Error handling and energy estimation for error resilient near-threshold computing

Rengarajan Ragavan

N° d'ordre : ANNÉE 2017

UNIVERSITÉ DE RENNES 1

UNIVERSITE BRETAGNE LOIRE

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Bretagne Loire

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Traitement du Signal et Télécommunications

École doctorale MATISSE

présentée par

Rengarajan RAGAVAN

préparée à l'unité de recherche UMR6074 IRISA
Institut de recherche en informatique et systèmes aléatoires - CAIRN
École Nationale Supérieure des Sciences Appliquées et de Technologie

---

Error Handling and Energy Estimation Framework For Error Resilient Near-Threshold Computing

Thèse soutenue à Lannion
le 22 septembre 2017

devant le jury composé de :

Edith BEIGNÉ
Directrice de Recherche à CEA Leti / rapporteur

Patrick GIRARD
Directrice de Recherche à CNRS, LIRMM / rapporteur

Lorena ANGHEL
Professeur à Grenoble INP, TIMA / examinateur

Daniel MÉNARD
Professeur à INSA Rennes, IETR / examinateur

Olivier SENTIEYS
Professeur à l'Université de Rennes 1 / directeur de thèse

Cédric KILLIAN
Maître de Conférences à l'Université de Rennes 1 / co-directeur de thèse

To My Parents, My Brother, and My Teachers

Acknowledgements

I sincerely express my gratitude to my thesis supervisor Prof. Olivier Sentieys who gave me the opportunity to work him in prestigious INRIA/IRISA labs. His guidance and confidence in me gave extra boost and cultivated more interest in me to do this research work in a better way. His feedback and comments refined the quality of my research contributions and this thesis dissertation.

I am very grateful to my thesis co-supervisor Dr. Cédric Killian for his encouragement and support. He was inspirational to me throughout my Ph.D. tenure. His suggestions and critical evaluation helped me to innovatively address various bottlenecks in this thesis work. I register my special thanks to Cédric for his efforts in translating the long abstract of this thesis to French.

I would like to thank my colleague Philippe Quémerais, Research Engineer, IRISA labs for his extensive support in configuring and tutoring various CAD tools. I acknowledge my fellow research student Benjamin Barrois for his help in modelling approximate arithmetic operators.

I'm thankful to Mme. Laurence Gallas for her French lessons that helped me a lot during my stay in Lannion. Also, I would like to thank all the faculty members, researchers, and colleagues of ENSSAT who made my stay in Lannion most cherishable.

This thesis work has seen the light of the day thanks to Université de Rennes 1, and IRISA laboratory for funding and facilitating this thesis work. Thanks are due to Nadia, Joelle, and Angelique for their support in administrative matters and during my missions abroad.

I also express my gratitude to the entire jury panel for reading my thesis dissertation and taking part in the thesis defense. Also, special thanks to the reviewers for their valuable review comments that improved the quality of this dissertation.

I offer my sincere thanks to all my friends and well-wishers who were supporting me all these years and looking for my betterment. Last but not least, I express my everlasting gratitude to my parents, my brother and almighty without them I cannot be what I am today.

10th October 2017 Rengarajan

Lannion, France

Abstract

Advancement in CMOS IC technology has improved the way in which integrated circuits are designed and fabricated over decades. Scaling down the size of transistors from micrometers to few nanometers has given the capability to place more tinier transistors in a unit of chip area that upheld Moore's law over the years. On the other hand, feature size scaling imposed various challenges like physical, material, power-thermal, technological, and economical that hinder the trend of scaling. To improve energy efficiency, Dynamic Voltage Scaling (DVS) is employed in most of the Ultra-Low Power (ULP) designs. In near-threshold region (NTR), the supply voltage of the transistor is slightly more than the threshold voltage of the transistor. In this region, the delay and energy both are roughly *linear* with the supply voltage. The I_{on}/I_{off} ratio is higher compared to the sub-threshold region, which improves the speed and minimizes the leakage. Due to the reduced supply voltage, the circuits operating in NTR can achieve a 10× reduction in energy per operation at the cost of the same magnitude of reduction in the operating frequency. In the present trend of sub-nanometer designs, inherent low-leakage technologies like FD-SOI (Fully Depleted Silicon On Insulator), enhance the benefits of voltage scaling by utilizing techniques like Back-Biasing. However reduction in V_{dd} augments the impact of variability and timing errors in sub-nanometer designs.

The main objective of this work is to handle timing errors and to formulate a framework to estimate energy consumption of error resilient applications in the context of near-threshold computing. It is highly beneficial to design a digital system by taking advantage of NTR, DVS, and FD-SOI. In spite of the advantages, the impact of variability and timing errors outweighs the above-mentioned benefits. There are existing methods that can predict and prevent errors, or detect and correct errors. But there is a need for a unified approach to handle timing errors in the near-threshold regime.

In this thesis, *Dynamic Speculation* based error detection and correction is explored in the context of adaptive voltage and clock overscaling. Apart from error detection and correction, some errors can also be tolerated or, in other words, circuits can be pushed beyond their limits to compute incorrectly to achieve higher energy efficiency. The proposed error detection and correction method achieves 71% overclocking with 2% additional hardware cost. This work involves extensive study of design at gate level to understand the behaviour of gates under overscaling of supply voltage, bias voltage and clock frequency (collectively called as *operating triads*). A bottom-up approach is taken: by studying trends of energy vs. error of basic arithmetic operators at transistor level. Based on profiling of the arithmetic operators, a tool flow is formulated to estimate energy and error metrics for different operating triads. We achieve maximum energy efficiency of 89% for arithmetic operators like 8-bit and 16-bit adders at the cost of 20%

faulty bits by operating in NTR. A statistical model is developed for the arithmetic operators to represent the behaviour of the operators for different variability impacts. This model is used for approximate computing of error resilient applications that can tolerate acceptable margin of errors. This method is further explored for execution unit of a VLIW processor. The proposed framework provides a quick estimation of energy and error metrics of benchmark programs by simple compilation in using C compiler. In the proposed energy estimation framework, characterization of arithmetic operators is done at transistor level, and the energy estimation is done at functional level. This hybrid approach makes energy estimation faster and accurate for different operating triads. The proposed framework estimates energy for different benchmark programs with 98% accuracy compared to SPICE simulation.

Abbreviations

AVS	**A**daptive **V**oltage **s**caling
BER	**B**it **E**rror **R**ate
BKA	**B**rent-**K**ung **A**dder
BOX	**B**uried **OX**ide
CCS	**C**onditional **C**arry **S**elect
CDM	**C**ritical **D**elay **M**argin
CGRA	**C**oarse-**G**rained **R**econfigurable **A**rchitecture
CMF	**C**ommon **M**ode **F**ailure
CMOS	**C**omplementary **M**etal-**O**xide-**S**emiconductor
DDMR	**D**iverse **D**ouble **M**odular **R**edundancy
DMR	**D**ouble **M**odular **R**edundancy
DSP	**D**igital **S**ignal **P**rocessor
DVFS	**D**ynamic **V**oltage **F**requency **s**caling
DVS	**D**ynamic **V**oltage **s**caling
EDAP	**E**nergy **D**elay **A**rea **P**roduct
EDP	**E**nergy **D**elay **P**roduct
FBB	**F**orward **B**ody **B**iasing
FEHM	**F**lexible **E**rror **H**andling **M**odule
FF	**F**lip-**F**lop
FIR	**F**inite **I**mpulse **R**esponse
FPGA	**F**ield-**P**rogramming **G**ate **A**rray
FDSOI	**F**ully **D**epleted **S**ilicon **o**n **I**nsulator
FU	**F**unctional **U**nit
GRAAL	**G**lobal **R**eliability **A**rchitecture **A**pproach for **L**ogic
HFM	**H**igh **F**requency **M**ode

HPM	**H**ardware **H**erformance **M**onitor
IC	**I**ntegrated **C**hip
ILA	**I**ntegrated **L**ogic **A**nalyzer
ILEP	**I**nstruction **L**evel **P**ower **E**stimation
ILP	**I**nstruction **L**evel **P**arallelism
IoT	**I**nternet **o**f **T**hings
ISA	**I**nstruction **S**et **A**rchitecture
LFBFF	**L**inear **F**eed **B**ack **F**lip **F**lop
LFM	**L**ow **F**requency **M**ode
LFSR	**L**inear **F**eedbacl **S**hift **R**egister
LSB	**L**east **S**ignificant **B**it
LUT	**L**ook-**U**p **T**able
LVDS	**L**ow-**V**oltage **D**ifferential **S**ignaling
LVT	**L**ow **VT**
MMCM	**M**ixed **M**ode **C**lock **M**anager
MSB	**M**ost **S**ignificant **B**it
MSFF	**M**aster **S**lave **F**lip **F**lop
NOP	**N**o **OP**eration
NTR	**N**ear-**T**hreshold **R**egion
PE	**P**rocessing **E**lement
POFF	**P**oint **o**f **F**irst **F**ailure
PUM	**P**ath **U**nder **M**onitoring
RBB	**R**everse **B**ody **B**iasing
RCA	**R**ipple **C**arry **A**dder
RTL	**R**esistor- **T**ransistor **L**ogic
RVT	**R**egular **VT**
SET	**S**ingle **E**vent **T**ransients
SEU	**S**ingle **E**vent **U**pset
SNR	**S**ignal to **N**oise **R**atio
SNW	**S**ingle **N-W**ell
SOI	**S**ilicon **o**n **I**nsulator
SPICE	**S**imulation **P**rogram with **I**ntegrated **C**ircuit **E**mphasis
SPW	**S**ingle **P-W**ell

STA	**S**tatic **T**iming **A**nalysis
STR	**S**ub-**T**hreshold **R**egion
SW	**S**ingle **W**ell
TFD	**T**iming **F**ault **D**etector
TMR	**T**riple **M**odular **R**edundancy
UTBB	**U**ltra-**T**hin **B**ody and **B**ox
VEX ISA	**VEX** **I**nstruction **S**et **A**rchitecture
VLIW	**V**ery **L**arge **I**nstruction **W**ord
VOS	**V**oltage **O**ver-**S**caling

Contents

List of Figures

List of Tables

Chapter 1

Résumé étendu

1.1 Introduction

L'efficacité énergétique et la gestion des erreurs sont les deux challenges majeurs dans la tendance grandissante de l'internet des objets. Une pléthore de techniques, appelées ultra faible consommation, ont été explorées par le passé afin d'améliorer l'efficacité énergétique des systèmes numériques. Ces techniques peuvent généralement se classer au niveau technologique, au niveau architecture, ou au niveau conception. Au niveau technologique, différentes technologies CMOS et SOI, par exemple le FDSOI de STMicroelectronics ou le FinFET, bénéficient de manière inhérente d'un faible courant de fuite et de composants à faible consommation pour la conception numérique. De manière similaire, de nombreuses solutions architecturales à faible consommation sont utilisées pour augmenter le gain énergétique et le débit des architectures numériques, comme par exemple le pipeline, l'entrelacement, ou encore les architectures asynchrones. Enfin, au niveau conception, des techniques telles que la désactivation de l'horloge ou de l'alimentation, et l'adaptation de la fréquence et de la tension, permettent également de réduire la consommation énergétique.

Au fil des ans, la gestion de la tension d'alimentation a été utilisée comme technique principale pour augmenter l'efficacité énergétique des systèmes numériques. En effet, diminuer la tension d'alimentation permet de réduire de manière quadratique la

consommation énergétique du système selon l'équation suivante :

$$E_{total} = V_{dd}^2.C_{load} \tag{1.1}$$

Dans la littérature, des variantes de la technique de gestion de tension ont été proposées pour améliorer l'efficacité énergétique des systèmes numériques [1], [2]. L'ajustement dynamique de la fréquence et de la tension est devenu la principale technique pour réduire l'énergie en diminuant les performances d'une marge acceptable. D'un autre côté, la diminution de la finesse de gravure des circuits intégrés augmente les risques dus aux effets de variabilité. Les effets de variabilité causent des différences de fonctionnement dans les circuits liés aux variations de fabrication, d'alimentation et de température des circuits (Process, Voltage and Temperature – PVT). Les effets de la diminution de tension et de fréquence couplés aux problèmes de variabilité rendent les systèmes pipelinés plus vulnérables aux erreurs temporelles [3]. Des mécanismes sophistiqués de détection et de correction des erreurs sont requis pour gérer ces erreurs. Dans la littérature, de nombreuses méthodes sont proposées, comme celles basées sur la méthode du double échantillonnage [4].

Les principales limitations des méthodes actuelles de double échantillonnage sont : une fenêtre de spéculation fixe (déphasage entre les deux horloges), le coût en ressources dû aux besoins de buffers et de détecteur de transition, et enfin la complexité des méthodes, comme par exemple la génération et la propagation des bulles (technique Bubble Razor). De même pour les techniques d'emprunt de temps (time borrowing) qui nécessitent des flots d'outil de conception spécifiques afin de resynchroniser l'architecture. Les méthodes de redondance matériel, telles que la duplication et la triplication sont très consommatrices de ressources, ce qui les rend plus intéressantes pour des architectures reconfigurables, du type FPGA, disposant de nombreuses ressources inutilisées. Les applications basées sur le calcul probabiliste peuvent tolérer une marge d'erreur sans compromettre la qualité de la sortie générée. Ainsi, il y a un besoin d'unifier les approches afin de gérer les erreurs en fonction des applications et du besoin utilisateur. La méthode requise doit être capable de détecter et de corriger des erreurs tout en permettant à l'utilisateur de choisir un compromis entre la précision des calculs et l'efficacité énergétique. Finalement, il y a un besoin d'associer les différentes techniques de très faible consommation telles que l'ajustement dynamique de la tension et l'alimentation proche du seuil de basculement. Dans cette thèse, nous proposons

un *framework* pour estimer l'énergie et gérer les erreurs dans le contexte d'architecture alimentée proche du seuil pour les applications résistantes aux erreurs.

1.2 Contributions

Les contributions de la thèse sont les suivantes.

1. Une spéculation temporelle dynamique basée sur l'ajustement dynamique de la fréquence et la correction d'erreur pour un chemin de données pipeliné est proposée. Cette méthode permet d'identifier dynamiquement la possibilité d'augmenter la fréquence du système, et permet également de répondre aux effets de variabilité en la diminuant. Cela est effectué sur la base d'une mesure de la marge temporelle entre les signaux de sortie des registres et l'horloge via le principe du registre déphasé (shadow register). S'il y a des erreurs temporelles dues à des effets de variabilité, un mécanisme de correction embarqué permet de gérer les erreurs sans ré-exécuter les instructions. Contrairement aux méthodes de type Razor, des buffers additionnels ne sont pas requis pour les chemins logiques les plus court. L'efficacité de cette méthode est mise en œuvre sur le FPGA Virtex 7 de Xilinx. Les résultats montrent qu'une augmentation de la fréquence de 71% peut être atteinte avec un surcoût matériel limité.

2. La précision et l'efficacité énergétique de différents additionneurs et multiplieurs sont caractérisées pour de nombreuses configurations de fonctionnement. Un Framework est proposé pour modéliser statistiquement le comportement des opérateurs sujets à un ajustement agressif de la tension. Les modèles générés peuvent être utilisés dans le calcul approximatif au niveau algorithmique. Les résultats de simulation montrent une réduction énergétique pouvant atteindre 89% en contrepartie d'un taux d'erreur binaire de 20%.

3. Un Framework est proposé pour estimer la consommation énergétique au niveau fonctionnel d'une architecture de processeur à mot d'instruction très long (Very Long Instruction Word – VLIW). Un flot d'outil a été développé pour analyser des programmes de référence (benchmarks) et rapidement estimer des métriques d'énergie et d'erreur pour différents paramètres de fonctionnement. Une rapide estimation est possible en utilisant un compilateur C et VEX. Cette estimation peut

être utilisée pour configurer dynamiquement les paramètres de fonctionnement afin d'atteindre un meilleur compromis entre l'énergie et la précision de calcul. Une validation de notre Framework est effectuée sur le processeur VLIW p-Vex. L'unité d'exécution du processeur est caractérisée pour différentes configurations et la consommation d'énergie au niveau fonctionnelle est estimée avec le Framework proposé.

1.3 Gestion des erreurs temporelles

Comme illustré en Fig. 1.1, les architectures numériques sont contraintes temporellement avec une marge positive pour la synchronisation en visant un compromis entre performance et effet des variabilités. Une architecture subissant des effets de variabilité peut voir une augmentation dans les délais de propagation de ses portes résultant en erreurs temporelles (le signal atteint sa valeur finale après le déclenchement du registre de sortie). De telles erreurs peuvent être corrigées mais au prix d'un surcoût matériel [4]. La méthode Razor est une technique bien connue pour la détection et correction des erreurs temporelles dans les systèmes numériques, elle s'inspire de la méthode du double échantillonnage [4]. Dans les architectures utilisant le double échantillonnage, un registre supplémentaire, appelé registre déphasé (shadow register) est utilisé en plus du registre de sortie afin de capturer la valeur de sortie à des instants différents. Le registre déphasé utilise une horloge déphasée (shadow clock), par rapport à l'horloge principale. L'écart temporel entre le front montant de l'horloge principale et de l'horloge déphasée est appelé *fenêtre de spéculation* (ϕ). La fenêtre de spéculation est utilisée pour comparer la valeur de sortie du registre principal et du registre déphasée afin de déterminer la validité de la sortie.

Il y a un compromis à effectuer entre la taille de la fenêtre de spéculation et le taux de couverture de faute. Pour une fenêtre de spéculation large, plus d'erreurs peuvent être détectées et corrigées, mais cela nécessite plus d'insertion de buffer pour les chemin logiques ayant un délais de propagation plus rapide que (ϕ) afin d'éviter des fausses détections. L'effet est inverse pour une fenêtre de spéculation courte. Les architectures avec une fenêtre de spéculation fixe optimisée pour certaines conditions de fonctionnement peuvent souffrir d'une augmentation du taux d'erreur sous certaines variabilités, comme par exemple les variations de température. Afin de répondre à ces

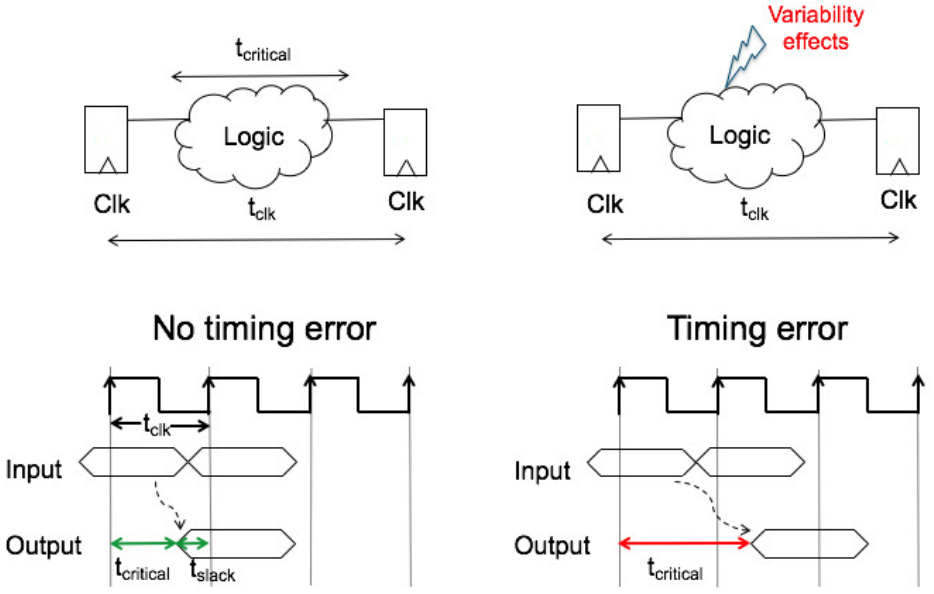


FIGURE 1.1: Illustration des erreurs temporelles dues à des effets de variabilité

limitations, une technique de double échantillonnage avec une fenêtre de spéculation dynamique est requise pour adapter (ϕ) en fonction des effets de variabilité.

1.3.1 Gestion des erreurs avec une fenêtre de spéculation dynamique

Dans cette thèse, nous proposons une méthode combinant une fenêtre de spéculation dynamique avec la technique du double échantillonnage pour la détection et correction des erreurs et pour augmenter ou diminuer dynamiquement la fréquence d'une architecture en fonction des effets de variabilité. La Fig. 1.2 illustre la mise en œuvre de la fenêtre de spéculation dynamique. Dans une architecture pipelinée, les chemins logiques sont contraints temporellement deux registres. L'architecture de la Fig. 1.2 est composée de n chemins logiques, notés $Path_i$, et contraints entre la sortie d'un registre R_i et de l'entrée d'un registre Q_i. Pour une analyse plus simple, nous considérons le chemin $Path_1$ parmi les n chemin logiques. Le chemin $Path_1$ est contraint entre la sortie de R_1 et l'entrée de Q_1. Contrairement aux techniques utilisant un seul registre déphasé [5], [6], et [7], deux registres déphasés S_1 et T_1 sont ajoutés pour capturer la sortie de $Path_1$. T_1 est utilisé pour mesurer la marge temporelle disponible afin d'augmenter la fréquence de l'architecture. S_1 est utilisé pour capturer la donnée valide de sortie dans le cas ou le délais de propagation du $Path_1$ ne respecte pas les contraintes temporelle de l'horloge principale M_clk du à des effets de variabilité. De manière similaire aux architectures

à double échantillonnage, nous supposons que le registre S_1 capture toujours la donnée valide. Dans l'architecture proposée, les trois registres S_1, Q_1 et T_1 utilisent la même fréquence d'horloge mais avec des phases différentes pour capturer la donnée de sortie. Le registre de sortie Q_1 capture la sortie au front montant de M_clk, alors que le registre déphasé S_1 capture la sortie au front descendant de M_clk. Le registre déphasé T_1 capture la donnée au front montant de l'horloge déphasée S_clk générée par le générateur de phase (phase generator). Deux portes XOR comparent la donnée capturée par le registre principal Q_1 avec S_1 et T_1. Les sorties des XOR, correspondant à des signaux d'erreurs, sont capturées dans des registres appelés respectivement E_1 et e_1.

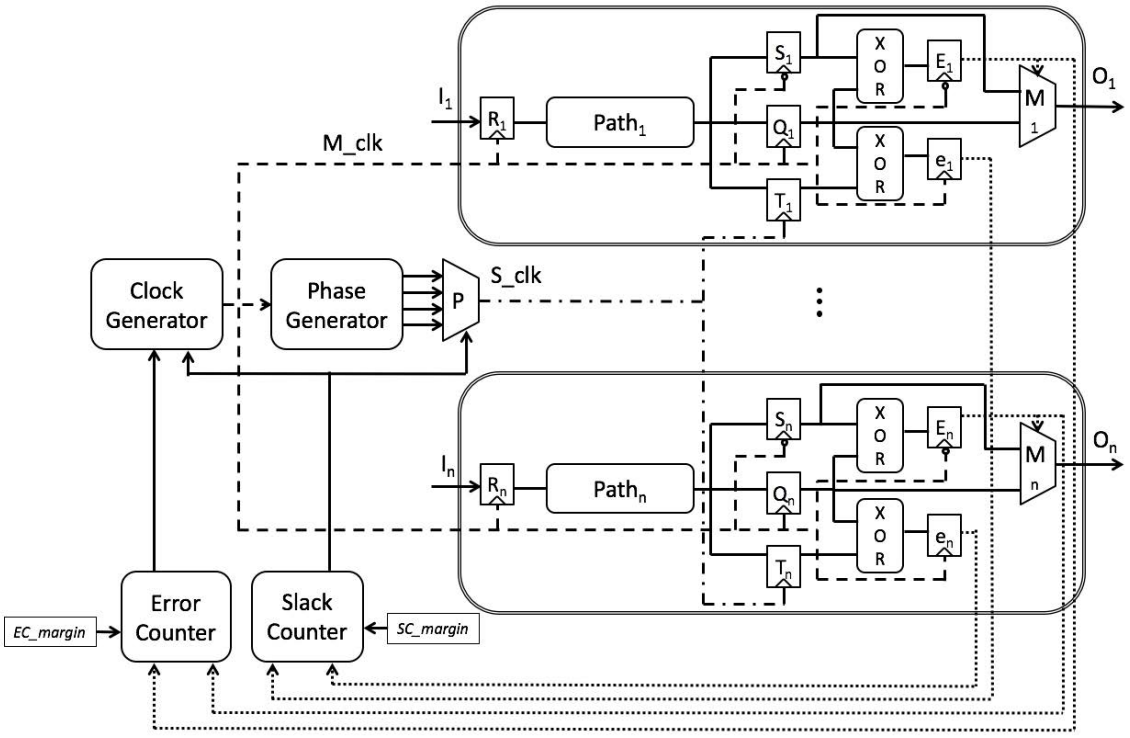


FIGURE 1.2: Architecture proposée pour la mise en œuvre de la fenêtre de spéculation dynamique

Comme expliqué pour le chemin $Path_1$, il est possible de rajouter les registres déphasés supplémentaires pour n'importe quel nombre de chemin au sein d'une architecture. Le nombre de chemin devant utiliser la spéculation dynamique est défini par l'utilisateur en se basant sur les informations temporelles via le rapport de synthèse de l'architecture. En général, cela s'exprime en pourcentage de la marge critique de temps (Criticical Delay Margin – CDM). Tous les chemins avec un délais de propagation plus grand que t_{CDM} doivent utiliser la spéculation dynamique, et t_{CDM} s'exprime par

l'équation suivante

$$t_{CDM} = t_{critical} - (t_{critical}.CDM) \tag{1.2}$$

dans laquelle $t_{critical}$ est la valeur du chemin critique. Dans la Fig. 1.2, CDM = 100%. Cela signifie que les n chemins de l'architecture disposent de registre pour la spéculation dynamique. Pour des applications tels que les filtres à réponse impulsionnelle finie (Finite Impulse Response – FIR), dans lesquelles presque tous les chemins ont le même délai de propagation, un CDM de 100% est requis. Au contraire, pour des applications avec des chemins ayant des délais de propagation plus distribués, le CDM peut varier de 10 à 20%.

1.3.2 Boucle de retour pour augmentation ou diminution de la fréquence

La Fig. 1.2 montre la boucle de retour de la méthode proposée. Les signaux d'erreur E_i et e_i des chemins monitorés sont connectés respectivement au compteur d'erreur (Error Counter) et au compteur de marge (Slack Counter). Pour différentes fenêtres de spéculation ϕ_i, *Error Counter* et *Slack Counter* comptent les variations dans les chemins concernés.

- Le Slack Counter détermine le nombre d'erreur liée à l'augmentation de la fréquence ou à la diminution de la tension d'alimentation.
- Le Error Counter détermine le nombre d'erreur temporelle liée à des effets de variabilité.

Pour une architecture, utilisant une horloge configurée au maximum de la fréquence possible et sans effet liée à la variabilité, *Error Counter* et *Slack Counter* sont à 0. Cela indique qu'il n'y a pas d'erreur dans l'architecture. Afin d'augmenter les performances, l'architecture peut subir une augmentation de la fréquence. Le *générateur de phase* dans la boucle de retour est utilisé pour générer différentes phases pour l'horloge S_clk. Cette horloge sert à déterminer s'il y a une marge sûre afin d'augmenter la fréquence (sans générer d'erreur) en mesurant la marge temporelle disponible. Dans la méthode proposée, un registre dédié est utilisé pour mesurer la marge disponible. Lorsque la différence de phase ϕ entre M_clk et S_clk augmente, les chemins critiques de l'architecture ont tendance à devenir fautifs indiquant la limite pour augmenter la

fréquence de manière sûre. Le registre déterminant la marge temporelle T_i détecte cette erreur temporelle et augmente le *Slack Counter* du nombre de chemin défaillant. Dans ce cas, *Error Counter* reste à zéro tant que le pipeline ne subit pas d'erreur. En fonction de la valeur du *Slack Counter*, une augmentation de la fréquence ou une diminution de la tension est gérée par la boucle de retour.

Certaines applications tolérantes aux erreurs peuvent tolérer un certain nombre d'erreurs. Dans la boucle de retour, *SC_margin* est une marge d'erreur tolérable définie par l'utilisateur. Pour les applications pouvant tolérer les erreurs temporelles, *SC_margin* peut être configurée pour permettre un réglage de fréquence ou de tension afin d'augmenter respectivement les performances ou l'efficacité énergétique, même en présence d'erreurs. Dans le cas où des effets de variabilité se manifestent, des chemins logiques vont subir des erreurs, ce qui va incrémenter la valeur du *Error Counter* en fonction du nombre de chemin fautif. Cela déclenche le mécanisme de correction d'erreur consistant à fournir à l'étage suivant du pipeline la valeur stockée dans le registre déphasé S_i au lieu de celle stockée dans le registre principal. En fonction de la valeur du *Error Counter*, la boucle de retour peut décider de réduire la fréquence d'horloge ou d'augmenter la tension d'alimentation afin de réduire les erreurs temporelles. Comme mentionné précédemment, *Error Counter* permet de configurer une marge d'erreur tolérable pour les applications résistantes aux erreurs. Tant que cette marge n'est pas dépassée, l'architecture ne subit pas de réajustement sur la fréquence ou la tension d'alimentation. De même, *EC_margin* configure la limite pour les erreurs temporelles dues à des effets de variabilité qui peuvent être tolérées sans compromettre la performance ou l'efficacité énergétique.

SC_margin et *EC_margin* sont définis comme pourcentage du CDM. Pour des applications nécessitant une grande précision, *SC_margin* et *EC_margin* sont mis à 0%. Cela force la boucle de retour à augmenter la fréquence de manière sûre (sans générer d'erreur) et ne tolère aucune erreur temporelle dû à des problèmes de variabilité. Les marges tolérables d'erreur pour l'*Error Counter* et le *Slack Counter* sont déterminées à partir des rapports de synthèse et de placement de l'architecture.

1.3.3 Validation du concept sur FPGA Virtex de Xilinx

La fenêtre de spéculation dynamique de la méthode du double échantillonage est implémentée dans une carte de développement Xilinx Virtex VC707 tel qu'illustré en Fig. 1.3. Dans

cette expérimentation, nous avons démontré la gestion dynamique de la fréquence dans un FPGA Virtex 7 via la méthode de spéculation dynamique. Cette méthode peut être étendue à l'adaptation de la tension sur les plate-formes qui sont compatible avec la gestion de cette dernière. La chaîne d'outil Vivado est utilisée pour synthétiser et implémenter les architectures de démonstration dans le FPGA. Un vecteur de test (test-bench) aléatoire est créé pour chaque architecture via un générateur de vecteur LFSR de 80 bits. L'oscillateur LVDS programmable par IIC est utilisé pour générer la fréquence d'horloge pour le générateur de vecteur de test et l'architecture de démonstration. Comme illustré en Fig. 1.3, les signaux de contrôles du compteur d'erreurs (*Error Counter*) et du compteur de marge (*Slack Counter*) déterminent la fréquence de sortie de l'oscillateur LVDS. Les différentes phases des horloges générées sont obtenues via le module de gestion d'horloge intégré au FPGA (*Mixed-Mode Clock Manager : MMCM*). La température du coeur est mesurée via le moniteur XADC, et la température est réglée en changeant la vitesse de rotation du ventilateur.

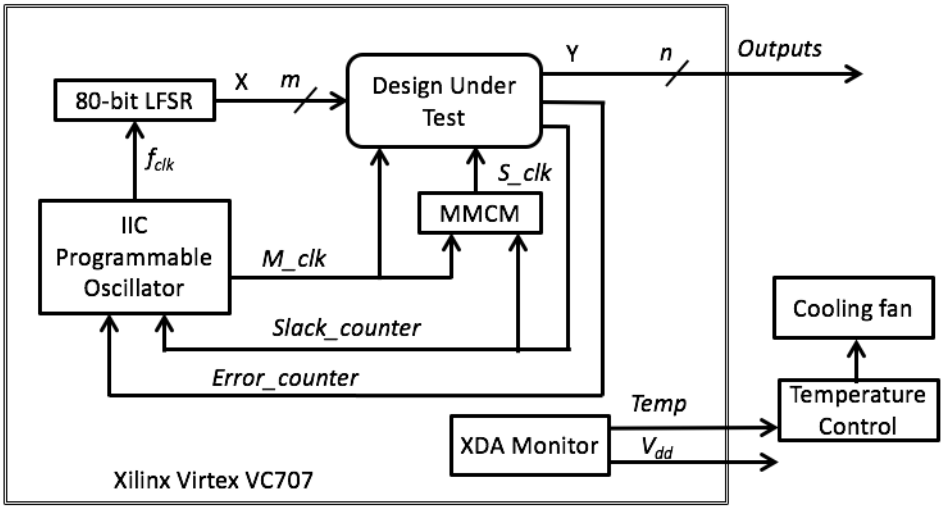


FIGURE 1.3: Configuration de l'expérimentation de l'overclocking basé sur la spéculation dynamique

Pour une gestion dynamique de la fréquence sans erreur, la marge *Slack Counter* est configurée à zéro. A la température ambiante de 28°C, lorsque *Error Counter* et *Slack Counter* sont égaux à zéro, la fenêtre de spéculation ϕ est augmentée à chaque cycle d'horloge de 0° à 180° par intervalle de 25° jusqu'à ce que le *Slack Counter* enregistre une erreur. Au delà de 180°, et jusqu'à 360°, la fenêtre de spéculation se répète dû à la nature périodique de l'horloge. Lorsque le *Slack Counter* enregistre une

erreur, cela indique que la marge maximale a été définie. Le système est mis en pause et est overlocké de manière sûre en respectant la marge disponible. Lorsque la fenêtre de spéculation ϕ est balayée, le pipeline fonctionne normalement dès lors que les sorties du registre Q_i et du registre déphasé S_i capturent la donnée valide. Concernant la correction des erreurs dues aux effets de variabilités, si *Erreur Counter* détecte que plus de 2% des chemins monitorés ne respectent pas les contraintes de temps, le système est mis en pause et la fréquence du système est diminuée par multiple d'un pas de fréquence δ afin de réduire ces erreurs (dans notre plateforme de test, le pas de fréquence est $\delta = 5MHz$). Étant donnée que corriger plus d'erreurs réduit la bande passante de l'architecture, diminuer la fréquence est une mesure nécessaire. Le multiplexeur M_i, tel qu'illustré dans la Fig. 1.2, corrige l'erreur en connectant la sortie du registre déphasé S_i au prochain étage du pipeline. Dès que la température diminue, la fréquence de fonctionnement de l'architecture est de nouveau augmentée en fonction de la marge mesurée tel que décrit précédemment.

Pour mettre en avant l'adaptation dynamique de la fréquence et la capacité à détecter et corriger les erreurs de la méthode proposée, nous avons implémenté un ensemble d'architectures de test avec la méthode Razor [5] et la méthode proposée basé sur la fenêtre de spéculation dynamique. Les architectures de test utilisées dans ces expérimentations sont des éléments de chemin de données tels que les filtres FIR, et de nombreuses version d'architectures 32bits d'additionneurs et de multiplieurs. Après implémentation, les rapports générés permettent de définir les chemins critiques de chaque architecture.

Pour une architecture à température ambiante d'environ 25°C, la fenêtre de spéculation ϕ est augmentée à chaque cycle d'horloge. Si le *Slack Counter* est égale à zéro à la plus large fenêtre de spéculation, alors la fréquence d'horloge peut être augmentée de 40% étant donnée qu'à 180°, ϕ correspond à la moitié de la période d'horloge. Cela implique que tous les chemins critiques de l'architectures ont une marge positive égale à la moitié de la période d'horloge. L'augmentation de la fréquence est effectuée en stoppant l'architecture et en augmentant la fréquence d'horloge à partir de l'oscillateur LVDS. Après ce changement de fréquence, l'étape de balayage de phase recommence à 0° jusqu'à ce que *Slack Counter* enregistre une erreur. Alors que l'architecture fonctionne à une fréquence augmentée F_{max} à 28°C, la température de la puce est augmentée en faisant varier la fréquence de rotation du ventilateur de refroidissement. Due

TABLE 1.1: F_{max} estimée et F_{max} augmentée sans faute pour des architectures à 28°C

Architecture	F_{max} estimée (MHz)	F_{max} sans faute à 28°C (MHz)	Marge d'augmentation de F_{max} à 28°C
8-tap 8-bit FIR	167	260	55%
Multiplieur non-signé 32-bit	76	130	71%
Multiplieur signé 32-bit Wallace Tree	80	135	69%
Additionneur 32-bit Kogge-Stone	130	215	64%
Additionneur 32-bit Brent-Kung	135	216	60%

TABLE 1.2: Impact de la température sur l'augmentation de la fréquence d'architectures à 50°C

Architectures	% des chemins défaillant à 50°C pour F_{max} à 28°C	F_{max} sans faute à 50°C (MHz)	Marge d'augmentation sans faute à 50°C
Filtre 8-tap 8-bit FIR	12	180	8%
Miltiplieur non-signé 32-bit	22	85	12%
Miltiplieur signé 32-bit Wallace Tree	26	85	7%
Additionneur 32-bit Kogge-Stone	13	180	38%
Additionneur 32-bit Brent-Kung	21	180	33%

à l'augmentation de la température, les chemins critiques de l'architecture commencent à défaillir. Lorsque *Error Counter* détecte plus que 2% des chemins monitorés sont fautifs, l'architecture est mis en pause et la fréquence d'horloge est réduite par multiple de 5MHZ jusqu'à ce que la marge *Error Counter* est sous les 2%.

La Table 1.1 montre la F_{max} estimée pour chaque architecture de test ainsi que la F_{max} maximale pouvant être atteinte sans erreur par la méthode de détection d'erreur proposée à une température ambiante de 28°C. Par exemple, pour l'architecture du filtre FIR, Vivado a estimé F_{max} à 167MHz. Cependant, cette architecture peut avoir sa fréquence augmentée, à température ambiante, de plus de 55% jusqu'à 260MHz et sans erreur, et jusqu'à 71% pour les autres architectures de test. A la fréquence maximale sans erreur, la température est augmentée de 28°C à 50°C, à laquelle *Error Counter* détecte que 12% des chemins de données sont défaillants. Étant donnée que le nombre de chemins fautifs est supérieur à la marge configurée à 2%, la fréquence d'horloge de l'oscillateur LVDS est diminuée jusqu'à 180MHz. Cette diminution de la fréquence permet de maintenir le nombre de chemin fautif à moins de 2%. La Table 1.2 liste l'impact de la température et la marge d'augmentation de la fréquence sans erreur pour

une température de 50°C pour toutes les architectures de test. Le pourcentage de chemin monitoré qui sont défaillant à 50°C montrent l'impact de la température lorsque l'on overclock l'architecture.

Lorsque la température redescend à 28°C, la marge temporelle disponible est mesurée en temps réel et la fréquence d'horloge est augmentée afin d'atteindre des performances maximales. Ces résultats démontrent la gestion dynamique de la fréquence d'horloge, ainsi que la capacité de détection et de correction des erreurs de la méthode proposée avec un surcoût matériel limité.

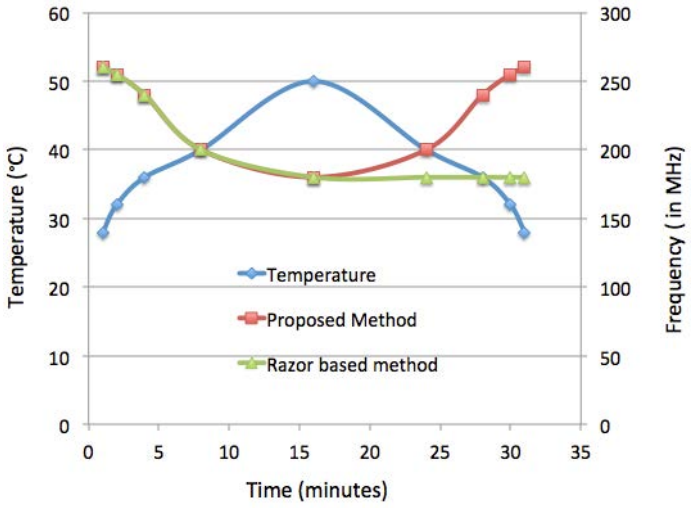


FIGURE 1.4: Comparaison entre une boucle de retour utilisant la méthode Razor et la technique proposée

La Fig. 1.4 montre la comparaison entre la fenêtre de spéculation dynamique et la méthode Razor pour un filtre FIR 8-bit 8-tap au sein d'un FPGA Virtex 7. La température du circuit est modifiée en faisant varier la vitesse de rotation du ventilateur. Comme montré en Fig. 1.4, la température du circuit est augmentée à partir de la température ambiante de 28°C jusqu'à 50°C, puis diminuée à nouveau jusqu'à 28°C. Lorsque la température commence à augmenter, les chemins critiques de l'architecture commencent à défaillir, ce qui diminue la fréquence d'horloge via la méthode proposée et celle utilisant Razor afin de rester sous la limite des 2% de chemin fautif. Initiallement, à température ambiante, l'architecture fonctionne à 260MHz. A la température de 50°C la fréquence est diminuée à 180MHz. Lorsque la température retourne à celle d'origine, la méthode proposée est capable de mesurer la marge disponible et d'augmenter la fréquence d'horloge. Au contraire, la méthode Razor ne modifie pas la fréquence. En

effet, cette méthode n'intègre pas de mesure de marge temporelle disponible. Cela donne un avantage certain à la méthode proposée.

1.4 Ajustement agressif de l'alimentation pour les applications résistantes aux erreurs

Comme mentionné dans la section précédente, des applications basées sur des algorithmes statistiques et probabilistes, par exemple pour le calcul vidéo, la reconnaissance d'image, l'exploitation large échelle de données et de texte, ont une capacité inhérente à tolérer les incertitudes liées au matériel. De telles applications peuvent fonctionner avec des erreurs, évitant ainsi le besoin de matériel supplémentaire pour détecter et corriger les erreurs. Ainsi, les applications résilientes aux erreurs sont une opportunité pour la conception d'architecture approximative pour satisfaire les besoins de calcul avec une meilleure efficacité énergétique. Dans les applications résilientes aux erreurs, le calcul approximatif peut être introduit à différents niveaux et à différentes granularités.

Dans ce travail, nous utilisons une diminution agressive de la tension d'alimentation (voltage overscaling) dans les opérateurs arithmétiques ce qui cause délibérément des approximations au niveau du circuit. En réduisant la tension d'alimentation vers le niveau du seuil des transistors, des erreurs dues à des délais de propagation peuvent survenir. Le comportement des opérateurs arithmétiques fonctionnant à la région proche, ou sous le seuil, est différent par rapport à une tension d'alimentation élevée (super threshold region). Dans le cas d'un additionneur à propagation de retenu (Ripple Carry Adder – RCA), lorsque la tension d'alimentation diminue, le comportement attendue est une défaillance des chemins critiques, en commençant par le plus long jusqu'au plus court, en fonction de la diminution de la tension d'alimentation. La Fig. 1.5 montre l'effet de la diminution de tension pour un RCA 8 bits. Lorsque la tension d'alimentation est diminuée de 1V à 0,8V, les bits de poids forts commencent à être erronés. Lorsque la tension continue à diminuer, de 0,7V à 0,6V, le plus grand taux d'erreur binaire (Bit Error Rate – BER) apparaît sur les bits de poids moyens plutôt que sur les bits de poids forts. Pour une alimentation de 0,5V, presque tous les bits atteignent 50% ou plus de BER. Ce comportement impose des limitations dans la modélisation des opérateurs arithmétiques dans la région proche/sous le seuil en utilisant des modèles

classiques. Le comportement des opérateurs arithmétiques pour le réglage agressif de l'alimentation peut être caractérisé avec des simulations SPICE. Cependant le simulateur SPICE nécessite un temps long (4 jours avec un CPU 8 cœurs) pour simuler de manière exhaustive un ensemble de vecteurs d'entrées nécessaire pour caractériser les opérateurs arithmétiques.

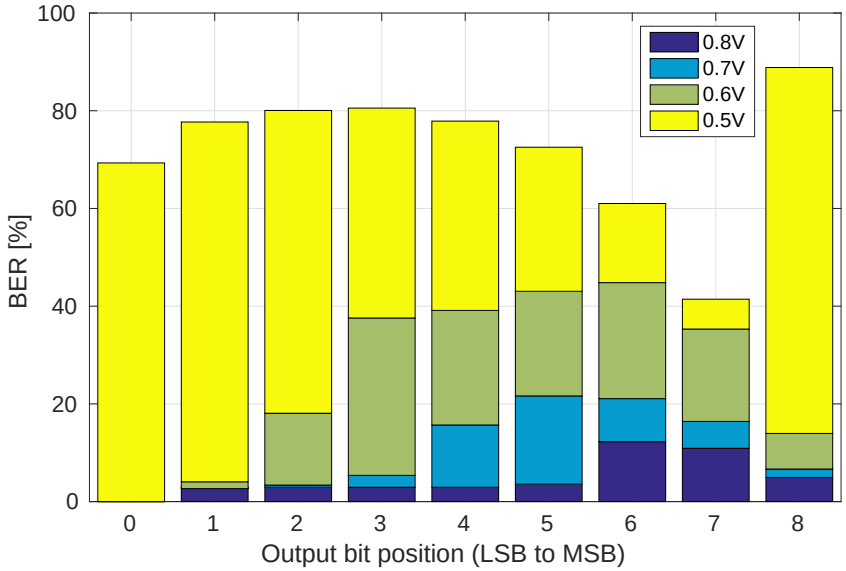


FIGURE 1.5: Distribution du BER sur les bits de sortie d'un RCA 8 bits sous différentes tensions d'alimentation

1.4.1 Modélisation d'opérateur arithmétique avec gestion agressive de la tension d'alimentation

Comme indiqué précédemment, il y a un besoin de développer de nouveaux modèles pour simuler le comportement d'opérateur arithmétique fautif au niveau fonctionnel. Dans cette section, nous proposons une nouvelle technique de modélisation qui est scalable pour des opérateurs de grande taille et compatible avec de nombreuses configurations arithmétiques. Le modèle proposé est précis et permet des simulations rapides au niveau algorithme en imitant l'opérateur fautif via des paramètres statistiques.

Comme la réduction d'alimentation agressive génère en priorité des fautes dans le chemin de données combinatoires le plus long, il y a un lien clair entre le chemin de propagation de retenue pour un additionneur donné et l'erreur résultante de l'addition. La Fig. 1.6 illustre le besoin de relation entre l'opérateur matériel contrôlé par la triade de configuration (constituée de la tension d'alimentation V_{dd}, la période d'horloge T_{clk} et

la tension de polarisation inverse V_{bb}) et le modèle statistique contrôlé par des paramètres statistiques Pi. La connaissance des valeurs d'entrées fournie les informations nécessaires concernant la plus longue chaîne de propagation de retenu, ainsi les valeurs d'entrées sont utilisées pour générer les paramètres statistiques qui contrôlent le modèle équivalent. Ces paramètres statistiques sont obtenus à travers un processus d'optimisation hors ligne qui minimise les différences entre les sorties de l'opérateur et de son modèle statistique équivalent, avec le respect de certaines métriques. Dans ce travail, nous avons utilisé trois métriques pour calibrer l'efficacité de notre modèle statistique : l'erreur quadratique moyenne (Mean Square Error – MSE), la distance de Hamming (Hamming distance) et la distance de Hamming pondérée (Weighted Hamming distance).

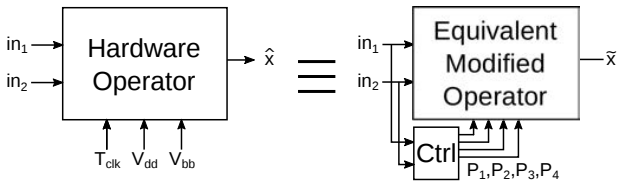


FIGURE 1.6: Équivalence fonctionnelle de l'additionneur matériel

1.4.2 Validation du concept: modélisation d'additionneurs

Dans le reste de cette section, une validation du concept est effectuée en appliquant une gestion agressive de l'alimentation (Voltage Overscaling – VOS) pour différentes configurations d'additionneurs. Tous les additionneurs subissent le VOS et sont caractérisés par le flot décrit par la Fig. 1.7. La Fig. 1.8 montre le flot de conception pour modéliser les opérateurs avec VOS. Comme illustré en Fig. 1.6, un modèle rudimentaire de l'opérateur matériel est créé avec les vecteurs d'entrées et les paramètres statistiques. Pour un vecteur d'entrée donné, les sorties du modèle et de l'opérateur matériel sont comparées en se basant sur les métriques définies précédemment. Le comparateur illustré en Fig. 1.8 génère le rapport signal sur bruit (Signal to Noise Ration – SNR) et la distance de Hamming pour déterminer la qualité du modèle en fonction des métriques. Le SNR et la distance de Hamming sont renvoyés dans un algorithme d'optimisation afin de configurer plus finement le modèle représentant l'opérateur sous VOS. Dans le cas d'additionneurs, seulement un paramètre Pi est utilisé pour le modèle statistique. Il est défini par C_{max}, la longueur de la plus grande chaîne de retenu à propager.

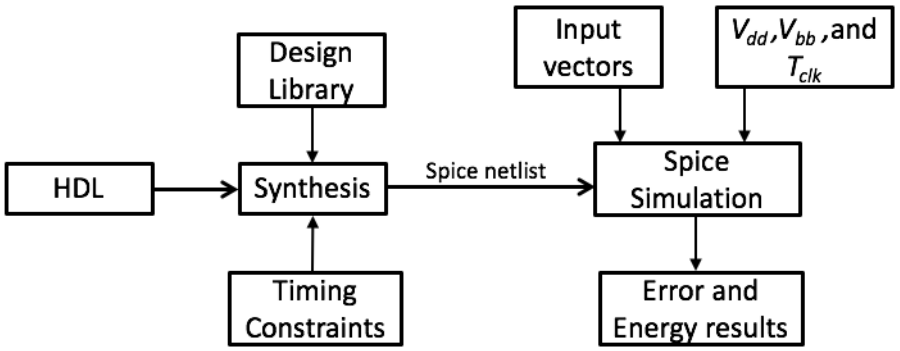


FIGURE 1.7: Flot de caractérisation pour opérateur arithmétiques

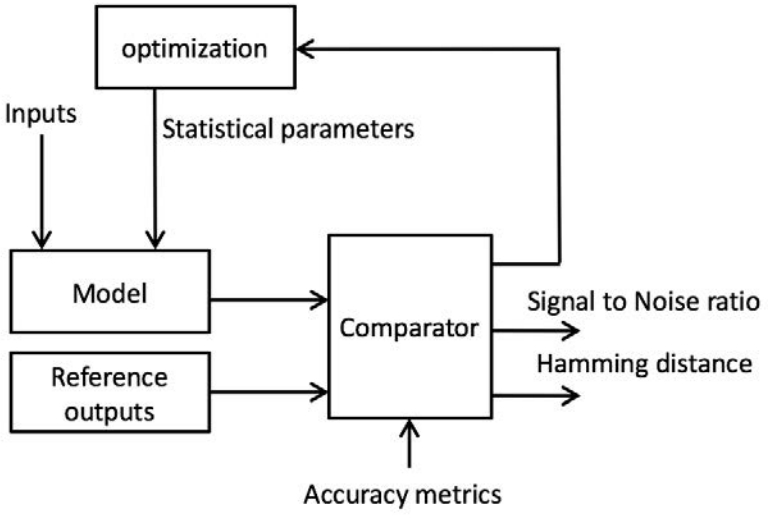


FIGURE 1.8: Flot de conception pour la modélisation des opérateurs sous VOS

Ainsi, en fonction des paramètres de fonctionnement (T_{clk}, V_{dd}, V_{bb}) et d'un couple d'entrée (in1, in2), le but est de trouver Cmax en minimisant la distance entre les sorties de l'opérateur matériel et de son modèle équivalent. Cette distance peut être définie par les métriques déjà présentées. Ainsi, Cmax est donnée par:

$$C_{max}\left(in_1, in_2\right) = \underset{C\in[0,N]}{Argmin} \left\|\hat{x}\left(in_1, in_2\right), \tilde{x}\left(in_1, in_2\right)\right\|$$

où $\|x, y\|$ est la métrique de distance choisie appliquée à x et y. Etant donnée que l'espace d'exploration pour caractériser C_{max}, en considérant toutes les entrées possibles, est potentiellement très élevée, C_{max} est caractérisé uniquement en terme de probabilité d'apparition par une fonction liée à la plus grande chaine de propagation théorique des entrées, et notée $P\left(C_{max} = k | C^{th}_{max} = l\right)$. De cette manière, l'espace d'exploration

constitué de 2^{2N} possibilités est réduit à $(N+1)^2/2$. La Table 1.3 donne les valeurs de probabilité requises pour l'additionneur équivalent modifié afin de produire une sortie.

TABLE 1.3: Table de probabilité de propagation de Carry pour l'additionneur 4-bit équivalent modifié

$C_{max}C^{th}_{max}$	0	1	2	3	4
0	1	$P(0\|1)$	$P(0\|2)$	$P(0\|3)$	$P(0\|4)$
1	0	$P(1\|1)$	$P(1\|2)$	$P(1\|3)$	$P(1\|4)$
2	0	0	$P(2\|2)$	$P(2\|3)$	$P(2\|4)$
3	0	0	0	$P(3\|3)$	$P(3\|4)$
4	0	0	0	0	$P(4\|4)$

L'algorithme d'optimisation utilisé pour construire l'additionneur modifié est montré en Algorithme 1. Lorsque les entrées (in_1, in_2) sont dans le vecteur d'entrée, la sortie de l'additionneur matériel $\hat{x}$ est calculée. Basé sur la paire particulière (in_1, in_2), la chaîne maximale de retenue C^{th}_{max} correspondant aux entrées est déterminée. La sortie $\tilde{x}$, de l'additionneur modifié avec trois paramètres d'entrées (in_1, in_2, C), est calculée. La distance entre $\hat{x}$ et $\tilde{x}$ est calculée en fonction des métriques définies pour différentes itérations en C. Le flot continue pour la totalité des vecteurs d'entrées.

begin
  $P(0 : N_{bit\_adder} | \ 0 : N_{bit\_adder}) := 0$;
  $max\_dist := +\infty$;
  $C_{max\_temp} = 0$;
  for *variable* $in_1, in_2 \in training\_inputs$ **do**
    $\hat{x} := add\_hardware(in_1, in_2)$ $C^{th}_{max} := max\_carry\_chain(in_1, in_2)$;
    for *variable* $C \in C^{th}_{max}$ *down to* 0 **do**
      $\tilde{x} := add\_modified(in_1, in_2, C)$;
      $dist := \|\hat{x}, \tilde{x}\|$;
      if $dist <= max\_dist$ **then**
        $dist\_max := dist$;
        $C_{max\_temp} := C$;
      end
    end
    $P(C_{max\_temp} | C^{th}_{max}) + +$
  end
  $P(: | :) := P(: | :)/size(training\_outputs)$;
end

Algorithm 1: L'algorithme d'optimisation

Après que le processus d'optimisation hors ligne soit effectué, l'additionneur modifié équivalent peut être utilisé pour générer les valeurs de sortie de n'importe quel couple d'entrée in1 et in2. Pour imiter l'opérateur exact sujet à du VOS, l'additionneur équivalent est utilisé de la façon suivante :

1. Extraction de la chaîne de propagation de retenue maximale C_{max}^{th} qui serait produite par l'addition exacte de in_1 et in_2.

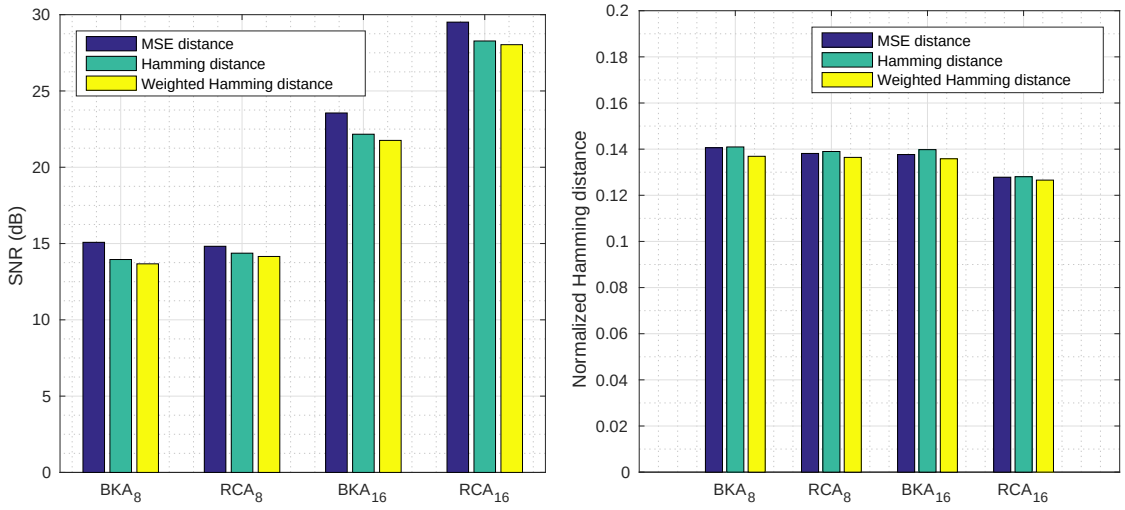
2. Sélection d'un nombre aléatoire qui déterminera la colonne la table de probabilité, et choix de la valeur en fonction du C_{max} qui détermine la ligne.

3. Calcul de la somme de in_1 et de in_2 avec la chaîne de propagation de retenue limitée à C_{max}.

La Fig. 1.9 montre l'erreur d'estimation de la modélisation de différents additionneurs basée sur les métriques définies. Les simulations SPICE sont effectuées pour les 43 triades de configuration avec des vecteurs d'entrée de taille 20K. Les vecteurs d'entrée sont choisis de manière à ce que tous les bits d'entrées aient la même probabilité de propager une retenue. Fig. 1.9a représente le SNR pour les additionneurs 8 et 16 bits RCA et BKA. La métrique MSE montre un SNR plus grand, suivie par la distance de Hamming et la distance de Hamming pondérée. Étant donnée que la MSE et la distance de Hamming pondérée prennent compte du poids des bits, ils montrent un SNR plus grand que la distance de Hamming. La Fig. 1.9b montre les métriques normalisées pour les quatre additionneurs. Sur cette figure, la MSE et la distance de Hamming sont presque identiques, avec un léger avantage pour la distance de Hamming, ce qui est logique étant donnée que cette métrique donne le même impact à toutes les positions de bit. Les deux additionneurs 8 bits n'ont pas le même comportement en terme de distances entre les sorties de l'additionneur matériel et de l'additionneur modifié. D'un autre côté, le 16-bit RCA est meilleur en terme de SNR par rapport au BKA de même taille. Ces résultats démontrent la précision de l'approche proposée pour modéliser le comportement des opérateurs subissant du VOS dans un contexte de calcul approximatif.

1.5 Estimation d'énergie dans un processeur $\rho - VEX$

Dans les sections précédentes, les techniques de modélisation d'opérateurs arithmétiques sous VOS et de gestion d'erreur basée sur la spéculation dynamique ont été proposées. Afin d'étendre le champs d'application de ces méthodes, nous proposons dans cette section d'utiliser un processeur à mot d'instruction très long (Very Long Instruction Word – VLIW) afin d'étudier le bénéfice de la VOS et de la spéculation dynamique.



(A) Signal to Noise Ratio (B) Normalized Hamming Distance

FIGURE 1.9: Erreur d'estimation du modèle pour différents additionneurs et métriques de distance

Un VLIW est une architecture de processeur conçue pour exploiter du parallélisme au niveau instruction. Contrairement aux architectures pipelinées et super scalaires, le VLIW déplace la complexité au niveau du compilateur qui décide quelle instruction doit être exécutée en parallèle, ce qui résulte en une architecture plus simple. L'architecture de processeur VLIW gagne en popularité dans les systèmes embarqués étant donnée sa plus faible consommation (liée à une architecture plus simple) et à sont haut niveau de parallélisme d'instruction. Dans ce travail, le processeur $\rho - VEX$ est utilisé pour nos expérimentations. C'est un processeur VLIW softcore configurable et extensible basé sur l'architecture de jeu d'instruction VEX [8]. La Fig 1.10 montre le schéma d'un cluster $\rho - VEX$. Estimer l'énergie est une étape importante et nécessaire pour utiliser efficacement les processeurs VLIW. Cela permet au concepteur de configurer le parallélisme logiciel et matériel afin d'atteindre une haute efficacité énergétique avec un maximum de performance. Estimer la consommation d'énergie au niveau des instructions permet une compréhension claire de la consommation des différents modules d'un cluster pour différentes configurations. Dans la littérature, des techniques d'estimation de puissance pour microprocesseurs sont proposées à différents niveaux pour évaluer un programme logiciel en terme de puissance consommée. Dans ce travail nous proposons un framework pour estimer la consommation énergétique au niveau instruction de l'unité d'exécution d'un cluster $\rho - VEX$. Nous considérons un contexte alliant alimentation proche du seuil et calcul approximatif. L'estimation énergétique au niveau instruction est réalisée

en deux étapes:

- Caractérisation : l'unité d'exécution d'un cluster p-Vex est caractérisation pour différentes triades de fonctionnement afin d'obtenir la précision au niveau registre.
- Estimation : basée sur la caractérisation de l'unité d'exécution, l'estimation d'énergie est effectuée en utilisant un compilateur C pour différentes instructions et pour différents modes opératoires.

Finalement, nous proposons une procédure de validation pour s'assurer de la précision de la méthode d'estimation énergétique.

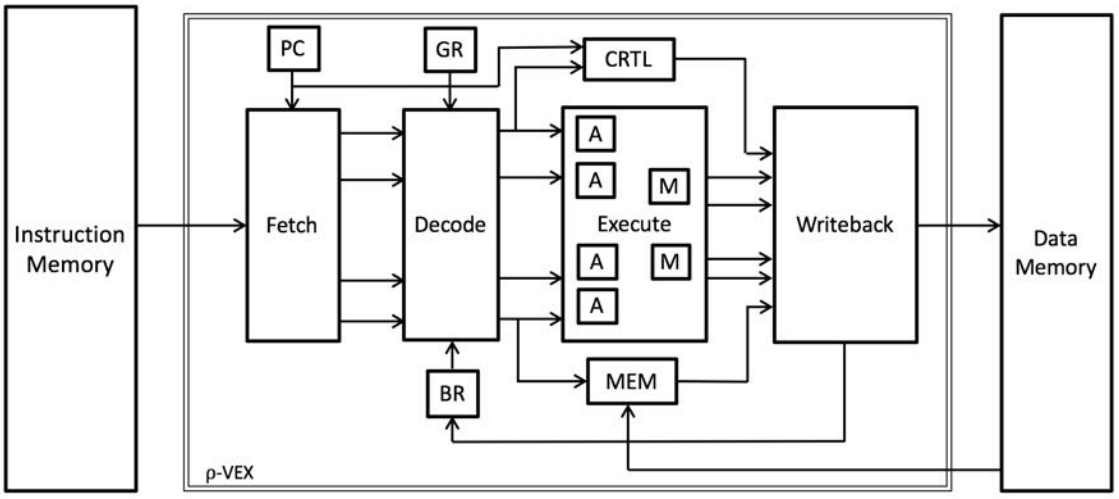


FIGURE 1.10: Schéma de l'organisation d'un $\rho - VEX$ cluster

1.5.1 Caractérisation d'une unité d'exécution d'un cluster $\rho - VEX$

Pour déterminer avec précision l'énergie consommée par tous les modules, tels que les Unités Arithmétiques et Logiques (UAL) et les multiplieurs, l'unité d'exécution d'un cluster p-VEX est caractérisée avec des simulations SPICE. Dans la méthode employée, la description HDL d'une unité d'exécution est synthétisée en utilisant Synposys Design Compiler (version H-2013.03-SP5-2). La bibliothèque de conception FDSOI 28nm à faible courant de fuite, ainsi que des contraintes temporelles définies par l'utilisateur sont utilisées pour synthétiser l'unité d'exécution.

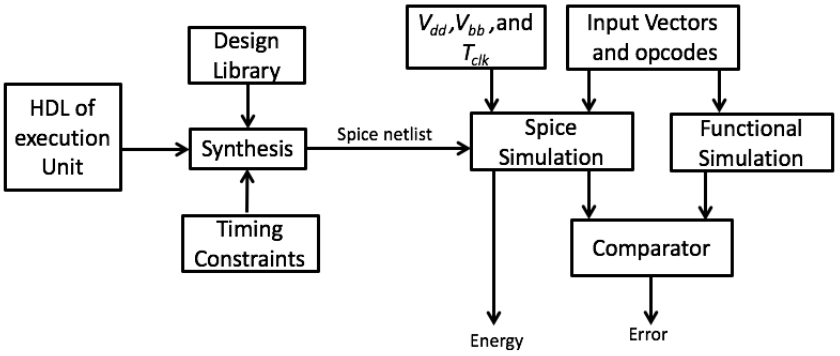


FIGURE 1.11: Flot de caractérisation d'une unité d'exécution d'un cluster $\rho - VEX$

La netlist SPICE de la synthèse est simulée dans le simulateur Mentor Eldo SPICE. Des simulations au niveau transistor sont effectuées pour différentes triades de fonctionnement choisies en fonction des rapports temporels obtenus lors de la synthèse. Un registre à décalage à rétroaction linaire (Linear Feedback Shift Register – LFSR) de 80 bits est utilisé pour générer aléatoirement des vecteurs d'entrées. Concernant les testbenchs, la fonction de génération de vecteur (pattern source function) de SPICE est utilisée pour générer les différents opcodes pour le module à tester. Il en résulte 1000 ensembles de sortie via les simulations SPICE de chaque triade de fonctionnement qui sont comparés avec les sorties de la simulation fonctionnelle effectuée au niveau porte afin de déterminer les erreurs. L'énergie par opération consommée est mesurée dans la simulation SPICE pour différentes opérations. Dans les simulations, la tension d'alimentation V_{dd} varie de 1 à 0,4V. A 1V, les performances des opérateurs sont élevées et sans erreur. La tension V_{dd} est réduite par pas de 0,1V jusqu'à atteindre 0,4V qui correspond à la région proche du seuil. Lorsque V_{dd} diminue, le délai de propagation de la logique combinatoire augmente. Il en résulte des erreurs temporelles. Dans la technologie FDSOI, la technique de polarisation inverse (Back Biasing) est utilisée pour faire varier le seuil de commutation Vt des transistors afin d'augmenter les performances ou de réduire le courant de fuite. Dans les simulations, la tension de polarisation inverse V_{bb} varie entre 0 et $\pm 2V$.

La Fig. 1.12 montre l'énergie par opération par rapport au taux d'erreur binaire pour une UAL effectuant des additions pour différentes triades de configuration. A partir de ce graphique, est il est évidant que lorsque V_{dd}=1V et V_{bb}=2V, l'énergie par opération est au maximum. La Fig. 1.13 est une version zoomée de la partie gauche du

graphique de la Fig. 1.12. La Fig. 1.13 montre la tendance obtenue par la diminution de la tension et l'impact engendré par la réduction de l'énergie par opération. Lorsque V_{dd}=0,5V, l'énergie par opération est réduite de 85% sans erreur. La Fig. 1.14 est la version zoomée de la partie droite de la Fig. 1.12. Pour une application pouvant tolérer un BER de 10%, V_{dd} peut être réduit jusqu'à 0,4V avec une période d'horloge divisée par deux afin d'obtenir une efficacité énergétique de 93% avec un BER acceptable de 7,8%. Les autres opérations de l'UAL montrent des caractéristiques similaires. Comme montré par ces graphiques, dans la région super-threshold (V_{dd} de 0,7 à 1V), il n'y a pas de bénéfice engendré par la technique de polarisation inverse étant donné que le BER est le même pour toutes les triades : sans erreur. Au contraire, dans la région proche du seuil (V_{dd} de 0,4 à 0,6V), les bénéfices de la polarisation inverse sont évidents.

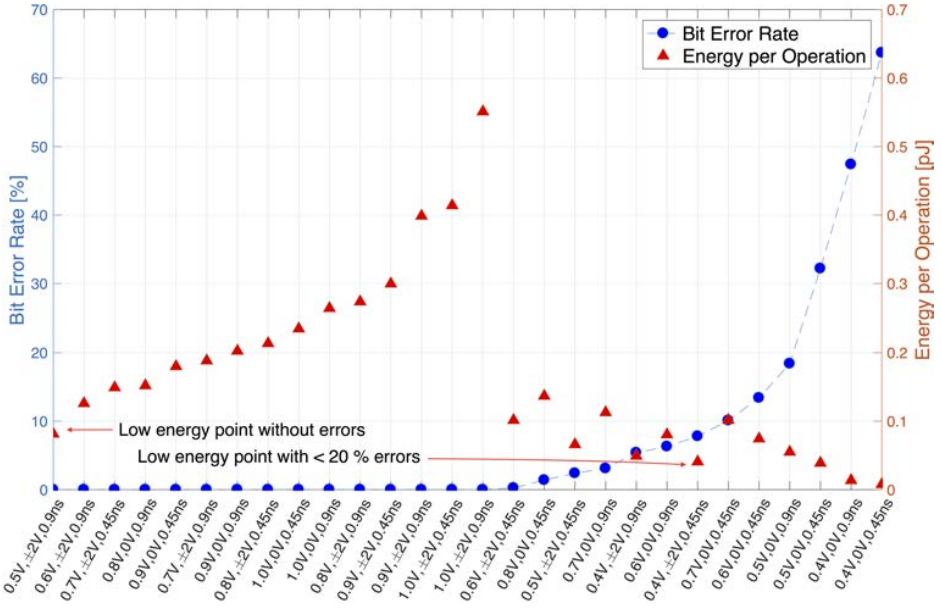


FIGURE 1.12: Énergie consommée en fonction du BER pour une UAL effectuant une addition

Les modèles existant de puissance et d'énergie pour leur estimation au niveau instruction de plateforme VLIW ne sont pas développés dans le contexte d'alimentation proche du seuil. Cette limitation réduit la réutilisation de tels modèles pour des applications comme le traitement d'image et de vidéo, la fouille de données, etc., pour lesquels il y a un intérêt à configurer l'alimentation proche du seuil afin d'obtenir une haute efficacité énergétique au prix d'erreurs temporelles acceptables et paramétrables par l'utilisateur. La méthode de caractérisation proposée est utilisée pour caractériser le comportement d'une unité d'exécution d'un cluster $\rho - VEX$ pour diverses instructions

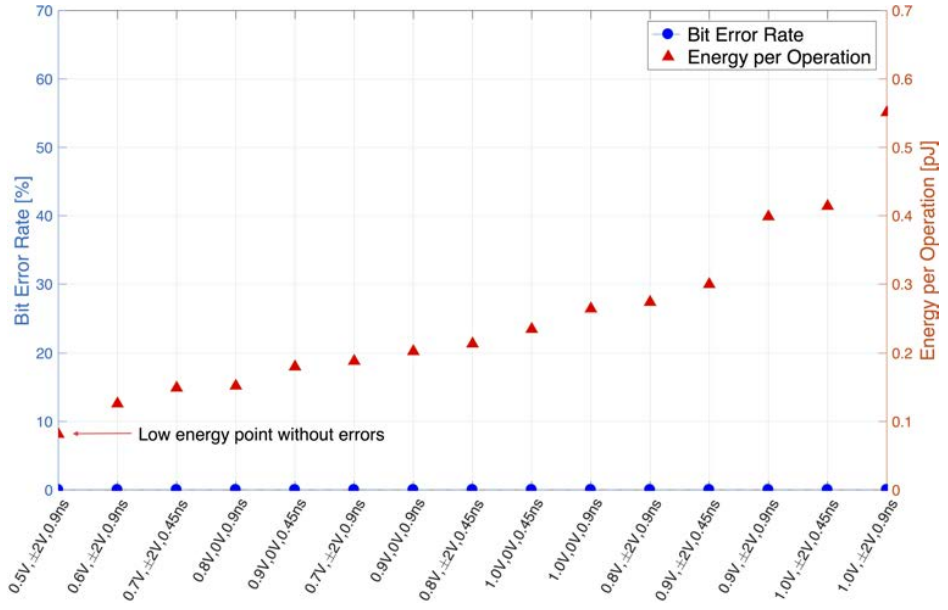


FIGURE 1.13: Gestion de la tension d'une UAL pour atteindre une haute efficacité énergétique

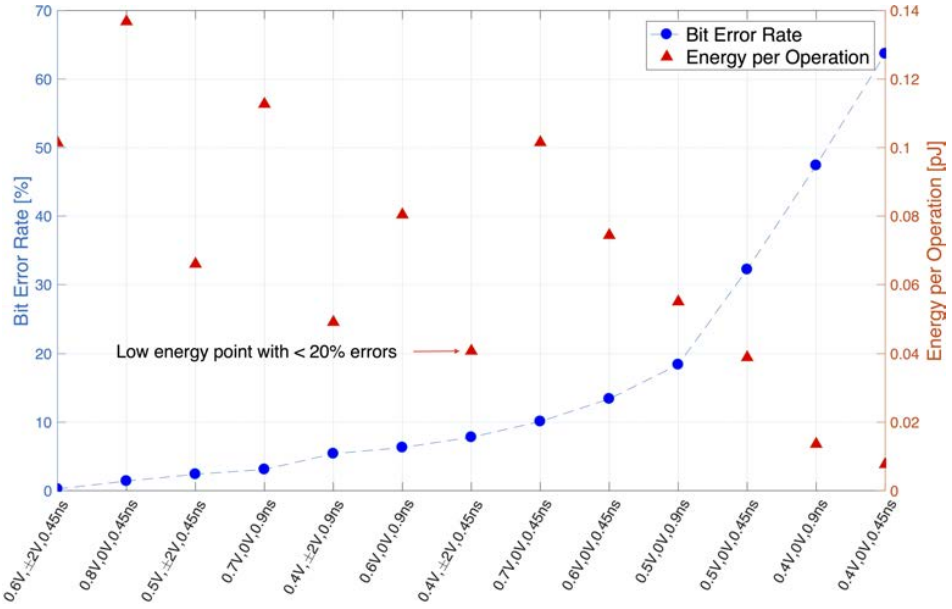


FIGURE 1.14: BER d'une UAL pour le calcul approximatif

et triades de fonctionnement. Avec les informations d'énergie et d'erreurs extraites pour les différentes opérations, une estimation de l'énergie consommée au niveau instruction peut être formulée sous forme d'un framework pour des benchmarks en langage C.

1.5.2 Méthode proposée pour l'estimation de l'énergie au niveau instruction

Dans ce travail, nous proposons une méthode d'estimation énergétique au niveau fonctionnel. La consommation énergétique de différentes instructions est estimée en se basant sur la caractérisation de l'unité d'exécution d'un cluster $\rho - VEX$, détaillée dans la section précédente. En général, l'exécution d'une instruction apparaît à différents étages du pipeline : fetch, decode, execute, et writeback. L'énergie consommée par chacun de ces étages est respectivement notée E_{fetch}, E_{decode}, $E_{totexec}$, E_{reg}, et $E_{writeback}$. La somme de toutes ces énergies donne l'énergie totale consommée pour exécuter une instruction. Dans le total d'énergie consommée par une instruction, l'étage execute domine par rapport aux autres. L'objectif premier de ce travail est d'estimer avec précision l'énergie consommée par l'étage execute pour diverses instructions et avec différentes configurations. Dans la section précédente, la procédure de caractérisation est détaillée pour l'UAL et le multiplieur d'un cluster $\rho - VEX$ selon différentes configurations. Il y a au total 73 instructions différentes pour un $\rho - VEX$. Chaque instruction, selon le type d'opération, active un certain nombre de blocs dans le cluster. Par exemple, une instruction peut nécessiter deux additions et deux multiplications. Les syllabes (une opération encodée) 0 et 3 d'une instruction seront allouées pour les additions, et les syllabes 1 et 2 seront allouées pour les multiplications. A l'aide de la caractérisation des unités UAL et MUL, l'énergie consommée d'une instruction, pour n'importe quelle configuration de fonctionnement, est déterminée par la somme des énergies des 4 unités fonctionnelles. Etant donné que les unités fonctionnelles sont caractérisées pour de nombreuses triades de fonctionnement, il est simple d'estimer l'énergie pour n'importe quel benchmark par une simple compilation à travers un compilateur VEX. Cette méthode peut être facilement adoptée pour estimer la consommation énergétique pour différents processeurs sans avoir besoin de re-caractériser. Le reste de cette section explique la procédure d'estimation d'énergie pour l'étage execute d'un cluster $\rho - VEX$.

Pour estimer la consommation énergétique au niveau instruction, le programme benchmark est compilé via la chaine d'outils VEX et les fichiers intermédiaires *.s* et *.cs.c* sont générés. Le fichier *.cs.c* contient la macro-déclaration de chaque opération VEX, par exemple ADD, XOR, etc. Dans la section précédente, l'énergie par opération est calculée à partir des simulations SPICE. En connaissant le nombre de fois qu'une opération est

exécutée il est possible d'estimer l'énergie consommée pour n'importe quelle instruction. En étendant cette méthode à toutes les instructions d'un programme, la consommation énergétique totale peut être estimée. Pour un programme avec P séquence de N instruction $(i_1, i_2, i_3, ..., i_N)$ où chaque instruction i_n contient L différentes opérations, l'estimation énergétique de l'unité d'exécution peut s'écrire:

$$E_{tot}(i) = \sum_{\forall k \in L} E_{exe}(o_k) \times M_k \tag{1.3}$$

$$E_{exe}(P) = \sum_{\forall n \in N} E_{tot}(i_n) \tag{1.4}$$

où $E_{tot}(i_n)$ représente l'énergie consommée par l'unité d'exécution pour exécuter une instruction (i_n). $E_{exe}(o_k)$ est l'énergie consommée par une opération (o_k), et M_k est le nombre de fois que l'opération (o_k) est exécutée. L'énergie consommée par différentes opérations $E_{exe}(o)$ est obtenue à travers la phase de caractérisation pour différentes conditions de fonctionnement. De plus, l'Eqn. 1.3 représente l'énergie totale consommée par l'unité d'exécution pour une instruction i_n. L'Eqn. 1.4 représente l'énergie totale consommée par l'unité d'exécution pour exécuter un programme P avec N instructions. Étant donné que l'on se focalise uniquement sur l'unité d'exécution d'un cluster $\rho{-}VEX$, seulement l'énergie consommée par cette unité est estimée. En considérant une consommation énergétique arbitraire pour les autres modules, l'énergie totale consommée par un cluster $\rho - VEX$ pour exécuter un programme P peut s'écrire:

$$E_{total}(P) = E_{fetch}(P) + E_{decode}(P) + E_{totexec}(P) + E_{reg}(P) + E_{writeback}(P) \tag{1.5}$$

où $E_{fetch}(P)$, $E_{decode}(P)$, $E_{reg}(P)$, $E_{writeback}(P)$ représentent l'energie consommée pour les étages fetch, decode, registers, writeback, lors de l'execution du programme considéré.

1.5.3 Validation de la méthode

La méthode d'estimation d'énergie est implémentée pour un ensemble de benchmarks écrit en C. Pour tous les benchmarks, le fichier *.cs.c* est modifié afin d'augmenter un compteur pour chaque exécution spécifique d'une opération. Via les équations Eqn. 1.3 et Eqn. 1.4, l'énergie consommée de l'unité d'exécution d'un cluster p-Vex est estimée pour différentes triades de fonctionnement. Les benchmarks choisis sont les plus utilisés

TABLE 1.4: Détails des programmes de test

Benchmarks	Total Number of Instructions	Total number of traces
abs	5	1
find_min	41	7
crc8_comp	57	7
crc16_comp	57	7
bphash	60	9
binary_search	88	11
fir_basic	115	16
calc_neigbhour	135	14

dans les applications de mathématiques, traitement du signal et cryptographie, et sont issus de la suite Mediabench. La Table 1.4 montre la liste des benchmarks ainsi que le total d'instruction pour chaque programme.

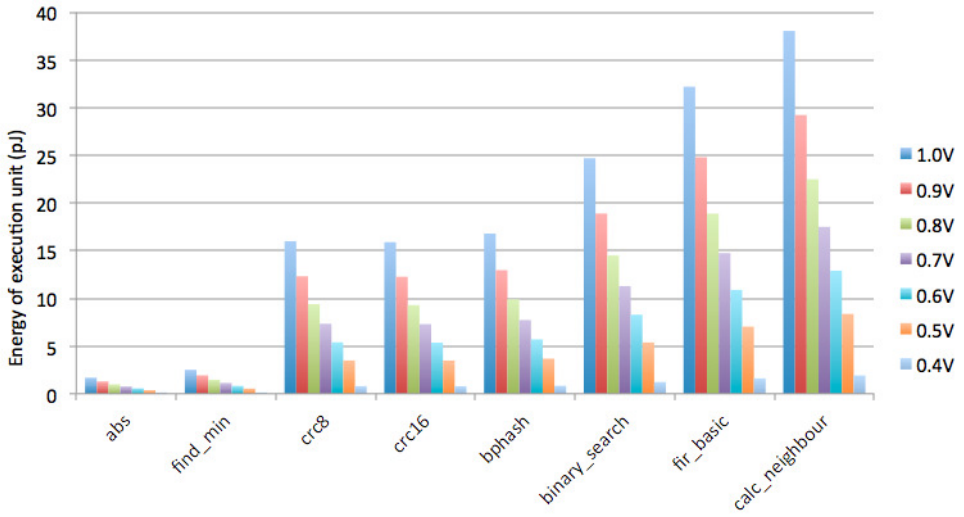


FIGURE 1.15: Estimation d'énergie pour différents benchmarks et pour différents V_{dd}, $V_{bb} = 0$V, et $T_{clk} = 0.9$ns

Le fichier *.cs.c* modifié est exécuté dans un compilateur C afin d'estimer l'énergie consommée pour différentes triades. La Fig. 1.15 montre l'estimation d'energie pour l'unité d'exécution du cluster $\rho - VEX$ pour chaque programme et pour différentes valeurs de V_{dd} allant de 1V à 0,4V sans polarisation inverse. Réduire V_{dd} de 1V à 0,4V réduit en moyenne la consommation énergétique par un facteur 20. Ce gain significatif est limité par les erreurs temporelles mis en avant sur le graphique montrant l'énergie en fonction du BER dans la section caractérisation. Cette limitation peut être compensée en utilisant de la polarisation inverse pour limiter le BER à une marge acceptable au

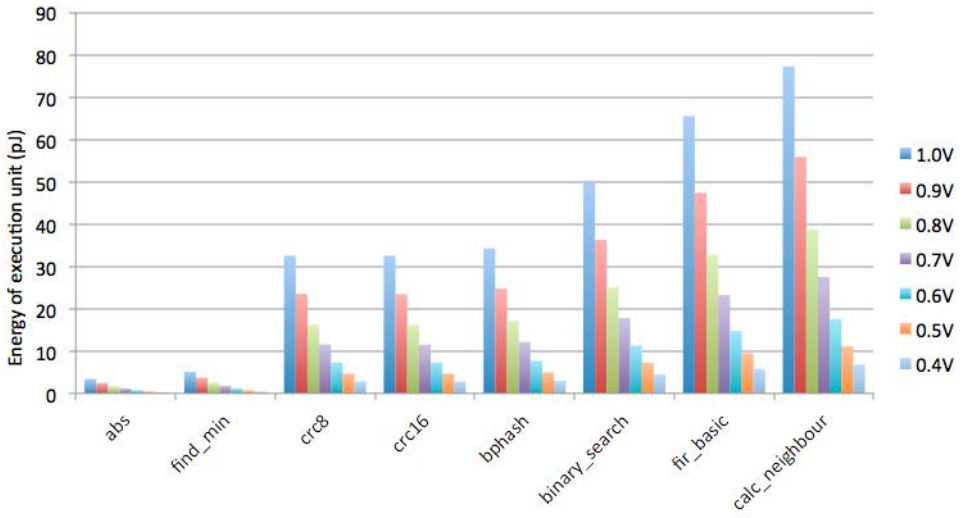


FIGURE 1.16: Estimation d'energie d'une unité d'exécution pour différents benchmarks pour différents V_{dd}, V_{bb} = 2V, et T_{clk} = 0.9ns

détriment d'une réduction du gain en énergie. La Fig. 1.16 montre l'estimation d'énergie pour chaque application avec un V_{bb} de 2V. Avec de la polarisation inverse et pour V_{dd} à 0,4V, l'énergie est réduite en moyenne d'un facteur 11.

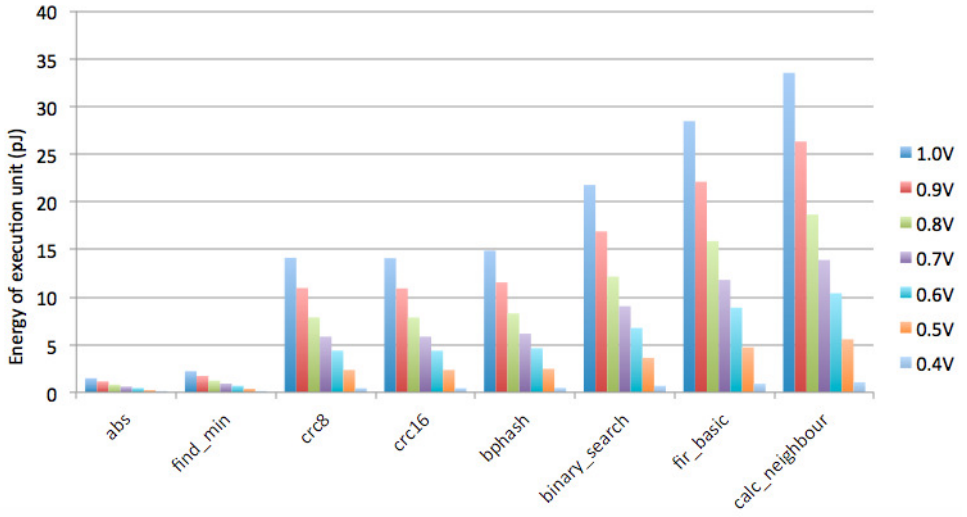


FIGURE 1.17: Estimation d'energie d'une unité d'exécution pour différents benchmarks pour différents V_{dd}, V_{bb} = 0V, et T_{clk} = 0.45ns

En plus de la diminution de la tension d'alimentation, une augmentation de la fréquence d'horloge est effectuée pour réduire la période T_{clk} de 0,9ns à 0,45ns. Cela augmente les gains en énergie au coût d'une diminution du BER. Pour une période d'horloge réduite, les programmes atteignent une réduction énergétique d'un facteur 31 pour un V_{dd} de 0,4V par rapport à 1V. Avec un V_{bb} de 2V, les erreurs temporelles sont

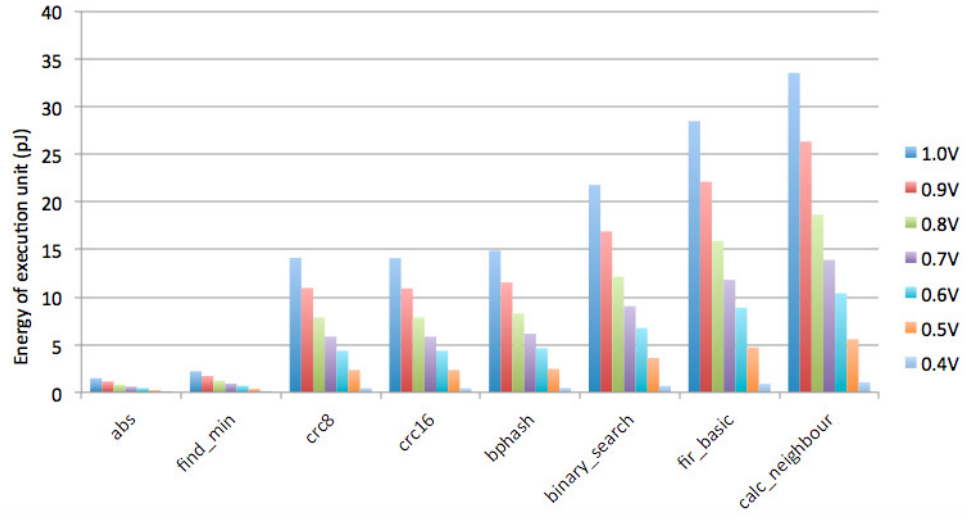


FIGURE 1.18: Estimation d'energie d'une unité d'exécution pour différents benchmarks pour différents V_{dd}, $V_{bb} = 2$V, et $T_{clk} = 0.45$ns

limitées et la consommation énergétique est divisée par 10 en moyenne par rapport à un V_{dd} de 1V. Le Framework proposé permet de comprendre de façon simple le comportement de l'unité d'exécution d'un cluster $\rho - VEX$ pour différentes triades. Avec ce Framework, un concepteur peut définir le meilleur compromis entre l'énergie et la précision de calcul pour n'importe quel programme en choisissant, en cours de fonctionnement, la triade à configurer.

1.6 Conclusion

Dans cette thèse, un Framework est proposé pour gérer les erreurs temporelles et pour estimer la consommation énergétique des applications tolérantes aux erreurs dans un cadre de calcul proche du seuil. Dans un environnement où l'énergie est limitée, maintenir un circuit fonctionnel pour une longue période est un challenge majeur dans la conception actuelle de circuit numérique. De nombreuses techniques dites faible consommation sont explorées a différents niveaux d'abstraction afin d'améliorer l'efficacité énergétique. La gestion dynamique de l'alimentation est une technique phare permettant d'augmenter l'efficacité énergétique des architectures en réduisant la tension d'alimentation en cours de fonctionnement. Dans ce travail, nous avons exploré le calcul proche du seuil afin d'améliorer les gains en énergie en réduisant l'alimentation au plus proche du seuil de

commutation des transistors. Réduire la tension d'alimentation rend les circuits plus sensibles aux effets de variabilité et aux erreurs temporelles. Nous avons proposé d'utiliser la technologie FDSOI, qui est de manière inhérente une technologie à faible courant de fuite, afin de contrer les effets de la variabilité en utilisant la technique de polarisation inverse. Différentes techniques de gestions des erreurs, comme celle du double échantillonnage, sont généralement utilisées pour gérer les erreurs. La majorité des techniques basées sur le double échantillonnage utilisent une fenêtre fixe de spéculation pour détecter et corriger les erreurs temporelles. Dans ce travail, nous avons proposé l'utilisation d'une fenêtre dynamique de spéculation permettant de détecter et de corriger les erreurs temporelles tout en adaptant la tension d'alimentation au plus proche de la tension de seuil. Nous avons démontré les avantages de cette spéculation dynamique via une plateforme FPGA de Xilinx. Nous avons overclocké des applications de benchmarks au sein du FPGA au delà de la fréquence maximale estimée. De plus, les effets de variabilité ont été introduits sur le FPGA en faisant varier sa température. Dans ces expérimentations, la méthode proposée est capable d'ajuster dynamiquement la fréquence d'horloge afin de contrer les impacts liés à la variabilité et en respectant des marges d'erreurs définies par l'utilisateur. Nous avons atteint une augmentation de la fréquence d'horloge de 71% avec la méthode de spéculation dynamique sans erreur, avec un surcoût limité de 1,9% de LUT et 1,7% de registres.

En plus des aspects de détection et de correction des erreurs, ce travail explore également le domaine du calcul approximatif. Dans ce domaine émergeant, les erreurs de calcul sont délibérément tolérées afin d'atteindre une haute efficacité énergétique. Dans notre méthode de spéculation dynamique, l'utilisateur peut décider de tolérer un nombre d'erreurs en spécifiant une marge. En fonction d'un niveau de précision à atteindre selon l'application visée, l'utilisateur peut choisir un compromis entre précision et énergie. En effet, l'utilisateur peut décider en cours de fonctionnement, s'il souhaite une haute précision ou une haute efficacité énergétique en configurant la tension d'alimentation proche ou loin du seuil. La gestion agressive de la tension (Voltage OverScaling – VOS) peut être efficacement appliquée à des applications tolérantes aux erreurs. Ainsi, nous avons proposé un Framework pour modéliser de manière statistique des opérateurs arithmétiques sous VOS. Le comportement des opérateurs alimentés proche du seuil est différent d'un V_{dd} nominal. Nous avons effectué une étude approfondie de ces opérateurs au niveau transistor afin de caractériser leur comportement pour du VOS. Les modèles

statistiques peuvent être utilisés au niveau algorithme pour simuler l'effet du VOS afin de choisir la bonne triade de configuration (V_{dd}, V_{bb} et T_{clk}) pour atteindre une haute efficacité énergétique avec une perte de précision acceptable en fonction de l'application.

Pour étendre le champs d'application du VOS et de la spéculation dynamique, nous avons exploré une plateforme reconfigurable via un processeur VLIW. L'unité d'exécution d'un cluster du processeur VLIW $\rho-VEX$ est étudiée. Nous avons proposé un Framework pour caractériser, évaluer et valider l'estimation énergétique de cette unité pour différentes triades de fonctionnement. L'unité fonctionnelle du cluster $\rho-VEX$ est caractérisée au niveau transistor afin d'extraire avec précision l'énergie consommée et les erreurs de données pour différentes triades. Des simulations au niveau transistor sont très longues même pour un programme simple. Ainsi, nous avons proposé une estimation d'énergie au niveau instruction par une simple compilation en C d'un programme. De ce fait, nous pouvons estimer la consommation d'énergie d'un programme pour différentes configurations très rapidement à partir d'une caractérisation au niveau transistor. Nous avons également proposé une procédure de validation alternative pour s'assurer de la précision de l'estimation énergétique en comparant avec une simulation complète SPICE des benchmarks. Dans ce Framework nous avons atteint une précision d'estimation d'énergie ayant seulement 2% d'erreurs en moyenne.

1.7 Perspective

Un des principaux challenges dans la conception de circuit numérique est d'atteindre une haute efficacité énergétique pour un coût minimum en terme de surface de silicium tout en fournissant un compromis acceptable entre précision et performance du circuit. Etant donnée la dépendance quadratique entre la consommation d'énergie et la tension d'alimentation, réduire V_{dd} mène à une meilleure efficacité énergétique. Une simple observation des différentes technologies CMOS sur les dix dernières années montre la tendance à la diminution de la tension d'alimentation. Dans la région proche du seuil, la latence et l'énergie sont fortement linéaire par rapport à V_{dd}. Utiliser des transistors dans cette région offre un meilleur contrôle sur le compromis énergie-erreur. La technologie FDSOI est utilisée pour améliorer les bénéfices de l'alimentation proche du seuil. Cependant, l'impact de la variabilité et les erreurs temporelles l'emportent sur les

bénéfices produits par cette baisse de la tension d'alimentation. Dans cette thèse, nous avons adressé cette problématique via trois contributions.

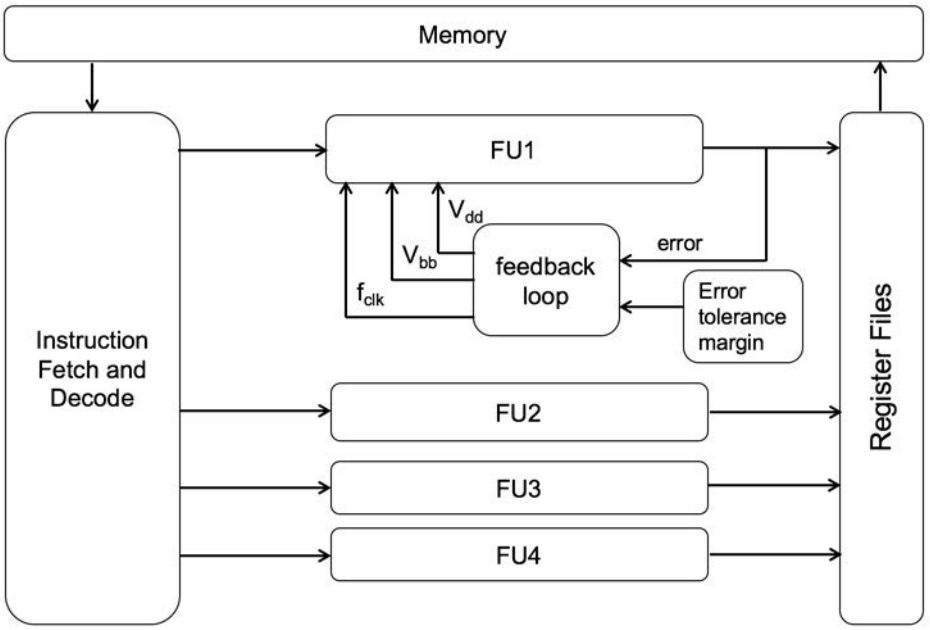


FIGURE 1.19: Perspective générale de la thèse

La Fig. 1.19 mets en lumière la perspective de cette thèse. On considère un cluster VLIW avec ses différents étages pipelinés : chargement d'instruction (instruction *fetch*), décodage d'instruction (*decode*), l'étape d'exécution est représentée par une unité fonctionnelle (FU), des files de registres (*register files*) et un bloc mémoire (*memory*). Dans une exécution classique, les instructions sont chargées et décodées par les étages *fetch* et *decode*. Les différentes opérations de chaque instruction sont exécutées dans les FU en parallèles. La sortie est sauvegardée dans la file de registres et la mémoire. Afin d'atteindre une haute efficacité énergétique, la tension de chaque FU peut être diminuée. La diminution du V_{dd} peut générer des erreurs qui peuvent être gérées en utilisant les méthodes expliquées dans le Chapitre 3. Dans le Chapitre 3, les limitations des techniques sont discutées en détail. Ainsi, le besoin pour une approche unifiée pour gérer les erreurs est nécessaire afin d'atteindre un compromis énergie-erreurs.

La Fig. 1.19 montre cette approche unifiée permettant de gérer les erreurs temporelles dans le contexte d'alimentation proche du seuil et de calcul approximatif pour des applications tolérantes aux fautes. Les FU sont monitorés en utilisant la méthode proposée de spéculation dynamique utilisant une boucle de retour. Le principal avantage de la méthode proposée est de fournir une tolérance aux erreurs adaptable en fonction

d'une marge d'erreur définissable par l'utilisateur. A l'aide de la mesure temps réel de la marge temporelle disponible, la méthode proposée peut prédire les possibilités d'erreurs. En cas de besoin, un mécanisme de prévention peut être mise en oeuvre afin de configurer la triade d'opération. En fonction de la marge d'erreur configurée par l'utilisateur, les erreurs peuvent être détectées et corrigées ou ignorées en cours de fonctionnement selon le besoin des différentes applications. La boucle de contre-réaction effectue l'adaptation de la fréquence d'horloge ou de la tension d'alimentation lorsqu'il y a besoin d'augmenter l'efficacité énergétique ou les performances. La méthode proposée fournit une flexibilité pour le concepteur en fonction des besoins.

Dans ces travaux, le comportement des différents opérateurs arithmétiques sont étudiés pour différentes triades de fonctionnement. Cela fournit un aperçu détaillé de la génération et de la propagation dans les différents opérateurs. A l'aide de cette étude, la spéculation dynamique est utilisée tel qu'illustrée en Fig. 1.19 afin de réduire l'énergie dans les opérateurs arithmétiques en temps réel. Afin de facilement comprendre différents compromis possibles, un framework d'estimation d'énergie est proposée pouvant simuler n'importe quel programme de test via un compilateur C afin de fournir l'estimation énergétique selon différentes triades de fonctionnement. Cela fournit une aide au développeur pour comprendre les bénéfices et les impacts, ainsi que les coûts pour atteindre le meilleur compromis entre énergie et erreurs.

Chapter 2

Introduction

Advancement in CMOS (Complementary Metal-Oxide-Silicon) IC (Integrated Chip) technology has improved the way in which integrated circuits are designed and fabricated over decades. Gordon Moore predicted in 1965, that the number of components that could be incorporated per integrated circuit will double every two years [9]. Scaling down the size of transistors from micrometers to few nanometers has given the capability to place more and more tinier transistors in a unit of chip area that upheld the Moore's law over the years. Scaling feature size expanded the functional capability of ICs multi-fold by accommodating more components. Components once placed on Printed Circuits Board (PCB) along with processor are now packed inside single chip as popularly know as System-on Chip (SoC). Prime requirements in any IC design is to increase the performance (by reducing gate delay) and to reduce power consumption and area (silicon footprint). As per general scaling rule, feature size scaled down by a factor of K results in reduction of gate delay by K, power by K^2, and energy by K^3. This makes scaling an important technique to achieve all the three prime requirements of IC designing. In 2007, International Technology Road-map for Semiconductors (ITRS) highlighted slowdown in the trend of feature size scaling and forecasted that the scaling trend of CMOS will come to a halt at 22nm [10]. Various challenges like Physical, Material, Power-thermal, Technological, and Economical that hinder the scaling trend have been discussed in [11]. Out of the listed challenges, physical limitations due to incremental tunneling and leakage currents because of smaller channel length pose a major challenge to scaling of CMOS devices. ITRS stated in 2011, "One of the key problems that designers face due to further shrinking of features sizes is the increasing variability

of design-related parameters, resulting either from variations of fabrication parameters or from the intrinsic atomistic nature" [12]. Also, to improve energy efficiency supply voltage (V_{dd}) scaling [1] is employed in most of the Ultra-Low Power (ULP) designs. However reduction in V_{dd} augments the impact of variability in sub-nanometer designs.

2.1 Variability in CMOS

Variability is defined as the discrepancies in the functioning of circuits due to variation in Process, Voltage and Temperature (PVT) of the devices. In the following sub-sections, each of these variations is discussed briefly.

2.1.1 Process variations

Various imperfections in transistor manufacturing process that lead to process variations are listed in [13].

- Variations in the critical dimension of devices happen due to imperfections in fabrication technology because of higher wavelength light is used in lithography to make smaller size transistors.
- In a doped channel, impurity atoms placed by dopant implantation randomly fluctuate causing variations in threshold voltage of the devices. This effect is called Random Dopant Fluctuation (RDF).
- Variations in gate oxide thickness (TOX) induces variations in threshold voltage. This effect is more pronounced in technologies of 30nm and below.

Process variations are broadly classified into global and local based on the cause and impact of the variations. Global variations impact all the transistors equally due to discrepancies in parameters like oxide thickness and dopant concentration. In contrast, local variations impacts are different for different transistors.

2.1.2 Voltage variations

Supply voltage V_{dd} plays an important role in proper functioning of digital circuits. Reduction in V_{dd} results in increased gate delay t_p, which in turn affects the performance of the digital circuits

$$t_p = \frac{V_{dd}.C_{load}}{k(V_{dd} - V_t)^n} \tag{2.1}$$

where C_{load} is the load capacitance, V_t is the threshold voltage, k and n are technology parameter that determines drain current characteristics. For a long-channel device, value of n is 2 and for a short-channel device $1 \leq n < 2$. Supply voltage variations are mainly caused by IR drop and di/dt noise [13].

- IR drop, is due to the voltage drop because of the resistance of interconnects.
- di/dt noise is due to the voltage drop because of time-varying current drawn by the inductance of the package leads.

Apart from the above-mentioned voltage variations, this impact is also seen in the controlled supply voltage scaling implemented in the digital circuit to reduce the energy consumption of the circuit.

2.1.3 Temperature variations

As the chip density getting doubled every two years, temperature of the device increases proportionally to the chip density. For every 10°C increase in the operating temperature, failure rate roughly doubles [14] due to reduced carrier mobility and increased interconnect resistance. In general, gate delay t_p increases with increasing temperature and decreasing V_{dd}. However, at low supply voltages, t_p decreases with increasing temperature known as *inverted temperature dependence* (ITD). Due to ITD it becomes really difficult to analytically determine the impact of temperature variations on t_p. Effect of ITD is discussed in [15].

- Due to ITD, it becomes hard to define corners by independently varying V_{dd} and V_t.
- It becomes more difficult to find short paths which cause hold time violations.

- Impact of ITD worsens as V_{dd} decreases and V_t increases.

2.2 Voltage Scaling

Feature size scaling increased the chip-density by packing more and more tinier transistors per unit area of a chip. But operating all of the transistors at any given time is highly impossible and improbable. When the number of transistors operating at a given time increases, power-density of the chip increases. Higher power-density demands better heat-sink, which in turn increases the cost of cooling system. Cooling systems are huge in size compared to the size of chip made of sub-nanometer transistors. This road-block in power-management due to increased chip density is known as *Power Wall*. To handle *Power Wall*, without compromising the chip-density, *Voltage Scaling* technique has been developed. Total power consumption of a digital design can be represented as

$$P_{total} = \alpha.V_{dd}^2.F_{clk}.C_{load} + V_{dd}.I_{short} + V_{dd}.I_{leak} \tag{2.2}$$

where α is switching activity, F_{clk} is clock frequency, I_{short} represents total short current between V_{dd} and Gnd, and I_{leak} represents total leakage current of the design. As shown in Equation 2.2, total power consumption (sum of dynamic, static, and leakage power) is directly proportional to the supply voltage of the design. Scaling down the V_{dd} results in drastic reduction in P_{total} of the design. By reducing V_{dd} of the entire chip or different parts of the chip, the heating issue can be handled in densely packed chips. Over the years, voltage scaling has been used as a prominent technique to improve energy efficiency in digital systems, scaling down the supply voltage effects in quadratic reduction in energy consumption of the system, as

$$E_{total} = V_{dd}^2.C_{load} \tag{2.3}$$

In literature, different variants of voltage scaling techniques are proposed to improve the energy efficiency of digital designs [1], [2].

2.2.1 Dynamic Voltage and Frequency Scaling (DVFS)

Dynamic Voltage and Frequency Scaling (DVFS) is a voltage scaling technique in which the supply voltage and frequency of the design are varied dynamically to improve the energy efficiency at the cost of little performance loss. Fig. 2.1 shows the distribution of tasks ($T1$, $T2$, $T3$...) without and with DVFS. By applying DVFS, V_{dd} corresponding to each task can be varied at runtime by utilizing the idle time between these tasks to compensate for the increase in latency due to reduction in V_{dd}. By reducing V_{dd}, latency of the task is increased according to the availability of idle time between tasks. Task distribution with DVFS scheme improves the energy efficiency in microprocessors [16]. Some of the commercial processors like Intel's Core 2 Quad and later, AMD's K6, have successfully implemented DVFS technique like *SpeedStep* [17] and *PowerNow* [18] respectively. DVFS can be classified based on the granularity as macro and micro. For a design with multiple modules, DVFS can be applied with different parameters for each of the modules. DVFS at micro level provides more enhanced control to improve the energy efficiency at module level at the cost of complex DVFS controller and feedback network.

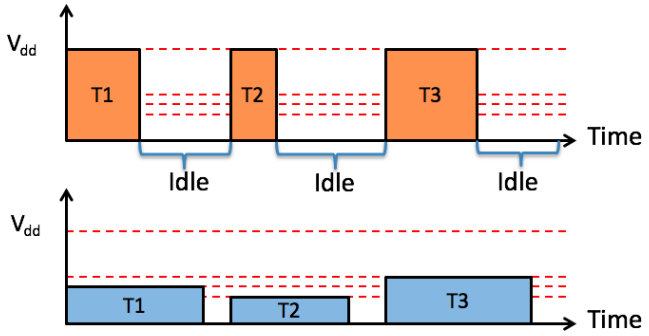


FIGURE 2.1: Task distribution without and with DVFS scheme

Based on the policy determination, DVFS can be further classified as open-loop based *Dynamic Voltage Scaling* (DVS) and closed-loop based *Adaptive Voltage Scaling* (AVS). DVS is shown in Fig. 2.2 is the most common form of DVFS used in digital designs [17], [18]. Based on task profiling, various operating voltage and frequency are pre-determined for different tasks of the design. A controller, according to the task profile, chooses pre-determined voltage and frequency values stored in a look-up table to improve the energy efficiency of the design. Intel's *speedstep* technology [17] is based on DVS approach, the values of clock frequency and supply voltage corresponding to

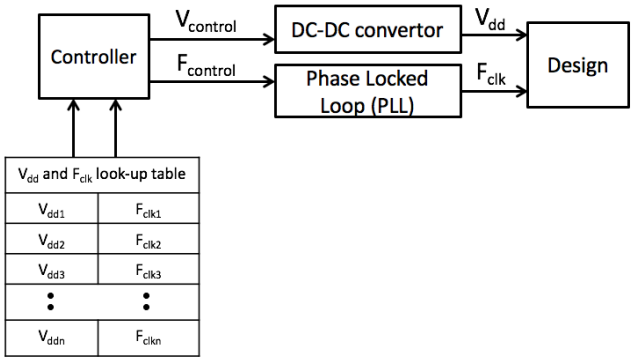


FIGURE 2.2: Block diagram of Dynamic Voltage Scaling (DVS)

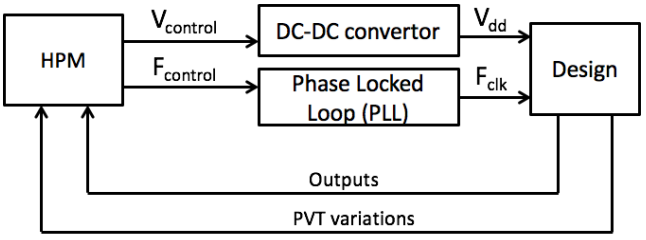


FIGURE 2.3: Block diagram of Adaptive Voltage Scaling (AVS)

specific models are stored in a read-only memory of the BIOS. The operating modes corresponding to maximum and minimum clock frequency are referred as High Frequency Mode (HFM), and Low Frequency Mode (LFM) respectively. The Model Specific Register (MSR) ensures scaling of supply voltage and clock frequency between HFM and LFM. A similar approach is used by AMD's *PowerNow* technology [18].

In AVS shown in Fig. 2.3, operating voltage and frequency are determined at runtime based on feedback from Hardware Performance Monitor (HPM) [19], [20], [21]. HPM monitors variations due to PVT, performance and energy consumption of the system. AVS system improves energy efficiency multi-fold at the cost of more silicon footprint for HPM. On contrast, DVS is less complex and easy to implement with limited energy gains.

2.2.2 CMOS Transistor Operating Regions

In CMOS technology, the transistor can operate in three regions namely, 1) super-threshold, 2) near-threshold, and 3) sub-threshold region. In general, super-threshold

region is used for digital design due to the high performance of the transistor in this regime. In super-threshold region, transistors consume more energy to deliver the high performance. Reducing supply voltage has been explored in greater extent to unlock the opportunities of higher energy efficiency by operating the transistor at near or sub threshold voltage [22]. In super-threshold region, the transistor is supplied with maximum supply voltage corresponding to the specific technology. The strong conduction in this regime results in high switching speed, less impact due to variations, and high energy consumed by the transistors. In super-threshold region, the delay and energy consumption of the circuits are monotonically varied with respect to the supply voltage scaling. The quadratic dependence of the energy with the supply voltage scaling results in more energy consumption in the super-threshold region. For ultra-low power applications like sensor networks, monitoring devices, etc., which generally operate with lesser frequencies, supply voltage can be scaled down to achieve reduction in energy consumption. In general, lower margin for voltage scaling is around 0.7 V to 0.8 V, for ultra-low power applications, this margin can be further lowered to 0.3 V to 0.4 V depending on the technology used.

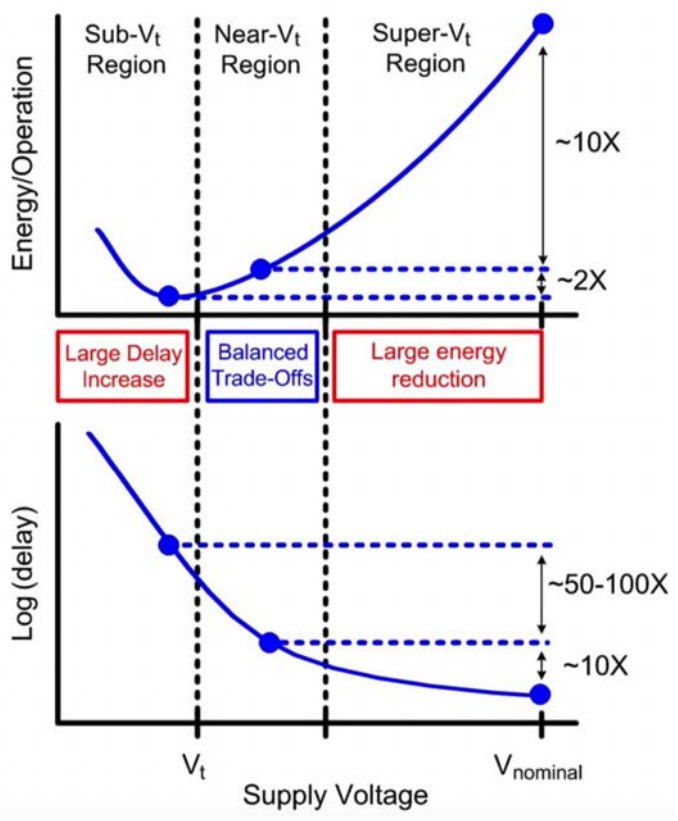


FIGURE 2.4: Energy and delay in different V_{dd} regimes [22]

2.2.2.1 Sub-Threshold Region

In sub-threshold region (STR), the supply voltage of the transistor is slightly lesser than the threshold voltage of the transistor. Though in general terms it is said that the transistor is in cut-off mode, there is a little sub-threshold current flowing through the channel. In sub-threshold region, the drain current I_d decreases exponentially, thus a non-zero gate voltage that is less than V_t will still produce a drain current that is larger than the off current($I_{off} = I_d$ when $V_{gs} = 0$V). The ratio of I_{on}/I_{off} knows as sub-threshold slope determines the speed of the transistor. Steeper the slope, faster the transition between OFF and ON logic levels. Though the ratio of I_{on}/I_{off} is positive in the sub-threshold region, a lower I_{on}/I_{off} ratio can lead to functional problems [23]. Also, the transient behavior is much slower because the on-current in this region is orders of magnitude less than in strong inversion (super-threshold) region. In sub-threshold region, delay increases exponentially when the supply voltage is scaled down. Since the leakage energy is linear with the circuit delay, the fraction of leakage energy increases with the supply voltage scaling. Fig. 2.4 shows the plots of energy and delay vs. supply voltage in different operating regions. In sub-threshold region, energy per operation is 12× less than the super-threshold region. On the other hand, delay increases by 50-100×. Also, the leakage energy dominates the benefits from supply voltage scaling in Sub-V_t region.

2.2.2.2 Near-Threshold Region

In near-threshold region (NTR), the supply voltage of the transistor is slightly more than the threshold voltage of the transistor. In this region both delay and energy are roughly *linear* with the supply voltage, as shown in Fig. 2.4. The I_{on}/I_{off} ratio is more when compared to the sub-threshold region, which improves the speed and minimizes the leakage. Due to the reduced supply voltage, the circuits operating in NTR can achieve a 10× reduction in energy per operation at the cost of the same magnitude of reduction in operating frequency. As a result of reduced energy per operation and operating frequency, more cores can be operated in the estimated power budget. Parallelism in the execution of the application can be used to reduce the effect of the frequency degradation in the near-threshold regime [24]. Another issue is that sub- and near-threshold designs are more sensitive to variations due to RDF [25]. In [26] and [27], it is shown that

Silicon on Insulator (SOI) technology is highly immune to RDF variations because the high-k dielectric reduces the gate leakage currents and good channel electrostatic control diminishes the drain leakage currents.

2.3 FDSOI Technology

Fully Depleted Silicon On Insulator (FDSOI) is a planar CMOS based on SOI technology [28], [29]. To improve energy efficiency different methods like DVFS, Dynamic V_t control, etc. are developed. But due to the limitations of Bulk CMOS technology, all these techniques under performed than their maximum potential. To outweigh the benefits of low power techniques, FDSOI technology has been developed. In Bulk CMOS, voltage scaling trend slowed, because scaling V_{dd} demands scaling of V_t to maintain noise margin. Scaling V_t is limited due to high leakage current in bulk CMOS. This, in turn, limits V_{dd} scaling. Fig. 2.5, shows the cross-section view of Bulk versus FDSOI device. Unlike Bulk, in FDSOI there is a thin Buried Oxide (BOX) layer between body and substrate of the device. BOX layer confines charge movement to the thin channel which makes FDSOI an intrinsically low leakage technology. Thin BOX layer reduces junction capacitance between source/drain and substrate, thus total dynamic power is lowered. On the other hand, BOX layer isolates the body from the base substrate causing *floating body* major limitation in SOI technologies. The floating body can accumulate charge over time, which causes variation in threshold voltage of the device. In FDSOI, floating body problem is addressed by removing dopants from the channel. Also, the undoped channel makes it fully depleted and reduces the effect of RDF.

FDSOI devices are in general classified as *Low* V_t (LVT) and *Regular* V_t (RVT). *Low* V_t devices are constructed on *Flip Well*, meaning PMOS on P-well and NMOS on N-well. *Regular* V_t devices are constructed on *Conventional Well*, with PMOS on N-well and NMOS on P-well. LVT devices with Forward Body-Bias (FBB) provides boost to performance by lowering V_t of the device. Data from ST-Microelectronics shows, in near-threshold regime ($V_{dd} = 0.4V$), 1V FBB provides 120% boost to performance when compared to no body-biasing. On the other hand 18% performance boost in super-threshold regime ($V_{dd} = 1V$). RVT devices with Reverse Body-Bias (RBB) reduces leakage current by increasing V_t of the device. This makes FDSOI with body-biasing a favourable candidate for near-threshold computing [30].

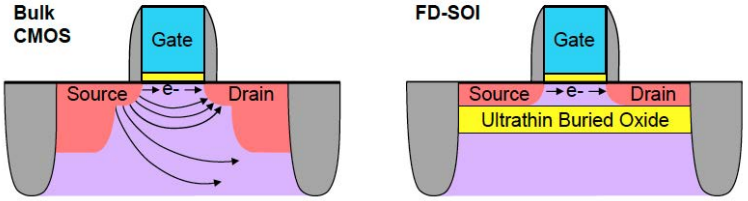


FIGURE 2.5: Bluk CMOS vs FDSOI [31]

2.4 Context of the work

In the present trend of sub-nanometer designs, inherent low-leakage technologies like FD-SOI, enhance the benefits of voltage scaling techniques. Back-biasing or body-biasing is a key feature of FDSOI devices, by which V_t of the transistors can be altered dynamically to counter the impact of variability. Back-biasing is also used to obtain optimum trade-off between performance and tolerance towards noise. In [32], back biasing opportunities in UTBB (Ultra-Thin Body and Box) FDSOI technology for sub-threshold digital design is discussed. In [32], apart from LVT and RVT, other single well (SW) approaches are also presented. *Single N-Well (SNW)* and *Single P-Well (SPW)* devices can be seen at intermediate options between LVT and RVT. SNW and SPW outperform RVT in terms of performance and LVT in terms of leakage reduction. Multi-V_t approach is advocated for designing pipelined datapaths. Where the critical paths can be designed with high performance devices, while the other paths can be designed with low leakage devices. In [29], different ways to solve multi-V_t co-integration challenges for UTBB FDSOI are addressed in detail.

The main objective of this work is to address the variability issues and handling timing errors for error resilient applications in the context of near-threshold computing. In the above sections, merits of near-threshold computing, dynamic voltage scaling, and FDSOI are highlighted. It is highly beneficial to design a digital system by taking advantage of high energy efficiency of NTC, tolerance towards soft errors and body-biasing of FDSOI, and the optimum trade-off between performance and energy efficiency of DVS. In spite of the advantages, the impact of variability and timing errors outweighs the above-mentioned benefits. There are existing methods that can predict and prevent errors, or detect and correct errors. But there is a need for a unified approach to handle timing errors in near-threshold regime.

In this thesis, dynamic speculation based error detection and correction is explored in the context of adaptive overscaling. Apart from error detection and correction, some errors can also be tolerated, or in other words, circuits can be pushed beyond their limits to compute incorrectly to achieve higher energy efficiency. It involves extensive study of design at gate level to understand the behaviour of gates under overscaling of supply voltage, bias voltage and clock frequency (collectively called as operating triads). A bottom-up approach is taken: by studying trends of energy vs error of basic arithmetic operator at gate level to architectures like VLIW (Very Long Instruction Word) processors with multiple arithmetic operators. Based on the profiling of arithmetic operators, tool flow to estimate energy and error metrics for different operating triads is proposed. A model has been developed for the arithmetic operators to represent the behaviour of the operators for different variability impacts. This model can be used for approximate computing of error resilient applications that can tolerate acceptable margin of errors. This method is further explored for execution unit of $\rho - VEX$ VLIW processor. Based on the study, it is possible to give a quick estimation of energy and error metrics of a benchmark by a simple compilation of a C program.

In the context of voltage scaling and impact of variability in sub-nanometer designs, the prime objectives of this thesis work are to address the following

- adaptive error detection and correction scheme for Dynamic Voltage Scaling
- handling timing errors at near-threshold regime to optimize energy efficiency and performance trade-offs
- extending the near-threshold and error handling methods for VLIW systems.

2.5 Contributions

The contributions of this thesis work are as follows.

1. A dynamic speculation based adaptive overclocking and error correction for pipeline logic path is proposed. Based on slack measurement and shadow register principle, this method can identify the opportunity to overclock the system dynamically, and at the same time can respond to variability effects by underclocking the system

based on dynamic speculation. If there are timing errors due to variability effects, inbuilt correction mechanism handles the errors without executing the instruction again. Unlike Razor methods, additional buffers are not required for shorter logic paths. Proof of concept is developed using a Xilinx Virtex-7 FPGA. Results show that a maximum of 71% overclocking is achieved with a limited area overhead.

2. Energy efficiency and accuracy of different adder and multiplier configurations are characterized using several operating triads. A framework has been formulated to model the statistical behaviour of arithmetic operators subjected to voltage overscaling that can be used for approximate computing at the algorithmic level. Simulation results show that a maximum of 89% energy reduction is achieved at the tolerable output bit error rate of 20%.

3. A framework is proposed to estimate the functional level energy consumption of VLIW architecture. Functional Units (FU) of the VLIW are characterized for different operating triads. An estimation tool-flow has been developed to analyze benchmark programs and quickly estimate energy and error metrics for different operating conditions. Quick estimation is possible by simple C compilation of benchmark programs using VEX and C compiler. This estimation can be used to configure the operating triads dynamically to achieve a better trade-off between energy efficiency and accuracy of the computing. Proof of concept is developed with $\rho-VEX$ VLIW processor based on the proposed framework. Execution unit of $\rho-VEX$ VLIW processor is characterized for different operating triads and functional level energy consumption is estimated using the proposed framework.

2.6 Organization of the Thesis

The thesis is organized as follows. Chapter 3 discusses timing errors and methods to handle errors. Error handling methods are reviewed briefly under the classification of (1) Predict and Prevent errors, (2) Detect and Correct errors, and (3) Accept and Living with errors. Advantages and limitations of the existing error handling methods are explained.

In Chapter 4, impact of overclocking in FPGA is discussed with experimental results. Double sampling and slack measurement methods are explained as the basis for

the contribution later described in the chapter. Proposed Dynamic Speculation Window architecture and Adaptive Over/Under-Clocking feedback loop are explained in detail. FPGA implementation of the proposed method and results are further explained to highlight the advantages of this method. At last, some of the related works are compared and contrasted with the proposed method.

In Chapter 5, approximate computing is introduced, and some of the existing approximate arithmetic operators are discussed. Later in the chapter, characterization of arithmetic operators and behaviour of arithmetic operators in near-threshold regime are explained. Rest of the chapter deals with modelling of approximate arithmetic operators based on the characterization of energy and error metrics. Experimental results on benchmarks are discussed in detail to validate the proposed methodology.

In Chapter 6, basic introduction to $\rho - VEX$ VLIW processor is given as the basis for the contribution. Existing methodologies to estimate instruction level energy in VLIW are discussed. Later in the chapter, characterization of functional units for different operating triads and the proposed energy estimation framework along with validation procedure of the proposed energy estimation method are explained. As proof of concept, energy and error estimation in $\rho - VEX$ processor is discussed based on the proposed framework. Experimental setup and estimation tool-flow are explained and results are discussed.

The thesis concludes with Chapter 7 by collating the inferences derived from the previous chapters. An overall picture of energy efficiency versus error tolerance in the context of variability impacts and handling timing errors in design based on Near-Threshold computing, Approximate Computing, Dynamic Voltage Scaling, and FDSOI is provided. Finally, some perspectives on future work are also outlined.

Chapter 3

Handling Variability and Timing errors

Variation in process, voltage, and temperature of CMOS technology results in unfaithful computation. In Chapter 2, variability in CMOS was introduced. In literature, different methods are proposed to mitigate variability at technology level, design level, and software level [33]. Errors due to variability effects are broadly classified as *Hard errors* and *Soft errors*. Hard errors are caused by defects in design or process of the circuit. These errors are repetitive in nature and make the circuit fail in the same manner for a given design and process parameters. Soft errors are caused by striking of physical particles resulting in data corruption. Soft errors are most commonly observed in memory components, resulting in invalid data being read or processed. But soft errors in FPGAs may change the functionality of the programmed FPGA and result in catastrophic failures. Impact of variability in voltage scaling leads to hard errors known as *Timing errors* and increases soft error vulnerability. As shown in Fig. 3.1, digital designs are time constrained with an acceptable positive slack to strike an optimum balance between performance and variability effects. When a design is exposed to variability effects, increase in propagation delay results in timing failure. The operating frequency of the design has to be reduced to lower the impact of variability at the cost of reduced throughput. The impact of variability is more severe in designs with lower-power techniques like DVS, AVS, etc.

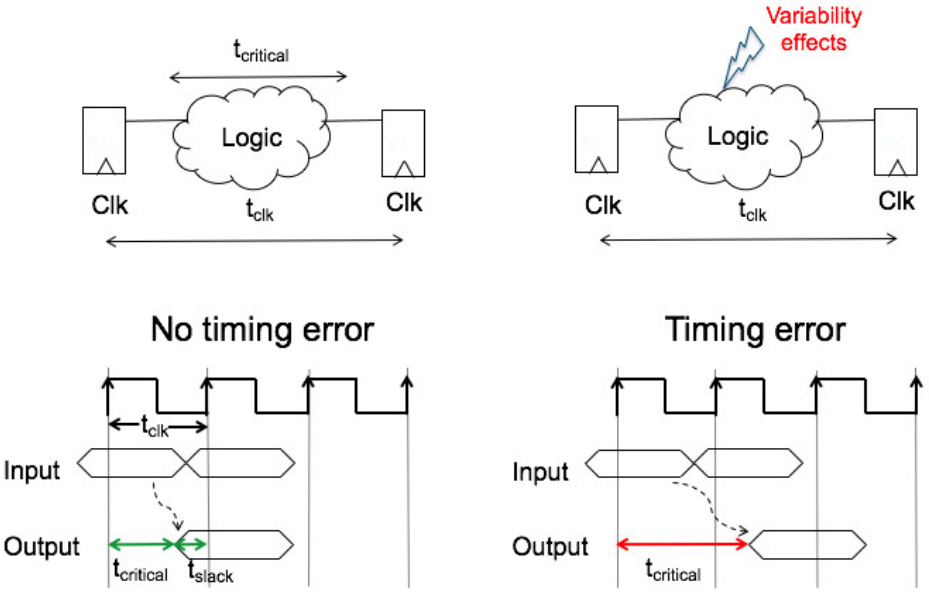


FIGURE 3.1: Timing errors due to the variability effects

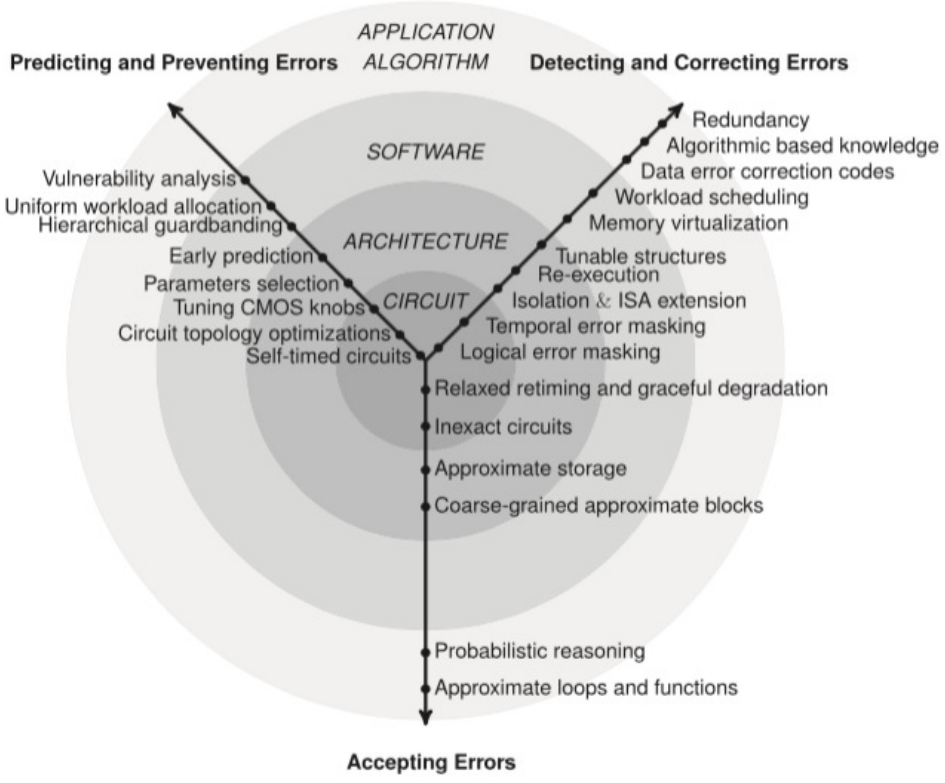


FIGURE 3.2: Taxonomy of timing error tolerance at different abstractions [33]

3.1 Handling Errors

As the supply voltage of the design gets scaled down, the failure rate keeps increasing, which demands reliable and fault-tolerant systems that can work faithfully with above-mentioned low-power techniques. Fig. 3.2 shows taxonomy of timing error tolerance at different stages of abstraction. Handling error can be performed in three different dimensions:

- Predicting and Preventing Errors,
- Detecting and Correcting Errors,
- Accepting and Living with Errors.

Above dimensions of handling errors are applied at circuit, architecture, software, and application levels. This work primarily targets handling errors at circuit and architecture levels. Most common way of handling the error is by predicting the error, thereby preventing it by employing necessary safeguards. For example, designing a circuit with large positive slack to handle timing errors falls under this category. This way of handling errors is more optimistic and takes a toll on the performance of the circuit. In the past few years, the trend is to detect and correct errors rather than predicting and preventing errors. Methods like double sampling popularized this concept [4]. By pushing the performance of the design to point of first failure (POFF) thereby improving the energy efficiency. When an error is detected, error correction mechanism is invoked to correct the errors at the cost of reduced performance. These methods demand additional hardware to detect and correct errors. Finally, in the era of IoT (Internet of Things), accepting and living with errors is the trend of the day. To improve energy efficiency, operating constraints of the circuit are pushed beyond the limits thereby deliberately allowing errors to happen and living with those errors up to an acceptable margin. Approximate Computing or Stochastic Computing for error resilient applications exhibit this dimension of handling or living with errors. Based on the application, design environment, and end-user requirements, error handling techniques are employed.

In the following sections, some of the existing error handling techniques under each dimension are discussed.

3.2 Predicting and Preventing Errors

Predicting and Preventing Errors is a class of error detecting method in which the timing errors are predicted using additional hardware and preventive measures are taken to avoid those errors. Following sub-sections explain the techniques used in this class of error detection methods.

3.2.1 Critical Path Emulator

In [1], an error prediction and prevention scheme based on critical path emulation has been proposed. Fig. 3.3 shows the speed detector which contains three logic paths constrained between registers. The three logic paths are, 1) replica of the critical path of the circuit (CP), 2) same critical path with 3% additional delay ($CP+$), and 3) the reference path. A Test Data Generator emits test pattern and the output data comparator matches the data from all the three logic paths. As the reference path always outputs correct data even at the low V_{DD}, outputs from the other two paths are compared with reference path. When both the CP and $CP+$ are wrong, then the supply voltage is increased, if both the CP and $CP+$ are correct, then the supply voltage is decreased. The supply voltage is left unaltered when only CP is correct and $CP+$ is wrong. This method needs additional hardware to generate and compare the test data. Also, this method can not be used for reconfigurable architectures where the critical path changes based on the functionality of the design.

3.2.2 Canary Flip-Flops

Canary Flip-Flops are used in voltage scaling methods to observe the proximity of timing errors due to the effect of scaling the supply voltage. It is a common practice to reduce the V_{dd} of a datapath during standby state (inactivity state in which supply voltage is lowered to required minimum retention potential). Registers and other storage elements are very sensitive to voltage scaling. When the V_{dd} is reduced below the data retention potential of the registers, data stored in the registers get corrupted. Different variability effects directly impact V_{dd} and cause sudden sags in the supply voltage. To predict and prevent such errors in pipelined datapaths, canary flip-flops are used along with datapath registers [34]. Fig. 3.4 shows the schematic of canary based error prediction

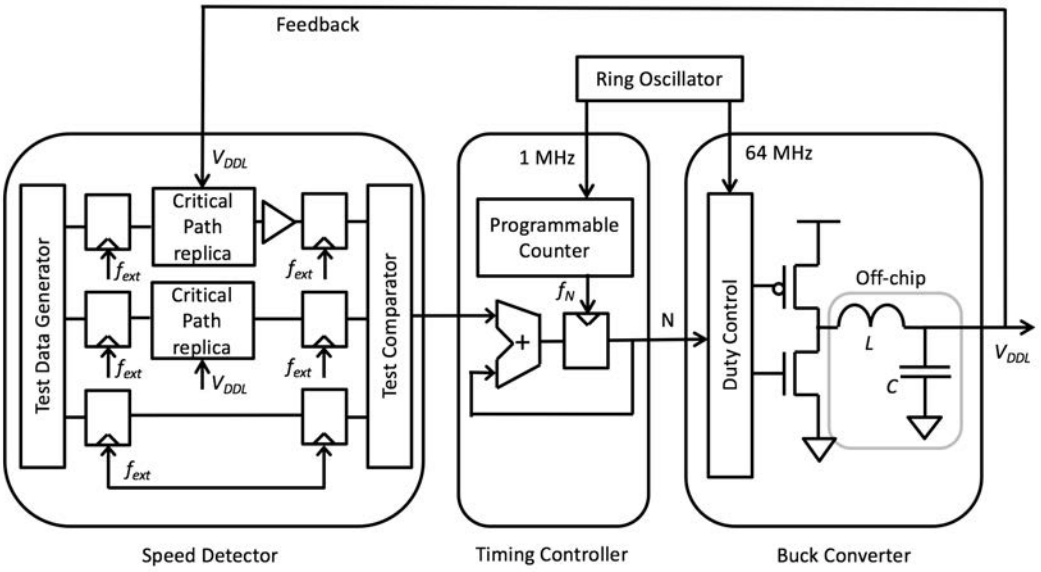


FIGURE 3.3: Critical Path Emulation based error detection [1]

and prevention of errors. Canary Flip-Flops are designed to fail at higher V_{dd} compared to the registers in the datapath pipeline during standby mode. As shown in Fig. 3.4, Canary Flip-Flops can also be designed like Razor methods [35] with an additional delay before the shadow FF. Unlike Razor, where the error is detected and corrected, Canary FFs predict and prevent timing errors. When V_{dd} is scaled below the safe margin, canary flip-flops fail before the pipeline registers and send an *alert* signal to prevent the failure in the circuit. Canary flip-flops are designed to fail at higher V_{dd} by proper sizing of master-slave flip-flops (MSFF) or linear feedback flip-flops (LFBFF). A closed-loop control is used to sense the feedback from the canary flip-flops and control the standby voltage to prevent corruption of data in pipeline registers. Canary flip-flops with different failure voltages, help to identify large process variations within the die, thereby V_{dd} and V_t variations can be circumvented. Based on the design profiling, the user determines the granularity and placement of canary flip-flops. Unlike Razor, Canary logic does not need shadow clock and dedicated clock tree for shadow clock. Though this method offers a safe way to scale the standby voltage, the usability of this method is limited to circuits with longer sleep than active time. In [36], limitations of Canary logic based DVS are discussed.

Prevent and predict methods are more suitable for designs with relaxed area and

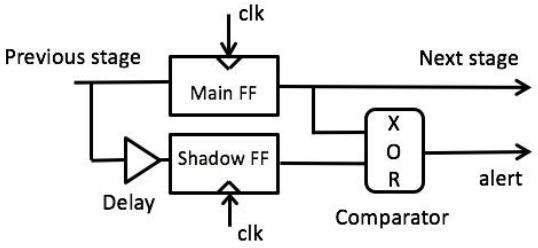


FIGURE 3.4: Schematic of Canary Flip Flop based error prediction

power constraints. Critical path emulation method demands substantial additional hardware to predict the possibility of errors. Due to Inverse Temperature Dependence (ITD), when the supply voltage is scaled down, a non-critical path can become a critical path at certain temperature [15]. This makes the whole setup to be re-calibrated for different critical paths. In Canary logic, there is no guarantee that canary based DVS can prevent timing errors in main FF. This limitation made way for the second dimension of error handling *Detecting and Correcting Errors*.

3.3 Detecting and Correcting Errors

3.3.1 Double Sampling Methods

When the supply voltage of a logic is reduced, the propagation delay of all the components in that logic path will increase resulting in wrong data getting latched in the registers for the given timing constraint as shown in Fig. 3.1. Double sampling techniques are used to detect and correct errors in the pipelined architectures by sampling the output at different timing instance. In literature, different techniques are proposed [4] for double sampling. In the following sub-sections, few of the prime double sampling techniques are discussed.

3.3.1.1 Razor I

Razor I is a timing error recovery technique used in pipelined architecture, where pipeline registers are accompanied by a shadow latch as shown in the Fig. 3.5 [35]. A delayed clock signal (*Shadow clk*) triggers the shadow latch, which double samples the output. The output from both the main flip-flop and shadow latch are fed to an XOR gate which

generates an error signal if there is a difference in the outputs. The error signal from XOR gate in every such logic path is fed to a multi-input XOR gate, which generates the error signal for the whole circuit as shown in Fig. 3.5.

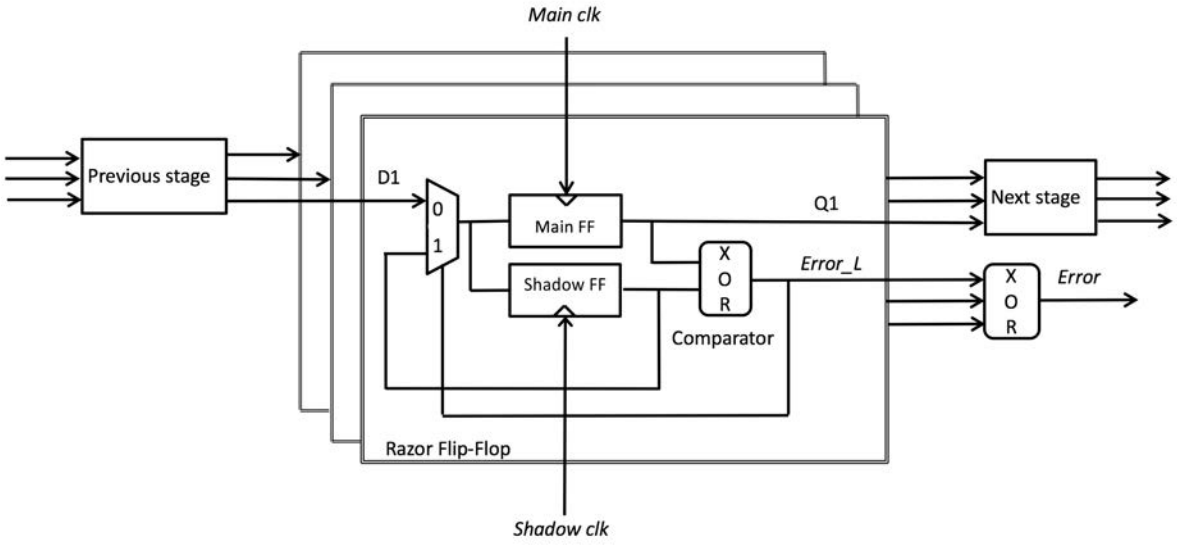


FIGURE 3.5: Razor I based error detection [35]

The time interval between main and delayed clock edges is called *speculation window* denoted by T_{sw}. When the propagation delay of the constrained path T_p is less than the sum of T_{clk} and T_{sw}, there is no error in the logic path. When $T_{clk} < T_p < T_{clk}+T_{sw}$ due to the scaled down supply voltage the main flip-flop latches the wrong data, while the shadow latch holds the correct data due to the extra speculation time. An XOR gate will assert the error flag and the correct data in the shadow latch is copied to the main flip-flop. There are some limitations in this method in determining the speculation window. When the propagation delay T_p is less than the speculation window T_{sw} the shadow latch samples the value before the main flip-flop and asserts the error flag thus generates a false alarm. To prevent data corruption due to this limitation, minimum-path length constraints have to be added at each Razor flip-flop input. This results in the addition of buffers to slow down fast paths, which therefore increases power and area overhead. In re-configurable systems, this effect will be in the order of magnitude [35]. The shadow latch can only fix timing faults not soft errors like Single Event Upset (SEU) and Single Event Transients (SET) which are caused by ions or radiation. In the Razor I method, the error rate increases exponentially beyond the point-of first failure.

In [5], a similar approach to detect timing errors in FPGA-based circuits is proposed. In this work, FPGA circuits are subjected to overclocking and the timing errors in critical paths due overclocking are detected by shadow registers added to the critical paths. The prime objective of this work is to monitor the effect of overclocking and to decide how much overclocking can be applied with respect to the errors detected by the shadow registers. This work outlines the scope of tolerating timing errors within a margin to gain improvement in performance by overclocking. There is no scope for error correction in this work.

3.3.1.2 Razor II

In order to avoid the above-mentioned limitations of Razor I, Razor II is proposed in [37]. In Razor II, only error detection is performed using shadow flip-flops and the error correction is performed using architecture replay (the last instruction is executed again to obtain correct data). This makes the Razor II architecture simple and less area consuming. But the architecture replay based error correction reduces the throughput of the design. Razor II is also capable of detecting SER errors due to radiation. Fig. 3.6 shows the schematic of Razor II latch and its internal components like clock generator, Transition Detector (TD) and Detection Clock (DC) generator. As shown in Fig. 3.7, when the input data transitions occur after the rising edge of the clk, the shorter negative pulse on DC suppresses the TD and the error flag is not asserted. However, during the speculation window, if the input data transitions occur after the rising edge of the clk, the TD detects the transition in the node N and results in assertion of the error signal and instruction roll-back is carried out. The speculation windows of Razor II is from the rising edge of the DC to falling edge of the clk. This can be extended at the cost of more buffers to tackle the hold constraints. Razor II can also detect soft errors in memory elements or propagated from logic to latch, as the TD is active during both the phases of the clk. But when the DC is low, there is a chance that soft errors may not be detected since the DC suppresses the TD.

Razor was designed for pipelined microprocessors, but it has never achieved commercial success due to its overhead in fault correction in complex and high-frequency instruction pipeline logic [5]. This is because the pipeline has to be stalled whenever the error has to be corrected. All the existing error detection and correction methods

use fixed speculation window for the double sampling and overclocking of the system without any adaptive feedback to tackle the variability effects.

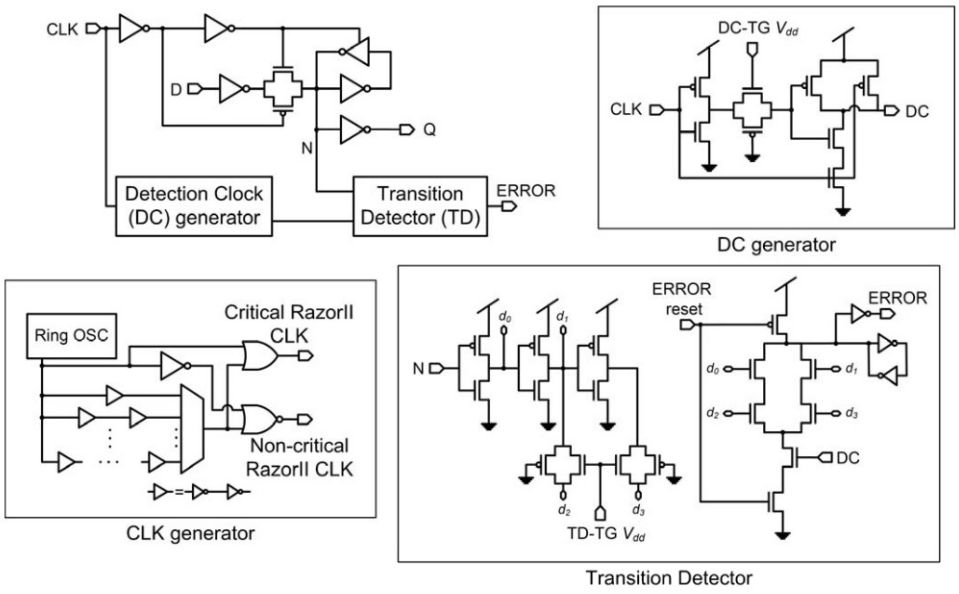


FIGURE 3.6: Schematic of Razor II latch and its internal components [37]

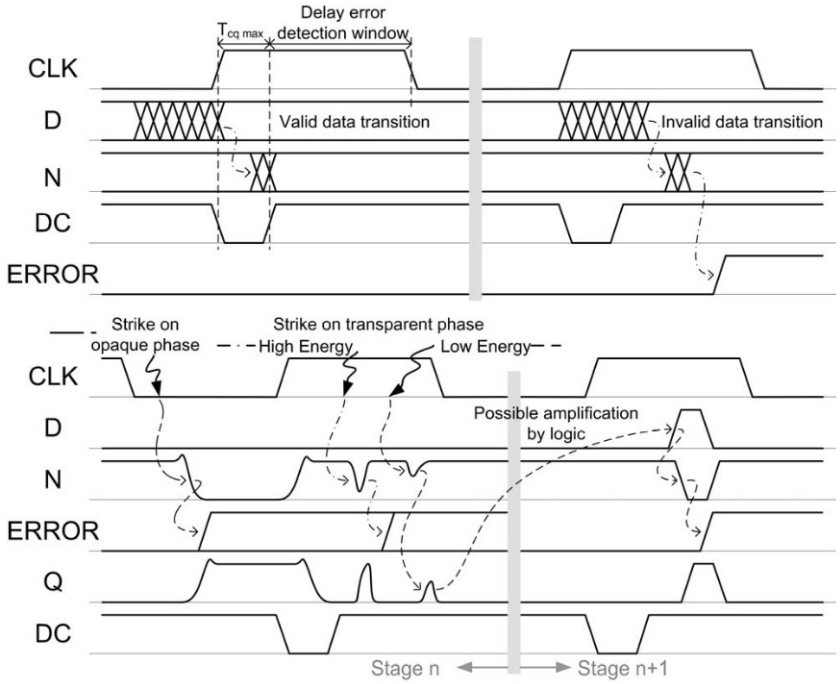


FIGURE 3.7: Razor II timing plot [37]

3.3.2 Time Borrowing Methods

Time borrowing methods are another form of double sampling technique where latches are used in place of registers. By taking the advantage of level triggering of latches, timing error generated at one stage of the pipeline will be corrected by borrowing time from the next stage of the pipeline. This type of local correction mechanism prevents error from propagating across the pipeline.

3.3.2.1 Bubble Razor

As explained above, both the Razor methods have limitations due to timing issues. They achieve high performance at the expense of long hold time which may lead to race failure. Similarly, implementation of these methods is limited by architectural invasiveness. In [38] an error detection technique based on two-phase latch timing and local replay mechanism is proposed to overcome the limitations faced by both the Razor methods. A conventional flip-flop based datapath can be converted into a two-phase latch based datapath as shown in the Fig. 3.8. The logic delay in each phase is balanced such that no time borrowing occurs during error-free operation. If the data arrives after the latch opens due to variations or voltage scaling, Bubble Razor asserts error flag, which guarantees that the error is caused by long paths taking more than a clock phase instead of by short paths. With two-phase latches, when one latch is opening the latch in the preceding stage is already closed. This feature of Bubble Razor provides more speculation window and breaks the dependency of speculation window on short path constraints unlike in Razor I and II. Bubble Razor uses *time borrowing* as a correction mechanism by which architecture modifications and replaying instructions are avoided. Because of its local correction mechanism, the function of the whole chip does not need to be stalled. Fig. 3.9 shows the error detection and propagation of bubbles. A failure will occur when data arrives after a latch closes, which can arise if the time borrowing effect is not corrected and compounds through multiple stages. Upon detection of timing error, error clock gating control signals called *bubbles* are propagated to neighboring latches. A bubble causes a latch to skip its next transparent clock phase, giving it an additional cycle for correct data to arrive. With two-phase latches, if a latch stalls, data is not immediately lost because its neighboring latches operate out of phase. In order to not lose data, neighboring latches must stall one clock phase later. Therefore half a clock

cycle to propagate bubble and another half a cycle to correct the error. To prevent the bubbles from propagating indefinitely along loops, a bubble propagating algorithm is proposed in [38]. 1) A latch that receives a bubble from one or more of its neighbors stall and sends its other neighbors (input and output) a bubble half-cycle later; 2) A latch that receives a bubble from all of its neighbors stall but does not send out any bubbles as shown in Fig. 3.9. Multiple errors during the same cycle will cause multiple bubble stalling sequences to take place at the same time, but when stall events collide, they combine and are corrected in a single stall cycle, reducing correction overhead.

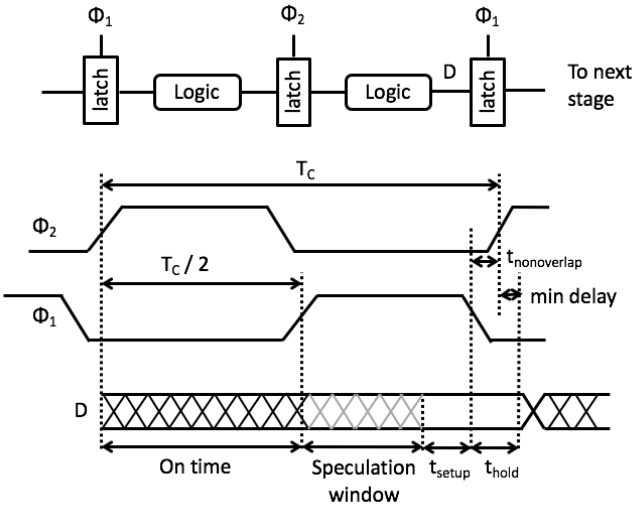


FIGURE 3.8: Schematic of Bubble Razor method [38]

3.3.3 GRAAL

Global Reliability Architecture Approach for Logic (GRAAL) is a latch based error detection method used for pipelined architectures [39]. Compared to Razor methods and other double sampling error detection methods where there is a penalty in duplicating hardware or execution, GRAAL compares the result of a single computation observed at two different instants. In this method, the logic constrained between flip-flops of the pipelined architecture is partitioned into two equal slices and the slave latch of the flip-flop is moved within the combinational logic by *retiming*. Two latches (masters and slave) in every flip-flop of the conventional pipeline architecture clocked by non-overlapping clocks $\phi 1$ and $\phi 2$ as shown in Fig. 3.10 . Thus, when one slice is computing,

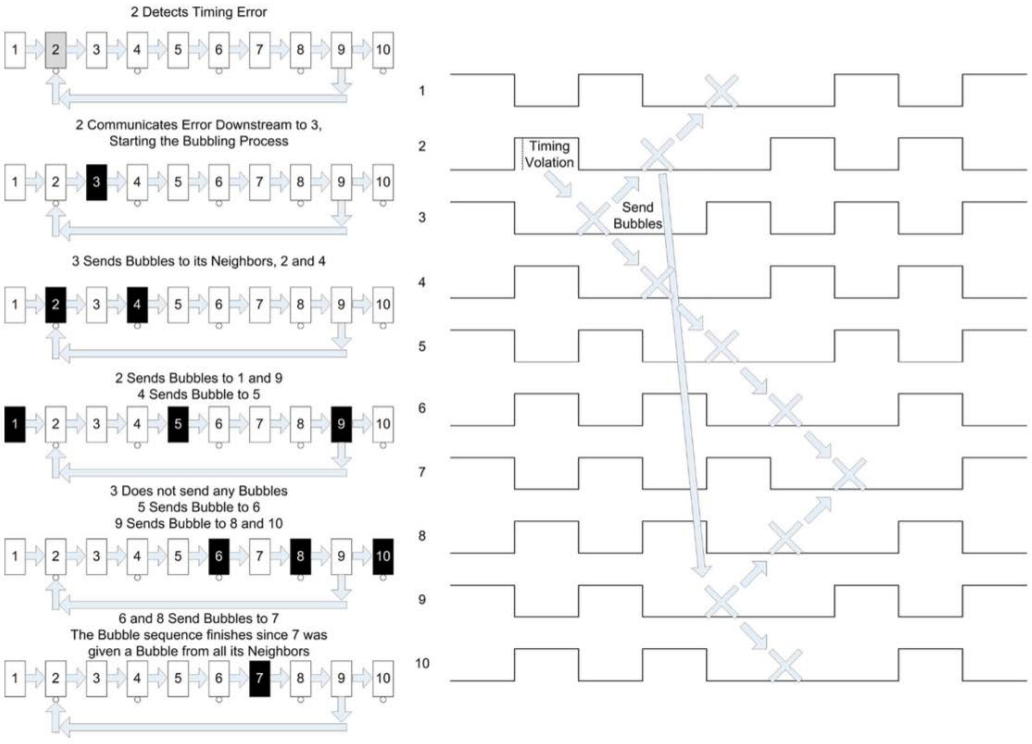


FIGURE 3.9: Bubble Razor timing plot [38]

the adjacent stage is idle and its outputs are stable, enabling checking for temporary-faults. This is not possible in flip-flop based design where the computations happen simultaneously. An efficient tool is required to partition the combination logic into two slices, such that the critical path of both the slices are equal and their sum is equal to the delay of the un-partitioned combinational logic. For maximizing the fault tolerance efficiency, clocks $\phi 1$ and $\phi 2$ have significant non-overlapping period such that each stage will maintain stable values. In [39], duty cycle of $\phi 1$ and $\phi 2$ are 25% leaving another 25% of non-overlapping period between them. The frequency of the clock has to be designed in a way such that the falling edge of the clock $\phi 1$ occurs after the output from the combination circuit is stable and the delay is denoted as D as shown in Fig. 3.11, where δ is the minimum delay of the block. Therefore it is possible to compare the output of combinational blocks $CC1$ and $CC3$ against the outputs of the respective latches $L1$ and $L3$ until the time period equal to $0.25T + \delta$, which is from the falling edge of $\phi 1$ to δ time after the raising edge of $\phi 2$. There is no extra shadow latch required to detect the error, also the speculation window is much wider compared to Razor methods, which results in improved fault tolerance efficiency. In order to correct the errors, as shown in Fig. 3.11, redundant flip-flops and MUXs are used like in Razor methods to memorize

and restore the original data.

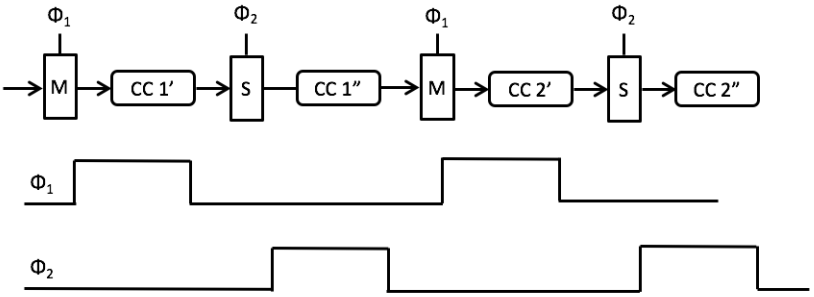


FIGURE 3.10: Schematic of latch based GRAAL method [39]

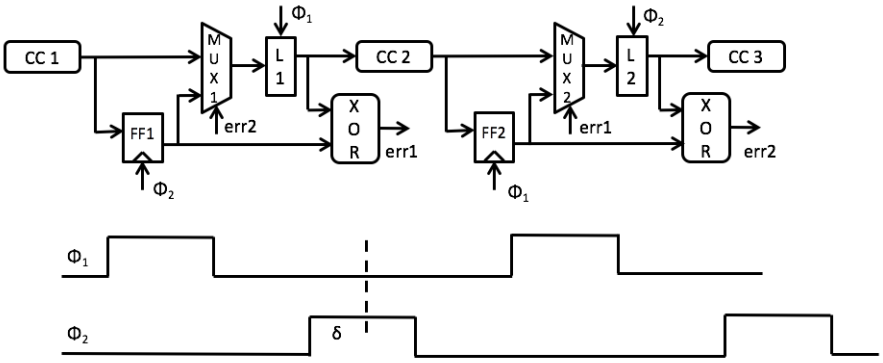


FIGURE 3.11: GRAAL error correction using shadow flip-flop [39]

Compared to register based double sampling methods, latch based timing borrowing methods provide wider speculation window to detect more timing errors. Though time borrowing methods reduce the need for additional hardware to detect and correct errors, specialized tool flow to obtain retimed circuits makes it more complicated. Also, complicated bidirectional bubble propagation mechanism can outweigh the benefits of energy saving in the context of near-threshold regime.

3.3.4 Redundancy Methods

Redundancy methods are used in digital circuits to repeat the same function or instruction multiple times and based on voting mechanism, majority result will be considered as a valid result of the computation. Redundancy methods are commonly used to detect and correct timing errors due to transient faults induced by radiation and other variability effects [40]. Redundancy can be achieved at *hardware* level by duplicating the

functionality by additional hardware to execute same instruction multiple times in parallel or at *software* level by repeating the same instruction multiple times in the same hardware. The *software* level redundancy is also called time redundancy. Unlike the above-discussed Razor methods, where the pipeline registers/latches are duplicated to achieve tolerance towards timing errors, in hardware redundancy, entire processing elements (PE) are duplicated. Since the context of this thesis is limited to hardware-based methods, in the below sub-sections common hardware-based redundancy methods for error detection and correction are discussed.

3.3.4.1 Flexible Error Handling Module (FEHM) based Triple Modular Redundancy (TMR)

TMR is a hardware redundancy method, in which the functionality of a PE is duplicated in two additional PEs. Based on voting mechanism the valid output is determined. This type of hardware redundancy methods are most suitable for reconfigurable platform, due to the availability of unused hardware to achieve redundancy [41]. In [42] a low cost TMR for reconfigurable platform is proposed. In this method, Functional Units (FU) of neighbouring PEs are clubbed together to execute an instruction. Output of these FUs are passed to FEHM to determine the valid result as shown in Fig. 3.12.

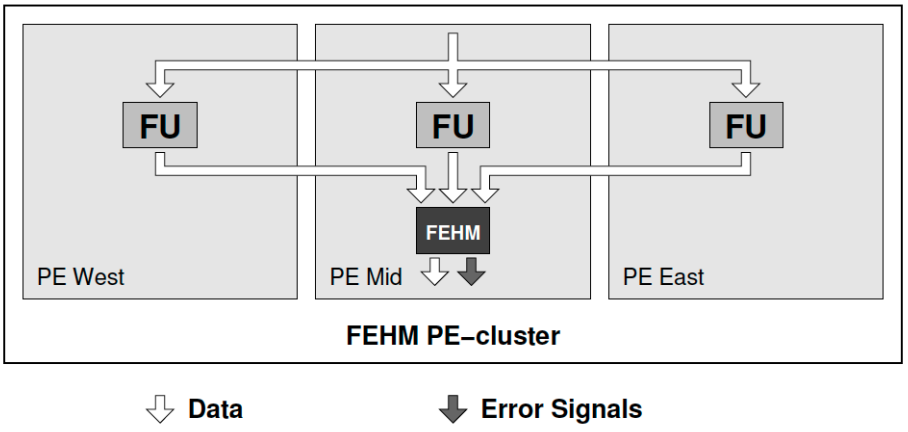


FIGURE 3.12: TMR based on FEHM [42]

Fig. 3.13 shows the schematic of Flexible Error Handling Module. In the Data error detection unit, outputs of all the channels are compared and error signal is sent to Transition error detection unit and Voter mechanism. Voter mechanism selects the

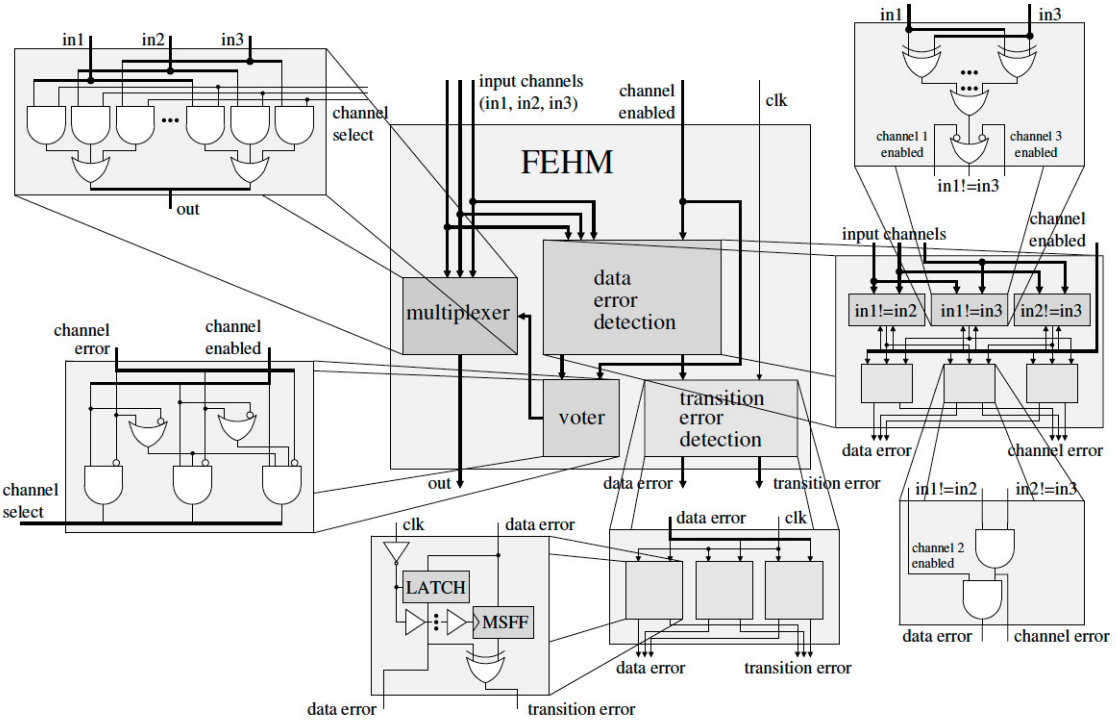


FIGURE 3.13: Schematic of FEHM block [42]

valid output out of the three channels. In [41], this method is further improved by run-time reconfiguration to dynamically control the number of spare FUs needed for fault-tolerant application mapping. Also, timing constraints and conditions to avoid false errors in timing error detection in CGRAs are elaborated in [43]. The primary drawback of this method is substantial hardware consumption which consumes more power. Also, this method has no scope in processor designs due to the unavailability of resources like in reconfigurable platforms.

3.3.4.2 Diverse Double Modular Redundancy (DDMR)

In conventional modular redundancy, Common Mode Failures (CMFs) like voltage drop and global variability effects affect all the redundant modules equally and affects the error detection capability of these methods. Design diversity is used to resolve this problem by implementing same functionality by different structural approaches. Design diversity can be introduced at different abstractions of IC design flow. At gate level using different gates to implement the same functionality, at architecture level adopting different architectures for the same functionality, and so on [44]. In [45], existing DDMR

solutions are discussed in details for DSP (Digital Signal Processor) and arithmetic circuits.

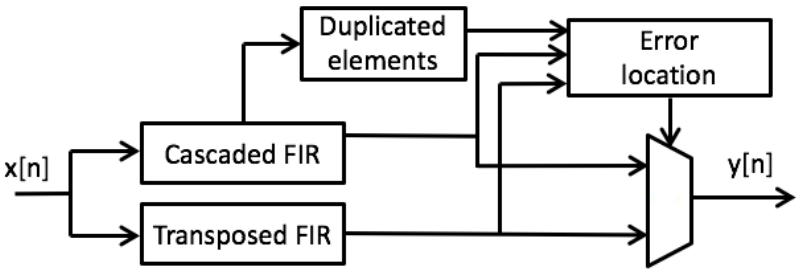


FIGURE 3.14: DDMR based FIR (Finite Impulse Response) filter [46]

Fig. 3.14, shows the DDMR based FIR filter. In this method, two diverse FIR (Finite Impulse Response) structures namely Cascaded and Transposed FIR, are used to perform the same functionality. Both the filters are diverse by architecture. The error detection mechanism is used to determine the correct output from two diverse designs. In this method, both structures have to be designed in a way their error profiles are mutually exclusive. Based on the profiling, error detector understand the nature of the error and selects the valid output from either of the structures.

The nature of hardware redundancy method is to maintain the accuracy of the computation by consuming substantial addition of hardware resources. In both FEHM based TMR, and DDMR methods are suitable for large reconfigurable devices with plenty of hardware resources to duplicate the functionality. These methods are not suitable for smaller designs (in the context of IoT) which demand higher energy efficiency at the cost of acceptable accuracy loss.

3.4 Accepting and Living with Errors

3.4.1 Inexact Computing or Approximate Computing

The final dimension of error handling is to tolerate and live with errors. Human perception levels are limited in the domain of vision and hearing between a minimum and maximum thresholds. When the errors are allowed beyond those thresholds, there will not be much compromise in terms of perceiving the quality of the image, video, and

audio. Therefore limitation in human perception level can be used to allow errors in the computation or to design inexact hardware to perform approximate computing. This provides two advantages: 1) no need for additional hardware to detect and correct errors, and 2) energy and power savings can be achieved by trading the performance and accuracy. Approximate computing can be achieved by inexact hardware and/or by probabilistic approach in software and architecture. In [47], probabilistic approach in the context of device modelling and circuit design is discussed in detail. A probabilistic CMOS inverter can be described as erroneous with a probability p and correct with a probability *1-p*. In contrast to the deterministic CMOS inverter, the probabilistic inverter exhibits wrong switching with respect to the probability of error p, where p denotes the impact of all variability issues. A statistical model of the device is created by knowing p. The probability of failure p is determined by varying V_{dd}, noise sources and other device parameters. In [47], a 32-bit Conditional Carry Select (CCS) adder is constructed with different V_{dd} for MSBs and LSBs to reduce power consumption. For a pixel error in an HD Frame (1280 x 720) video streaming, 40% power reduction is claimed.

Pruning based approximation in real-time video processing is proposed in [48]. Computation complexity of a direct Cholesky-decomposition-based solver in real-time is very demanding. Based on observation, many elements in the decomposition can be pruned with reduced accuracy is proposed. This results in 75% reduction in the number of operations per solve, yielding benefits in terms of energy, computation time, and area. In [49], a brief review of recent developments in approximate computing at different abstractions from programming language to transistor level are discussed. Usability and feasibility of approximate computing is determined by error metrics. Different error metrics to determine the impact of approximate computing are also described in [49].

Approximations based on pruning methods are more rigid due to the removal of some hardware to reduce the energy cost. This method lacks the dynamicity to switch between different energy-accuracy trade-off points. On the other hand, methods based on multiple voltage V_{dd} offer a better trade-off between energy efficiency and accuracy but needs additional hardware like level-shifters to bridge the modules with different supply voltage.

3.4.2 Lazy Pipelines

In [50], approximate computing technique based on Voltage Over-Scaling (VOS) is proposed to improve the power efficiency and reduce error rate by utilizing vacant cycles in VOS functional units. Based on voltage over-scaling and exploiting unutilized execution cycles of FUs, Lazy Pipeline architecture is proposed. In this architecture, every FU (*precise FU*) in the pipeline is accompanied by an another FU (*imprecise FU*) that is operated at lower V_{dd}. Based on the required accuracy level definition in the instructions, computations can be either performed by precise or imprecise FU. Set of FUs for each precision level is recommended instead of scaling V_{dd} of a single FU. For an application with n different precision levels a set of n FUs, each computing at a specific precision, is needed. This increases the hardware consumption and complexity of the design. Based on profiling of vacant cycles (slack), FU with reduced V_{dd} that can match the available slack is used to compute the instruction.

In addition to hardware complexity, ISA (Instruction Set Architecture) of the architecture also needs to be extended with precision marking for instructions to choose an FU out of the set of FUs with different precision levels. The set of potential instructions that can be subjected to approximate computing has to be determined offline. This reduces the scope of dynamicity and quick adaptation to architectural changes.

3.5 Conclusion

In this chapter important insights about various error detection and correction methods are provided. The chapter starts with an introduction to timing errors and taxonomy of timing error tolerance at different levels of abstractions. Three major classifications of error handling techniques are presented namely, 1) Predict and Prevent, 2) Detect and Correct, and 3) Accept and Live. Predict and Prevent methods are suitable for designs with longer sleep time and more relaxed area and power constraints. This limits the use of these methods for design with ultra-low power techniques like DVS. Different detect and correct techniques based on double sampling and timing borrowing are explained in detail. The major limitations of the double sampling methods are fixed speculation window, need for additional hardware like buffers, transition detectors etc., and complex error correction methods like generation and propagation of bubbles. Likewise, time

borrowing method needs specialized tool flow to retime the design. Redundancy methods like TMR and DDMR are only suitable for reconfigurable architecture with plenty of unused resources. Finally, this chapter discusses the scope for accepting and living with errors. Applications based on probabilistic computing can tolerate some margin of error without compromising much of the output quality. Methods like pruning based approximation, inexact computing with multiple V_{dd} are discussed and their limitations are highlighted. Therefore, there is a need for a unified approach to handle errors based on the application and end-user need. The required method should be able to detect and correct errors at the same time, must give-up some margin of accuracy to gain more in terms of energy based on user preference. Finally, the method should accommodate different ultra-low power techniques like DVS, and Near-Threshold Computing to achieve enhanced energy gains. In the rest of the thesis, we propose methods and frameworks to meet these requirements and to overcome the mentioned limitations of the existing methods.

Chapter 4

Dynamic Speculation Window based Error Detection and Correction

4.1 Introduction

Dynamic voltage and frequency scaling became the prominent way to reduce the energy consumption in digital systems by trading the performance of the system to an acceptable margin. Effect of scaling coupled with variability issues make highly pipelined systems more vulnerable to timing errors [3]. Such errors have to be corrected at the cost of additional hardware for proper functioning [4]. In Chapter 3, different error handling methods are discussed in detail. The Razor method proposed in [35] is a very popular error detection and correction architecture for timing errors in digital systems, inspired by the more general double-sampling technique [4]. In the double-sampling architecture, an additional sampling element, known as shadow register, is used along with the main output register to sample the output at different timing instances. The shadow registers are clocked by a delayed version of the main clock known as the shadow clock. The timing interval between the rising edges of the main clock and the shadow clock is termed as the *speculation window* (ϕ). The speculation window is used to compare the main and shadow registers to determine the validity of the output.

There is a trade-off between the width of the speculation window and the fault coverage. For a wider speculation window, more errors could be detected and corrected, but this requires more buffer insertion for logic paths that are shorter than the speculation window to avoid false error detection. And vice-versa for a narrower speculation windows. Designs with fixed speculation window optimized for certain operating condition might suffer from an increased error rate under variability issues such as thermal effects. In order to tackle these limitations, a double-sampling technique with dynamic speculation window is required to dynamically vary the speculation window with respect to the variability effects.

Razor methods are not suitable for pipelined microprocessors due to its overhead in fault correction in complex and high-frequency instruction pipeline logic [5]. Razor-like fault detection and correction techniques are used in reconfigurable architectures like Field-Programmable Gate Arrays (FPGA) and Coarse-Grain Reconfigurable Arrays (CGRAs) due to moderate clock frequencies and function-specific pipelines [5], [6]. All the existing error detection and correction methods use fixed speculation window for the double sampling and overclocking of the system without any adaptive feedback to tackle the variability effects.

In the rest of the chapter, the impact of overclocking and different double sampling based error detection and correction methods are discussed. Subsequently, the idea of *Dynamic Speculation* architecture and its advantages are proposed. Further dynamic speculation based adaptive overclocking is explained. Proof of concept of the proposed adaptive overclocking is explained with FPGA implementation. Finally, results of the proposed method are discussed and contrasted with existing error detection and correction in the context of DVFS.

4.2 Overclocking and Error Detection in FPGAs

4.2.1 Impact of Overclocking in FPGAs

In general, FPGA designs can be safely overclocked by a significant ratio with respect to the maximum operating frequency estimated by the FPGA's synthesis and place-and-route tool flow. This gives the advantage of increasing the implementation throughput

without any design-level modifications. As a motivating example, an 8-bit 8-tap FIR digital filter is synthesized and implemented in the Xilinx Virtex VC707 evaluation board as shown in Fig. 4.1. The maximum operating clock frequency estimated by Vivado 2014.4 tool flow, with performance optimized synthesis, is 167 MHz. A test bench is created with an 80-bit LFSR, feeding random input patterns to the design, and the output of the design is monitored using Vivado's Integrated Logic Analyzer (ILA). A functional simulation output obtained from RTL simulation is used to compare the output of the ILA. Onboard IIC programmable LVDS (Low-Voltage Differential Signaling) oscillator is used to provide clock frequency for the pattern generator and the design under test. The supply voltage of the FPGA is not changed in this experiment. XADC system monitor is used to record core voltage and temperature of the FPGA. At the ambient temperature, the clock frequency is increased from 160 MHz to 380 MHz and the output data is recorded for all the frequency steps.

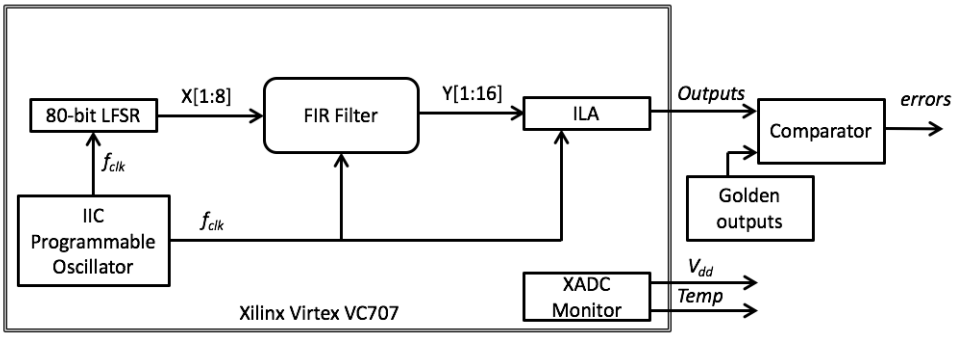


FIGURE 4.1: Schematic of FIR filter overclocking in Virtex FPGA

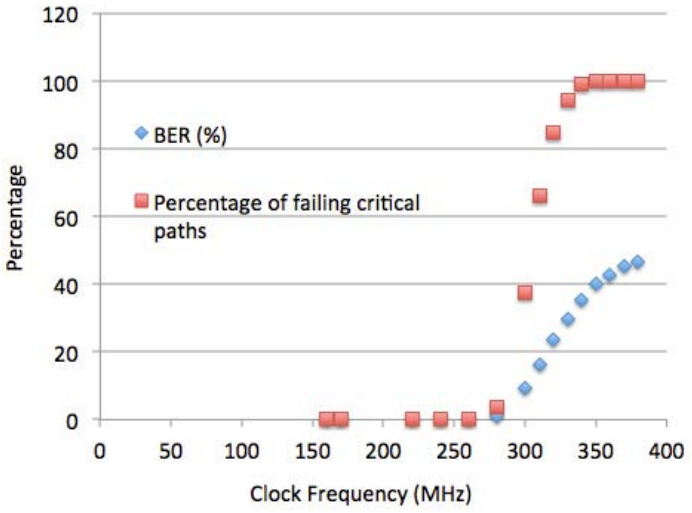


FIGURE 4.2: Timing errors (percentage of failing critical paths and output bit error rate) vs. Overclocking in FPGA configured with an 8-bit 8-tap FIR filter

Fig. 4.2 shows the evolution of Bit Error Rate (ratio of faulty bits over total output bits) at the output of the design with respect to overclocking. In Fig. 4.2, the first plot (orange squares) represents the percentage of critical paths that fail when the FPGA is overclocked. Similarly, the second curve (blue diamonds) represents the output BER (Bit Error Rate). As shown in Fig. 4.2, there are no errors recorded until 260MHz, which gives the safe overclocking margin of 55% with respect to the maximum operating frequency estimated by Vivado, and therefore a performance gain of 1.55×. Beyond this safe overclocking margin, some critical paths in the design start to fail and the output error rate increases exponentially with respect to the increase in the clock frequency. Overclocking up to 300 MHz (×1.8 the operating frequency) leads to BER of 10%, which would be still acceptable for applications that tolerate approximate computations [51]. Around 350 MHz, all critical paths fail and the bit error rate converges to 50%. By employing error detection and correction mechanisms such as double-sampling methods, FPGA designs can be adaptively overclocked to improve performance at run time according to varying operating conditions (e.g., temperature).

4.2.2 Double Sampling

Error detection based on the double-sampling scheme requires less area and power overhead than traditional concurrent detection methods such as Double Modular Redundancy (DMR) [4]. In the double-sampling method, instead of replicating the entire pipeline, an additional sampling element (latch or flip-flop) is added at the output of the pipeline to sample the output data at two different time instances using the main clock (M_{clk}) and a shadow clock (S_{clk}). In Fig. 4.3, a logic path $Path_1$ is constrained between the input register R_1 and the output register Q_1. An additional shadow register S_1 also samples the output data at different timing instance. An XOR gate is used to compare the data from the registers Q_1 and S_1. An error register E_1 is used to record any discrepancies between output data from Q_1 and S_1. The multiplexer M_1 is used to select the valid output for error correction based on error signal from the register E_1. In [52], instead of using a delayed shadow clock, the delay is added to the combinational logic output to prevent the variation due to aging effects. Different versions of double-sampling methods are proposed in [52]. Razor I [35], Razor II [37], Bubble Razor [38], GRAAL [39] are commonly used in digital designs for detecting timing errors and counteracting the effect of dynamic voltage and frequency scaling (DVFS). Bubble Razor [38]

and GRAAL [39] techniques are based on *time borrowing* principle. In these methods, registers are converted into latches and logic retiming is applied to the combinational circuits to utilize time borrowing from the consecutive stage of the pipeline in case of a timing error. In this thesis, register based double sampling is used to avoid design complexity involved in time borrowing methods.

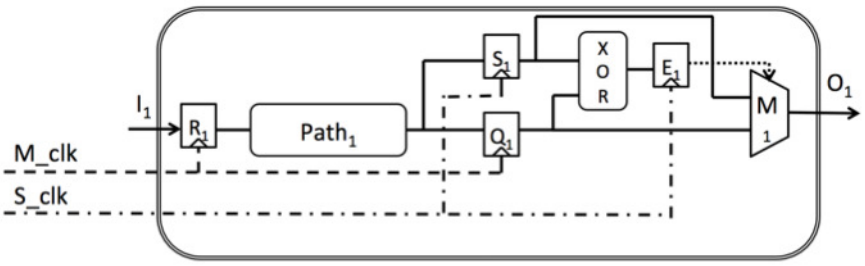


FIGURE 4.3: Principle of the double-sampling method to tackle timing errors

All the doubling sampling techniques use a fixed speculation window and the detected errors are corrected either by copying the valid output from the shadow register or architecture replay technique is used. Due to the fixed speculation window, there is no dynamicity in these methods, which cannot adaptively change the frequency of the design in both directions (overclocking and underclocking) at run time.

Since the shadow clock always lags behind the main clock, when the critical paths in the design fail to meet the timing constraints due to the temperature or other variability effects, the shadow register can detect and correct the errors. However, the fixed speculation window does not give the adaptability to measure the slack and to overclock the design. Also, in methods like Razor [35], additional buffers are needed for logic paths that are shorter than the speculation window, which results in increased area overhead.

4.2.3 Slack Measurement

In [53], [54], and [55], online slack measurement techniques are proposed to determine the delay of the combinational components in the pipeline at run time. An additional sampling register is added to the output of the pipeline, which samples the data before the main output register of the pipeline. As shown in Fig. 4.4, the output of the pipeline $Path_1$ is sampled by both main register Q_1 and shadow register T_1 using the main clock M_{clk} and the shadow clock S_{clk} respectively. Both M_{clk} and S_{clk} are of same clock frequency but with different phases. This method is very similar to the double

sampling technique discussed in the previous subsection. The only difference is that the shadow clock S_{clk} always leads the main clock M_{clk} by a phase difference ϕ. An XOR gate is used to compare the outputs from both main register Q_1 and shadow register T_1. The output of the XOR gate is sampled by error register e_1. To determine the available positive slack of the pipeline $Path_1$, the phase difference ϕ is increased until the error register e_1 records an error. Due to the leading shadow clock, this method is not capable of detecting timing errors in the critical paths that violate timing constraints due to temperature or other variability effects.

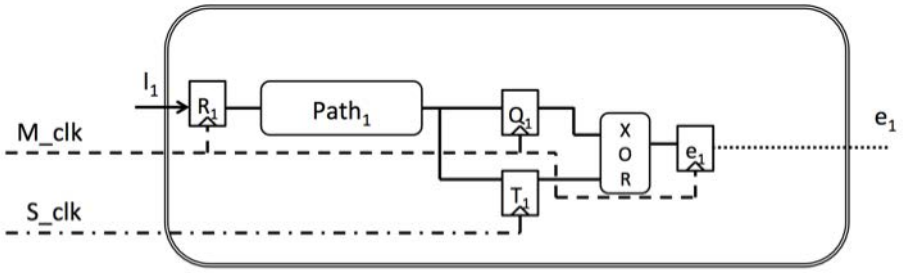


FIGURE 4.4: Principle of online slack measurement for overclocking

4.3 Proposed Method for Adaptive Overclocking

In this thesis, dynamic speculation window combined with the double-sampling method is proposed for error detection and correction and to adaptively overclock or underclock the design based on variability effects. The following sections present the proposed architecture of a dynamic speculation window and the adaptive feedback loop for over-/under-clocking the design.

4.3.1 Dynamic Speculation Window Architecture

Fig. 4.5, shows the schematic of the proposed dynamic speculation window architecture. In a pipelined digital design, logic paths are time constrained between input and output registers. Fig. 4.5 shows a design with n logic paths $Path_i$ constrained between input and output registers R_i and Q_i respectively. In the rest of this section, let us consider one path $Path_1$ out of n logic paths for easy understanding. As shown in Fig. 4.5, logic path $Path_1$ is constrained between input register R_1 and output register Q_1. Two shadow registers S_1 and T_1 are added to sample the output of the $Path_1$, in contrast to the one

shadow register approach in [5], [6], and [7]. T_1 is used to measure the available slack in the path which can be used to overclock the design. While S_1 is used to sample the valid data when the delay of the $Path_1$ is not meeting the timing constraints of the main clock M_clk due to the variability effects. Similar to double-sampling architectures, it is assumed that the shadow register S_1 always samples the valid data. In this setup, all three registers S_1, Q_1 and T_1 use the same clock frequency but different phases to sample the output. The main output register Q_1 samples the output at the rising edge of M_clk, while shadow register S_1 samples the output at the falling edge of M_clk. Shadow register T_1 samples the output at the rising edge of the shadow clock S_clk generated by the phase generator. Two XOR gates compare the data in main and shadow registers S_1 and T_1, the corresponding error values are registered in E_1 and e_1. The registers E_1 and e_1 are clocked by falling and rising edge of the M_clk respectively.

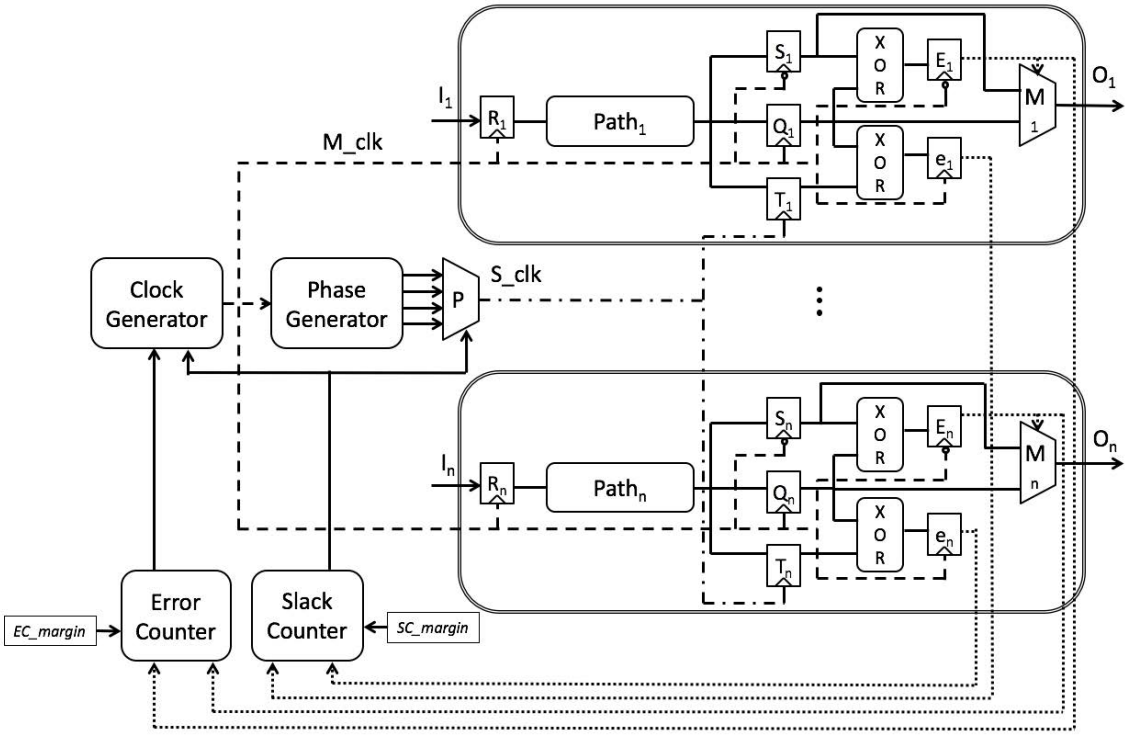
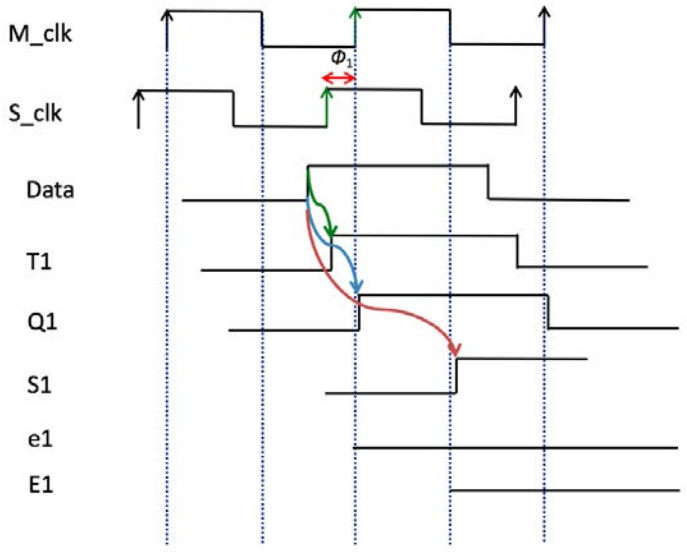
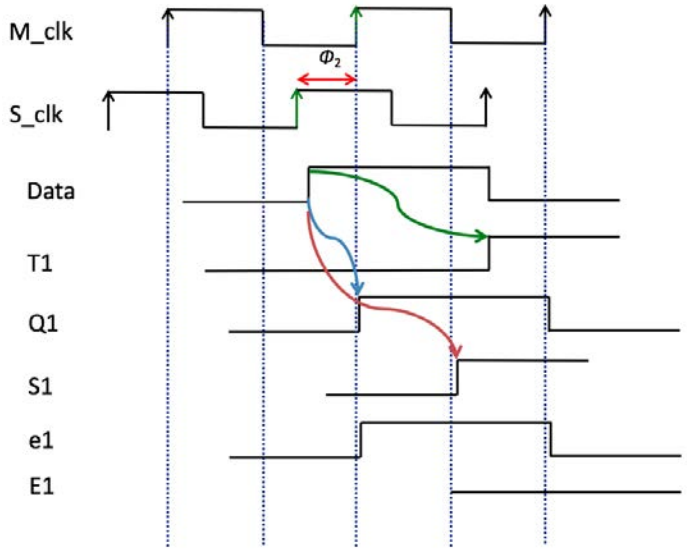


FIGURE 4.5: Principle of the dynamic speculation window based double-sampling method. Solid lines represent data and control between modules, dashed lines represent main clock, dash-dot-dash lines represent shadow clock, and dotted lines represent feedback control

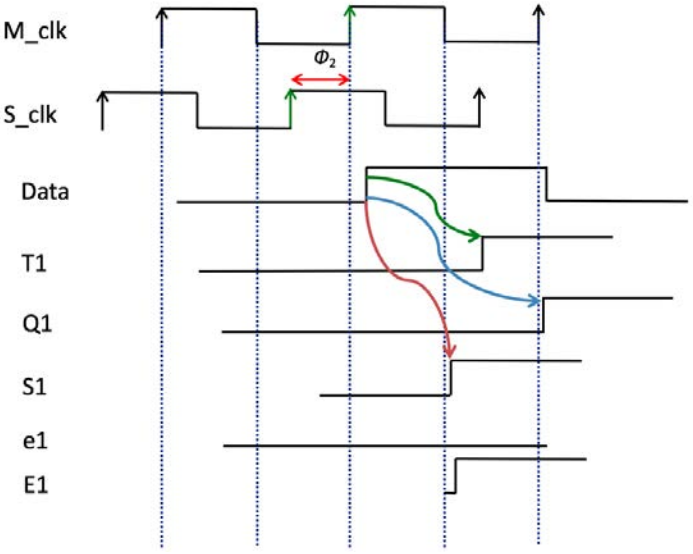
Fig. 4.6 shows the timing diagram of the proposed method. In Fig. 4.6a, the shadow clock (S_clk) leads the main clock (M_clk) by phase ϕ_1. Since the $Data$ (output of $Path_1$) is available before the rising edge of the shadow clock, the valid data is sampled



(A) Scenario 1



(B) Scenario 2



(C) Scenario 3

FIGURE 4.6: Timing diagram of the proposed method. (a): Slack measurement phase, (b): Maximum overclocking margin, and (c): Impact of temperature at maximum overclocking frequency

by T_1. Registers Q_1 and S_1 also sample the valid data in the rising and falling edges of the main clock respectively. Since the main and shadow registers sampled the same data, e_1 and E_1 are held at logic zero representing no timing error. In Fig. 4.6b, the shadow clock leads the main clock by a wider phase ϕ_2, which results in invalid data being sampled by T_1, while Q_1 and S_1 sample the valid data. In this case, the error signal e_1 is asserted at the rising edge of the main clock, while E_1 is still at logic zero, which indicates the overclocking using speculation window greater than or equal to ϕ_2 will result in errors. In Fig. 4.6c, due to temperature or other variability effects, the logic delay of the $Path_1$ violates the timing constraints of both the shadow and main clocks. This results in wrong data getting sampled by the shadow register T_1 and main register Q_1. The shadow register S_1 samples the valid data since $Data$ is valid before the falling edge of the main clock. Since both T_1 and Q_1 sample wrong data, e_1 is not asserted, while E_1 is asserted due to the difference between Q_1 and S_1 registers. Fig. 4.5 shows the schematic of error correcting mechanism in the proposed method. Here the outputs of the main register Q_1 and the shadow register S_1 are connected to the multiplexer M_1. When the error signal E_1 is equal to zero, multiplexer output M_1 passes the output of Q_1. When E_1 is held high, the output of S_1 is passed to the next stage of the pipeline.

As shown for the single logic path, additional shadow registers can be added to any number of logic paths in the design. The number of logic paths that need to have dynamic speculation is defined by the user based on the synthesis timing report of the design. In general, this is expressed as percentage of critical delay margin (CDM). All the paths with path delay greater than t_{CDM} are subjected to dynamic speculation and t_{CDM} as expressed

$$t_{CDM} = t_{critical} - (t_{critical}.CDM) \tag{4.1}$$

where $t_{critical}$ is the critical path delay. In Fig. 4.5, CDM = 100%, which means all the n logic paths of the design are monitored with dynamic speculation registers. For applications like FIR filters, where almost all the paths have equal logic delay, 100% CDM is required. For applications with well distributed path delays, CDM may range from 10% to 20%.

4.3.2 Adaptive Over-/Under-clocking Feedback Loop

Fig. 4.5 shows the feedback loop of the proposed method. The error signals E_i and e_i from all the monitored paths are connected to *Error Counter* and *Slack Counter* respectively. For different speculation windows ϕ_i, *Error Counter* and *Slack Counter* count the discrepancies in the paths being monitored.

- The *Slack Counter* denotes the number of timing errors due to overclocking or voltage overscaling the design.
- The *Error Counter* denotes the number of timing errors due to variability effects.

For a design, clocked at the maximum clock frequency, considering no variability effects, both *Error Counter* and *Slack Counter* will be zero. This indicates no timing errors in the design. To increase the performance, the design can be subjected to overclocking. *Phase Generator* in the feedback loop is used to generate different phases of the shadow clock *S_clk* which is used to determine the maximum safe overclocking margin (overclocking with out timing errors) by measuring the available slack. In the proposed setup, a dedicated register is used to measure the available slack in the design. This isolates the functioning of the pipeline from slack measuring process. When the phase difference ϕ between *M_clk* and *S_clk* increases, critical paths in the design tends to fail to indicate the limit of maximum safe overclocking margin. Slack register T_i first detects those timing errors and increment the *Slack Counter* with respect to the number of failing paths. In this case, *Error Counter* will be still zero since the functioning of the pipeline is not disturbed. Based on the *Slack Counter* value, overclocking or voltage overscaling is controlled in the feedback loop.

Certain error resilient applications can tolerate some amount of errors in the computation. In the feedback loop, *SC_margin* is the user-defined error tolerance margin due to overclocking or voltage overscaling. For applications that can tolerate timing errors, *SC_margin* can be set to allow overclocking or voltage overscaling to improve the performance or energy efficiency of the design respectively.

Under variability effects, some or most of the logic paths in the design will fail, the value of the *Error Counter* is incremented with respect to the number of failing paths. This triggers the error correction mechanism to pass valid data from shadow

register S_i to the next stage. Based on the *Error Counter* value, feedback loop decides to lower the clock frequency or increase the supply voltage to reduce the timing errors. As mentioned earlier, for error resilient applications, *EC_margin* can be set to tolerate acceptable error margin without lowering the clock frequency or increasing the supply voltage.

Therefore, *SC_margin* sets the limit for timing errors that can be tolerated in order to increase the performance or energy efficiency of the design. Likewise, *EC_margin* sets the limit for timing errors due to variability effects that can be tolerated without compromising the performance or energy efficiency. Both *SC_margin* and *EC_margin* are defined as a percentage of CDM. For applications that demand high accuracy both *SC_margin* and *EC_margin* are set to 0%. This forces the feedback loop to safely overclock (without any timing errors) the design and no tolerance towards timing errors due to variability effects. The tolerable error margin of *Error Counter* and *Slack Counter* are determined from the timing reports after synthesis and placement of the design.

4.4 Proof of Concept using Xilinx Virtex FPGA

The proposed Dynamic Speculation Window in the double-sampling method is implemented in a Xilinx Virtex VC707 evaluation board as shown in Fig. 4.7. In this experiment, we have demonstrated adaptive overclocking in Xilinx Virtex 7 FPGA based on dynamic speculation method. This method can be further extended to voltage overscaling in platforms that are compatible with voltage scaling. Vivado tool flow is used to synthesize and implement the RTL benchmarks into the FPGA. A testbench is created for all the benchmark designs with an 80-bit LFSR pattern generator for random inputs to the design under test. Onboard IIC programmable LVDS oscillator is used to provide clock frequency for the pattern generator and the design under test. As shown in Fig. 4.7, control signals from the *Error Counter* and the *Slack Counter* determine the output frequency of the LVDS oscillator. Different phases of the generated clock frequency are obtained from the inbuilt Mixed-Mode Clock Manager (MMCM) module in the FPGA [56]. The core temperature is monitored through XADC system monitor, and the temperature is varied by changing the RPM (rotations per minute) of the cooling fan.

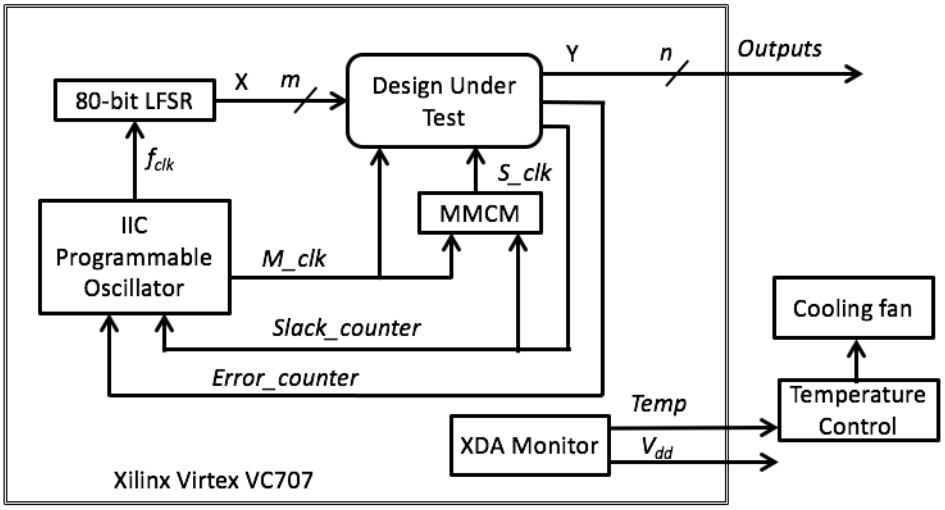


FIGURE 4.7: Experimental setup of Adaptive overclocking based on Dynamic Speculation

For error-free overclocking, *Slack Counter* margin is set to zero. At the ambient core temperature of around 28°C, when *Error Counter* and *Slack Counter* are equal to zero, the speculation window ϕ is increased for every clock cycle from 0° to 180° at discrete intervals of 25° until the *Slack Counter* records an error. Beyond 180° up to 360°, the speculation window width repeats due to periodic nature of the clock. Once the slack counter records an error, the system is put on halt and safely overclocked by trading the available slack. While the speculation window ϕ is swept, the pipeline functions normally since the output register Q_i and the shadow register S_i sample the same valid data. Regarding error correction due to variability effects, if *Error Counter* records more than 2% of the monitored paths not meeting the timing constraints, the system is put on halt and the operating frequency of the system is decreased in multiples of a clock frequency step δ to reduce the errors (in our test platform the frequency step is $\delta = 5MHz$). Since correcting more errors will reduce the throughput of the design, decreasing the operating frequency is a necessary measure. Multiplexer M_i, as shown in Fig. 4.5, corrects the error by connecting the output of the shadow register S_i to the next stage of the pipeline. Once the temperature falls back, the operating frequency of the design is again increased based on the slack measurement as described earlier.

To highlight the adaptive overclocking and timing error detection and correction capability of the proposed method, we have implemented a set of benchmark designs in both Razor-based method [5] and the proposed dynamic speculation window based

method. Benchmarks used in this experiments are common datapath elements like FIR filter (8 taps, 8 bits), and various versions of 32-bit adder and 32-bit multiplier architectures. After implementation, timing reports are generated to spot the critical paths in the design. Timing and location constraints are used to place the shadow registers S_i and T_i close to the main output register Q_i for all the output bits of the design. For the Razor-based method, all the monitored paths are annexed with one shadow register S_i in contrast to the proposed method with two shadow registers, and the logic paths that are shorter than speculation window are automatically annexed with buffers by the synthesis tool. For comparison purpose, both the Razor-based method and the proposed method are subjected to identical test setup. In both cases, the design is overclocked beyond the maximum frequency estimated by Vivado and the FPGA's core temperature is varied to introduce variability effects in the design.

After the synthesized bitstream is programmed into the FPGA and the design is initially clocked at the maximum frequency (F_{max}) estimated by Vivado for each design. At the ambient core temperature of around 28°C, the speculation window ϕ is increased for every clock cycle and if the *Slack Counter* is zero at the widest speculation window, then the clock frequency can be increased up to 40% because at 180°, ϕ corresponds to half of the clock period. That implies all the critical paths in the design have positive slack equals to half of the clock period. Overclocking is done by halting the design and increasing the clock frequency from the LVDS oscillator. The LVDS oscillator is programmed through the IIC bus with the pre-loaded command word for the required frequency. After changing the frequency, again the phase sweep starts from 0° until the *Slack Counter* records an error. This way, the error-free overclocking margin of each design is determined with respect to the maximum operating frequency estimated by Vivado.

While operating at the safe overclocking F_{max} at 28°C, the core temperature of the FPGA is varied by changing the RPM of the cooling fan. Due to the increase in temperature, the critical paths of the design starts to fail. When the *Error Counter* records an error in more than 2% of the monitored paths, the design is put on halt and the frequency is reduced in multiples of 5MHz until the *Error Counter* margin is below 2%. Since the shadow register S_i latches the correct output, errors are corrected without re-executing for that particular input pattern. The tolerance margin is used to

TABLE 4.1: Synthesis results of different benchmark designs

Benchmarks	Area without error detection		Area overhead for Proposed method		Area overhead for Razor method	
	LUTs	FFs	LUTs	FFs	LUTs	FFs
8-tap 8-bit FIR Filter	1279	1547	2.5%	2%	2%	1%
Unsigned 32-bit Multiplier	7270	4511	1.9%	1.7%	2.3%	0.9%
Signed 32-bit Wallace Tree Multiplier	5997	4458	2.8%	1.8%	3.2%	1%
32-bit Kogge-Stone Adder	3944	4117	3.7%	1.7%	4%	1.2%
32-bit Brent-Kung Adder	3753	4053	3.3%	1.9%	3.8%	1%

TABLE 4.2: Estimated F_{max} and safe overclocking F_{max} of benchmark designs at 28°C

Benchmarks	Estimated F_{max} (MHz)	Safe overclocking F_{max} at 28°C (in MHz)	Safe overclocking margin at 28°C
8-tap 8-bit FIR Filter	167	260	55%
Unsigned 32-bit Multiplier	76	130	71%
Signed 32-bit Wallace Tree Multiplier	80	135	69%
32-bit Kogge-Stone Adder	130	215	64%
32-bit Brent-Kung Adder	135	216	60%

reduce the overhead in error correction mechanism which can affect the throughput of the design.

Table 4.1 provides synthesis results of the benchmarks without any error detection technique, with razor flip-flops [5], and with proposed error detection and correction technique. The area overhead of the proposed method compared to the original design is kept between 3.6% to 5.4%, which demonstrates the applicability of the technique in real designs. Table 4.1 also provides the maximal frequency F_{max} estimated by Vivado. The Look-Up Table (LUT) overhead of the proposed method is on average 0.4% less compared to the Razor implementation, since no buffers are needed for the shorter paths. However, Flip-Flop (FF) overhead of the proposed method is 0.8% more compared to the Razor method due to the two shadow flip-flops used for slack measurement and error correction.

Table 4.2, shows the estimated F_{max} of all the benchmark designs and the safe overclocking F_{max} that can be reached by the proposed error detection method at an ambient temperature of 28°C. As an example, for the FIR filter design, Vivado estimated F_{max} is 167MHz. However, this design can be overclocked at the ambient temperature by more than 55% up to 260MHz, without any error, and up to 71% for the other

TABLE 4.3: Impact of temperature while overclocking in benchmark designs at 50°C

Benchmarks	% of monitored paths fail at 50°C for F_{max} at 28°C	Safe overclocking F_{max} at 50°C (in MHz)	Safe overclocking margin at 50°C
8-tap 8-bit FIR Filter	12	180	8%
Unsigned 32-bit Multiplier	22	85	12%
Signed 32-bit Wallace Tree Multiplier	26	85	7%
32-bit Kogge-Stone Adder	13	180	38%
32-bit Brent-Kung Adder	21	180	33%

benchmarks. Fig. 4.8, shows the plot of safe overclocking frequency of all the benchmark designs that can be reached by the proposed error detection method for different FPGA's core temperature. At the safe overclocking frequency, the temperature is increased from 28°C to 50°C at which *Error Counter* records 12% of the paths failed. Since the percentage of failing paths is more than the pre-determined margin of 2%, the clock frequency of the LVDS oscillator is brought down to 180MHz. Scaling down the frequency from 260MHz to 180MHz contains the percentage of failing paths below 2% margin. Once the temperature falls back to 28°C, the available slack is measured at runtime and the clock frequency is increased to achieve maximum performance.

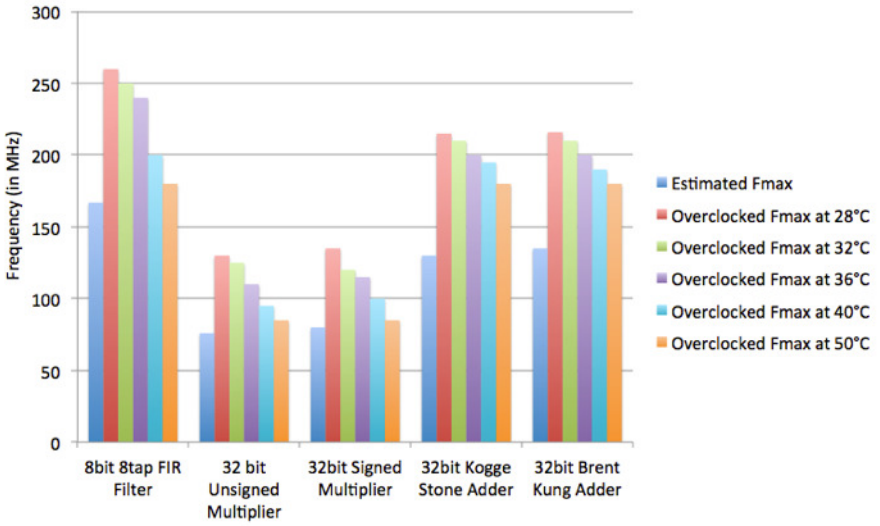


FIGURE 4.8: Overclocking versus temperature for different benchmarks

Table 4.3 lists the impact of temperature and the safe overclocking margin at the high temperature of 50°C for all the benchmarks. The percentage of monitored paths that fails at 50°C shows the impact of the temperature while overclocking the design.

The operating frequency has to be scaled down to limit these errors which can be eventually corrected by the error correction technique in the proposed method. Even at the higher temperature of 50°C, the FIR filter is safely overclocked up to 8% compared to the maximum frequency estimated by Vivado. Similarly, at 50°C, both unsigned and signed multiplier architectures can be safely overclocked up to 12% and 7% respectively. A maximum safe overclocking margin of 38% is achieved for the Kogge-Stone adder and 33% by the Brent-Kung adder while operating at 50°C. These results demonstrate the adaptive overclocking, and error detection and correction capability of the proposed method with a limited area overhead in the FPGA resources.

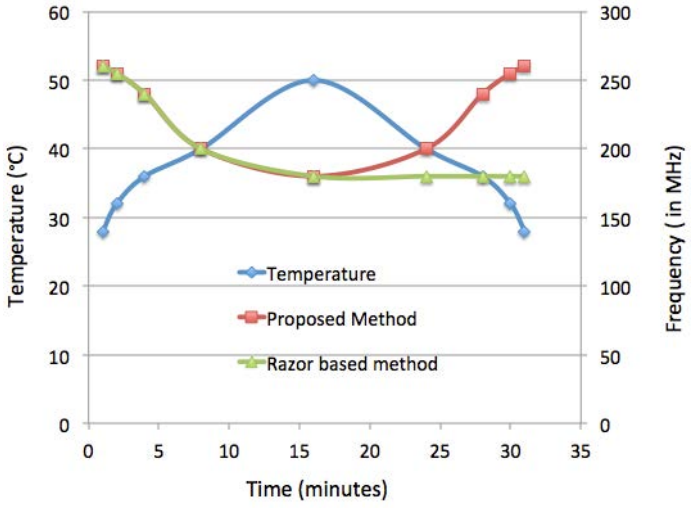


FIGURE 4.9: Comparison of Razor-based feedback look and the proposed dynamic speculation window. Curve in blue diamond represents the FPGA's core temperature. Curve in red square represents the frequency scaling by the proposed method. Green triangle curve represents the frequency scaling by the Razor-based method

Fig. 4.9, shows the comparison of the proposed dynamic speculation window and the Razor-based overclocking for the 8-bit 8-tap FIR filter design in Virtex 7 FPGA. The core temperature of the FPGA is varied by controlling the speed of the cooling fan. As shown in Fig. 4.9, the FPGA's core temperature is increased from the ambient room temperature of 28°C to 50°C and then decreased again back to the room temperature. When the temperature increases, critical paths in the design starts to fail, which makes the proposed method and the Razor-based method to scale down the frequency to limit the error below 2% of monitored paths. Initially, at the room temperature, the design is running at 260 MHz. At the temperature of 50°C, the frequency is scaled down to 180 MHz. When the temperature falls back to the room temperature, the proposed method is able to measure the available slack in the pipeline and to increase the frequency to overclock the design and therefore to increase performance. Under similar test setup,

the Razor-based feedback loop scales down the frequency as the temperature increases. However, when the temperature falls back, there is no change in the frequency. This is because the Razor implementation does not have the slack measurement in place to adaptively change the frequency when the temperature is reducing. Due to the additional shadow register placed in the proposed method, the system is able to measure the slack available in the pipeline which is traded to overclock the design according to temperature fluctuations. This gives a more upper hand for the proposed dynamic speculation window based error detection method over the classical Razor-based timing error detection.

4.5 Comparison with Related Works

4.5.1 DVFS with slack measurement

In [7], an online slack measurement technique based on [53], [54], and [55] is used to scale the supply voltage or frequency to increase the energy savings. The output of the path under monitoring (PUM) is sampled by both main register and shadow register using same clock frequency but different phases as shown in Fig. 4.4.

Here the basic assumption is that the main register meets the timing constraints in all situations, in other words, there is no scope for timing errors or error correction in this method. The shadow register is clocked by different phases of the main clock. An error register records errors by comparing output data from both the main and shadow registers. The number of logic paths to be monitored by inserting a shadow register is pre-determined (represented as a certain percentage of the Critical Delay Margin) in contrast to [35]. This reduces the area cost by not appending shadow registers for all the logic paths in the circuit. Also, the paths that are shorter than the speculation window are not monitored, which voids the need for buffer insertion in shorter paths. When the shadow clock leads the main clock by phase ϕ, if the PUM finished the execution before the rising edge of the shadow clock, a valid output data is sampled by both the shadow and main registers. The phase ϕ is further increased until the shadow register records invalid data while main register samples the valid data. The measured slack is compared with the pre-determined guardband to increase the energy savings by scaling the voltage or frequency. This method is providing energy saving without any scope for error detection or correction. Compared to the proposed dynamic speculation method,

since the shadow clock is leading the main clock, it cannot detect the error when the critical path violates the main clock timing constraints due to temperature and other variability effects.

In [57],[58], an in-situ monitor (ISM) is proposed to identify timing errors due to variability effects. In this method, three speculation windows are evaluated using three delay chains and a multiplexer to detect the timing errors. Unlike the proposed dynamic speculation method, this method can only detect timing errors but not provide any scope for overclocking to achieve energy gains. In [58], the effect of temperature is studied to demonstrate the robustness of the method, but the impact of dynamic temperature variations and the adaptability of the ISM to overscale or underscale the frequency and voltage is not well established.

4.5.2 Timing error handling in FPGA and CGRA

In [5], the advantages of using Razor-based error detection and correction in FPGA implementations to increase energy efficiency are discussed. In this method, a timing fault detector (TFD) is added to the pre-determined number of logic paths in the FPGA implementation. The TFD block consists of a shadow register, an XOR gate, and a fault register. Adding shadow registers is carried out by altering the netlist obtained during the post-placement stage. Based on the timing report and the post-placement netlist, unused resources in the FPGA are used for TFDs. Since the TFDs are placed after the placement and routing of the main design, the delay between the output of the logic path and the shadow register might not be same as the delay between the output of the logic path and the main register. This difference in the delay between the main register and the shadow register is compensated by adjusting the shadow clock lag. After placing the TFDs, the FPGA is overclocked beyond the maximum frequency estimate given by the FPGA timing tool under fixed temperature and supply voltage. The scope of this method is only detecting the fault rate for different frequencies beyond the maximum frequency. No error correction technique is proposed nor the variability issues are discussed.

In [6], overclocking CGRAs using Razor is presented. This method focuses on implementing Razor in 2D arrays and propagating stall signals to correct the errors due to overclocking. However, this method is also not discussing adaptively changing the frequency based on temperature and variability effects. Similarly, [43] presents timing

error handling in CGRAs using triple modular redundancy (TMR). Hardware redundancy methods consume more power and area compared to time redundancy methods and not suitable for designs with tighter area constraint.

4.6 Conclusion

In this chapter, limitations of fixed speculation window used in the existing double sampling methods to detect and correct errors are discussed. We proposed dynamic speculation window architecture for adaptive overclocking or underclocking to an achieve optimal trade-off between energy efficiency and performance. The proposed dynamic speculation method is implemented in Xilinx FPGA platform as proof of concept. A comparative study of adaptive overclocking based on dynamic speculation and Razor method is done to highlight the advantage of the proposed method. The proposed method uses double-sampling and slack measurement to adaptively overclock and underclock the design. The maximum of 71% safe overclocking margin at ambient temperature has been achieved at the cost of shadow registers, XOR gates, and counters, which results in a maximum area overhead of 1.9% LUTs and 1.7% FFs over the original design, as shown in Table 4.1. Instead of merely overclocking the design this method detects timing errors and corrects it in real time. This method can be easily expanded to other form of datapaths and also reconfigurable architecture like CGRAs. Also, this method can be extended to voltage overscaling with compatible platforms. In this chapter, we have shown that the dynamic speculation improves the scope of error handling method based on detect and correct. Also, the proposed architecture provides the flexibility for the user to reduce the accuracy to achieve enhanced energy gains. This provides the scope for accepting and living with errors (approximate computing). In the following chapters, approximate computing is discussed in the context of near-threshold computing. The dynamic speculation architecture is used in approximate computing to obtain an optimistic trade-off between accuracy and voltage.

Chapter 5

VOS (Voltage OverScaling) for Error Resilient Applications

5.1 Introduction

In the earlier chapters, voltage scaling techniques and their potential to unlock the opportunities of higher energy efficiency by operating the transistors near or below the threshold are discussed. After the advent of inherent low-leakage technologies like Fully Depleted Silicon On Insulator (FDSOI), Near-Threshold Computing (NTC) has gained more importance in VLSI due to improved resistance towards various variability effects like Random Dopant Fluctuations (RDF). Body-biasing technique in FDSOI provides greater flexibility to control the trade-off between performance and energy efficiency based on the application need. In spite of the improvement in techniques and technology, near-threshold computing is still seen as a no-go zone for conventional sub-nanometer designs, due to timing errors introduced by the supply voltage scaling and need for additional hardware to detect and correct such timing errors.

Error-resilient computing is an emerging trend in IoT, in which accuracy of the computing can be traded to improve the energy efficiency and to lower the silicon footprint of the design [49]. Emerging class of applications based on statistical and probabilistic algorithms used for video processing, image recognition, text and data mining, machine learning, have the inherent ability to tolerate hardware uncertainties. Such error-resilient applications that can live with errors, void the need for additional hardware to detect

and correct errors. Also, error-resilient applications provide an opportunity to design approximate hardware to meet the computing needs with higher energy efficiency and tolerable accuracy loss. In error-resilient applications, approximations in computing can be introduced at different stages of computing and at varying granularity of the design. Using probabilistic techniques, computations can be classified as significant and non-significant at different design abstraction levels like algorithmic, architecture, and circuit levels.

In the rest of the chapter, existing approximation methods for arithmetic operators are discussed. The need for characterizing and modelling of arithmetic operators is presented. The framework used to characterize the behaviour of arithmetic operators at near-threshold regime is discussed. Based on the characterization, statistical modelling of arithmetic operators for error resilient applications is presented. As a proof of concept, different configurations of adders are modelled and the results are elaborated.

5.2 Approximation in Arithmetic Operators

Approximations in arithmetic operators are broadly classified based on the level at which approximations are introduced [49]. This section reviews approximation methods proposed in the literature at physical and architectural levels.

5.2.1 Approximation at Circuit Level

In [50], operators of a Functional Unit (FU) are characterized by analyzing the relationship between V_{dd} scaling and Bit Error Rate (BER), ratio of faulty output bits over total output bits. Based on the characterization, for every FU in the pipeline, one more *imprecise* FU running at lower V_{dd} is designed. According to the application's need, a selected set or all the FUs in the pipeline are accompanied by an *imprecise* counterpart in the design. Based on the user-defined precision level, computations are performed either by *precise* or *imprecise* FU in the pipeline. Instead of duplicating every FU with an imprecise counterpart, a portion of the FU is replaced by imprecise or approximate design as discussed in [49]. Fig. 5.1 shows this principle where least significant inputs are processed by an *approximate* operator and most significant inputs are processed by an *accurate* operator to increase the energy efficiency at the cost of acceptable accuracy

loss. In [59] n-bit Ripple Carry Adder (RCA) based on near-threshold computing is proposed with two parts; k-bit LSBs approximated while $(n-k)$ bits computed by precise RCA.

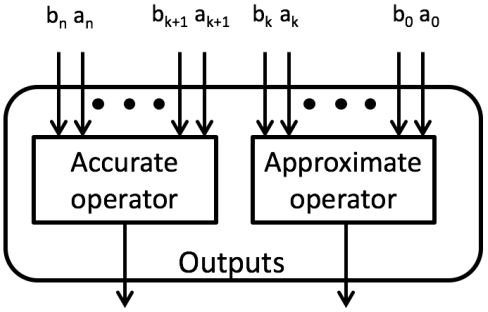


FIGURE 5.1: Accurate approximate configuration of [49]

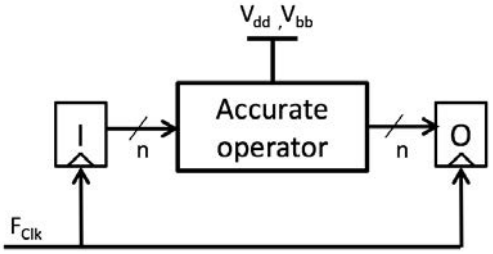


FIGURE 5.2: Approximate operator based on VOS

Another class of circuit-level approximation is achieved by applying dynamic voltage and frequency scaling to an accurate operator, as depicted in Fig. 5.2. Due to the dynamic control of voltage and frequency, timing errors due to scaling can be controlled flexibly in terms of trade-off between accuracy and energy. This method is referred as Voltage Over-Scaling (VOS) in [50]. Similar to VOS, clock overgating based approximation is introduced in [60]. Clock overgating is done by gating clock signal to selected flip-flops in the circuit during execution cycles in which the circuit functionality is sensitive to their state. In all the approximation methods at the circuit level, in addition to the deliberate approximation introduced, the impact of variability has to be considered to achieve an optimum balance between accuracy and energy. Decoupling the data and control processing is proposed in [61] to mitigate the impact of variation in near-threshold approximate designs. Also, technologies like FDSOI provide good resistance towards the impact of variability.

5.2.2 Approximation at Architectural level

Approximation at architectural level is discussed in [62], where accuracy control is handled by bitwidth optimization and scheduling algorithms. Also, other forms of architectural-level approximations are discussed in [63], where probabilistic pruning based approximation method is proposed. In this method, a design is optimized by

removing certain hardware components of the design and/or by implementing an alternate way to perform the same functionality with reduced accuracy. In [47], a probabilistic approach is discussed in the context of device modeling and circuit design. In this method, noise is added to the input and output nodes of an inverter and the probability of error is calculated by comparing the output of the inverter with a noise-free counterpart. In [64], new classes of pruned speculative adders are proposed by adding gate-level pruning in speculative adders to improve Energy Delay Area Product (EDAP). Though there is a claim that the pruned speculative adder will show higher gains when operated at sub-threshold region, no solid justification is given in [64]. In general, approximations introduced by pruning methods are more rigid in nature, which lacks the dynamicity to switch between various energy-accuracy trade-off points.

On contrast, circuit level approximations based on voltage scaling are more flexible, easy to implement and offer dynamic control over energy-accuracy trade-off. Approximation introduced by supply voltage scaling offers *dynamic approximation*, by changing the operating triad (combination of supply voltage, body-biasing scheme, and clock frequency) of the design at runtime, which makes the user to control the energy-accuracy trade-off efficiently. In [65], limitations of voltage over-scaling based approximate adders such as need for level shifters and multiple voltage routing lines are mentioned. These limitations can be overcome by employing uniform voltage scaling along the pipeline or at larger granularity.

In this chapter, for the above-mentioned advantages, circuit-level approximations are explored which can further be extended to algorithmic level by using a statistical model for approximate operators. This work makes the following contributions:

- Characterizing energy efficiency and accuracy of different adder configurations using several operating triads (supply voltage, body-biasing voltage, clock period).
- Formulating a framework to model the statistical behavior of arithmetic operators subjected to voltage over-scaling that can be used at algorithmic level.

5.3 Characterization of arithmetic operators

In this section, characterization of arithmetic operators is discussed for voltage over-scaling based approximation, as shown in Fig. 5.2. Characterization of arithmetic operators helps to understand the behaviour of the operators with respect varying operating triads. Adders and Multipliers are the most common arithmetic operators used in datapaths. In this work, different adder configurations are explored in the context of near-threshold regime.

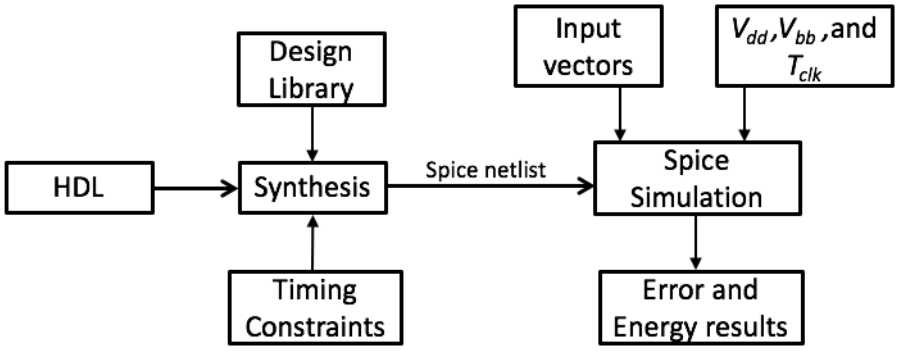


FIGURE 5.3: Proposed design flow for arithmetic operator characterization

Fig. 5.3 shows the characterization flow of the arithmetic operators. Structured gate-level HDL is synthesized with user-defined constraints. The output netlist is then simulated at transistor level using SPICE (Simulation Program with Integrated Circuit Emphasis) platform by varying operating triads (V_{dd}, V_{bb}, T_{clk}), where V_{dd} is supply voltage, V_{bb} is body-biasing voltage, and T_{clk} is clock period. In an ideal condition, the arithmetic operator functions without any errors. Also, EDA tools introduce additional timing margin in the datapaths during Static Timing Analysis (STA) due to clock path pessimism. This additional timing prevents timing errors due to variability effects. Due to the limitation in the availability of design libraries for near/sub-threshold computing, it is necessary to use SPICE simulation to understand the behaviour of arithmetic operators in different voltage regimes. By tweaking the operating triads, timing errors e are invoked in the operator and can be represented as

$$e = f(V_{dd}, V_{bb}, T_{clk}) \tag{5.1}$$

Characterization of arithmetic operator helps to understand the point of generation and propagation of timing errors in arithmetic operators. Among the three parameters in the triad, scaling V_{dd} causes timing errors due to the dependence of operator's propagation delay t_p on V_{dd}, such as

$$t_p = \frac{V_{dd}.C_{load}}{k(V_{dd} - V_t)^2} \tag{5.2}$$

Body-biasing potential V_{bb} is used to vary the threshold voltage (V_t), thereby increasing the performance (decreasing t_p) or reducing leakage of the circuit. Due to the dependence of t_p on V_t, V_{bb} is used solely or in tandem with V_{dd} to control the timing errors. Scaling down V_{dd} improves the energy efficiency of the operator due to its quadratic dependence to total energy. $E_{total} = V_{dd}^2.C_{load}$. A mere increase in T_{clk} does not reduce the energy consumption, though it will reduce the total power consumption of the circuit

$$P_{total} = \alpha.V_{dd}^2.\frac{1}{T_{clk}}.C_{load} \tag{5.3}$$

Therefore, T_{clk} is scaled along with V_{dd} and V_{bb} to achieve high energy efficiency.

5.3.1 Characterization of Adders

The adder is an integral part of any digital system. In this section, two adder configurations Ripple carry adder (RCA) and Brent-Kung adder (BKA) are characterized based on circuit level approximations. Ripple carry adder is a sequence of full adders with serial prefix based addition. RCA takes n stages to compute n-bit addition. In the worst case, carry propagates through all the full adders and makes it longest carry chain adder configuration. Longest carry chain corresponds to the critical path of the adder, based on which the frequency of operation is determined. In contrast, Brent-Kung adder is a parallel prefix adder. Fig. 5.4, shows the carry chain of Brent-Kung adder. BKA takes $2\log_2(n-1)$ stages to compute n-bit addition. In BKA, carry generation and propagation are segmented into smaller paths and executed in parallel. Black and gray cells in Fig. 5.4 represent carry generate and propagate, respectively. Behaviour of the arithmetic operator in near/sub-threshold region is different from the super-threshold region. In case of an RCA, when the supply voltage is scaled down, the expected behaviour is failure of critical path(s) from longest to the shortest with respect to the reduction in the supply voltage. Fig. 5.5 shows the effect of voltage over-scaling in 8-bit RCA. When the supply voltage is reduced from 1V to 0.8V, MSBs starts to fail. As the voltage is

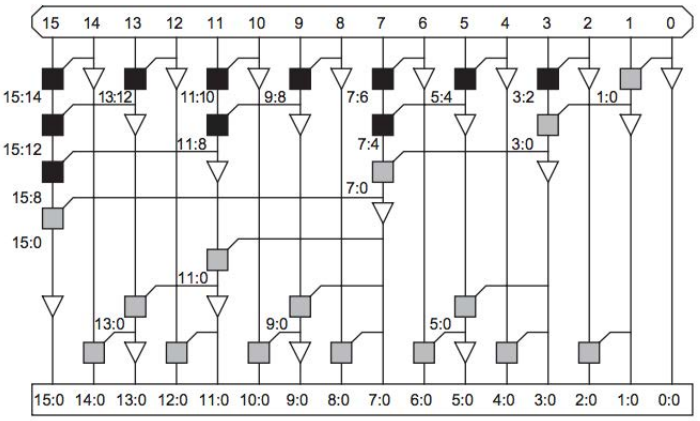


FIGURE 5.4: Carry Chain of Brent-Kung Adder [66]

further reduced to 0.7V and 0.6V more BER is recorded in middle order bits rather than most significant bits. For 0.5V V_{dd}, all the middle order bits reaches BER of 50% and above. Similar behaviour is observed in 8-bit BKA shown in Fig. 5.6 for v_{dd} values of 0.6V and 0.5V. This behaviour imposes limitations in modelling approximate arithmetic operators in near/sub-threshold using standard models. Behaviour of arithmetic operators during voltage over-scaling in near/sub-threshold region can be characterized by SPICE simulations. But SPICE simulators take long time (4 days with 8 cores CPU) to simulate an exhaustive set of input patterns needed to characterize arithmetic operators.

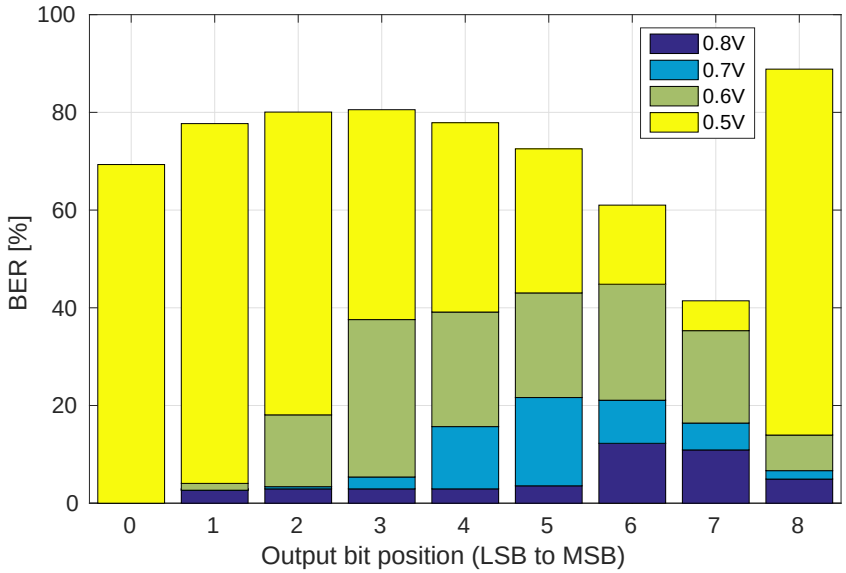


FIGURE 5.5: Distribution of BER in output bits of 8-bit RCA under voltage scaling

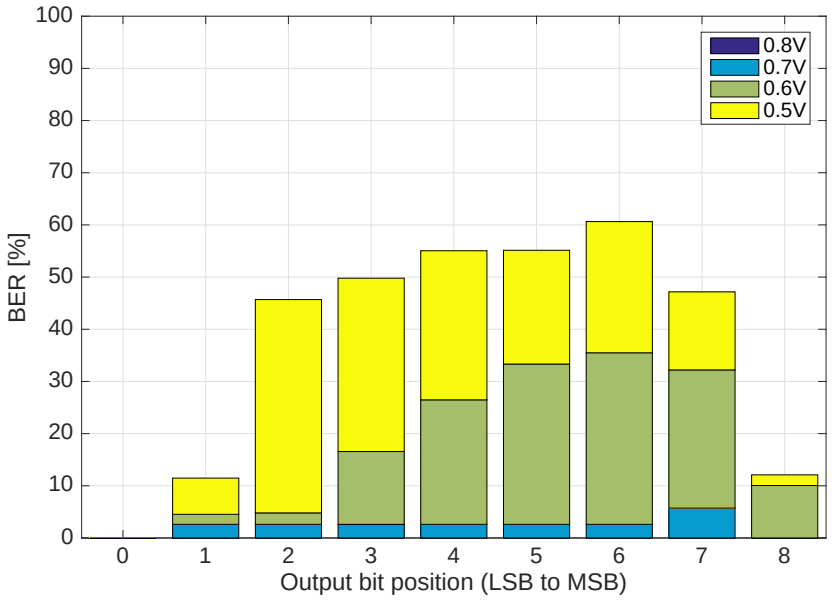


FIGURE 5.6: Distribution of BER in output bits of 8-bit BKA under voltage scaling

5.4 Modelling of VOS Arithmetic Operators

As stated previously, there is a need to develop models that can simulate the behavior of faulty arithmetic operators at the functional level. In this section, we propose a new modelling technique that is scalable for large-size operators and compliant with different arithmetic configurations. The proposed model is accurate and allows for fast simulations at the algorithm level by imitating the faulty operator with statistical parameters.

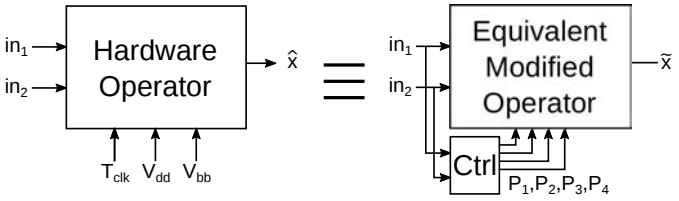


FIGURE 5.7: Functional equivalence for hardware adder

As VOS provokes failures on the longest combinatory datapaths in priority, there is clearly a link between the impact of the carry propagation path on a given addition and the error issued from this addition. Fig. 5.7 illustrates the needed relationship between hardware operator controlled by operating triads and statistical model controlled by statistical parameters P_i. As the knowledge of the inputs gives necessary information about the longest carry propagation chain, the values of the inputs are used to generate

the statistical parameters that control the equivalent model. These statistical parameters are obtained through an offline optimization process that minimizes the difference between the outputs of the operator and its equivalent statistical model, according to a certain metric. In this work, we used three accuracy metrics to calibrate the efficiency of the proposed statistical model:

- Mean Square Error (MSE) – average of squares of deviations between the output of the statistical model $\tilde{x}$ and the reference $\hat{x}$,

$$MSE(i) = \frac{1}{n}\sum_{i=1}^{n}(\hat{x}_i - \tilde{x}_i)^2 \tag{5.4}$$

- Hamming distance – number of positions with bit flip between the output of the statistical model $\tilde{x}$ and the reference $\hat{x}$,

$$d_{ham}(i) = \sum_{j=0}^{N-1}(\hat{x}_{i,j} \oplus \tilde{x}_{i,j}) \tag{5.5}$$

- Weighted Hamming distance – Hamming distance with weight for every bit position depending on their significance,

$$d_{w\_ham}(i) = \sum_{j=0}^{N-1}(\hat{x}_{i,j} \oplus \tilde{x}_{i,j}).2^j \tag{5.6}$$

5.4.1 Proof of Concept: Modelling of Adders

In the rest of the section, a proof of concept is illustrated by applying VOS on different adder configurations. All the adder configurations are subjected to VOS and characterized using the flow described in Fig. 5.3. Fig. 5.8 shows the design flow of modelling VOS operators. As shown in Fig. 5.7 rudimentary model of the hardware operators is created with the input vectors and the statistical parameters. For the given input vectors, output of both the model and the hardware operator is compared based on the defined set of accuracy metrics. The comparator shown in Fig. 5.8 generates signal to noise ratio (SNR) and Hamming distance to determine the quality of the model based on the accuracy metrics. SNR and Hamming distance are fed back to the optimization algorithm to further fine tune the model to represent the VOS operator. In the case of

adders, only one parameter P_i for the statistical model is used and is defined as C_{max}, the length of the maximum carry chain to be propagated.

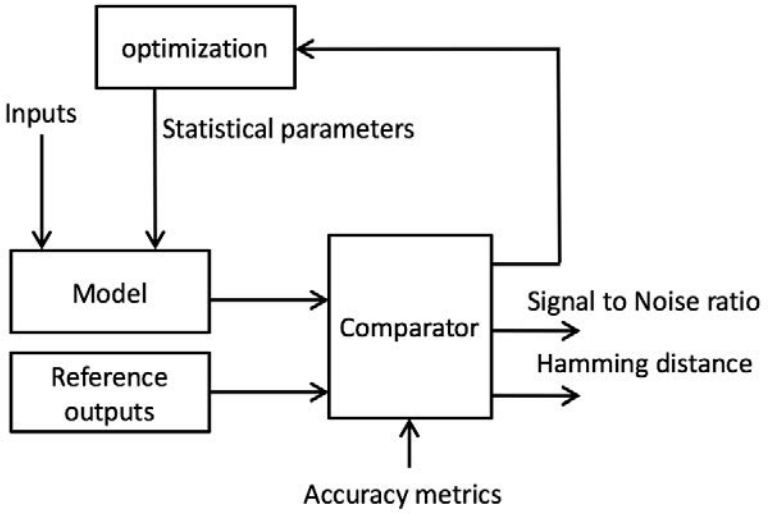


FIGURE 5.8: Design flow of modelling of VOS operators

Hence, given the operating parameters $(T_{clk}, V_{dd}, V_{bb})$ and a couple of inputs (in_1, in_2), the goal is to find C_{max}, minimizing the distance between the output of the hardware operator and the equivalent modified adder. This distance can be defined by the above listed accuracy metrics. Hence, C_{max} is given by:

$$C_{max}\left(in_1, in_2\right) = \underset{C\in[0,N]}{Argmin} \left\|\hat{x}\left(in_1, in_2\right), \tilde{x}\left(in_1, in_2\right)\right\|$$

where $\|x, y\|$ is the chosen distance metric applied to x and y. As the search space for characterizing C_{max} for all sets of inputs is potentially very high, C_{max} is characterized only in terms of probability of appearing as a function of the theoretical maximal carry chain of the inputs, denoted as $P\left(C_{max} = k | C^{th}_{max} = l\right)$. This way, the mapping space of 2^{2N} possibilities is reduced to $(N+1)^2/2$. Table 5.1 gives the template of the probability values needed by the equivalent modified adder to produce an output.

TABLE 5.1: Carry propagation probability table of modified 4-bit adder

$C_{max}C^{th}_{max}$	0	1	2	3	4
0	1	$P(0\|1)$	$P(0\|2)$	$P(0\|3)$	$P(0\|4)$
1	0	$P(1\|1)$	$P(1\|2)$	$P(1\|3)$	$P(1\|4)$
2	0	0	$P(2\|2)$	$P(2\|3)$	$P(2\|4)$
3	0	0	0	$P(3\|3)$	$P(3\|4)$
4	0	0	0	0	$P(4\|4)$

The optimization algorithm used to construct the modified adder is shown in Algorithm 2. When the inputs (in_1, in_2) are in the vector of training inputs, output of the hardware adder configuration $\hat{x}$ is computed. Based on the particular input pair (in_1, in_2), maximum carry chain C_{max}^{th} corresponding to the input pair is determined. Output $\tilde{x}$ of the modified adder with three input parameters (in_1, in_2, C) is computed. The distance between the hardware adder output $\hat{x}$ and modified adder output $\tilde{x}$ is calculated based on the above defined accuracy metrics for different iterations of C. The flow continues for the entire set of training inputs.

begin
  $P(0 : N_{bit\_adder} | \ 0 : N_{bit\_adder}) := 0;$
  $max\_dist := +\infty;$
  $C_{max\_temp} = 0;$
  for *variable* $in_1, in_2 \in training\_inputs$ **do**
    $\hat{x} := add\_hardware(in_1, in_2)$ $C_{max}^{th} := max\_carry\_chain(in_1, in_2);$
    for *variable* $C \in C_{max}^{th}$ *down to* 0 **do**
      $\tilde{x} := add\_modified(in_1, in_2, C);$
      $dist := \|\hat{x}, \tilde{x}\|;$
      if $dist <= max\_dist$ **then**
        $dist\_max := dist;$
        $C_{max\_temp} := C;$
      end
    end
    $P(C_{max\_temp} | C_{max}^{th}) + +$
  end
  $P(: | :) := P(: | :)/size(training\_outputs);$
end

Algorithm 2: Optimization Algorithm

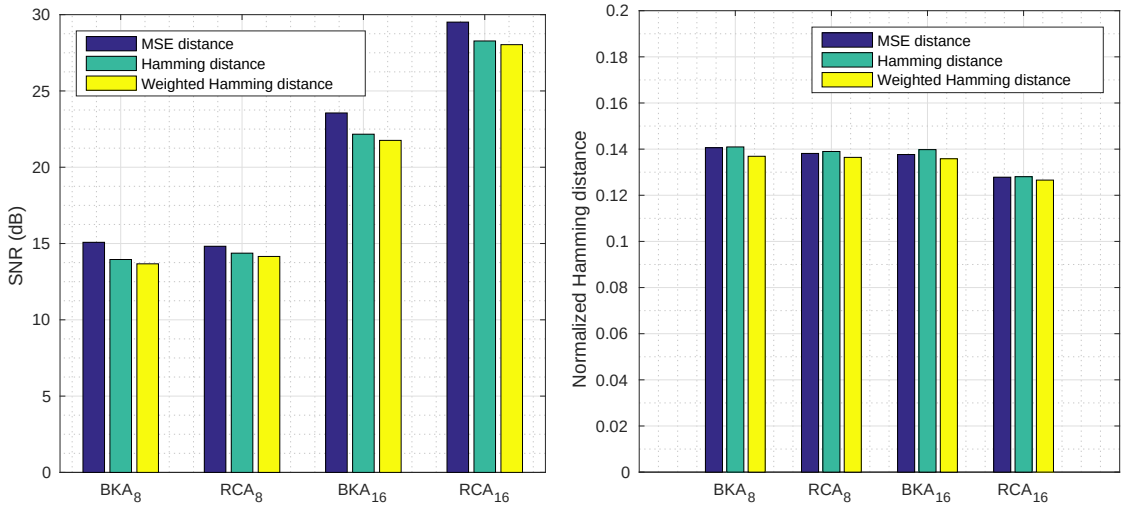
After the off-line optimization process is performed, the equivalent modified adder can be used to generate the outputs corresponding to any couple of inputs in_1 and in_2. To imitate the exact operator subjected to VOS triads, the equivalent adder is used in the following way:

1. Extract the theoretical maximal carry chain C_{max}^{th} which would be produced by the exact addition of in_1 and in_2.

2. Pick of a random number, choose the corresponding row of the probability table, in the column representing C_{max}^{th}, and assign this value to C_{max}.

3. Compute the sum of in_1 and in_2 with a maximal carry chain limited to C_{max}.

Fig. 5.9 shows the estimation error of model of different adders based on the above defined accuracy metrics. SPICE simulations are carried out in 43 operating triads with 20K input patterns. Input patterns are chosen in such a way that all the input bits carry equal probability to propagate carry in the chain. Fig. 5.9a plots the Signal to Noise Ratio (SNR) of 8- and 16-bit RCA and BKA adders. MSE distance metric shows higher mean SNR, followed by Hamming distance and weighted Hamming distance metrics. Since MSE, and weighted Hamming distance are taking the significance of bits into account, their resulting mean SNRs are higher than for the Hamming distance metric. Fig. 5.9b shows the plot of normalized Hamming distance of all the four adders. In this plot, MSE and Hamming distance metrics are almost equal, with a slight advantage for non-weighted Hamming distance, which is expected since this metric gives all bit positions the same impact. Both the 8-bit adders have the same behaviour in terms of the distance between the output of hardware adder and modified adder. On the other hand, 16-bit RCA is better in terms of SNR compared to its BKA counterpart. These results demonstrate the accuracy of the proposed approach to model the behavior of operators subjected to VOS in terms of approximation.



(A) Signal to Noise Ratio (B) Normalized Hamming Distance

FIGURE 5.9: Estimation error of the model for different adders and distance metrics

5.5 Experiments and Results

In our experiments, we characterized 8- and 16-bit ripple carry adder (RCA) and Brent-Kung adder (BKA) using 28nm-FDSOI technology. Table 5.2 shows the synthesis results (area, static plus dynamic power, critical path) of different adder configurations. Post-synthesis SPICE netlist of all the adders are generated and simulated using Eldo SPICE (version 12.2a). Table 5.3 shows the different operating triads used to simulate the adders. Clock period (T_{clk}) of the adders is chosen based on the synthesis timing report. Supply voltage (V_{dd}) of all the simulations is scaled down from 1.0V to 0.4V in steps of 0.1V and body-biasing potential (V_{bb}) of -2V, 0V, and, 2V. Pattern source function of SPICE testbench is configured with specific input vectors to test the adder configurations. The circuit under test is subjected to 20K simulations for every different operating triad with the same set of input patterns. Energy per operation corresponding to different operating triads is calculated from the simulation results. Output values generated from the SPICE simulation are compared against the golden (ideal) outputs corresponding to the input patterns. Automated test scripts calculate various statistical parameters like BER (ratio of faulty output bits over total output bits), MSE and bit-wise error probability (ratio of number of faulty bits over total bits in every binary position) for all the test cases.

TABLE 5.2: Synthesis Results of 8 and 16 bit RCA and BKA

Benchmarks	Area (μm^2)	Total Power (μW)	Critical Path (ns)
8-bit RCA	114.7	170	0.28
8-bit BKA	174.1	267.7	0.19
16-bit RCA	224.5	341	0.53
16-bit BKA	265.5	363.4	0.25

TABLE 5.3: Operating triads used in Spice simulation

Benchmarks	T_{clk} (ns)	V_{dd} (V)	V_{bb} (V)
8-bit RCA	0.5, 0.28, 0.19, 0.13	1 to 0.4	-2 to 2
8-bit BKA	0.5, 0.19, 0.13, 0.064	1 to 0.4	-2 to 2
16-bit RCA	0.7, 0.53, 0.25, 0.20	1 to 0.4	-2 to 2
16-bit BKA	0.7, 0.25, 0.20, 0.15	1 to 0.4	-2 to 2

Fig. 5.10 and Fig. 5.11 show the plots of BER vs Energy/Operation of 8-bit RCA and BKA adders. Likewise, plots of BER vs Energy/Operation of 16-bit RCA and BKA

adders are shown in Fig. 5.12 and Fig. 5.13 respectively. The label of the *x-axis* of the plots show the operating triads in the format T_{clk} (ns), V_{dd} (V), and V_{bb} (V) respectively. In all the adder configurations, energy/operation decreases and BER increases in sync with the supply voltage over-scaling. Table 5.4 shows the maximum energy efficiency (amount of energy saving compared to ideal test case) achieved by 8-bit and 16-bit RCA and BKA in different BER ranges. Table 5.5 shows the number of operating triads and BER at maximum energy efficiency for different adder configurations. Due to the parallel prefix structure, BKA adders show staircase pattern in BER plot shown in Fig. 5.11 and Fig. 5.13. On other hand, RCA adders based on serial prefix show exponential pattern in BER plot.

Energy/operation curve of all the four plots show two patterns corresponding to 0% BER, and BER greater than 0%. Effect of voltage over-scaling is visible in the left half of the plots, where energy/operation is gradually reduced in sync with the reduction in V_{dd} while BER is at 0%. Another important observation is that the effect of body-biasing is helping to keep the BER at 0% in this region of the plot. Both the 8-bit RCA and BKA adders operated at 0.5V V_{dd} with forward body bias of 2V V_{bb}, achieve a maximum energy efficiency of 76% and 75% respectively at 0% BER. Similarly, 16-bit RCA and BKA achieve a maximum energy efficiency of 60% and 59% respectively at 0% BER, while operating at 0.6V for V_{dd} with forward body bias V_{bb} of 2V. This set of operating triads provides high energy efficiency without any loss in accuracy of the computation by taking advantage of near-threshold computing and body-biasing technique.

TABLE 5.4: Energy Efficiency in 8-bit and 16-bit Ripple Carry and Brent-Kung Adders

BER Range	Number of Triads				Max. Energy Efficiency (%)			
	8-RCA	8-BKA	16-RCA	16-BKA	8-RCA	8-BKA	16-RCA	16-BKA
0%	16	14	15	18	76	75.3	60.5	73.3
1% to 10%	15	7	15	9	87	65.3	83.6	84
11% to 20%	2	5	6	3	74	89	86.2	73.3
21% to 25%	3	2	2	–	92	82.8	90.8	–

On the right half of the BER vs Energy/Operation plots, where the BER is greater than 0%, energy curve starts in three branches and tapers down when the BER reaches 40% and above. In those three branches, operating triads with body-biasing are the most energy efficient followed by triads without body-biasing and finally triads with overclocking. In 8-bit BKA adder, 28 out of 43 operating triads operate within 0% to

TABLE 5.5: BER in 8-bit and 16-bit Ripple Carry and Brent-Kung Adders

BER Range	Number of Triads				BER at Max. Energy Efficiency (%)			
	8-RCA	8-BKA	16-RCA	16-BKA	8-RCA	8-BKA	16-RCA	16-BKA
0%	16	14	15	18	0	0	0	0
1% to 10%	15	7	15	9	8	8.8	6	9.1
11% to 20%	2	5	6	3	11	16.1	17.5	18.1
21% to 25%	3	2	2	–	22	25	22.1	–

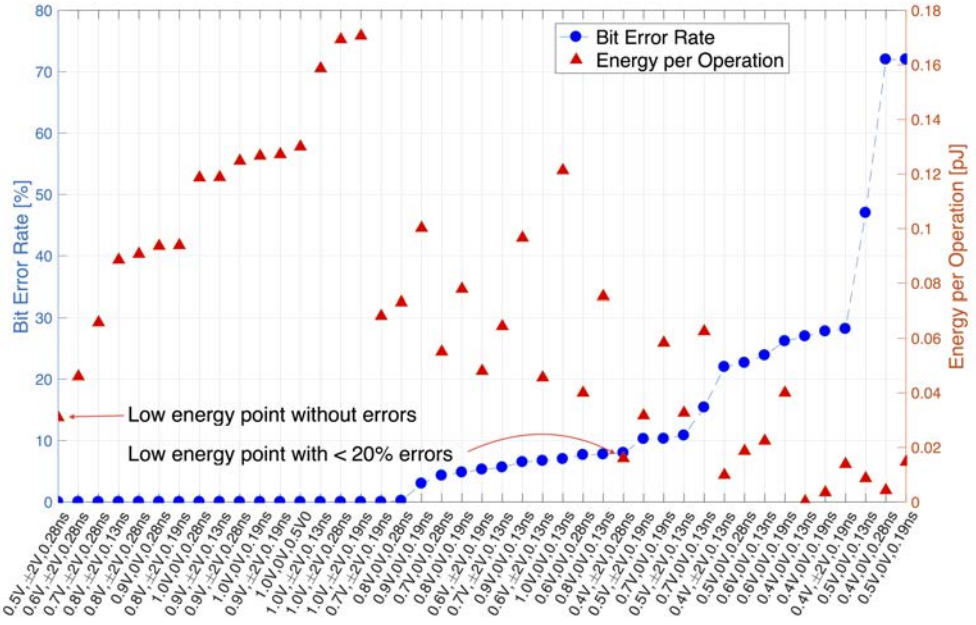


FIGURE 5.10: Bit-Error Rate vs. Energy/Operation for 8-bit RCA

25% BER. Similarly, 36 triads operate within 0% to 25% BER in 8-bit RCA adder. In 16-bit BKA and RCA adders, correspondingly 30 and 38 operating triads operate within BER range of 0% to 25%. For an application with an acceptable error margin of 25%, 8-bit RCA, 8-bit BKA, 16-bit RCA, and 16-bit BKA can be operated at V_{dd} =0.4V with forward body bias V_{bb}=2V to achieve the maximum energy efficiency of 92%, 89%, 90.8%, and 84%, respectively.

Approximation in arithmetic operators based on voltage over-scaling, provides *dynamic approximation*, which makes the user to control the energy-accuracy trade-off efficiently by changing the operating triad of the design at runtime. In this method, a *dynamic approximation* can be achieved without any design-level changes or addition of extra logic in the arithmetic operators unlike accuracy configurable adder proposed

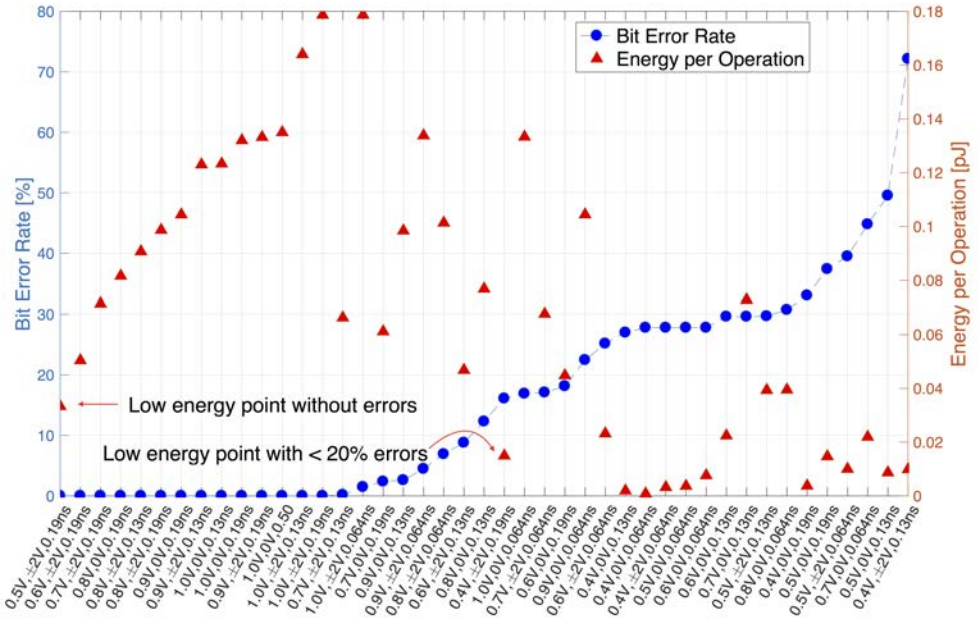


FIGURE 5.11: Bit-Error Rate vs. Energy/Operation for 8-bit BKA

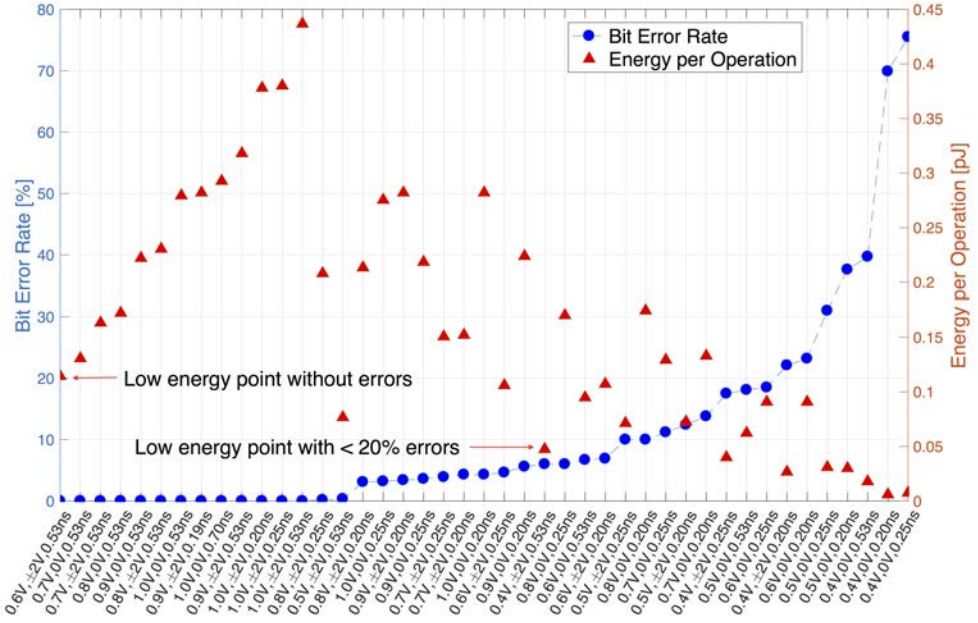


FIGURE 5.12: Bit-Error Rate vs. Energy/Operation for 16-bit RCA

in [67]. Dynamic speculation techniques like in [68] and Chapter 4 can be used to estimate the BER at runtime to switch between different triads to achieve high energy efficiency with respect to user-defined error margin. 8-bit RCA and BKA can be dynamically switched from accurate to approximate mode by merely scaling down V_{dd} from

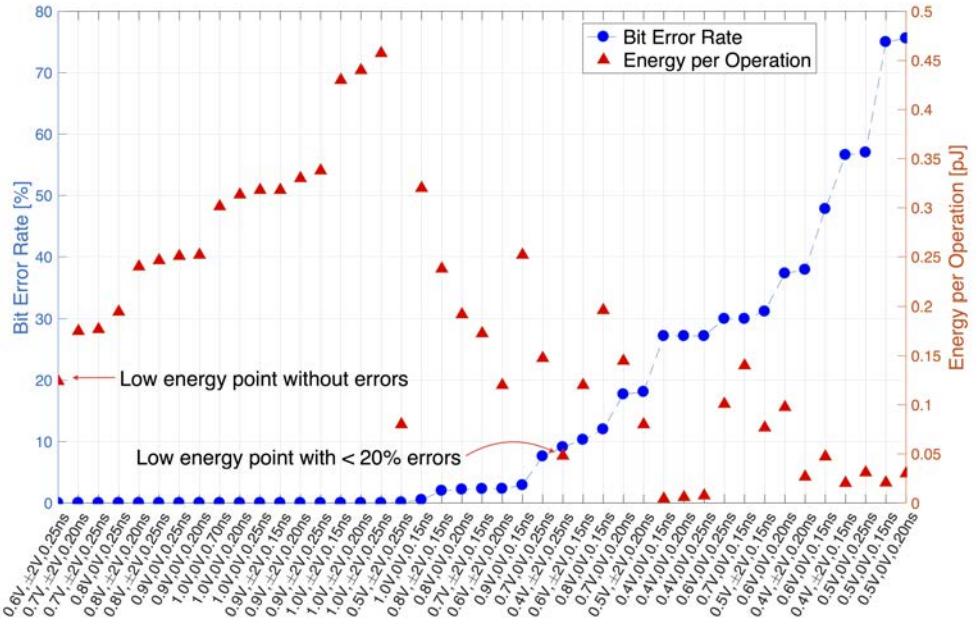


FIGURE 5.13: Bit-Error Rate vs. Energy/Operation for 16-bit BKA

0.5V to 0.4V at the cost of 8% BER to increase energy efficiency from 76% to 87%. Similarly in 8-bit RCA, switching from accurate to approximate mode is possible by reducing V_{dd} from 0.5V to 0.4V at the cost of 16% BER to increase energy efficiency from 75% to 89%. BKA adder configuration records more BER compared to RCA because of more logic paths of the same length due to parallel prefix structure. Likewise in 16-bit RCA, accurate to approximate mode can be switched by scaling V_{dd} from 0.6V to 0.4V at the cost of 6% BER to increase energy efficiency from 60% to 84%. In 16-bit BKA, accurate to approximate mode can be switched by scaling V_{dd} from 0.6V to 0.4V at the cost of 9% BER to increase energy efficiency from 59% to 84%. Both 16-bit adders provide leap of 24% increase in energy efficiency at the maximum cost of 9% BER compared to accurate mode.

5.6 Conclusion

In this chapter, voltage over-scaling is presented to highlight the possible trade-off between energy efficiency and approximation in arithmetic operators that can be used for error-resilient applications. In this work, different configurations of adders are characterized using different operating triads to generate a statistical model for approximate

adder. Likewise, different multiplier configurations are also characterized and the details are available in Appendix A. A framework has been laid down to construct statistical model by characterizing approximate operators based on voltage over-scaling. Maximum energy efficiency of 76% has been achieved in 8-bit RCA while operating at 0.5V V_{dd} without any accuracy loss. By increasing the effect of voltage over-scaling from 0.5V to 0.4V, energy efficiency is further increased to 87% at the cost of 8% BER. All the adder configurations have shown maximum energy gains of up to 89% within 16% of BER and 92% within 22% of BER. Dynamic approximation can be used in these adders by employing dynamic speculation method proposed in Chapter 4 to detect and correct errors at runtime.

Chapter 6

Energy Estimation Framework for VLIW Processors

6.1 Introduction

High performance at low power consumption has been the primary requirement in continuously evolving processor architectures. Parallelism has been the way to achieve high performance in processor architectures by splitting a task into multiple tinier chunks and executing them in parallel. Parallelism can be broadly classified into Hardware Parallelism and Software Parallelism. In simple terms, hardware parallelism is achieved by hardware multiplicity and sophisticated architecture. Hardware parallelism offers high performance at the cost of high silicon footprint. Advancement in fabrication technologies and reduction in feature size have made it possible to put even 100+ cores in general purpose processors. Hardware parallelism is implemented in different ways like pipelining, multi-core system, multiprocessor systems, etc. In a uni-processor environment, architectures like Pipelining, Superscalar, VLIW (Very Long Instruction Word) are the common form of hardware parallelism. A conventional processor would take multiple cycles to compute an instruction. Present day processors can issue multiple instructions per cycle. This increases the execution speed, thereby increasing the throughput of the processor multifold. A multiprocessor system with n processors, each processor capable of issuing m instructions per cycle, has the capability to compute nm threads of instructions in parallel.

To exploit the potential of hardware parallelism, software should support parallelism as well. Software parallelism is defined by the dependence of control and data in programs. Instruction Level Parallelism (ILP) is defined as number of instructions that can be executed simultaneously. ILP can be implemented either at hardware or software level. At the hardware level, the processor performs dynamic parallelism by deciding at runtime which instructions to execute in parallel. On the other hand, at the software level, the compiler performs static parallelism during compilation time. For applications like image and video processing, graphics, data mining, etc., there is a scope for high level of parallelism at instruction level. Also, these applications have the tendency to tolerate errors in computation, thereby allowing deliberate approximations in the computing. Earlier chapters discussed in detail the advantages and challenges in approximate computing in the context of near-threshold regime. In Chapter 5, the need for statistical models to represent the behaviour of arithmetic operators was also discussed.

In the rest of the chapter, scope for approximation in VLIW platform is discussed. A framework is formulated to estimate energy vs accuracy trade-offs of the execution unit of $\rho - VEX$ VLIW processor [69].

6.2 VLIW Processor Architecture

An VLIW is a processor architecture designed to exploit instruction level parallelism. Processor architecture based on pipelining can provide parallelism by partial overlap of instructions. Superscalar architecture evolved over pipelining architecture and execute two or more consecutive instructions in parallel. Parallelism of Superscalar is limited to data independent instructions. Pipelining architecture follows *In-Order* execution where instructions are executed in the order they appear in the program. Most of the time, consecutive instructions are not independent, which limits the parallelism offered by pipelining. To fully exploit the benefits of parallelism, instructions have to be executed based on data flow not in the order they appear in the program. This style of execution is called *Out-of-Order* execution. In *Out-of-Order* execution, the processor examines the instruction window of the consecutive instructions in re-order buffer. Independent instructions in instruction window with readily available data are executed by the processor. Superscalar architecture follows both *In-Order* (ex. PowerPC 604e)

and *Out-of-Order* (ex. PowerPC 620) execution. All these methods have complicated hardware to make the decisions internally to achieve parallelism.

In contrast, VLIW shifts the complexity to the compiler in deciding which instructions to be executed in parallel, resulting in simpler hardware. VLIW processors are made of multiple Reduced Instruction Set Computer (RISC) like functional units operating independently. RISC processors consume more power in order to handle resource utilization for a variety of applications. On the other hand, VLIW processor architecture is gaining more popularity in the embedded design due to its inherent low-power design (use of simple hardware) and high instruction level parallelism. Unlike superscalar designs, in VLIW processors, the number of functional units can be increased without additional sophisticated hardware to improve the parallelism. Different types of application require different processor organizations, thus a VLIW architecture implemented using reconfigurable platform supports a wide range of applications. Generic binaries can be used to dynamically change the processor organization to better suit the application need [70]. In the present trend of IoT, VLIW is used widely in image processing, computer vision, deep learning, etc. Movidius Myriad2 is a manycore vision processing unit based on VLIW processors with vector and SIMD operations capable for high speed parallelism in a clock cycle [71]. Some of the available commercial and academic VLIW platforms are TriMedia media processors by NXP, ST200 family by STMicroelectronics based on the Lx architecture, $\rho - VEX$ from TU Delft, Intel's Itanium IA-64 Explicitly Parallel Instruction Computing (EPIC), Elbrus 2000, etc.

In VLIW architecture, several operations are formatted in a single instruction thus it is termed as *Very Long Instruction Word (VLIW)*. By fetching one instruction, all the operations in the instruction are issued in parallel. Most common VLIW processor architectures contain a minimum of four execution units. Therefore four different operations can be issued in an instruction corresponding to four execution units. Unlike most superscalar designs where the instruction width is 32 bits or fewer, in VLIW the instruction width is 64 bits or more. The compiler analyses the data dependencies and resource availability to generate the sequence of instructions to be executed. Unlike superscalar, where complex instruction-dispatch logic blocks guess the branch-prediction to do the speculative computation of branch logic, VLIW uses heuristic or profiling to decide the direction of the branch. This results in simpler instruction-dispatch, less design time, reduced power consumption, good performance, and low cost. The compiler performs

various optimization like loop unrolling, aggressive inlining, software pipelining, trace scheduling, etc., to improve instruction level parallelism.

6.3 $\rho - VEX$ VLIW Architecture

In this work, $\rho - VEX$ an extensible, configurable soft-core VLIW processor based on VEX ISA (Instruction Set Architecture) [8] is used for experiments in the later sections. The VEX ISA support multi-cluster machines, where each cluster provides separate VEX implementation. Each cluster supports multiple issue widths. $\rho - VEX$ is designed with four stages consisting of fetch, decode, execute and writeback stages. $\rho - VEX$ follows the standard configuration of 1-cluster VEX machine. Fig 6.1, shows the schematic of 4-issue $\rho - VEX$ VLIW processor. Similar to a VEX cluster, $\rho - VEX$ has 4 Arithmetic Logic Units (A), 2 Multipliers (M), 1 Branch control (CTRL), 1 Memory access unit (MEM), 64 32-bit general purpose registers (GR), 8 1-bit branch registers (BR), and a program counter (PC) [69].

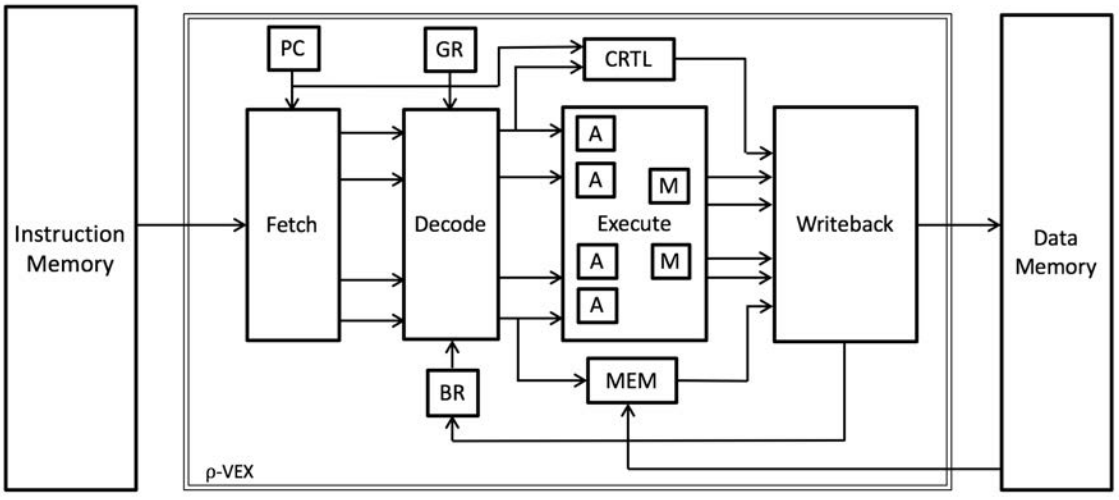


FIGURE 6.1: Schematic of $\rho - VEX$ Organization

6.3.1 $\rho - VEX$ Cluster Instruction Set

The $\rho - VEX$ cluster is based on VEX Instruction Set Architecture (VEX ISA). The VEX ISA is loosely modeled on the ISA of the HP/ST Lx (ST200) family of VLIW embedded cores. VEX ISA supports multi-cluster cores, with each cluster supporting

instructions with a distinct number of operations. In this work, 4-issue $\rho - VEX$ processor in single cluster configuration is used. The *Fetch* unit fetches a VLIW instruction from Instruction Memory, which is then decoded by the *Decode* unit. In *Decode* unit, operands required are fetched from the register contents of *GR* and *BR*. The instruction is executed in the *Execute* unit along with *CTRL* and *MEM* unit. All arithmetic and logical operations are carried out by the ALUs and Multipliers. The *CTRL* unit handles all jump and branch operations, while *MEM* unit handles all load and store operations. The *Writeback* unit performs all write activities to update *GR*, *BR*, *PC* and *MEM* units. Fig. 6.2 shows the instruction layout in 4-issue $\rho - VEX$ cluster. An instruction consists of four 32-bit syllables, with each syllable containing opcode bits, register addresses, and meta data. Also, allowed issue-slots per Functional Units (FU) are also depicted in Fig. 6.2. Available four ALUs are distributed equally to all the four syllables. Two multipliers are allocated to syllables 2 and 1, *MEM* and *CTRL* units are allocated to syllables 3 and 0 respectively. The $\rho - VEX$ cluster supports 73 instructions of which 42 instructions for ALU, 11 multiplier instructions, 11 control instructions, and 9 instructions for memory operations. Description and semantics of all the instructions in VEX ISA are provided in [72].

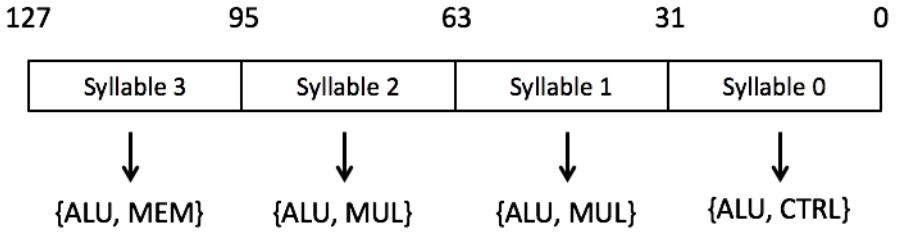


FIGURE 6.2: Instruction layout of $\rho - VEX$ cluster

6.3.2 $\rho - VEX$ Software Toolchain

The software toolchain for $\rho - VEX$ provided by [73] and [69] contains VEX C compiler, ANSI C libraries, VEX Simulation system, and $\rho - ASM$ (assembler for the $\rho - VEX$ processor). Fig. 6.3 shows the toolchain for $\rho - VEX$ processor. The first level C compiler converts a C program into a VEX executable. In this step, the C program is subjected to C preprocessing, then converted to *.s* assembly language (ASM) file. The *.s*

file is fed to compiled simulator environment, where *.s* file is converted to *.cs* file. Host C compiler processes the *.cs* file and generates host object files. Host linker links simulation support library, cache simulation library, VEX C library, and application object files. At the end, a VEX executable is generated that can be executed by invoking it.

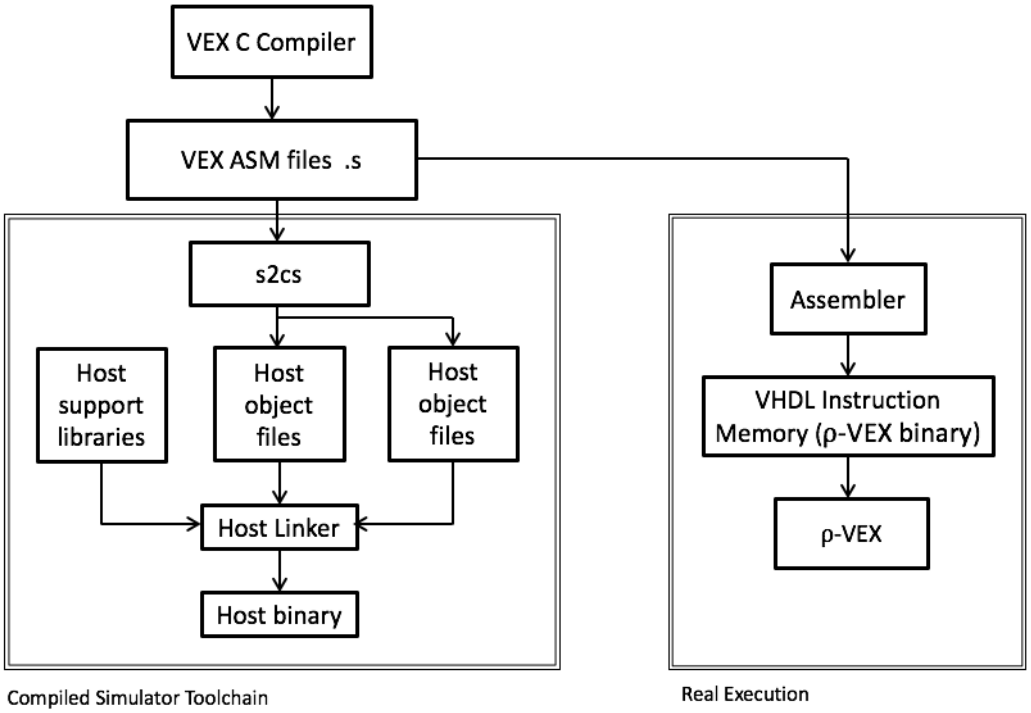


FIGURE 6.3: Toolchain for $\rho - VEX$ processor

In the flow of compilation and simulation, the intermediary files like VEX assembly file (*.s*) and compiled-simulator translated file (*.cs.c*) are generally masked and the final VEX executable is created. In addition to the standard semantic of the instructions, *.cs* file also contains other interesting statistics and information can be used to estimate energy consumption at the instruction level. The VEX compiler uses *trace scheduling* in optimization phase. In *trace scheduling*, loops are converted into a sequence of codes by using loop unrolling and static branch prediction. By default VEX compiler performs *-O2* optimization, meaning minimal loop unrolling and trace scheduling. The *.s* shows the profile of trace scheduling, which can be used to understand the flow of execution. In the command line interface of the VEX C compiler *-mas_g flag* and *-S* attributes are used to output *.cs.c* and *.s* intermediary files.

6.4 Power and Energy Estimation in VLIW processors

Estimating energy is an important and necessary step to efficiently utilize VLIW processors. This allows the designer to manage hardware and software parallelism to achieve high energy efficiency with maximum performance. Estimating energy consumption at instruction level will provide a clear understanding of energy consumption by different modules of a cluster at different configurations. In literature, different levels of power estimation techniques for microprocessors are proposed to efficiently evaluate the software in terms of power consumption. Power estimation is broadly classified as follows based on different abstractions:

- System Level
- Instruction Level
- Micro-architecture Level
- RT Level
- Gate Level
- Transistor Level

Power estimation methods from system level to micro-architectural level are less accurate but faster than gate-level and RT-level power estimations. In literature, almost all of the power estimation techniques are based on Instruction-Level power models. At instruction level, power estimation is simpler, faster, and also provides enough details to under the behaviour of different modules of the processor.

6.4.1 Instruction-Level Power Estimation

Instruction-level power estimation (ILEP) provides a clear understanding of power cost vs. software implementation in a microprocessor system. Once the power models are generated, power estimation for any application can be obtained by simple C compilers. Also, it helps to make best design decisions at higher abstraction level [74]. Instruction-level power consumption technique is based on measuring the current drawn by the processor while it executes certain instructions repeatedly. Based on the current consumption pattern, power models are designed to represent the execution behaviour of

the processor for any particular instruction [75]. This method is very accurate and uses linear regression with instruction and architectural level variables such as average bit switching activity in the instruction register, address bus, etc. [74]. Another way of generating power models is discussed in [76] by estimating power cost for modules based on average capacitance that would switch when the given module is activated. Once the power costs for different modules like ALU, control units, and memory are determined, average power consumption of instructions using these modules are computed by simple arithmetic.

In general, power models described above are proposed for hardcore processors like Intel 486DX2, Intel i960 family, etc. Most of the power models provide estimation on the average power consumption of the processor for a given application. Based on those estimations, the designer can modify the hardware configuration or software parameters to reduce the power consumption of any given program. These methods lack the dynamicity to provide power consumption for different configurations at the same time, so the user has to repeat the process to estimate average power for different configurations. When considering VLIW processors, traditional instruction-level power modeling approaches imply new challenges because of varying issue width capability in VLIW processors [77]. Also, estimation based on power model at instruction-level lacks insight on the causes of power consumption within the cluster [78]. This limitation is addressed in [79] by generating power models at the micro-architectural level rather than at instruction-level.

In [74], gate-level energy models are presented to do instruction level power analysis. In this method, two different implementations of the microcontroller are studied. The energy consumption of each instruction is different in each of those implementations. By studying these differences and data correlation on the energy consumption of those instructions, parts of the design can be resynthesized for low power application. Based on the energy estimates of few instructions, energy consumption of few sample programs are predicted. In [80], analytical energy models are proposed to estimate the energy consumption of a nested loop executed on a VLIW ASIP for different vector widths. In this energy model, energy consumption is estimated for a vector computing unit and program as a function of vector width. The total energy of an instruction for any given

vector width w is represented as

$$E_t(w) = E_d(w) + E_s(w) \tag{6.1}$$

where $E_d(w)$ is dynamic energy spent in data path and $E_s(w)$ is static energy that depends on program execution time.

In [77], an energy model of a VLIW processor at instruction-level with the accuracy of the micro-architectural level is proposed. On contrast to traditional energy model, where instruction energy for each given clock cycle is estimated without considering the other instruction in the pipeline, the pipeline-aware method proposed in [77] considers the overlapping of instructions and estimates more accurate energy consumption. The estimation of energy consumed by the processor during the execution of the sequence w with N instructions is expressed as

$$E(w) = \sum_{\forall n \in N} (E(w_n \mid w_{n-1}) + E_c + c_o) \tag{6.2}$$

where $E(w_n \mid w_{n-1})$ is the energy dissipated by the data path associated with the execution of instruction w_n, E_c is the total energy dissipated by the control unit and c_o is the energy constant associated with start-up sequence of the processor. Energy dissipated by instruction w_n is estimated by considering the overlap of previous instruction w_{n-1} in the pipeline.

In [78], power models for instruction-level power estimation with RTL-level accuracy are proposed. In this method, the average energy consumption of an instruction w_n is

$$E(w_n) = B + \alpha_n \times c_{syl} + m \times p \times S + l \times q \times M \tag{6.3}$$

where B is the average energy base cost (while executing NOPs), α_n is the number of syllables different from NOPs, c_{syl} is average energy consumption associated with execution of a syllable. The third term is the summation of additive average energy consumption due to a miss event on the D-cache. The fourth term is related to I-cache misses. Similar approach is used in [81], where the base energy is associated with execution of NOP energy and incremental energy is added for each of the instructions in the execution set. The inter execution-set is modeled as linear equations to represent functional to functional instruction switching, function to NOP switching, and variability in the length

of execution. In [82], power models in [78] are improved to estimate inter-instruction energy consumption by characterizing each operation in isolation, then by clustering operations considering their average energy cost known as *instruction clustering*.

All these power macro-models used to estimate energy consumption at instruction-level are developed in the context of standard (normal) operating conditions. For processor designs with ultra-low power techniques like voltage over-scaling the traditional power models lack the capability to estimate instruction-level energy consumption where the supply voltage is dynamically changed to counter the variability effects. Also, in near-threshold regime, energy metric is predominately used to measure the gains of voltage scaling techniques. This demands a need for an energy estimation method for VLIW processors in the scope of near-threshold regime.

In the rest of the chapter, an instruction-level energy estimation method for execution unit of $\rho - VEX$ cluster is presented. In this work, the energy estimation is in the context of near-threshold regime and approximate computing. Therefore, along with energy metric, error metric is also estimated to highlight the reduction in accuracy. Energy estimation at high abstraction level are much faster than transistor-level based estimation methods. But the higher abstraction level energy estimations lack the accuracy when compared to transistor-level. On average, gate-level energy estimations are 10% to 15% less accurate than the transistor-level estimations [83]. Primary objective of this work is to propose a framework to estimate energy consumption of execution unit of $\rho - VEX$ cluster with the accuracy of transistor-level and at the speed of instruction-level methods.

6.5 Energy Estimation in $\rho - VEX$ processor

The execution unit of $\rho - VEX$ cluster consists of 4 ALUs and 2 Multipliers. Fig 6.4 shows the block diagram of 32-bit ALU, and Multiplier (MUL) in the $\rho - VEX$ cluster of Fig 6.1. Different operations of the ALU and MUL are chosen by 6-bit opcodes, *alu_op* and *mul_op* respectively. In this work, instruction level energy estimation is done in two phases as follows.

- Characterization: In this phase to obtain RT-level accuracy, the execution unit of $\rho - VEX$ cluster is characterized for different operating triads.

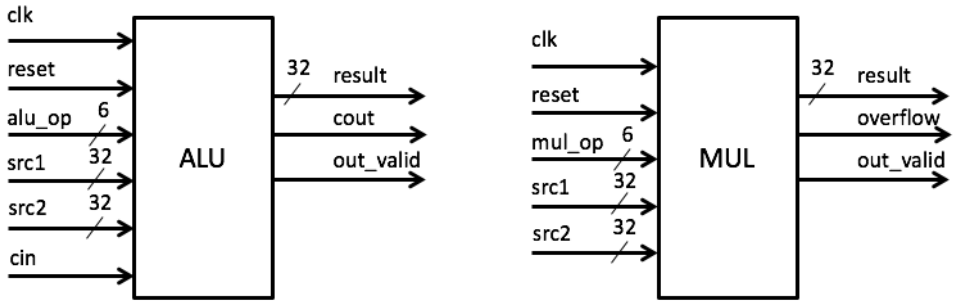


FIGURE 6.4: Block diagram of ALU and MUL units in $\rho - VEX$ cluster

- Estimation: Based on the characterization of the execution unit, energy estimation is done using C compiler for different instructions for different operating modes.

6.5.1 Characterization of Execution Unit in $\rho - VEX$ Cluster

To determine accurate energy consumption by the module like ALUs and Multipliers, the execution unit of $\rho - VEX$ cluster is characterized based on SPICE simulation. Fig. 6.5, shows the schematic of characterization process of the execution unit. In this method, HDL description of the execution unit is synthesized using Synopsys Design Compiler (version H-2013.03-SP5-2). Inherent low leakage FDSOI 28nm design library and user-defined timing constraints are used to synthesize the execution unit.

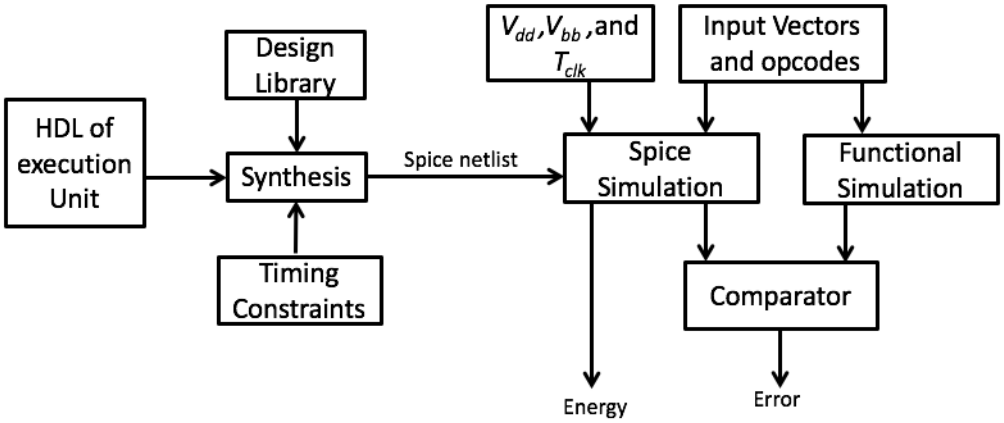


FIGURE 6.5: Schematic of characterization of execution unit of $\rho - VEX$ cluster

The SPICE netlist from the synthesizer is simulated using Mentor Eldo SPICE simulator. Transistor-level simulations are performed for different operating triads chosen based on the synthesis timing report. An 80-bit Linear Feedback Shift Register (LFSR) is used to generate random input vectors. In the testbench, pattern source function of SPICE is used to provide different opcodes for the module under test. Outputs of 1K SPICE simulations of each operating triads are compared with the output from the functional simulation done at gate level to find the errors. Energy consumed per operation is measured in SPICE simulation for different operations. In the simulations, supply voltage V_{dd} is varied from 1V to 0.4V. At 1V, performance of the operator is high without any errors. Then V_{dd} is reduced in steps of 0.1V till it reaches near-threshold regime of 0.4V. When V_{dd} decreases, the propagation delay of the combinational logic increases, thereby resulting in timing errors. In FDSOI technology, the back biasing technique is used to vary the threshold voltage V_t of the transistor in order to either increase the performance or to reduce the leakage. In the simulations, the back biasing voltage V_{bb} is varied between 0V or $\pm 2V$.

Fig. 6.6 shows the energy per operation versus bit error rate plot of ALU performing addition in different operating triads. From the plot it is evident that when V_{dd} = 1V and V_{bb} = 2V, energy per operation is at its maximum. Fig. 6.7 is a zoomed version of the left side of the plot Fig. 6.6. Fig. 6.7 shows the trend of voltage scaling and its impact in reducing the energy per operation. When the V_{dd} = 0.5V, the energy per operation is reduced by 85% without any bit errors. Fig. 6.8 is a zoomed version of the right side of the plot Fig. 6.6. For an application that can tolerate BER of 10%, V_{dd} can be further reduced to 0.4V and the clock period is halved to increase energy efficiency to 93% with the acceptable BER of 7.8%. Other operations of the ALU also show the similar characteristics. Fig. 6.9, and Fig. 6.10 show the similar plot for subtraction and division operations respectively. For subtraction operation, the maximum energy efficiency of 85% without any bit errors is achieved at V_{dd} = 0.6V. By pushing the limits of voltage scaling and back biasing, energy efficiency can be increased to 91% with BER of 4,7% while operating at V_{dd} = 0.4V and V_{bb} =2V. Similarly, for division operation, energy efficiency of 88% is achieved while operating at V_{dd} = 0.5V and V_{bb} =2V at the cost of 9.4% BER.

Fig. 6.11, Fig. 6.12, and Fig. 6.13 show the plot of energy per operation versus bit error rate for logical operations AND, OR, and XOR respectively. Like arithmetic

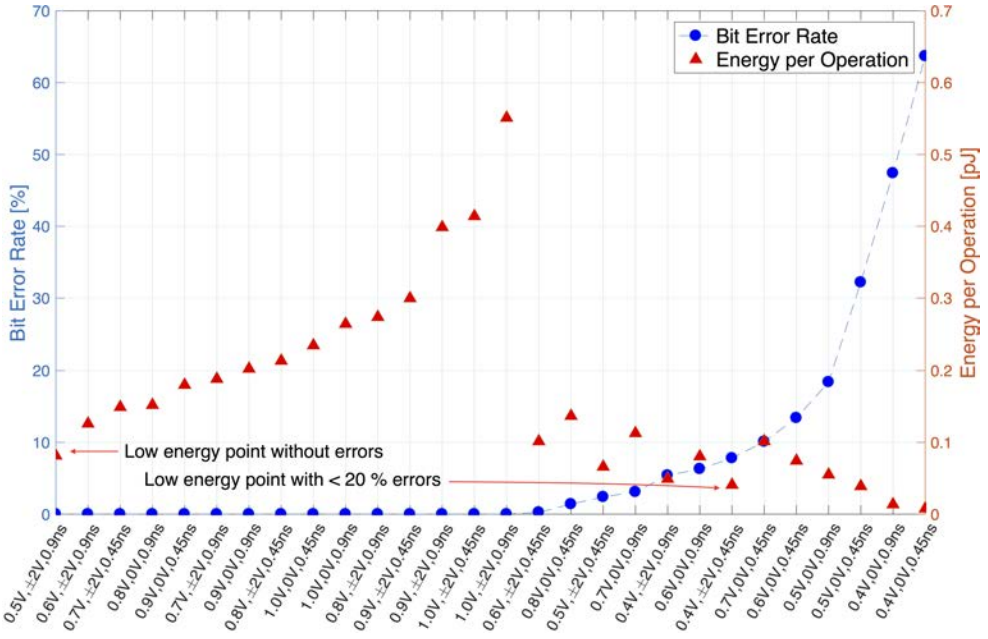


FIGURE 6.6: Energy versus Bit Error Rate (BER) plot of ALU executing ADD operation

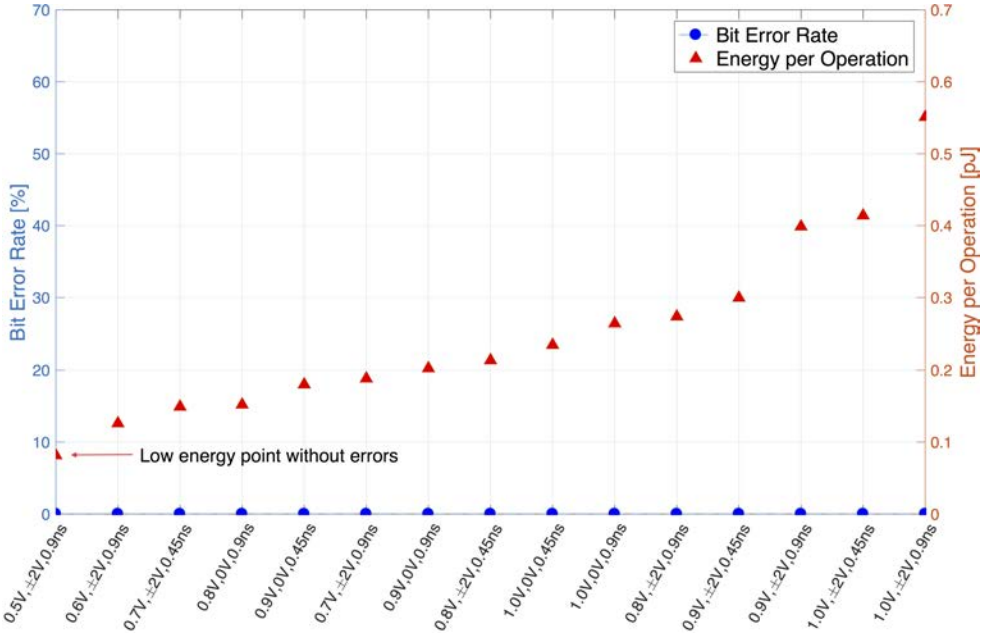


FIGURE 6.7: Voltage scaling of ALU to achieve high energy efficiency

operations, logical operations also show the similar trend of energy versus bit error rate. In AND operation, energy efficiency of 91% is achieved with BER of 5% by reducing V_{dd} to 0.4V with V_{bb} =2V. Likewise, energy efficiency of 92% with BER of 8.5% is achieved for OR operation. And XOR operation reaches 91% energy efficiency with 5.7% while operating in V_{dd} to 0.4V with V_{bb} =2V. Back-biasing technique improves the performance

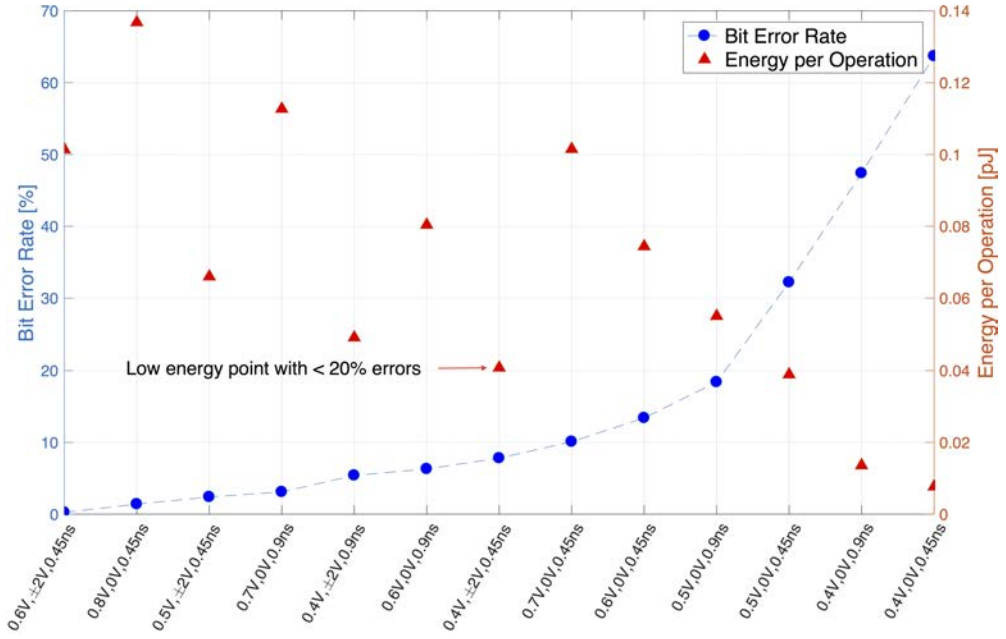


FIGURE 6.8: Bit Error Rate (BER) plot of ALU and scope for approximate computing

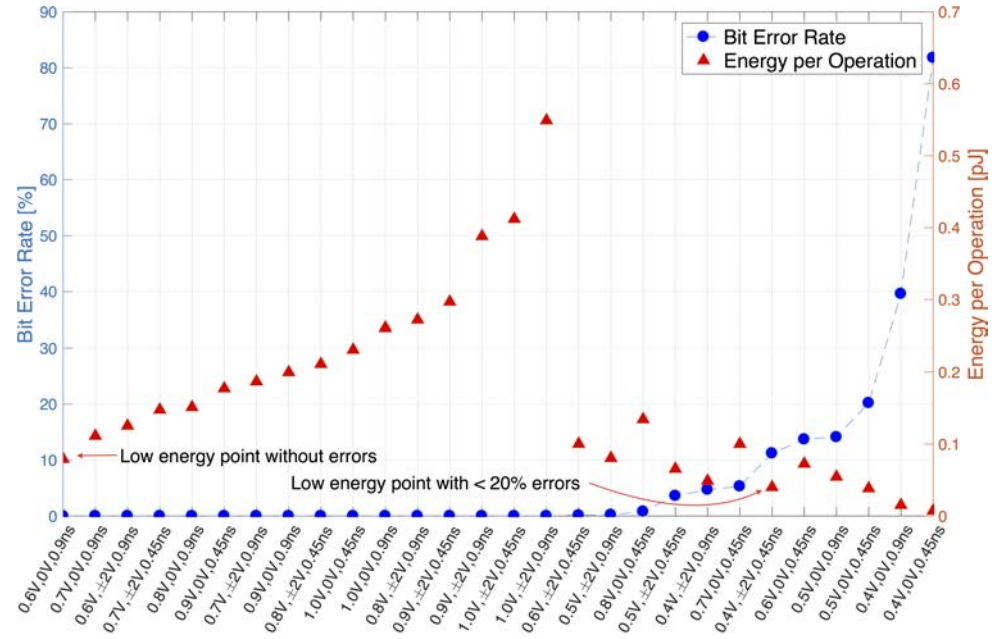


FIGURE 6.9: Energy versus Bit Error Rate (BER) plot of ALU executing SUB operation

of the system by consuming more energy per operation. As shown in the plots, in super-threshold regime (V_{dd} = 0.7V to 1.0V), there is no benefit of back-biasing technique, since the BER is same for operating triads with and without errors. While operating at near-threshold regime (V_{dd} = 0.4V to 0.6V), benefits of back-biasing technique are clearly evident.

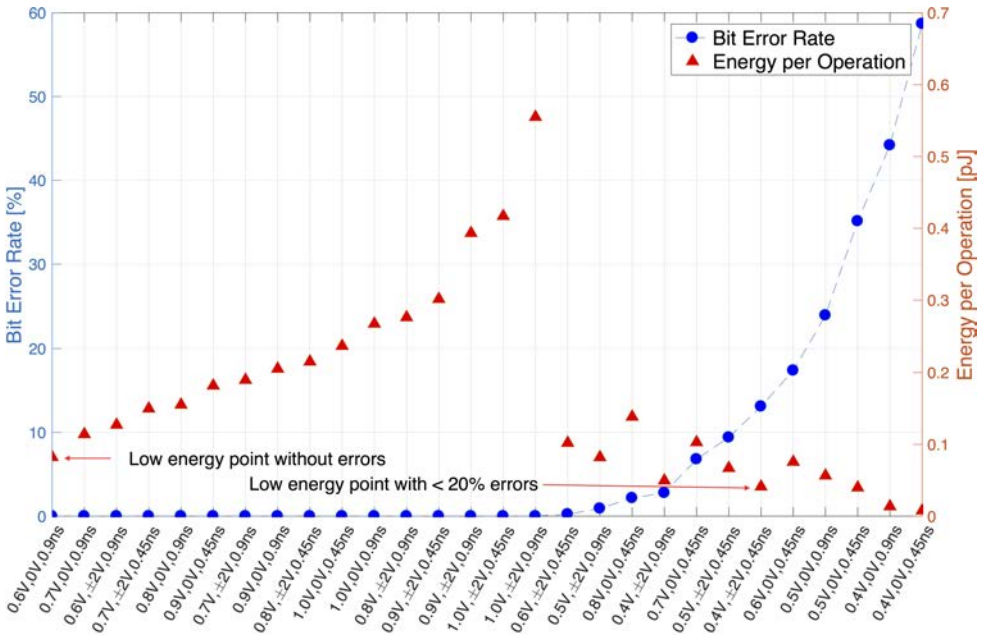


FIGURE 6.10: Energy versus Bit Error Rate (BER) plot of ALU executing DIV operation

The existing energy and power models for instruction level energy/power estimation of VLIW platforms are not developed in the context of near-threshold regime. This limitation reduces the usability of such models for applications like image and video processing, data mining, etc., where there is a scope for scaling the supply voltage to near-threshold regime to extract higher energy efficiency with the cost of acceptable timing errors as defined by the user. The proposed characterization method is used to characterize the behaviour of the execution unit of $\rho - VEX$ cluster for a variety of instructions and for different operating triads. With the extracted energy and error information for various operations, instruction-level energy can be estimated by formulating a framework for benchmark programs based on C language.

6.5.2 Proposed Instruction-level Energy Estimation Method

In this work, we propose functionality based energy estimation method. Energy consumption of different instructions are estimated based on the characterization of execution unit of $\rho - VEX$ cluster discussed in the previous section. Fig. 6.14 shows the execution unit (highlighted) of the $\rho - VEX$ cluster. In general, execution of an instruction happens in different pipeline stages like fetch, decode, execute, and write-back. Energy consumed by each of these stages are denoted as E_{fetch}, E_{decode}, E_{tot_exec},

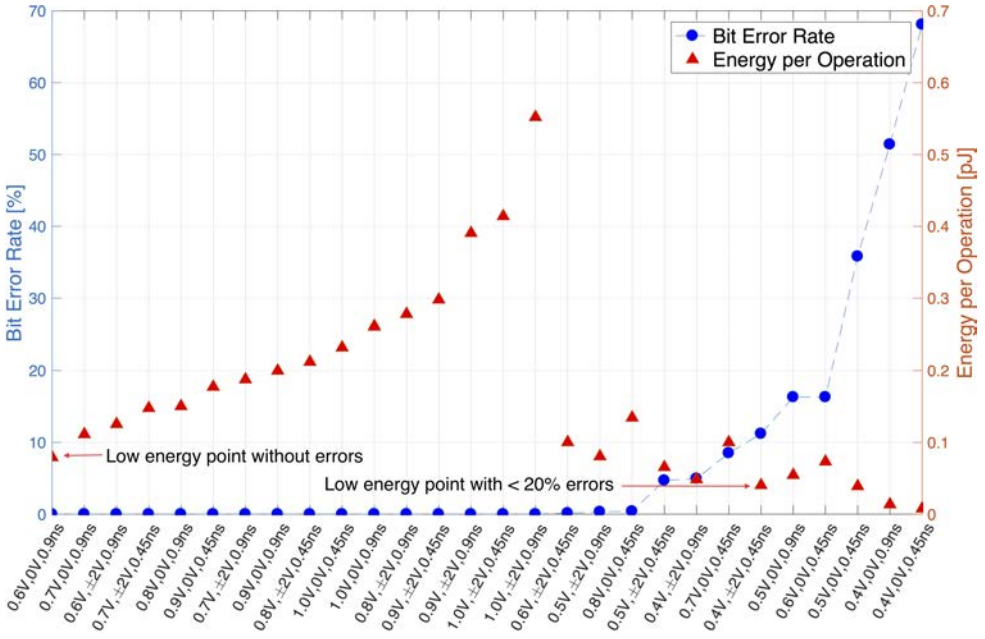


FIGURE 6.11: Energy versus Bit Error Rate (BER) plot of ALU executing AND operation

E_{reg}, $E_{writeback}$ respectively. Sum of the all these components gives the total energy consumed to execute an instruction. In the total energy consumed by an instruction, execute stage dominates when compared to other stages in the pipeline. The primary focus of this work is to accurately estimate the energy consumed by execute stage for variety of instructions in different operating conditions. In the previous section, the characterization procedure is detailed for ALU and Multiplier of the $\rho - VEX$ cluster for different operating conditions. There are totally 73 instructions in $\rho - VEX$. Each instruction based on the type of operation excites certain blocks of the cluster. For example, in an instruction with two additions and two multiplications, syllables 0 and 3 of the instruction will be allocated for addition and syllables 1 and 2 will be allocated for multiplication. Based on the characterization of ALU and MUL units, energy consumption of the instruction for any given operating condition is determined as the sum of the energy consumed by all the four functional units. Since the functional units are characterized for various operating triads, it is very simple to estimate the energy for any benchmark program by simple compilation using VEX compiler. This method can be easily adopted to estimate energy consumption for different processor organization without any need for re-characterization. Rest of the section explains the proposed energy estimation procedure of $\rho - VEX$ cluster's execution stage.

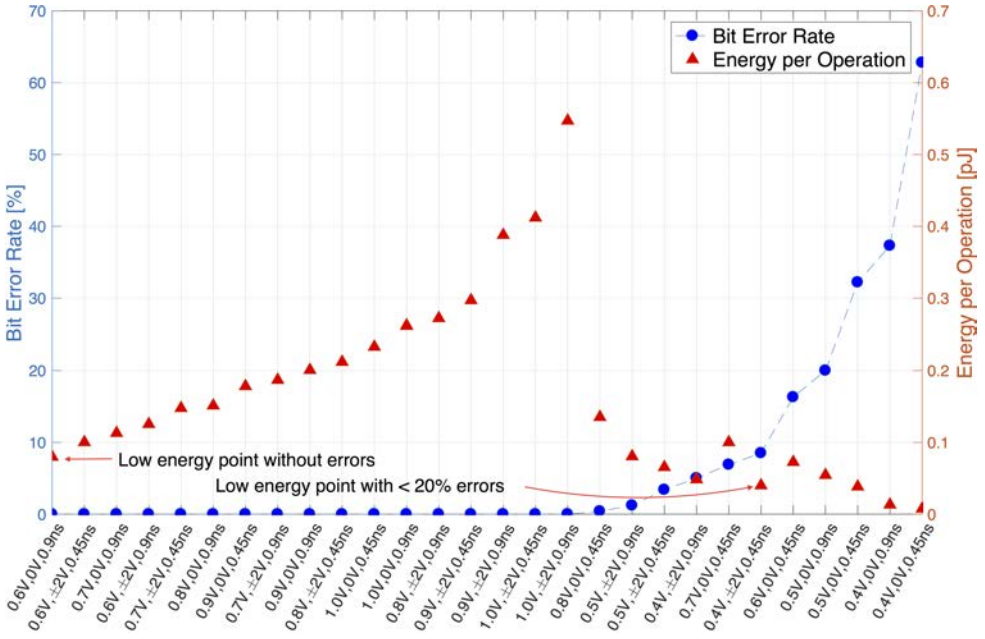


FIGURE 6.12: Energy versus Bit Error Rate (BER) plot of ALU executing OR operation

To estimate instruction-level energy consumption, benchmark programs are compiled in the VEX Toolchain and intermediary *.s* and *.cs.c* files are generated. The *.cs.c* file contains macro declaration for every VEX operations like ADD, XOR etc. In the previous section, energy per operation for different operations are calculated from the SPICE simulation. By knowing the number of times a particular operation is executed, it is possible to estimate the energy consumed for any given instruction. By expanding this method to all the instructions in a program, the total energy consumption of the program can be estimated. For a program P with sequence of N instructions $(i_1, i_2, i_3, ..., i_N)$, where each instruction i_n contains L different operations, the estimated energy of execution unit can be stated as follows

$$E_{tot}(i) = \sum_{\forall k \in L} E_{exe}(o_k) \times M_k \tag{6.4}$$

$$E_{exe}(P) = \sum_{\forall n \in N} E_{tot}(i_n) \tag{6.5}$$

where $E_{tot}(i_n)$ represents energy consumed by the execution unit to execute an instruction (i_n). $E_{exe}(o_k)$, is the energy consumed by an operation (o_k), and M_k is the number of times operation o_k is executed. The energy consumed by different operations $E_{exe}(o)$ is obtained from the characterization phase for different operating conditions. Therefore

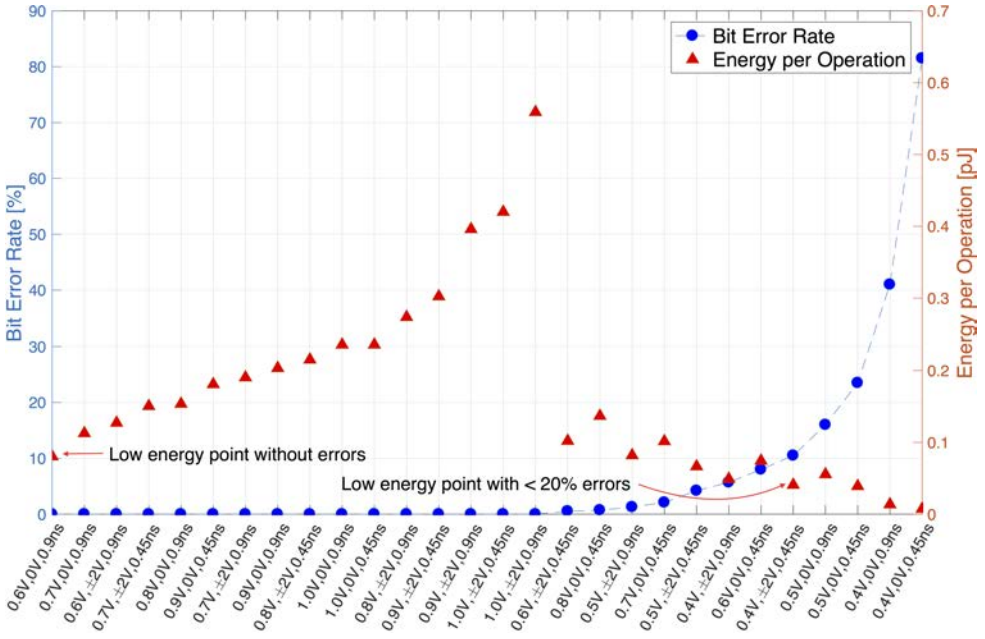


FIGURE 6.13: Energy versus Bit Error Rate (BER) plot of ALU executing XOR operation

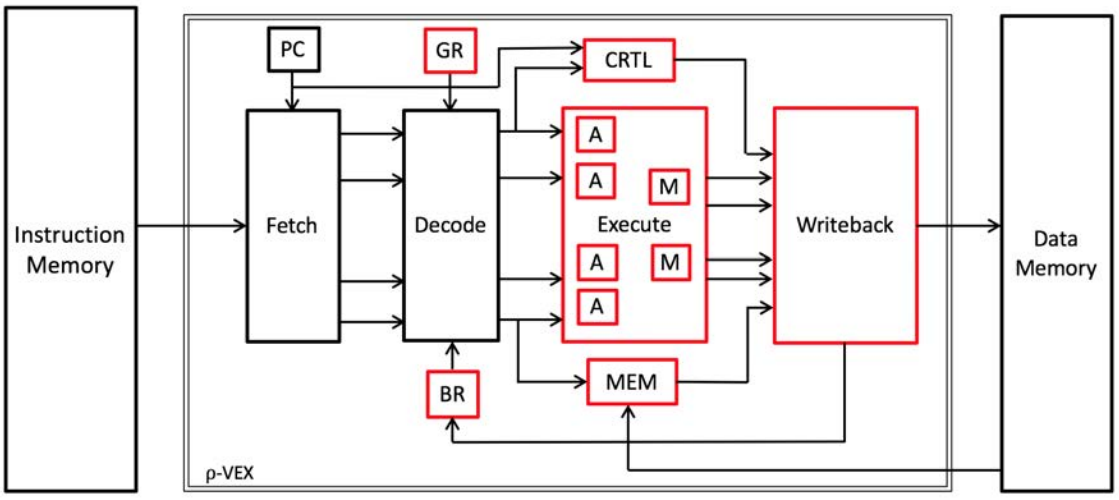


FIGURE 6.14: $\rho - VEX$ cluster blocks (highlighted) characterized of for energy estimation

Eqn. 6.4 represents the total energy consumed by the execution unit for a particular instruction i_n. Eqn. 6.5 represents the total energy consumption of the execution unit to execute a program P with N different instructions. Since the focus of this work is only the execution unit of the $\rho - VEX$ cluster, only the energy consumed by the execution unit is estimated. Considering arbitrary energy consumption for other modules, total

energy consumption of the $\rho - VEX$ cluster to execute P can be stated as

$$E_{total}(P) = E_{fetch}(P) + E_{decode}(P) + E_{totexec}(P) + E_{reg}(P) + E_{writeback}(P) \quad (6.6)$$

where $E_{fetch}(P)$, $E_{decode}(P)$, $E_{reg}(P)$, $E_{writeback}(P)$ represent energy consumed by fetch, decode, registers, writeback modules while executing the program P respectively.

Table. 6.1 shows the energy consumed by the execution unit for different types of instructions (ALU, multiplier, control, and memory). The energy is estimated for different supply voltage without body biasing. From Table. 6.1, it is evident that the multiplication consumes more energy followed by memory, control, and arithmetic and logic instructions. Most of the arithmetic and logical instructions consume almost the same amount of energy with few exceptions. Likewise, all the control instructions consume more or less same amount of energy. Instead of exhaustively estimating energy for all the instructions, based on the grouping of the instructions, energy estimation can be achieved. Table. 6.2 shows the energy consumption of instructions with $\pm 2V$ body biasing. Body biasing is used to improve the performance of the circuit at the cost of increased energy consumption. For MPY instruction at V_{dd}=1V, the energy consumption has doubled with body biasing results in 30% reduction in latency. Scaling down the V_{dd} to 0.4V towards near-threshold regime reduces the energy consumption of MPY by 18× compared to energy consumption at V_{dd}=1V. This gives the flexibility to the user to decide upon different operating points based on the desired energy vs. accuracy trade-off.

6.6 Validation and Proof of Concept

In this section, validation procedure for the proposed energy estimation method is discussed. A proof-concept of the proposed method is provided with details of the benchmark programs and experimental results.

6.6.1 Validation of Energy Estimation Method

Validation of an energy model is a necessary step to ensure the level of accuracy of energy estimation by the models. In the literature, energy/power model based energy estimation

TABLE 6.1: Energy consumption of different VLIW instructions

Instructions	Energy consumption in pJ (V_{bb}=0V)					
	V_{dd}=1V	V_{dd}=0.8V	V_{dd}=0.7V	V_{dd}=0.6V	V_{dd}=0.5V	V_{dd}=0.4V
sxtb	0.293	0.175	0.144	0.108	0.07	0.0145
cmp	0.276	0.163	0.124	0.09	0.057	0.0139
slct	0.29	0.169	0.132	0.098	0.063	0.0142
mov	0.292	0.172	0.138	0.105	0.065	0.0142
shr	0.29	0.169	0.132	0.098	0.063	0.0142
min	0.267	0.153	0.118	0.082	0.055	0.0137
orc	0.259	0.149	0.11	0.078	0.054	0.0136
xor	0.266	0.153	0.112	0.08	0.055	0.0136
add	0.264	0.152	0.112	0.08	0.055	0.0136
sub	0.261	0.151	0.111	0.79	0.054	0.0152
and	0.261	0.15	0.111	0.79	0.054	0.0136
mpy	0.296	0.196	0.154	0.12	0.077	0.0163
br	0.271	0.158	0.12	0.084	0.055	0.0137
goto	0.282	0.165	0.128	0.094	0.061	0.014
return	0.276	0.163	0.124	0.090	0.058	0.0139
xnop	0.276	0.163	0.124	0.090	0.057	0.0139
ldb	0.292	0.172	0.138	0.106	0.068	0.0143

is validated against gate-level simulation on the set of benchmarks to determine the accuracy of the models. In [78], RTL macro-model of a VLIW core, register file, and cache are validated against transistor-level simulation of the circuit. In the proposed energy estimation method, the execution unit of the $\rho-VEX$ cluster is characterized at transistor-level for different operating triads. Energy estimation is done by C compiler at higher abstraction level based on the data from transistor-level characterization and the user-defined operating conditions. Since the proposed estimation method is based on transistor-level characterization, to validate this method hardware prototype is required to perform real-time experiments. Due to unavailability of hardware prototype, we devise an alternate validation procedure for the proposed energy estimation method.

In this work, the energy estimation done at higher abstraction level is validated by comparing with the energy estimates at the transistor-level simulation for the selected benchmarks. Transistor-level simulations provide most accurate energy estimates by simulating the benchmark program at SPICE level. In this work, an entire $\rho-VEX$ cluster is synthesized using 28nm FDSOI technology and SPICE simulation of the benchmark programs are performed using ELDO SPICE. The energy estimation of the benchmark program from the proposed method is compared with full SPICE simulation to

TABLE 6.2: Energy consumption of different VLIW instructions with body bias

Instructions	Energy consumption in pJ (V_{bb}=±2V)					
	V_{dd}=1V	V_{dd}=0.8V	V_{dd}=0.7V	V_{dd}=0.6V	V_{dd}=0.5V	V_{dd}=0.4V
sxtb	0.582	0.295	0.214	0.13	0.0835	0.0514
cmp	0.569	0.28	0.198	0.1285	0.082	0.0504
slct	0.575	0.287	0.208	0.1293	0.0828	0.0509
mov	0.58	0.293	0.21	0.1298	0.0831	0.0513
shr	0.575	0.288	0.208	0.1295	0.0828	0.0509
min	0.56	0.276	0.194	0.1276	0.0815	0.0496
orc	0.544	0.2696	0.1855	0.1242	0.08	0.048
xor	0.559	0.2768	0.1904	0.1272	0.0802	0.0492
add	0.551	0.2736	0.1883	0.126	0.0815	0.0492
sub	0.549	0.272	0.1869	0.1254	0.0805	0.0488
and	0.552	0.2728	0.1876	0.1254	0.0805	0.0484
mpy	0.599	0.31	0.225	0.141	0.085	0.0523
br	0.564	0.278	0.196	0.128	0.0818	0.0498
goto	0.573	0.285	0.205	0.129	0.0825	0.0507
return	0.57	0.28	0.198	0.1285	0.082	0.0504
xnop	0.57	0.28	0.198	0.1285	0.082	0.0504
ldb	0.58	0.293	0.21	0.1298	0.0831	0.0514

determine the accuracy of the proposed method.

The validation is performed for test cases of different ALU operations with different addressing modes listed as following.

- In the *scenario 1*, the same operation is repeated multiple times in the same addressing mode.
- In the *scenario 2*, different addressing modes of the same operation are executed in sequence to estimate the difference in energy consumption due to varying addressing modes while the functionality is same.
- In the *scenario 3*, different operations with different addressing modes are used.

SPICE simulation of benchmark program on $\rho - VEX$ cluster takes a substantial amount of time and computing resources. In order to simplify the validation process, test program with different arithmetic and logical operations is created inline with the above listed test scenarios. In order to further simplify the validation process, some modules of the VLIW cluster are executed at C level. The test program is compiled and assembled by the VEX C compiler. The output ASM code is translated to series of opcode signals

in SPICE format by a translator script developed in this work. The translated opcode signals are fed to the execution unit of $\rho - VEX$ cluster for transistor-level simulation with random inputs. This method gives the accurate energy consumption of the test program next to real time hardware implementation. The energy consumption data from the transistor-level simulation are compared with the data of the energy estimation method proposed in this work.

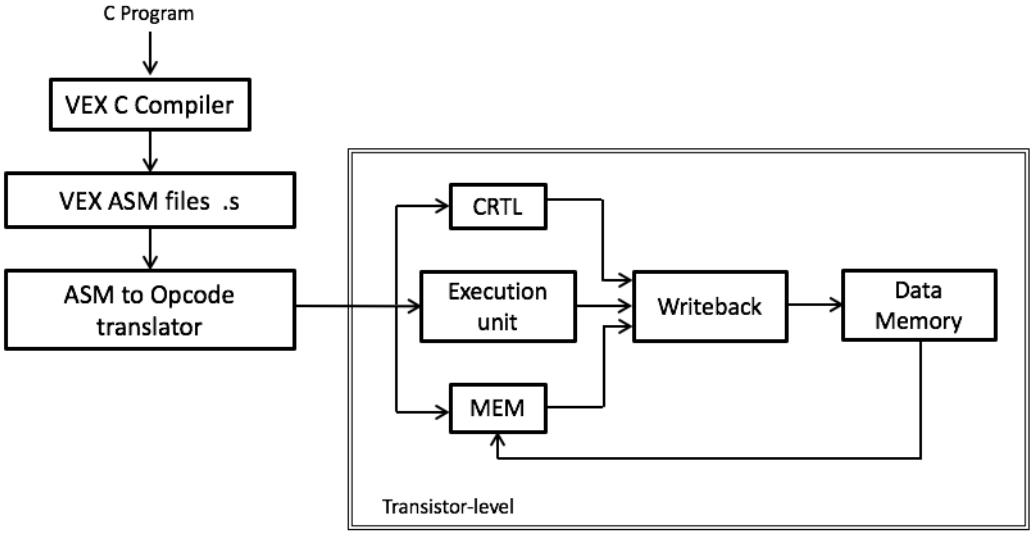


FIGURE 6.15: Flow graph of validation method

In *scenario 1*, register-direct and immediate addressing modes are used for different arithmetic and logical operations. The test program with the sequence of instructions is simulated using ELDO SPICE. From this experiment, we observed no big difference in the energy consumption of any of the arithmetic and logical operations while repeating in the same addressing mode. In *scenario 2*, all the arithmetic and logical operations in immediate addressing mode consume slightly more energy compared to register-direct addressing mode. *Scenario 3* is used to study the inter-instruction effects while executing different operations. For the energy estimation methods, where the base energy is computed as NOP and incremental energy is augmented with base energy to estimate energy consumption for different operations. In such models, inter-instruction effects have to be compensated by arbitrary offset value or by separate models for inter-instruction energy proposed in [81]. In this work, we propose energy estimation on functional-level by studying the impact of different instructions on different modules of the $\rho - VEX$ cluster. Therefore, inter-instruction effects are inherently handled in the

proposed method, which voids the need for additional offset to improve the accuracy of the estimated energy.

TABLE 6.3: Validation of proposed energy estimation method

Test scenario	Average Error	Maximum Error
Scenario 1 (immediate addressing mode)	0.23%	1.13%
Scenario 1 (register-direct addressing mode)	0.19%	0.89%
Scenario 2	0.22%	1.36%
Scenario 3	2%	2.34%

Table 6.3 shows the validation results of the proposed energy estimation method against transistor-level simulation of benchmarks. The test program for different test scenarios is simulated at SPICE level as shown in Fig. 6.15. Similarly, proposed energy estimation method is used to estimate energy for all the test scenarios of the test program. Finally both the results are compared to validate the accuracy of the proposed energy estimation method. Average error and maximum error are the two figure of merits used to determine the accuracy of the proposed method. In *scenario 1* for different arithmetic and logical operations with immediate addressing mode, there is an average error of 0.23% in the proposed method compared to transistor-level simulation. Likewise, for register-direct addressing mode, the average error is 0.19%. In both the cases, the average error is negligible and the maximum error is also within the acceptable margin. In *scenario 2*, where an operation is executed repeatedly in different addressing modes, the average error is 0.22% and the maximum error is 1.36%. *Scenario 3*records highest average error of 2% and the maximum error of 2.34%. In the validation procedure, execution stage of $\rho - VEX$ cluster along with other modules are simulated at transistor-level. This provides more accurate energy estimation at the cost of enormous simulation time. The error in the proposed method is due to estimation at high-level C compilation based on transistor-level characterization of functional units. The aim of this work is to estimate energy consumption by simple C compilation which is many times faster compared to transistor-level simulations. Because of the transistor-level characterization of functional units, the average error in energy estimation is within the acceptable limits. In the proposed energy estimation method benchmark programs can be compiled and energy can be estimated in few minutes compared to hours of SPICE simulation based energy estimation.

6.6.2 Proof of Concept

The proposed energy estimation method is implemented for a set of benchmark programs written in C. For all the benchmarks, *.cs.c* file is modified in a way to increment a global counter every time a particular operation is executed. Based on the Eqn. 6.4 and 6.5, energy consumed by the execution unit of the $\rho - VEX$ cluster is estimated for different operating triads. The benchmarks chosen are programs commonly used in most of the mathematical, signal processing, and cryptography applications taken from Mediabench suite. Table 6.4 shows the list of benchmarks with a total number of instructions and traces in each of the benchmark program.

TABLE 6.4: Benchmark Programs

Benchmarks	Total Number of Instructions	Total number of traces
abs	5	1
find_min	41	7
crc8_comp	57	7
crc16_comp	57	7
bphash	60	9
binary_search	88	11
fir_basic	115	16
calc_neigbhour	135	14

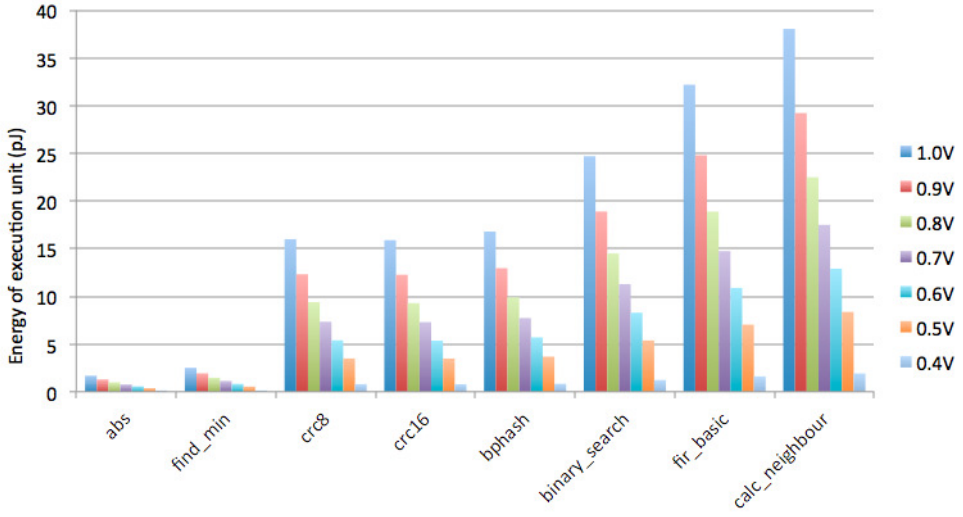


FIGURE 6.16: Energy estimation of execution unit for Benchmarks with different V_{dd}, and no back-biasing at T_{clk} = 0.9ns

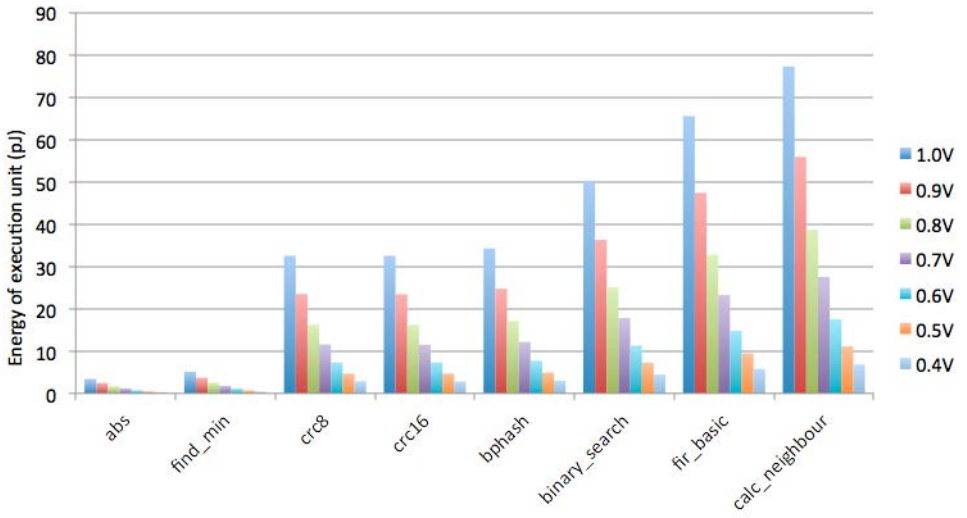


FIGURE 6.17: Energy estimation of execution unit for Benchmarks with different V_{dd}, with back-biasing V_{bb} = 2V, and T_{clk} = 0.9ns

The modified *.cs.c* is executed using a C compiler to estimate the energy consumption for different operation triads. Fig. 6.16 shows the energy estimation of the execution unit of $\rho - VEX$ cluster for benchmark programs under different V_{dd} ranging from 1V to 0.4V without back-biasing. Reducing V_{dd} from 1V to 0.4V reduces the energy consumption of execution unit by 20 × on average for all the benchmarks. But the significant energy gains are limited by the timing errors as shown in energy versus BER plots in the characterization section. This limitation can be overcome by applying body-biasing to limit BER to an acceptable margin at the cost of reduced energy gains. Fig. 6.17 shows the energy estimation of benchmarks with body-biasing V_{bb} = 2V. With body-bias, at V_{dd} =0.4V, energy consumption reduces by 11 × on average for all the benchmarks.

In addition to voltage over-scaling, overclocking is employed by reducing the clock period T_{clk} from 0.9ns to 0.45ns. This improves the energy gains at the cost of increased BER. At reduced clock period, benchmark programs achieve 31 × reduction in energy consumption at 0.4V compared to 1V. With body-biasing of V_{bb} = 2V, timing errors are limited and the energy consumption is reduced by 10 × on average compared to energy consumption at 1V. The proposed framework provides an easy way to understand the behaviour of the execution unit of $\rho - VEX$ cluster for different operating triads. With the proposed framework, the user can decide the optimum trade-off between energy versus accuracy for any given program by choosing most suitable operating triads at runtime.

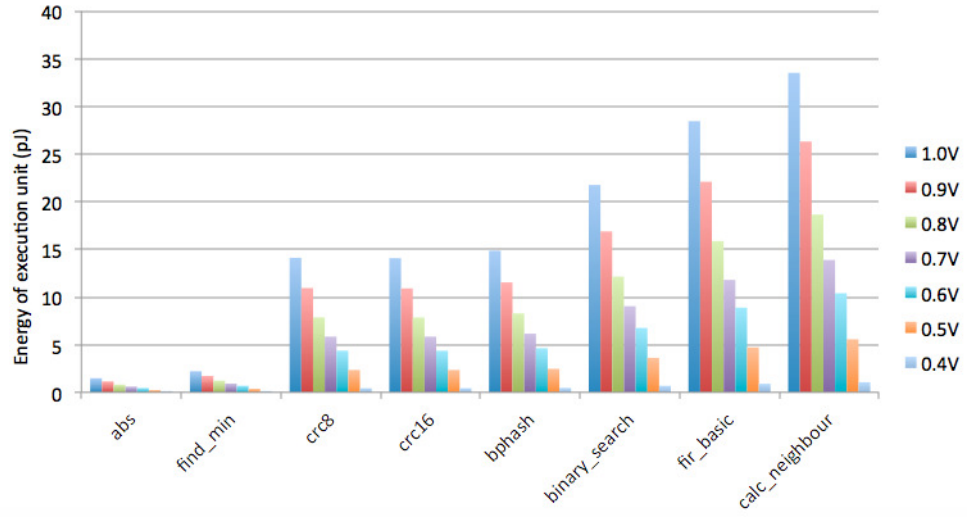


FIGURE 6.18: Energy estimation of execution unit for Benchmarks with different V_{dd}, and no back-biasing at $T_{clk} = 0.45$ns

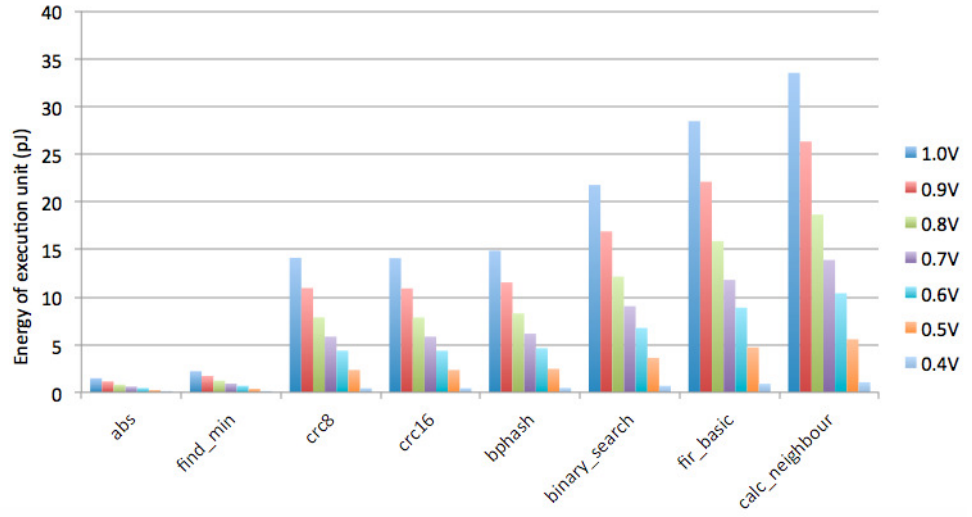


FIGURE 6.19: Energy estimation of execution unit for Benchmarks with different V_{dd}, with back-biasing $V_{bb} = 2$V, and $T_{clk} = 0.45$ns

6.7 Conclusion

In this chapter, an instruction-level energy estimation framework is defined and presented for VLIW platform in the context of approximate computing in near-threshold regime. The energy estimation is done at two stages namely *Characterization* and *Estimation*. In the *Characterization* step, components like ALU, Multiplier in the execution unit of $\rho - VEX$ cluster are subjected to voltage overscaling and frequency overclocking to profile energy consumption and losses in computation accuracy. In the *Estimation*

step, based on the characterization profile, energy consumption of execution unit can be easily estimated for any given program by simple C compilation process using VEX Toolchain. A validation procedure is developed to estimate the accuracy of the proposed energy estimation framework. In this work, instruction-level energy consumption is estimated for 8 benchmark programs. The proposed framework estimation is 98% accurate when compared to full SPICE simulation of the benchmark program. The benefit of using body-biasing technique is highlighted to reduce error rate at near-threshold regime at the cost of reduced energy gains. The proposed energy estimation framework is simple, very fast, and accurate in estimating the energy consumption for different operating conditions. In this work, scope for energy estimation is limited to execution of $\rho - VEX$ cluster, this can be further expanded to other modules like *fetch*, *decode*, and *writeback* in the cluster.

Chapter 7

Conclusion and Perspectives

7.1 Overview

In this thesis, a framework is proposed to handle timing errors and to estimate energy consumption of error resilient applications in the scope of near-threshold computing. In the energy crunching environment, keeping the device functional for the long haul is the major challenge in the present trend of digital design. Various low-power techniques are explored at different levels of abstraction to improve the energy efficiency. Dynamic voltage scaling is a prominent low-power technique used to increase the energy efficiency of the designs by reducing the supply voltage at runtime. In this work, we have explored near-threshold computing to enhance the energy gains by scaling down the supply voltage close to the threshold voltage of the transistor. Reducing the supply voltage makes the design more vulnerable to variability effects and timing errors. We have proposed to use FDSOI, an inherent low-leakage technology to counter the variability effects by effective body-biasing technique. Different error handling techniques like double sampling methods are conventionally used to tackle errors. Most of the double sampling techniques use fixed speculation window to detect and correct the timing errors. In this work, we have proposed dynamic speculation window-based error detection and correction method to handle the timing errors while scaling the supply voltage close to the threshold voltage. We have demonstrated the advantage of dynamic speculation method in Xilinx FPGA platform along with conventional double sampling method. We overclocked the benchmark programs in FPGA beyond the estimated maximum frequency to invoke timing errors in the design. Also, variability effects are introduced in the design

by controlling the RPM of the cooling fan on the FPGA. In these experiments, the proposed dynamic speculation method is able to dynamically adjust the clock frequency to counter the variability impacts with respect to the user-defined tolerable error margin. We achieved 71% safe overclocking margin in the proposed dynamic speculation method with the minimal hardware overhead of 1.9% LUTs and 1.7% FFs.

Apart from error detection and correction, this work also explores the domain of approximate computing. In this emerging style of computing, error in computing is deliberated to achieve high energy efficiency. In our proposed dynamic speculation based error handling method, the end user can decide to tolerate certain margin of errors. Based on the level of accuracy required for different classes of applications, the user can determine the optimum trade-off between accuracy and energy efficiency. At runtime, the user can decide to achieve higher accuracy or energy efficiency by operating at higher supply voltage or at near-threshold voltage respectively. Voltage overscaling can be efficiently applied for error-resilient applications. Error-resilient applications have the natural tendency to tolerate certain acceptable margin of errors. Therefore, we propose a framework to statistically model the arithmetic operators for VOS. The behaviour of arithmetic operators in the near-threshold voltage is different from the super-threshold regime. We have performed an extensive study of these operators at transistor level to characterize the behaviours of the arithmetic operators for VOS. Based on the characterization of the arithmetic operators, a method to statistically model these operators is proposed. These models reflect the behaviour of arithmetic operators in different operating conditions. This makes energy and error estimation of the arithmetic circuits simple and very accurate for different operating conditions. Various error metrics like MSE, Hamming distance to ensure the accuracy of the model. These statistical models can be used at the algorithmic level to simulate the effect of VOS on the design to choose right operating triads to achiever higher energy efficiency with minimal loss in the accuracy.

To further extend the scope of dynamic speculation and VOS, we explored programmable platform like VLIW processors. Execution unit of the $\rho - VEX$ VLIW processor cluster is studied in this thesis work. We proposed a framework to characterize, evaluate, and validate energy estimation of $\rho - VEX$ cluster execution unit for different operating triads. The functional units of the $\rho - VEX$ cluster are characterized

at transistor level to extract accurate energy and error data for different operating triads. Transistor-level simulations are very slow even for the simpler program. Therefore, we proposed an instruction-level energy estimation method by a simple compilation of C program. By this, we can estimate the energy of a program for different operating conditions very quickly from accurate transistor-level characterization. We also proposed an alternate validation procedure to ensure the accuracy of the proposed energy estimation method by comparing with full SPICE simulation of the benchmarks. In this framework, we achieved very accurate energy estimation with an average error of less than 2%.

7.2 Unified Approach

An important challenge in digital circuit design is to achieve high energy efficiency at minimal cost in terms of silicon footprint and acceptable compromise in accuracy or performance of the circuit. Due to the quadratic dependence of energy consumption on the supply voltage, reducing V_{dd} yields higher energy efficiency. A simple observation of different CMOS technology in the past decade shows the trend of voltage scaling. In near-threshold region, delay and energy are roughly linear with V_{dd}. Operating transistors in the near-threshold regime offer better control over energy-error trade-off. FDSOI technology is used to improve the benefits of near-threshold computing. But the impact of variability and timing errors outweighs the benefits of near-threshold computing. In this thesis, we addressed this problem by three contributions.

Fig. 7.1 shows an overall perspective of this thesis. Fig. 7.1 shows a VLIW cluster with different pipeline stages like instruction fetch, decode, execution stage represented by functional units (FU), register files and memory blocks. In a typical execution flow, instructions are fetched and decoded by instruction fetch and decode stage. Different operations in each of the instructions are executed by different FUs in parallel. The outputs are recorded in the register files and memory. To achieve high energy efficiency, voltage scaling can be applied to FUs. Reduction in V_{dd} results in timing errors that can be handled using various methods explained in Chapter 3. In Chapter 3, limitations of the existing methods are discussed in detail. Also, the need for a unified approach to handle errors is insisted to achieve best energy-error trade-off.

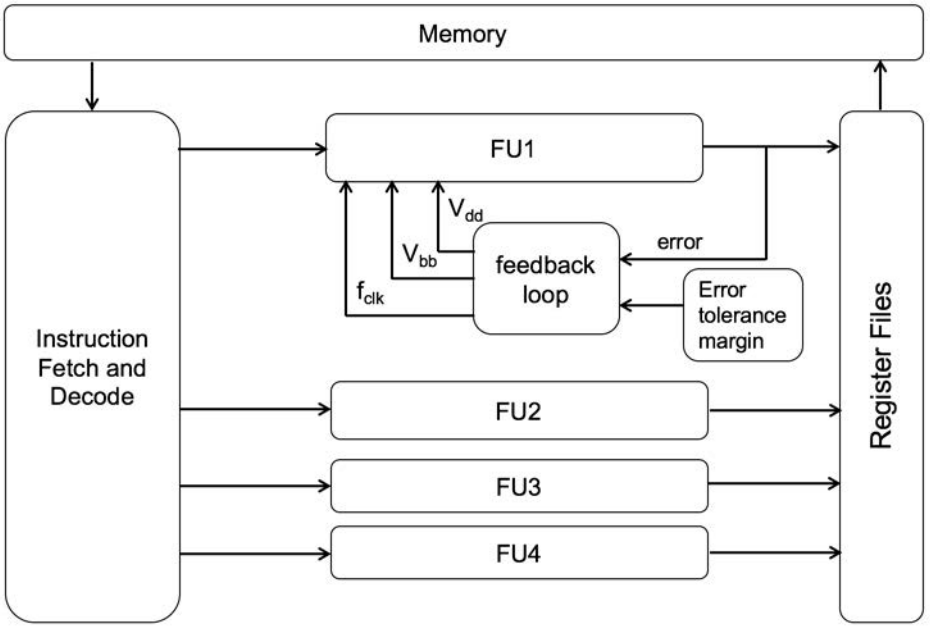


FIGURE 7.1: Overall perspective of the thesis

Fig. 7.1 shows the unified approach to handle timing errors in the context of near-threshold computing and approximate computing for error-resilient applications. The FUs are monitored using the proposed dynamic speculation based adaptive feedback loop. The highlight of the proposed dynamic speculation method is adaptive error tolerance based on user-defined error margin. With runtime slack measurement, the proposed method can predict the possiblities of timing errors. If required, preventive mechanism can be invoked by ajusting the operating triads. Based on the user defined error tolerance margin, errors can be detected and corrected or ignored at runtime, for different application needs, due to the presence of additional slack register. The adaptive feedback loop does overclocking or voltage overscaling when there is a scope for performance improvement or higher energy efficiency. The proposed method gives flexibility for the application developer to extract higher energy efficiency based on the nature of the application and user requirement.

In this work, the behaviour of different arithmetic operators is studied under various operating triads. This provides detailed insight of error generation and propagation in different arithmetic operators under the influence of voltage overscaling. Based on this study, dynamic speculation is used as shown in Fig. 7.1 to reduce energy consumption in arithmetic operators at runtime. In order to easily understand different energy-error trade-off points, energy estimation framework is proposed for VLIW processors. The

proposed energy estimation framework simulates any given benchmark program using C compiler and provides energy estimation for different operating triads. This helps the application developer to understand the benefits and costs to achieve best possible energy-error trade-off.

7.3 Future Work

7.3.1 Improvements in Dynamic Speculation Window Method

Some of the high end, off-the-shelf commercial FPGAs support voltage scaling. Dynamic speculation window-based error handling method can be experimented with voltage overscaling in FPGA platform to further enhance the feedback loop. By this, it is more realistic to demonstrate the advantages of dynamic speculation to counter the variability effects by adjusting both voltage and frequency at runtime.

7.3.2 Improving VOS of Approximate Operators

In this work, adders and multipliers are studied at near-threshold regime to create the statistical model to mimic the behaviour of these operators. A proof of concept is shown in this work by modelling different adder configurations. Similar to adders, some study is done for multipliers by utilizing adder models in the final stage of parallel prefix multiplier. This can be further extended by studying the tree structure used in these multipliers to generate partial products. Likewise, other arithmetic operators can also be modelled to create a library of VOS operators. By this, it will be very easy to simulate the trade-off given by approximate computing for any complex design. In this work, a near-threshold logic library is designed by sizing the transistors to achieve higher energy gains. The statistical model can be linked to the near-threshold logic library to improve the accuracy of the model.

7.3.3 Improving Energy Estimation of $\rho - VEX$ Cluster

In this work, only the execution unit of the $\rho - VEX$ cluster is characterized to estimate the energy. The proposed framework can be further improved by modelling the entire

cluster using SPICE simulations. Already some of the other modules like decoder, etc. are simulated at SPICE level. Once the whole cluster is modelled, there is a lot of scope to dynamically change the processor configuration by altering the issue width and other parameters. Since the functional units are same, the existing framework can estimate the energy for different configurations at runtime.

Publications

[1] RahulKumar Bhudhwani, Rengarajan Ragavan, and Olivier Sentieys. Taking advantage of correlation in stochastic computing. In *IEEE Proc. of the Intl. Symp. on Circuits and Systems (ISCAS)*, pages 1–4, 2017.

[2] Rengarajan Ragavan, Benjamin Barrois, Cedric Killian, and Olivier Sentieys. Pushing the limits of voltage over-scaling for error-resilient applications. In *IEEE/ACM Proc. Conf. of the Design Automation & Test in Europe (DATE)*, pages 476–481, 2017.

[3] Rengarajan Ragavan, Cedric Killian, and Olivier Sentieys. Low complexity on-chip distributed DC-DC converter for low power WSN nodes. In *IEEE New Circuits and Systems Conference (NEWCAS)*, pages 1–4, 2015.

[4] Rengarajan Ragavan, Cedric Killian, and Olivier Sentieys. Adaptive overclocking and error correction based on dynamic speculation window. In *IEEE Computer Society Annual Symp. on VLSI (ISVLSI)*, pages 325–330, 2016.

Appendix A

Characterization of Multipliers

Like adders, multipliers also play an important role in most common functional units. In this section, two multiplier configurations are characterized at circuit level based on voltage overscaling. Multiplier design is sectioned into Partial Product Generation (PPG), Partial Product Accumulation (PPA), and final stage adder. Schematics of two multiplier configurations used in this work are shown in Fig. A.1, and Fig. A.2.

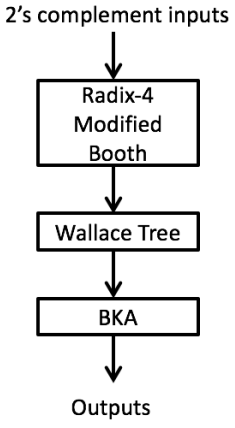


FIGURE A.1: Signed multiplier (Radix-4 modified Booth, Wallace Tree, and BKA)

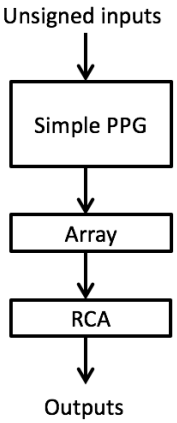


FIGURE A.2: Unsigned multiplier (Simple PPG, Array, and RCA)

Fig. A.1 shows signed multiplier configuration with Radix-4 modified Booth PPG, Wallace Tree PPA and BKA as final stage adder. Similarly, unsigned multiplier configuration with Simple PPG, array based PPA, and RCA as final stage adder is shown in Fig. A.2. Fig. A.3 shows BER vs. energy plot of 16-bit signed multiplier. The left part of the plot shows voltage scaling without any errors. By scaling the V_{dd} to 0.6V with

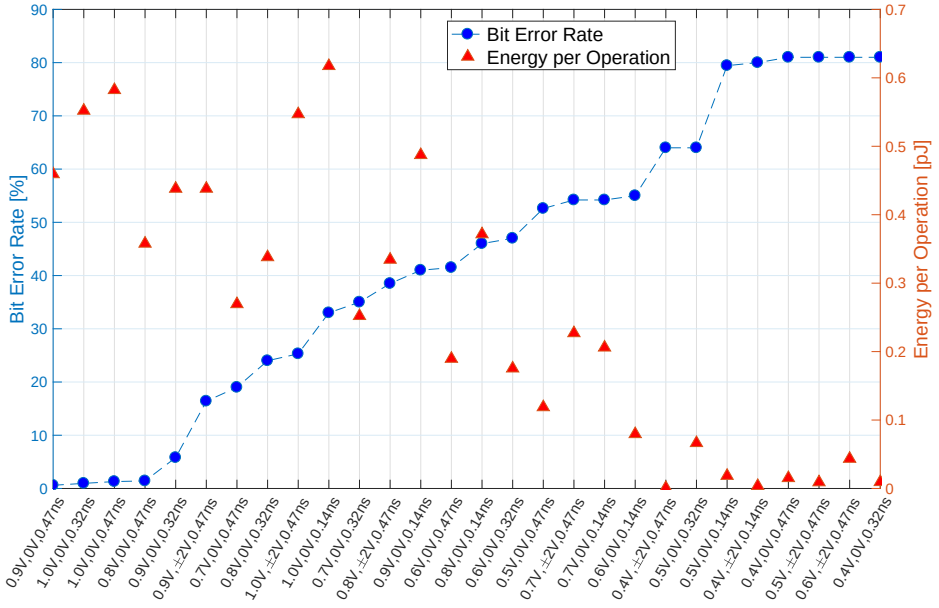


FIGURE A.3: Distribution of BER in output bits of 16-bit Signed Multiplier

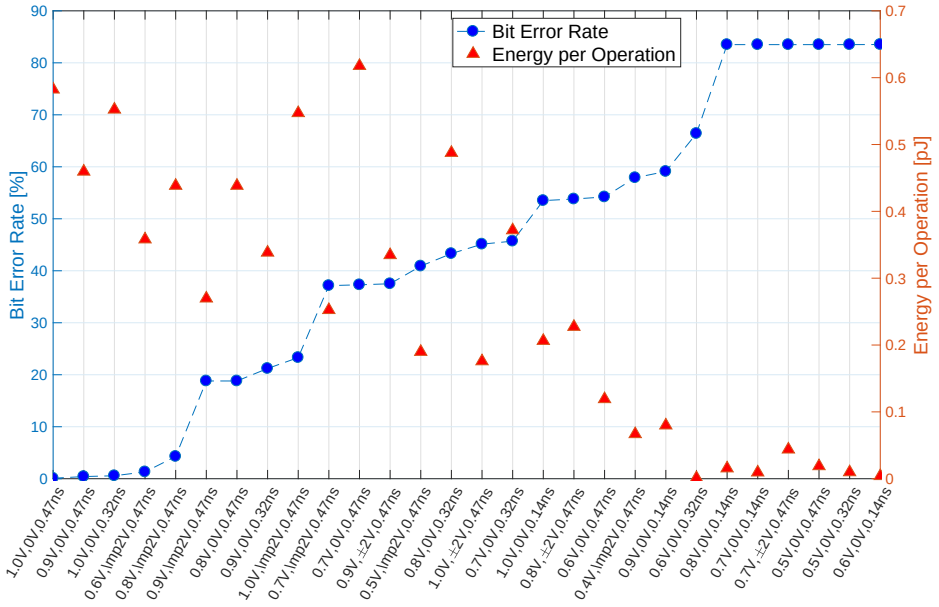


FIGURE A.4: Distribution of BER in output bits of 16-bit Unsigned Multiplier

±2V back-biasing 72% energy efficiency can be achieved. By further reducing the V_{dd} to 0.5V with ±2V, 81% energy efficiency can be achieved with the acceptable error rate of 21%. The right side of the plot shows reduction in energy by reducing the V_{dd} with very high BER. Similar plot of 16-bit unsigned multiplier is shown in Fig. A.4. In this multiplier, 73% energy efficiency can be achieved with BER of 1% at V_{dd} = 0.6V and

$V_{bb} = \pm 2$V. The efficiency can be further improved to 82% with BER of 20% at $V_{dd} = 0.5$V and $V_{bb} = \pm 2$V.

As shown in Chapter 5, like adders, multipliers can also be statistically modelled for VOS. These parallel prefix multipliers use Brent-Kung and Ripple Carry adder in their final stage. This gives the advantage of using same BKA and RCA VOS models in this stage. The partial product generation and partial product accumulation stages have to be modelled to obtain the full model of the multipliers. This is one of the future work of this thesis.

Bibliography

[1] Tadahiro Kuroda and Kojiro Suzuki. Variable supply-voltage scheme for low-power high-speed CMOS digital design. *IEEE Journal of Solid-State Circuits*, 33(3):454–462, 1998.

[2] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proc. of the 2010 Intl. Conf. on Power Aware Computing and Systems*, pages 1–8, 2010.

[3] Edward Stott, Zhenyu Guan, Joshua M Levine, Justin SJ Wong, and Peter YK Cheung. Variation and reliability in FPGAs. *IEEE Design & Test*, 30(6):50–59, 2013.

[4] Michael Nicolaidis. Double-sampling design paradigm—a compendium of architectures. *IEEE Trans. on Device and Materials Reliability*, 15(1):10–23, 2015.

[5] Edward Stott, Joshua M Levine, Peter YK Cheung, and Nachiket Kapre. Timing fault detection in fpga-based circuits. In *IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 96–99. IEEE, 2014.

[6] Alexander Brant, Ahmed Abdelhadi, Douglas HH Sim, Shao Lin Tang, Michael Xi Yue, and Guy GF Lemieux. Safe overclocking of tightly coupled CGRAs and processor arrays using razor. In *IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 37–44. IEEE, 2013.

[7] Joshua M Levine, Edward Stott, George Constantinides, and Peter YK Cheung. Online measurement of timing in circuits: For health monitoring and dynamic voltage & frequency scaling. In *IEEE 20th Annual International Symposium on*

Field-Programmable Custom Computing Machines (FCCM), pages 109–116. IEEE, 2012.

[8] Joseph A Fisher, Paolo Faraboschi, and Cliff Young. *Embedded computing: a VLIW approach to architecture, compilers and tools*. Elsevier, 2005.

[9] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.

[10] ITRS. International Technology Roadmap for Semiconductors 2007 ed. Executive Summary. Technical report, ITRS, 2011.

[11] Nor Zaidi Haron and Said Hamdioui. Why is CMOS scaling coming to an end? In *International Design and Test Workshop*, pages 98–103, 2008.

[12] ITRS. International Technology Roadmap for Semiconductors 2011 ed. Design. Technical report, ITRS, 2011.

[13] Martin Wirnshofer. Sources of variation. In *Variation Aware Adaptive Voltage Scaling For Digital CMOS Circuits*, Springer series in advanced microelectronics , Vol 41, chapter 2. Springer, 2013. ISBN 9789400761964.

[14] Jui-Ming Chang and Massoud Pedram. Low power design. In *Power optimization and synthesis at behavioral and system levels using formal methods*, chapter 1. Springer Science & Business Media, 2012.

[15] Ali Dasdan and Ivan Hom. Handling inverted temperature dependence in static timing analysis. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 11(2):306–324, 2006.

[16] D Suleiman, M Ibrahim, and I Hamarash. Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction. In *International Conference on Electrical and Electronics Engineering*, 2005.

[17] Intel. Enhanced Intel® speedstep® technology for the intel® pentium® M processor. Technical report, Intel, 2004.

[18] AMD. AMD powernow technology. Technical report, AMD, 2000.

[19] Ivan Miro-Panades, Edith Beigné, Yvain Thonnart, Laurent Alacoque, Pascal Vivet, Suzanne Lesecq, Diego Puschini, Anca Molnos, Farhat Thabet, and Benoit

Tain. A fine-grain variation-aware dynamic *vdd*-hopping AVFS architecture on a 32 nm GALS MPSoC. *IEEE Journal of Solid-State Circuits*, 49(7):1475–1486, 2014.

[20] Robin Wilson, Edith Beigne, Philippe Flatresse, Alexandre Valentian, Fady Abouzeid, Thomas Benoist, Christian Bernard, Sebastien Bernard, Olivier Billoint, and Sylvain Clerc. A 460MHz at 397mV, 2.6 GHz at 1.3 V, 32b VLIW DSP, embedding fmax tracking. In *IEEE Intl. Solid-State Circuits Conf. Digest of Technical Papers (ISSCC)*, pages 452–453. IEEE, 2014.

[21] Davide Rossi, Antonio Pullini, Igor Loi, Michael Gautschi, Frank Kagan Gurkaynak, Adam Teman, Jeremy Constantin, Andreas Burg, Ivan Miro-Panades, and Edith Beignè. 193 MOPS/mW@ 162 MOPS, 0.32 V to 1.15 V voltage range multi-core accelerator for energy efficient parallel and sequential digital processing. In *IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS XIX)*, pages 1–3. Ieee, 2016.

[22] Ronald G Dreslinski, Michael Wieckowski, David Blaauw, Dennis Sylvester, and Trevor Mudge. Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits. *Proceedings of the IEEE*, 98(2):253–266, 2010.

[23] Benton H. Calhoun and David Brooks. Can subthreshold and near-threshold circuits go mainstream? *IEEE Micro*, 30(4):80–85, 2010.

[24] Ulya R Karpuzcu, Krishna B Kolluru, Nam Sung Kim, and Josep Torrellas. VARIUS-NTV: A microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–11. IEEE, 2012.

[25] David Blaauw and Bo Zhai. Energy efficient design for subthreshold supply voltage operation. In *IEEE Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, pages 4–pp, 2006.

[26] K Cheng, A Khakifirooz, P Kulkarni, S Ponoth, J Kuss, D Shahrjerdi, LF Edge, A Kimball, S Kanakasabapathy, and K Xiu. Extremely thin SOI (ETSOI) CMOS with record low variability for low power system-on-chip applications. In *IEEE International Electron Devices Meeting (IEDM)*, pages 1–4, 2009.

[27] J-P Noel, O Thomas, C Fenouillet-Beranger, M-A Jaud, P Scheiblin, and A Amara. A simple and efficient concept for setting up multi-VT devices in thin BOX fully-depleted SOI technology. In *Proceedings of the European Solid State Device Research Conference (ESSDERC)*, pages 137–140, 2009.

[28] Edith Beigné, Alexandre Valentian, Bastien Giraud, Olivier Thomas, Thomas Benoist, Yvain Thonnart, Serge Bernard, Guillaume Moritz, Olivier Billoint, and Y Maneglia. Ultra-wide voltage range designs in fully-depleted silicon-on-insulator FETs. In *IEEE/ACM Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 613–618. EDA Consortium, 2013.

[29] Bertrand Pelloux-Prayer, Alexandre Valentian, Bastien Giraud, Yvain Thonnart, Jean-Philippe Noel, Philippe Flatresse, and Edith Beigne. Fine grain multi-VT co-integration methodology in UTBB FD-SOI technology. In *IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 168–173, 2013.

[30] Yeter Akgul, Diego Puschini, Suzanne Lesecq, Edith Beigné, Ivan Miro-Panades, Pascal Benoit, and Lionel Torres. Power management through DVFS and dynamic body biasing in FD-SOI circuits. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014.

[31] Linley Gwennap. FD-SOI offers alternative to FINFET. Technical report, The Linley Group, 2016.

[32] Ramiro Taco, Itamar Levi, Alex Fish, and Marco Lanuzza. Exploring back biasing opportunities in 28nm UTBB FD-SOI technology for subthreshold digital design. In *IEEE 28th Convention of Electrical & Electronics Engineers in Israel (IEEEI)*, pages 1–4, 2014.

[33] Abbas Rahimi, Luca Benini, and Rajesh K Gupta. Variability mitigation in nanometer CMOS integrated systems: A survey of techniques from circuits to software. *Proceedings of the IEEE*, 104(7):1410–1448, 2016.

[34] Benton H Calhoun and Anantha P Chandrakasan. Standby power reduction using dynamic voltage scaling and canary flip-flop structures. *IEEE Journal of Solid-State Circuits*, 39(9):1504–1511, 2004.

[35] Dan Ernst, Shidhartha Das, Seokwoo Lee, David Blaauw, Todd Austin, Trevor Mudge, Nam Sung Kim, and Krisztián Flautner. Razor: circuit-level correction of timing errors for low-power operation. *IEEE Micro*, 2004.

[36] Yuji Kunitake, Toshinori Sato, Hiroto Yasuura, and Takanori Hayashida. Possibilities to miss predicting timing errors in canary flip-flops. In *IEEE 54th Intl. Midwest Symp. on Circuits and Systems (MWSCAS)*, pages 1–4, 2011.

[37] Shidhartha Das, Carlos Tokunaga, Sanjay Pant, Wei-Hsiang Ma, Sudherssen Kalaiselvan, Kevin Lai, David M Bull, and David T Blaauw. RazorII: In situ error detection and correction for PVT and SER tolerance. *IEEE Journal of Solid-State Circuits*, 44(1):32–48, 2009.

[38] Matthew Fojtik, David Fick, Yejoong Kim, Nathaniel Pinckney, David Money Harris, David Blaauw, and Dennis Sylvester. Bubble razor: Eliminating timing margins in an ARM Cortex-M3 processor in 45 nm cmos using architecturally independent error detection and correction. *IEEE Journal of Solid-State Circuits*, 48(1):66–81, 2013.

[39] Michael Nicolaidis. GRAAL: a new fault tolerant design paradigm for mitigating the flaws of deep nanometric technologies. In *IEEE International Test Conference (ITC)*, pages 1–10. IEEE, 2007.

[40] Robert Baumann. Soft errors in advanced computer systems. *IEEE Design & Test of Computers*, 22(3):258–266, 2005.

[41] Thomas Schweizer, Anja Küster, Sven Eisenhardt, Tommy Kuhn, and Wolfgang Rosenstiel. Using run-time reconfiguration to implement fault-tolerant coarse grained reconfigurable architectures. In *IEEE 26th International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 320–327, 2012.

[42] Thomas Schweizer, Philipp Schlicker, Sven Eisenhardt, Tommy Kuhn, and Wolfgang Rosenstiel. Low-cost TMR for fault-tolerance on coarse-grained reconfigurable architectures. In *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 135–140, 2011.

[43] Thomas Schweizer, Wolfgang Rosenstiel, Luigi Vaz Ferreira, and Marcus Ritt. Timing error handling on CGRAs. In *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 1–6, 2013.

[44] Subhasish Mitra, Nirmal R Saxena, and Edward J McCluskey. A design diversity metric and analysis of redundant systems. *IEEE Transactions on Computers*, 51 (5):498–510, 2002.

[45] Pedro Reviriego, Chris J Bleakley, and Juan Antonio Maestro. Diverse double modular redundancy: A new direction for soft-error detection and correction. *IEEE Design & Test*, 30(2):87–95, 2013.

[46] Pedro Reviriego, Chris J Bleakley, and Juan Antonio Maestro. Structural DMR: A technique for implementation of soft-error-tolerant fir filters. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 58(8):512–516, 2011.

[47] Jaeyoon Kim and Sandip Tiwari. Inexact computing for ultra low-power nanometer digital circuit design. In *IEEE/ACM International Symposium on Nanoscale Architectures*, pages 24–31, 2011.

[48] Michael Schaffner, Frank K Gurkaynak, Aljosa Smolic, Hubert Kaeslin, and Luca Benini. An approximate computing technique for reducing the complexity of a direct-solver for sparse linear systems in real-time video processing. In *ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2014.

[49] Jie Han and Michael Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *IEEE European Test Symposium (ETS)*, pages 1–6, 2013.

[50] G Tziantzioulis and AM Gok. Lazy pipelines: Enhancing quality in approximate computing. In *Proc. Conf. of the Design Automation & Test in Europe*, pages 1381–1386, 2016.

[51] Amir Yazdanbakhsh, Divya Mahajan, and Bradley Thwaites. Axilog: Language support for approximate hardware design. In *Proc. Conf. of the Design Automation & Test in Europe (DATE)*, pages 812–817, 2015.

[52] Michael Nicolaidis. Time redundancy based soft-error tolerance to rescue nanometer technologies. In *IEEE Proc. of the VLSI Test Symp.*, pages 86–94, 1999.

[53] Joshua M Levine, Edward Stott, George Constantinides, and Peter YK Cheung. SMI: Slack measurement insertion for online timing monitoring in FPGAs. In *Proc. Int. Conf. on Field Programmable Logic and Applications (FPL)*, pages 1–4, 2013.

[54] Justin SJ Wong, Pete Sedcole, and Peter YK Cheung. Self-measurement of combinatorial circuit delays in FPGAs. *ACM Trans. on Reconfigurable Technology and Systems (TRETS)*, 2(2):10, 2009.

[55] Pete Sedcole, Justin S Wong, and Peter YK Cheung. Characterisation of FPGA clock variability. In *IEEE Computer Society Annual Symp. on VLSI (ISVLSI)*, pages 322–328, 2008.

[56] Xilinx. *VC707 Evaluation Board for the Virtex-7 FPGA User Guide*, v1.6 edition, April 2015.

[57] A Benhassain, F Cacho, V Huard, S Mhira, L Anghel, C Parthasarathy, A Jain, and A Sivadasan. Robustness of timing in-situ monitors for AVS management. In *IEEE International Reliability Physics Symposium (IRPS)*, pages CR–4, 2016.

[58] A Benhassain, S Mhira, F Cacho, V Huard, and L Anghel. In-situ slack monitors: taking up the challenge of on-die monitoring of variability and reliability. In *IEEE International Verification and Security Workshop (IVSW)*, pages 1–5. IEEE, 2016.

[59] LB Soares and Sergio Bampi. Near-threshold computing for very wide frequency scaling: Approximate adders to rescue performance. In *IEEE New Circuits and Systems Conference (NEWCAS)*, pages 1–4, 2015.

[60] Younghoon Kim, Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. Designing approximate circuits using clock overgating. In *Proc. of the 53rd Annual Design Automation Conference*, page 15, 2016.

[61] Ismail Akturk, Nam Sung Kim, and Ulya R Karpuzcu. Decoupled control and data processing for approximate near-threshold voltage computing. *IEEE Micro*, 35(4): 70–78, 2015.

[62] Chaofan Li, Wei Luo, Sachin S Sapatnekar, and Jiang Hu. Joint precision optimization and high level synthesis for approximate computing. In *Proc. of the 52nd Annual Design Automation Conference*, page 104, 2015.

[63] Avinash Lingamneni and Christian Enz. Energy parsimonious circuit design through probabilistic pruning. In *Proc. Conf. of the Design Automation & Test in Europe (DATE)*, pages 1–6, 2011.

[64] Jeremy Schlachter, Vincent Camus, and Christian Enz. Near/sub-threshold circuits and approximate computing: The perfect combination for ultra-low-power systems. In *IEEE Computer Society Annual Symp. on VLSI*, pages 476–480, 2015.

[65] Avinash Lingamneni, Christian Enz, Krishna Palem, and Christian Piguet. Designing energy-efficient arithmetic operators using inexact computing. *Journal of Low Power Electronics*, 9(1):141–153, 2013.

[66] Weste Neil HE. *CMOS VLSI Design: A Circuits And Systems Perspective, 4/E.* Pearson Education India, 2006.

[67] Andrew B Kahng and Seokhyeong Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Proc. of the Annual Design Automation Conference*, pages 820–825, 2012.

[68] Rengarajan Ragavan, Cedric Killian, and Olivier Sentieys. Adaptive overclocking and error correction based on dynamic speculation window. In *IEEE Computer Society Annual Symp. on VLSI (ISVLSI)*, pages 325–330, 2016.

[69] Stephan Wong, Thijs Van As, and Geoffrey Brown. ρ-VEX: A reconfigurable and extensible softcore VLIW processor. In *International Conference on Electrical and Computer Engineering (ICECE)*, pages 369–372, 2008.

[70] Anthony Brandon and Stephan Wong. Support for dynamic issue width in VLIW processors using generic binaries. In *Proc. Conf. of the Design Automation & Test in Europe (DATE)*, pages 827–832, 2013.

[71] Alireza Dehghani, Aubrey Dunne, David Moloney, and Oscar Deniz. Application processor description. Technical report, Movidius, 2016.

[72] Thijs Van As. ρ-VEX: A reconfigurable and extensible softcore VLIW processor. Technical report, Delft University of Technology, 2008.

[73] Hewlet-Packard Laboratories. VEX toolchain. Technical report, HP, 2011.

[74] Chaitali Chakrabarti and Dinesh Gaitonde. Instruction level power model of microcontrollers. In *IEEE Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, volume 1, pages 76–79, 1999.

[75] Vivek Tiwari, Sharad Malik, Andrew Wolfe, and Mike Tien-Chien Lee. Instruction level power analysis and optimization of software. In *Technologies for Wireless Computing*, pages 139–154. Springer, 1996.

[76] Huzefa Mehta, Robert Michael Owens, and Mary Jane Irwin. Instruction level power profiling. In *IEEE Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 6, pages 3326–3329, 1996.

[77] Vittorio Zaccaria, Mariagiovanna Sami, Donatella Sciuto, and Cristina Silvano. *Power estimation and optimization methodologies for VLIW-based embedded systems*. Springer Science & Business Media, 2007. ISBN 1402073771.

[78] Luca Benini, Davide Bruni, Mauro Chinosi, Christina Silvano, Vittorio Zaccaria, and Roberto Zafalon. A power modeling and estimation framework for vliw-based embedded systems. In *Proc. Intl. Workshop on Power And Timing Modeling, Optimization and Simulation PATMOS*, volume 1, pages 2–3, 2001.

[79] David M Brooks, Pradip Bose, Stanley E Schuster, Hans Jacobson, Prabhakar N Kudva, Alper Buyuktosunoglu, J Wellman, Victor Zyuban, Manish Gupta, and Peter W Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[80] Erkan Diken, Rosilde Corvino, and Lech Jóźwiak. Rapid and accurate energy estimation of vector processing in vliw asips. In *Mediterranean Conference on Embedded Computing (MECO)*, pages 33–37, 2013.

[81] Sourav Roy, Rajat Bhatia, and Ashish Mathur. An accurate energy estimation framework for VLIW processor cores. In *International Conference on Computer Design (ICCD)*, pages 464–469. IEEE, 2007.

[82] Andrea Bona, Mariagiovanna Sami, Donatella Sciuto, Vittorio Zaccaria, Cristina Silvano, and Roberto Zafalon. Energy estimation and optimization of embedded VLIW processors based on instruction clustering. In *Proceedings of the annual Design Automation Conference*, pages 886–891. ACM, 2002.

[83] Michael Meixner and Tobias G Noll. Limits of gate-level power estimation considering real delay effects and glitches. In *International Symposium on System-on-Chip (SoC)*, pages 1–7. IEEE, 2014.