TEST TECHNIQUES FOR APPROXIMATE DIGITAL CIRCUITS

Marcello Traiola

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En SYAM – Systèmes Automatiques et Micro-Electroniques

École doctorale : I2S – Information, Structures et Systèmes

Unité de recherche : LIRMM – Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier

TECHNIQUES DE TEST POUR CIRCUITS DIGITAUX BASÉS SUR LE CALCUL APPROXIMATIF

TEST TECHNIQUES FOR APPROXIMATE DIGITAL CIRCUITS

Présentée par Marcello TRAIOLA

Le 25 septembre 2019

Devant le jury composé de

Alberto BOSIO	Professeur à l'École Centrale de Lyon, INL, Lyon	Directeur de thèse
Patrick GIRARD	Directeur de recherche au CNRS, LIRMM, Montpellier	Co-Directeur de thèse
Arnaud VIRAZEL	Maître de conférence à l'Université de Montpellier, LIRMM, Montpellier	Co-encadrant
Olivier SENTIEYS	Directeur de recherche a l'INRIA, IRISA, Rennes	Rapporteur
Matteo SONZA REORDA	Professeur au Politecnico di Torino, Italie	Rapporteur
Lirida Alves de Barros NAVINER	Professeur à Telecom ParisTech, Paris	Présidente

UNIVERSITÉ DE MONTPELLIER

"No man ever steps in the same river twice, for it's not the same river and he's not the same man. "

Heraclitus

UNIVERSITY OF MONTPELLIER

Abstract

Graduate School for Information, Structures and Systems (I2S)
Laboratory of Computer Science, Robotics, and Microelectronics of Montpellier (LIRMM)

Doctor of Philosophy

Test Techniques for Approximate Digital Circuits

by Marcello TRAIOLA

Approximate Computing (AxC) is increasingly emerging as a new design paradigm to produce more efficient computation systems by meticulously reducing the computation quality. In particular, AxC has been successfully applied to Integrated Circuits (ICs), in the last years. Hence, concerning the test of such new class of ICs, namely Approximate Integrated Circuits (AxICs), new challenges – as well as new opportunities – have emerged. In this thesis, we provide a thorough analysis of issues related to testing procedures for AxICs and present innovative techniques to deal with them. We resort to an illustrative example having the twofold aim of: (i) guiding the reader through the AxIC testing challenges and (ii) illustrating the proposed solutions to correctly overcome them, while suitably taking advantage of opportunities coming from approximation. We analyze experimentally all the proposed test techniques for AxICs. Experimental outcomes show that the synergy of the proposed techniques leads to achieve important results.

Resumé (FR)

Au cours des dernières décennies, la demande d'efficacité informatique n'a cessé de croître. L'avènement d'applications de nouvelle génération consommatrices d'énergie d'un côté, et d'appareils portables basse consommation de l'autre, exige un nouveau paradigme informatique capable de faire face aux exigences concurrentes des défis technologiques actuels [1]. Ces dernières années, plusieurs études sur les applications dites (en anglais) de *Recognition, Mining and Synthesis (RMS)* ont été menées [1]–[4]. Une particularité très intéressante a été identifiée : la *résilience intrinsèque* de ces applications. Une telle propriété permet aux applications RMS d'être très tolérantes aux erreurs. Ceci est dû à différents facteurs, tels que les données bruyantes traitées par ces applications, les algorithmes non déterministes utilisés et les réponses non uniques possibles [1]. Ces propriétés ont été exploitées par un nouveau paradigme informatique de plus en plus établi : le *calcul approximé* (AxC) [1], [2].

L'AxC profite intelligemment de la résilience intrinsèque des applications RMS pour réaliser des gains en termes de consommation électrique, de temps de fonctionnement et/ou de surface de puce. En effet, en introduisant des assouplissements sélectifs des spécifications non critiques, certaines parties du système informatique cible peuvent être simplifiées, pour finalement atteindre l'objectif de l'AxC. De plus, l'AxC est capable de cibler différentes couches des systèmes informatiques, du matériel au logiciel [2].

Dans cette thèse, nous nous concentrons sur les circuits intégrés approximés (AxICs), qui sont le résultat de l'application AxC au niveau matériel. En particulier, nous nous concentrons sur l'approximation fonctionnelle des circuits intégrés, utilisée au cours des dernières années afin de concevoir efficacement les AxICs [5]–[25]. En raison de la pertinence croissante des AxICs, il devient important de relever les nouveaux défis pour tester de tels circuits. À cet égard, certains travaux [26]–[29] ont attiré l'attention sur les défis que représente l'approximation fonctionnelle pour les procédures de test. En même temps, l'approximation fonctionnelle des circuits intégrés offre également des possibilités. Plus en détails - d'une part - le concept de *circuit acceptable* change : alors qu'un circuit est conventionnellement *bon* si ses réponses ne sont jamais différentes de celles attendues, dans le contexte AxIC certaines réponses inattendues peuvent encore être acceptables. Pour la même raison -

d'autre part - certains *fautes acceptables* peuvent ne pas être détectées, ce qui mène à un gain de rendement de production (c.-à-d., augmentation du pourcentage de circuits acceptables, parmi tous les circuits fabriqués). Pour mesurer l'erreur produite par un AxIC, plusieurs métriques d'erreur ont été proposées dans la littérature [30].

Dans cette thèse, nous présentons un ensemble de techniques de test pour les circuits approximés. En particulier, nous nous concentrons sur trois phases fondamentales du déroulement du test. Premièrement, la classification des fautes AxIC en *non-redundant* et *ax-redundant*. (c.-à-d. catastrophique et acceptable, respectivement) en fonction d'un seuil d'erreur (c.-à-d. la quantité maximale tolérable d'erreur). Cette classification permet d'obtenir deux listes de fautes (c.-à-d. non-redundant et ax-redundant). Ensuite, nous proposons une génération automatique de séquences de test (en anglais Automatic Test Pattern Generation ou ATPG) qui soit "consciente de l'approximation". Les tests obtenus préviennent les défaillances catastrophiques en détectant les fautes non-redundant. En même temps, ils minimisent la détection sur les ax-redundant. Enfin – puisque dans certains cas le gain de rendement obtenu ne correspond toujours pas à celui attendu, à cause de la structure propre des AxICs – nous proposons une technique pour classer correctement les AxICs dans les catégories "catastrophiquement défectueux" et "acceptablement défectueux", après l'application du test.

1. Contexte et informations générales

Dans ce chapitre, nous rassemblons quelques informations de base, qui seront utiles pour bien comprendre cette thèse et en tirer profit.

Tout d'abord, nous décrivons brièvement le test conventionnel de circuits intégrés. Nous rappelons les principes de base du test conventionnel pour les circuits numériques intégrés. Après une brève classification des différents objectifs du test, nous passons en revue la modélisation des fautes, les concepts de simulation des fautes, la procédure de génération de test et quelques concepts de base de la *conception en vue du test*, tels que la conception du *boundary scan* et le test automatique intégré (BIST).

Deuxièmement, nous passons en revue différents aspects du calcul approximé (AxC). En particulier, nous décrivons le problème abordé par l'AxC et les différents contextes dans lesquels il a été appliqué. En effet, plusieurs travaux ont abordé le problème de l'identification des parties appropriées d'un système informatique pour l'application de l'AxC. Ensuite, nous avons montré que l'AxC a une très large gamme d'applications. En effet, des études sur l'AxC au niveau logiciel, l'AxC au niveau architectural et l'AxC au niveau circuit ont été menées au cours des deux

dernières décennies. En particulier, nous décrivons les circuits intégrés approximés (AxICs).

Enfin, nous regroupons ces deux thèmes, dont l'union fait l'objet de cette thèse. Nous montrons comment les propriétés inhérentes aux AxICs nous ont conduit à reconsidérer les procédures de test et à proposer de nouvelles solutions. En d'autres termes, dans cette thèse, nous présentons des études sur *les techniques de test matériel pour les circuits intégrés approximés*.

2. Test des circuits approximés

L'un des problèmes majeurs qui affectent aujourd'hui la technologie CMOS à l'échelle nanométrique est ce qu'on appelle en anglais *process variability* ou *variabilité*. La variabilité est le résultat de la nature aléatoire des processus physiques qui ont lieu pendant la fabrication des circuits intégrés. Les circuits CMOS à l'échelle du nanomètre subissent l'effet de la variabilité et des mécanismes de dégradation, qui mènent à une baisse de rendement du procédé de fabrication [31].

L'AxC vise à transformer ce problème en opportunité. L'idée de base est d'accepter les erreurs en tant que propriété intrinsèque des circuits intégrés et de concevoir des circuits approximés optimisés fonctionnant indépendamment des erreurs. À cet égard, l'objectif ultime est d'augmenter le rendement du procédé de fabrication (c.-à-d. le pourcentage de circuits acceptables, parmi tous les circuits fabriqués), en acceptant les circuits dégradés qui fonctionnent de façon acceptable. Pour atteindre un tel objectif, les procédures de test doivent être repensées pour tenir compte de l'approximation introduite.

Par conséquent, nous devons examiner l'impact d'AxC sur le rôle des tests au niveau matériel. Dans le contexte des AxICs, le concept de *circuit défectueux* change et nécessite une enquête approfondie. Comme décrit dans la section précédente, l'approximation fonctionnelle vise à réaliser des gains d'efficacité (temps/surface/énergie) en assouplissant certaines exigences de précision. Afin d'obtenir des résultats satisfaisants, les concepteurs modifient attentivement la structure du circuit pour introduire une erreur *acceptable*. Pour définir la signification de *acceptable*, les concepteurs utilisent des métriques d'erreur. Ensuite, ils définissent des seuils d'erreur pour fixer l'erreur maximale autorisée (c.-à-d. acceptable).

Dans le contexte des tests, l'impact des fautes qui peuvent apparaître dans un circuit peut être mesuré et exprimé en erreur en utilisant de telles métriques. Si la mesure obtenue s'avère supérieure au seuil acceptable, le circuit doit être rejeté. Cependant, il peut arriver que l'erreur mesurée reste en dessous du seuil acceptable,

alors l'AxIC ne doit pas être rejeté. Par conséquent, dans ce contexte, les procédures de test ont un double rôle :

- rejeter les circuits dont l'erreur observée est supérieure au seuil, et
- éviter de détecter les fautes acceptables.

Il en résulte une augmentation du rendement et possiblement une réduction des coûts de test (c.-à-d. pour vérifier moins de défaillances, il faut moins de vecteurs de test).

De plus, en fonction de la métrique d'erreur, l'impact de la faute change. En effet, en stimulant un AxIC défectueux avec un vecteur d'entrée i, on peut mesurer l'erreur e_{s_i} - causée par la faute f_s - en utilisant une mesure M. En considérant le même vecteur d'entrée i mais une autre métrique $\widehat{M}$, l'erreur due à la même faute f_s est mesurée comme $\widehat{e_{s_i}}$. Généralement, e_{s_i} et $\widehat{e_{s_i}}$ ont des valeurs différentes. De plus, en stimulant le circuit défectueux avec deux vecteurs d'entrée différents i et j, les erreurs mesurées seront e_{s_i} et e_{s_j}, pour la métrique M, et $\widehat{e_{s_i}}$ et $\widehat{e_{s_j}}$, pour la métrique $\widehat{M}$. Là encore, les quatre erreurs ont généralement des valeurs différentes. Par conséquent, la faute f_s peut être considérée comme *acceptable* ou comme *catastrophique* selon la (ou les) métrique(s) considérée(s) pour l'application finale. En conséquence, les procédures de test doivent être attentivement repensées afin de relever les défis posés par l'approximation et de tirer profit des possibilités qui s'offrent. C'est pourquoi les tests conscients de l'approximation entrent en jeu. Nous identifions trois phases principales de tests conscients de l'approximation – ou Approximation-Aware (AxA) testing :

AxA fault classification Dans cette phase, les fautes sont classées en *catastrophique* (à tester) et *acceptable* (à ne pas tester), selon certains paramètres.

AxA test pattern generation Cette phase concerne la génération de vecteurs de test capables de couvrir tous les fautes catastrophiques et de laisser - autant que possible - les fautes acceptables non détectées.

AxA test set application Après l'application des séquences de test, une classification supplémentaire doit être effectuée. L'AxIC testé est classé comme *catastrophiquement défectueux*, ou *acceptablement défectueux*, ou *sans fautes*.

Par conséquent, seuls les AxICs classifiés en tant que *catastrophiquement défectueux* seront rejetés. Il en résulte une augmentation du rendement, puisque certains circuits défectueux - mais encore acceptables - ne seront pas rejetés.

Dans cette thèse, nous analysons en détail les phases du test conscient de l'approximation. De plus, nous présentons différentes techniques de mise en œuvre des tests conscients de l'approximation et d'optimisation de la qualité et de l'efficacité des tests. Nous effectuons des expériences approfondies pour évaluer leur efficacité.

3. Classification des fautes consciente de l'approximation

La complexité de la classification des fautes est influencée par le choix de la métrique d'erreur. En effet, l'erreur causée par une faute – ainsi que l'effort pour la mesurer – peut changer de manière significative en fonction de la métrique considérée. Comme nous l'avons souligné dans les sections précédentes, des métriques d'erreur sont nécessaires pour déterminer l'approximation des systèmes informatiques. En effet, il est obligatoire de mesurer l'erreur introduite par les approximations pour pouvoir produire des systèmes donnant de bons résultats. À différents niveaux d'abstraction, nous pouvons définir des métriques d'erreur appropriées. Pour les techniques d'approximation appliquées au niveau matériel, certaines mesures d'erreur bien acceptées existent. Par exemple, parmi ces métriques, on peut compter *l'erreur absolue maximale*, la *probabilité d'erreur*, *l'erreur absolue moyenne*. Pour classer une faute comme non-redundant selon la métrique d'erreur absolue maximale, il suffit de prouver une seule condition : l'existence d'une séquence de test conduisant le circuit défectueux à présenter une erreur supérieure au seuil d'erreur. Au contraire, pour classer une faute comme non-redundant selon la métrique de probabilité d'erreur et d'erreur absolue moyenne, il faut prouver que la probabilité et la moyenne de l'erreur ne dépassent pas les seuils d'erreur. Pour y parvenir, la contribution de l'ensemble exhaustif de vecteurs d'entrée doit être évaluée. En conséquence, il s'avère peu complexe d'évaluer l'impact d'une faute lorsqu'on considère des métriques pour lesquelles une seule condition doit être vérifiée, comme l'erreur absolue maximale. Inversement, classer les fautes selon des métriques qui impliquent le calcul d'une moyenne est un problème de complexité $O(2^n)$, où n est le nombre de bits en entrée.

Dans ce chapitre, nous présentons deux techniques pour traiter de la classification des fautes, en considérant les deux types de mesures. Les deux techniques sont basées sur une architecture spécifique capable de classer les fautes en non-redundant et ax-redundant en mesurant leur impact sur les sorties de l'AxIC. L'idée fondamentale est de "cacher" les fautes ax-redundant au moyen d'une *boîte de filtrage*. Ainsi, pour une faute donnée, une condition d'anomalie n'est générée que si la faute entraîne des erreurs catastrophiques. Une telle architecture de classification n'est jamais fabriquée, mais seulement utilisée au moment de la conception pour classer les

fautes.

4. Génération de vecteurs de test consciente de l'approximation

Dans ce chapitre, nous discutons du problème de génération des séquences de test consciente de l'approximation et présentons nos propositions pour y remédier. Comme précédemment indiqué, le rôle de la génération des séquences de test conscients de l'approximation est double : (i) les vecteurs de test doivent détecter tous les fautes non-redundant, afin d'éviter des erreurs catastrophiques aux sorties du circuit ; (ii) les séquences de test devraient détecter le moins de fautes ax-redundant possible, afin de ne pas considérer l'AxIC comme défectueux lorsqu'il est encore acceptable. En d'autres termes, un ensemble de tests qualitativement bon devrait atteindre 100% *non-redundant FC* (nR FC) et 0% *ax-redundant FC* (axR FC). Cependant, deux problèmes peuvent affecter la procédure de génération des séquences de test, en ce qui concerne les AxICs :

1. Afin d'atteindre 100% nR FC, il n'est pas toujours possible d'éviter de tester certains fautes ax-redundant (i.e., axR FC > 0%) ;
2. Les procédures conventionnelles de génération de tests pourraient ne pas être capable d'obtenir un ensemble de tests de bonne qualité.

Le premier problème est intrinsèque à la structure de l'AxIC testé, le second est relatif aux algorithmes conventionnels de génération de test. Par conséquent, un AxIC encore fonctionnel affecté par une faute ax-redundant serait rejetée en phase de test. Le phénomène en raison duquel un bon produit est considéré comme défectueux par le processus de test est communément appelé en anglais *over-testing*. Ce phénomène, si mal géré, finira par entraîner une importante diminution du rendement.

Ensuite dans ce chapitre, nous montrons comment la technique de classification des fautes présentée dans le chapitre précédent traite avec succès aussi la *génération des séquences de test consciente de l'approximation*, en détectant tous les fautes non-redundant. Néanmoins, la technique est limitée par certaines conditions particulières (c.-à-d. que la métrique utilisée dans la classification est l'erreur absolue maximale). De plus, nous montrons que en considérant différents ensembles de tests obtenant une non-Redundant Fault Coverage (nR FC) de 100%, différentes valeurs de ax-Redundant Fault Coverage (axRedundant Fault Coverage - axR FC) sont obtenues. Les techniques existantes ne permettent pas de résoudre le problème de trouver le meilleur ensemble de tests, c.-à-d. celui qui atteint une nR FC de 100% et qui minimise l'axR FC. C'est pourquoi nous proposons une technique plus

générale - basée sur une sélection minutieuse des séquences de test - conçue spécifiquement pour la génération des séquences de test consciente de l'approximation. Enfin, nous comparons les résultats des différentes techniques de génération des séquences de test, c.-à-d. la génération conventionnelle (ATPG ne prenant en compte que les fautes non-redundant), la génération consciente de l'approximation (c.-à-d. les séquences générées au même moment de la classification) et la génération consciente de l'approximation avec sélection de séquences.

Bien que les résultats obtenus soient assez bons, ils sont encore loin des résultats idéaux. Par conséquent, nous devons recourir à *AxA test set application* pour améliorer encore la qualité des tests.

5. Application de vecteurs de test consciente de l'approximation

Pour améliorer la qualité finale du processus de test, l'*application des tests consciente de l'approximation* joue un rôle important. Dans cette phase, nous avons besoin de techniques capables - en observant les réponses du circuit - de distinguer entre la détection d'une faute ax-redundant (le test passe) et celle d'une faute non-redundant (l'AxIC est rejeté). Dans ce chapitre, nous présentons *l'application de vecteurs de test conscients de l'approximation*. Tout d'abord, nous montrons et discutons les problèmes liés à l'application des tests dans le contexte des AxICs. Nous montrons qu'il n'est pas toujours possible d'éviter la détection de certains fautes ax-redundant, à cause de la structure des AxICs.

Pour éviter le phénomène d'over-testing qui en résulte, nous devons reconsidérer la phase d'application du test. En détail, après l'application des séquences de test à l'AxIC sous test, nous devons vérifier que la sortie de l'AxIC remplit certaines conditions et pas seulement si elle diffère de la sortie attendue.

Dans la littérature, aucune technique n'a été présentée jusqu'à présent pour traiter de cet aspect. Néanmoins, une technique présentée dans [32] pour les circuits conventionnels, le *threshold testing*, peut être adapté aux AxICs. C'est pourquoi nous essayons d'adapter cette technique aux AxICs. Malheureusement, des conditions restrictives spécifiques doivent être remplies pour que la technique soit appliquée avec succès.

Donc, nous proposons une nouvelle technique d'*application des tests consciente de l'approximation* pour faire face aux limitations rencontrées. La technique est basée sur le concept bien connu d'analyse de signature, appliqué aux architectures de test automatique intégrées (BIST) dans les années 70 [33]. Le résultat obtenu avec la technique proposée est vraiment bon. Nous décrivons également le phénomène de

l'aliasing dans le contexte des AxICs et évaluons quelques méthodes correctives pour y faire face.

6. Conclusions

L'introduction du paradigme du calcul approximé dans le panorama des technologies de l'information a apporté de multiples possibilités à des degrés divers. L'objectif fondamental du calcul approximé est d'améliorer l'efficacité du système (temps/surface/énergie) en assouplissant les exigences de précision des résultats. Le calcul approximé a été appliquée à différents niveaux des systèmes informatiques, du matériel au logiciel, en passant par les architectures. Parmi tous les travaux des deux dernières décennies, le calcul approximé a également été utilisé pour réaliser une nouvelle classe de circuits intégrés, c.-à-d. des circuits intégrés approximés ou AxIC. L'introduction d'une nouvelle classe de circuits a apporté de nouveaux défis, ainsi que de nouvelles opportunités, concernant le test et la vérification des puces. En particulier, les concepteurs de puces approximées modifient attentivement la structure du circuit pour introduire une erreur acceptable, afin d'obtenir des résultats satisfaisants. Pour définir correctement le concept d'erreur *acceptable*, les concepteurs utilisent *métriques d'erreur*. Ensuite, ils définissent des *seuils d'erreur* pour fixer l'erreur maximale autorisée (c.-à-d., acceptable). Par conséquent, le concept de circuit *défectueux* change. En effet, deux nouvelles catégories de défaillances sont introduites : les fautes ax-redundant (c.-à-d. causant des erreurs acceptables) et les fautes non-redundant (c.-à-d. causant des erreurs catastrophiques). Dans le contexte du test, la classe d'une faute détectable peut être déterminée en mesurant l'erreur causée à la sortie de l'AxIC. Si l'erreur mesurée est supérieure au seuil acceptable, le circuit doit être rejeté. Cependant, il peut arriver que l'erreur mesurée reste en dessous du seuil acceptable, alors l'AxIC ne doit pas être rejeté. Par conséquent, dans ce contexte, le rôle test change comme suit :

- les AxICs dont l'erreur observée est supérieure au seuil doivent être rejetés ;
- les AxICs affecté par des fautes acceptables ne doit pas être rejeté.

En conséquence, cela conduit à une augmentation du rendement de production.

En conséquence de ces considérations, nous introduisons les tests AxA, composés essentiellement de trois phases : (i) Classification des fautes consciente de l'approximation, (ii) Génération de séquences de test consciente de l'approximation, et (iii) Application des tests consciente de l'approximation. Toutes les phases de test conscientes de l'approximation apportent des contributions importantes à l'objectif final du test, dans le contexte des AxICs. Tout au long de la thèse, nous discutons en

détail de toutes les phases de test conscientes de l'approximation et nous présentons plusieurs techniques pour traiter de chaque aspect. Nous montrons que la synergie des techniques proposées permet d'obtenir des résultats optimaux.

Acknowledgements

I would like to thank some people for supporting me in the last three years. This thesis would not have been possible without them.

Thanks to my thesis supervisors, Alberto Bosio, Patrick Girard and Arnaud Virazel, for their precious guidance through this exciting and challenging path.

Thanks to the precursor of all this, Mario Barbareschi, for his valuable advice.

Thanks to Matteo Sonza Reorda and Olivier Sentieys for agreeing to review my work and to join the thesis committee. Thanks to Lirida Naviner, for agreeing to join the thesis committee.

Thanks to my family, my brother, my mother, my father, for being there for me, always, no matter what. Thanks to my new family, the love of my life Ada, for strongly supporting and lovingly encouraging me, every day more.

Thanks to my colleagues and friends, Bastien, Clement, Emanuele, Ilaria, Linh, Mathieu, Safa, and all the others at LIRMM: they made this "journey" easier and cheerful. Thanks to Caroline Lebrun, without whom all the administrative procedures would have been impossible. Thanks to the LIRMM, where I learned a lot. Thanks to Montpellier, where I spent three wonderful years,

Thanks to all the people who have shared with me a piece of their life.

Contents

List of Figures

List of Tables

List of Abbreviations

ATE	**A**utomatic **T**est **E**quipment
ATPG	**A**utomatic **T**est **P**attern **G**eneration
AxA	**A**pproximation **A**ware
AxC	**A**pproximate **C**omputing
AxIC	**A**pproximate **I**ntegrated **C**ircuit
axR	**a**pproximation-**R**edundant
AxRFM	**Ax**-**R**edundant **F**ault **M**asking
BFE	**B**it-**F**lip **E**rror
BIST	**B**uilt-**I**n **S**elf-**T**est
DfT	**D**esign **f**or **T**estability
DUT	**D**evice **U**nder **T**est
EM	**E**rror **M**agnitude
EP	**E**rror **P**robability
eYI	**e**xpected **Y**ield **I**crease
FA	**F**ull **A**dder
FC	**F**ault **C**overage
FFA	**F**ault **F**iltering **A**rchitecture
IC	**I**ntegrated **C**ircuit
ILP	**I**nteger **L**inear **P**rogramming
IoT	**I**nternet **o**f **T**hings
MAE	**M**ean **A**bsolute **E**rror
ME	**M**ean **E**rror
MSE	**M**ean **S**quared **E**rror
nR	**n**on-**R**edundant
RMS	**R**ecognition, **M**ining and **S**ynthesis
RYG	**R**elative **Y**ield **G**ain
SaF	**S**tuck-**a**t-**F**ault
SAT	Boolean **SAT**isfiability
SCT	**S**ingle-**C**ondition-**T**est
TF	**T**ransition **F**ault
UUT	**U**nit **U**nder **T**est
WCBFE	**W**orst **C**ase **B**it-**F**lip **E**rror
WCE	**W**orst **C**ase **E**rror
YIL	**Y**ield **I**ncrease **L**oss

Introduction

Despite significant energy efficiency improvements in the semiconductor industry, computer systems keep consuming more and more energy [1]. Many widely used applications – such as Recognition, Mining and Synthesis (RMS) applications – are increasingly deployed as mobile applications and on Internet of Things (IoT) structures. Therefore, it is necessary to improve the next-generation silicon devices and architectures on which these applications will run. The *inherent resiliency property* of RMS applications has been thoroughly investigated over the last few years [1]–[4]. This interesting property leads applications to be tolerant to errors – as long as their

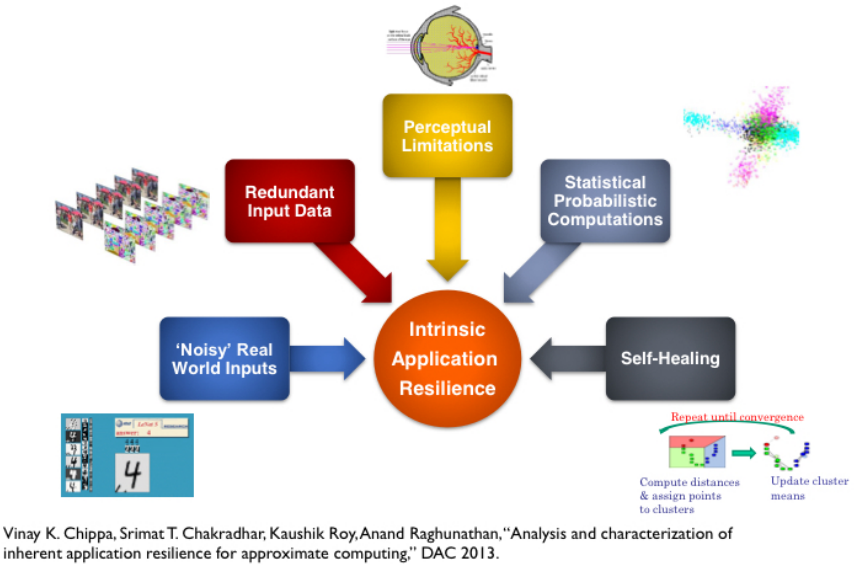


FIGURE 1: Inherent resiliency property [4]

results remain close enough to the expected ones. As shown in Figure 1, the main sources of error tolerance for these applications are:

- noisy real-world inputs,
- redundant data,
- perceptual limitations of individuals who will use the computation output,
- non-deterministic algorithms which lead to non-unique outcomes, and
- self-healing capable systems.

Approximate Computing (AxC) [1], [2] is an emerging computing paradigm which takes advantage of the inherent resiliency property. AxC has garnered increasing interest in the scientific community in the last years. It is based on the intuitive observation that selectively relaxing non-critical specifications may lead to improvements in power consumption, run time, and/or chip area. AxC has been applied to the whole digital system stack, from hardware to applications.

This work focuses on *Approximate Integrated Circuits (AxICs)*. AxICs stem from the application of AxC at hardware level. A widely used method to design those circuits is *functional approximation* of conventional integrated circuits (ICs) [5]–[25]. We focus more specifically on the testing aspects of functionally approximate ICs. Indeed, since approximation changes the functional behavior of ICs, we have to revisit techniques to test them. In fact, previous studies [26]–[29] have shown that circuit approximation brings along challenges for testing procedures, but also opportunities. In particular, approximation procedures intrinsically lead the circuit to produce errors, which have to be taken into account in test procedures. Error can be measured according to different error metrics [30]. On the one hand, the occurrence of a defect in the circuit can lead it to produce unexpected catastrophic errors. On the other hand, some defects can be tolerated, when they do not induce errors over a certain threshold. This phenomenon could lead to a yield increase, if properly investigated and managed. To deal with such aspects, conventional test flow should be revisited. Therefore, we introduce *Approximation-Aware testing (AxA testing)*. We identify three main AxA testing phases: (i) AxA fault classification, (ii) AxA test pattern generation and (iii) AxA test set application. Briefly, the first phase has to classify faults into *catastrophic* (to test) and *acceptable* (not to test); the test pattern generation has to produce test vectors able to cover all the catastrophic faults and, at the same time, to leave acceptable faults undetected; finally, the test set application needs to correctly classify AxICs under test into *catastrophically faulty*, *acceptably faulty*, *fault-free*. Only AxICs falling into the first group will be rejected.

In this thesis, we thoroughly discuss the three phases of AxA testing, and we present a set of AxA test techniques for approximate circuits.

- Firstly, we work on the classification of AxIC faults into *catastrophic* and *acceptable* according to an error threshold (i.e. the maximum tolerable amount of error). This classification provides two lists of faults (i.e. catastrophic and acceptable).

- Then, we propose an approximation-aware (ax-aware) Automatic Test Pattern Generation (ATPG). Obtained test patterns prevent catastrophic failures by detecting catastrophic defects. At the same time, they minimize the detection of

acceptable ones.

- Finally – since the AxIC structure often leads to a yield gain lower than expected – we propose a technique to correctly classify AxICs into "catastrophically faulty", "acceptably faulty", "and fault-free", after the test application.

To evaluate the proposed techniques, we perform extensive experiments on state-of-the-art AxICs.

Chapter 1

Context and background concepts

Contents

In this chapter, we put together some background information, which will be useful for fully understand and profit from this thesis. Firstly, we briefly describe conventional Integrated Circuit (IC) testing. IC testing represents the *technical focus* of this thesis. Secondly, we review different aspects of Approximate Computing (AxC). In particular, we describe approximate integrated circuits (AxICs), that constitute the *context* of this work. Finally, we put together the two aforementioned topics, the union of which forms the subject of this thesis. We show how inherent properties of AxICs led us to reconsider the test procedures and to propose new solutions. In other words, in this thesis we present studies on *hardware test techniques for approximate integrated circuits*.

1.1 Conventional IC testing

In this section we recall some basic principles of conventional IC testing, which will be useful in different parts of this thesis. The concepts reported are not intended to be exhaustive. Extensive disquisitions on the concepts reported below can be found in [34].

As sketched in Figure 1.1, in digital testing, binary patterns (or *test patterns*) are applied to circuit's inputs. Responses are compared with the expected ones (*golden* responses). If they match, the circuit is considered good, otherwise it is marked as

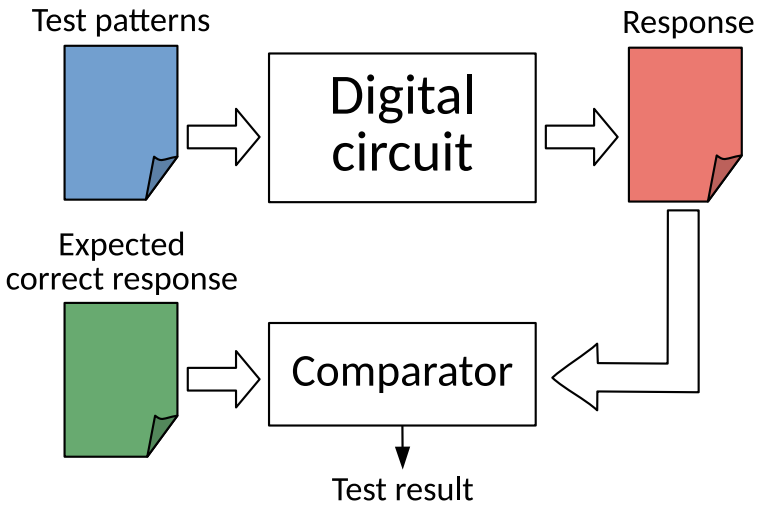


FIGURE 1.1: Digital testing [34]

faulty. VLSI testing can be classified depending on the goal it is intended to serve:

Verification testing Before sending a new design to production, its correctness and adherence to specifications must be verified. In this phase, functional and parametric tests are applied to ICs. Physical quantities (AC, DC) are measured

under different operative conditions. As a result, the operating limits of the chips are determined. Thus, correction on the design are performed and the final specifications are set. Verification testing can be employed also during the product lifetime to possibly improve the design and the process yield. Usually, this is done on chips rejected by production test or in the field.

Production testing After chip manufacturing, production testing must determine whether the actual fabricated devices meet specifications or not under normal operating conditions. Test vectors must have a high coverage of modeled faults and – since every device must be tested – test time (thus the cost) must be kept short. Test is performed either at the speed required by the application of the device or at the speed guaranteed by the supplier.

Burn-in testing Burn-in puts chips in high temperature environments, while applying production tests and over-voltage power supply. By doing this over a certain period, the reliability of the tested devices is ensured. Indeed, some chips that passed production tests can still fail after a very short time. Failures of this kind are usually called *infant mortality* failures. Defects causing this kind of failures must be detected early. In order to accelerate the occurrence of such failures, high temperatures turn out to be effective. Depending on the desired trade-off between reliability and cost, the burn-in time changes. Long burn-in time is expensive but usually leads to have a more reliable device.

Incoming Inspection Before system integration, customer may want to inspect devices. Indeed, discovering defective devices before the integration is by far less expensive than perform a system diagnosis. Depending on the customer needs, test procedures may be similar to production testing, more thorough or dependent on the specific system application.

More in general, two types of tests are performed on VLSI chips:

Parametric Tests Necessary to decide whether the chip pins meet various non-functional specifications, such as rise and fall times, setup and hold times, low and high voltage thresholds, and low and high current.

Functional tests Necessary to determine whether the chip internal digital logic behaves as designed. Tests consist in driving input vectors to the Device Under Test (DUT) and verifying that the corresponding responses match the expected ones. The goal is testing the proper operation of internal chip nodes.

In this latter category, the *manufacturing test* (or *hardware test*) aims at discovering any manufacturing defect. In 1959, Eldred proposed tests capable of observing the

internal state of signals in large digital system, by propagating their effect at primary outputs [35]. This type of tests are commonly referred to as *structural test*, because they depend on the internal structure of the circuit. As a consequence, algorithms based on IC's internal structure can be developed. Fault models – briefly discussed in next section – are the core of structural test algorithms. In this thesis, we focus on structural tests.

Finally, functional and physical characteristics, type of device, technology, desired reliability, and environment specification determine the type of test equipment to use. Testers – popularly known as Automatic Test Equipment (ATE) – drive the inputs and monitor the outputs of a DUT. Test data obtained from the ATE helps to accept or reject the DUT. Moreover, information about the fabrication process and about design problems can be extracted.

1.1.1 Defect modeling

To correctly describe an *incorrect* electronic system, different terms have to be defined. Below, we report common definitions of Defect, Error and Fault.

Defect Unintended difference between the implemented hardware and its design. Defects can occur during manufacture, as well as during the device lifetime.

Error A wrong output signal produced by a defective system. An error is caused by some defect in the hardware.

Fault An abstraction model of a defect.

Even if a defect is present within an IC, its manifestation might never happen. In general, given the list of all possible defects (modeled as faults) that can occur within an IC, a subset of them is referred to as *detectable faults*. A fault is defined as detectable if it exists an input pattern sensitizing and propagating the fault effect to outputs. From now on in the text, we will refer to defect and to its model – the fault – interchangeably.
Fault modeling is performed at different levels of abstraction:

Behavioral level Sometimes referred to as *high leve*l, behavioral level fault models may not have correspondence in manufacturing defects. Mostly, they are used in design verification rather than testing.

Logic level or Register-transfer level (RTL) At this level, we find fault models usually built by considering the *netlist*, i.e., the circuit component list and their inter-connections. Stuck-at fault model is the most popular and used one in

digital testing. Among others, we find delay fault model and bridging fault model.

Component level At this level we find lower abstraction level, such as the transistor level. Stuck-open fault model, which is a technology-dependent model, is mainly used at this level. Mostly, analog circuit testing resorts to component level fault models.

In this thesis, we focus on logic level fault models, since we address digital integrated circuit testing.

In the following, we report some definitions concerning faults, in order to provide possible inexperienced readers with some useful terms for the rest of the thesis.

Stuck-at fault model In this abstraction, a circuit net is considered to be permanently set at a constant value. By assigning a fixed (0 or 1) value to an input or an output of a logic gate or to a flip-flop in the circuit, the SaF model represents this condition. The SaF model is the most popular fault model used in practice for digital IC testing. The most popular forms are the *single stuck-at faults*. In this abstraction, a single fault line is assumed to be present in the IC, either stuck-at-1 (sa1) or stuck-at-0 (sa0).

Delay fault model Defects modeled by delay fault model prevent the correct data from reaching outputs at the right time. Among different types of delay faults models we find transition faults, gate-delay faults, path-delay faults.

Redundant fault In a combinational circuit, a redundant fault does not modify the circuit's output for any input combination. Thus, a test detecting a redundant fault cannot exist. Redundant faults are a subset of the more general *untestable faults*. In sequential circuits, faults for which no test pattern can be found fall into the untestable fault category.

Multiple fault The condition that simultaneous single faults affect the same circuit is referred to as multiple fault. Multiple Stuck-at faults model is usually not considered, due to the tremendous complexity. Moreover, a very high percentage of these faults are covered by single stuck-at faults tests.

Equivalent faults If two faults f_1 and f_2 lead a circuit to have the exact same function, they are defined as *equivalent*. A test detecting f_1 detects also f_2 and vice-versa. This leads to *fault collapsing*: partitioning all the faults of a circuit into disjoint equivalence sets and selecting one fault from each equivalence set to test. For a circuit having n lines (thus $2n$ single stuck-at faults) the equivalence

between $2(n^2 - n)$ pairs of faults should be determined, which is complex. Therefore, for stuck-at fault model, the fault equivalence is usually determined between faults affecting each Boolean gate.

1.1.2 Fault simulation

In the design of VLSI circuits, the concept of *simulation* is of great importance. Firstly, it serves the purpose of verifying the circuit correctness. Secondly, it verifies whether and how efficiently a test set fulfill its purpose.

The circuit correctness verification is a fundamental step of the design activity. After the synthesis process, the produced netlist is verified by a *true-value simulator*, i.e., it produces the responses of the defect-free circuit. Since the goal is to verify the circuit functionality according to the specification, the input stimuli applied by the simulator to the circuit are based on the specification. Any errors lead to change the design to make responses to all stimuli match the specification.

Simulation is also used for the development of manufacturing tests. A so-called *fault simulator* acts like a true-value simulator with the capability to simulate a faulty-circuit. Once the verified circuit netlist is available, the fault simulator can verify the coverage of a given set of input stimuli (usually, the verification ones) for a given fault list. Faults covered by the given set are marked as *detected* and the *Fault Coverage* is measured.

Fault Coverage (FC) The ratio of the number of faults detected by a set of test patterns to the total number of faults in the fault list.

An adequate FC (98% - 100%) is usually required in order to ship high quality devices to the customers. A good-quality test is a test that can minimize the number of faulty circuits sold, while keeping the test cost acceptable.

Test quality The test quality is expressed as *defect level* (or *field reject rate*): the fraction of chips that, despite having passed the test, are actually faulty. Defect level is expressed as *parts per million (ppm)*. High quality tests are considered as providing chips with a defect level of 100 ppm or lower.

Process variations, such as impurities in materials, dust particles, etc., can produce defects during the manufacture. In turns, defects can cause circuits to fail. Process variation effects reflect on the *process yield*:

Process yield The fraction (or percentage) of acceptable parts (thus, sold) among all fabricated parts is commonly referred to as process yield, or simply as *yield*.

In a typical case, a newly designed chip has a low yield, at its early manufacturing period. Thanks to process diagnosis and correction, higher process maturity is achieved and, thus, significantly higher yield.

> While the role of conventional testing is rejecting defective circuits, yet it cannot improve the process yield. All along this thesis, we will discuss the role of IC testing when approximate computing comes into play. It turns out that – in this particular context – test procedures have a different role which includes the opportunity to increase the process yield.

Verification stimuli may not produce an adequate FC. As shown in the next subsection, test generator programs can produce new test vectors to increase the FC.

1.1.3 Test generation

In late-fifties, Eldred highlighted the necessity for the structural testing of logic circuits to prevail over the classic functional test [35]. He argued that formulating test conditions at the level of the components is *"the only way in which all conditions of operation of each logical function can be uniquely [...] defined and all logical components within each logical function can be made to perform the task to which they are assigned [...] thereby producing a minimum program which tests and detects failure"*. The goal of structural test is to verify the presence of the minimal set of faults in the circuit. Therefore, the application of fault equivalence is important to reduce the final set of faults to test.

Automatic Test Pattern Generation (ATPG) serves the purpose of producing patterns to test the internal structure of a digital circuit, starting from its netlist description. The commonly used method in ATPG, namely *path sensitization* is based on three steps:

1. fault *injection* in the circuit netlist;
2. fault *activation*;
3. fault effect *propagation* toward circuit outputs.

To briefly describe path sensitization, let us resort to the stuck-at fault model (see Subsection 1.1.1). Let us assume that we want to test if a line l is stuck to a constant value (say 1). The test vector v detecting that fault is composed of input values such that:

- the line l is set to the opposite value of the fault (say 0). This is commonly referred to as fault *sensitization* or *activation* or *excitation*;

- the effect of the previous action is propagated to circuit outputs. This is commonly referred to as fault *propagation* or *path sensitization*.

By simulating the pattern with the fault-free circuit, we obtain the fault-free output value (expected output). Now, let us assume that an actual stuck-at fault (say Sa1) occurs at line l. In presence of the fault, circuit outputs will be different from expected. Therefore, by applying the test vector v to the circuit and knowing the expected output, we are able to detect the fault by observing a difference between actual and expected outputs.

> In the context of conventional IC test, even a little difference between the nominal behavior and the manufactured IC's leads to reject the circuit. Later in this thesis, we will discuss this aspect when approximate computing is considered. In this particular context, the value of the difference between the nominal behavior and the manufactured IC's is important. In fact – under specific conditions – the manufactured circuit may be still accepted even if some defects occur.

Unfortunately, the described ATPG method works correctly only for combinational circuits, i.e. without cycles. In fact, any circuit with cycles will lead the aforementioned method to fall into an infinite loop. ATPG methods for sequential circuits exist but are usually very resource-consuming and sometimes inefficient. The main difficulty for *sequential ATPG* is to control and to observe the internal state of the circuit.

Therefore, *design-for-testability (DfT)* comes into play. As stated by Agrawal and Seth [36], *"testability is the property of a circuit that makes it easy (and sometimes possible!) to test"*. DfT refers to the set of design techniques for ICs aiming at improving the testability of the target design. The most popular DfT technique is the scan

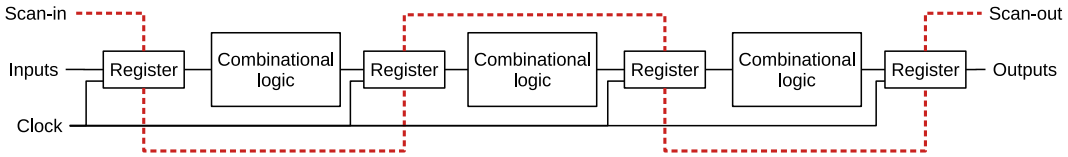


FIGURE 1.2: Scan design

design. Figure 1.2 depicts the scan design basic concept. This technique aims at increasing controllability and observability of flip-flops in a sequential design. This is done by connecting all the flip-flops together, to form a long shift register when a *test-mode* is activated. By shifting logic values in and out, it is possible to control and observe the internal state of the circuit. As a result, this approach converts the ATPG problem for sequential circuits into a well-known and more tractable ATPG

for combinational circuits. Of course, this comes at the cost of extra hardware, delay, and test time. Such cost may be justified by time-to-market-driven decisions. In fact, existing ATPG for sequential circuits often introduce design delay, while ATPG for combinational logic has a predictable test development time.

1.1.4 Built-In Self-Test

As the VLSI matured, the complexity of microelectronic systems grew considerably. As a consequence, performing IC testing became more and more difficult. As mentioned in the previous subsection, design-for-testability techniques contributed to simplify the testing of circuits at the cost of some additional resources. In 1977, the concept of *signature* was introduced by Frohwerk as a new method to determine IC correctness [33].

Test signature The *word* resulting from the compaction of IC test responses is defined test *signature*

This reduced the IC test to a comparison of two signatures. In details, after the design of the patterns to test the IC, the responses of the fault-free IC are compacted in a test signature (golden signature). When the manufactured IC is tested by applying the same patterns, the signature obtained by compacting the responses is compared with the golden one. If they match, the circuit is considered fault-free. Otherwise, it is marked as faulty. Sometimes, it can happen that the signature of a faulty circuit matches the golden one. This is referred to as *aliasing* phenomenon.

Aliasing During the compaction of the circuit's response, a signature of a bad device may match the golden signature. This is due to the information loss during the compaction. When aliasing occurs, a failing circuit might pass the test and be shipped to the customer.

As a matter of fact, the real breakthrough was the application of signature analysis to the so-called Built-In Self-Test (BIST).

Built-In Self-Test (BIST) A circuit which is capable to autonomously determine whether it is fault-free or not has BIST capabilities.

In Figure 1.3 we report the generic BIST architecture as presented in [34]. In detail – when the test mode is activated – test patterns are applied to the circuit and a signature is generated. Then, the latter is compared with the golden signature, which was generated by the fault-free circuit and stored within the BIST architecture. If the two signatures are identical, the circuit is considered fault-free. Otherwise, a malfunction

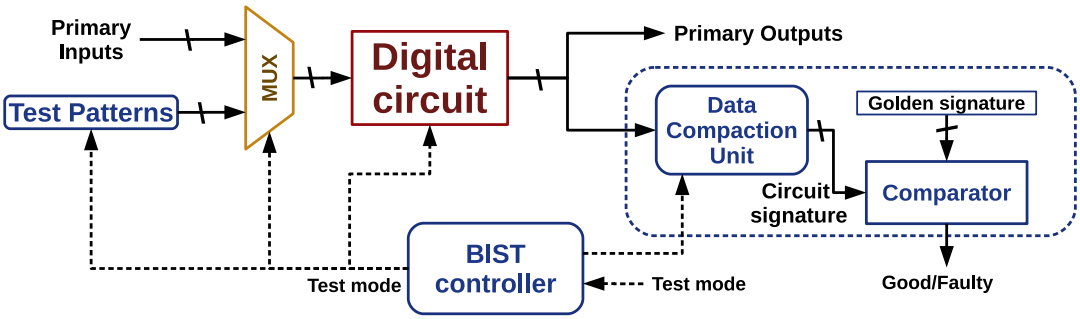


FIGURE 1.3: Generic BIST process [34]

is detected. Different pattern generation and signature compaction methods exist in the literature. An extensive review of those methods can be found in [34].

With the BIST introduction, test became part of the system functionalities rather than a procedure performed only occasionally. Indeed, during its lifetime, a modern digital system is tested very often. As a consequence, test must be performed in the most rapid and efficient way possible. BIST serves this purpose very efficiently and, when properly designed, the extra hardware cost is more than balanced by the benefits in terms of reliability and reduced maintenance cost.

1.2 Approximate computing (AxC)

Let us now introduce the context of this thesis, the *Approximate Computing (AxC)*. Some definition of Approximate Computing have been provided in last years:

> *"Approximate computing [...] is based on the intuitive observation that while performing exact computation or maintaining peak-level service demand require high amount of resources, allowing selective approximation or occasional violation of the specification can provide disproportionate gains in efficiency."* — Mittal, 2016 [2]

> *"By relaxing the numerical equivalence between the specification and implementation of error-tolerant applications, approximate computing deliberately introduces 'acceptable errors' into the computing process and promises significant energy-efficiency gains."* — Xu, Mytkowicz, Kim, 2016 [1]

> *"A new design paradigm, Approximate Computing (AxC), has been established to investigate how computing systems can be more energy efficient, faster, and less complex. Intuitively, instead of performing exact computation and, consequently, requiring a high amount of resources, AxC aims to selectively relax the specifications, trading accuracy off for efficiency."* — Bosio, Menard, Sentieys, 2019 [37]

The three definitions above respectively highlight *the problem*, *the context*, and *the purpose* of AxC.

The problem As energy-demanding applications more and more establish in the information technology scene, it is expected that in the next decades the amount of information will grow so much that it will exceed available resources [2].

The context The so-called Recognition, Mining and Synthesis (RMS) applications have a very interesting peculiarity, i.e., *error resiliency* [4]. Indeed, RMS applications turn out to be intrinsically tolerant to errors thanks to different factors, such as noisy data processed, non-deterministic algorithms used, and possible non-unique outcomes.

The purpose In the last decades, several works have been inspired by the opportunity that aforementioned resilient applications brought along for energy-efficiency. The purpose of AxC is to represent a new paradigm that drives the investigation of new energy-efficient computing solutions.

AxC has drawn the attention of both academia and industry, and a lot of works have focused on different aspects of AxC [1], [2].

1.2.1 How to determine where to apply AxC?

Approximation can be applied at different levels. Specifically – according to the classification in [1] – there are three main categories of so-called *approximate kernels*: Software-level, Architectural-level, and Circuit-level. All the three categories have in common the need of *identify* and *characterize* the resilient parts of the target system. Generally, when the *approximable* regions are somehow known, the target applications/systems are "annotated" to express the opportunity to approximate [38], [39]. Alternatively, the resiliency of the different parts of the system can be explored by means of sensitivity analysis [4], [40]. Other techniques resort to dynamic output monitoring to adapt the accuracy of the computation at run-time [41]–[44]. To suitably identify the approximation opportunities, the above methods resort to *metrics* to measure the accuracy loss as the approximation is introduced. As reported in [2], several error metrics have been used in the literature, such as:

- *Peak Signal-to-Noise Ratio (PSNR)*, *Structural SIMilarity (SSIM)*, *pixel difference* for image/video processing algorithms (e.g., JPEG, MPEG);

- *classification/clustering accuracy* for the classification/clustering algorithms (e.g., k-means);
- *ranking accuracy* for document search algorithms (e.g., Supervised Semantic Indexing).

Furthermore, more generic metrics can be used to evaluate approximate systems' accuracy. For instance, the Error Probability (EP) measures the percentage of erroneous outputs produced by an approximate system/application compared to its precise version.

1.2.2 Software-level AxC

At software level, AxC has been employed to provide programmers with the possibility to realize complex yet energy-efficient programs. This task is possible thanks to the abstraction of the *approximation concept* by means of *approximation-aware (ax-aware) programming languages*, *ax-aware correctness analysis engines*, and *ax-aware compilers*.

Ax-aware programming languages The main goal of programming languages is to allow programmers to express *what to do* instead of *how to do* it, by using resource abstraction. Likewise, approximation-aware programming languages help programmers expressing *randomness* [1]. Examples of such languages are in [38], [39], [45], which provide the programmers with approximation-related syntax.

Ax-aware correctness analysis The goal of analyzing an "ax-aware source code" is to build a model of it. The goals are (i) to state whether the code is correct or not and (ii) verifying if the code respects some properties about the produced output error. To do so, *probabilistic modeling* is suitable [1]. Some probabilistic model checking works have been proposed, such as [46], [47]. Other propositions focus on adapting conventional static and dynamic program analysis to compute the probability of critical output deviations in the final program [48]–[51].

Ax-aware compilers In general, the goal of compilers is to translate the source code into a sequence of tasks that the underlying hardware system has to perform. In addition, ax-aware compilers can exploit the information gathered from the ax-aware source code and the ax-aware analysis to transform the program semantics. The final goal is to sacrifice some accuracy (within some boundaries) to improve energy consumption or performance. Examples are the use of loop

perforation (execute fewer iterations than usual) [52], and of operand bit-width reduction [53].

1.2.3 Architectural-level AxC

At architectural level, the fundamental components are *computing units*, *memories* and *storage devices*. When building a computer system, the ideal goal is to obtain high-performance processing units at a low energy cost, and to obtain a good trade off between performance and density, for memories and storage units [1]. AxC has put into play a new parameter to push farther next generation hardware components, i.e., the quality. Indeed, by sacrificing some quality, one can further improve performance, density and energy efficiency.

Approximate computing units Classic computing units are usually grouped into two broad categories: *general purpose* computing units and *special purpose* computing units. General purpose units combine high-level instructions to realize generic tasks. On the contrary, special purpose units are built to fast execute a set of predefined actions. Along this same lines, AxC has been applied (i) to enhance general purpose computing units that execute selected instruction (or code segments) in an energy-efficient fashion [54], [55] and (ii) to transform whole approximable algorithms into neural accelerators [56].

Approximate memories Some problems limit the energy efficiency of conventional SRAMs and DRAMs, in the precise domain. SRAMs start producing errors when the operating voltage decreases under a threshold, and they are also vulnerable to particle strikes if not properly protected by using big memory cells. AxC profits from data resiliency by systematically storing the least significant bits in energy-efficient small SRAM cells [57]–[59]. Conventional DRAMs need to be refreshed periodically, which entails a big energy consumption. To take advantage of error-resilient data, AxC techniques apply longer refresh periods to memory rows storing those data to improve energy-efficiency [60]–[62]. In [63], also multi-level approximate memory architecture based on data significance analysis was proposed. Furthermore, the applicability for approximate computing of emerging non-volatile memories, such as the Spin Transfer Torque Magnetic RAM (STTMRAM), has been evaluated in [64].

Approximate storage In solid-state storage units, a lot of effort in terms of energy/latency is required to precisely store and retrieve multiple data. AxC comes into play

when data precision can be relaxed, thus storage and retrieval can be performed with less effort. Moreover, storing resilient data into overused storage blocks increases the lifetime of the storage unit [65].

1.2.4 Circuit-level AxC

Finally, we come to circuit-level, where AxC has basically been applied in two ways: (i) *over-scaling* and (ii) *functional approximation*. Over-scaling consists in lowering the circuit supply voltage to reduce its energy consumption. If the circuit is systematically designed to profit from over-scaling [66], [67], the timing errors are negligible compared to the energy gain. Nevertheless, the energy gain for over-scaling techniques turns out to be still small [1]. Therefore, a considerable amount of works has been presented on circuit *functional approximation*: the circuit functionality is systematically changed – thus, some controlled errors are introduced – to achieve energy-efficient circuits. So far, three main approaches have been used to design approximate integrated circuits (AxICs):

Ad-hoc approximate circuits RMS applications mostly rely on simple arithmetic operations, such as addition and multiplication. A lot of work have been done to realize energy-efficient and performance-enhanced approximate adders [10], [11], [22], [23], [25], [68]–[71]. A comprehensive review and comparison can be found in [6]. Moreover, non-volatile logic-in-memory approximate adders were proposed in [72]. Specifically, Spin Torque Transfer Magnetic Tunnel Junction (STT-MTJ) was used to implement a magnetic full adder. Furthermore, also a lot of effort has been put in the design of approximate multipliers [5], [7], [8], [13], [25].

Approximate circuit synthesis Unlike above discussed arithmetic circuits, for general logic circuits we cannot use ad-hoc techniques, due to the exponential complexity that VLSI circuits bring along. In the eighties, logic minimization techniques were proposed to cope with this complexity and drive the cutting-edge automated logic synthesis techniques [73]. First attempts of AxC-oriented methodologies have been proposed in [14] and [74] to implement the automated synthesis of AxICs. The main challenge was the absence of a well-accepted error model for general circuits. In fact, only simple error models were used. Therefore, some frameworks to flexibly represent the error were proposed in [15], [20], [75]. Finally, also RTL-level [76] and HLS-level [77], [78] languages were proposed to guide the approximation-oriented logic synthesis.

Hardware neural accelerators The intrinsic approximate nature of neural accelerators are particularly suitable to implement approximate functions. Different works have been proposed on hardware neural network implementations both with digital logic circuits [56] (precise and reliable) and analog circuits [79] (compact and energy-efficient). Furthermore, also ReRAM crossbar arrays were used to implement really energy-efficient solutions [80]. However, challenges related to the interfacing energy overhead and to the premature technology still have to be faced.

1.2.5 Error Metrics for Approximate Computing

As highlighted in previous subsections, error metrics are needed to drive the approximation of computing systems. Indeed, it is mandatory to measure the error introduced by approximations to correctly produce systems delivering good-enough results. At different abstraction levels, we can define suitable error metrics. For low-level-abstraction approximation techniques, such as circuit-level ones, some well-accepted error metrics exist. Among commonly used metrics for AxICs we can mention *Error Magnitude* (EM), *Bit-Flip Error* (BFE), *Worst Case Error* (WCE), *Mean Absolute Error* (MAE), *Mean Squared Error* (MSE), *Error Probability* (EP), and *Worst Case Bit-Flip Error* (WCBFE) [30], defined as follows:

$$\mathrm{EM}_i = \left| O_i^{approx} - O_i^{precise} \right|,\ i \in \mathcal{I} \tag{1.1}$$

$$\mathrm{BFE}_i = \sum_{j=0}^{n-1} (O_{i,j}^{approx}) \oplus (O_{i,j}^{precise}),\ i \in \mathcal{I} \tag{1.2}$$

$$\mathrm{WCE} = \max_{\forall i \in \mathcal{I}} \left| O_i^{approx} - O_i^{precise} \right| \tag{1.3}$$

$$\mathrm{MAE} = \frac{\sum_{\forall i \in \mathcal{I}} \left| O_i^{\mathrm{approx}} - O_i^{\mathrm{precise}} \right|}{2^n} \tag{1.4}$$

$$\mathrm{MSE} = \frac{\sum_{\forall i \in \mathcal{I}} \left| O_i^{\mathrm{approx}} - O_i^{\mathrm{precise}} \right|^2}{2^n} \tag{1.5}$$

$$\mathrm{EP} = \sum_{\forall i \in \mathcal{I}:\ O_i^{\mathrm{approx}} \neq O_i^{\mathrm{precise}}} \frac{1}{2^n}. \tag{1.6}$$

$$\mathrm{WCBFE} = \max_{\forall i \in \mathcal{I}} \sum_{j=0}^{n-1} (O_{i,j}^{approx}) \oplus (O_{i,j}^{precise}) \tag{1.7}$$

where:

$i \in \mathcal{I}$	input value within the set of all possible inputs $\mathcal{I}$
$O_i^{precise}$	precise output integer representation, for input i
O_i^{approx}	approximate output integer representation, for input i
n	number of input signals to the circuit
$O_{i,j}$	j-th bit of the O_i output (precise or approx)

However, as it can be deduced from Subsection 1.2.1, for higher abstraction levels, error metrics are application-dependent. Thus, approximation techniques should take into account the final application that the approximate system will execute. Unfortunately, this not happens at all levels, yet. Authors in [81] show that, while circuit-level approximations provide a promising energy gain, this is not reflected at application level. Indeed, they compared carefully sized (via truncation and rounding) fixed-point arithmetic operators and state-of-the-art approximate arithmetic circuits. They used both the approaches to implement different real-life applications and discovered that low-level approximated circuits (or low-level operators) lead to a lower gain compared to carefully sized arithmetic (high-level) operators. This happens because low-level operator approximation is performed by ignoring the context where such operator will be used (i.e., other operations in the application).

The relationships between metrics at different levels has not been thoroughly studied, yet. Preliminary studies have been proposed to evaluate the impact of local approximations on real-life application output by using error propagation models [82]–[84].

1.3 Testing circuits in approximate context

In this section, we introduce the topic of this thesis: *test techniques for approximate circuits*. In order to correctly understand the motivations of this work, we firstly need to present one of the major problems nowadays affecting nano-scale CMOS technology, i.e. *process variation* or *variability*:

> *"Random errors, usually denoted as variability, are the result of the stochastic nature of many physical processes that take place during the fabrication of integrated circuits.* [...] *Continuous scaling of CMOS technologies into the nanometer range has increased the effect of variability and degradation mechanisms on the yield and reliability of CMOS circuits and systems." — Gielen et al. 2008 [31].*

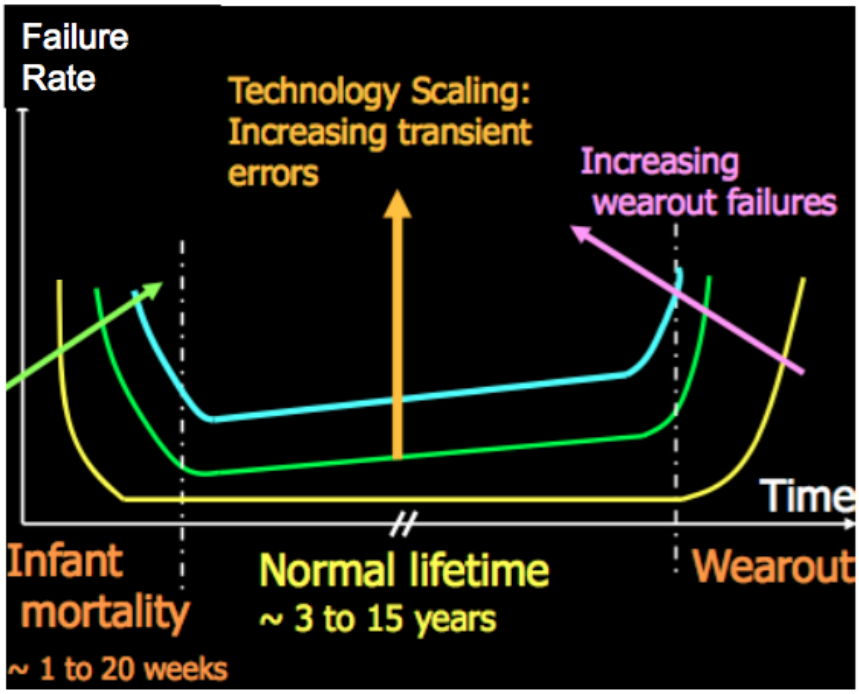


FIGURE 1.4: Reliability decrease with technology scaling [85]

As depicted in Figure 1.4 from [85], the continuous technology scaling of CMOS technology is affecting more and more the reliability of integrated circuits. In fact, as the technology shrinking process further pushes the miniaturization of CMOS transistors, the normal lifetime of ICs is the more and more reduced.

AxC, as described in the last section, aims at transforming this problem into an opportunity. The basic idea is to "embrace" errors as an intrinsic property of integrated circuits and systematically design optimized approximate circuits functioning regardless of errors. In this regard, the ultimate goal is to increase the production yield (i.e., the percentage of acceptable circuits, among all fabricated circuits), by accepting degraded circuits that still work acceptably. To achieve such a goal, test procedures have to be re-designed to be aware of the introduced approximation.

Therefore, we need to consider how AxC impacts on the role of hardware testing. In the context of AxICs, the concept of *faulty* circuit changes and needs a thorough investigation. As described in previous section, functional approximation aims to achieve gains in efficiency (time/area/energy) by relaxing some accuracy requirements. In order to still obtain satisfying results, designers carefully modify the circuit structure to introduce *acceptable error*. In order to define the concept of *acceptable* error, designers resort to **error metrics**. Then, they define **error thresholds** to fix the maximum allowed (i.e., acceptable) error.

In the testing context, the impact of detectable faults can be measured and expressed as error by using such metrics. If the obtained measure turns out to be

higher than the acceptable threshold, then the circuit has to be rejected. However, it may happen that the measured error stays below the acceptable threshold, then the AxIC must not be rejected. Therefore, in this context, test procedures have a twofold role:

- reject circuits whose observed error is greater than the threshold, and
- avoid detecting acceptable faults.

This ultimately leads to yield increase and possibly to the test cost reduction (i.e., fewer test vectors are needed to test fewer faults).

Besides, depending on the error metric, the fault impact changes. Indeed, by stimulating a faulty AxIC with an input vector i, we can measure the error e_{s_i} – caused by the fault f_s – by using a metric M. By considering the same input vector i but another metric $\widehat{M}$, the error due to the same fault f_s is measured as $\widehat{e_{s_i}}$. Usually, e_{s_i} and $\widehat{e_{s_i}}$ have different values. Therefore, the fault f_s can be considered as *acceptable* or as *catastrophic* depending on the metric(s) considered for the final application. As a result of this consideration, test procedures have to be carefully redesigned in order to address the challenges introduced by the approximation and to profitably take advantage of the opportunities. And that is why Approximation-Aware (AxA) testing comes into play. We identify three main AxA testing phases:

AxA fault classification In this phase faults are classified into *catastrophic* (to test) and *acceptable* (not to test), according to some metrics.

AxA test pattern generation This phase addresses the generation of test vectors able to cover all the catastrophic faults and to leave – as much as possible – acceptable faults undetected.

AxA test set application After the application of the test patterns, a further classification needs to be performed. The AxIC under test is classified either as *catastrophically faulty*, or *acceptably faulty*, or *fault-free*.

As a result, only AxICs falling into the *catastrophically faulty* group will be rejected. This ultimately leads to a yield increase, since some faulty circuits – yet still acceptable – will not be rejected.

1.4 Chapter summary

In this chapter we reviewed the background notions on which this thesis relies. In Section 1.1, we recalled basic principles of conventional testing for integrated digital

circuits. After a brief classification of test's different goals, we reviewed defect modeling, fault simulation concepts, test generation procedure, and some basic design-for-test approaches, such as *scan design* and *built-in self-test*.

Afterwards, in Section 1.2, we reviewed the basic principles of Approximate Computing (AxC) paradigm. In particular, we described the problem addressed by AxC and the different contexts in which it has been applied. Indeed, several works addressed the problem of identifying the suitable parts of a computing system for applying AxC. Then, we showed that AxC has a very wide range of application. Indeed, studies on software-level AxC, architectural-level AxC, and Circuit-level AxC have been conducted in the last two decades.

Finally, we discussed the impact of AxC on the existing test procedures for logic integrated circuits (ICs). Specifically, AxC led to the creation of a new class of IC, the Approximate ICs (AxICs). As a consequence, test procedures for AxICs has to face some challenges. However, AxICs introduced also some opportunities from which test procedures can profit to improve test outcomes. This is, indeed, the topic of this thesis: we propose techniques to suitably deal with the test of AxICs and profit from the opportunities brought along by approximate computing.

Chapter 2

Approximation-Aware (AxA) testing

Contents

In the context of approximate circuits (AxICs), test role has to be reconsidered. Indeed, in presence of a fault, the actual error value at circuit's output becomes significant. According to [86], we classify AxIC faults into two groups, i.e., *non-redundant* faults and *approximation-redundant* (ax-redundant) faults. Non-redundant faults lead to error values higher than the acceptable threshold (catastrophic faults). Those faults must be detected in the testing phase. Conversely, ax-redundant faults cause error values lower than the threshold (acceptable faults). Those faults must not lead to AxIC rejection. Therefore, in this context, the test objective is twofold:

1. avoiding that AxICs affected by non-redundant faults are shipped to the customer;
2. ensure that AxICs affected by ax-redundant faults are not rejected.

The general and fundamental underlying assumption is the single fault condition, widely used in test techniques [34]. The AxA testing key advantage is the **yield increase**. Indeed, avoiding the detection of ax-redundant faults leads to reject fewer circuits, while guaranteeing that AxICs shipped to customers still respect error constraints.

2.1 AxA testing phases

We identify three phases in AxA testing, i.e. *fault classification*, *test pattern generation*, and *test set application*. Each phase needs some adaptations, compared to the conventional testing approach, to be properly applied to AxICs. Below, we describe the different phases and introduce some useful metrics that we use all along this thesis to evaluate the AxA testing techniques.

2.1.1 AxA Fault Classification

While in conventional test techniques faults are classified into *detectable*, *redundant* and *undetectable* w.r.t. their detectability, in AxA testing the fault classification needs to be extended w.r.t. error metrics. In this perspective, the *detectable* class is extended by including, as sub-classes, the two aforementioned ax-redundant and non-redundant classes. The part of detectable faults classified as ax-redundant constitutes the *expected Yield Increase* (eYI), expressed as follows:

$$\text{eYI} = \frac{\text{ax-redundant faults}}{\text{total faults}} \tag{2.1}$$

The purpose of such a metric is to establish an upper bound to the achievable yield gain. To turn eYI in an actual gain, we have to go through the other two phases.

2.1.2 AxA Test Pattern Generation

In conventional testing, we generate input vectors to test all the faults classified as detectable. In AxA testing, test vectors should target only non-redundant faults, in order to prevent catastrophic errors at circuit outputs. Moreover, the obtained test vectors should detect as few ax-redundant faults as possible. Indeed, a test vector testing a non-redundant fault could also detect an ax-redundant fault. This, in turn, would lead to consider the AxIC as faulty, although it is still acceptable. This phenomenon is also known as *over-testing*, i.e., a good product is considered as faulty by the test process. This can lead to a yield increase lower than expected. Therefore, the concept of *test set quality* needs to be revisited by dividing the fault coverage (FC) into *ax-redundant FC* (axR FC) and *non-redundant FC* (nR FC), as defined below:

$$\text{axR FC} = \frac{\text{detected ax-redundant faults}}{\text{ax-redundant faults}} \tag{2.2}$$

$$\text{nR FC} = \frac{\text{detected non-redundant faults}}{\text{non-redundant faults}} \tag{2.3}$$

The first one has to be kept as low as possible, the second one has to be maximized.

2.1.3 AxA Test Set Application

In the conventional test set application phase, observing a circuit response different from the expected one always leads to circuit rejection. On the contrary, in AxA testing, whether the erroneous response is due to an ax-redundant fault or to a non-redundant fault must be taken into account. The test still passes if an ax-redundant fault caused the error, otherwise it fails.

We use another metric to evaluate the effect of the AxA testing procedures on the yield, the *Yield Increase Loss* (YIL), defined below:

$$\text{YIL} = \frac{\text{detected ax-redundant faults}}{\text{total faults}} \tag{2.4}$$

It describes the value of the yield increase **not achieved** due to the detection of ax-redundant faults. The YIL is in the range $[0, \text{eYI}]$. We can observe that the YIL can be expressed also as follows:

$$\text{YIL} = \text{axR FC} \cdot \text{eYI} \tag{2.5}$$

This means that the axR FC metric represents the part of eYI that is not actually achieved, after the whole test procedure application. Therefore, if $\text{axR FC} = 0$ then $\text{YIL} = 0$ (i.e., maximum yield increase). On the contrary, if $\text{axR FC} = 1$ then

YIL = eYI, thus the achieved yield increase is null.

2.1.4 Relationships between AxA test phases

Below, we describe how the three AxA testing phases influence each other.

Fault classification impact on test pattern generation The result of the fault classification impacts on the effort needed in the test pattern generation. For instance, let us consider a generic AxIC where a lot of faults are classified as ax-redundant. Generating test vectors detecting all the non-redundant faults and avoiding the detection of all the ax-redundant ones would be a hard task, maybe impossible. On the contrary, an AxIC with a few ax-redundant faults would more probably lead to a high-quality test vector generation. Ultimately, as discussed in Chapter 1, this depends on the error metric.

Test pattern generation impact on test set application The test set quality, obtained in generation phase, determines the effort necessary in test set application phase to correctly detect faults. Indeed, if the generation phase succeeds in obtaining a 0% axR FC, then no extra effort is required in test application phase compared to the conventional one. Conversely, if the axR FC cannot be kept at 0%, then extra effort is necessary in test application phase to distinguish non-redundant faults from ax-redundant ones. This aspect is especially critical in the context of self-testing hardware (see Chapter 6).

2.2 Related work

The three AxA testing phases can be separated or somehow mixed together. In [32], *threshold testing* principle was introduced and applied to conventional circuits in order to increase the production yield. Although the threshold testing was not applied in the AxICs context, it is an example of *non-conventional testing*. In this technique, the criterion to identify acceptable faults is defining a threshold based on the numerical error magnitude (see Equation 1.1) observed at circuit outputs. By imposing vector generation constraints, authors were able to produce test vectors targeting non-acceptable faults. Specifically, given an input vector, it could generate either an error higher than the threshold or lower, in presence of a detectable fault. In the first case, authors classified such fault as non-acceptable. Thus, they included the vector in the test-set. Conversely, if no input vector could sensitize above-threshold errors for the given fault, they classified it as acceptable. In this way, they were capable of classifying faults and generating test vectors only for non-acceptable faults at the

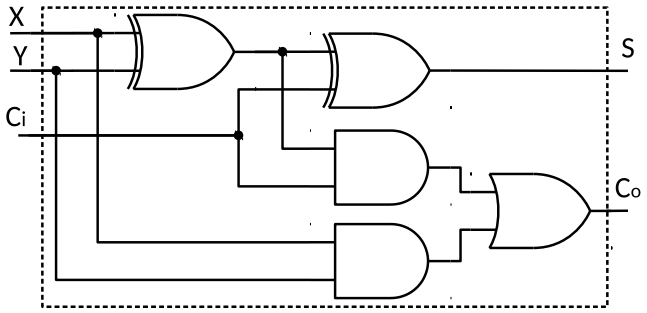
same time. Nevertheless, a test vector detecting a non-acceptable fault could still detect an acceptable one. Therefore, authors modified the test set application phase to verify whether the test responses were under the threshold or not. Threshold testing was applied only to non-approximate ICs and by considering only error magnitude metric. Thus, it can be considered as a special case of AxA testing [26]. In the next section, we introduce an illustrative example and briefly summarize the AxA testing purpose.

2.3 Illustrative example

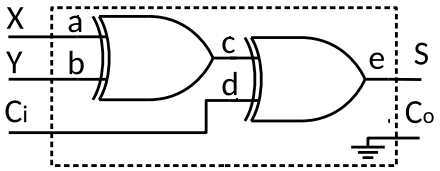
Let us now introduce the simple example in Figure 2.1. We will refer to it all along the thesis to discuss the different aspects of the AxA testing.

In the figure, we report a 1-bit Full Adder (FA) (2.1a) and an approximate version of it (2.1b). We obtained the approximate version by simply setting the output $C_o = 0$ in the FA and re-synthesizing the circuit. This functional approximation led to a more efficient circuit, i.e. with reduced area (2 logic gates instead of 5) and lower delay (2 logic levels instead of 3), but with some errors at outputs. Figure 2.1c reports the truth tables of both the circuits. For the reader convenience, we also report the integer representation of both the circuit outputs (see "Int" column). As reported in Figure 2.1d, by considering all the possible circuit inputs $i \in \mathcal{I}$, we can calculate the error values according to metrics described by Equations 1.3, 1.4, 1.5, 1.6, and 1.7. Values reported in Figure 2.1d are a direct consequence of the approximation. They constitute the error threshold values of the AxIC, fixed by specification and known at design time.

Depending on the application context within which the AxIC will be utilized, considering a specific error threshold can be more appropriate than another. Erroneous values produced by the AxIC are supposed to be never higher than the threshold considered for the final application. However, in the manufacturing phase, some defects can occur. As a result, the output's error value can unexpectedly be higher than the threshold. Therefore, the fault classification has to recognize faults leading to such condition (i.e., non-redundant faults) and faults that cause error lower than the threshold (i.e., ax-redundant faults). Afterwards, the test pattern generation aims at producing high quality test sets, i.e. able to maximize nR FC and to minimize axR FC (see Equations 2.2 and 2.3). Finally, the test set application analyzes test responses to avoid over-testing, ultimately increasing the final yield.



(A) Golden circuit



(B) Approximate Circuit

Input ($\mathcal{I}$)				Golden output ($O_i^{precise}$)			Approximate output (O_i^{approx})		
i	C_i	X	Y	C_o	S	Int	C_o	S	Int
0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	1	1	0	1	1
2	0	1	0	0	1	1	0	1	1
3	0	1	1	1	0	2	0	0	0
4	1	0	0	0	1	1	0	1	1
5	1	0	1	1	0	2	0	0	0
6	1	1	0	1	0	2	0	0	0
7	1	1	1	1	1	3	0	1	1

(C) Truth tables

Error Metric	Value
WCE	2
MAE	1
MSE	2
EP	0.5
WCBFE	1

(D) Error metric table

FIGURE 2.1: Schematics of the Full adder (a) and of its approximation (b) obtained by re-synthesizing the circuit with $C_o = 0$; (c): truth tables of both goden (i.e., non-approximate) and approximate versions. Output's integer representation for both circuits are also reported ("Int" column); (d): approximate circuit's error metric values.

2.4 Chapter summary

In this chapter we introduced the Approximation-Aware (AxA) testing. We classified faults affecting an AxIC into approximation-redundant (ax-redundant) and non-redundant. Respectively, those are acceptable and catastrophic faults.

AxA testing has two basic objectives: (i) detecting all non-redundant faults affecting an AxIC and (ii) ensure that AxICs affected by ax-redundant faults are not rejected. In particular, the second objective has e key advantage, i.e. the yield increase.

AxA testing is composed of three phases: (i) fault classification, test pattern generation and test set application. Briefly, classification has to classify faults into non-redundant (to test) and ax-redundant (not to test). Test pattern generation produces

test vectors to cover all the non-redundant faults and to leave ax-redundant ones undetected. Test set application classifies AxICs into *catastrophically faulty*, *acceptably faulty*, *fault-free*.

We presented an illustrative example to suitably describe the proposed techniques all along the thesis. We also defined some metrics to evaluate the techniques.

Chapter 3

AxA fault classification

Contents

In this chapter, firstly we discuss how the fault classification complexity is impacted by the error metric choice (Section 3.1). Indeed, as previously discussed, the error caused by a fault – along with the effort to measure it – can change significantly depending on the considered metric. Then, in Sections 3.2 and 3.3, we describe the issues related to classifying faults when considering different error metrics. Furthermore, we introduce two techniques to realize the fault classification in different conditions. Finally, in Section 3.2.3, we show related works on AxA fault classification.

3.1 Problem statement

In Table 3.1, we report the error threshold value alterations caused by all possible Stuck-at faults in the approximate FA (Figure 2.1). The fault list was generated with a commercial tool [87] with the fault collapsing option active. We highlight in red solid-bordered boxes the non-acceptable error values, i.e. higher than the respective thresholds t (Table 2.1d). Hereinafter, we use the notation *SaX@N* to indicate

Net	Fault	WCE *t=2*	MAE *t=1*	MSE *t=2*	EP *t=0.5*	WCBFE *t=1*
a	Sa0	3	1	2	0.625	2
a	Sa1	2	1.25	2	0.875	2
b	Sa0	3	1	2	0.625	2
b	Sa1	2	1.25	2	0.875	2
c	Sa0	2	1	1.5	0.75	2
c	Sa1	3	1.25	2.5	0.75	2
d	Sa0	3	1	2	0.625	2
d	Sa1	2	1.25	2	0.875	2
e	Sa0	3	1.5	3	0.875	2
e	Sa1	2	0.75	1	0.625	2

TABLE 3.1: Approximate full adder error metric values for all possible Stuck-at faults, under single-fault assumption.

a "stuck-at-X affecting the net N", where X can be either the value 1 or 0 and N is the label of the net. Please, refer to Figure 2.1-b for the net labels. By observing the table, we can firstly remark that not all the metrics are impacted by the same faults. While all the faults impact EP and WCBFE, some faults affect the WCE and not MAE and MSE, some others have an effect on the MAE and not on WCE and MSE. Furthermore, in some particular cases, faults even reduce the observed error (green dash-bordered boxes in Table 3.1). Moreover, we report in Figure 3.1-a the *error magnitude (EM) profile* of the *fault-free* approximate circuit (i.e. the circuit produces errors due to the approximation and not due to manufacturing defects); also, three EM profiles in presence of a fault are reported. Specifically, we show the EM profile

for the following faults: Sa0@a (Figure 3.1-b), Sa1@a (Figure 3.1-c), and Sa1@e (Figure 3.1-d). The figures show how the EM profile changes differently, depending on

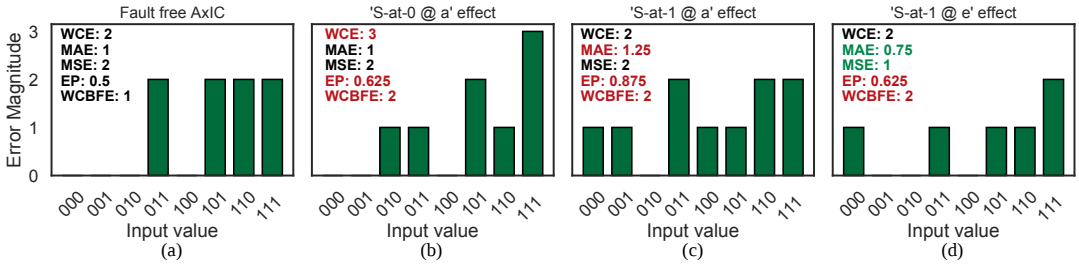


FIGURE 3.1: (a) Error profile of the fault-free approximate circuit; (b) approximate circuit error profile in presence of the S-at-0 fault at the *a* net; (c) approximate circuit error profile in presence of the S-at-1 fault at the *a* net; (d) approximate circuit error profile in presence of the S-at-1 fault at the *e* net.

the fault. As a result, the errors measured by the different metrics change, too. The fault impact on MAE and MSE depends on the variation of each bar in the graph. In other words, it depends on the EM of the AxIC for each possible input. Similarly, the impact of the fault on the EP depends on the variation of the total number of input vectors generating an error. WCE and WCBFE values change only if the maximum possible error changes, as a result of the fault.

Let us focus on WCE as the relevant metric for a final application. To classify the fault as non-redundant, it is sufficient to prove the existence of an input vector leading the error to exceed the WCE threshold (i.e., 2). This is the case for Sa0@a (Figure 3.1-b). The input vector '111' leads to $EM = 3$. Conversely, if we can prove that such input vector does not exist (as for some other faults, in the example), we can classify the fault as ax-redundant w.r.t. the WCE metric.

On the other hand, when performing the classification according to metrics such as MAE, MSE and EP, the task becomes more complex. Since each input vector contributes to the final error measure, finding a single input vector i for which the fault effect increases the EM is not enough to classify the fault as non-redundant w.r.t. MAE, MSE and EP. In fact, we could find another vector j "balancing" the effect of i. In our example, in the case in Figure 3.1-b, we can see how the vectors '010', '011', '110', and '111' "balance" each other effects: some vectors increase the error value, some others decrease it, ultimately leading to a null effect according to the metric. Therefore, we need to measure the fault impact on the final error for all possible circuit input vectors. When the complexity of the measure becomes unmanageable, a workload-dependent subset of input vectors may be used.

In conclusion, it turns out to be less complex to evaluate the impact of a fault when considering metrics for which only a single condition has to be verified, as the WCE. Henceforth, we refer to those metrics as *single-condition-test (SCT) metrics*.

Conversely, classifying faults according to metrics which involve the calculation of a mean is a $O(2^n)$ complexity problem, where n is the number of input bits. We refer to such metrics as *Mean Error metrics (ME metrics)*.

3.2 SCT-metric-aware fault classification

A common idea characterizes all the works in the literature related to AxIC fault classification, when considering SCT-metrics. To present it, let us refer to the aforementioned FA example (Figure 2.1), using the WCE as SCT metric. The maximal amount of allowed error (treshold t) according to the WCE is 2. The AxIC is considered faulty if it produces any deviation δ from the golden value which is greater than 2. Any fault f modifying the AxIC output can either lead to reject the circuit ($\delta > 2$, non-redundant fault) or not ($\delta \leq 2$, ax-redundant fault). Therefore, to classify a fault as non-redundant, the existence of an input vector I leading the faulty AxIC to exhibit $\delta > 2$ has to be demonstrated. If such vector does not exist, then the fault is classified as ax-redundant. To do so, a *delta module* calculating the deviation caused by f can easily be embedded in the scheme represented in Figure 3.2. A

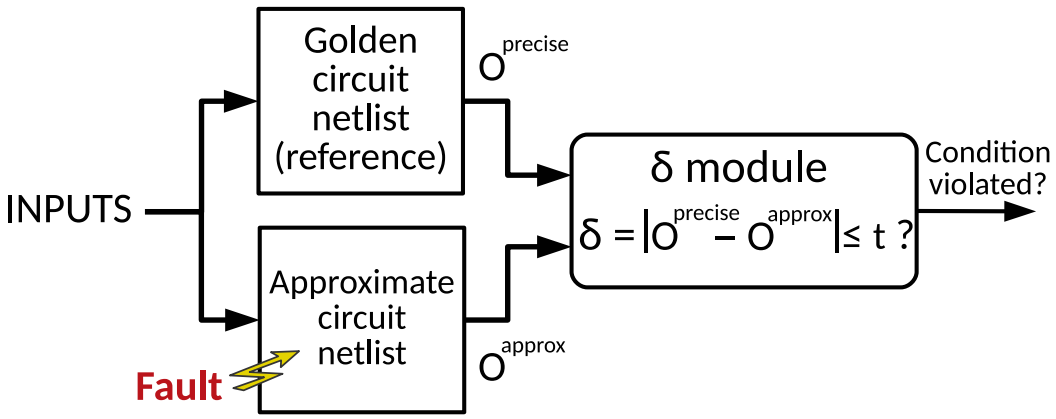


FIGURE 3.2: SCT-metric-aware classification scheme

digital circuit can easily implement the mentioned scheme. By using an automatic test pattern generation (ATPG), one can find the aforementioned input vector I, if it exists. Likewise, one can also resort to a Boolean SATisfiability problem (SAT) formulation to represent the scheme. By solving the SAT problem, one can prove whether the input vector I exists or not. Both SAT and ATPG-based techniques formally prove whether the vector I exists or not [26]. Concerning the FA example, the I vector exists for five out of ten faults, classified as non-redundant according to WCE metric (see Table 3.1).

3.2.1 Proposed technique

In [88], we propose an ATPG-based fault classification for AxICs. We exploit the efficient ATPG structural algorithms to classify faults according to the WCE metric and, at the same time, to obtain test vectors detecting non-redundant faults (see Chapter 4). As mentioned, the underlying idea is to embed the AxIC into the architecture shown in Figure 3.2. Let us call it *ax-redundant fault masking (AxRFM) architecture*. Then, we use the ATPG to correctly classify the faults. The AxRFM fundamental property is that, in absence of faults, no inputs violates the delta module condition (i.e., no unacceptable errors are produced). Therefore, only faults leading to an EM greater than the threshold can violate the delta module condition. As a result, the ax-redundant faults cannot violate the delta condition, thus they cannot be detected by the ATPG. Figure 3.3 depicts the overall flow of the proposed approach, that is

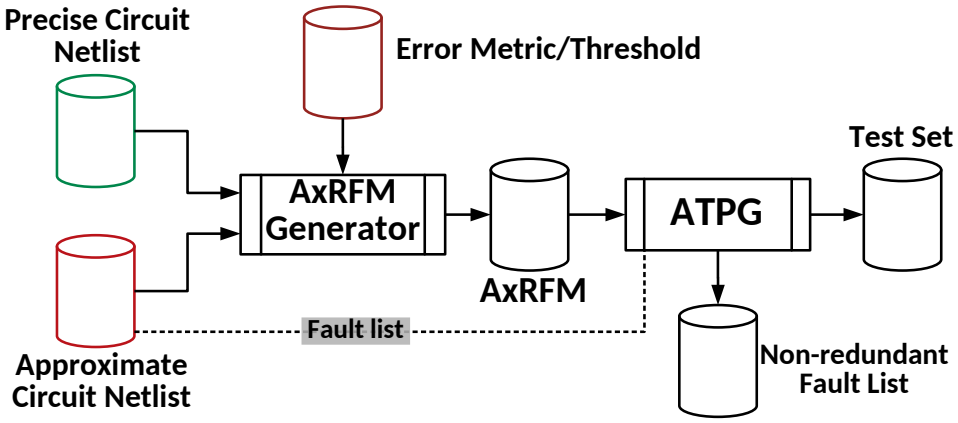


FIGURE 3.3: A schematic view of the proposed flow

composed of two main steps:

1. the AxRFM Generation,
2. the ATPG.

The first step requires as inputs both precise and approximated circuit netlists, the knowledge of the SCT error metric and the corresponding threshold. The obtained AxRFM is able to determine whether a given input vector produces unacceptable errors (i.e., an output with an EM exceeding the threshold) or not. By targeting only AxIC's faults, in the second step the ATPG will be able to sensitize only non-redundant faults. As it can be imagined, the AxRFM will never be manufactured. Its only purpose is the fault classification at design time.

3.2.2 Experimental results

We applied the ATPG-based classification technique on a large set of approximate arithmetic circuits from the public approximate component library EvoApprox8b [25]. Specifically, we carried out experiments on more than 1100 different AxICs, namely 8-bit adders (Add8), 8-bit, 16-bit and 32-bit multipliers (Mul8, Mul16, Mul32). Authors of [25] obtained adders by functional approximation of a Ripple-Carry Adder (RCA), a Carry-Select Adder (CSA), a Carry-Look-ahead Adder (CLA), a multiple Tree Adder (TA) and a Higher Valency Tree Adder (HVTA). As for multipliers, they were obtained by functional approximation of Ripple-Carry Array, multiple Carry-Save Array and Wallace Tree architectures. Table 3.2 reports, for each family, the

Circuit family	Units	WCE range
Add8	448	1 - 168
Mul8	471	1 - 3204
Mul16	60	$3.9 \cdot 10^4$ - $8.5 \cdot 10^8$
Mu32	153	$7.3 \cdot 10^{10}$ - $1.8 \cdot 10^{18}$

TABLE 3.2: EvoApprox8b Circuits' WCE range

number of circuits (*Units* column) and the WCE range. For example, in the 8-bit Adders family we have 448 circuits with a WCE between 1 and 168. In the experiments, we resorted to Stuck-at-Fault (SaF) and Transition Fault (TF) models and a commercial ATPG [87], instrumented using the conventional options. Table 3.3 shows experimental results from [88]. As discussed in Section 2.1, the results of

	SaF				TF			
	eYI*			Avg.	eYI*			Avg.
	Avg.	Max.	Min.	Time(s)	Avg.	Max.	Min.	Time(s)
Add8	19%	99%	0%	0.64	25%	99%	0%	0.64
Mul8	55%	85%	1%	0.91	55%	84%	1%	1.01
Mul16	62%	94%	28%	0.96	64%	97%	23%	1.04
Mul32	85%	99%	41%	2.60	87%	99%	42%	2.78

*eYI: expected Yield Increase

TABLE 3.3: ATPG-based fault classification results [88], in terms of expected Yield Increase (eYI)

the AxA Fault Classification phase can be described by using the expected Yield Increase (eYI) metric (Equation 2.1), which measures the portion of faults classified as ax-redundant. Therefore, for a given AxIC, eYI value expresses the upper bound for the final achievable yield gain. In the experiments, for the majority of the circuits, eYI was above 50%, on average. Only for 8-bit adders it was on average around 19%, when using the SaF model, and 25%, when using the TF model. Concerning the execution time, we performed all the classifications in less than 3 seconds, on average.

3.2.3 Related works

In the literature, some studies dealing with AxA fault classification according to SCT-metrics have been presented. In [86], authors propose a SAT-based solution. Briefly, a SAT problem is defined as the problem of determining whether a combination of Boolean variables assignments satisfying a given Boolean formula exists, (i.e., the final value of the formula is TRUE). Basically, they exploit the so-called *Approximation Miter* (AxMi) presented in [89] to perform the fault classification. For each fault in the fault list, an AxMi is constructed, having the structure shown in Figure 3.2. Then, they derive a SAT problem instance out of it. The resolution of the SAT problem provides the fault classification. With the AxMi designed in this way, it is possible to identify a combination of Boolean variable values such that the given error threshold is violated. If an input is found, the current fault is classified as non-redundant; otherwise, it is classified as approximation-redundant. Afterwards, they use conventional ATPG to generate test vectors, by targeting only the non-redundant faults. They address only SCT metrics (i.e., WCE and WCBFE). Authors performed experiments on a large set of AxICs. Specifically, they used 16-bit adders from [11], [22], [23], [70], some arithmetic designs proposed in [90], along with EPFL [91] and ISCAS-85 [92] benchmarks. All the mentioned circuits are precise. Therefore, they applied a previously proposed dedicated approximation scheme [93]. Furthermore, they performed experiments on 11 state-of-the-art approximate 16-bit adders from [11], [22], [23], [70]. They targeted the Stuck-at-Fault model. Table 3.4 reports experimental results from [86]. Results show significant

	WCE				WCBFE			
	eYI*			Avg.	eYI*			Avg.
	Avg.	Max.	Min.	Time(s)	Avg.	Max.	Min.	Time(s)
Ax 16-bit adders[1]	58%	71%	42%	16	29%	68%	10%	5
Arith designs[2]	47%	77%	18%	3355	35%	81%	3%	89
EPFL bench[3]	33%	62%	7%	5833	44%	62%	11%	10291
ISCAS-85 bench[4]	40%	81%	9%	302	45%	78%	3%	1517

*eYI: expected Yield Increase
[1]from [11], [22], [23], [70] [2]from [90] [3]from [91] [4]from [92]

TABLE 3.4: SAT-based fault classification results [86] in terms of expected Yield Increase (eYI)

expected Yield Increase (eYI) (Equation 2.1) values, especially when considering the WCE metric. Indeed, on average, the eYI is between 33% and 58% with a maximum of 81%. For the WCBFE metric, results show an average between 29% and 45% with again a maximum of 81%.

In [94] another SAT-based solutions is proposed to classify faults according to the

WCE metric and obtain test vectors to detect non-redundant faults (see Chapter 4). Authors performed experiments on some circuits developed in [24]. Specifically, they used some floating-point circuits (Adder, Comparator, Multiplier, Divider, and Sqrt) and two fixed-point circuits (Multiplier and Divider) with variable fraction parts (5, 8 and 11 bits). They targeted the Stuck-at-Fault model and considered different accepted error margins ranging from 5% to 30%. In [94] authors expressed their results in terms of Fault Reduction (FR). For example, for a non-redundant faults reduction from 100 to 80 (thus leaving 20 ax-redundant faults), they expressed the result as FR $= \frac{100}{80} = 1.25$. To be consistent with other results reported, we converted their results in terms of eYI by applying the subsequent formula:

$$\text{eYI} = 1 - \frac{1}{\text{FR}}. \tag{3.1}$$

In the above example, eYI $= 1 - \frac{1}{1.25} = 0.2$. By resorting to Equation 2.1, we find the correct number of ax-redundant faults, i.e. 20.

		Floating-point circuits						Fixed-point multiplier						Fixed-point divider					
	Error margin	5%	10%	15%	20%	25%	30%	5%	10%	15%	20%	25%	30%	5%	10%	15%	20%	25%	30%
eYI*	Avg.	13%	32%	52%	64%	73%	78%	20%	43%	55%	65%	71%	76%	15%	37%	52%	63%	69%	73%
	Max.	19%	39%	58%	71%	80%	84%	30%	55%	66%	73%	77%	80%	23%	50%	64%	72%	76%	80%
	Min.	7%	29%	47%	58%	65%	68%	7%	20%	35%	44%	53%	62%	3%	14%	29%	39%	47%	55%
Average Time (s): 4376																			

*eYI: expected Yield Increase

TABLE 3.5: SAT-based fault classification results [94] in terms of expected Yield Increase (eYI)

In Table 3.5, we report the eYI results. Also in this work, results show a significant eYI. On average, they obtained result in the range between 13% and 78% eYI.

3.2.4 Comparison

Now, let us make some observations. As reported in [26], both SAT and ATPG-based techniques formally prove whether a fault is non-redundant or ax-redundant. Therefore, state-of-the-art techniques [86], [94], as well as our proposed one [88] (Subsection 3.2.1), are able to correctly classify faults into non-redundant and ax-redundant. Moreover, circuits on which the experiments have been performed are different. However, they are fairly comparable in size and complexity. Thus, to compare the proposed technique with the state-of-the-art ones, we can refer to the execution time.

The technique presented in [86] required, on average, from 5 to $1.0291 * 10^4$ seconds to complete the classification task for different circuit classes (see Table 3.4). Similarly, the technique presented in [94] reported an average execution time of $4.376 * 10^3$ seconds (see Table 3.5). On the contrary, our proposed technique entails

a much shorter execution time. Indeed, as reported in Table 3.3, proposed technique average execution time is shorter than 3 seconds.

3.3 ME-metric-aware fault classification

As highlighted in Section 3.1, classifying faults according to ME metrics is more complex compared to SCT metrics. As shown in Figure 3.1-b, concerning MAE and MSE metrics (Equations 1.4 and 1.5) we are interested in studying the impact of a fault on the error magnitude for all input combinations (i.e., the variation of $EM_i \; \forall i \in \mathcal{I}$). Alternatively, an application-workload-related subset of input vectors $\mathcal{J} \subset \mathcal{I}$ can be used. The goal is to understand whether a fault impact increases or not the value of the sum of all the errors, for all input combinations (i.e., the term $\sum_{\forall i \in \mathcal{I}} \left| O_i^{\text{approx}} - O_i^{\text{precise}} \right|$). As for EP (Equation 1.6), we want to measure the impact of a fault on the number of inputs combinations which cause $O^{\text{approx}} \neq O^{\text{precise}}$.

3.3.1 Proposed technique

In [95], we address the fault classification problem by considering ME metrics. We propose the *Fault Filtering Architecture* (FFA) shown in Figure 3.4. Given a fault, an input vector i, and a ME metric, the FFA is able to determine whether such fault changes or not the metric value for the given vector (i.e., a single bar in Figure 3.1-a) and also to compute the magnitude of the error variation (δ_i in the figure). Moreover, since we measure the error variation δ_i, we do not need to know the actual error threshold value. To show the idea behind this approach, we consider a faulty (fa)

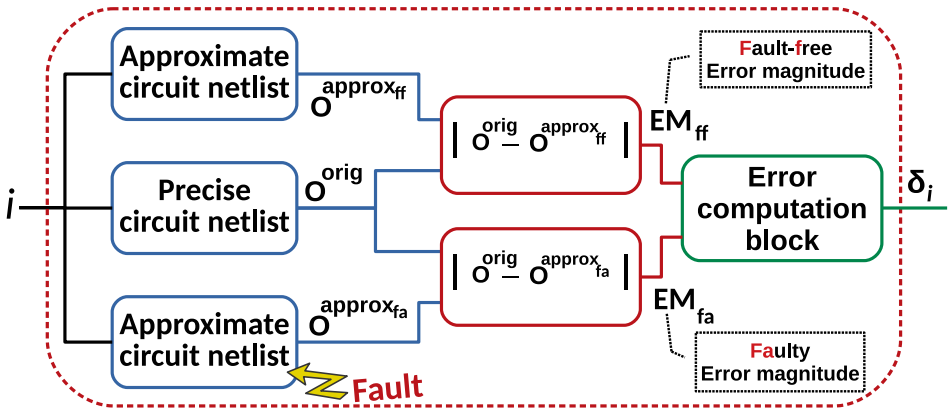


FIGURE 3.4: Fault Filtering Architecture (FFA)

generic AxIC affected by a fault f and a fault-free (ff) AxIC. We use the MAE metric

(Equation 1.4) to calculate the error in both cases, as follows:

$$MAE_{ff} = \frac{\sum_{\forall i \in \mathcal{I}} \left| O_i^{\text{approx}_{ff}} - O_i^{\text{orig}} \right|}{2^n} = \frac{\sum_{\forall i \in \mathcal{I}} EM_{ff_i}}{2^n} \tag{3.2}$$

$$MAE_{fa} = \frac{\sum_{\forall i \in \mathcal{I}} \left| O_i^{\text{approx}_{fa}} - O_i^{\text{orig}} \right|}{2^n} = \frac{\sum_{\forall i \in \mathcal{I}} EM_{fa_i}}{2^n} \tag{3.3}$$

Then, since we are interested in the MAE variation, we calculate the difference (δ), as follows:

$$\delta\left(MAE\right) = MAE_{fa} - MAE_{ff} = \frac{\sum_{\forall i \in \mathcal{I}} EM_{fa_i} - EM_{ff_i}}{2^n} = \frac{\sum_{\forall i \in \mathcal{I}} \delta\left(MAE\right)_i}{2^n} \tag{3.4}$$

The $\delta\left(MAE\right)_i$ value is the output of the FFA when input i is applied and MAE metric is considered. Equation 3.4 represents the target value of the investigation: the variation of the metric value, due to the fault. To obtain $\delta\left(MAE\right)$, the evaluation of $\delta\left(MAE\right)_i, \forall i \in \mathcal{I}$ has to be performed. Finally, if $\delta\left(MAE\right)$ is less than or equal to zero, then the fault f is classified as ax-redundant and filtered. Otherwise, the fault is classified as non-redundant. The same considerations can be applied to the MSE metric. Thus, the number of faults that will be filtered is the same for the two metrics.

Concerning EP metric, let us introduce the following function:

$$u(EM_i) = \begin{cases} 1, & \text{if } EM_i > 0 \\ 0, & \text{if } EM_i = 0 \end{cases} \tag{3.5}$$

By combining Equation 3.5 with EP metric (Equation 1.6), we calculate the EP of fault-free (ff) and faulty (fa) generic AxICs as follows:

$$EP_{ff} = \frac{\sum_{\forall i \in \mathcal{I}} u\left(EM_{ff_i}\right)}{2^n} \tag{3.6}$$

$$EP_{fa} = \frac{\sum_{\forall i \in \mathcal{I}} u\left(EM_{fa_i}\right)}{2^n} \tag{3.7}$$

$$\delta\left(EP\right) = EP_{fa} - EP_{ff} = \frac{\sum_{\forall i \in \mathcal{I}} u\left(EM_{fa_i}\right) - u\left(EM_{ff_i}\right)}{2^n} = \frac{\sum_{\forall i \in \mathcal{I}} \delta\left(EP\right)_i}{2^n} \tag{3.8}$$

The $\delta\left(EP\right)_i$ value is the output of the FFA when input i is applied and EP metric is considered. To obtain $\delta\left(EP\right)$, the evaluation of $\delta\left(EP\right)_i, \forall i \in \mathcal{I}$ has to be performed.

Circuit family	Units	EP range	MAE range	MSE range
Add8	448	0.4% - 98.4%	0.25 - 31.5	0.25 - 3040
Mul8	471	0.1% - 99.1%	0.248 - 649.211	$0.248 - 7.286 \cdot 10^5$
Mul16	60	100%	$5.92 \cdot 10^3 - 3.69 \cdot 10^8$	N/A
Mu32	153	100%	$1.36 \cdot 10^{10} - 8 \cdot 10^{17}$	N/A

TABLE 3.6: EvoApprox8b Circuits' EP, MAE and MSE ranges

If the $\delta\,(EP)$ value is less than or equal to zero, then the fault f can be considered as ax-redundant. Otherwise, the fault is classified as non-redundant.

In conclusion, by using the exhaustive set of input vectors $\mathcal{I}$, we perform the classification. An application-workload-related subset $\mathcal{J} \subset \mathcal{I}$ can be used if the complexity of $\mathcal{I}$ is not manageable. Simulating vectors belonging to $\mathcal{I}$ (or $\mathcal{J}$), while

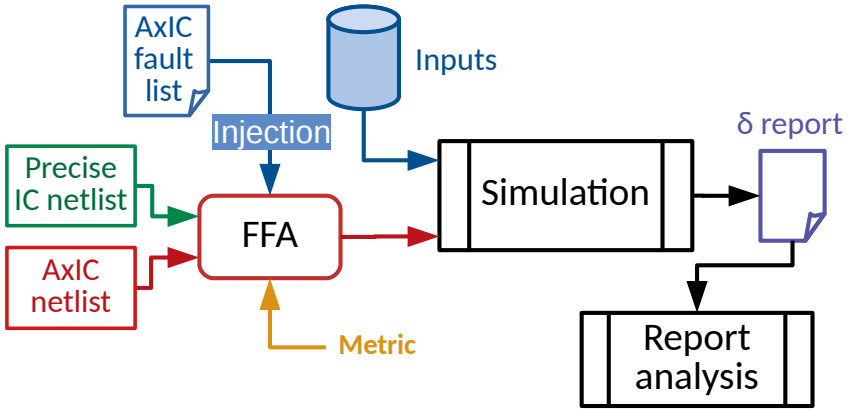


FIGURE 3.5: Overall flow

injecting – one by one – all the faults, allows us to measure all the δ_i values. Figure 3.5 sketches the overall flow. The FFA is never manufactured. It is only used to support the fault classification.

3.3.2 Experimental Results

In [95], we applied the FFA-based technique on small circuits (i.e., 8-bit adders and multipliers from EvoApprox8b library [25]) by using the exhaustive set of input vectors $\mathcal{I}$. The simulation produced a detailed report about the fault impact on the EM profile. In Table 3.7, we report the results. We performed the fault classification

	MAE and MSE				EP			
	eYI*			Avg.	eYI*			Avg.
	Avg.	Max.	Min.	Time(s)	Avg.	Max.	Min.	Time(s)
Add8	2%	12%	0%	448	1%	9%	0%	107
Mul8	7%	21%	0%	72165	3%	10%	0%	924

*eYI: expected Yield Increase

TABLE 3.7: ME-metric-aware fault classification results of [95], in terms of expected Yield Increase (eYI)

by using MAE, MSE and EP metrics and the Stuck-at-fault model. It was possible to perform the analysis of both MAE and MSE metrics with the same experiments. Therefore, the eYI obtained was the same. In the case of multipliers, up to 21% eYI was obtained when analyzing the MAE and MSE metrics and up to 10% when evaluating EP metric. For 8-bit adders, we achieved up to 12% eYI when considering MAE and MSE metrics and up to 9% in the case of EP. When looking at the average results, for 8-bit multipliers, 7% eYI was achieved for MAE and MSE and 3% for the EP. For 8-bit adders, only 2% eYI for MAE and MSE and 1% for EP were attained. Concerning the average execution time, results showed that it is quite long. This is due to the intrinsic complexity of the problem.

Afterwards, we extended the experimental results by adding those obtained with the rest of the EvoApprox8b library [25], specifically the 16-bit and 32-bit approximate multipliers. The whole set of possible inputs for 16-bit multipliers is composed of 2^{32} vectors (i.e., all the combinations of two 16-bit operands). For 32-bit multipliers, we reach 2^{64} vectors. Therefore, exhaustive analysis is quite time- and energy-consuming. A workload-dependent analysis helps to cope with such a high complexity. Thus, we performed experiments by using an input vector subset $\mathcal{J} \subset \mathcal{I}$ generated randomly. In Table 3.8, we report results obtained with a random input vector set composed of 2^{12} vectors. Note that the table reports only results for

	MAE and MSE			
	eYI*			Avg.
	Avg.	Max.	Min.	Time(s)
Mul16	12%	61%	1%	181
Mul32	21%	82%	1%	1765

*eYI: expected Yield Increase

TABLE 3.8: ME-metric-aware fault classification results for random workload experiments, in terms of expected Yield Increase (eYI).

MAE and MSE, since EP values are already 100% by design (i.e., due to the approximation), for the examined circuits (see Table 3.6). Consequently, all the faults are ax-redundant by design. As reported in Table 3.8, an average of 12% eYI was achieved for 16-bit multipliers and 21% for 32-bit multipliers. When analyzing results, we have also to bear in mind that examined circuits have intrinsic quite high ME-metrics thresholds, due to aggressive approximation (see Table 3.6). This contributes to the higher eYI values. As expected, execution time of workload-related experiments is reduced, compared to exhaustive ones.

In conclusion, the task itself is complex if addressed exhaustively. Nevertheless, if workload-dependent analysis are carried out, complexity becomes manageable. In the context of approximate computing this a fair assumption, since the performed

approximations are application-dependent. As a consequence, application-related workload can be used to classify faults.

As last observation, no other techniques to classify faults according to ME-metrics have been proposed, so far. Therefore, any comparisons would not be significant.

3.4 Chapter summary

In this chapter we presented the AxA fault classification. We showed and discussed the issues related to the fault classification in the context of AxICs. In particular, we observed that the complexity of the task drastically changes depending on the considered error metric. We showed how some metrics – referred to as Single Condition Test (SCT) metrics – entail a smaller effort for the fault classification compared to metrics based on the calculation of a mean – referred to as Mean Error (ME) metrics.

Firstly, we presented a technique to deal with fault classification when considering SCT metrics. The technique is based on a *classifying architecture*. Such a structure allows classifying faults into non- and ax-redundant by measuring their impact on AxIC's output. The basic idea is to "hide" ax-redundant faults by using a *filtering box*. Thus, for a given fault, an anomaly condition is generated only if the fault leads to catastrophic output errors. The classifying architecture is never manufactured. It is only used at design time to classify faults. We also presented related works in the literature dealing with AxA fault classification. By comparing results obtained with our proposed technique with those obtained by state-of-the-art techniques, we highlighted the reduced execution time entailed by our proposition (less than 3 seconds, on average).

Secondly, we presented a technique to deal with fault classification when considering ME metrics. Again, the idea is to individuate non-redundant faults by using a filtering mechanism. For a given fault, we were able to state whether the consequent errors were catastrophic or acceptable. So far and to the best of our knowledge, the presented technique is the first of its kind.

Chapter 4

AxA test pattern generation

Contents

In this chapter we discuss the AxA test pattern generation problem and show our propositions to address it. As discussed in Section 2.1, the role of AxA test pattern generation is twofold: (i) test vectors should detect all non-redundant faults, in order to prevent catastrophic errors at circuit outputs; (ii) the test set should detect as few ax-redundant faults as possible, in order to not consider the AxIC as faulty when it is still acceptable. In other words, a qualitatively good test set should achieve 100% *non-redundant FC* (nR FC) and 0% *ax-redundant FC* (axR FC). However, two problems can affect the test pattern generation procedure, as far as it concerns AxICs:

1. in order to achieve 100% nR FC, it is not always possible to avoid testing some ax-redundant faults (i.e., axR FC > 0%);

2. conventional test generation procedures might not be able to achieve a qualitatively good test set.

The first problem is intrinsic to the structure of the AxIC under test, the second one is relative to conventional test generation algorithms. Consequently, a still-good AxIC affected by an ax-redundant fault would be rejected in test phase, leading to a yield decrease. The phenomenon due to which a good product is considered as faulty by the test process is commonly referred to as *over-testing*. This phenomenon, if not properly managed, will eventually cause some yield reduction.

Let us put aside for a moment the first problem. Discussion and propositions regarding it are postponed to Chapter 5. In Section 4.1, we discuss the second problem in details. In Section 4.2 we show how the technique presented in Subsection 3.2.1 [88] partially addresses the issue. Then, in Section 4.3, we present a new technique designed specifically to address AxA test pattern generation.

4.1 Problem statement

Let us refer to the FA example in Figure 2.1 to illustrate the mentioned problems. In Table 4.1, we indicate again the error threshold value alterations caused by all possible Stuck-at faults in the approximate FA. Furthermore, we report all the input vectors detecting each fault. Firstly, let us assume that the fault classification is performed by using the MSE metric (threshold $t = 2$). Table 4.1 shows that two faults lead the error to be catastrophic, Sa1@c ($MSE = 2.5$) and Sa0@e ($MSE = 3$). Both vectors 4 and 7 detect the two faults. However, both vectors detect also three ax-redundant faults (37.5% axR FC). Moreover, there is no vector detecting all the non-redundant faults and achieving 0% axR FC. This highlights the first aforementioned problem: to achieve 100% nR FC, it is not always possible to have also 0% axR FC. This, in turns, leads to a yield increase lower than expected (YIL > 0). The same

Net	Fault	WCE *t=2*	MAE *t=1*	MSE *t=2*	EP *t=0.5*	WCBFE *t=1*	Test vectors* 0	1	2	3	4	5	6	7
a	Sa0	3	1	2	0.625	2			x	x			x	x
a	Sa1	2	1.25	2	0.875	2	x	x			x	x		
b	Sa0	3	1	2	0.625	2		x		x		x		x
b	Sa1	2	1.25	2	0.875	2	x		x		x		x	
c	Sa0	2	1	1.5	0.75	2		x	x			x	x	
c	Sa1	3	1.25	2.5	0.75	2	x			x	x			x
d	Sa0	3	1	2	0.625	2					x	x	x	x
d	Sa1	2	1.25	2	0.875	2	x	x	x	x				
e	Sa0	3	1.5	3	0.875	2		x	x		x			x
e	Sa1	2	0.75	1	0.625	2	x			x		x	x	

*0="000", 1="001",..., 7="111"

TABLE 4.1: Approximate full adder test vectors for all possible Stuck-at faults, under single-fault assumption.

phenomenon occurs when considering the MAE metric (threshold $t = 1$). In this case, five non-redundant faults are detected (Sa1@a, Sa1@b, Sa1@c, Sa1@d, Sa0@e). The best test vector combination turns out to be {0, 4}, having 100% nR FC and still 40% axR FC. We can find other combinations, such as {0, 1}, {0, 2}, and {1, 4}, which achieve 100% nR FC but also 60% axR FC. Thus, they have a *lower quality*. So, we begin to see the second mentioned problem, well illustrated by resorting to the WCE metric (threshold $t = 2$). Five non-redundant faults emerge from the classification (Sa0@a, Sa0@b, Sa1@c, Sa0@d, Sa0@e). Among all the vector combinations that test the five faults, some have a higher quality than others. For example, the combination {1, 3, 6} attains 100% nR FC but also 100% axR FC. The combination {3, 4} achieves 100% nR FC and 80% axR FC. The best solution is to use only the vector {7}, which indeed covers 100% of non-redundant faults, while having 0% axR FC. An ideal *AxA test pattern generation* technique should produce the qualitatively best test set for the relative metric.

Conventional ATPG algorithms do not give any guarantee of high-quality test vector generation, when it comes to AxICs. To illustrate the phenomenon, we used a commercial ATPG [87] to create test sets for the approximate FA of our example (Figure 2.1). We instrumented the ATPG using the conventional options (static and dynamic compaction) and used the Stuck-at-Fault model. For each metric, we used the corresponding non-redundant fault list and we executed the ATPG to generate test vectors. This is the test flow used in other state-of-the-art works [27], [86], [95]. In Table 4.2, we report, for each metric, the solutions obtained with conventional ATPG in term of test sets, axR FC and YIL, along with the ideal solutions (i.e., solutions that an ax-aware generation should find: 100% nR FC and axR FC as low as possible). We report also the non-redundant fault lists obtained from the fault classification,

	Conventional ATPG solution[1]			Ideal solution[1]		
Metric	Test sets	axR FC[2]	YIL[2]	Test sets	axR FC[2]	YIL[2]
MSE	{4}	37.50%	30%	{4} or {7}	37.50%	30%
WCE	{1, 6, 0}	100.00%	50%	{7}	0.00%	0%
MAE	{1, 7, 0}	100.00%	50%	{0, 4}	40.00%	20%

[1]100% non-redundant FC always achieved [2]Lower is better

non-redundant fault lists:
MSE: Sa1@c, Sa0@e
WCE: Sa0@a, Sa0@b, Sa1@c, Sa0@d, Sa0@e
MAE: Sa1@a, Sa1@b, Sa1@c, Sa1@d, Sa0@e

axR FC = Ax-redundant FC
YIL = Yield Increase Loss

TABLE 4.2: Test vector generation results when using an ideal ax-aware test vector generation and a conventional ATPG tool [87] on the example circuit in Figure 2.1.

for each metric. Firstly, as expected, 100% nR FC was achieved for all experiments. For the MAE metric, the conventional ATPG test set was {1,7,0} and led to an axR FC of 100% (50% YIL). As already mentioned, the best test set {0,4} achieves 40% axR FC (20% YIL). Concerning the WCE metric, results were even worse: conventional ATPG generated {1,6,0}, leading to 100% axR FC (50% YIL), while the single vector {7} achieves 0% axR FC (0% YIL). Only for MSE metric results were optimal with both approaches. Note that when considering EP or WCBFE, all the faults are non-redundant and need to be tested. In this case, no approximation-aware technique is needed. Therefore, we did not include EP and WCBFE metrics in the experiment. In conclusion, conventional ATPG techniques do not guarantee qualitatively the best solutions. This is due to the fact that state-of-the-art ATPG algorithms have not been designed to avoid testing some faults while generating test vectors.

An ideal AxA test pattern generation technique generates qualitatively the best possible test set. Nevertheless, the AxIC structure may still lead to obtain axR FC > 0%, as shown in the example above. This problem is addressed in Chapter 5. In next section, we report the techniques that we propose to address the AxA test pattern generation.

Proposed AxA pattern generation techniques

We propose basically two techniques to deal with AxA test pattern generation. The first one, proposed in [88] and referred to as *ax-aware generation* technique, is shown in Subsection 4.2. The application of the first technique is limited to some particular cases. Therefore, in [96], we proposed a more general technique based on a careful *pattern selection*. We show it in Subsection 4.3.

4.2 An Ax-aware technique

As discussed in Section 3.2, in [88] we propose an ATPG-based technique to classify faults into ax-redundant and non-redundant, when considering an SCT metric. Simultaneously the technique produces test vectors to detect only non-redundant faults. By resorting to Figures 3.2 and 3.3, one can see that – thanks to the delta module – the ATPG produces test vectors which activate and sensitize only non-redundant faults. This implies that:

- test vectors testing all the non-redundant faults are produced;
- when the AxIC is affected by a non-redundant fault, at least one test vector produces an output O^{approx} such that the deviation $\delta = \left|O^{precise} - O^{approx}\right| > t$.

Nevertheless, in test phase – i.e. after AxIC manufacturing (thus without delta module) – some test vectors might still detect an ax-redundant fault affecting the AxIC. This is due, in part, to the fact that test vectors are generated to detect only non-redundant faults but no effort was made to minimize the axR FC. Moreover, the technique is only applicable when the fault classification is made by using an SCT metric. These consideration are also applicable to works in [32], [94]. For these reasons, we introduced a more general technique based on a careful *pattern selection*.

4.3 An ILP-formulated Pattern Selection Procedure

In [96], we propose a new *Approximation-Aware ATPG* (Ax-Aware ATPG) whose goal is to produce test vectors reducing the axR-FC, compared to conventional ATPG, while not impacting nR-FC. This ultimately leads to actually increase the yield. The novel technique relies on a new engine capable of finding, among a set of input vectors, the smallest subset minimizing the axR-FC coverage. Specifically, the engine generates an input vector set $\mathcal{S}$ and searches within it for the optimal combination $\mathcal{V}$ which attains the required coverage. Generally, the set $\mathcal{S}$ will be itself a sub-set of the exhaustive input vector set. Indeed, while for the FA example introduced in Section 2.3 it was feasible to search within the exhaustive vector set (see Section 4.1), this will not be feasible for big circuits. In line with all conventional ATPG techniques, the ax-aware ATPG only needs as inputs the AxIC under test and its fault list. The only additional constraint is that the AxIC's faults have to be formerly classified into non- and ax-redundant. This is possible by using one of the state-of-the-art fault classification techniques (see Chapter 3). Figure 4.1 depicts the proposed test generation flow. While the approach used in previous works – here referred to as *conventional generation* – uses only non-redundant fault list to generate test vectors,

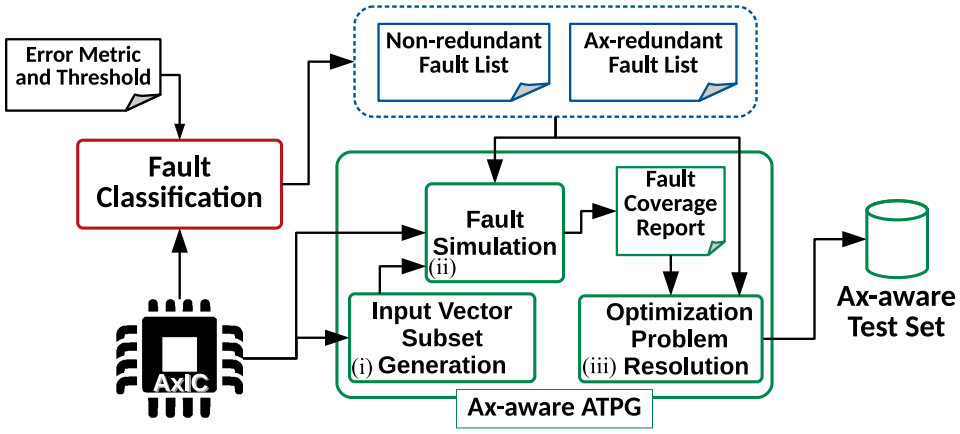


FIGURE 4.1: Proposed Approximation-Aware ATPG

the proposed one utilizes also the ax-redundant fault list. In this way, it is possible to discern which faults have not to be covered. Hereafter, we detail the proposed ATPG approach, as illustrated in Figure 4.1. Ax-aware ATPG is composed of three main phases: (i) input vector subset generation, (ii) fault simulation and (iii) optimization problem formulation and resolution. The first phase takes as input the AxIC and generates a configurable number of input vectors. In this phase, different algorithms for input vector generation can be used. As already discussed, all of them should generate a sub-set $\mathcal{S}$ of the exhaustive vector set. The final result will change depending on the generated $\mathcal{S}$. Fault simulation phase takes as input the generated $\mathcal{S}$, the AxIC and the two fault lists (ax-redundant and non-redundant). The output of the fault simulation phase is a *fault coverage (FC) report* which records, for each fault, all the input vectors in $\mathcal{S}$ covering it. Finally, the goal of the third phase is to find the smallest subset $\mathcal{V} \subset \mathcal{S}$ which minimizes the axR-FC and achieves total nR-FC. If $\mathcal{S}$ corresponds to the exhaustive vector set (see example in Section 4.1), the output solution will be the global optimal one (i.e., the best possible vector combination). When $\mathcal{S}$ is a sub-set of the exhaustive vector set, the third phase will produce a *local optimal final solution* (i.e., the best combination, among vectors in $\mathcal{S}$). To accomplish this task, we formulate an optimization problem, by using the fault coverage report, the vector set $\mathcal{S}$ and the fault lists. This leads to a system of linear inequalities whose solution will be the final *ax-aware test set*.

The proposed ax-aware ATPG is independent of the specific fault classification technique and of the error metrics and thresholds. Indeed, as long as a fault classification is correctly produced, the ax-ATPG is applicable. In the next subsection, we briefly recall the mathematical formalization of an optimization problem and, specifically, of an integer linear programming problem. Indeed, the ax-aware ATPG problem, as we formulate it, falls into this category.

4.3.1 Optimization problem

Largely used in several fields of Engineering, Finance and Economics, *mathematical optimization* helps with the selection of an optimal element, among a set of available alternatives, while respecting some criteria. In practice, mathematical optimization aims at finding the minimum/maximum of a so-called *objective function* $f(x_0, x_1, .., x_{n-1})$ by systematically assigning values to its so-called *decision variables* $(x_0, x_1, .., x_{n-1})$. The final solution has to respect a given set of constraint equations. The geometric region delimited by those constraints is usually called *feasible region*. A *linear programming* (LP) problem is an optimization problem where the objective function is linear, meaning that it has the following form:

$$f(x_0, ..., x_{n-1}) = c_0 x_0 + c_1 x_1 + ... + c_{n-1} x_{n-1} \tag{4.1}$$

for some coefficients $c_i \in \mathcal{R},\ i = 0, ..., n-1$. The feasible region is the set of solutions to a finite number of linear inequalities, of the form:

$$\begin{aligned} a_{0,0}x_0 + a_{0,1}x_2 + ... + a_{0,n-1}x_{n-1} &\leq b_0 \\ a_{1,0}x_1 + a_{1,1}x_2 + ... + a_{1,n-1}x_{n-1} &\leq b_1 \\ &\vdots \\ a_{m-1,0}x_1 + a_{m-1,1}x_2 + ... + a_{m-1,n-1}x_{n-1} &\leq b_{m-1} \end{aligned} \tag{4.2}$$

for some coefficients $a_{ji} \in \mathcal{R},\ j = [0, m-1] \quad i = [0, n-1]$.
Furthermore, an LP problem whose variables are restricted (totally or partially) to be integers is referred to as an *integer linear programming* (ILP) problem. An ILP problem can be expressed in a canonical form, as follows:

$$\begin{cases} min/max & \overline{c}^T \overline{x} \\ subject\ to & \overline{\overline{A}}\overline{x} \leq \overline{b}, \\ & \overline{x} \geq 0, \\ and & \overline{x} \in \mathbb{Z}^n. \end{cases} \tag{4.3}$$

We use the notation $\overline{\overline{A}}$ and $\overline{x}$ to express that A is a matrix and x is a vector. Finding a solution to the above system of linear inequalities means finding the objective function's min/max values which lies in the feasible region. Thus, the solution will be a combination of the decision variable values.

In the next subsection, we detail how we represent the ax-aware ATPG problem as an ILP problem.

4.3.2 Ax-aware ATPG as an ILP problem

Let us now discuss how the ax-aware ATPG problem can be represented as an ILP problem. The input vector subset $\mathcal{S}$, the two fault lists (i.e., non- and ax-redundant) and the fault coverage report are used for this goal. A conceptual model of the *fault coverage report*, mentioned in Subsection 4.3, is shown in Table 4.3. As shown, the

	v_0	v_1	v_2	....	v_{d-2}	v_{d-1}
f_{nr_0}	x			....	x	
f_{axr_0}		x		....		x
f_{nr_1}	x		x	....	x	
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$f_{axr_{l-1}}$	x		x	....		x
$f_{nr_{h-1}}$		x		....	x	

TABLE 4.3: Fault coverage (FC) report conceptual model

report contains all the correspondences between each fault and the input vectors covering it. Non-redundant faults are expressed as f_{nr_k} and ax-redundant faults as f_{axr_j}, for k $\in [0, h-1]$ and j $\in [0, l-1]$. Vectors are expressed as v_i, for i $\in [0, d-1]$. Each vector $v_i \in \mathcal{S}$ covers a number $v_i(nr)$ of non-redundant faults and a number $v_i(axr)$ of ax-redundant faults. Among those input vectors, we want to find the smallest subset $\mathcal{V}$ covering the smallest number of ax-redundant faults and the whole set of non-redundant ones. Therefore, to build the ILP problem, we firstly define the decision variables vector as follows:

$$\overline{x} = \{\overline{f_{axr}}, \overline{f_{nr}}, \overline{v}\} \tag{4.4}$$

where $\overline{f_{axr}}$, $\overline{f_{nr}}$ and $\overline{v}$ represent ax-redundant faults, non-redundant faults and input vectors, respectively. All the variables composing the $\overline{x}$ vector are binary. Each variable expresses whether the corresponding fault (or vector) is included or not in the final solution. Then, we need to express our goals as an objective function. We want to minimize two functions: the number of covered ax-redundant faults and the number of test vectors. The resulting *multi-objective* function is as follows:

$$min\left(\sum_{i=0}^{d-1} v_i\,, \sum_{j=0}^{l-1} f_{axr_j}\right), \tag{4.5}$$

where $v_i \in \{0,1\}\ \forall i \in [0, d-1]$ and $f_{axr_j} \in \{0,1\}\ \forall j \in [0, l-1]$ (i.e., binary variables). As it can be easily remarked, we deal with two competing objectives. Thus, a Pareto front of multiple optimal solutions is defined by this problem formulation. Therefore, we want to find an optimal solution laying on the Pareto front. In order to choose among all the optimal solutions, we use the weighted sum scalarization

method [97] to transform the multi-objective optimization problem into a single-objective one. This allows us to use single-objective resolution methods to resolve a multi-objective problem. The resulting function is as follows:

$$min\left(w_1\sum_{i=0}^{d-1}v_i + w_2\sum_{j=0}^{l-1}f_{axr_j}\right), \tag{4.6}$$

where the weights w_1 and w_2 define the importance of minimizing the two objective functions, respectively. Since we consider the two objectives equally important, we assign the same weight to both the functions ($w_1 = w_2 = 1$). Certainly, other weight value combinations can be used (see Subsection 4.3.3). Finally, the resulting objective function is as follows:

$$min\left(\sum_{i=0}^{d-1}v_i + \sum_{j=0}^{l-1}f_{axr_j}\right) \tag{4.7}$$

Concerning the vector $\overline{c}$ in Equation 4.3, it is composed of unitary coefficients corresponding to $\overline{v}$ and $\overline{f_{axr}}$ components and of zeros corresponding to $\overline{f_{nr}}$ components. Finally, we want to ensure that the final solution lies within the feasible region. Thus, by means of the fault coverage report, we set up some constraints as follows:

(i) the nR-FC has to be maximum:

$$\sum_{k=0}^{h-1}f_{nr_k} = T_{nrf} \tag{4.8}$$

where $f_{nr_k} \in \{0,1\}\ \forall k \in [0, h-1]$ and T_{nrf} is the total number of non-redundant faults;

(ii) the solution must contain at least one test vector:

$$\sum_{i=0}^{d-1}v_i \geq 1 \tag{4.9}$$

where $v_i \in \{0,1\}\ \forall i \in [0, d-1]$;

(iii) If a fault is involved in the solution, at least one vector covering it has to be in the solution, too:

$$\sum_{i=0}^{d-1}v_{if_{axr_j}} \geq f_{axr_j} \qquad \forall j \in [0, l-1] \tag{4.10}$$

$$\sum_{i=0}^{d-1}v_{if_{nr_k}} \geq f_{nr_k} \qquad \forall k \in [0, h-1] \tag{4.11}$$

where $v_{if_{axr_j}}$ and $v_{if_{nr_k}}$ are binary variables expressing whether the vector v_i covers or not the faults f_{axr_j} and f_{nr_k}, respectively. This information is obtained from FC report;

(iv) If a vector is involved in the solution, all the faults it covers have to be in the solution, too:

$$\sum_{j=0}^{l-1} f_{axr_j v_i} \geq v_i(axr) \cdot v_i \qquad \forall i \in [0, d-1] \tag{4.12}$$

$$\sum_{k=0}^{h-1} f_{nr_k v_i} \geq v_i(nr) \cdot v_i \qquad \forall i \in [0, d-1] \tag{4.13}$$

where $f_{axr_j v_i}$ and $f_{nr_k v_i}$ are binary variables expressing whether the faults f_{axr_j} and f_{nr_k} are covered by the vector v_i, respectively (obtained from FC report). $v_i(axr)$ and $v_i(nr)$ are the number of ax-redundant faults and of non-redundant faults covered by v_i, respectively. All the above described constraint equations contribute to form the $\overline{\overline{A}}$ matrix and the $\overline{b}$ vector of Equation 4.3. In the next subsection, we discuss the resolution method we used to solve the ILP problem.

ILP problem resolution As reported in many studies (such as [98], [99]), while LP problems are solvable in polynomial time, ILP problems are NP-Hard. In order to find a solution to the ax-aware ATPG ILP problem, we use a method for solving combinatorial optimization problems, the *Branch and Bound* (B&B). The method relaxes the integrality constraint of the variables and solves the resulting LP problem in polynomial time. Then, if the solution does not contain all integer variables, the method splits the problem into two disjoint sub-problems and repeats the computations ending up in a tree exploration. The procedure iterates until a feasible integer solution is found. Illustrating the (B&B) algorithm is out of the scope of this manuscript. Details on the algorithm can be found in [100]. If it exists, the B&B will find the ILP problem's optimal integer solution. This means that - among the generated input vectors (see phase I in sec 4.3) - B&B will find the smallest subset achieving the smallest possible axR-FC and the total nR-FC (T_{nrf} in Equation 4.8). Therefore, generating different input vector sets will lead to different solutions. To give a preliminary hint on the validity of the proposed approach, we applied it to our example (see Section 2.3). Next subsection details the results.

Proof of concept The inputs to the ax-aware ATPG are the AxIC and its non-redundant and ax-redundant fault lists. The latter are produced as output of the fault classification phase. Without loss of generality, in this example we use the MAE (Equation 1.4) as error metric to classify faults. Ax-aware ATPG's first phase is the input vector generation. Given the tiny size of this example, we use the exhaustive input vector set. Then, we perform the fault simulation and we obtain the fault coverage report summarized in Table 4.4. The input vector name convention is: vector '0' = 000, vector '1' = 001, ... , vector '7' = 111. Finally, the fault coverage

Fault	MAE	Classification[1]		v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
Sa0@a	1	ax-red.	f_{axr_0}			x	x			x	x
Sa1@a	1.25	non-red.	f_{nr_0}	x	x			x	x		
Sa0@b	1	ax-red.	f_{axr_1}		x		x		x		x
Sa1@b	1.25	non-red.	f_{nr_1}	x		x		x		x	
Sa0@c	1	ax-red.	f_{axr_2}		x	x			x	x	
Sa1@c	1.25	non-red.	f_{nr_2}	x			x	x			x
Sa0@d	1	ax-red.	f_{axr_3}					x	x	x	x
Sa1@d	1.25	non-red.	f_{nr_3}	x	x	x	x				
Sa0@e	1.5	non-red.	f_{nr_4}		x	x		x			x
Sa1@e	0.75	ax-red.	f_{axr_4}	x			x		x	x	

[1]Faults classified according to MAE. Threshold = 1 $v_0 = 000$, $v_1 = 001$, ..., $v_7 = 111$

TABLE 4.4: Fault coverage report, for the example circuit (see Figure 2.1). Faults are classified according to MAE metric (threshold=1)

report is used in the third phase: formulate the optimization problem. To correctly model the ILP problem, we set the parameters as follows: $d = 8$, $l = 5$, $h = 5$, and $T_{nrf} = 5$.

The objective function is expressed as follows:

$$min\left(\sum_{i=0}^{7} v_i + \sum_{j=0}^{4} f_{axr_j}\right) \tag{4.14}$$

where $v_i \in \{0,1\}\ \forall i \in [0,7]$ and $f_{axr_j} \in \{0,1\}\ \forall j \in [0,4]$. The ax-redundant (axR) faults (f_{axr_j}) are all listed in Table 4.4.

Concerning constraints, we define the following:

(i):

$$\sum_{k=0}^{4} f_{nr_k} = 5 \tag{4.15}$$

where $f_{nr_k} \in \{0,1\}\ \forall k \in [0,4]$; The non-redundant (nR) faults (f_{nr_k}) are all listed in Table 4.4.

(ii):

$$\sum_{i=0}^{7} v_i \geq 1 \tag{4.16}$$

where $v_i \in \{0,1\}\ \forall i \in [0,7]$;

(iii):

$$\sum_{i=0}^{7} v_{if_{axr_j}} \geq f_{axr_j} \qquad \forall j \in [0,4] \tag{4.17}$$

$$\sum_{i=0}^{7} v_{if_{nr_k}} \geq f_{nr_k} \qquad \forall k \in [0,4] \tag{4.18}$$

where – for each f_{nr_k} and f_{axr_j} – the values of $v_{if_{nr_k}}$ and $v_{if_{axr_j}}$ are 0 or 1 depending on whether v_i covers f_{nr_k} and f_{axr_j} or not. From the report in Table 4.4, we can deduce those values. Specifically, the 'x' symbol expresses that the vector v_i covers the correspondent fault.

(iv):

$$\sum_{j=0}^{4} f_{axr_j v_i} \geq v_i(axr) \cdot v_i \qquad \forall i \in [0,7] \tag{4.19}$$

$$\sum_{k=0}^{4} f_{nr_k v_i} \geq v_i(nr) \cdot v_i \qquad \forall i \in [0,7] \tag{4.20}$$

where – for each v_i – the values of $f_{axr_j v_i}$ and $f_{nr_k v_i}$ are 0 or 1 depending on whether f_{nr_k} and f_{axr_j} are covered by the vector v_i or not. $v_i(axr)$ and $v_i(nr)$ are the number of ax-redundant faults and of non-redundant faults covered by v_i. Again, from Table 4.4 we can deduce this information. For instance, for vector 2, from the table we can deduce $v_2(axr) = 2$, $v_2(nr) = 3$. The faults covered by vector 2 are Sa0@a, Sa1@b, Sa0@c, Sa1@d, and Sa0@e.

By solving the problem, we obtain the results in Table 4.5. As shown, the solution to the ILP problem led to a reduction of the covered ax-redundant faults from five (conventional ATPG Table 4.2) to two, while still covering all the non-redundant faults. Table 4.6 reports the comparison between conventional and ax-aware gener-

Non-redundant	Covered[1]	Ax-redundant	Covered[2]
Sa1@a	✓	Sa0@a	✗
Sa1@b	✓	Sa0@b	✗
Sa1@c	✓	Sa0@c	✗
Sa1@d	✓	Sa0@d	✓
Sa0@e	✓	Sa1@e	✓
$\mathcal{V}$ = {0,4}			

[1]✓ is desired; [2]✗ is desired

TABLE 4.5: ILP problem solution

ated test vectors, in terms of covered axR faults. The ax-aware ATPG relative improvement over the conventional one is calculated as the difference between the covered axR faults divided by the conventional ATPG covered axR faults. Moreover, the obtained solution is an optimal one. Indeed, as discussed in Subsection 4.1, the vector subset $\mathcal{V} = \{0, 4\}$ is optimal, thus there is no other subset leading to a lower axR-FC. The B&B resolving method finds the optimal solution among the analyzed

	Conventional ATPG			Ax-aware ATPG	
Test Vector Number	0	1	7	0	4
Test Vector Value	000	001	111	000	100
Covered AxR faults		5		2	
Relative Improvements	$\frac{5-2}{5} \cdot 100 = 60\%$				

TABLE 4.6: Ax-unaware and ax-aware generated test vectors comparison

vectors. Since we have used the exhaustive input vector set, for this example the B&B finds an absolute optimal solution.

The illustrated example shows two important results: on the one hand, it is possible to drastically reduce the axR-FC; on the other hand, some ax-redundant faults cannot remain undetected. This issue is discussed and addressed in Chapter 5. In the next section, we report experimental results obtained by applying the proposed methodology to a set of state-of-the-art AxICs. Moreover, we used three different input vector generation algorithms and compared their performance.

4.3.3 Experimental results

In this paragraph, we report results presented in [96] for the proposed ax-aware ATPG technique, which aims at mitigating the over-testing effects. We performed experiments on some state-of-the-art AxICs. Specifically, we analyzed Accuracy-Configurable Approximate (ACA) adders from [11], Gracefully-Degrading Adders (GDA) from [22], Generic Accuracy configurable (GeAr) adders from [23], Error Tolerant Adders (ETAII) from [70], and some EvoApprox8b library AxICs [25] (add8_051, add8_036, add8_012, add8_045).

Firstly – without loss of generality – we performed the fault classification by using the technique proposed in [88] and we used the WCE (Equation 1.3) as error metric. Thus, for each AxIC, we obtained ax-redundant and non-redundant fault lists.

Secondly, we used the conventional ATPG to produce the test set and we measured the nR FC and axR FC. The goal was to compare ax-aware ATPG and conventional ATPG results to evaluate the improvements. To determine both nR-FC and axR-FC, we fault simulated the two fault lists with the test set obtained with the conventional ATPG. As expected, results showed that 100% of non-redundant faults were detected, for all the AxICs. Concerning axR-FC, in Table 4.7, we report the results (along with the AxICs attributes in terms of total fault number F_{tot}, non-redundant fault number F_{nR}, and ax-redundant fault number F_{axR}). As it can be

Circuit	F_{tot}	F_{nR}	F_{axR}	conventional ATPG		
				axR-FC[1]		Vectors[1]
add8_051	156	152	4	4	(100.00%)	10
add8_036	126	111	15	12	(80.00%)	10
add8_012	99	93	6	6	(100.00%)	10
add8_045	72	71	1	1	(100.00%)	9
GeAr_N8_R2_P2	154	75	75	55	(73.33%)	5
ACA_I_N8_Q5	216	57	113	77	(68.14%)	5
GDA_St_N8_M8_P3	202	73	123	100	(81.30%)	6
GeAr_N16_R6_P4	159	57	102	89	(87.25%)	5
ACA_II_N16_Q8	188	68	120	83	(69.17%)	4
ETAII_N16_Q8	237	71	166	118	(71.08%)	6
GDA_St_N16_M4_P4	366	69	297	267	(89.90%)	10
GDA_St_N16_M4_P8	375	54	321	246	(76.64%)	6
GeAr_N16_R4_P4	188	68	120	83	(69.17%)	4
GeAr_N16_R4_P8	199	49	150	105	(70.00%)	6
GeAr_N16_R2_P4	275	43	232	152	(65.52%)	6
ACA_II_N16_Q4	356	33	323	245	(75.85%)	5
ETAII_N16_Q4	356	33	323	245	(75.85%)	5
ACA_I_N16_Q4	454	52	402	302	(75.12%)	7
Average				**79.35%**		

[1]Lower is better

TABLE 4.7: AxICs attributes and conventional ATPG ineffectiveness evidences

seen, the conventional ATPG produces test vectors covering on average 79% of ax-redundant faults. In the best case, conventional ATPG covered 65% of ax-redundant faults. For some AxICs, 100% of the ax-redundant faults were detected, ultimately leading to totally undermine the fault classification effort.

Finally, we generated the test set with the proposed technique and analyzed the improvements w.r.t. conventional ATPG. In the next two sections we discuss the experimental setup and the results.

Setup We performed experiments by formulating the ILP problem objective function in two ways: (i) as described in Equation 4.7 – to which we refer as multi-objective (MO) – and (ii) by considering only the single-objective (SO) function minimizing the covered ax-redundant faults, as follows:

$$min \left(\sum_{j=0}^{l-1} f_{axr_j} \right). \tag{4.21}$$

The latter corresponds to setting $w_1 = 0$ and $w_2 = 1$ in Equation 4.6. We used this objective function to verify if the choice of not minimizing the test set lenght has an impact on axR-FC reduction. We express the results as the improvement percentage compared to the conventional ATPG's results, which are reported in the previous subsection (Table 4.7). We used the following equations to calculate improvements

in terms of axR-FC and produced test vectors:

$$\text{axR FC improv.} = \frac{\text{conventional ATPG axR-FC} - \text{ax-aware ATPG axR-FC}}{\text{conventional ATPG axR-FC}} \cdot 100 \quad (4.22)$$

$$\text{vectors improv.} = \frac{\text{conventional ATPG vectors} - \text{ax-aware ATPG vectors}}{\text{conventional ATPG vectors}} \cdot 100 \quad (4.23)$$

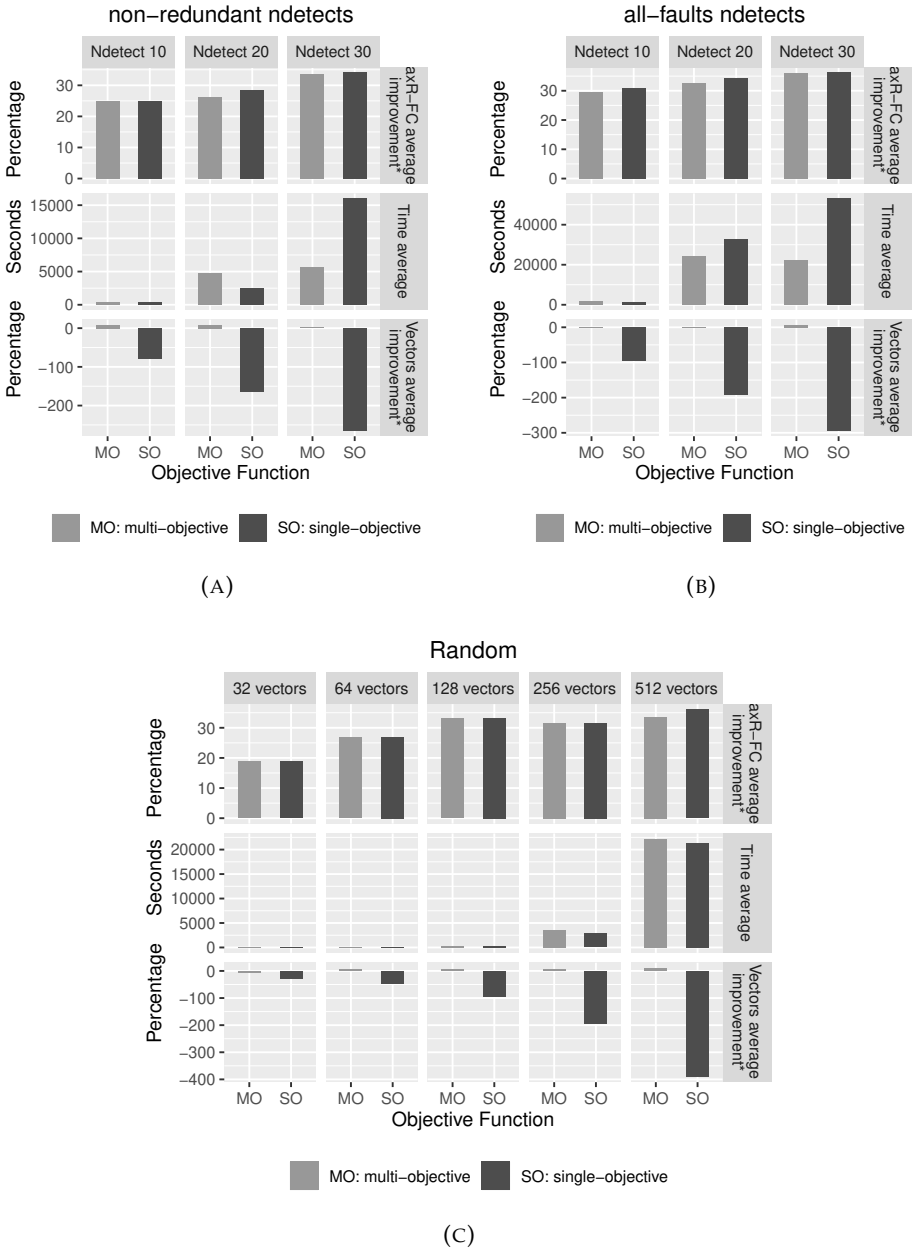
Furthermore, we used three different input vector generation methods in the experiments. Indeed, as discussed in Subsection 4.3, different input vector generation methods (phase I of the ax-ATPG technique) can give different results. The goal was to produce input vector subsets of different dimensions and by using different techniques, in order to compare results. Intuitively, the larger the input vector subsets (i.e., the ILP problem search space), the better the results.

1. The first method we used to generate input vector subsets was the conventional ATPG with the option for generating input vectors covering each fault up to n times (namely *"ndetects n"*) and by targeting only non-redundant faults. We refer to this input vector generation method as *"non-redundant ndetects"*. Note that, despite the misleading method's name, we used the conventional ATPG only to generate input vector subsets (i.e., phase I in Subsection 4.3). We performed experiments for different values of n, namely 10, 20, and 30.

2. Then, we performed experiments by using another input vector generation method. We employed again the conventional ATPG with the "ndetects n" option and we targeted the whole AxIC fault list (both ax-redundant and non-redundant faults). We refer to this vector generation method as *"all-faults ndetects"*. Experiments were carried out for different values of n (10, 20, and 30).

3. Finally, we carried out experiments by using random input vector sets of different dimensions (32, 64, 128, 256, and 512) generated by using a software library for pseudo-random sequences. In this case, in order to achieve the required nR-FC, we slightly modified the input vector generation phase: after generating a random sequence, we fault-simulated it. If the nR-FC did not satisfy the requirement, we generated a new random sequence. The process was re-iterated until the random sequence satisfied the required coverage. This was necessary especially with tiny random set (i.e., 32).

Result discussion In Figure 4.2, we report the experimental results. Figure 4.2a reports the improvements achieved with the *non-redundant ndetect* input vector generation method. Improvements achieved with the *all-faults ndetect* input vector generation method are shown in Figure 4.2b. Finally, Figure 4.2c reports achieved improvements for *random ndetect* input vector generation method. Note that nR-FC is



(A) (B)

(C)

*proposed technique improvements w.r.t. conventional ATPG (see Table 4.7) — higher is better

FIGURE 4.2: Average results for *"non-redundant ndetects"* (a), *"all-faults ndetects"* (b), and *"random"* (c) vector generation methods

not mentioned in the figures since, for all the experiments, it was the same as the conventional ATPG (i.e., 100%). Results confirmed that an important improvement is possible over the conventional ATPG. On average, the proposed ax-aware ATPG - for both MO and SO functions - led to an improvement spanning from 19% up to 36% of axR-FC reduction, compared to conventional ATPG. In some cases, we were able to totally avoid covering ax-redundant faults, thus achieving 100% improvement. In general, for circuits with low error values it was possible to achieve a

more important improvement. For few cases, it was not possible to find a better test vector subset than the one produced by conventional ATPG. This phenomenon was more likely to happen when we used tiny input vector subset (e.g., "random 32"). In those cases, the resulting ILP search space was not big enough to allow the solving algorithm to find a suitable solution. Nevertheless, for the "all-faults ndetects" experiment campaigns, we did not experience this phenomenon. The "all-faults ndetects", indeed, turned out to be the input vector generation method who gave better results. On average, results obtained by using this input generation method spanned from 29% up to 36% axR-FC reduction improvement and, as mentioned above, for all experiments we achieved an actual axR-FC reduction. Moreover, by using the *all-fault ndetect 10* method, we obtained a larger axR-FC improvement (30%) than by using the *non-redundant ndetect 20* method (25%), in a shorter time (~1800 s VS ~4800 s). Finally, no substantial differences were observed between MO and SO functions. Indeed, in both cases, a very similar average axR-FC improvement was achieved in a fairly comparable amount of time.

As far as it concerns test set dimension, on average the ax-aware ATPG did not have a significant impact on test set length, when resorting to MO function. Indeed, the test set dimension improvement w.r.t. conventional ATPG spanned from -5% and +9%, on average. Conversely, when resorting to SO function, the test set was always larger compared to conventional ATPG, on average. Indeed, average test set dimension improvement in this case spanned from -27% to -390% ($\approx$ 4 times larger).

Therefore, the ax-aware ATPG with the MO function turned out to be preferable to obtain a significant axR-FC improvement over the conventional ATPG while fairly not impacting the test length.

4.4 Evaluation

In this section we compare the *conventional* test pattern generation with the proposed solutions. Hereafter, as *conventional generation* we will refer to techniques that use the conventional ATPG to generate test sets (see Figure 4.3): the non-redundant fault list is generated by using a fault classification technique and then used as target for the ATPG. [86] and [95] use this kind of pattern generation. Then, as *ax-aware generation* we refer to techniques that simultaneously individuate non-redundant faults and generate test vectors covering them. This technique is used in [32], [88], [94]. Figure 4.4 sketches the ax-aware generation technique flow. Finally, we refer to the technique presented in [96] as *pattern selection*. We recall that, for the pattern selection technique, we need to generate a set of input vectors from which we choose

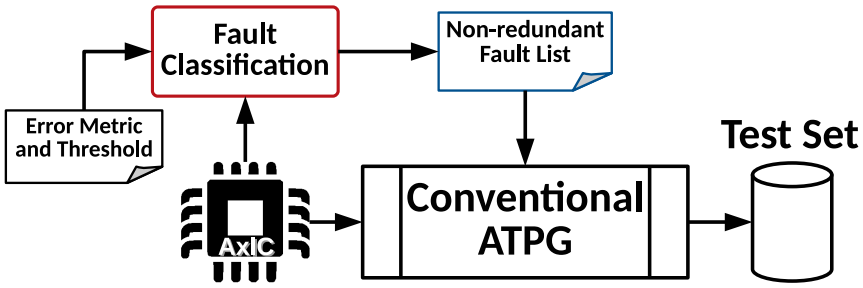


FIGURE 4.3: Conventional test pattern generation schema

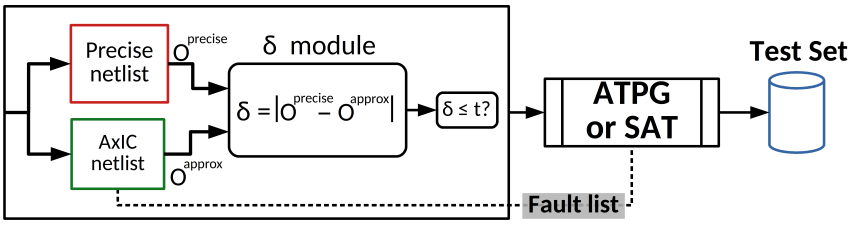


FIGURE 4.4: Ax-aware test pattern generation schema

the best subset by building and solving a combinational problem. For those experiments we used as input vector generation method a *mixed approach*, based on the insights in [88] and [96]. Specifically, we used the test set generated by the ax-aware technique and enriched it as follows: we exploited the *"non-redundant ndetects"* method [96] (see Subsection 4.3.3) in the context of the schema in Figure 4.4, presented in [88]. More intuitively, the goal was to generate a lot of *ax-aware input vectors* and then select the set detecting as few ax-redundant faults as possible.

To fairly compare the quality of the test sets produced by the different approaches, we performed experiments on AxICs from the EvoApprox8b library [25], introduced in Section 3.2. We considered the WCE as error metric. Firstly, we performed the fault classification. Then, we obtained test patterns by using the three approaches, i.e. conventional, ax-aware, and pattern selection generation. Then, we performed a fault-simulation by using the generated patterns and the two fault lists (i.e., non-redundant and ax-redundant faults) in order to measure nR and axR FCs (see Equations 2.2 and 2.3).

The achieved nR FC was always 100%, which confirms that all the techniques achieve the first objective of AxA testing (see Chapter 2). Then, in Table 4.8, we report results in terms of axR FC for the four AxIC groups (8-bit adders and 8-, 16-, and 32-bit multipliers), as well as in terms of Yield Increase Loss (YIL).

As shown, conventional generation technique exhibits an average axR FC between 65% and 92%, with peaks at 100%, corresponding to a YIL between 14% and 50%. Significant lower (thus better) axR FC and YIL values were achieved by using ax-aware and pattern selection techniques. Ax-aware generation technique showed

Conventional generation[1]

	Add8		Mul8		Mul16		Mul32	
	axR FC[2]	YIL[2]	axR FC[2]	YIL[2]	axR FC[2]	YIL[2]	axR FC[2]	YIL[2]
Min	0.00%	0.00%	73.55%	5.20%	64.00%	16.53%	72.06%	33.00%
Max	100.00%	77.45%	99.37%	73.92%	100.00%	50.00%	97.73%	75.79%
Avg	**65.81%**	**14.04%**	**91.43%**	**43.20%**	**92.08%**	**31.63%**	**90.94%**	**50.55%**
Time[3]	**0.61**		**0.87**		**0.91**		**2.3**	

Ax-aware generation[1]

	Add8		Mul8		Mul16		Mul32	
	axR FC[2]	YIL[2]	axR FC[2]	YIL[2]	axR FC[2]	YIL[2]	axR FC[2]	YIL[2]
Min	0.00%	0.00%	17.81%	1.02%	51.90%	16.62%	11.03%	6.47%
Max	100.00%	71.57%	95.43%	74.18%	96.88%	40.00%	85.86%	39.87%
Avg	**43.17%**	**9.66%**	**83.03%**	**39.62%**	**77.11%**	**26.02%**	**47.61%**	**24.98%**
Time[3]	**0.64**		**0.91**		**0.96**		**2.60**	

Pattern selection generation[1]

	Add8		Mul8		Mul16		Mul32	
	axR FC[2]	YIL[2]	axR FC[2]	YIL[2]	axR FC[2]	YIL[2]	axR FC[2]	YIL[2]
Min	0.00%	0.00%	17.81%	1.02%	48.00%	14.33%	11.03%	6.47%
Max	84.85%	54.95%	91.16%	69.51%	91.98%	36.36%	85.65%	39.87%
Avg	**33.49%**	**7.61%**	**76.48%**	**36.39%**	**72.97%**	**24.78%**	**47.46%**	**24.90%**
Time[3]	**5**		**2275**		**21230**		**2343**	

[1]100% non-redundant FC always achieved [2]Lower is better [3]Average time in seconds

axR FC = Ax-redundant FC YIL = Yield Increase Loss

TABLE 4.8: Ax-redundant FC (axR FC) and Yield Increase Loss (YIL) results. YIL and axR FC indicate the absolute and the relative loss of yield increase, respectively (see Section 2.1).

average axR FC between 43% and 83%, corresponding to a YIL between 9% and 39% (lower is better). Pattern selection generation further improved results, by obtaining axR FC between 33% and 76%, corresponding to a YIL between 7% and 36%. Concerning execution time, we can easily see that the proposed pattern selection generation technique entails a much longer time. This is due to the intrinsic complexity of the ILP problem.

By using Equation 4.22, we can calculate the improvement of the two techniques (higher is better) compared to conventional technique (see Table 4.9). The average obtained axR FC improvement for ax-aware generation technique was between 9% (for Mul8) and 47% (for Mul32), corresponding to a YIL improvement between 8% and 50%. For pattern selection generation, the improvement compared to conventional technique was between 16% and 49% for axR FC and between 15% and 50% for YIL.

Even though obtained results are quite good, they are still far to be ideal. Indeed, while generating ax-aware test patterns improve the test quality, it turns out that some ax-redundant faults are still detected. Ultimately, this leads to a yield

Ax-aware generation								
	Add8		Mul8		Mul16		Mul32	
	axR FC	YIL	axR FC	YIL	axR FC	YIL	axR FC	YIL
Improv.[1]	34.40%	31.20%	9.19%	8.29%	16.26%	17.74%	47.65%	50.58%

Pattern selection generation								
	Add8		Mul8		Mul16		Mul32	
	axR FC	YIL	axR FC	YIL	axR FC	YIL	axR FC	YIL
Improv.[1]	49.11%	45.80%	16.35%	15.76%	20.75%	21.66%	47.81%	50.74%

[1]Higher is better

TABLE 4.9: Improvements obtained by using ax-aware generation and pattern selection generation techniques, compared to conventional generation technique. Higher is better.

increase lower than expected (YIL > 0%). This is due to the intrinsic structure of the AxICs, as discussed at the beginning of the Chapter. Nevertheless, we have to consider that the reported results were obtained by using *conventional test set application* techniques, i.e. no effort was made to distinguish between ax-redundant and non-redundant faults, in test application phase. Therefore, by introducing proper *AxA test set application* techniques, the test quality can be augmented. Next chapter addresses such aspects.

4.5 Chapter summary

In this chapter we presented the AxA test pattern generation. We showed and discussed the issues related to the test pattern generation in the context of AxICs. In particular, test patterns have to cover all non-redundant faults and as few as possible ax-redundant ones. The technique presented in Subsection 3.2.1, as well as other techniques in the literature [32], [94], successfully deal with the generation of test patterns detecting all the non-redundant faults.

Nevertheless, these techniques are limited by some specific conditions (i.e., the considered metric used in the fault classification is an SCT metric). Moreover, we noticed that different test sets achieving 100% non-Redundant Fault Coverage (nR FC) achieve different values of ax-Redundant Fault Coverage (axR FC). Existing techniques do not address the problem of finding the best test set, i.e. minimizing the axR FC. Therefore, we proposed a more general technique – based on a careful *pattern selection* – designed specifically to address AxA test pattern generation.

Finally, we compared the outcomes of the different test pattern generation techniques, i.e. the conventional generation (i.e., ATPG considering only non-redundant faults [86], [95]), the ax-aware generation (i.e., test patterns generated at the same time of the classification [32], [88], [94]) and the ax-aware generation with pattern

selection [96]. Results showed an average axR FC improvement spanning from 9% to 47% for the ax-aware generation compared to conventional one. Moreover, the ax-aware generation with pattern selection technique pushed the limits even further, by delivering from 16% to 49% axR FC improvements compared to conventional generation.

Although results were quite good, they are still quite far from the ideal ones. Therefore, in next chapter we introduce the *AxA test set application* to further improve test quality.

Chapter 5

AxA test set application

Contents

As shown in the previous chapter, test pattern generation techniques can generate qualitatively different test sets. To improve the final test process quality, *Test set application* plays an important role. In this phase, we need techniques able – by observing circuit's responses – to distinguish between the detection of an ax-redundant fault (i.e., the test passes) and a non-redundant one (i.e., the AxIC is rejected). In Section 5.1 we show and discuss the issue. In literature, no techniques have been presented to deal with such aspect so far. Nevertheless, a technique originally presented for conventional ICs [32], the *threshold testing* (see Subsection 2.2), can be adapted to AxICs. In Section 5.2, we investigate the threshold testing suitability in the AxIC testing context. Afterwards, in Section 5.3, we propose a new AxA test set application technique and evaluate it.

5.1 Problem statement

As mentioned in the previous section, the proper structure of an AxIC usually makes impossible for a test set to avoid the detection of some ax-redundant faults [96].

To show the issue, we resort to our example introduced in Section 2.3, i.e., the approximate full adder.

Input $\mathcal{I}$				$^{*}O^{precise}$	Fault-free $^{\dagger}O^{approx}$	Faulty $^{\dagger}O^{approx}$									
i	C_i	X	Y			Sa0@a	Sa1@a	Sa0@b	Sa1@b	Sa0@c	Sa1@c	Sa0@d	Sa1@d	Sa0@e	Sa1@e
0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1
1	0	0	1	1	1	1	0	0	1	0	1	1	0	0	1
2	0	1	0	1	1	0	1	1	0	0	1	1	0	0	1
3	0	1	1	2	0	1	0	1	0	0	1	0	1	0	1
4	1	0	0	1	1	1	0	1	0	1	0	0	1	0	1
5	1	0	1	2	0	0	1	1	0	1	0	1	0	0	1
6	1	1	0	2	0	1	0	0	1	1	0	1	0	0	1
7	1	1	1	3	1	0	1	0	1	1	0	0	1	0	1
Fault classification (MAE‡):						ax-red.	non-red.	ax-red.	non-red.	ax-red.	non-red.	ax-red.	non-red.	non-red.	ax-red.

*Precise output;
†Approximate output;
‡Mean Average Error (MAE) = 1

Value: vector i detects the fault → *Value* is different from fault-free O_i^{approx}
Value: approximate circuit output. → *Value* is different from $O_i^{precise}$.

TABLE 5.1: Output (in integer format) of the example circuit (see Figure 2.1) for different cases: precise (Fig 2.1a), fault-free approximate (see Fig 2.1b), and faulty approximate with different Stuck-at faults.

In the left part of Table 5.1, we report the outputs of the precise IC ($O^{precise}$) and of the fault-free AxIC (O^{approx}), for each input vector $i \in [0,7]$. Output values are reported as integer (e.g., 00 = "0", 01 = "1", etc.). To measure the error, we used the Mean Average Error (MAE) metric (Equation 1.4). The MAE in the example is 1. This is the threshold value, which must not be altered by the presence of defects introduced during the manufacturing phase.

In order to illustrate the problem, we report in the right part of Table 5.1 the impact of each stuck-at fault on the AxIC output. As already shown in Chapter 3,

based on the difference between the obtained faulty outputs (faulty O_i^{approx}) and the precise output ($O_i^{precise}$), faults are classified. If the MAE is greater than the threshold ($\frac{\sum_{\forall i \in \mathcal{I}} \left| O_i^{\text{approx}} - O_i^{\text{precise}} \right|}{2^n} > 1$, in the example), the fault is non-redundant. Otherwise, if the MAE is lower that or equal to the threshold ($\frac{\sum_{\forall i \in \mathcal{I}} \left| O_i^{\text{approx}} - O_i^{\text{precise}} \right|}{2^n} \leq 1$, in the example), the fault is ax-redundant. We report the class of each fault in the last row of the table.

Secondly, in the table we report in red solid-bordered boxes the *faulty* O_i^{approx} values that differ from the *fault-free* O_i^{approx} ones. Thanks to this output difference, in test application phase we can detect whether a fault affected the AxIC or not. While in conventional test each difference between actual and expected outputs leads to reject the circuit, when it comes to AxICs we have to reconsider this mechanism. Indeed, a test vector intended to detect a non-redundant fault can also detect an ax-redundant one, ultimately rejecting a still-acceptable circuit. For example, in Table 5.1, we can remark that the vector 4 detects four non-redundant faults (Sa1@a, Sa1@b, Sa1@c, Sa0@e), but also one ax-redundant fault (Sa0@d). As reported in last chapter, in [96] we proposed a technique to generate test patterns which detects all the non-redundant faults but also minimize the number of detected ax-redundant faults (see Section 4.3). Unfortunately, it is often impossible to avoid the detection of some ax-redundant faults. For instance, we can easily note that – among all the possible test sets – the best is the couple {0, 4}. The two vectors detect 100% of the non-redundant faults (i.e., five faults). Nevertheless, they detect also 40% of ax-redundant faults (two out of five). Specifically, Sa1@e is detected by vector 0 and Sa0@d by vector 4. Therefore, while the *expected yield gain* is of five faults out of ten (i.e., the five ax-redundant faults), by using the classic test application, we still detect two ax-redundant faults. In other words, from 50% expected yield gain (five ax-redundant faults avoided, out of ten total faults) we drop to 30% (three ax-redundant faults avoided, out of ten total faults).

To avoid this over-testing phenomenon, we need to reconsider the test application phase. In details, after the application of the test patterns to the AxIC under test, we need to verify that the actual output meets some conditions and not only whether it differs from the expected output.

5.2 A state-of-the-art solution

As discussed in Subsection 2.2, the approach proposed in [32] (i.e., the *threshold testing*) can be considered as a special case of AxA testing [26]. In threshold testing, after test vectors for the *intolerable faults* (i.e., non-redundant) are created, authors

go through a slightly modified test set application phase, by adding a further verification, as follows. Once a test vector is applied to the IC, the test responses are compared with the golden ones (those produced by a fault-free IC). If the difference is lower than a given threshold, the circuit is considered still acceptable.

We apply the threshold testing to AxICs as follows. For each test vector applied to the AxIC, test responses are compared with the golden ones (i.e., those produced by the non-approximated circuit). If the arithmetic difference (i.e., the Error Magnitude (EM)) is not greater than the threshold, the test passes. Otherwise, the circuit is rejected.

5.2.1 Suitability investigation

In order to preliminary study the technique suitability for AxICs we apply it to our FA example (Figure 2.1). We consider the WCE and the MAE as error metrics. In Section 4.1, we obtained different test sets for our FA example, by using conventional and ax-aware generation techniques. We report them in Table 5.2. Now, we use them to preliminary evaluate the threshold testing technique. Therefore, we

	WCE		MAE	
	Ax-aware	Conventional	Ax-aware	Conventional
Test set*	{7}	{1, 6, 0}	{0, 4}	{1, 7, 0}

*0="000", 1="001", ... , 7="111"

TABLE 5.2: Test set generated using an ideal ax-aware test vector generation and a conventional ATPG tool [87] on the example circuit in Figure 2.1. (see Section 4.1)

simulate each vector for all the test sets in presence of each fault (both ax-redundant and non-redundant) and we obtain the EM of the circuit's output. We report results in Table 5.3. Then, if the Error Magnitude (EM) is greater than the threshold t, the circuit is considered faulty and rejected, otherwise the test passes.
Two key observations emerge from the results:

- As shown at the top of the table, the technique worked **only for the WCE metric**;
- the technique worked **only when using ax-aware vectors**.

Indeed, all the non-redundant faults classified with WCE metric ($t = 2$) and tested by ax-aware test vectors gave an EM value higher than the threshold (> 2). All the ax-redundant faults gave an EM value under the threshold (≤ 2). Conventional ATPG vectors were not able to attain the same result. Moreover, as shown in the bottom of the table, for MAE metric ($t = 1$) the technique did not work at all. In fact,

		Non-redundant fault list (result must be > threshold)					Ax-redundant fault list (result must be ≤ threshold)				
Metric threshold	**WCE = 2**	Sa0@a	Sa0@b	Sa1@c	Sa0@d	Sa0@e	Sa1@a	Sa1@b	Sa0@c	Sa1@d	Sa1@e
Ax-aware vectors	vector 7 EM:	3 ✓	3 ✓	3 ✓	3 ✓	3 ✓	2 ✓	2 ✓	2 ✓	2 ✓	2 ✓
Conventional ATPG vectors	vector 1 EM:	0 ✗	1 ✗	0 ✗	0 ✗	1 ✗	1 ✓	0 ✓	1 ✓	1 ✓	0 ✓
	vector 6 EM:	1 ✗	2 ✗	2 ✗	1 ✗	2 ✗	2 ✓	1 ✓	1 ✓	2 ✓	1 ✓
	vector 0 EM:	0 ✗	0 ✗	1 ✗	0 ✗	0 ✗	1 ✓	1 ✓	0 ✓	1 ✓	1 ✓
		✓$EM > 2$		✗$EM \leq 2$			✓$EM \leq 2$		✗$EM > 2$		

Metric threshold	**MAE = 1**	Sa1@a	Sa1@b	Sa1@c	Sa1@d	Sa0@e	Sa0@a	Sa0@b	Sa0@c	Sa0@d	Sa1@e
Ax-aware vectors	vector 0 EM:	1 ✗	1 ✗	1 ✗	1 ✗	0 ✗	0 ✓	0 ✓	0 ✓	0 ✓	1 ✓
	vector 4 EM:	2 ✓	2 ✓	1 ✗	1 ✗	2 ✓	1 ✓	1 ✓	2 ✗	2 ✗	1 ✓
Conventional ATPG vectors	vector 1 EM:	1 ✗	0 ✗	0 ✗	1 ✗	1 ✗	0 ✓	1 ✓	1 ✓	0 ✓	0 ✓
	vector 7 EM:	2 ✓	2 ✓	3 ✓	2 ✓	3 ✓	3 ✗	3 ✗	2 ✗	3 ✗	2 ✗
	vector 0 EM:	1 ✗	1 ✗	1 ✗	1 ✗	0 ✗	0 ✓	0 ✓	0 ✓	0 ✓	1 ✓
		✓$EM > 1$		✗$EM \leq 1$			✓$EM \leq 1$		✗$EM > 1$		

✓= good decision ✗ = bad decision

TABLE 5.3: Example of test set application technique by [32] used on the FA example (Figure 2.1).

some non-redundant faults were masked and some ax-redundant were detected by using both ax-aware and conventional vectors. In conclusion, to properly apply the threshold testing technique [32] to AxICs, three constraints need to be satisfied:

1. Golden circuit test responses must be known.
2. The considered metric must be an SCT metric (e.g., WCE, WCBFE).
3. AxIC test vectors must be produced with an ax-aware generation technique.

5.2.2 Experimental results

To corroborate the statement, we applied the technique to AxICs from the EvoApprox8b library [25]. We considered the WCE as error metric. We used test sets obtained by using both conventional and ax-aware generation techniques, as shown in Section 4.1. In Tables 5.4 and 5.5, we report experimental results. By comparing

		Conventional pattern generation				Ax-aware pattern generation[2]			
		Add8	Mul8	Mul16	Mul32	Add8	Mul8	Mul16	Mul32
nR FC[1]	Min	27.78%	62.98%	68.63%	77.18%	100%	100%	100%	100%
	Max	100%	100%	100%	100%	100%	100%	100%	100%
	Avg	**97.23%**	**91.60%**	**93.60%**	**94.60%**	**100%**	**100%**	**100%**	**100%**

[1]Should be always 100% [2]with and without pattern selection

TABLE 5.4: Non-redundant FC results when using the test set application technique by [32].

results with those in Table 4.8, we can clearly notice that the technique gave optimal results (i.e., 100% nR FC and 0% axR FC) when ax-aware test patterns were used. Conversely, not very good outcomes were achieved when using conventional test patterns. Indeed, although axR FC (Table 5.5) gave better results w.r.t. Table 4.8, the

Conventional pattern generation

	Add8		Mul8		Mul16		Mul32	
	axR FC[1]	YIL[1]	axR FC[1]	YIL[1]	axR FC[1]	YIL[1]	axR FC[1]	YIL[1]
Min	0.00%	0.00%	0.84%	0.50%	2.38%	0.57%	2.92%	1.49%
Max	100.00%	10.71%	70.00%	15.29%	45.24%	17.92%	41.41%	18.28%
Avg	**26.85%**	**3.28%**	**14.69%**	**5.47%**	**15.31%**	**5.40%**	**17.85%**	**8.18%**

Ax-aware pattern generation (with and without pattern selection)

	Add8		Mul8		Mul16		Mul32	
	axR FC[1]	YIL[1]	axR FC[1]	YIL[1]	axR FC[1]	YIL[1]	axR FC[1]	YIL[1]
Min	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Max	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Avg	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**

[1]Lower is better

TABLE 5.5: Ax-redundant FC (axR FC) and Yield Increase Loss (YIL) results when using the test set application technique by [32].

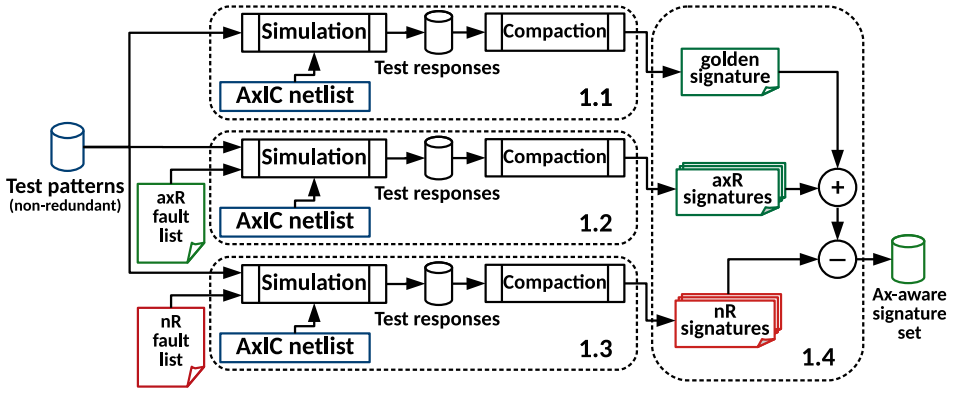
nR FC (Table 5.4) did not always reach 100%. This leads to undermine the first key aspect of the AxA testing, i.e. detecting all the non-redundant faults, which cause catastrophic errors.

In conclusion, the test set application technique from [32] guarantees a high-quality test outcomes for AxICs only under certain conditions.

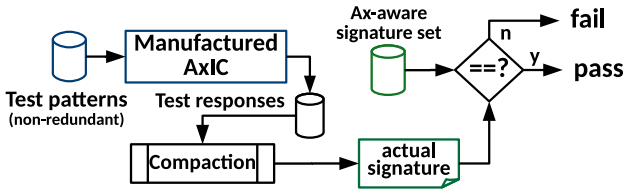
5.3 A new AxA test set application technique

The limitations of threshold testing technique [32] led us to propose a new *approximation-aware test application* technique [101] to mitigate the over-testing effect. We drew our inspiration from a concept introduced in late seventies, the *signature analysis* [33], which – as discussed in Chapter 1 – is mostly used in self-testing hardware techniques. In particular, *Built-In Self-Test (BIST)* approach compacts test responses together into a signature, which is used to verify whether the Unit Under Test (UUT) is faulty or not. In detail, when the test mode is activated, test patterns are applied to UUT and a signature is generated. Then, the latter is compared with the golden signature, which was generated by the fault-free circuit and stored within the BIST architecture. If the two signatures are identical, the circuit is considered fault-free. Otherwise, a malfunction is detected. Different compaction methods can be used to produce the signature. An extensive review of those methods can be found in [34].

Basically, we propose to generate multiple signatures, one for each ax-redundant fault, and compare them with test responses. If there is at least one match, then the AxIC is considered acceptable. Otherwise, the circuit is rejected. The proposed technique is intended to be used for external test (i.e., test are applied by using an Automatic Test Equipment (ATE)). Of course, it can be also used in a BIST context.



(A) phase 1, design time



(B) phase 2, test time (post-manufacturing)

FIGURE 5.1: Proposed test application technique

5.3.1 Proposed technique

The proposed technique is independent of the specific metric considered during the fault classification process, of the precise circuit test responses and of the specific test pattern generation technique. We assume as preconditions to have:

- ax-redundant and non-redundant fault lists;
- the test patterns detecting non-redundant faults.

As depicted in Figure 5.1, the proposed test application technique is composed of two phases, described below:

At design time we simulate test patterns with the AxIC netlist and compact the responses together to form a *golden signature* (1.1). Then, we perform the same procedure while injecting, one by one, all the ax-redundant (axR) faults into the AxIC netlist. This results in *ax-redundant signatures* (1.2). Hence, we apply the same process to non-redundant (nR) faults, in order to obtain *non-redundant signatures* (1.3). Finally, we perform the union between golden and ax-redundant signatures, hence we remove signatures in common with non-redundant ones (if any) (1.4). We usually refer as *aliasing* to the phenomenon for which some *bad* signatures overlap *good* ones [34] (see Subsection 1.1.4). The output of this phase is what we call *ax-aware signature set*.

At test time (post-manufacturing) after applying test patterns to the manufactured AxIC, we compact test responses and compare the *actual signature* with all the signatures in the ax-aware signature set. If at least one of the comparisons matches, than the test passes, otherwise the circuit is rejected.

It can be easily deduced that the proposed technique is independent of the specific fault classification and pattern generation techniques employed. Indeed, it is based only on the analysis of the AxIC's test responses.

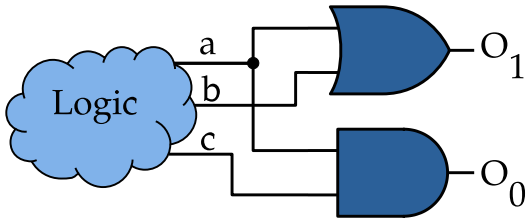
As mentioned in the beginning of the section, different response compaction methods can be used. Moreover, the proposed technique can be used for both external testing and self-testing. Concerning external testing, the *Automatic Test Equipment (ATE)* software can be modified to implement any compaction (e.g., hashing algorithm such as MD5, SHA, etc.). On the other hand, concerning self-testing hardware approaches as the BIST, other techniques exist, such as *one-count*, *transition count*, *Linear Feedback Shift Register (LFSR)*, etc [34].

5.3.2 Signature aliasing problem

As previously mentioned, the overlapping phenomenon of two signatures is usually referred to as *aliasing*. In details, as reported in Chapter 1, during the test response compaction, a signature of a faulty circuit can match the fault-free circuit one. This is due to the loss of information caused by the compaction itself [34].

We extend the meaning of the aliasing in the context of AxIC testing. Let us resort to a tiny example to show the issue. In Figure 5.2a, we depict a hypothetical circuit where some logic produces three signals (a,b,c) which drive the circuit outputs (O_1O_0) through two logic gates. Figure 5.2b reports the truth table of the two output signals as function of a, b, and c. The column 'int' reports the integer representation of the fault-free circuit output. Let us assume that the faults Sa1@a and Sa1@b are classified as non-redundant and ax-redundant respectively. To test these two faults we can use different vectors (e.g., *vector 0* or *vector 1*). If the test pattern generator selects the *vector 0* to test the two faults, then the signature will be identical for both Sa1@a and Sa1@b, because the faulty output is the same for both faults when *vector 0* is applied . This will lead our technique to reject the circuit even when Sa1@b (ax-redundant) occurs. Therefore, we extend the definition of *aliasing* as follows:

Aliasing During the test response compaction, a non-redundant signature can match an ax-redundant one.



(A)

Vector i	a	b	c	O_1	O_0	int	Sa1@a	Sa1@b	...
0	0	0	0	0	0	0	2	2	...
1	0	0	1	0	0	0	3	2	...
2	0	1	0	1	0	2	2	2	...
3	0	1	1	1	0	2	3	2	...
4	1	0	0	1	0	2	2	2	...
5	1	0	1	1	1	3	3	3	...
6	1	1	0	1	0	2	2	2	...
7	1	1	1	1	1	3	3	3	...
Fault classification:							non-red.	ax-red	...

Value : vector i detects the fault. *Value* is different from 'int_i'

(B)

FIGURE 5.2: Aliasing effect

A simple solution to reduce the aliasing probability is to generate test patterns to detect faults multiple times. Nevertheless, this also increments the final number of test length, thus the cost.

Another solution is to impose some constraints to the test pattern generator to systematically select patterns to avoid aliasing. In the example shown, selecting *vector 1* instead of *vector 0* would solve the problem. Indeed, the faulty output when applying *vector 1* is different for the two faults, thus the signatures will differ, as well.

5.3.3 Experimental results

In this paragraph we discuss experimental results reported in [101]. To evaluate the technique effectiveness, we applied it to a set of AxICs taken from the literature. Specifically, we used Accuracy-Configurable Approximate (ACA) adders from [11], Gracefully-Degrading Adders (GDA) from [22], Generic Accuracy configurable (GeAr) adders from [23], Error Tolerant Adders (ETAII) from [70], and some EvoApprox8b library AxICs [25] (add8_051, add8_036, add8_012, add8_045). Without loss of generality, we used the technique in [88] to perform the fault classification, by resorting to the WCE (Equation 1.3) as error metric. In this way, for each AxIC, we obtained ax-redundant and non-redundant fault lists. Then, we generated test patterns with a commercial ATPG tool [87], instrumented with the classic options (static and dynamic compaction). To generate the patterns, we targeted only the non-redundant

fault list. This is the *conventional* test flow used in [86], [95]. It is worth repeating that any techniques for fault classification and test pattern generation can be employed and any error metric can be used.

Finally, we applied the proposed technique. In details, we simulated the obtained test patterns with the AxIC netlist while injecting the different faults and compacted the responses to obtain the ax-aware signature set, as shown in Figure 5.1a. To compact test responses into signatures, we used a software approach. Specifically, once collected test responses into regular computer files, we used the *md5sum* computer program to calculate MD5 hashes out of them. This constituted the *ax-aware signature set*. In the actual test phase, after the AxIC manufacture, the ax-aware signature set has to be employed, as shown in Figure 5.1b.

To measure the technique efficacy, in [101] we introduce a metric, namely Relative Yield Gain (RYG), expressed as follows:

$$\text{RYG} = 1 - \frac{\text{detected ax-redundant faults}}{\text{total ax-redundant faults}} = 1 - \text{axR FC} \tag{5.1}$$

The RYG measures the part of expected yield gain that is actually achieved as a result of the whole AxA test process. RYG values range from 0 to 1. $RYG = 0$ means that all the ax-redundant faults are detected by test procedure; thus all the faulty, yet acceptable, AxICs are rejected. $RYG = 1$ means that the detection of all ax-redundant faults is avoided, thus the yield gain is as high as expected. To count the number of ax-redundant faults still detected, we enumerated the ax-redundant signatures overlapping the non-redundant ones.

In Table 5.6, we show experimental results. In the first column, we report the name of the analyzed circuits. In the second column, we report the percentage of ax-redundant faults detected with the conventional test (i.e., without our technique). Then, third column reports results obtained with the proposed technique. As it can be seen, the relative yield gain was drastically improved. On average, we achieved 99.84% RYG. For fourteen circuits out of eighteen (~ 77%) the obtained relative yield gain was 100%. For the remaining four circuits, the RYG was always greater than 98%. Such RYG reduction was due to the phenomenon described in Subsection 5.3.2, i.e. aliasing.

To mitigate the aliasing effect, we generated test patterns to detect faults twice. In details, we instrumented the ATPG with the option *-ndetects 2*. As reported in the fourth column of Table 5.6, the aliasing phenomenon was correctly overcome for all the four circuits. The cost of detecting the faults twice was to double the number of test patterns.

Circuit	[1]Relative Yield Gain (%) with conventional test	[1]Relative Yield Gain (%) with proposed technique		Execution Time (s)
		Single detection	Double detection	
add8_051	0.00%	100.00%	-	0.648
add8_036	20.00%	100.00%	-	0.636
add8_012	0.00%	100.00%	-	0.532
add8_045	0.00%	100.00%	-	0.496
GeAr_N8_R2_P2	26.67%	100.00%	-	0.636
ACA_I_N8_Q5	31.86%	99.46%	100.00%	0.764
GDA_St_N8_M8_P3	18.70%	100.00%	-	0.704
GeAr_N16_R6_P4	12.75%	100.00%	-	0.724
ACA_II_N16_Q8	30.83%	100.00%	-	0.772
ETAII_N16_Q8	28.92%	98.58%	100.00%	0.924
GDA_St_N16_M4_P4	10.10%	100.00%	-	1.324
GDA_St_N16_M4_P8	23.36%	100.00%	-	1.276
GeAr_N16_R4_P4	30.83%	100.00%	-	0.78
GeAr_N16_R4_P8	30.00%	99.40%	100.00%	0.852
GeAr_N16_R2_P4	34.48%	99.62%	100.00%	1.016
ACA_II_N16_Q4	24.15%	100.00%	-	1.152
ETAII_N16_Q4	24.15%	100.00%	-	1.22
ACA_I_N16_Q4	24.88%	100.00%	-	1.476
Average	**20.65%**	**99.84%**	**100.00%**	**0.89 s**

[1] Higher is better

TABLE 5.6: Ax-R faults detected with proposed technique compared to conventional test

Clearly, ad hoc methods can be implemented to overcome aliasing. As an example, some test pattern generation techniques discussed in Chapter 4 generate test patterns that intrinsically avoid the aliasing phenomenon [32], [88], [94]. Indeed, those techniques generate test patterns that always produce error values greater than the threshold when detecting non-redundant faults. On the contrary, error values lower than the threshold are produced when detecting ax-redundant faults. Therefore, non-redundant signatures cannot overlap ax-redundant ones. Next section shows results also with ad hoc methods. Finally, concerning the experiment execution time, the table's last column shows a run-time always smaller than 1.5 seconds (0.89 seconds, on average).

5.4 Evaluation

In this subsection we evaluate whether the proposed technique resolves the problem highlighted at the end of last chapter: in Section 4.4, we remarked that, while test ax-aware test pattern generation techniques (with and without pattern selection) provide good improvements compared to conventional ATPG, some ax-redundant

faults are still detected in the test application phase. This is due to the intrinsic structure of AxICs.

Therefore, we applied the proposed technique to AxICs from the EvoApprox8b library [25], as earlier, to evaluate the improvements. We used the same experimental setup as in Section 5.2, i.e. we used test sets obtained by using both conventional and ax-aware generation techniques, considered WCE as error metric, and used MD5 hashes to generate signatures. To be compliant with results shown all

Conventional pattern generation[1]

	Add8		Mul8		Mul16		Mul32	
	axR FC[2]	YIL[2]	axR FC[2]	YIL[2]	axR FC[2]	YIL[2]	axR FC[2]	YIL[2]
Min	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Max	66.67%	3.28%	50.00%	1.62%	5.26%	1.82%	2.60%	0.98%
Avg	**1.05%**	**0.13%**	**0.88%**	**0.30%**	**0.45%**	**0.16%**	**0.30%**	**0.12%**

Ax-aware pattern generation (with and without pattern selection)[1]

	Add8		Mul8		Mul16		Mul32	
	axR FC[2]	YIL[2]	axR FC[2]	YIL[2]	axR FC[2]	YIL[2]	axR FC[2]	YIL[2]
Min	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Max	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Avg	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**

[1]100% non-redundant FC always achieved [2]Lower is better

TABLE 5.7: Ax-redundant FC (axR FC) and Yield Increase Loss (YIL) results when using the proposed test set application technique

along the manuscript, in Table 5.7, we report experimental results in terms of axR FC and YIL. As reported, results prove a drastic improvement. In the upper part of the table, it can be seen that the axR FC and YIL were drastically reduced when conventionally generated patterns were used. On average, we achieved axR FC between 0.3% and 1.05% and a corresponding YIL in the range 0.12% – 0.3%. The gap between ideal results – i.e., 0% for both axR FC and YIL – and the obtained ones is due to the aliasing phenomenon, discussed in Subsection 5.3.2. By using corrective methods, results can be further improved.

Indeed, as reported in the lower part of the table and as predicted in Section 5.3.3, for ax-aware generated patterns, the yield gain was always maximum, i.e. the actual yield increase was always equal to expected Yield Increase (eYI), established in fault classification phase (see Chapter 3). Indeed, in experiments both axR FC and YIL were always 0%.

More importantly, in contrast with threshold testing [32], for all the experiments nR FC was always 100%.

5.5 Chapter summary

In this chapter we presented the AxA test set application. Firstly, we showed and discussed the issues related to the test set application in the context of AxICs. We showed with an example that it is not always possible to avoid detecting some ax-redundant faults, due to the AxIC structure.

Then, we tried to adapt an existing technique for conventional ICs to AxICs (the threshold testing [32]). Unfortunately, we discovered that specific conditions have to be met in order to use the technique.

Therefore, we proposed a new AxA test set application technique to deal with the encountered limitations. The technique is based on the well-know signature analysis concept, successfully applied to built-in self-test architectures in the seventies [33]. Result obtained with the proposed technique were really good. We also described the aliasing phenomenon in the AxIC context and evaluated some corrective methods to deal with it.

Chapter 6

Discussion and conclusions

Contents

In this Chapter, we briefly review the concepts and techniques presented all along this thesis and draw the future directions.

6.1 Summary and considerations

The introduction of approximate computing paradigm in the panorama of information technology, brought multiple opportunities to different extents. The fundamental goal of approximate computing is to improve the system efficiency (time/area/energy) by relaxing result's accuracy requirements. Approximate computing has been applied at different levels of the computing systems, from hardware to software, passing through architectures. Among all the works of the last two decades, approximate computing has been also employed to realize a new class of integrated circuits, i.e. approximate integrated circuits or AxICs. The introduction of a new class of circuits brought along new challenges, as well as new opportunities, concerning chip test and verification. In particular, approximate chip designers carefully modify the circuit structure to introduce *acceptable error*, in order to still obtain satisfying results. To correctly define the *acceptable* concept, designers resort to *error metrics*. Then, they define *error thresholds* to fix the maximum allowed (i.e., acceptable) error. Therefore, the concept of *faulty* circuit changes. Indeed, two new classes of faults are introduced: ax-redundant faults (i.e., faults causing acceptable errors) and non-redundant faults (i.e., faults causing catastrophic errors). In the testing context, the class of a detectable fault can be determined by measuring the caused error at AxIC's output. If the measured error is higher than the acceptable threshold, then the circuit has to be rejected. However, it may happen that the measured error stays below the acceptable threshold, then the AxIC must not be rejected. Therefore, in this context, test role changes as follows:

- circuits whose observed error is greater than the threshold must be rejected;
- circuit affected by acceptable faults must not be rejected.

This ultimately leads to yield increase and possibly to test cost reduction.

As a consequence of these considerations, we introduced AxA testing, basically composed of three phases: (i) AxA fault classification, (ii) AxA test pattern generation, and (iii) AxA test set application. All AxA testing phases bring important contributions to the final test goal, in the context of AxICs:

AxA fault classification separates catastrophic faults from the acceptable ones. Results of this phase determine the expected Yield Increase (eYI). Achieving an actual yield increase as much close as possible to eYI is one of the AxA testing final goals,

along with the detection of all catastrophic faults. The actual yield increase is the result of the synergy between AxA test pattern generation and AxA test set application.

AxA test pattern generation produces test sets to detect all the catastrophic faults, while detecting as few acceptable ones as possible. Unfortunately, avoiding the detection of some acceptable faults is not always possible. The percentage of covered acceptable fault is measured by using the approximation-Redundant Fault Coverage (axR FC).

AxA test set application must distinguish catastrophic faults from acceptable ones, by observing test responses. In this way axR FC is further reduced and thus the yield is actually increased.

6.1.1 Contributions

All along the thesis we thoroughly discussed all the AxA test phases and presented techniques to deal with each aspect. In Chapter 3, we presented techniques to successfully deal with the fault classification task, when considering different types of metrics. The proposed techniques are based on building a *classifying architecture*, which allows the fault classification into non-redundant and ax-redundant by measuring fault impact on AxIC's output. In Chapter 4, we presented two techniques to generate test patterns with different properties. The first one is based on the simultaneous fault classification (discussed in Chapter 3) and test pattern generation. The resulting test patterns generate output errors always greater than the error threshold when detecting non-redundant faults. This technique is particularly suitable with SCT metrics. Moreover, as shown in Section 5.4, the technique leads to ideal results in terms of yield increase, when AxA test set application techniques are employed [32], [101]. The second test pattern generation technique is based on a systematic test pattern selection. In brief, different test patterns detecting different faults are examined and the best subset – achieving 100% nR FC and minimizing axR FC – is chosen. The selection is performed by formulating and solving an integer linear programming problem. By merging the two proposed test pattern generation approaches – as shown in the experimental Section 4.4 – further improvements can be obtained, compared to the conventional ATPG. We also discovered that AxICs present some intrinsic structural limitations. Indeed, it is often impossible avoiding the detection of some ax-redundant faults. For this reason, in Chapter 5, we presented a test set application technique to drastically improve the yield increase. The

technique is based on the well-know signature analysis concept, successfully applied to built-in self-test architectures in the seventies [33]. The proposed technique allowed us to reach ideal results almost in all cases. In summary, we showed that the synergy of the techniques proposed for these last two phases (i.e., test pattern generation and test application) led to achieve optimal results.

6.1.2 Considerations

Now, let us express some further considerations. Concerning AxA test set application, the technique that we introduced in Section 5.3 achieved very good results even when no particular AxA test pattern generation techniques were employed (see results in Section 5.4). We could claim that there is no need to include AxA test pattern generation techniques in the test flow to achieve a final high quality AxA test. However, we have to take into account the cost of implementing the proposed AxA test set application technique. In external test (post-manufacturing), it can be implemented without a big overhead. Indeed, the Automatic Test Equipment (ATE) software can be modified to implement a signature analysis, as discussed in Section 5.3. Nevertheless, when it comes to self-testing hardware, a non-negligible overhead has to be taken into account. As an example, we can consider the BIST architecture (discussed in Section 1.1.4). In BIST, given a test set, test responses are compacted together into a signature. The latter is compared with the golden one (i.e., the signature generated by the fault-free circuit), stored within the BIST architecture. If the two signature are identical, the circuit is considered fault-free. As

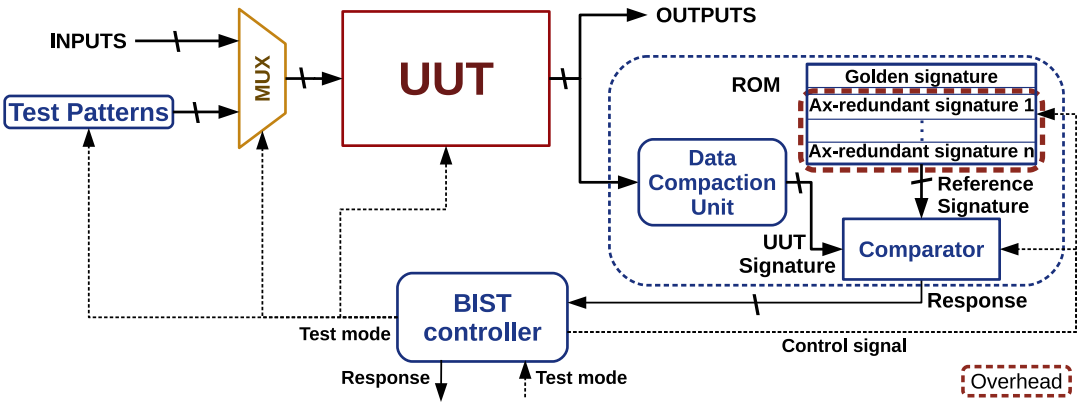


FIGURE 6.1: ax-aware BIST hypothetical architecture

suggested by Figure 6.1, to turn the conventional BIST into an *ax-aware BIST*, the technique discussed in Section 5.3 can be implemented. Thus, multiple *ax-redundant* signatures need to be stored within the circuit, leading to area overhead and to extra time to perform the test (i.e., to compare the test response with all the ax-redundant signatures). The number of ax-redundant signatures depends on the quality of the

generated AxA test set. The higher the quality (low axR FC), the fewer the number of ax-redundant signatures, thus the overhead. Therefore, from the perspective of self testing, improving the AxA test pattern generation, as discussed in Chapter 4, becomes important.

In conclusion, combining AxA test pattern generation and test application techniques turns out to be necessary to improve the overall AxA test quality. Anyhow, the choice of which AxA test techniques to embed in the test flow depends on testing requirements.

6.2 Future perspectives

In this section, we discuss further potential contexts of application for the proposed work, besides approximate integrated circuits. Furthermore, we draw future research directions.

6.2.1 Contexts of application

The techniques presented in this thesis were designed in the context of AxIC testing. Nevertheless, they can be adapted to any kind of domain needing the selective test of fault subsets in integrated circuits. For instance, in [102] and [103], faults that cannot produce any failures in the operational conditions of embedded processor cores were classified as *functionally untestable*. In [104], the classification was extended to special purpose systems (i.e., built to perform a single application). In this scenario faults that cannot produce any failures, due to the specific application code executed by the CPU, are classified as *on-line functionally untestable*. According to the ISO26262 automotive standard terminology, these faults are called "safe faults application dependent". In safety critical applications, achieving a sufficient fault coverage according to the target reliability figure (e.g. ISO 26262 for automotive, DO-254 for avionics, IEC 61508 for industrial systems) is crucial. To do this, the identification of functionally untestable faults and their exclusion from the testing process are necessary. Moreover, this permits reducing over-testing effects, which, in turns, increase the yield and thus the profit of semiconductor companies. In [102]–[104], authors particularly focused on identifying functionally untestable faults. However, no techniques were proposed to actually generating test patterns to avoid the detection of such faults, nor to actually increase the yield. Therefore, test pattern generation and test set application techniques described in Chapters 4 and 5 may be useful to extend the framework presented in [103]. Indeed, as long as faults are properly classified, the proposed techniques are applicable.

Another possible context of application is represented by the Deep Neural Networks (DNNs). DNNs have gained prominence in recent years, also in safety critical applications. As an example, they are being deployed on hardware accelerators in self-driving cars for real-time image classification. Several DNN hardware acceleration techniques have been proposed in last years [105]–[107]. The application context in which such systems are deployed involves human lives. Therefore, DNN hardware accelerators must be compliant with safety standards. As a result, works to assess the reliability of these systems, are emerging. For example, in [108] and [109] the impact of soft errors and of permanent faults on DNN systems was characterized. It turned out that the DNN resilience depends on multiple factors, (e.g., data types, actual values, data reuses, and network structure). This means that faults can affect two similar systems in different ways, depending on those aforementioned factors. A catastrophic fault for a system can be totally harmless to another. Basing on this insight, the online test efficiency of DNN hardware accelerators can be improved by using the techniques proposed in this thesis. Indeed, fault classification techniques inspired by the ones proposed in [88], [95] (Chapter 3) can be designed to classify faults into acceptable and catastrophic. Consequently, the test pattern generation technique proposed in [96] (Chapter 4) can produce test patterns detecting all the catastrophic conditions and as less acceptable ones as possible. Finally, the test set application technique proposed in [101] (Chapter 5) can cope with the classification of faulty scenarios into catastrophic and acceptable.

6.2.2 Future research directions

As highlighted in the last section, all the AxA testing phases contribute to the high-quality test of AxICs. However, all the mentioned techniques do not come without cost. In particular, while acceptable faults constitute an opportunity of increasing the production yield, unfortunately their detection is not straightforward to avoid. In fact, AxA test techniques introduce some overhead (e.g., more difficult test pattern generation, overhead in test set application, especially in BIST) to achieve this goal.

An immediate solution that comes to mind is to eliminate the problem at source: by drawing inspiration from Design for Testability, *Approximation for Testability (AfT)* could suitably tackle the issue. Carefully introducing acceptable faults into the AxIC until only catastrophic ones are left would bring two advantages:

1. further (safe!) approximation, thus gains in (area/power/timing) and
2. testability increase.

To do so, an iterative process composed of (i) fault classification, (ii) fault injection, and (iii) circuit re-synthesis should be implemented. As a result, testing a

so-obtained AxIC would not require more effort than conventional test, since no acceptable faults would be present anymore. Moreover, the introduced approximation would respect the error threshold by definition.

This and other AfT techniques might represent future directions for the testing of approximate integrated circuits.

Chapter 7

Scientific Contributions

In this chapter, we list our scientific contributions relevant to the treated topic.

Publications in International Journals

1. L. Anghel, M. Benabdenbi, A. Bosio, M. Traiola, and E. I. Vatajelu, "Test and reliability in approximate computing", *Journal of Electronic Testing*, vol. 34, no. 4, pp. 375–387, 2018, ISSN: 1573-0727. DOI: 10.1007/s10836-018-5734-9

2. M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "A test pattern generation technique for approximate circuits based on an ilp-formulated pattern selection procedure", *IEEE Transactions on Nanotechnology*, pp. 1–1, 2019, ISSN: 1536-125X. DOI: 10.1109/TNANO.2019.2923040

3. M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "A survey of testing techniques for approximate integrated circuits", *Proceedings of IEEE (under review)*, 2020

Publications in Proceedings of International Conferences

1. I. Wali, M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Towards approximation during test of integrated circuits", in *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2017, pp. 28–33. DOI: 10.1109/DDECS.2017.7934574

2. M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Towards digital circuit approximation by exploiting fault simulation", in *2017 IEEE East-West Design Test Symposium (EWDTS)*, 2017, pp. 1–7. DOI: 10.1109/EWDTS.2017.8110108

3. M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Testing approximate digital circuits: Challenges and opportunities", in *2018 IEEE 19th*

Latin-American Test Symposium (LATS), 2018, pp. 1–6. DOI: 10.1109/LATW.2018.8349681

4. M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "On the comparison of different atpg approaches for approximate integrated circuits", in *2018 IEEE 21st International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2018, pp. 85–90. DOI: 10.1109/DDECS.2018.00022

5. L. Sekanina, Z. Vasicek, A. Bosio, M. Traiola, P. Rech, D. Oliveria, F. Fernandes, and S. Di Carlo, "Special session: How approximate computing impacts verification, test and reliability", in *2018 IEEE 36th VLSI Test Symposium (VTS)*, 2018, pp. 1–1. DOI: 10.1109/VTS.2018.8368628

6. M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Investigation of mean-error metrics for testing approximate integrated circuits", in *2018 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2018, pp. 1–6. DOI: 10.1109/DFT.2018.8602939

7. M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Maximizing yield for approximate integrated circuits", in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020

Publications in International Workshops without Proceedings

1. I. Wali, M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Can we approximate the test of integrated circuits?", in *3rd Workshop On Approximate Computing (WAPCO)*, 2017, pp. 1–7. [Online]. Available: https://wapco.e-ce.uth.gr/papers/SESSION2/paper_2_1.pdf

2. M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Testing integrated circuits for approximate computing applications", in *4rd Workshop On Approximate Computing (WAPCO)*, 2018, pp. 1–7

3. M. Traiola, A. Virazel, and P. Girard, "On the testing of approximate integrated circuits for embedded applications considering average-error metrics", in *AxC18 3rd Workshop on Approximate Computing (in conjuction with European Test Symposium (ETS))*, 2018, pp. 1–7

Publications in National Conferences without Proceedings

1. M. Traiola, A. Virazel, P. Girard, and A. Bosio, "A case study on the approximate test of integrated circuits", in *Colloque du GDR SoC2*, 2017, pp. 1–7

2. M. Traiola, A. Virazel, P. Girard, and A. Bosio, "Automatic test pattern generation for approximate integrated circuits", in *Colloque du GDR SoC2*, 2018, pp. 1–7

3. M. Traiola, A. Virazel, P. Girard, and A. Bosio, "Test techniques for approximate integrated circuits", in *Colloque du GDR SoC2*, 2019, pp. 1–7

Bibliography

[1] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey", *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, 2016, ISSN: 2168-2356. DOI: 10.1109/MDAT.2015.2505723.

[2] S. Mittal, "A survey of techniques for approximate computing", *ACM Comput. Surv.*, vol. 48, no. 4, 62:1–62:33, Mar. 2016, ISSN: 0360-0300. DOI: 10.1145/2893356.

[3] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design", in *2013 18th IEEE European Test Symposium (ETS)*, 2013, pp. 1–6. DOI: 10.1109/ETS.2013.6569370.

[4] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing", in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–9. DOI: 10.1145/2463209.2488873.

[5] S. Rehman, B. S. Prabakaran, W. El-Harouni, M. Shafique, and J. Henkel, "Heterogeneous approximate multipliers: Architectures and design methodologies", in *Approximate Circuits: Methodologies and CAD*, S. Reda and M. Shafique, Eds. Springer International Publishing, 2019, pp. 45–66, ISBN: 978-3-319-99322-5. DOI: 10.1007/978-3-319-99322-5_3.

[6] H. Jiang, J. Han, and F. Lombardi, "A comparative review and evaluation of approximate adders", in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, ser. GLSVLSI '15, Pittsburgh, Pennsylvania, USA: ACM, 2015, pp. 343–348, ISBN: 978-1-4503-3474-7. DOI: 10.1145/2742060.2743760.

[7] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture", in *2011 24th Internatioal Conference on VLSI Design*, 2011, pp. 346–351. DOI: 10.1109/VLSID.2011.51.

[8] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel, "Architectural-space exploration of approximate multipliers", in *Proceedings of the 35th International Conference on Computer-Aided Design*, ser. ICCAD '16, Austin, Texas: ACM, 2016, 80:1–80:8, ISBN: 978-1-4503-4466-1. DOI: 10.1145/2966986.2967005.

[9] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "Impact: Imprecise adders for low-power approximate computing", in *IEEE/ACM International Symposium on Low Power Electronics and Design*, 2011, pp. 409–414. DOI: 10.1109/ISLPED.2011.5993675.

[10] J. Miao, K. He, A. Gerstlauer, and M. Orshansky, "Modeling and synthesis of quality-energy optimal approximate adders", in *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2012, pp. 728–735.

[11] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs", in *DAC Design Automation Conference 2012*, 2012, pp. 820–825. DOI: 10.1145/2228360.2228509.

[12] V. K. Chippa, S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing: An integrated hardware approach", in *2013 Asilomar Conference on Signals, Systems and Computers*, 2013, pp. 111–117. DOI: 10.1109/ACSSC.2013.6810241.

[13] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication", *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 984–994, 2015, ISSN: 0018-9340. DOI: 10.1109/TC.2014.2308214.

[14] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications", in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2010, pp. 957–960. DOI: 10.1109/DATE.2010.5456913.

[15] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: Systematic logic synthesis of approximate circuits", in *DAC Design Automation Conference 2012*, 2012, pp. 796–801. DOI: 10.1145/2228360.2228504.

[16] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits", in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2013, pp. 1367–1372. DOI: 10.7873/DATE.2013.280.

[17] J. Miao, A. Gerstlauer, and M. Orshansky, "Multi-level approximate logic synthesis under general error constraints", in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2014, pp. 504–510. DOI: 10.1109/ICCAD.2014.7001398.

[18] Y. Wu and W. Qian, "An efficient method for multi-level approximate logic synthesis under error rate constraint", in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6. DOI: 10.1145/2897937.2897982.

[19] D. Shin and S. K. Gupta, "A new circuit simplification method for error tolerant applications", in *Design, Automation Test in Europe (DATE)*, 2011, pp. 1–6. DOI: 10.1109/DATE.2011.5763248.

[20] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, "Aslan: Synthesis of approximate sequential circuits", in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014. DOI: 10.7873/DATE.2014.377.

[21] L. Holik, O. Lengal, A. Rogalewicz, L. Sekanina, Z. Vasicek, and T. Vojnar, "Towards formal relaxed equivalence checking in approximate computing methodology", *2nd Workshop On Approximate Computing (WAPCO)*, 2016. [Online]. Available: https://wapco.e-ce.uth.gr/2016/papers/SESSION2/wapco2016_2_1.pdf.

[22] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application", in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 48–54. DOI: 10.1109/ICCAD.2013.6691096.

[23] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder", in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6. DOI: 10.1145/2744769.2744778.

[24] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "Axbench: A multiplatform benchmark suite for approximate computing", *IEEE Design Test*, vol. 34, no. 2, pp. 60–68, 2017, ISSN: 2168-2356. DOI: 10.1109/MDAT.2016.2630270.

[25] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approx adders and multipliers for circuit design and benchmarking of approximation methods", in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2017, pp. 258–261. DOI: 10.23919/DATE.2017.7926993.

[26] I. Polian, "Test and reliability challenges for approximate circuitry", *IEEE Embedded Systems Letters*, vol. 10, no. 1, pp. 26–29, 2018, ISSN: 1943-0663. DOI: 10.1109/LES.2017.2754446.

[27] A. Chandrasekharan, D. Große, and R. Drechsler, *Design Automation Techniques for Approximation Circuits: Verification, Synthesis and Test*. Springer, 2019. DOI: 10.1007/978-3-319-98965-5.

[28] L. Anghel, M. Benabdenbi, A. Bosio, M. Traiola, and E. I. Vatajelu, "Test and reliability in approximate computing", *Journal of Electronic Testing*, vol. 34, no. 4, pp. 375–387, 2018, ISSN: 1573-0727. DOI: 10.1007/s10836-018-5734-9.

[29] L. Sekanina, Z. Vasicek, A. Bosio, M. Traiola, P. Rech, D. Oliveria, F. Fernandes, and S. Di Carlo, "Special session: How approximate computing impacts verification, test and reliability", in *2018 IEEE 36th VLSI Test Symposium (VTS)*, 2018, pp. 1–1. DOI: 10.1109/VTS.2018.8368628.

[30] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders", *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1760–1771, 2013, ISSN: 0018-9340. DOI: 10.1109/TC.2012.146.

[31] G. Gielen, P. D. Wit, E. Maricau, J. Loeckx, J. Martin-Martinez, B. Kaczer, G. Groeseneken, R. Rodriguez, and M. Nafria, "Emerging yield and reliability challenges in nanometer cmos technologies", in *Design, Automation and Test in Europe (DATE)*, 2008, pp. 1322–1327. DOI: 10.1109/DATE.2008.4484862.

[32] Z. Jiang and S. K. Gupta, "An atpg for threshold testing: Obtaining acceptable yield in future processes", in *Proceedings. International Test Conference*, 2002, pp. 824–833. DOI: 10.1109/TEST.2002.1041836.

[33] R. A. Frohwerk, "Signature analysis: A new digital field service method", 1977.

[34] M L. Bushnell and V D. Agarwal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Jan. 2000. DOI: 10.1007/b117406.

[35] R. D. Eldred, "Test routines based on symbolic logical statements", *J. ACM*, vol. 6, no. 1, pp. 33–37, Jan. 1959, ISSN: 0004-5411. DOI: 10.1145/320954.320957.

[36] V. Agrawal, S. Seth, and I. C. Society, *Tutorial test generation for VLSI chips*. Computer Society Press, 1988, ISBN: 9780818687860. [Online]. Available: https://books.google.fr/books?id=WC1TAAAAMAAJ.

[37] A. Bosio, D. Menard, and O. Sentieys, *A Comprehensive Analysis of Approximate Computing Techniques: From Component- to Application-Level*, DATE 2019 - 22nd IEEE/ACM Design, Automation and Test in Europe, Mar. 2019. [Online]. Available: https://hal.inria.fr/hal-01941757.

[38] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Enerj: Approximate data types for safe and general low-power computation", in *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '11, San Jose, California, USA: ACM, 2011, pp. 164–174, ISBN: 978-1-4503-0663-8. DOI: 10.1145/1993498.1993518.

[39] J. Bornholt, T. Mytkowicz, and K. S. McKinley, "Uncertain< t >: A first-order type for uncertain data", Tech. Rep. MSR-TR-2013-46, 2013. [Online]. Available: https://www.microsoft.com/en-us/research/publication/uncertaint-a-first-order-type-for-uncertain-data/.

[40] Qian Zhang, F. Yuan, R. Ye, and Q. Xu, "Approxit: An approximate computing framework for iterative methods", in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2014, pp. 1–6. DOI: 10.1109/DAC.2014.6881424.

[41] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke, "Sage: Self-tuning approximation for graphics engines", in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2013, pp. 13–24.

[42] W. Baek and T. Chilimbi, "Green: A framework for supporting energy-conscious programming using controlled approximation", ACM SIGPLAN, 2010. [Online]. Available: https://www.microsoft.com/en-us/research/publication/green-framework-supporting-energy-conscious-programming-using-controlled-approximation/.

[43] M. Ringenburg, A. Sampson, I. Ackerman, L. Ceze, and D. Grossman, "Monitoring and debugging the quality of results in approximate programs", *SIGARCH Comput. Archit. News*, vol. 43, no. 1, pp. 399–411, Mar. 2015, ISSN: 0163-5964. DOI: 10.1145/2786763.2694365.

[44] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Rumba: An online quality management system for approximate computing", in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 554–566. DOI: 10.1145/2749469.2750371.

[45] M. Carbin, S. Misailovic, and M. C. Rinard, "Verifying quantitative reliability for programs that execute on unreliable hardware", *SIGPLAN Not.*, vol. 48, no. 10, pp. 33–52, Oct. 2013, ISSN: 0362-1340. DOI: 10.1145/2544173.2509546.

[46] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen, "The ins and outs of the probabilistic model checker mrmc", *Performance Evaluation*, vol. 68, no. 2, pp. 90 –104, 2011, Advances in Quantitative Evaluation of Systems, ISSN: 0166-5316. DOI: 10.1016/j.peva.2010.04.001.

[47] M. Kwiatkowska, G. Norman, and D. Parker, "Prism: Probabilistic symbolic model checker", in *Computer Performance Evaluation: Modelling Techniques and Tools*, T. Field, P. G. Harrison, J. Bradley, and U. Harder, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 200–204, ISBN: 978-3-540-46029-9. DOI: 10.1007/3-540-46029-2_13.

[48] S. Chaudhuri, S. Gulwani, R. Lublinerman, and S. Navidpour, "Proving programs robust", in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11, Szeged, Hungary: ACM, 2011, pp. 102–112, ISBN: 978-1-4503-0443-6. DOI: 10.1145/2025113.2025131.

[49] S. Misailovic, M. Carbin, S. Achour, Z. Qi, and M. C. Rinard, "Chisel: Reliability- and accuracy-aware optimization of approximate computational kernels", *SIGPLAN Not.*, vol. 49, no. 10, pp. 309–328, Oct. 2014, ISSN: 0362-1340. DOI: 10.1145/2714064.2660231.

[50] M. Rinard, "Probabilistic accuracy bounds for fault-tolerant computations that discard tasks", in *Proceedings of the 20th Annual International Conference on Supercomputing*, ser. ICS '06, Cairns, Queensland, Australia: ACM, 2006, pp. 324–334, ISBN: 1-59593-282-8. DOI: 10.1145/1183401.1183447.

[51] A. Sampson, P. Panchekha, T. Mytkowicz, K. S. McKinley, D. Grossman, and L. Ceze, "Expressing and verifying probabilistic assertions", *SIGPLAN Not.*, vol. 49, no. 6, pp. 112–122, Jun. 2014, ISSN: 0362-1340. DOI: 10.1145/2666356.2594294.

[52] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard, "Quality of service profiling", in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10, Cape Town, South Africa: ACM, 2010, pp. 25–34, ISBN: 978-1-60558-719-6. DOI: 10.1145/1806799.1806808.

[53] E. Schkufza, R. Sharma, and A. Aiken, "Stochastic optimization of floating-point programs with tunable precision", *SIGPLAN Not.*, vol. 49, no. 6, pp. 53–64, Jun. 2014, ISSN: 0362-1340. DOI: 10.1145/2666356.2594302.

[54] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming", *SIGARCH Comput. Archit. News*, vol. 40, no. 1, pp. 301–312, Mar. 2012, ISSN: 0163-5964. DOI: 10.1145/2189750.2151008.

[55] U. R. Karpuzcu, I. Akturk, and N. S. Kim, "Accordion: Toward soft near-threshold voltage computing", in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 72–83. DOI: 10.1109/HPCA.2014.6835977.

[56] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs", in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 449–460. DOI: 10.1109/MICRO.2012.48.

[57] S. Z. Gilani, N. S. Kim, and M. Schulte, "Scratchpad memory optimizations for digital signal processing applications", in *2011 Design, Automation Test in Europe*, 2011, pp. 1–6. DOI: 10.1109/DATE.2011.5763158.

[58] D. J. Palframan, N. S. Kim, and M. H. Lipasti, "Precision-aware soft error protection for gpus", in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 49–59. DOI: 10.1109/HPCA.2014.6835966.

[59] M. Shoushtari, A. BanaiyanMofrad, and N. Dutt, "Exploiting partially-forgetful memories for approximate computing", *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 19–22, 2015, ISSN: 1943-0663. DOI: 10.1109/LES.2015.2393860.

[60] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving dram refresh-power through critical data partitioning", *SIGARCH Comput. Archit. News*, vol. 39, no. 1, pp. 213–224, Mar. 2011, ISSN: 0163-5964. DOI: 10.1145/1961295.1950391.

[61] A. Raha, S. Sutar, H. Jayakumar, and V. Raghunathan, "Quality configurable approximate dram", *IEEE Transactions on Computers*, vol. 66, no. 7, pp. 1172–1187, 2017, ISSN: 0018-9340. DOI: 10.1109/TC.2016.2640296.

[62] M. Jung, D. M. Mathew, C. Weis, and N. Wehn, "Invited: Approximate computing with partially unreliable dynamic random access memory — approximate dram", in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–4. DOI: 10.1145/2897937.2905002.

[63] Y. Chen, X. Yang, F. Qiao, J. Han, Q. Wei, and H. Yang, "A multi-accuracy-level approximate memory architecture based on data significance analysis", in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 385–390. DOI: 10.1109/ISVLSI.2016.84.

[64] N. Sayed, F. Oboril, A. Shirvanian, R. Bishnoi, and M. B. Tahoori, "Exploiting stt-mram for approximate computing", in *2017 22nd IEEE European Test Symposium (ETS)*, 2017, pp. 1–6. DOI: 10.1109/ETS.2017.7968217.

[65] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories", in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2013, pp. 25–36.

[66] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta-functions for approximate computing", in *2011 Design, Automation Test in Europe*, 2011, pp. 1–6. DOI: 10.1109/DATE.2011.5763154.

[67] R. Ragavan, B. Barrois, C. Killian, and O. Sentieys, "Pushing the limits of voltage over-scaling for error-resilient applications", in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 476–481. DOI: 10.23919/DATE.2017.7927036.

[68] W. Liu, L. Chen, C. Wang, M. O'Neill, and F. Lombardi, "Inexact floating-point adder for dynamic image processing", in *14th IEEE International Conference on Nanotechnology*, 2014, pp. 239–243. DOI: 10.1109/NANO.2014.6967953.

[69] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, "Approximate xor/xnor-based adders for inexact computing", in *2013 13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013)*, 2013, pp. 690–693. DOI: 10.1109/NANO.2013.6720793.

[70] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application", in *Proceedings of the 2009 12th International Symposium on Integrated Circuits*, 2009, pp. 69–72.

[71] T. Ban, B. Wang, and L. Naviner, "Design, synthesis and application of a novel approximate adder", in *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2018, pp. 488–491. DOI: 10.1109/MWSCAS.2018.8624023.

[72] H. Cai, Y. Wang, L. A. B. Naviner, Zhaohao Wang, and W. Zhao, "Approximate computing in mos/spintronic non-volatile full-adder", in *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2016, pp. 203–208. DOI: 10.1145/2950067.2950101.

[73] R. K. Brayton, A. L. Sangiovanni-Vincentelli, C. T. McMullen, and G. D. Hachtel, *Logic Minimization Algorithms for VLSI Synthesis*. Norwell, MA, USA: Kluwer Academic Publishers, 1984, ISBN: 0898381649. DOI: 10.1007/978-1-4613-2821-6.

[74] J. Miao, A. Gerstlauer, and M. Orshansky, "Approximate logic synthesis under general error magnitude and frequency constraints", in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 779–786. DOI: 10.1109/ICCAD.2013.6691202.

[75] M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Towards digital circuit approximation by exploiting fault simulation", in *2017 IEEE East-West Design Test Symposium (EWDTS)*, 2017, pp. 1–7. DOI: 10.1109/EWDTS.2017.8110108.

[76] A. Yazdanbakhsh, D. Mahajan, B. Thwaites, J. Park, A. Nagendrakumar, S. Sethuraman, K. Ramkrishnan, N. Ravindran, R. Jariwala, A. Rahimi, H. Esmaeilzadeh, and K. Bazargan, "Axilog: Language support for approximate hardware design", in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 812–817. DOI: 10.7873/DATE.2015.0513.

[77] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "Abacus: A technique for automated behavioral synthesis of approximate computing circuits", in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014, pp. 1–6. DOI: 10.7873/DATE.2014.374.

[78] Chaofan Li, Wei Luo, S. S. Sapatnekar, and Jiang Hu, "Joint precision optimization and high level synthesis for approximate computing", in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6. DOI: 10.1145/2744769.2744863.

[79] R. S. Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger, "General-purpose code acceleration with limited-precision analog computation", in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 505–516. DOI: 10.1109/ISCA.2014.6853213.

[80] B. Li, P. Gu, Y. Shan, Y. Wang, Y. Chen, and H. Yang, "Rram-based analog approximate computing", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 12, pp. 1905–1917, 2015, ISSN: 0278-0070. DOI: 10.1109/TCAD.2015.2445741.

[81] B. Barrois, O. Sentieys, and D. Menard, "The hidden cost of functional approximation against careful data sizing — a case study", in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 181–186. DOI: 10.23919/DATE.2017.7926979.

[82] M. Traiola, A. Savino, M. Barbareschi, S. D. Carlo, and A. Bosio, "Predicting the impact of functional approximation: From component- to application-level", in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, 2018, pp. 61–64. DOI: 10.1109/IOLTS.2018.8474072.

[83] M. Traiola, A. Savino, and S. D. Carlo, "Probabilistic estimation of the application-level impact of precision scaling in approximate computing applications", *to appear in Microelectronics Reliability*, 2019.

[84] J. Castro-Godínez, S. Esser, M. Shafique, S. Pagani, and J. Henkel, "Compiler-driven error analysis for designing approximate accelerators", in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1027–1032. DOI: 10.23919/DATE.2018.8342163.

[85] S. Hamdioui, "Electronics and computing in nano-era: The good, the bad and the challenging", in *2015 10th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, 2015, pp. 1–1. DOI: 10.1109/DTIS.2015.7127342.

[86] A. Chandrasekharan, S. Eggersglüß, D. Große, and R. Drechsler, "Approximation-aware testing for approximate circuits", in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 239–244. DOI: 10.1109/ASPDAC.2018.8297312.

[87] (). Tetramax, [Online]. Available: https://www.synopsys.com/.

[88] M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Testing approximate digital circuits: Challenges and opportunities", in *2018 IEEE 19th Latin-American Test Symposium (LATS)*, 2018, pp. 1–6. DOI: 10.1109/LATW.2018.8349681.

[89] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler, "Precise error determination of approximated components in sequential circuits with model checking", in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6. DOI: 10.1145/2897937.2898069.

[90] T. U. Aoki Laboratory. (2016), [Online]. Available: http://www.aoki.ecei.tohoku.ac.jp/arith.

[91] L. Amarú, P.-E. Gaillardon, and G. D. Micheli. (2015). The epfl combinational benchmark suite, [Online]. Available: http://infoscience.epfl.ch/record/207551.

[92] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the iscas-85 benchmarks: A case study in reverse engineering", *IEEE Design Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999, ISSN: 0740-7475. DOI: 10.1109/54.785838.

[93] M. Soeken, D. Große, A. Chandrasekharan, and R. Drechsler, "Bdd minimization for approximate computing", in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016, pp. 474–479. DOI: 10.1109/ASPDAC.2016.7428057.

[94] A. Gebregiorgis and M. B. Tahoori, "Test pattern generation for approximate circuits based on boolean satisfiability", in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 1028–1033. DOI: 10.23919/DATE.2019.8714898.

[95] M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Investigation of mean-error metrics for testing approximate integrated circuits", in *2018 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2018, pp. 1–6. DOI: 10.1109/DFT.2018.8602939.

[96] ——, "A test pattern generation technique for approximate circuits based on an ilp-formulated pattern selection procedure", *IEEE Transactions on Nanotechnology*, pp. 1–1, 2019, ISSN: 1536-125X. DOI: 10.1109/TNANO.2019.2923040.

[97] S. Gass and T. Saaty, "The computational algorithm for the parametric objective function", *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 39–45, 1955. DOI: 10.1002/nav.3800020106.

[98] R. Kannan and C. L. Monma, "On the computational complexity of integer programming problems", in *Optimization and Operations Research*, R. Henn, B. Korte, and W. Oettli, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, pp. 161–172, ISBN: 978-3-642-95322-4. DOI: 10.1007/978-3-642-95322-4_17.

[99] C. H. Papadimitriou, "On the complexity of integer programming", *J. ACM*, vol. 28, no. 4, pp. 765–768, Oct. 1981, ISSN: 0004-5411. DOI: 10.1145/322276.322287.

[100] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems", *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960, ISSN: 00129682, 14680262. [Online]. Available: http://www.jstor.org/stable/1910129.

[101] M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Maximizing yield for approximate integrated circuits", in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020.

[102] P. Bernardi, M. Bonazza, E. Sanchez, M. Sonza Reorda, and O. Ballan, "On-line functionally untestable fault identification in embedded processor cores", in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2013, pp. 1462–1467. DOI: 10.7873/DATE.2013.298.

[103] A. Riefert, R. Cantoro, M. Sauer, M. Sonza Reorda, and B. Becker, "A flexible framework for the automatic generation of sbst programs", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 10, pp. 3055–3066, 2016, ISSN: 1063-8210. DOI: 10.1109/TVLSI.2016.2538800.

[104] R. Cantoro, A. Firrincieli, D. Piumatti, M. Restifo, E. Sanchez, and M. S. Reorda, "About on-line functionally untestable fault identification in microprocessor cores for safety-critical applications", in *2018 IEEE 19th Latin-American Test Symposium (LATS)*, 2018, pp. 1–6. DOI: 10.1109/LATW.2018.8349679.

[105] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning", *SIGPLAN Not.*, vol. 49, no. 4, pp. 269–284, Feb. 2014, ISSN: 0362-1340. DOI: 10.1145/2644865.2541967.

[106] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks", in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 367–379. DOI: 10.1109/ISCA.2016.40.

[107] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network", in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16, Seoul, Republic of Korea: IEEE Press, 2016, pp. 243–254, ISBN: 978-1-4673-8947-1. DOI: 10.1109/ISCA.2016.30.

[108] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications", in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17, Denver, Colorado: ACM, 2017, 8:1–8:12, ISBN: 978-1-4503-5114-0. DOI: 10.1145/3126908.3126964.

[109] A. Bosio, P. Bernardi, A. Ruospo, and E. Sanchez, "A reliability analysis of a deep neural network", in *2019 IEEE Latin American Test Symposium (LATS)*, 2019, pp. 1–6. DOI: 10.1109/LATW.2019.8704548.

[110] M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "A survey of testing techniques for approximate integrated circuits", *Proceedings of IEEE (under review)*, 2020.

[111] I. Wali, M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Towards approximation during test of integrated circuits", in *2017 IEEE 20th*

International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS), 2017, pp. 28–33. DOI: 10.1109/DDECS.2017.7934574.

[112] M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "On the comparison of different atpg approaches for approximate integrated circuits", in *2018 IEEE 21st International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2018, pp. 85–90. DOI: 10.1109/DDECS.2018.00022.

[113] I. Wali, M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Can we approximate the test of integrated circuits?", in *3rd Workshop On Approximate Computing (WAPCO)*, 2017, pp. 1–7. [Online]. Available: https://wapco.e-ce.uth.gr/papers/SESSION2/paper_2_1.pdf.

[114] M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Testing integrated circuits for approximate computing applications", in *4rd Workshop On Approximate Computing (WAPCO)*, 2018, pp. 1–7.

[115] M. Traiola, A. Virazel, and P. Girard, "On the testing of approximate integrated circuits for embedded applications considering average-error metrics", in *AxC18 3rd Workshop on Approximate Computing (in conjuction with European Test Symposium (ETS))*, 2018, pp. 1–7.

[116] M. Traiola, A. Virazel, P. Girard, and A. Bosio, "A case study on the approximate test of integrated circuits", in *Colloque du GDR SoC2*, 2017, pp. 1–7.

[117] ——, "Automatic test pattern generation for approximate integrated circuits", in *Colloque du GDR SoC2*, 2018, pp. 1–7.

[118] ——, "Test techniques for approximate integrated circuits", in *Colloque du GDR SoC2*, 2019, pp. 1–7.