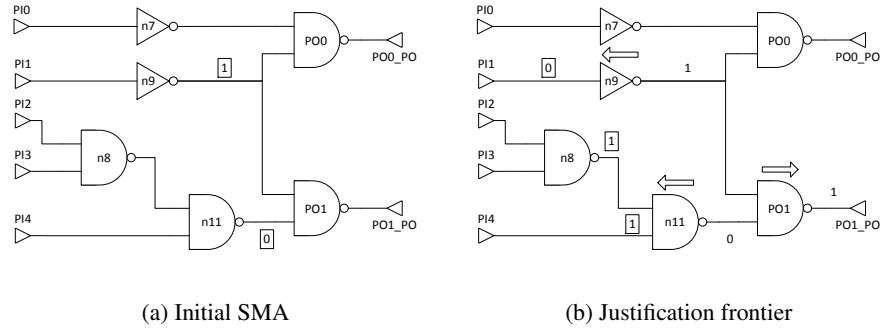


Figure A.91: Implication of fault $n11/1$ in the over-approximate circuit - second approximation

Finally, the fault $n11/1$ is implied. Departing from the initial SMA $n9=1$ and $n11=0$ (as shown in the Figure A.91a), the final set of assignments of the Figure A.91b is obtained by direct implication. Among them, only the node $PI1$ is identified as a valid substitution candidate. By comparing the assignment at this node with the assignment at the fault location ($n11=0$), the candidate is identified as a direct substitution from $PI1$ to $PO1$. The fault associated with this approximation is $PI1 \rightarrow PO1/1$.

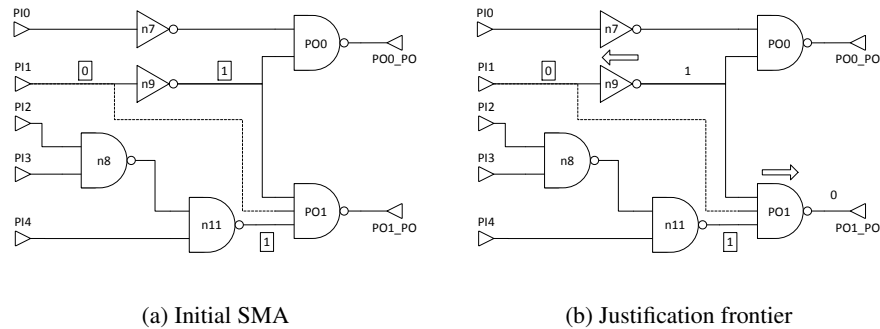


Figure A.92: Implication of substitution candidate $PI1 \rightarrow PO1/1$ - second approximation

Although this is the same node substitution already discovered in the previous steps of the algorithm, the probability of its associated fault has to be computed once more. The initial SMA for this fault is compound by the assignments $PI1=0$, $n9=1$ and $n11=1$, as shown in the Figure A.92a. Then, the assignment $n9=1$ becomes justified by $PI1=0$, while $PO1$ is assigned to 0 because of the values of $n9$ and $n11$. All the implication process is summarized in the Figure A.92b. Finally, the J-SMA $PI1=0$ and $n11=1$ is obtained, which has associated a probability $P(PI1 \rightarrow PO1/1) = P(\overline{PI1}) \cdot P(n11) =$

$$0.5 \cdot 0.625 = 0.3125.$$

Both identified substitution faults, $PI1 \rightarrow PO0/1$ and $PI1 \rightarrow PO1/1$, are included along with all the remaining circuit faults, whose current probability values are listed in the Table A.6. Among all these faults, the one with the lowest probability is selected as the next fault to approximate, which turns out to be the over-approximate fault $n8 \rightarrow n11/1$, with a probability of 0.0625.

3rd approximation: $n8 \rightarrow n11/1$

After selecting the fault $n8 \rightarrow n11/1$, its approximation condition is obtained in the approximate circuit. Then, the probability of each over-approximate fault is updated, and the EP is adjusted according to the new fault probabilities. These steps coincide with the third iteration in the example with dynamic testability measures, and the results obtained are exactly the same.

Later, the fault $n8 \rightarrow n11/1$ is approximated in the over-approximate circuit, and marked as approximated. In this new approximation the node $n11$ is transformed into an inverter and the node $n8$ is removed as it is left dangling. The new over-approximate circuit is the same of the Figure A.49b.

Once the approximation of the new fault has been performed, all the remaining faults are re-implied in this new approximate circuit. With respect to the remaining over-approximate faults, none of them is identified as redundant. While for the under-approximate faults, a new search for substitution candidates is conducted.

Once more, no substitution candidates are deduced from the faults $n9 \rightarrow PO0/1$, $n9 \rightarrow PO1/1$, $PO0/0$ and $PO1/0$ due to the constraints imposed to the substitution candidates. With respect to the fault $n7/1$, the new approximation does not modify the set of assignments inferred for this fault, which are the same than in the previous iteration of the algorithm (see Figure A.89b). As a consequence, the same substitution candidate from $PI1$ to $PO0$ is identified, which corresponds to the substitution fault $PI1 \rightarrow PO0/1$. The probability associated to this fault is computed exactly as in the previous iteration, resulting in a probability $P(PI1 \rightarrow PO0/1) = 0.25$.

The fault $n11/1$ is implied in the last place. The initial SMA $n9=1$, $n11=0$ of Figure A.93a is first applied, and then it is propagated through the circuit, finishing in the set of assignments of the Figure A.93b. Nodes $n11$ and $PO1$ belong to the output cone of the fault $n11/1$, and nodes $PI4$ and $n9$ are immediate inputs of $n11$ and $PO1$ respectively, and therefore all of them are excluded. Only the node $PI1$ is identified as a valid substitution candidate, and due to the assigned values to $PI1$ and $n11$, the substitution is a direct one.

This substitution candidate has associated the virtual fault $PI1 \rightarrow PO1/1$, whose probability is now computed. The initial SMA of the fault is formed by the assignments $PI1=0$, $n9=1$ and $n11=1$, as shown in the Figure A.94a, which are then propagated through the circuit. $n9=1$ and $n11=1$ are respectively justified by $PI1=0$ and $PI4=0$, while at the same time they set $PO1=0$. All this process is summarized in the Figure A.94b. In the end, the J-SMA $PI1=0$, $PI4=0$ is inferred. The probability of this J-SMA is computed as the product of the probabilities of the assignments belonging to it, thus having $P(PI1 \rightarrow PO1/1) = P(\overline{PI1}) \cdot P(\overline{PI4}) = 0.5^2 = 0.25$.

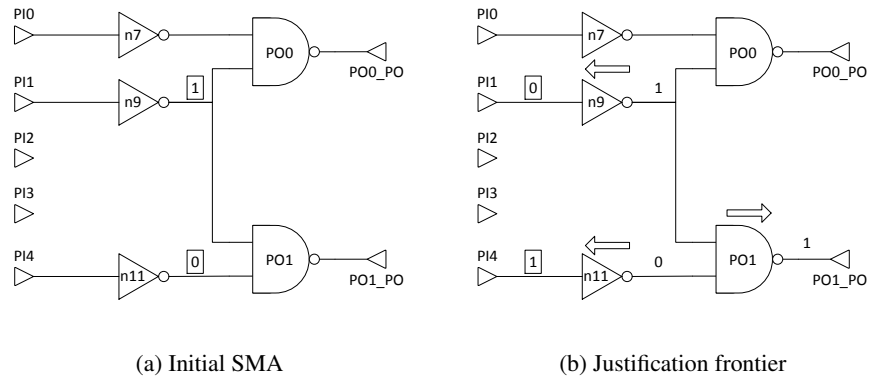


Figure A.93: Implication of fault n11/1 in the over-approximate circuit - third approximation

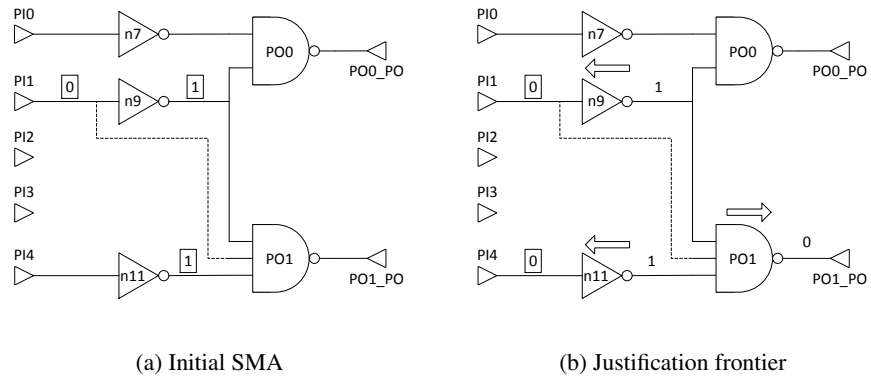


Figure A.94: Implication of substitution candidate $PI1 \rightarrow PO1/1$ - third approximation

Finally, both substitution faults $PI1 \rightarrow PO0/1$ and $PI1 \rightarrow PO1/1$ are considered along with all the remaining circuit faults (whose updated probabilities are included in the Table A.8) to select the next fault to approximate. Among them, the fault with the lowest probability is $PI0/1$, with a value of 0.125, which becomes the next approximated fault. With this, a new iteration of the algorithm begins.

4th approximation: $PI0/1$

The approximation condition of the new selected fault, $PI0/1$, have to be computed in the first place. This is an over-approximate fault, so the probabilities of all the remaining over-approximate faults (including $PI0/1$ itself) have to be updated with the aid of the approximation condition just computed. Then, the global EP is updated too based in the accumulation of all probability variations. These steps correspond with the fourth iteration of the example of section A.4, and the results obtained are exactly the same.

In this point, the selected fault can be finally approximated and removed from the candidate fault list. The approximation is performed over the over-approximate circuit. As a result of the approximation of the fault $PI0/1$ the circuit of the Figure A.54b is generated, where the output $PO0$ has been tied to a logic constant.

After performing the fault approximation, all the remaining faults are implied in this new approximate circuit. This time, an over-approximate fault ($PO0/1$) is detected to be redundant, which is removed from the candidate fault list along with $PI0/1$. On the other hand, the under-approximate faults are implied to identify new substitution candidates. Because the over-approximate circuit has been modified, the substitution candidates found in the previous iteration are no longer valid.

Some of the under-approximate faults do not present substitution candidates in the current iteration. This time, faults $n7/1$, $n9 \rightarrow PO0/1$ and $PO0/0$ do not have a valid propagation path, and therefore they do not generate a valid set of assignments which allow identifying any substitution candidate. For its part, the fault $PO1/0$ only generates an assignment at the node $PO1$, which is not sufficient to infer any valid substitution candidate.

With the last approximated fault, the multiple fanout point at the output of the node $n9$ has been finally resolved, and therefore the remaining fault $n9 \rightarrow PO1/1$ can be implied in the search for substitution candidates. The assignments $n9=0$ and $n11=1$ form the SMA of this fault, as shown in the Figure A.95a. These assignments are then propagated by direct implication, as shown in the Figure A.95b. From the inferred set of assignments, $PI1=1$, $n9=0$, $n11=1$ and $PO1=1$ are excluded due to the constraints which conform a valid substitution candidate. However, the assignment $PI4=0$ fulfils all those constraints, and has the same value than the assignment at the fault location. As result, the direct substitution from $PI4$ to $PO1$ is identified as a valid candidate.

This new substitution has associated the virtual fault $PI4 \rightarrow PO1/1$. The probability of this fault is now computed. First, the initial SMA is inferred, which in this case is formed by the assignments $PI4=0$, $n9=1$ and $n11=1$, as shown in the Figure A.96a. These assignments are propagated through the circuit as indicated in the Figure A.96b. The final J-SMA of the fault contains the assignments $PI1=0$ and $PI4=0$. The probability of this set of conditions, which is equal to the probability of the substitution fault,

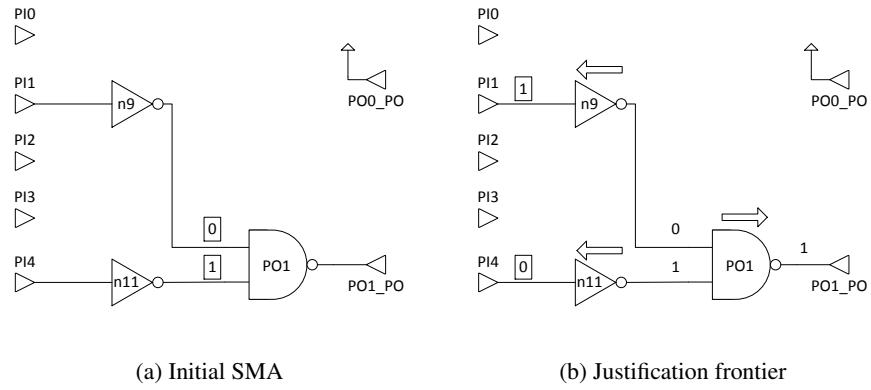


Figure A.95: Implication of fault $n9 \rightarrow PO1/1$ in the over-approximate circuit - fourth approximation

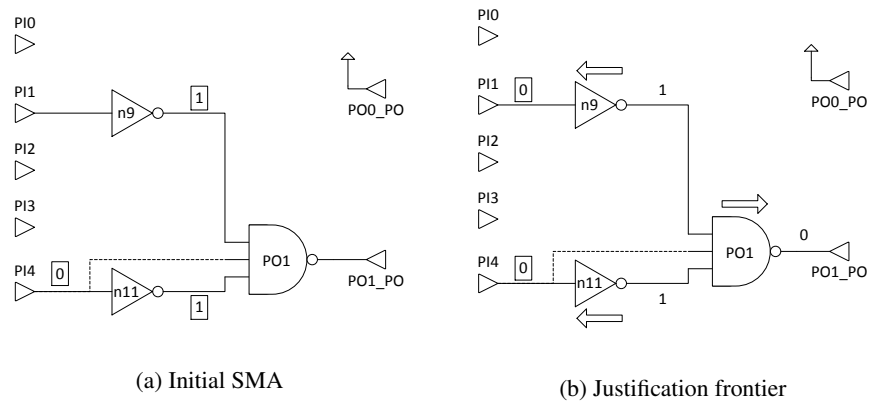


Figure A.96: Implication of substitution candidate $PI4 \rightarrow PO1/1$ - fourth approximation

is $P(PI4 \rightarrow PO1/1) = P(\overline{PI1}) \cdot P(\overline{PI4}) = 0.5^2 = 0.25$.

Finally, the fault n11/1 is implied again in the over approximate circuit. The last logic transformation of the over-approximation does not alter the set of assignments inferred for the fault n11/1 with respect to the previous iteration of the algorithm. Therefore, the set of assignments which appear in the Figure A.93b still apply here. As a consequence, the direct substitution from PI1 to PO1 is again identified as a valid candidate. This logic transformation has associated the substitution fault $PI1 \rightarrow PO1/1$, whose probability has to be computed. The implication of this fault in the new over-approximate circuit is analogous to the implication performed in the previous iteration (see Figure A.94 for details), with the same set of inferred assignments and the same J-SMA. In conclusion, the probability of this substitution fault is still $P(PI1 \rightarrow PO1/1) = 0.25$.

Both identified substitution candidates, $PI1 \rightarrow PO1/1$ and $PI4 \rightarrow PO1/1$, are included in the list of candidate faults along with all the remaining circuit faults, whose updated probabilities appear in the Table A.10. Among all these faults, the one with the lowest probability value is selected for the next iteration of the algorithm. In this case, the over-approximate fault PI1/1, with a probability equal to 0, is selected.

5th approximation: PI1/1

The new chosen approximation candidate, PI1/1, is implied in the over-approximate circuit in order to obtain its approximation condition. Then, the probability of all the remaining over-approximate faults is updated, including PI1/1 itself. And finally, the global EP is updated. These steps are analogous to the fifth iteration in the example with dynamic testability measures, and the results obtained perfectly match, so there is no need of replicating the computations here.

After all the fault probabilities have been updated, the fault PI1/1 can be finally approximated in the over-approximate circuit. As a result, the trivial approximation of Figure A.59b is generated, where the whole circuit logic has been removed, and both outputs are tied to logic constants. In this situation, there are no valid propagation paths for any of the faults in the circuit. Therefore, all the remaining over-approximate faults are considered as already approximated, and no substitution candidates can be identified. On the other hand, the under-approximate circuit is still an exact copy of the original c17 benchmark.

In this point, only the under-approximate faults remain as eligible candidates. The table A.12 collects the updated probability values of all these faults. Among them, the fault with the lowest probability is chosen to be approximated. But in this case, there are four possible candidates: $PI2 \rightarrow n8/1$, n7/1, $n9 \rightarrow PO1/1$ and n11/1, all of them with a probability equal to 0.1875. Among them, the logic transformation which produces the largest area savings is chosen, which limits the candidates to n7/1 and n11/1. Among them, finally n7/1 is arbitrarily chosen for the next iteration.

6th approximation: n7/1

The fault n7/1 is the first under-approximate fault to be approximated. In this case, the approximation condition of this fault is computed in the under-approximate circuit,

which is still an exact replica of the original circuit. Then, the probabilities of all the under-approximate faults are updated, including n7/1 itself. An finally, the EP is updated according to the accumulated probability changes. All these steps are identical to the sixth iteration on the example with dynamic testability measures, and they provide exactly the same results.

Now the selected fault can be approximated. As a result, the under-approximation is transformed into the circuit of Figure A.63b, where the node n7 has been left dangling and is removed, and the node PO0 is changed into an inverter. From now, the fault n7/1 is removed from the candidate fault list.

After the fault n7/1 has been approximated, every remaining fault is implied in this new under-approximate circuit. This time, the over-approximate faults are implied in order to identify substitution candidates, while the under-approximate faults are implied to detect additional faults which may have become redundant. In this iteration, no additional faults are detected as approximated.

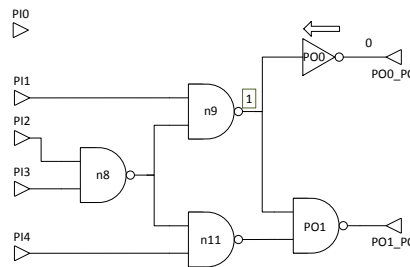


Figure A.97: Implication of fault PO0/1 in the under-approximate circuit - sixth approximation

For most of the under-approximate faults, no substitution candidates can be found. Faults $n8 \rightarrow n9/1$ and $n8 \rightarrow n11/1$ are located at a multiple fanout point, and therefore any node substitution performed over them would produce no area savings. For its part, the implication of the fault PO0/1 generates the set of assignments of Figure A.97. $PO0=1$ corresponds to the fault location, and $n9=1$ is its immediate input, so none of them meet the necessary constraints.

With respect to the fault PO1/1, it is implied in the under-approximate circuit with the only assignment $PO1=0$ as the initial SMA. Such assignment is justified by $n9=1$ (which is propagated to PO0) and $n11=1$, as shown in the Figure A.98. From this set of assignments, PO1 corresponds to the fault location, while n9 and n11 are immediate inputs of PO1, and therefore all of them are excluded. However, the assignment $PO0=0$ remains as a valid candidate. Both PO0 and PO1 are assigned to the same logic value and have the same parity (because the primary outputs are always assigned an even parity. See section 3.2 for details). Therefore, the direct substitution from PO0 to the primary output PO1 constitutes a valid candidate.

This new substitution candidate has associated the virtual fault $PO0 \rightarrow PO1_PO/1$, whose probability has to be computed. First, the initial SMA of Figure A.99a is in-

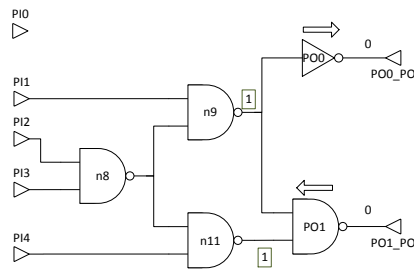
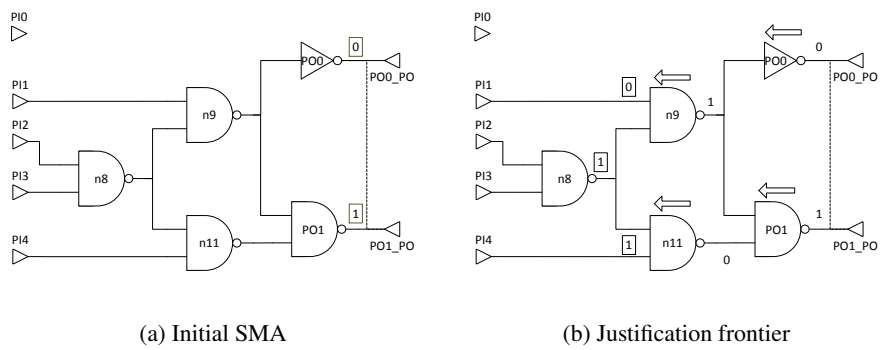


Figure A.98: Implication of fault PO1/1 in the under-approximate circuit - sixth approximation



(a) Initial SMA

(b) Justification frontier

Figure A.99: Implication of substitution candidate $PO0 \rightarrow PO1_PO/1$ - sixth approximation

ferred, which contains the assignments $PO0=0$ and $PO1=1$. Then this SMA is propagated by direct implication, as shown in the Figure A.99b. The assignment $PO0=0$ is justified by $n9=1$, which at the same time forces $n11=0$ in order to justify $PO1=1$. Then, $n11=0$ becomes justified by $PI4=1$ and $n8=1$, and consequently $n9=1$ must be justified by imposing $PI1=0$. In the end, the J-SMA is formed by the MAs $PI1=0$, $PI4=1$ and $n8=1$. The probability of this J-SMA, and the substitution fault by extension, is computed as $P(PO0 \rightarrow PO1_PO) = P(\overline{PI1}) \cdot P(PI4) \cdot P(n8) = 0.5^2 \cdot 0.75 = 0.1875$.

The identified substitution fault is included in the list of eligible candidates along with all the remaining under-approximate faults, whose updated probabilities are collected in the Table A.14. The fault with the lowest probability among all these candidates is selected next. In this case, there are up to four possible faults: $PI2 \rightarrow n8/1$, $n9 \rightarrow PO1/1$, $n11/1$ and the substitution fault $PO0 \rightarrow PO1_PO$, all of them with a probability of 0.1875. Among them, the fault which produces the largest area savings is finally selected, which turns out to be the substitution fault, $PO0 \rightarrow PO1_PO$.

7th approximation: substitution fault $PO0 \rightarrow PO1_PO/1$

Whenever a new fault is selected to be approximated, first it has to be implied in the correspondent approximate circuit in order to obtain its approximation condition. In the case of the selected fault in the current iteration (the substitution fault $PO0 \rightarrow PO1_PO/1$), such implication has already been performed in the previous iteration in order to compute the probability of the fault (see Figure A.99). The J-SMA of this fault contains the MAs $PI1=0$, $PI4=1$ and $n8=1$. Therefore, the approximation condition associated to this fault is $A_{PO0 \rightarrow PO1_PO} = \overline{PI1} \cdot PI4 \cdot n8$ with respect to the output $PO1$. This fault cannot propagate through $PO0$, and therefore the approximation condition with respect to that output is null.

Next, the intersection of this approximation condition with each remaining under-approximate fault is computed in the original circuit, with the idea of updating the probability of every fault. It must be noted that the approximation condition inferred for the substitution fault $PO0 \rightarrow PO1_PO/1$ is identical to the one computed during the seventh iteration of the example with dynamic testability measures for the fault $n11/1$. Therefore, the fault probabilities are updated exactly with the same results than in the correspondent iteration of the example of section A.4, summarized in the Table A.15. And in consequence, the global EP is updated to the same value as well.

After updating the fault probabilities, the approximation can be finally performed. This time the logic transformation is a node substitution, and therefore a new connection is created from the node $PO0$ to the primary output $PO1$. At the same time, the former connection from the node $PO1$ is removed as shown in the Figure A.100a. This logic transformation leaves dangling the nodes $PO1$ and $n11$, which can be safely removed, thus obtaining the equivalent circuit of Figure A.100b. By comparing this approximation with the corresponding one generated in the example with dynamic testability measures (Figure A.69b), finally the benefits of the node substitution approach can be observed, as here one additional logic gate has been saved with respect to the classic approach.

Then, all the remaining faults are implied in this new under-approximate circuit. Among the under-approximate faults, $n11/1$, $n9 \rightarrow PO1/1$ and $PO1/0$ have become re-

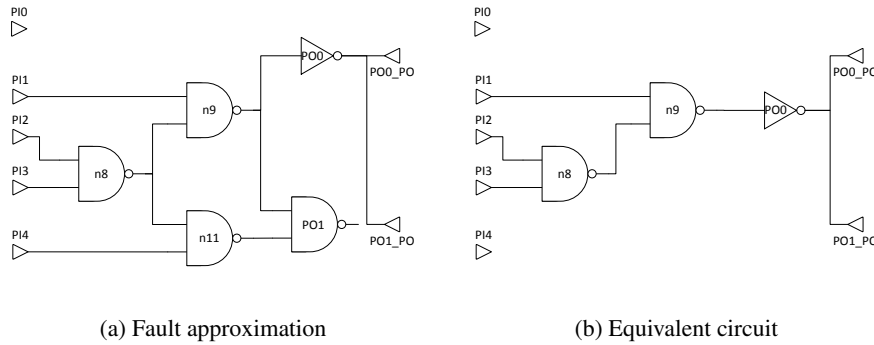


Figure A.100: Approximation of substitution fault $PO0 \rightarrow PO1_PO/1$

dundant, and therefore they are removed from the list of candidate faults. This means that the probabilities of these faults are not updated any more and therefore they no longer contribute to the EP, even if their current probabilities are greater than 0 (such is the case of the faults $n9 \rightarrow PO1/1$ and $PO1/0$). As a consequence, an underestimation of the EP may be produced in subsequent iterations.

Therefore, how the EP should be computed from now on to compute the correct result? In truth, what is happening is that the redundant circuit faults has been transformed into others. The fault $n9 \rightarrow PO1/1$ is now represented by the fault $n9 \rightarrow PO0/1$ itself. Therefore, when updating the EP in future iterations, the probability of the fault $n9 \rightarrow PO0/1$ should be counted twice for a realistic EP estimation: once with the full probability update (representing the contribution of $n9 \rightarrow PO0/1$ itself), and another time with just the probability of the input combinations which allow propagation to both outputs (representing the contribution of the fault $n9 \rightarrow PO1/1$). With respect to the fault $PO1/0$, it has been transformed into the fault stuck-at 0 in the new connection (i.e., into the fault $PO0 \rightarrow PO1_PO/0$). Therefore, the fault $PO0 \rightarrow PO1_PO/0$ should be included in the list of circuit faults. However, this mechanism of replacing the redundant faults with their equivalents in the new approximate circuit has not been implemented in the proposed node substitution approach, and therefore some imprecisions in the estimation of the EP may be performed from now on.

On the other hand, the over-approximate faults are implied in the new over-approximate circuit in order to find new substitution candidates. However, faults $n8 \rightarrow n11/1$ and $PO1/1$ have become redundant and therefore they do not generate a valid set of assignments. With respect to the fault $PO0/1$, exactly the same assignments inferred in the previous iteration are obtained here (see Figure A.97), which do not meet the constraints for a valid substitution candidate.

Finally, the fault $n8 \rightarrow n9/1$ is implied in the search for substitution candidates, because with the last approximation the multiple fanout point at the output of the node $n8$ has been removed. The initial SMA for this fault is formed by the assignments $PI1=1$ and $n8=0$, as shown in the Figure A.101a. These assignments are then propagated by direct implication. The assignment $n8=0$ is justified by $PI2=1$ and $PI3=1$. At the same time, $n8=0$ propagates to $n9=1$, and subsequently to both primary outputs, as shown in

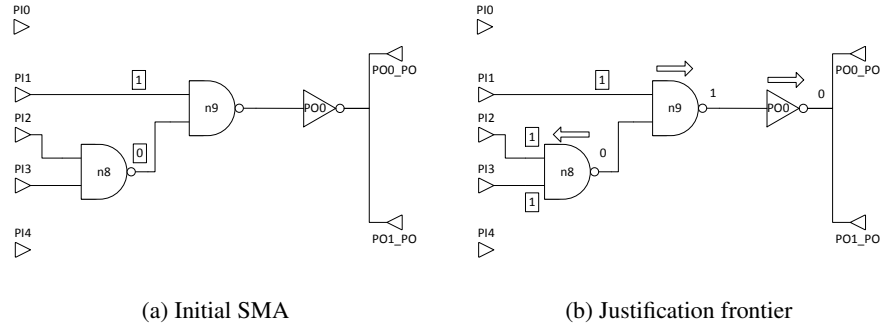


Figure A.101: Implication of fault $n8 \rightarrow n9/1$ in the under-approximate circuit - seventh approximation

Under-approx fault	Probability	Over-approx fault	Probability
$PI2 \rightarrow n8/1$	0.125	$PI0/0$	-
$PI3/1$	0.1875	$PI1/1$	-
$n7/1$	-	$PI2 \rightarrow n7/1$	-
$n9 \rightarrow PO0/1$	0.2812	$PI4/1$	-
$n9 \rightarrow PO1/1$	-	$n8 \rightarrow n9/1$	-
$n11/1$	-	$n8 \rightarrow n11/1$	-
$PO0/0$	0.3437	$PO0/1$	-
$PO1/0$	-	$PO1/1$	-

Table A.23: Fault probabilities after $PO0 \rightarrow PO1_PO/1$ approximation

the Figure A.101b. But none of these assignments point to a valid substitution candidate, because each one of them either belong to the output cone of the fault $n8 \rightarrow n9/1$, or is an immediate input of some of the nodes in the output cone of the fault.

In conclusion, there are no substitution candidates found. The selection of the next fault is performed then among the remaining under-approximate faults. The Table A.23 shows the current probability of all the circuit faults, without including the new fault appeared due to the node substitution just performed. The dashes denote those faults which have become redundant, and therefore they are not eligible. Among the candidate faults, $PI2 \rightarrow n8/1$ is next selected as the fault with the lowest probability.

8th approximation: $PI2 \rightarrow n8/1$

A new iteration of the approximation generation algorithm begins. The first step consists in obtaining the approximation conditions of the last selected fault, $PI2 \rightarrow n8/1$. This step is analogous to the computation of approximation conditions for the same fault in the eighth iteration of the example with dynamic testability measures. To sum up, the approximation condition $A_{PI2 \rightarrow n8/1} = PI1 \cdot \overline{PI2} \cdot PI3$ is obtained with respect

to both outputs.

Then, the probabilities of all the remaining under-approximate faults are updated with the usual method. It must be noted that the faults $n9 \rightarrow PO1/1$ and $PO1/0$ are now considered as approximated and therefore they are not updated. For the rest of the remaining under-approximate faults, their probabilities are updated exactly as in the eighth iteration of the example of section A.4, because the approximation conditions of the last approximated fault are identical in both examples.

Fault	Justification frontier			Probability				
				Former	Unmasked		Updated	
	PO0	PO1	PO0		PO1			
$PI2 \rightarrow n8/1$	$PI1$	$\overline{PI2}$	$PI3$	-	0.125	0.125	0	0
$PI3/1$	-	-	-	-	0.1875	0	0	0.1875
$n9 \rightarrow PO0/1$	$PI1$	$\overline{PI2}$	$PI3$	-	0.2812	0.125	0	0.1562
$PO0/0$	$PI1$	$\overline{PI2}$	$PI3$	-	0.3437	0.125	0	0.2187

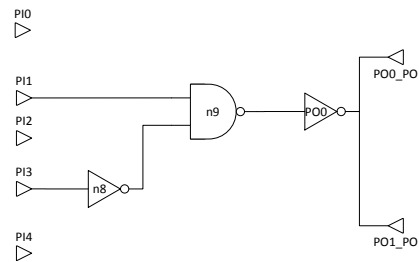
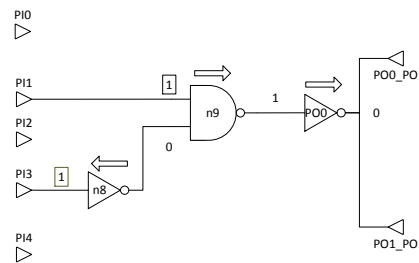
Table A.24: Summary of $PI2 \rightarrow n8/1$ approximation

The Table A.24 shows the results of the probability updating step. It contains, from left to right, the J-SMAs resulting from re-implying each fault with the approximation conditions of the fault $PI2 \rightarrow n8/1$ with respect to both outputs, the fault probability at the beginning of the iteration, the probability of the unmasked input vectors for each fault with respect to each output, and finally the updated probability values. Because the faults $n9 \rightarrow PO1/1$ and $PO1/0$ have not been considered in this example, the accumulated EP is updated with a different value than in the example with dynamic testability measures

$$\begin{aligned}
 EP_8 &= EP_7 + \frac{\sum_i P(f_i \cap A_{PI2 \rightarrow n8/1} \cap \overline{f_i \cap A_F})}{n} = \\
 &= 17.7734\% + \frac{3 \cdot 0.125}{16} = 20.1172\%
 \end{aligned}$$

However, an underestimation of the EP has been produced here. The result does not coincide with the EP estimation obtained in the corresponding iteration of the example with dynamic testability measures (which is equal to 21.2891%), despite that in this point, the approximate circuits generated are functionally equivalent (it can be seen by comparing the circuits of Figures A.76 and A.102). In addition, the EP estimated in the example of section A.4 coincides with the real EP value which appears in the Table A.22. This is due to not considering the new faults which appear as a consequence of the node substitution previously performed.

Now, the fault $PI2 \rightarrow n8/1$ can be finally approximated and removed from the list of candidate faults. As a result, the under-approximate circuit of the Figure A.102 is generated, where the node $n8$ has been transformed into an inverter. Compared with the corresponding approximate circuit in the example with dynamic testability measures (in the Figure A.76), it can be seen that the circuit generated in this example is more optimized, using one logic gate less than with the classic approach.

Figure A.102: Approximation of fault $PI2 \rightarrow n8/1$ Figure A.103: Implication of fault $PO0/1$ in the under-approximate circuit - eighth approximation

Once the new under-approximate circuit has been generated, all the remaining faults are implied in this new circuit. On the one hand, none of the remaining under-approximate faults is identified as redundant. On the other hand, the remaining over-approximate faults ($n8 \rightarrow n9/1$ and $PO0/1$) are implied to detect potential substitution candidates. The set of assignments obtained from the implication of the fault $n8 \rightarrow n9/1$ are shown in the Figure A.103. With respect of the fault $PO0/1$, the same assignments inferred in the previous iterations of the example are obtained ($n9=1$ and $PO0=0$, as in the Figure A.97). None of these assignments meets the constraints that define a valid substitution candidate.

Under-approx fault	Probability	Over-approx fault	Probability
$PI2 \rightarrow n8/1$	-	$PI0/0$	-
$PI3/1$	0.1875	$PI1/1$	-
$n7/1$	-	$PI2 \rightarrow n7/1$	-
$n9 \rightarrow PO0/1$	0.1562	$PI4/1$	-
$n9 \rightarrow PO1/1$	-	$n8 \rightarrow n9/1$	-
$n11/1$	-	$n8 \rightarrow n11/1$	-
$PO0/0$	0.2187	$PO0/1$	-
$PO1/0$	-	$PO1/1$	-

Table A.25: Fault probabilities after $PI2 \rightarrow n8/1$ approximation

Because no substitution candidates have been found, the selection of the next approximated fault is limited to the remaining under-approximate faults, which are collected in the Table A.25. Among them, the fault with the lowest probability is selected, which turns out to be $n9 \rightarrow PO0/1$.

9th approximation: $n9 \rightarrow PO0/1$

With the new fault $n9 \rightarrow PO0/1$ selected, its approximation conditions are computed as usual. With this goal, the fault $n9 \rightarrow PO0/1$ is implied in the under-approximate circuit. The implication process is analogous to the one performed in the ninth iteration of the example with dynamic testability measures for the fault $n9 \rightarrow PO1/1$ (in the Figure A.77b). The approximation condition has therefore the same assignments, $A_{n9 \rightarrow PO0/1} = PI1 \cdot \overline{PI3}$, although in this example the approximation condition applies to both outputs instead of just one of them.

Fault	Justification frontier		Probability			
			Former	Unmasked		Updated
	PO0	PO1		PO0	PO1	
$PI3/1$	$PI1 PI2 PI3$	-	0.1875	0.125	0	0.0625
$n9 \rightarrow PO0/1$	$PI1 \overline{PI3} n7$	-	0.1562	0.1875	0	-0.0313
$PO0/0$	$PI1 \overline{PI3}$	-	0.2187	0.25	0	-0.0313

Table A.26: Summary of $n9 \rightarrow PO0/1$ approximation

Next, the probabilities of the remaining under-approximate faults are updated. In this case, the approximation condition of the fault $n9 \rightarrow PO0/1$ combines the approximation conditions of the faults approximated in the two last iterations of the example with dynamic testability measures. Therefore, the fault implications and probability computations performed in the iterations 9 and 10 of section A.4 are repeated here, with the exception of the faults $n9 \rightarrow PO1/1$ and $PO1/0$ which have been already approximated. In summary, the set of inferred assignments and the updated probability values are collected in the Table A.26.

Once all the fault probabilities have been updated, it is time to update the global EP by accumulating the average variation of fault probabilities. Thus,

$$EP_9 = EP_8 + \frac{\sum_i P(f_i \cap A_{n9 \rightarrow PO0/1} \cap \overline{f_i \cap A_F})}{n} =$$

$$= 20.1172\% + \frac{0.125 + 0.1875 + 0.25}{16} = 23.6328\%$$

Again, a discrepancy with the estimated EP in the example with dynamic testability measures can be observed, which is again due to an incomplete fault list from the moment when the node substitution was applied.

Finally, the fault $n9 \rightarrow PO0/1$ can be approximated in the under-approximate circuit, generating the full trivial approximation as a result, where there is no additional logic, and the original circuit is completely unprotected. In this situation, all the propagations paths become blocked, so all the remaining faults are automatically approximated. Because there are no faults left, the approximation generation algorithm finishes here if not had stopped previously as a consequence of reaching a given EP target, specified in advance.

A.6 Approximation for FPGA implementation

The last example in this appendix corresponds to the case of approximation generation for FPGA, which is addressed in the section 6.2 of the thesis. In summary, this approach applies the approximation generation method based in static testability measures to a circuit structure based in LUTs, which is the most typical structure in FPGA technologies. This imposes some constraints in the set of faults which can be approximated, because now some logic transformations may cause a degradation of the target logic function without obtaining any benefits in terms of area overhead, i.e., not reducing either the number or size of LUTs, which is not desirable at all. These constraints are complemented with a procedure for direct approximation of binate faults, which is achieved by means of approximating two complimentary unate faults simultaneously. Unfortunately, this characteristic cannot be seen in this example, because c17 is a fully unate circuit. Readers are referred to the examples in section 6.2.3 about this topic.

Figure A.104 shows the representation of the c17 benchmark as a LUT structure. This is overlapped with the already known logic gate structure in order to see the correspondence between them. In the FPGA extension to circuit approximation, only faults located in the inputs or outputs of LUTs can be approximated. This means that faults

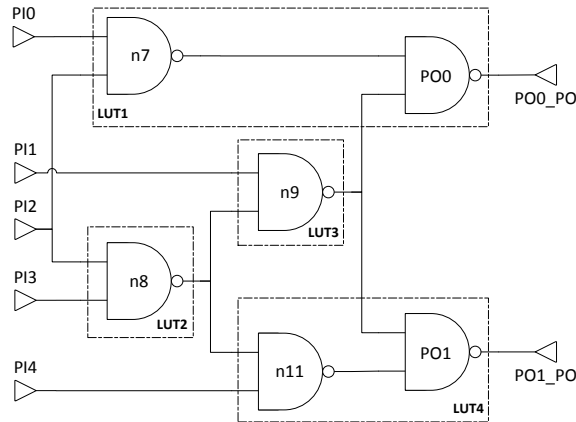


Figure A.104: LUT partitioning of c17 benchmark

in the outputs of nodes $n7$ and $n11$, which are internal to a LUT, are not eligible for approximation. In this example, the LUT structure has been obtained from the logic gate structure, and not in the contrary. This, as discussed in section 6.2.2, is usually suboptimal in terms of FPGA resource utilization, but it is applied here in order to reuse the testability measures from the example in section A.3, with the idea of facilitating the comparison among both examples. Therefore, the testability measures appeared in the Table A.1 for the approximation example in section A.3 are exactly the same in which this example is based.

The faults in the fault list are classified depending on if they produce an under- or an over-approximation, or if they are not eligible for approximation, as follows. This classification is almost identical to the example in section A.3 with the only exception of faults $n7/1$ and $n11/1$, which become not eligible because they are located inside of a LUT in the new circuit structure:

- Under-approximation faults: $PI2 \rightarrow n8/1$, $PI3/1$, $n9 \rightarrow PO0/1$, $n9 \rightarrow PO1/1$, $PO0/0$ and $PO1/0$.
- Over-approximation faults: $PI0/1$, $PI1/1$, $PI2 \rightarrow n7/1$, $PI4/1$, $n8 \rightarrow n9/1$, $n8 \rightarrow n11/1$, $PO0/1$ and $PO0/1$.
- Not eligible faults: $PI2/0$, $PI2/1$, $n8/0$, $n8/1$, $n9/0$ and $n9/1$ correspond to stem lines. In addition, faults $n7/1$ and $n11/1$ fall inside a LUT.

Although it is true in this case that by approximating internal faults $n7/1$ or $n11/1$ their respective LUTs would reduce the number of inputs, allowing some resource savings, this fact cannot be guaranteed in the general case, and because of that the proposed approach forbids approximating any fault internal to a LUT.

From now on, the successive phases of the approximation generation process are presented, starting with the full TMR solution, and progressively increasing the testability threshold up to obtaining the trivial approximation.

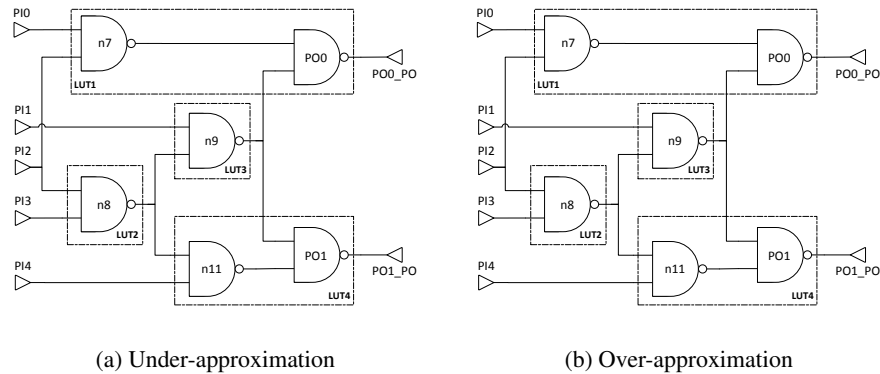


Figure A.105: Approximate circuits for a 0 threshold - Pure TMR

Setting any testability threshold equal or lower than the sensitivity of the less testable fault in the target circuit will generate a pure TMR scheme, because no faults are approximated. In the current example, this limit value corresponds to the fault $PI2 \rightarrow n7/1$, which has a testability of 0.114. The Figure A.105 shows the approximate circuits resulting from a testability threshold under 0.114, which are obviously exact replicas of the original c17 benchmark.

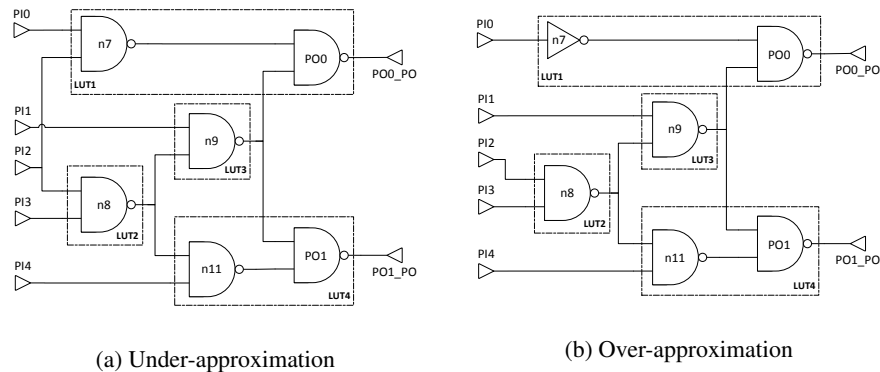


Figure A.106: Approximate circuits for a 0.12 threshold

With a testability threshold above 0.114, at least one fault is approximated and some deviations from the pure TMR scheme start appearing. The first selected value is 0.12, which makes just fault $PI2 \rightarrow n7/1$ to be approximated. The approximation of fault $PI2 \rightarrow n7/1$ affects the over-approximate circuit, according to the previous classification of faults. On the other hand, the under-approximate circuit presents no changes. The

resulting approximate circuits, shown in the Figure A.106, are the same than in the basic example with static testability measures from section A.3 for the same testability threshold. The approximation of the fault $PI2 \rightarrow n7/1$ allows simplifying the logic gate $n7$ from a NAND gate to an inverter, thus reducing the number of inputs of LUT number 1. Although the total number of LUTs has not been reduced yet, this transformation may achieve some marginal benefits in terms of the number of interconnections and configuration bits required.

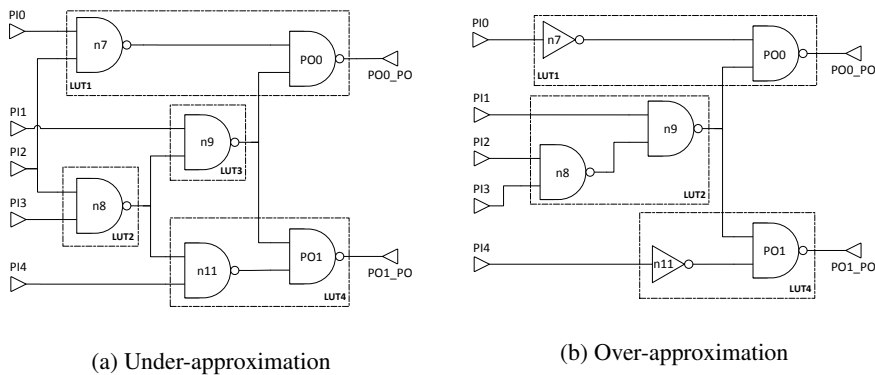


Figure A.107: Approximate circuits for a 0.135 threshold

The next selected value is 0.135, resulting in faults $PI2 \rightarrow n7/1$ and $n8 \rightarrow n11/1$ being approximated. It must be noted that faults whose testability equals the threshold value are not approximated, which is the case of the fault $n8 \rightarrow n9/1$. The parallelism with the original example of section A.3 is clear, and therefore it is not surprising that the resulting circuits from both approaches (see Figures A.107 and A.5) are identical for this testability threshold. In this case, the approximation of the fault $n8 \rightarrow n11/1$ allows not only reducing the size of LUT4, but also eliminating the multiple fanout point at the output of node $n8$, which in turn allows merging LUTs number 2 and 3 into a single LUT. This is the most interesting kind of logic transformations, those which effectively reduce the number of LUTs in the circuit.

The next step corresponds to a testability threshold of value 0.14. Similarly to the example of section A.3, the fault $n8 \rightarrow n9/1$ is now included in the group of approximated faults. The resulting approximate circuits are shown in the Figure A.108. The joint approximation of faults $n8 \rightarrow n9/1$ and $n8 \rightarrow n11/1$ results in LUT2 becoming dangling, and therefore it can be removed. In addition, LUT3 is simplified to just an inverting function. But the resulting over-approximate circuit can be further simplified by merging the LUT3 into the logic functions of the two other LUTs, resulting in the circuit of Figure A.108b with just two LUTs, one for each primary output.

Up to this point, the approximation example for FPGA coincided exactly with the example of the original approach with static testability measures. But from now on, both approaches diverge. The next approximated fault, according to the original approach, is $n11/1$. But this fault is now not eligible, because it is located inside of a LUT. Therefore, the testability threshold grows up to including the next valid fault,

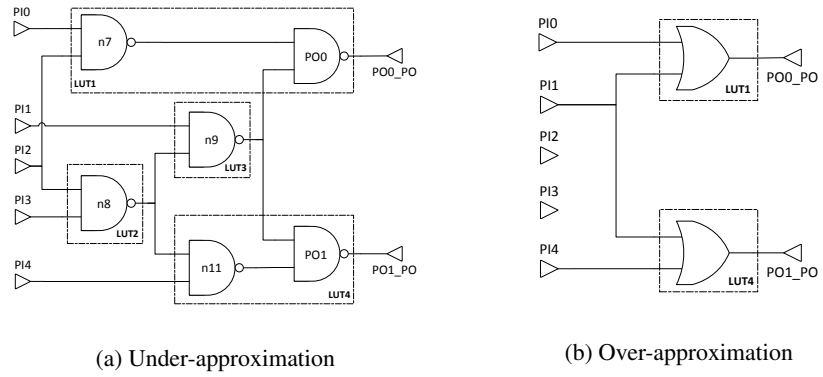


Figure A.108: Approximate circuits for a 0.14 threshold

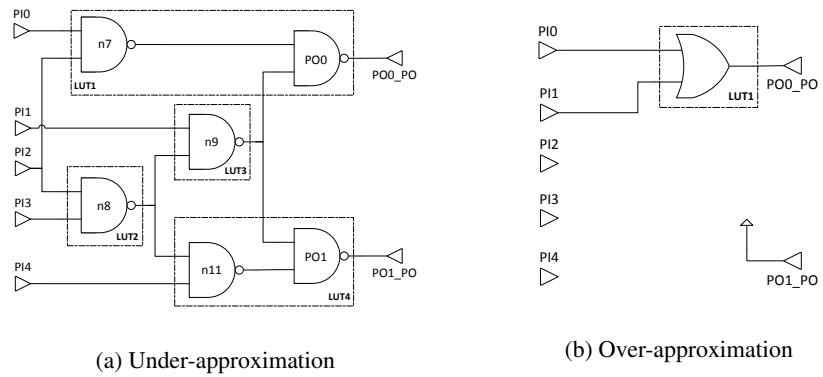


Figure A.109: Approximate circuits for a 0.185 threshold

which is $PI4/1$. Let us establish a testability threshold equal to 0.185. This fault, combined with the effects of the previous approximated faults, makes that primary output PO1 will be tied to a logic constant in the over-approximate circuit, reducing it to just one LUT, as it can be seen in the Figure A.109b. This, in turn, allows simplifying the logic in charge of voting, as discussed in section 4.2. On the other hand, the under-approximate circuit is still a exact replica of the target c17 benchmark, as no faults of that type have been approximated yet, contrary to what happens in the example of section A.3 for the same testability threshold.

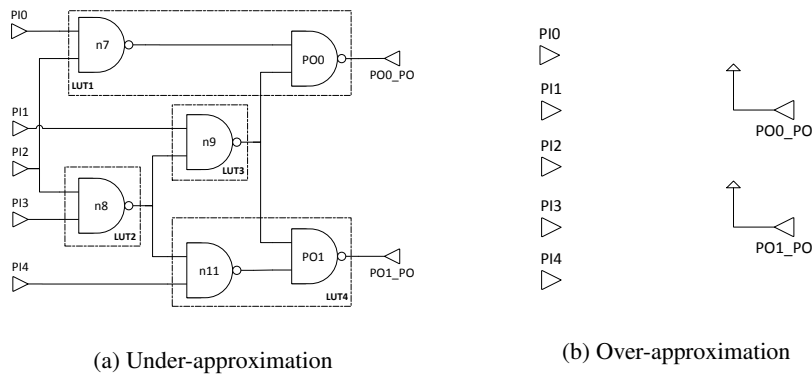


Figure A.110: Approximate circuits for a 0.19 threshold

The next selected testability threshold is 0.19, which results in the fault $PI0/1$ to be included in the list of approximated faults. By simplifying all the logic constants in the over-approximate circuit it is obtained the trivial over-approximation, and this holds for all the subsequent phases of the approximation process. While, the under-approximate circuit is still a copy of the original c17, as shown in Figure A.110

A testability threshold of 0.197 comes next, which makes the fault $PI3/1$ to be approximated along with all the previous faults. In the original example, the fault $n7/1$ became approximated as well, but in this example it is not taken into account because it is internal to a LUT. The fault $PI3/1$ is the first under-approximate fault being approximated in this example, and as a result the logic function of LUT3 is transformed into a simple inverter. Moreover, this LUT can be then merged into both of the subsequent LUTs (numbers 3 and 4), thus reducing the number of LUTs in the under-approximate circuit, as shown in the Figure A.111a.

Following with a testability threshold of 0.2, where the fault $n9 \rightarrow PO1/1$ passes to be included in the list of approximated faults. Note that the fault $PI2 \rightarrow n8/1$ is not approximated yet, because its probability is not lower than the selected testability threshold. With this new approximated fault, the multiple fanout point right after the node $n9$ is removed, allowing LUT3 to be merged with LUT1. And in the same way as in the previous step, LUT2 is reduced to just one input, and therefore it can be easily merged in both remaining LUTs. As result, the generated under-approximate circuit presents just 2 LUTs, as it can be seen in the Figure A.112.

Next a threshold of value 0.25 is selected. The fault $PI2 \rightarrow n8/1$ is added to the

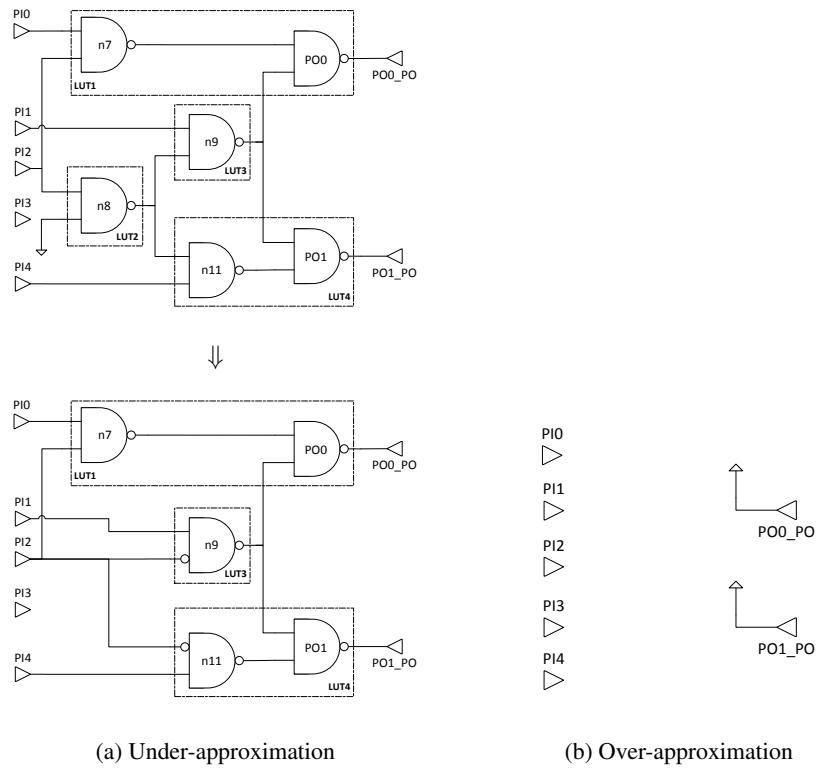


Figure A.111: Approximate circuits for a 0.197 threshold

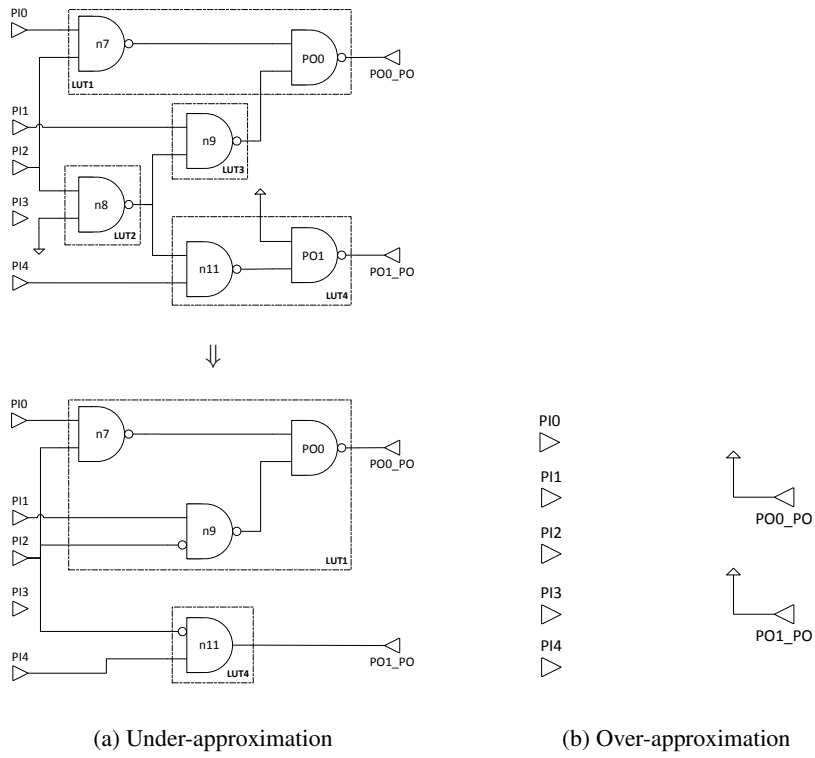


Figure A.112: Approximate circuits for a 0.2 threshold

approximated fault list. By simplifying all the logic constants, the approximate circuits of the Figure A.113 are obtained. In the case of the under-approximation, the result is that the primary output PO1 now receives a constant assignment, while the other output is determined by just a two input LUT. It must be noted the difference with respect to the original example (in section A.3), where at this point the trivial approximation was obtained already.

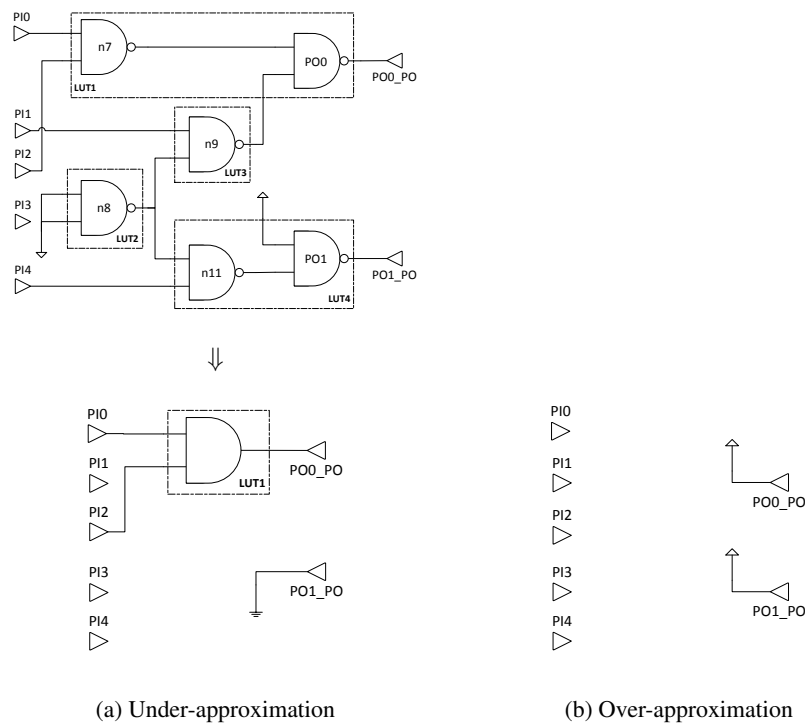


Figure A.113: Approximate circuits for a 0.25 threshold

Finally, in order to obtain the trivial approximation of Figure A.114, the testability threshold has to be raised up to 0.6. Although there are other valid faults with lower probabilities, namely $PI1/1$, $n9 \rightarrow PO0/1$, $PO0/1$, $PO1/1$ and $PO1/0$, all of them are redundant with respect to the set of previous approximated faults, and therefore they do not modify the resulting circuits from the previous step. Until fault $PO0/0$ is not included in the list of approximated faults, the trivial approximation is not generated.

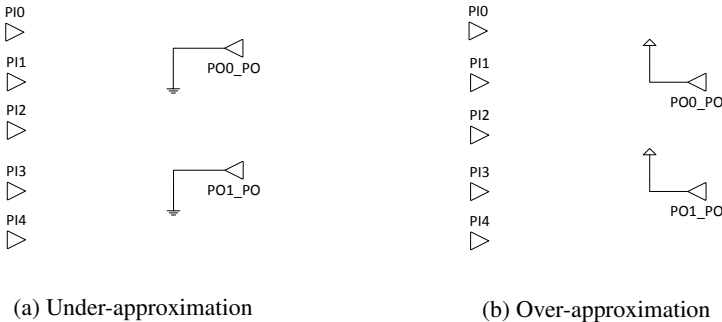


Figure A.114: Approximate circuits for a 0.6 threshold