Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων
Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων

Arithmetic-Aware Approximation Techniques for Energy-Efficient Inexact Circuits

Διπλωματικη Εργασια

του

ΚΩΝΣΤΑΝΤΙΝΟΥ ΑΣΗΜΑΚΟΠΟΥΛΟΥ

Επιβλέπων: Πεκμεστζή Κιαμάλ
Καθηγητής Ε.Μ.Π.

Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων
Αθήνα, Νοέμβριος 2018

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Arithmetic-Aware Approximation Techniques for Energy-Efficient Inexact Circuits

Διπλωματικη Εργασια

του

ΚΩΝΣΤΑΝΤΙΝΟΥ ΑΣΗΜΑΚΟΠΟΥΛΟΥ

Επιβλέπων: Πεκμεστζή Κιαμάλ
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 1η Νοεμβρίου 2018.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
....	....	....
Πεκμεστζή Κιαμάλ	Δημήτριος Σούντρης	Γεώργιος Γκούμας
Καθηγητής Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π

Αθήνα, Νοέμβριος 2018

(Υπογραφή)

....

Κωνσταντινος Ασημακοπουλος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Ευχαριστίες

Αρχικά Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή Πεκμεστζή Κιαμάλ για την ευκαιρία που μου έδωσε να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα.

Επίσης ευχαριστώ ιδιαίτερα τον υποψήφιο διδάκτορα Λέοντα Βασίλειο για την καθοδήγηση, τις χρήσιμες συμβουλές του και την τεχνική βοήθεια που μου παρείχε όποτε τη χρειάστηκα.

Τέλος θα ήθελα να ευχαριστήσω την οικογένειά μου για την στήριξη της καθόλη τη διάρκεια της φοίτησης μου στο Εθνικό Μετσόβιο Πολυτεχνείο.

Περίληψη

Τα τελευταία χρόνια με την ραγδαία ανάπτυξη των ενσωματωμένων συστημάτων έχει δημιουργηθεί η ανάγκη για υψηλή απόδοση σε συνδυασμό με χαμηλή κατανάλωση ενέργειας. Έτσι έκανε την εμφάνισή του ο προσεγγιστικός υπολογισμός (Approximate Computing) ο οποίος αξιοποιεί το γεγονός ότι κάποιες εφαρμογές έχουν ανοχή στο σφάλμα. Τέτοιες εφαρμογές μπορεί να περιλαμβάνουν media processing, machine learning, data mining και statistics..

Στην παρούσα διπλωματική εργασία πραγματοποιείται αναζήτηση νέων τεχνικών προσεγγιστικού υπολογισμού με σκοπό τη μείωση της κατανάλωσης ενέργειας και παράλληλα την υψηλή απόδοση. Επομένως, αναπτύχθηκαν 5 τεχνικές που πραγματοποιούν προσεγγίσεις στον τρόπο με τον οποίο εκτελείται ο πολλαπλασιασμός. Αφού υλοποιήθηκαν οι τεχνικές στη γλώσσα Verilog έγιναν προσομοιώσεις με σκοπό τον υπολογισμό της κατανάλωσης ενέργειας και του σφάλαματος κάθε τεχνικής. Οι μετρήσεις αυτές πραγματοποιήθηκαν με την βοήθεια των εργαλείων, Synopsys Design Compiler, Mentor Graphics ModelSim και Matlab. Τελικώς, για την σύγκριση των τεχνικών δημιουργήθηκαν διαγράμματα Pareto δύο ειδών. Το πρώτο είχε ως παραμέτρους την κατανάλωση ενέργειας και στο σφάλμα ενώ το δεύτερο τον απαιτούμενο χώρο και το σφάλμα.

Λέξεις Κλειδιά

Προσεγγιστικός υπολογισμός, αριθμητικά κυκλώματα, σχεδίαση ASIC, ανοχή στα σφάλματα, κατανάλωση ενέργειας

Abstract

In recent years, the embedded and mobile nature of modern computing systems has led to an increased need for high performance and energy efficiency. As a result, energy dissipation has become a first class concern in the design of integrated circuits. Towards this direction, approximate (or inexact) computing appears as an emerging and promising solution for energy-efficient systems design, exploiting the inherent error/noise resilience of various applications involving media processing, machine learning, data mining and statistics.

In this diploma thesis an exploration of new techniques of approximate computing is performed. Therefore, 5 techniques were developed with the purpose of energy dissipation and high accuracy. After the implementation of the techniques in Verilog code, several simulations were ran in order to compute the energy consumption and error. The tools used to perform these simulations were Synopsys Design Compiler, Mentor Graphics ModelSim and Matlab. Lastly, the comparison between the techniques was possible with the help of Pareto diagrams. Two types were presented. The first type is an Energy-Error diagram and the second an Area-Error diagram.

Keywords

Approximate Computing, Arithmetic Circuits, ASIC Design, Error Tolerance, Energy Efficiency

Contents

List of Figures

List of Tables

Εκτεταμένη Περίληψη

Εισαγωγή

Τα τελευταία χρόνια με την ραγδαία ανάπτυξη των ενσωματωμένων συστημάτων έχει δημιουργηθεί η ανάγκη για υψηλή απόδοση σε συνδυασμό με χαμηλή κατανάλωση ενέργειας. Ακολουθώντας αυτή την κατεύθυνση ο προσεγγιστικός υπολογισμός (Approximate Computing) είναι μία πολλά υποσχόμενη λύση. Η ιδέα αυτή βασίζεται στο γεγονός ότι πολύ συχνά τα ακριβή αποτελέσματα δεν είναι αναγκαία σε όλες τις εφαρμογές. Τέτοιες εφαρμογές μπορεί να περιλαμβάνουν media processing, machine learning, data mining and statistics. Αυτή η ανοχή όσον αφορά την ακρίβεια οφείλεται στους παρακάτω παράγοντες:

1. Περιορισμένη ανθρώπινη αντίληψη
2. Η πολυπλοκότητα που απαιτείται για την παραγωγή ακριβών αποτελεσμάτων
3. Τα περιττά δεδόμενα ή/και τα δεδομένα με θόρυβο στην είσοδο
4. Η ικανότητα της εφαρμογής να απορροφά τα σφάλματα
5. Οι πιθανοτικοί/στατιστικοί υπολογισμοί
6. Η δεκτικότητα του χρήστη σε χαμηλής ποιόητας αποτελέσματα

Αξιοποιώντας τα παραπάνω σε αυτή τη διπλωματική εργασία αναπτύχθηκαν 5 τεχνικές για τον υπολογισμό του αποτελέσματος της πράξης του πολλαπλασιαμού, η κάθε μία με διαφορετικές προσεγγίσεις και με σκοπό τη χαμηλότερη κατανάλωση ενέργειας.

Σύντομο Θεωρητικό Υπόβαθρο

Κωδικοποίση Modified Booth

Έστω δύο αριθμοί A, B σε συμπλήρωμα ως προς δύο. Το B δίνεται απο την σχέση: $B = \sum_{j=0}^{n/2-1} b_j^{MB} 4^j$. Στον επόμενο πίνακα 1 φαίνεται πώς διαμορφώνονται τα σήματα κατά τον τροποποιημένο αλγόριθμο του Booth (radix-4 encoding).

Δυαδικά Ψηφία			**Ψηφίο Modified Booth**	**Κωδικοποιημένο Ψηφίο b_j^{MB}**		
b_{2j+1}	b_{2j}	b_{2j-1}		**sign = s_j**	**x1 = one_j**	**x2 = two_j**
0	0	0	**0**	0	0	0
0	0	1	**+1**	0	1	0
0	1	0	**+1**	0	1	0
0	1	1	**+2**	0	0	1
1	0	0	**−2**	1	0	1
1	0	1	**−1**	1	1	0
1	1	0	**−1**	1	1	0
1	1	1	**0**	1 ή 0	0	0

Πίνακας 1: Πίνακας κωδικοποίησης Modified Booth

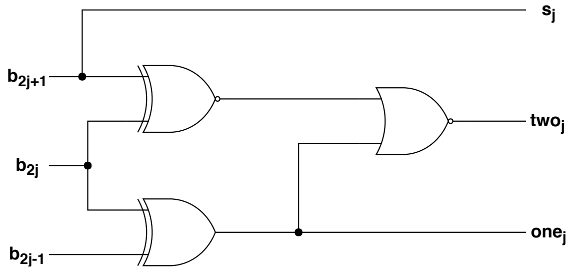
Οι λογικές εξισώσεις που προκύπτουν από τον παραπάνω πίνακα είναι:

$$one_j = b_{2j-1} \oplus b_{2j} \tag{0.1}$$

$$two_j = (b_{2j+1} \oplus b_{2j}) \cdot \overline{one_j} \tag{0.2}$$

$$s_j = b_{2j+1} \tag{0.3}$$

Στο επόμενο σχήμα 1 φαίνεται το κύκλωμα που υλοποιεί τις προαναφερθείσες εξισώσεις:



Σχήμα 1: *One*, *Two* και *s* Σήματα

Στη συνέχεια θα παρουσιαστεί ο τρόπος με τον οποίο παράγονται τα μερικά γινόμενα.

$$P = A \cdot B = \sum_{j=0}^{n/2-1} A \cdot b_j^{MB} \cdot 2^{2j} = ct + \sum_{j=0}^{n/2-1} PP_j \cdot 2^{2j} \tag{0.4}$$

όπου ct είναι ο διορθωτικός όρος και PP_j τα μερικά γινόμενα

Ο διορθωτικός όρος (ct) είναι αναγκαίος για την εξαγωγή σωστού αποτελέσματος και στην περίπτωση που κάποια μερικά γινόμενα είναι αρνητικοί αριθμοί. Στο παρακάτω σχήμα 2 φαίνονται τα βήματα που οηγγούν στον διορθωτικό όρο ct.

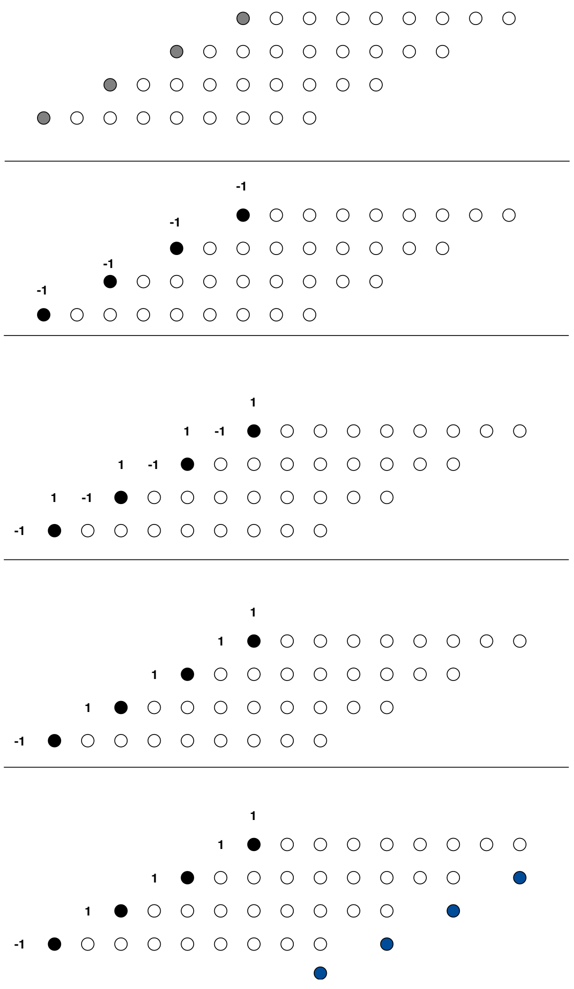
- Πρώτο βήμα: Τα γκρι κυκλάκια αντιπροσωπεύουν το ψηφίο της επέκτασης προσήμου για την περίπτωση που το A πολλαπλασιάζεται με τον αριθμό δύο ($two_j = 1$)

- Δεύτερο βήμα: Το MSB κάθε μερικού γινομένου έχει αρνητικό βάρος και γι΄ αυτό χρησιμοποιείται η σχέση $p + \bar{p} = 1$. Τα γκρι κυκλάκια αντικαθίστανται με τα μαύρα ($\bar{p}$) και επιπλέον προστίθεται ο παράγοντας -1 ($-p = \bar{p} - 1$).

- Τρίτο βήμα: Το μαθηματικό κόλπο $(2 - 1) = 1$ χρησιμοποιείται και προκύπτει $-1 = -2 + 1$.

- Τέταρτο βήμα: Πραγματοποιούνται οι αφαιρέσεις που προέκυψαν από το τρίτο βήμα.

- Πέμπτο βήμα: Προστίθενται τα μπλε κυκλάκια που αντιπροσωπεύουν τα διορθωτικά ψηφία (n_j). Αν το κωδικοποιημένο κατά Modified Booth ψηφίο έχει αρνητικό βάρος ($b_j^{MB} = -1, -2$,ι.ε. $s_j = 1$), τότε το διορθωτικό ψηφίο παίρνει την τιμή ένα έτσι ώστε να γίνει η μετάβαση από το συμπλήρωμα ως προς ένα στο συμπλήρωμα ως προς δύο. Αν όμως το βάρος είναι θετικό, τότε το διορθωτικό ψηφίο έχει την τιμή μηδέν. Αυτός ο μηχανισμός θα εξηγηθεί περαιτέρω, όταν παρουσιαστεί ο τρόπος με τον οποίο παράγονται τα μερικά γινόμενα.



Σχήμα 2: Μερικά γινόμενα και διορθωτικός όρος

Ο διορθωτικός όρος ct υπολογίζεται από την επόμενη εξίσωση:

$$ct = \sum_{j=0}^{3}(\overline{p}_{j,8}2^{8+2j}) \ + \ \{-2^{15} + 2^{13} + 2^{11} + 2^{9} + 2^{8} + n_3 2^6 + n_2 2^4 + n_1 2^2 + n_0 2^0\}$$

όπου $p_{j,8}$ αντιπροσωπεύει την επέκταση προσήμου

Η επέκταση προσήμου συμπεριλαμβάνεται στην παραγωγή των μερικών γινομένων. Επομένως, ο διορθωτικός όρος ct δίνεται από την επόμενη εξίσωση, όπου ct βρίσκεται σε συμπλήρωμα ως προς δύο:

$$ct = \sum_{j=0}^{n/2-1}(n_j \cdot 2^{2j}) \ + \ 2^n(1 \ + \ \sum_{j=0}^{n/2-2}(2^{2j+1}) - 2^{n-1}) \tag{0.5}$$

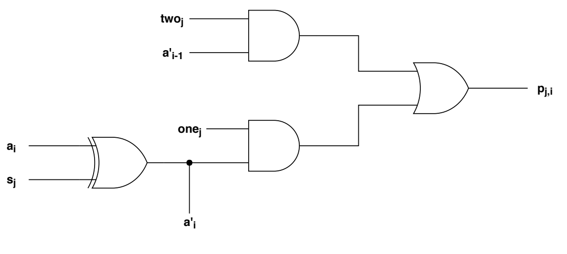
Τα μερικά γινόμενα υπολογίζονται με χρήση των σημάτων one_j, two_j και s_j μέσω των επόμενων εξισώσεων:

$$\begin{aligned} PP_j &= \overline{p}_{j,n} \cdot 2^{n+2J} \ + \ \sum_{i=0}^{n-1}(p_{j,i} \cdot 2^{i+2j}) && (0.6)\\ p_{j,i} &= ((a_i \oplus s_j) \cdot one_j) \ + \ ((a_{i-1} \oplus s_j) \cdot two_j) && (0.7)\\ p_{j,0} &= ((a_0 \oplus s_j) \cdot one_j) \ + \ (s_j \cdot two_j) && (0.8)\\ \overline{p}_{j,n} &= !(((a_{n-1} \oplus s_j) \cdot one_j) \ + \ ((a_{n-1} \oplus s_j) \cdot two_j)) && (0.9) \end{aligned}$$

Ακολουθεί ο πίνακας μερικών γινομένων μαζί με τον διορθωτικό όρο για έναν πολλαπλασιαστή 8×8.

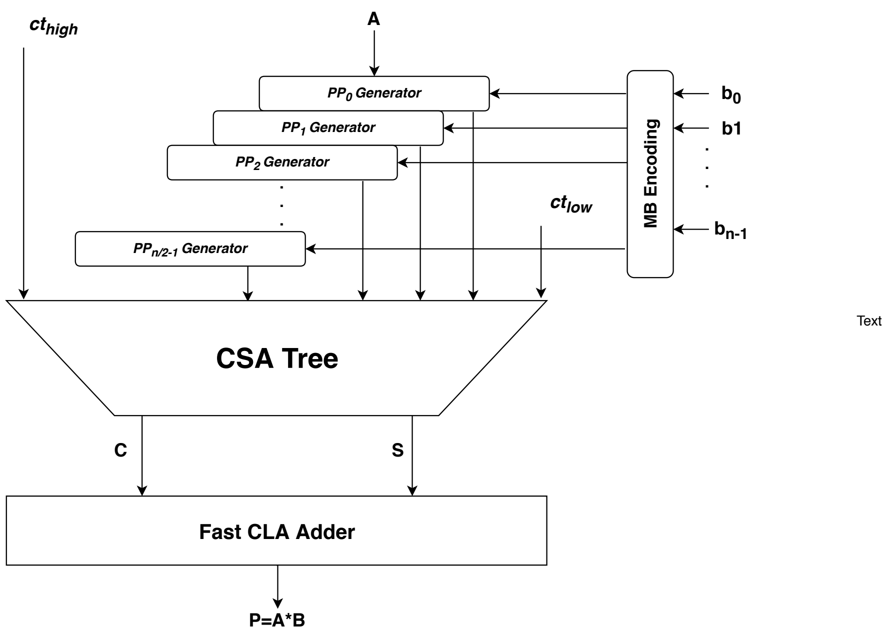
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
							1								
						1	$\overline{p}_{0,8}$	$p_{0,7}$	$p_{0,6}$	$p_{0,5}$	$p_{0,4}$	$p_{0,3}$	$p_{0,2}$	$p_{0,1}$	$p_{0,0}$
				1	$\overline{p}_{1,8}$	$p_{1,7}$	$p_{1,6}$	$p_{1,5}$	$p_{1,4}$	$p_{1,3}$	$p_{1,2}$	$p_{1,1}$	$p_{1,0}$		n_0
		1	$\overline{p}_{2,8}$	$p_{2,7}$	$p_{2,6}$	$p_{2,5}$	$p_{2,4}$	$p_{2,3}$	$p_{2,2}$	$p_{2,1}$	$p_{2,0}$		n_1		
1	$\overline{p}_{3,8}$	$p_{3,7}$	$p_{3,6}$	$p_{3,5}$	$p_{3,4}$	$p_{3,3}$	$p_{3,2}$	$p_{3,1}$	$p_{3,0}$		n_2				
									n_3						

Στο σχήμα 3 παρουσιάζεται το κύκλωμα που υλοποιεί την παραγωγή μερικών γινομένων.



Σχήμα 3: Μονάδα Παραγωγής Μερικών Γινομένων

Τελικά, στο σχήμα 4 παρουσιάζονται όλα τα δομικά στοιχεία ενός πολλαπλασιαστή τα οποία είναι οι γεννήτριες παραγωγής γινομένων, ο MB κωδικοποιητής, το CSA Wallace δέντρο και ο γρήγορος CLA αθροιστής.



Σχήμα 4: Πολλαπλασιαστής Τύπου Modified Booth

Υπάρχουσες τεχνικές στον προσεγγιστικό πολλαπλασιασμό

Οι προσεγγίσεις, που μπορούν να εφαρμοστούν σε έναν πολλαπλασιαστή, χωρίζονται σε δύο κατηγορίες. Η πρώτη κατηγορία περιλαμβάνει προσεγγίσεις που εφαρμόζονται στο στάδιο του Wallace δέντρου είτε αντίστοιχων μονάδων που πραγματοποιούν την άθροιση των μερικών γινομένων. Με αυτή την κατηγορία δεν ασχολείται η παρούσα διπλωματική. Η δεύτερη κατηγορία περιλαμβάνει προσεγγίσεις που εφαρμόζονται στο στάδιο παραγωγής των μερικών γινομένων ή στο στάδιο της MB κωδικοποίησης. Οι τεχνικές που ανήκουν σε αυτή την κατηγορία μπορούν με τη σειρά τους να χωριστούν σε 4 ομάδες:

- Απαλοιφή/Κούρεμα (Elimination/Pruning)
- Κωδικοποιήσεις Radix (Radix Encoding)
- Στρογγυλοποίηση/Διορθωτικοί Όροι (Rounding/Correction Terms)
- Δυναμική Κλιμάκωση (Dynamic Scaling)

Απαλοιφή/Κούρεμα (Elimination/Pruning)

Μία από τις σημαντικότερες τεχνικές αυτής της ομάδας είναι η απαλοιφή μερικών γινομένων (partial product perforation), όπως αναπτύχθηκε από τους Zervakis και άλλους [30].

Έστω δύο αριθμοί των n ψηφίων A,B. Το αποτέλεσμα του πολλαπλασιασμού τους προκύπτει μετά την άθροιση όλων των μερικών γινομέμων Ab_i , όπου b_i είναι το i-στο ψηφίο του B. Έτσι εξάγεται η εξίσωση:

$$A \times B = \sum_{i=0}^{n-1} Ab_i 2^i, \; b_i \in \{0,1\}. \tag{0.10}$$

Η προτεινόμενη τεχνική εφαρμόζει απαλοιφή k διαδοχικών μερικών γινομένων, τα οποία δεν συμπεριλαμβάνονται στο δέντρο υπολογισμού (accumulation tree), και έτσι απαλοίφονται και n πλήρεις αθροιστές(FA). Εφαρμόζοντας λοιπόν αυτή την τεχνική παράγεται η επόμενη εξίσωση:

$$A \times B|_{j,k} = \sum_{\substack{i=0,\\ i \notin [j,j+k)}}^{n-1} Ab_i 2^i, \; b_i \in \{0,1\}. \tag{0.11}$$

Πρέπει να σημειωθεί ότι $j \in [0, n-1]$ και $k \in [1, min(n-j, n-1)]$.

Αντίστοιχα για κωδικοποίηση Modified Booth το κατά προσέγγιση αποτέλεσμα δίνεται από την εξίσωση:

$$A \times B|_{j,k} = \sum_{\substack{i=0,\\ i \notin [j,j+k)}}^{n/2-1} Ab_i^{MB} 4^i, \; b_i^{MB} \in \{0, \pm 1, \pm 2\}. \tag{0.12}$$

Κωδικοποιήσεις Radix (Radix Encoding)

Όπως προηγουμένως έτσι και σε αυτή την ομάδα οι τεχνικές οδηγούν σε μειωμένο αριθμό μερικών γινομένων με τη διαφορά ότι αυτή η μείωση δεν οφείλεται σε απλή απαλοιφή αλλά σε αλλαγές στην κωδικοποίηση.

Μία σημαντική μέθοδος αυτής της ομάδας αναπτύχθηκε από τους συγγραφείς του [13] που την ονόμασαν Hybrid High Radix Encoding. Έστω A και B δύο αριθμοί των n ψηφίων. Ο B χωρίζεται σε δύο μέρη: Το MSB κομμάτι που αποτελείται από $n-k$ ψηφία, και το LSB κομμάτι των k ψηφίων. Για την παράμετρο k ισχύουν: $k \geq 4$, $k = 2m : m \in \mathbb{Z}$, με $m \geq 2$. Στο MSB μέρος εφαρμόζεται η radix-4 κωδικοποίηση. Από την άλλη πλευρά στο LSB μέρος εφαρμόζεται η high radix-2^k κωδικοποίηση. Ο πίνακας 2 παρουσιάζει την radix-4 κωδικοποίηση. Το B μπορεί να εκφραστεί ως εξής:

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i = \sum_{\substack{j=k/2,\\ k \geq 4}}^{n/2-2} y_j^{R4} 4^j + y_0^{R2^k} \tag{0.13}$$

όπου

$$y_j^{R4} = -2b_{2j+1} + b_{2j}b_{2j-1} \tag{0.14}$$

και

$$y_0^{R2^k} = -2^{k-1}b_{k-1} + 2^{k-2}b_{k-2} + \ldots + 2^1 b_1 + b_0 \tag{0.15}$$

Είσοδος			**R4 Ψηφίο**	**Έξοδος**		
b_{2j+1}	b_{2j}	$b_{2j\text{-}1}$	y_j^{R4}	$sign_j$	$\times 2_j$	$\times 1_j$
0	0	0	0	0	0	0
0	0	1	1	0	0	**1**
0	1	0	1	0	0	**1**
0	1	1	2	0	**1**	0
1	0	0	-2	**1**	**1**	0
1	0	1	-1	**1**	0	**1**
1	1	0	-1	**1**	0	**1**
1	1	1	0	**1**	0	0

Πίνακας 2: Ακριβής Κωδικοποίηση Radix-4

Η radix-4 κωδικοποίηση περιλαμβάνει $(n-k)/2$ ψηφία $y_j^{R4} \in \{0, \pm 1, \pm 2\}$, ενώ το $y_0^{R2^k} \in \{0, \pm 1, \pm 2, \pm 3, \ldots, \pm 2^{k-1} - 1, \pm 2^{k-1}\}$ αντιστοιχεί στην radix-2^k κωδικοποίηση. Συνολικά το B κωδικοποιείται σε $(n-k)/2+1$ ψηφία.

Αυτή η high radix κωδικοποίηση χαρακτηρίζεται από αυξημένη πολυπλοκότητα λόγω ότι πρέπει να υπολογιστούν τιμές που δεν είναι δυνάμεις του 2, και γι αυτό το λόγο προτάθηκε από τους συγγραφείς του [13] μια προσεγγιστική μέθοδος. Στο MSB μέρος εφαρμόζεται, όπως προηγουμένως η radix-4 κωδικοποίηση. Όμως, το LSB μέρος κωδικοποιείται κατά προσέγγιση. Συγκεκριμένα όλες οι τιμές που δεν είναι δυνάμεις του δύο καθώς και οι $k-4$ μικρότερες δυνάμεις του δύο θα στρογγυλοποιηθούν στην κοντινότερη τιμή εκ των τεσσάρων μεγαλύτερων δυνάμεων του 2 ή το 0. Το άθροισμα όλων των τιμών των κωδικοποιημένων ψηφίων $\hat{y}_0^{R2^k}$ είναι μηδέν. Έτσι το B προσεγγίζεται ως εξής:

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i = \sum_{\substack{j=k/2,\\ k\geq 4}}^{n/2-2} y_j^{R4} 4^j + \hat{y}_0^{R2^k} \tag{0.16}$$

όπου

$$y_j^{R4} \in \{0, \pm 1, \pm 2\} \tag{0.17}$$

και

$$\hat{y}_0^{R2^k} y_j^{R4} = \in \{0, \pm 2^{k-4}, \pm 2^{k-3}, \pm 2^{k-2}, \pm 2^{k-1}\} \tag{0.18}$$

Από τον πίνακα 2 εξάγονται τα σήματα:

$$sign_j = b_{2j+1} \tag{0.19}$$

$$\times 1_j = b_{2j-1} \oplus b_{2j} \tag{0.20}$$

$$\times 2_j = (b_{2j+1} \oplus b_{2j}) \cdot \overline{(b_{2j-1} \oplus b_{2j})} = (b_{2j+1} \oplus b_{2j}) \cdot \overline{\times 1_j} \tag{0.21}$$

Ο πίνακας 3 παρουσιάζει την προσεγγιστική υλοποίηση της radix-2^k κωδικοποίησης. Οι λογικές εξισώσεις των κωδικοποιημένων σημάτων, που ορίζουν το radix-2^k ψηφίο $\hat{y}_0^{R2^k}$, είναι:

$$sign = b_{k-1} \tag{0.22}$$

$$\times 2^{k-4} = (\bar{b}_{k-2} \cdot \bar{b}_{k-3} \cdot \bar{b}_{k-4} + b_{k-2} \cdot b_{k-3} \cdot b_{k-4}) \cdot (b_{k-4} \oplus b_{k-5}) \tag{0.23}$$

$$\times 2^{k-3} = \bar{b}_{k-1} \cdot \bar{b}_{k-2} \cdot (\bar{b}_{k-3} \cdot b_{k-4} \cdot b_{k-5} + b_{k-3} \cdot \bar{b}_{k-4}) + b_{k-1} \cdot b_{k-2} \cdot (b_{k-3} \cdot \bar{b}_{k-4} \cdot \bar{b}_{k-5} + \bar{b}_{k-3} \cdot b_{k-4}) \tag{0.24}$$

$$\times 2^{k-2} = \bar{b}_{k-2} \cdot b_{k-3} \cdot (b_{k-1} + b_{k-4}) + b_{k-2} \cdot \bar{b}_{k-3} \cdot (\bar{b}_{k-1} + \bar{b}_{k-4}) \tag{0.25}$$

$$\times 2^{k-1} = \bar{b}_{k-1} \cdot b_{k-2} \cdot b_{k-3} + b_{k-1} \cdot \bar{b}_{k-2} \cdot \bar{b}_{k-3}. \tag{0.26}$$

R2$^{\mathbf{k}}$ Ψηφίο		**Έξοδος**				
$y_0^{R2^k}$	$\hat{y}_0^{R2^k}$	$sign$	$\times 2^{k\text{-}1}$	$\times 2^{k\text{-}2}$	$\times 2^{k\text{-}3}$	$\times 2^{k\text{-}4}$
$[0,\ 2^{k\text{-}5})$	0	0	0	0	0	0
$[2^{k\text{-}5},\ 2^{k\text{-}4}+2^{k\text{-}5})$	$2^{k\text{-}4}$	0	0	0	0	**1**
$[2^{k\text{-}4}+2^{k\text{-}5},\ 2^{k\text{-}3}+2^{k\text{-}4})$	$2^{k\text{-}3}$	0	0	0	**1**	0
$[2^{k\text{-}3}+2^{k\text{-}4},\ 2^{k\text{-}2}+2^{k\text{-}3})$	$2^{k\text{-}2}$	0	0	**1**	0	0
$[2^{k\text{-}2}+2^{k\text{-}3},\ 2^{k\text{-}1})$	$2^{k\text{-}1}$	0	**1**	0	0	0
$[\text{-}2^{k\text{-}1},\ \text{-}2^{k\text{-}2}\text{-}2^{k\text{-}3})$	$\text{-}2^{k\text{-}1}$	**1**	**1**	0	0	0
$[\text{-}2^{k\text{-}2}\text{-}2^{k\text{-}3},\ \text{-}2^{k\text{-}3}\text{-}2^{k\text{-}4})$	$\text{-}2^{k\text{-}2}$	**1**	0	**1**	0	0
$[\text{-}2^{k\text{-}3}\text{-}2^{k\text{-}4},\ \text{-}2^{k\text{-}4}\text{-}2^{k\text{-}5})$	$\text{-}2^{k\text{-}3}$	**1**	0	0	**1**	0
$[\text{-}2^{k\text{-}4}\text{-}2^{k\text{-}5},\ \text{-}2^{k\text{-}5})$	$\text{-}2^{k\text{-}4}$	**1**	0	0	0	**1**
$[\text{-}2^{k\text{-}5},\ 0)$	0	**1**	0	0	0	0

Πίνακας 3: Προσεγγιστική Κωδικοποίηση Radix-2^k

Στρογγυλοποίηση/Διορθωτικοί Όροι (Rounding/Correction Terms)

Η τεχνική που επιλέχτηκε να παρουσιαστεί είναι εκείνη που αναπτύχθηκε απο τους συγγραφείς του [14] και αποτελεί χαρακτηριστικό παράδειγμα αυτής της ομάδας. Η τεχνική αυτή ονομάστηκε Hybrid Partial Product Perforation-Rounding.

Partial Product Perforation

Η μέθοδος αυτή είναι συνδυασμός δύο επιμέρους τεχνικών. Η πρώτη παρσουσιάστηκε σε προηγούμενη ομάδα. Στο [14] χρησιμοποιήθηκε ως εξής. Έστω A και B δύο αριθμοί των n ψηφίων σε συμπλήρωμα ως προς δύο. Τα k πρώτα συνεχόμενα μερικά γινόμενα απαλοίφησαν. Έτσι τα k λιγότερο σημαντικά κωδικοποιημένα κατά Modified Booth ψηφία δεν παράγονται. Άρα τα $2k$ LSBs του B(συμπεριλαμβανομένου του b_{-1}) περιττεύουν και επομένως απαλοίφονται. Τελικά το γινόμενο $A \times B$ υπολογίζεται προσεγγιστικά από την επόμενη εξίσωση:

$$A \times B|_k = \sum_{j=k}^{n/2-1} A \cdot b_j^{MB} \cdot 4^j \tag{0.27}$$

Partial Product Rounding

Το δεύτερο κομμάτι αυτής της μεθόδου είναι αυτό που την κατατάσσει σε αυτή την ομάδα. Η ιδέα είναι η μη χρησιμοποίηση των $m-1$ LSBs του A, και η προσθήκη του ψηφίου a_{m-1} στο υπόλοιπο πιο σημαντικό κομμάτι (A_m),όπως φαίνεται παρακάτω.

$$A_m + a_{m-1} = \langle a_{n-1}, a_{n-2} \dots\ a_m \rangle_{2'S} + a_{m-1} \tag{0.28}$$

Το ‘‘κουτσούρεμα’’ των $m-1$ LSBs θα οδηγούσε σε σημαντικά σφάλματα. Προκειμένου να αποφευχθεί αυτό, το τελευταίο LSB (a_{m-1}) προστίθεται στο A_m. Στη συνέχεια παρουσιάζονται οι δύο περιπτώσεις για τις διαφορετικές τιμές του a_{m-1}.

Αν $a_{m-1} = 0$, τότε τα κατά προσέγγιση μερικά γινόμενα ($\tilde{P}_j$) υπολογίζονται από την σχέση: $\tilde{P}_j = (A_m + 0) \cdot b_j^{MB} = A_m \cdot b_j^{MB}$

Εάν $a_{m-1} = 1$ και χρησιμοποιώντας τη σχέση $A_m + 1 = -\overline{A}_m$, τα κατά προσέγγιση μερικά γινόμενα ($\tilde{P}_j$) υπολογίζονται από την εξίσωση:
$\tilde{P}_j = (A_m+1) \cdot b_j^{MB} = (-\overline{A}_m) \cdot b_j^{MB} = \overline{A}_m \cdot (-b_j^{MB})$, όπου $(-b_j^{MB}) = (-1)^{\overline{s}_j} \cdot (2 \cdot two_j + one_j)$.

Χρησιμοποιώντας τη σχέση $A_m^* = A_m \oplus a_{m-1}$ προκειμένου να σχηματιστεί το A_m ή το $\overline{A}_m$ οι δύο περιπτώσεις μπορούν να συγχωνευτούν. Επιπλέον χρησιμοποιείται η σχέση $s_j^* = s_j \oplus a_{m-1}$ για να σχηματιστεί το b_j^{MB} ή $-b_j^{MB}$. Έτσι η σχέση που υπολογίζει τα μερικά γινόμενα διαμορφώνεται ως εξής: $\tilde{P}_j = A_m^* \cdot b_j^{MB^*}$, ωηερε $b_j^{MB^*} = (-1)^{s_j^*} \cdot (2 \cdot two_j + one_j)$.

Τα δύο προαναφερθέντα κομμάτια συνδυάζονται και σχηματίζουν την περιγραφόμενη τεχνική του [14] δημιουργώντας την εξίσωση:

$$A \times B = \sum_{j=k}^{n/2-1} \tilde{P}_j \cdot 4^j = \sum_{j=k}^{n/2-1} A_m^* \cdot b_j^{MB^*} \cdot 4^j, \text{ ωηερε } k \in [0, n/2-1) \text{ ανδ } m \in [0, n-1) \tag{0.29}$$

Ο διορθωτικός όρος ct στην περίπτωση αυτή έχει ελαφρώς διαφορετικά διορθωτικά ψηφία για να επιτευχθεί η μετάβαση από το συμπλήρωμα ως προς ένα στο συμπλήρωμα ως προς δύο και αυτά δίνονται από την παρακάτω εξίσωση:

$$c_j^* = c_j^* \cdot (one_j + two_j)$$

Δυναμική Κλιμάκωση (Dynamic Scaling)

Στην παρούσα διπλωματική δεν χρησιμοποιείται ούτε αναπτύσσεται κάποια τεχνική που να υπάγεται σε αυτή την κατηγορία. Αξίζει όμως να αναφερθούν δύο βασικές τεχνικές αυτής της ομάδας. Η πρώτη αναπτύχθηκε από τους Narayanamoorthy και άλλους στο [22], οι οποίοι πρότειναν έναν πολλαπλασιαστή που χρησιμοποιεί m διαδοχικά ψηφία σαν τμηματική είσοδο. Για να πετύχουν κλιμακούμενη ακρίβεια εισήγαγαν μια μέθοδο, την αποκαλούμενη static segment method, η οποία διαλέγει την αρχή του τμήματος που χρησιμοποιείται σαν είσοδο. Η δεύτερη προτάθηκε από τους συγγραφείς του [6], οι οποίοι αξιοποίησαν το γεγονός ότι κάποια ψηφία είναι πιο σημαντικά από άλλα. Έτσι πρότειναν μία τεχνική που εντοπίζει τη θέση του πιο σημαντικού άσσου και στη συνέχεια ο άσσος και τα επόμενα $k-2$ διαδοχικά ψηφία, ανάλογα με την επιθυμητή ακρίβεια, χρησιμοποιούνται για τον πολλαπλασιασμό. Για το κομμάτι που απορρίπτεται εφαρμόζεται μία προσέγγιση σύμφωνα με την ομοιόμορφη κατανομή, ώστε να μειωθεί το σφάλμα.

Προτεινόμενες Τεχνικές

Στην παρούσα διπλωματική αναπτύχθηκαν 5 τεχνικές για τον προσεγγιστικό πολλαπλασιασμό με σκοπό την υψηλή ακρίβεια και τη μείωση της κατανάλωσης. Οι τεχνικές που αναπτύχθηκαν είναι:

- Double High Radix Encoding
- Double High Radix with Perforation
- High Radix with Correction
- Perforation with Correction
- Asymmetric Perforation and Rounding

Double High Radix Encoding

Η ιδέα γι αυτή την τεχνική προήλθε από την προηγουμένως αναφερθείσα τεχνική, hybrid high radix encoding. Πρακτικά η κωδικοποίηση που εφαρμόστηκε στο B στην παρούσα τεχνική επεκτείνεται και εφαρμόζεται και στο A. Για την καλύτερη κατανόηση παρουσιάζονται οι παρακάτω εξισώσεις:

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i = A_1 + x_0^{R2^m} \tag{0.30}$$

$$\text{όπου } A_1 = -a_{n-1}2^{n-1} + \sum_{\substack{j=m \\ m\geq 4}}^{n-2} a_i 2^i + a_{m-1}2^{m-1} \tag{0.31}$$

$$\text{και } x_0^{R2^m} = -2^{m-1}a_{m-1} + 2^{m-2}a_{m-2} + \ldots + a_0 \tag{0.32}$$

A_1 είναι το MSB μέρος του A και το $x_0^{R2^m} \in \{0, \pm1, \pm2, \pm3, \ldots, \pm(2^{m-1}-1), -2^{m-1}\}$ αντιστοιχεί στην high radix-2^m κωδικοποίηση.

$$B \quad = \quad -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i = B_1 + y_0^{R2^k} \tag{0.33}$$

$$\text{όπου} \;\; B_1 = \sum_{\substack{j=k/2 \\ k\geq 4}}^{n/2-1} y_j^{R4} 4^j \tag{0.34}$$

$$\text{και} \;\; y_j^{R4} = -2b_{2j+1} + b_{2j} + b_{2j-1} \tag{0.35}$$

$$\text{και} \;\; y_0^{R2^k} = -2^{k-1}b_{k-1} + 2^{k-2}b_{k-2} + \ldots + b_0 \tag{0.36}$$

Στο MSB μέρος του B εφαρμόζεται η radix-4 κωδικοποίηση, ενώ στο LSB η high radix-2^k κωδικοποίηση. Στη συνέχεια γίνεται ο πολλαπλασιασμός με τον τρόπο που δείχνει η παρακάτω εξίσωση.

$$\begin{aligned} A \times B &= (A_1 + x_0^{R2^m}) \cdot (B_1 + y_0^{R2^k}) \\ &= (A_1 + x_0^{R2^m}) \cdot B_1 + (A_1 + x_0^{R2^m}) \cdot y_0^{R2^k} \\ &= A_1 \cdot B_1 + B_1 \cdot x_0^{R2^m} + A \cdot y_0^{R2^k} \end{aligned} \tag{0.37}$$

Όπως έχει ήδη αναφερθεί λόγω αυξημένης πολυπλοκότητας γίνεται η εξής προσέγγιση:

$$\tilde{A} = A_1 + \hat{x}_0^{R2^m} \tag{0.38}$$

$$\tilde{B} = B_1 + \hat{y}_0^{R2^k} \tag{0.39}$$

$$\text{όπου} \;\; \hat{x}_0^{R2^m} \in \{0, \pm 2^{m-4}, \pm 2^{m-3}, \pm 2^{m-2}, \pm 2^{m-1}\} \tag{0.40}$$

$$\text{και} \;\; \hat{y}_0^{R2^k} \in \{0, \pm 2^{k-4}, \pm 2^{k-3}, \pm 2^{k-2}, \pm 2^{k-1}\} \tag{0.41}$$

Τελικά το γινόμενο διαμορφώνεται όπως δείχνει η επόμενη εξίσωση:

$$\tilde{A} \times \tilde{B} = A_1 \cdot B_1 + B_1 \cdot \hat{x}_0^{R2^m} + A \cdot \hat{y}_0^{R2^k} \tag{0.42}$$

Οι κωδικοποιήσεις radix-4 και radix-2^k φαίνονται στους πίνακες 2 και 3 αντίστοιχα. Οι ίδιες λογικές εξισώσεις 0.19-0.26 εξακολουθούν να εφαρμόζονται για την υλοποίηση των κωδικοποιήσεων.

Double High Radix with Perforation

Η τεχνική αυτή είναι άμεσο αποτέλεσμα της προαναφερθείσας τεχνικής , Double High Radix Encoding, εάν απαλειφθεί ο όρος $A\hat{y}_0^{R2^k}$. Έτσι η εξίσωση που την περιγράφει είναι:

$$\tilde{A} \times \tilde{B} = A_1 \cdot B_1 + B_1 \cdot \hat{x}_0^{R2^m} \tag{0.43}$$

High Radix with Correction

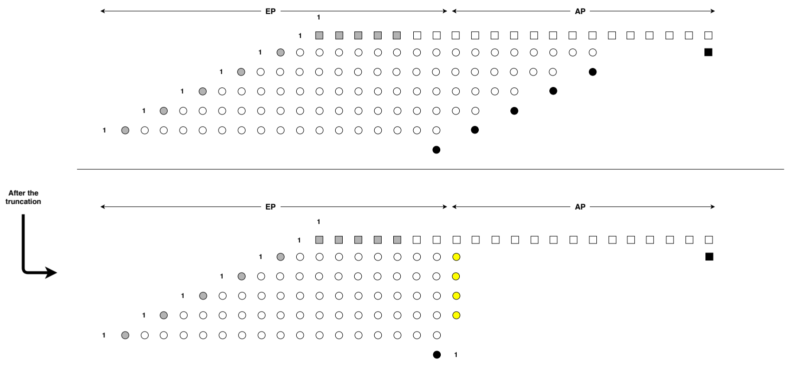
Αυτή η τεχνική αποτελεί συνδυασμό δύο μεθόδων. Η πρώτη είναι η hybrid high radix encoding, όπως έχει παρουσιαστεί προηγουμένως και στο [13]. Σύμφωνα με αυτή το γινόμενο δίνεται από την σχέση:

$$A \times B = A \cdot (B_1 \ + \ \hat{y}_0^{R2^k}) = A \cdot B_1 \ + \ A \cdot \hat{y}_0^{R2^k} \tag{0.44}$$

Στη συνέχεια η δεύτερη μέθοδος αξιοποιεί το γεγονός ότι κάποια ψηφία είναι πιο σημαντικά απο κάποια άλλα. Έτσι στον πίνακα των μερικών γινομένων που παράγεται από τον όρο $A \cdot B_1$ εφάρμοζεται αυτή η μέθοδος. Συγκεκριμένα, ο πίνακας χωρίζεται σε δύο σκέλη, στο ακριβές μέρος (exact part-EP) και στο κατά προσέγγιση μέρος (approximate part-AP). Το AP προσεγγίζεται ως εξής: Η αναμενόμενη τιμή μιας ομοιόμορφης κατανομής στο διάστημα των αριθμών $[0, 2^t - 1]$ είναι 2^{t-1}. Αυτή η προσέγγιση δεν εφαρμόζεται στο μερικό γινόμενο που παράγεται από τον όρο $A \cdot \hat{y}_0^{R2^k}$. Στο σχήμα 5 φαίνεται πώς εφαρμόζεται αυτή η τεχνική για έναν πολλαπλασιαστή 16×16. Οι αναμενόμενες τιμές της ομοιόμορφης κατανομής αποδίδονται από την λογική εξίσωση:

$$y'_i = \ \times 1_i + \ \times 2_i \text{ με } i = 1...4 \tag{0.45}$$

Ο επιπρόσθετος άσσος στον διορθωτικό όρο εισάγεται για περαιτέρω μείωση του σφάλματος, καθώς λειτουργεί σαν ένα είδος στρογγυλοποίησης προς τα πάνω.



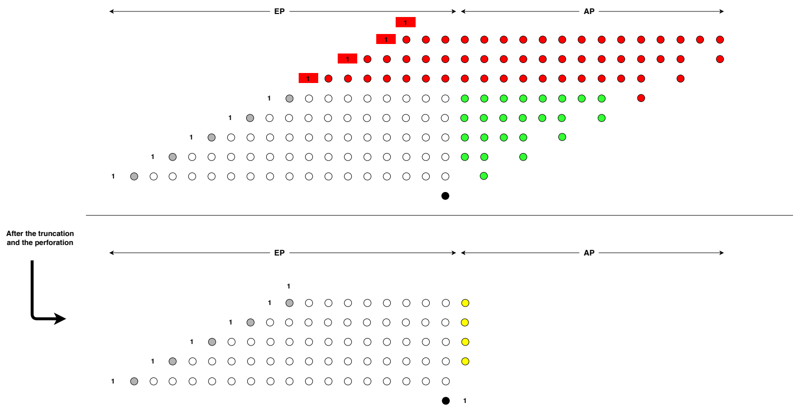
Σχήμα 5: RADC$|_{64,8}$

Perforation with Correction

Η ιδέα πίσω από αυτή την τεχνική είναι ίδια με αυτή της προηγούμενης με την διαφορά ότι δεν εφαρμόζεται radix-2^k κωδικοποίηση αλλά απαλοιφή μερικών γινομένων (Partial Product Perforation) όπως παρουσιάστηκε παραπάνω και στο [30]. Στο σχήμα 6 παρουσιάζεται η εφαρμογή αυτής της τεχνικής σε έναν 16×16 πολλαπλασιαστή. Η αναμενόμενη τιμή της ομοιόμορφης κατανομής αποδίδεται από την λογική εξίσωση:

$$y'_i = \ \times 1_i \ + \ \times 2_i \text{ με } i = 3...7 \tag{0.46}$$

Ο επιπρόσθετος άσσος στον διορθωτικό όρο εισάγεται και πάλι για περαιτέρω μείωση του σφάλματος, καθώς λειτουργεί σαν ένα είδος στρογγυλοποίησης προς τα πάνω.



Σχήμα 6: PERFOC$|_{3,8}$

Asymmetric Perforation and Rounding

Η τεχνική Asymmetric Perforation and Rounding είναι μία παραλλαγή της τεχνικής hybrid partial product perforation-rounding, η οποία αναπτύχθηκε στο [14]. Η απαλοιφή των μερικών γινομένων εφαρμόζεται όπως και στην προηγούμενη τεχνική (perforation with correction). Η διαφορά αυτής της τεχνικής και εκείνης που περιγράφηκε στο [14] είναι ο τρόπος με τον οποίο εφαρμόζεται η στρογγυλοποίηση. Συγκεκριμένα εφαρμόζεται στρογγυλοποίηση στο ψηφίο της θέσης t_i σε κάθε μερικό γινόμενο (όπου i παίρνει τις τιμές του πρώτου μη απαλοιφόμενου μερικού γινομένου έως του τελευταίου). Επομένως, τα $t_i - 1$ LSBs του A δεν χρησιμοποιούνται και το ψηφίο a_{t_i-1} προστίθεται στο υπόλοιπο κομμάτι, όπως δείχνει η παρακάτω εξίσωση:

$$A_{t_i} + a_{t_i-1} = \langle a_{n-1}, a_{n-2} \dots a_{t_i} \rangle_{2'S} + a_{t_i-1} \tag{0.47}$$

Στην περίπτωση που $a_{t_i-1} = 0$ τα κατά προσέγγιση μερικά γινόμενα ($\hat{P}_j$) υπολογίζονται ως εξής:

$$\hat{P}_j = (A_{t_i} + 0)y_j^{R4} = A_{t_i} y_j^{R4} \tag{0.48}$$

$$\text{όπου } y_j^{R4} \text{ υπολογίζεται στην (0.35)}$$

Στην περίπτωση που $a_{t_i-1} = 1$ και χρησιμποποιώντας τη σχέση $A_{t_i} + 1 = -\bar{A}_{t_i}$ τα μερικά γινόμενα υπολογίζονται από την εξίσωση:

$$\hat{P}_j = (A_{t_i} + 1)y_j^{R4} = (-\bar{A}_{t_i})y_j^{R4} = \bar{A}_{t_i}(-y_j^{R4}) \ , \tag{0.49}$$

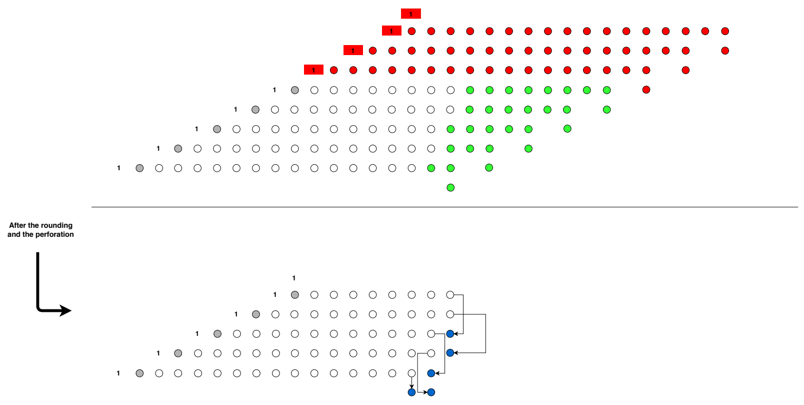
$$\text{όπου } -y_j^{R4} = (-1)^{\bar{s}_j}(x2_j + x1_j)$$

Οι δύο περιπτώσεις συνδυάζονται με την βοήθεια των σχέσεων
$A^*_{t_i} = A_{t_i} \oplus a_{t_i-1}$ για τον σχηματισμό του A_{t_i} ή του $-\bar{A}_{t_i}$ και $s^*_j = s_j \oplus a_{t_i-1}$ για τον σχηματισμό του y_j^{R4} ή του $-y_j^{R4}$. Έτσι τα μερικά γινόμενα υπολογίζονται από την εξίσωση:$\hat{P} =$

$A^*_{t_i} y_j^{R4*}$ όπου $-y_j^{R4} = (-1)^{\bar{s}_j}(x2_j + x1_j)$. Όσο για τα διορθωτικά ψηφία που πραγματοποιούν τη μετάβαση από το συμπλήρωμα ως προς ένα στο συμπλήρωμα ως προς δύο ισχύει:

$$c_j^* = s_j^* \wedge (x2_j \vee x1_j)$$

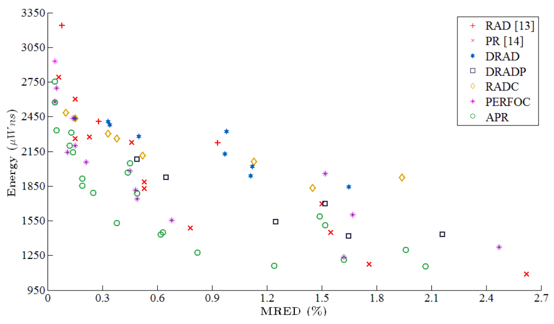
Στο σχήμα 7 παρουσιάζεται ένα παράδειγμα αυτής της τεχνικής. Η ασυμμετρία που φαίνεται στο σχήμα αυτό οφείλεται στο γεγονός ότι η στρυγγυλοποίηση δε μπορεί να εφαρμοστεί εντελώς κάθετα χωρίς παράλληλα να αυξηθεί το βάθος του δέντρου υπολογισμού των μερικών γινομένων (accumulation tree). Γι αυτό το λόγο εφαρμόζεται όσο πιο κάθετα είναι δυνατόν.



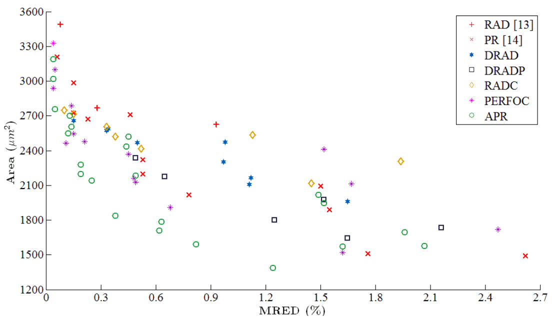
Σχήμα 7: Πίνακας Μερικών γινομένων του APR$|_{3,10}$

Πειραματικά Αποτελέσματα

Στο κεφάλαιο 4 παρουσιάζονται αναλυτικά τα αποτελέσματα κάθε μίας από τις 5 τεχνικές, όπως αποτυπώνονται στους πίνακες 4.1-4.10. Σε αυτούς του πίνακες αναγράφονται όλες οι μετρήσεις, όσον αφορά την κατανάλωση ενέργειας και τον απαιτούμενο χώρο, στη μέγιστη συχνότητα λειτουργίας αλλά και σε μία κοινή συχνότητα. Στα παρακάτω σχήματα παρουσιάζεται η αξιόλογηση κάθε τεχνικής σε διαγράμματα Pareto.

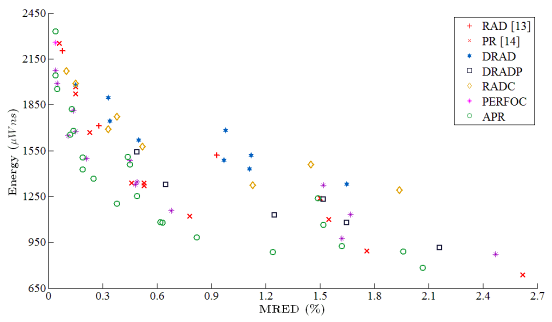


(α′) Energy-Error

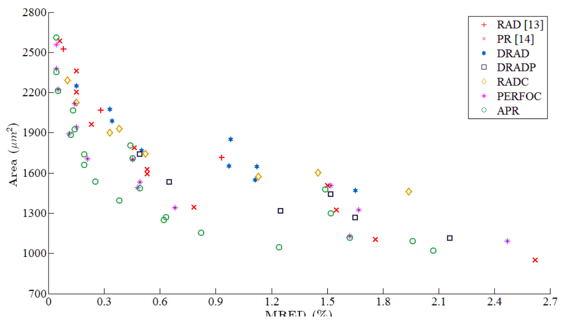


(β′) Area-Error

Σχήμα 8: Αξιολόγηση των τεχνικών σε μέγιστη συχνότητα λειτουργίας.



(α′) Energy-Error



(β′) Area-Error

Σχήμα 9: Αξιολόγηση των τεχνικών σε κοινή συχνότητα λειτουργίας.

Βάσει των παραπάνω σχημάτων 8 και 9, μπορεί κανείς εύκολα να συναγάγει ότι τα καλύτερα αποτελέσματα προέρχονται από την τεχνική asymmetric perforation and rounding και ότι η δεύτερη καλύτερη τεχνική είναι η perforation with correction. Σε πολλές δε περιπτώσεις έρχεται πολύ κοντά με την πρώτη. Οι υπόλοιπες τεχνικές βρίσκονται αρκετά μακριά από τις δύο καλύτερες λόγω του γεγονότος ότι οι προσεγγίσεις που έγιναν προκάλεσαν μεγάλη απώλεια στην ακρίβεια χωρίς να επιφέρουν ανάλογη μείωση στην κατανάλωση ενέργειας.

Τέλος, μία μελλοντική εργασία θα μπορούσε να αποτελέσει η υλοποίηση όλων των παραπάνω τεχνικών για μεγαλύτερο ή μικρότερο αριθμό ψηφίων. Μία επιπλέον δυνατότητα για μελλοντική εργασία θα μπορούσε να είναι η υλοποίηση μίας τεχνικής που θα είναι μία παραλλαγή της high radix with correction, με τη διαφορά ότι θα εφαρμόζεται η στρυγγυλοποίηση που χρησιμοποιήθηκε στην asymmetric perforation and rounding.. Οι συνδυασμοί που μπορούν να προκύψουν χρησιμοποιώντας τις τεχνικές από τις προαναφερθείσες κατηγορίες είναι πολλοί και η περαιτέρω μελέτη αυτών μπορεί να επιφέρει ακόμα καλύτερα αποτελέσματα από τα ήδη υπάρχοντα.

Chapter 1

Introduction-Motivation

In recent years, the embedded and mobile nature of modern computing systems has led to an increased need for high performance and energy efficiency. As a result, since the failure of Dennard scaling, energy dissipation has become a first class concern in the design of integrated circuits. Towards this direction, approximate (or inexact) computing appears as an emerging and promising solution for energy-efficient systems design [5], exploiting the inherent error/noise resilience of various applications. More explicitly, perfect answers are often unnecessary (or do not exist) in a large number of application domains involving media processing, machine learning, data mining and statistics [2]. This relaxation in the requirements for exactness is favored due to several factors [1]:

1. The limited human perception
2. The complexity of defining/producing exact results
3. The noisy and/or redundant input data
4. The application's capability to self-heal and absorb the errors
5. the probabilistic/statistical calculations
6. the user's intention to accept results of lower quality

Interestingly, benefiting from the aforementioned factors, error is considered as a commodity that can be traded for significant gains in performance, area, power, energy, etc. [16].

Targeting to take advantage of the error tolerance, massive research has been reported in the field of approximate computing at multiple layers of software and hardware [20, 29]. At software level, multiple abstractions have been proposed: approximation-aware programming languages that let the programmer define the accuracy of the results [25] and approximation-aware compilers that change the semantics of the programs to trade the accuracy of the results [19]. At hardware level, the main targets are the adders [7] and the multipliers [14], i.e., the core units of hardware accelerators. The approximations are applied at various design layers of abstraction, i.e., the application, algorithmic, architecture, gate and transistor layers [4]. Extensive research has also been conducted in approximate processors, using neural networks [3], quality programmable vectors [28] and approximate custom instructions [11].

Approximate methods have been extensively applied in the design of inexact circuits, due to delivering lower dynamic and leakage power consumption. Circuit approximations can be introduced through voltage over-scaling (VOS) [23], over clocking (OC) [10], and logic simplification [7]. The main focus of this diploma thesis is approximations applied in arithmetic circuits, and specifically the hardware multipliers, the most energy-hungry components of accelerators involving computationally intensive kernels (DSP, neural networks, etc.). The majority of existing works on inexact multipliers explores approximations either on the partial product generation [14, 30, 13] or the partial product accumulation [21, 24, 17]. These approximation targets are synergistic and can be applied in collaboration, increasing the total energy/area savings [8, 18].

Past research activities on approximate multipliers have shown that the direct application of inexact adders in the partial product accumulation is not very efficient in terms of accuracy, hardware complexity and other performance metrics [21]. On the other hand, approximations on the partial product generation deliver simpler partial product arrays, and thus, there is significant reduction in the critical paths and the total accumulation complexity. Inspired from the promising results of lossy partial product generation [14], in this work new approximate encodings for inaccurate yet energy-friendly hardware multipliers are explored. More specifically, 5 approximate design families, that can be configured to adjust the error-energy trade-off w.r.t. the acceptable accuracy loss, are proposed and implemented.

Chapter 2

Theoretical Background

2.1 Introduction

In this chapter all the techniques and algorithms required for the understanding of this diploma thesis will be extensively analyzed. Specifically the binary numeral system, booth algorithms **etc** will be explained.

2.2 Binary Numeral System

The binary numeral system is the most important number system in computer science. The basic idea behind it, is that every number is presented using only two digits $\{0, 1\}$, which are referred to as bits. The binary numeral system has as base the number 2 and every number can be written in it using the following equation:

$$A_{(10)} = \sum_{i=0}^{n-1} 2^i \, a_i = a_{n-1}a_{n-2}...a_{0\,(2)} \qquad (2.2.0.1)$$

where n is the total number of bits

For example the number 11 in the binary numeral system is 1011. Using the equation (2.2.0.1) the binary number 1011 corresponds to:
$1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 = 11$ .The total number of bits of a binary number (n) defines the maximum decimal number that can be displayed by the binary number. The maximum value is 2^n-1 and not 2^n because zero is included. Each digit is multiplied with a power of two which results in some bits being more significant than others. In the aforementioned example the bit multiplied with 2^3 is the most significant (MSB) and in all cases the bit multiplied with 2^0 is the least significant bit(LSB). In the table 2.1 some examples of numbers in both numerical systems are presented.

2.2.1 Two's complement notation

Binary numbers as defined so far can represent only positive values. In order to represent both positive and negative values, the two's complement notation is used, modifying

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Table 2.1: Counting from 0 to 15 in binary numeral system

the equation (2.2.0.1) as follows:

$$A_{(10)} = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} 2^i \; a_i = a_{n-1}a_{n-2}...a_{0\;(2)} \tag{2.2.1.1}$$

As it is shown in this equation (2.2.1.1) the MSB has negative weight and as a result all binary numbers whose MSB is '1' are negative. Consequently, the MSB is often called the sign bit. To get the additive inverse of a binary number, the two's complement of the absolute value is used.For instance using the number $0110_{(2)} = 6_{(10)}$, first the ones' complement is computed. This is achieved by inverting the bits of the number thus getting $0110 \rightarrow 1001$. To get from ones' complement to two's complement the only step required is to add 1. The result in the above example is 1010. To find which decimal number 1010 is the equation (2.2.1.1) is used resulting in $1010_{(2)} = -1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = -6_{(10)}$. The interval of values that can represent a n-bit number is $[-2^{n-2}, 2^{n-2} - 1]$.

2.2.2 Mathematical operation-Addition

The addition of two positive binary numbers is similar to the addition of decimal numbers. We will analyze the addition of binary numbers in all possible cases.

Addition of two positive numbers

If the MSB of both numbers is zero and the MSB of the result is also zero, it is certain that an overflow condition doesn't exist. For instance:

$$\begin{array}{rr} 0010 & 2 \\ +0100 & +4 \\ \hline 0110 & 6 \end{array}$$

Next an example is displayed, where overflow condition exists and the result is incorrect.

$$\begin{array}{rr} 0011 & 3 \\ +0101 & +5 \\ \hline 3\quad 1000 & -8 \end{array}$$

The solution to the overflow condition is called sign extension. Two numbers can be added without overflow if both of them are sign-extended before the addition happens. This means that their total number of bits is increased from n to $n+1$ by copying the MSB. Specifically: $a_{n-1}\ a_{n-2}\ ...\ a_0 \longrightarrow a_{n-1}\ a_{n-1}\ a_{n-2}\ ...\ a_0$. T

Addition of a negative and a positive number

In this case it is impossible that the operation leads to an overflow condition. Two examples are presented for both negative and positive results. First for the positive result:

$$\begin{array}{rr} 1011 & -5 \\ +0111 & +7 \\ \hline 0010 & 2 \end{array}$$

And second for the negative:

$$\begin{array}{rr} 0011 & 3 \\ +1001 & -7 \\ \hline 1100 & -4 \end{array}$$

Addition of two negative numbers

Similarly to the addition of two positive numbers the operation can also lead to overflow condition. The solution is the same as before, sign extension is applied. Two examples, one for correct and one for incorrect results, will be presented.First for the correct result:

$$\begin{array}{rr} 1011 & -5 \\ +1110 & -2 \\ \hline 1001 & -7 \end{array}$$

And second for the incorrect due to overflow:

$$\begin{array}{rr} 1001 & -7 \\ +1011 & -5 \\ \hline 0100 & 4 \end{array}$$

2.2.3 Carry-Save notation

Carry-save(CS) notation belongs to the category of Redundant Arithmetic Systems. There are more than one carry-save notation that can implement the same decimal number. The relation describing CS notation is $x^* = x^s + x^c$. So the value of a decimal number is written as the sum of two numbers. The advantage of this notation is the quick execution of the operation of addition and subtraction because of the lack of the carry propagation. The downside is that for a $n-$bit number in two's complement notation $2n$ bits are needed to describe him.

2.3 Booth algorithms

The Booth's multiplication algorithm was developed about four decades ago in order to perform the multiplication of two signed numbers in an efficient way. Since then it is widely used with various modifications.

2.3.1 Booth's multiplication Algorithm

In this subsection the Booth's multiplication algorithm will be examined. Let A,B be two $n-$bit signed numbers, which produce $2n-$ bit product P. The 2's complement A,B and P can be expressed as:

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \tag{2.3.1.1}$$

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \tag{2.3.1.2}$$

$$P = -p_{2n-1}2^{2n-1} + \sum_{i=0}^{2n-2} p_i 2^i \tag{2.3.1.3}$$

Using a simple mathematical trick B can be written: $B = 2B - B$. Thus, Z is formed:

$$\begin{array}{rcccccc}
2B = & -b_{n-1} & b_{n-2} & b_{n-3} & \dots & b_0 & 0 \\
-B = & 0 & b_{n-1} & -b_{n-2} & \dots & -b_1 & -b_0 \\
\hline
Z = & z_{n-1} & & z_{n-2} & \dots & z_1 & z_0
\end{array}$$

The bits of Z are: $z_0 = 0 - b_0$, $z_1 = b_0 - b_1$, ... , $z_{n-1} = -2b_{n-1} + b_{n-2} + b_{n-1} = b_{n-2} - b_{n-1}$ As a result the next equation is produced:

$$z_i = b_{i-1} - b_i \text{ , where } i = 0, 1, ..., n-1 \text{ and } b_{-1} = 0 \tag{2.3.1.4}$$

Thus B can be computed as follows:

$$B = \sum_{i=0}^{n-1} z_i 2^i \tag{2.3.1.5}$$

According to Booth's multiplication algorithm and using the previous equation (2.3.1.5) the next equation is formed:

$$P = A \cdot B = \sum_{i=0}^{n-1} A \cdot z_i \cdot 2^i = \sum_{i=0}^{n-1} A \cdot (b_{i-1} - b_i) \cdot 2^i \qquad (2.3.1.6)$$

It's easy to conclude that in each step of the algorithm ($i = 0, 1, ..., n-1$) the number A will be multiplied with one of the next set of numbers: $\{-1, 0, 1\}$. These are the three possible results of the equation (2.3.1.4). In the table 2.2, where PP is the partial product and initially $PP = 0$, the booth encoded digits are displayed as well as their meaning. The process of encoding begins from the right and continues to the left (Starting at $i = 0$ and at the beginning of each step $i = i + 1$).

$\mathbf{B_i}$	$\mathbf{B_{i-1}}$	**Encoded digits** $(\mathbf{B_{i-1} - B_i})$	**Meaning**
0	0	0	Add 0 to PP and arithmetic shift PP a single place to the right
0	1	1	Add A to PP and arithmetic shift PP a single place to the right
1	0	-1	Add $-A$ to PP and arithmetic shift PP a single place to the right
1	1	0	Add 0 to PP and arithmetic shift PP a single place to the right

Table 2.2: Booth encoding and Meaning

To clarify, an example is presented in table 2.3 .The two operands are $A = (0110)_2 = (6)_{10}$ and $B = (0111)_2 = (7)_{10}$. In each step $2^i\dot{X}$, where $X = -A$ or $X = A$ or $X = 0$ depending on the value of z_i, is added to the PP as concluded from the equation (2.3.1.6). The arithmetic shift to the right is used to avoid overflow and it's equivalent to sign-extension.

i	PP	z_i	**Comments**
0	0000	-1	Add $-2^0 \cdot A$
0	1010	-1	Arithmetic shift
1	11010	0	Add $2^1 \cdot 0$
1	11010	0	Arithmetic shift
2	111010	0	Add $2^2 \cdot 0$
2	111010	0	Arithmetic shift
3	1111010	1	Add $2^3 \cdot A$
3	0101010	1	Final result 42

Table 2.3: Example of multiplication of two numbers using the booth algorithm

Comparing the booth's multiplication algorithm to the conventional multiplication there are some great advantages:

1. The algorithm is the same for signed and unsigned numbers, thus it's independent from the sign of each number.

2. The conventional multiplication generates a partial product for every "1" of B whereas, the booth's multiplication algorithm generates a partial product for every change from "1" to "0" and vice versa, in the sequence of B. If there are consecutive "ones" ("1s") the encoded digit z_i is zero, so there is only an arithmetic shift to the right occurring to the PP. However, in some cases the booth's algorithm is less efficient than the conventional multiplication algorithm. This happens when more partial products are generated than in the conventional method.

3. Every encoded digit is independent from the previous digits, so the partial products can be generated immediately and then trough carry-save adders computed.

2.3.2 Modified Booth algorithm

The Modified booth algorithm is a variation of the previously presented booth's multiplication algorithm. The set of encoded digits is expanded to the set of numbers {-2, -1, 0, 1, 2}. Thus, the equation (2.3.1.5) is modified as follows:

$$
\begin{aligned}
B = \sum_{i=0}^{n-1} z_i 2^i &= \sum_{j=0}^{n/2-1} z_{2j} 2^{2j} + z_{2j+1} 2^{2n+1} \\
&= \sum_{j=0}^{n/2-1} (b_{2j-1} - b_{2j}) 2^{2j} + (b_{2j} - b_{2j+1}) 2^{2n+1} \\
&= \sum_{j=0}^{n/2-1} (-2b_{2j+1} + b_{2j} + b_{2j-1}) 4^j = \sum_{j=0}^{n/2-1} b_j^{MB} 4^j
\end{aligned}
\tag{2.3.2.1}
$$

In the table 2.4 the modified booth or radix-4 encoding is displayed.The bit b_{-1} is considered zero($b_{-1} = 0$).

To better understand the modified booth algorithm an example is presented. Two 8-bits numbers are multiplied, $A = (01010101)_2 = (85)_{10}$ and $B = (10000111)_2 = (-121)_{10}$. According to the radix-4 encoding, B is modified to $B = \overline{2}02\overline{1}$. This means that the numbers $2A = 010101010$, $2\overline{A} = 101010110$ and $\overline{A} = 110101011$ are needed for the execution of the algorithm. In the table 2.5 every step of the algorithm is displayed. In the case of the modified booth algorithm, an arithmetic shift by two places is required. Moreover the equation (2.3.2.1) shows, that each product is multiplied by $2^{2j} = 4^j$. This means that before $b_j^{MB} \cdot A$ is added to PP a two place shift to the left is occurred.

$\mathbf{b_{2j+1}}$	$\mathbf{b_{2j}}$	$\mathbf{b_{2j-1}}$	**Modified booth encoded digit** ($\mathbf{b_j^{MB}}$)
0	0	0	0
0	0	1	+1
0	1	0	+1
0	1	1	+2
1	0	0	−2
1	0	1	−1
1	1	0	−1
1	1	1	0

Table 2.4: Radix-4 encoding

j	PP	b_j^{MB}	**Comments**
0	000000000	−1	Add $-2^0 \cdot A$
0	110101011	−1	Arithmetic shift
1	11110101011	+2	Add $2^2 \cdot 2A$
1	01001010011	+2	Arithmetic shift
2	0001001010011	0	Add $2^4 \cdot 0$
2	0001001010011	0	Arithmetic shift
3	000001001010011	−2	Add -$2^6 \cdot 2A$
3	101011111010011	−2	Final result 10.285

Table 2.5: Example of multiplication of two numbers using the modified booth algorithm

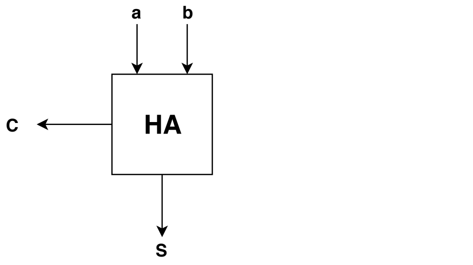
Modified booth algorithm shares the same advantages as the booth's multiplication algorithm. Furthermore radix-4 encoding makes sure,that the number of partial products is reduced, which leads to reduced delay and area of the circuits. However, the computation of $-2A$, $+2A$ and $-A$ increases the complexity of the circuit. In most applications the modified booth algorithm is used for the exact multiplication of two numbers.

2.4 Adders

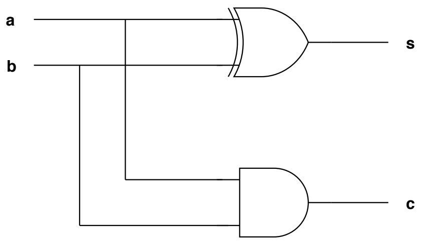
Addition is one of the most important operations of every arithmetic circuit. Their importance is connected with the fact that they are used in other operations, such as multiplication. Numerous types of adders and subtractors have been developed over the years. In this section the basic units of those adders and the adders themselves are being analyzed.

2.4.1 Half Adder (HA)

The simplest digital circuit of addition is a half adder (HA). HA receives two bits as input, it adds them, and it produces the sum (s) and the carry (c) as output. A half adder is displayed in the figure 2.1. s and c are given from the logical equations: $s = a \oplus b$, $c = a \cdot b$



(a) Half Adder(HA) as Black Box



(b) Half Adder(HA) Circuit

Figure 2.1: Half Adder(HA)

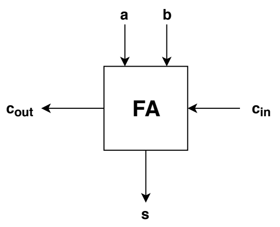
The truth table of the HA is presented in the table 2.6 .

a	b	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

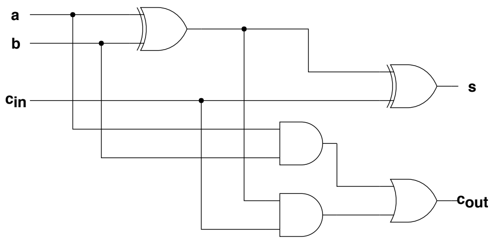
Table 2.6: Truth Table of HA

2.4.2 Full Adder(FA)

The full adder(FA) (Figure 2.2) is similar to the half adder with the difference that the input consist of three bits, a,b and c_{in}. The logical equations describing the FA are: $s = a \oplus b \oplus c_{in}$ and $c_{out} = (a \cdot b) + (a \cdot c_{in}) + (b \cdot c_{in})$ or $s = a \oplus b \oplus c_{in}$ and $c_{out} = (a \cdot b) + (c_{in} \cdot (a \oplus b))$. The second aforementioned set of logical equations is used for the generation of the circuit,which is displayed in figure 2.2b and consists of two half adders.



(a) Half Adder(HA) as Black Box



(b) Full Adder(FA) Circuit

Figure 2.2: Full Adder(FA)

The truth table of a FA is featured in table 2.7 .

a	**b**	$\mathbf{c_{in}}$	**s**	$\mathbf{c_{out}}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 2.7: Truth Table of FA

2.4.3 Serial Adder

The serial adder is able to add two n-bit numbers using just one full adder and one D flip-flop. It also needs two shift registers, one for the first number (a) and one for the second (b). The result is stored in the first register,where a was initially stored. In each clock cycle two bits of the numbers a and b are added, the first shift register gives away a bit of a and it stores a bit of the sum (s). In figure 2.3 a serial adder is displayed. The shift registers are not shown and i takes the values $\{0, 1, 2, .., n-1\}$.

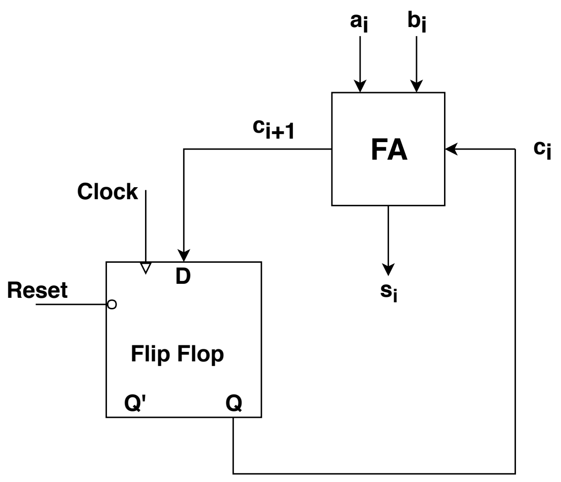


Figure 2.3: Serial Adder

2.4.4 Ripple-Carry Adder(RCA)

The ripple-carry adder(RCA) is also used for the addition of two n-bit numbers just as the serial adder. The difference is that the RCA consists of n full adders and doesn't use any D flip-flop. Figure 2.4 displays this type of adder.

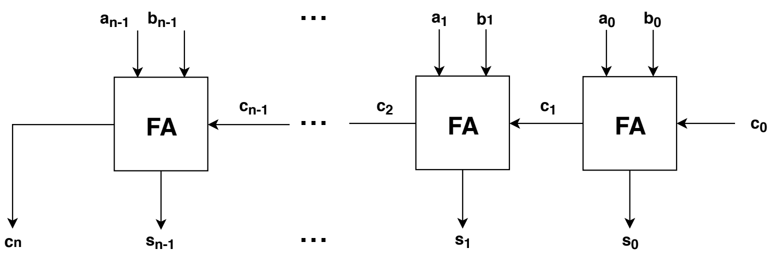


Figure 2.4: Ripple-Carry Adder

It is considered that c_0 is zero and c_n can be interpreted as the most significant bit (MSB) of the sum or as an overflow bit. The disadvantage of the RCA is, that the result is computed after all the carries are propagated. The function of RCA and serial adder is the same with the difference of the D flip-flop and the one FA. Therefore, the problem remains the same. The total delay of an RCA is $(2n + 1)\tau$. At the first level, the delay until the generation of c_1 is 3τ because of the xor-,and- and or-gate. At the next level the c_2 is generated after additional 2τ and in each following level 2τ are added to the total delay. Therefore, the $n - 1$ last levels produce a delay $2(n - 1)\tau$ and the first level 3τ. Thus, the total delay is $(2n + 1)\tau$

2.4.5 Carry-Save Adder(CSA)

The carry-save adder (CSA) is used for the addition of three or more n-bit numbers. As output it produces a number in carry-save notation as explained in subsection 2.2.3. In figure 2.5 a CSA, which has as input three n-bit numbers, is displayed. The three numbers are x, y and z. The numbers s (sum) and c (carry) are the output.

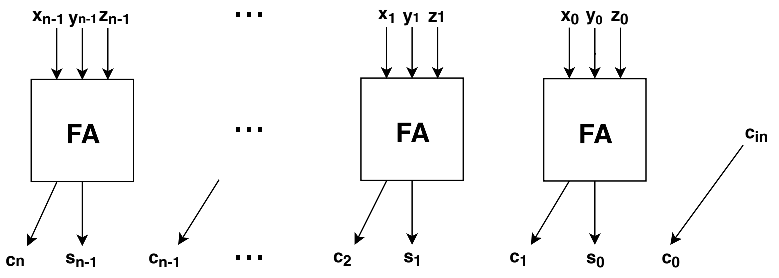


Figure 2.5: Carry-Save Adder

The carry-save adder has the advantage that the result is immediately ready in just one clock cycle. The downside is that the information is stored in two numbers and isn't in 2's complement notation. So if 2's complement is needed then the s (sum) and c (carry) are added with the help of a ripple-carry adder.

2.4.6 Carry-Look ahead Adder(CLA)

The aforementioned problem with the ripple-carry adder is that all the carries must be propagated in order to compute the whole result. Explicitly, for two n-bit numbers the total delay is $(2n+1)\tau$. To reduce the delay a widely used approach employs the principal of carry look-ahead, which calculates the carry signals in advance, based on the input signals. In figure 2.2b the circuit of a half adder is presented and the next signals are defined.

$$P_i = a_i \oplus b_i, \text{ where } i = 0, 1, 2, ..., n-1 \tag{2.4.6.1}$$

$$G_i = a_i \cdot b_i, \text{ where } i = 0, 1, 2, ..., n-1 \tag{2.4.6.2}$$

The output sum and carry can be defined as:

$$s_i = P_i \oplus c_i, \text{ where } i = 0, 1, 2, ..., n-1 \tag{2.4.6.3}$$

$$c_{i+1} = G_i + P_i \cdot c_i, \text{ where } i = 0, 1, 2, ..., n-1 \tag{2.4.6.4}$$

G_i is known as the carry Generate signal since a carry (c_{i+}) is generated whenever $G_i = 1$, regardless of the input carry (c_i). P_i is known as the carry propagate signal since whenever $P_i = 1$, the input carry is propagated to the output carry: $c_{i+1} = c_i$ (note that whenever $P_i = 1$,$G_i = 0$). The values of P_i and G_i can be computed immediately as they depend only on the input operand bits $(a_i$,$b_i)$. Computed values of all P_i's and G_i's are valid after one XOR-gate delay and AND-gate delay,respectively, after the operands are made valid. The Boolean expression of the carry outputs of various stages can be written as follows:

$$\begin{aligned}
c_1 &= G_0 + P_0 \cdot c_0 \\
c_2 &= G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot c_0 \\
c_3 &= G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot c_0 \\
c_4 &= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot c_0
\end{aligned}$$

The general Boolean expression of the carry output of the n^{th} stage is:

$$c_{n+1} = G_n + P_n \cdot G_{n-1} + P_n \cdot P_{n-1} \cdot G_{n-2} + ... + P_n \cdot P_{n-1} ... P_2 \cdot P_1 \cdot G_0 + P_{n-1} ... P_2 \cdot P_1 \cdot P_0 \cdot c_0 \tag{2.4.6.5}$$

In figure 2.6 a 4-bit carry-look ahead adder is displayed. The carry look ahead block is implemented in two-level circuits and this two-level implementation of the carry signals has a propagation delay of 2 gates (2τ).

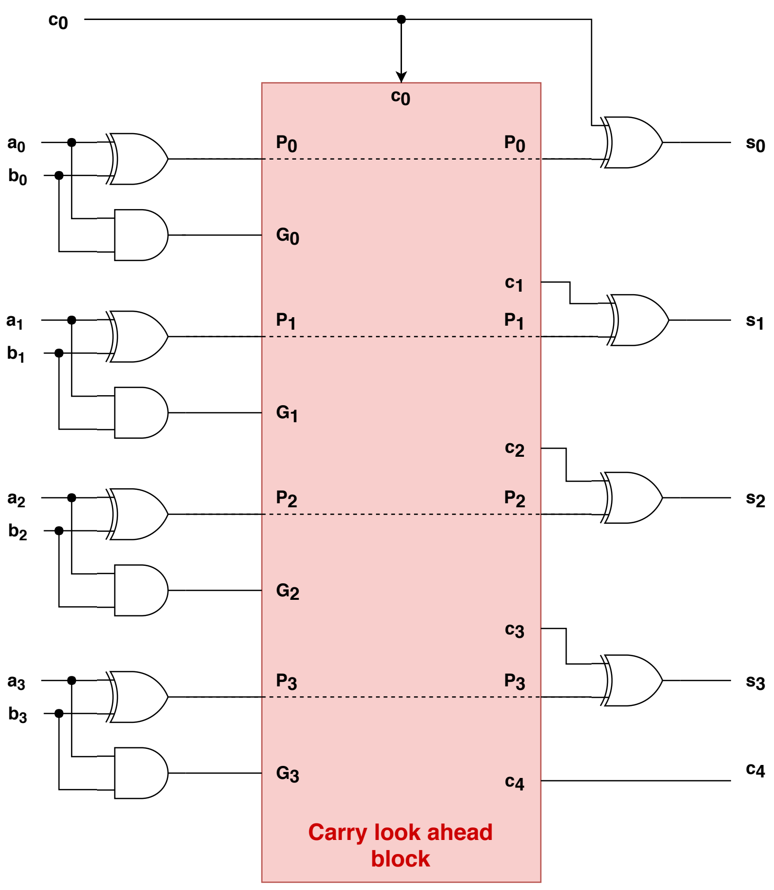


Figure 2.6: Carry-Look ahead Adder

The 4-bit carry-look ahead adder consists of three levels of logic:

- First level: It consists of four half adders and generates all the P and G signals. Output signals of this level will be valid after total delay of 1τ.
- Second level: The carry look ahead logic block generated the four carry signals (c_0,c_1,c_2 and c_3) as defined by the above Boolean expressions. These output carry signals will be valid after a total delay of 3τ.
- Third level: This level consists of four XOR-gates which generate the four sum signals (s_0, s_1, s_2 and s_3)

On one hand, the carry-look ahead adder generates the result after a total delay of 4τ. On the other hand, the ripple-carry adder performs the addition with total delay $(2n + 1)\tau$. In the case of a 4-bit RCA, it translates to 9τ. Thus there is a significant decrease of delay when using the carry-look ahead adder.

2.5 Multiplication

Multiplication is another very significant operation. In order to be performed some of the aforementioned adders must be used. In this section some types of multipliers will be analyzed and then it will be explained how the modified booth algorithm is implemented into circuits.

Let A,B be two unsigned m-,n-bit numbers, respectively. The operation of multiplication is analyzed as follows:

$$A \cdot B = A \cdot \sum_{i=0}^{n-1} 2^i b_i \tag{2.5.0.1}$$

In figure 2.7 the multiplication is performed.

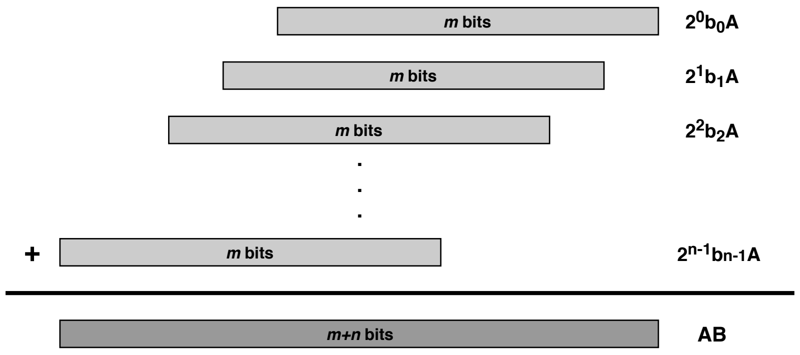


Figure 2.7: Multiplication of two unsigned number

The multiplication of two signed numbers in two's complement notation, is the same as the multiplication already described, with the sole difference that the MSB of each partial product has a negative weight.

The types of multipliers that will be analyzed can be applied for unsigned numbers. To extend to signed numbers in two's complement, small modifications are needed or some correction terms must be added. In general a parallel multiplier is based on the fact that the partial products can be simultaneously generated. For the unsigned numbers the result is expressed (P) as follows:

$$P = A \cdot B = \sum_{i=0}^{m-1} a_i 2^i \cdot \sum_{j=0}^{n-1} b_j 2^j = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} (a_i b_j) 2^{i+j} = \sum_{k=0}^{m+n-1} p_k 2^k \tag{2.5.0.2}$$

2.5.1 Parallel Multiplier with Carry Propagation

The parallel multiplier with carry propagation consists of AND-gates and full adders. The AND-gates are needed for the computation of the bits of the partial products ($a_i b_i$) and the FAs are used for the addition of the partial products. In this case the full adders form a ripple-carry adder. In figure 2.8 the basic component of the multiplier is displayed

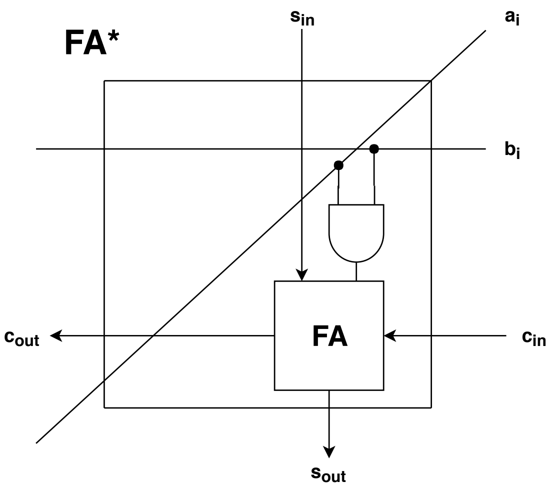


Figure 2.8: Circuit of FA*

The parallel multiplier with carry propagation consists of rows with these FA* units. In figure 2.9 an example of 4x4 parallel multiplier is displayed. The delay of the multiplier is dependent of the path that is created through the gray cells of FA*s. In this critical path the only AND-gate included is the one of the top right cell. That is because all the AND-gates produce their results simultaneously. In the critical path there are 3 $(n-1)$ FA* cells on the top horizontal line, 4 (m) FA* cells on the vertical line and 3 $(n-1)$ FA* cells on the bottom horizontal line (the common cells are included in the vertical line), so the total delay is $T = 10 \cdot T_{FA} + T_{AND}$. In general for A, B two m-,n-bit numbers, respectively, the total delay is $T = (2n+m-2)T_{FA} + T_{AND}$. In every other path chosen the total delay is equal to or less than T.

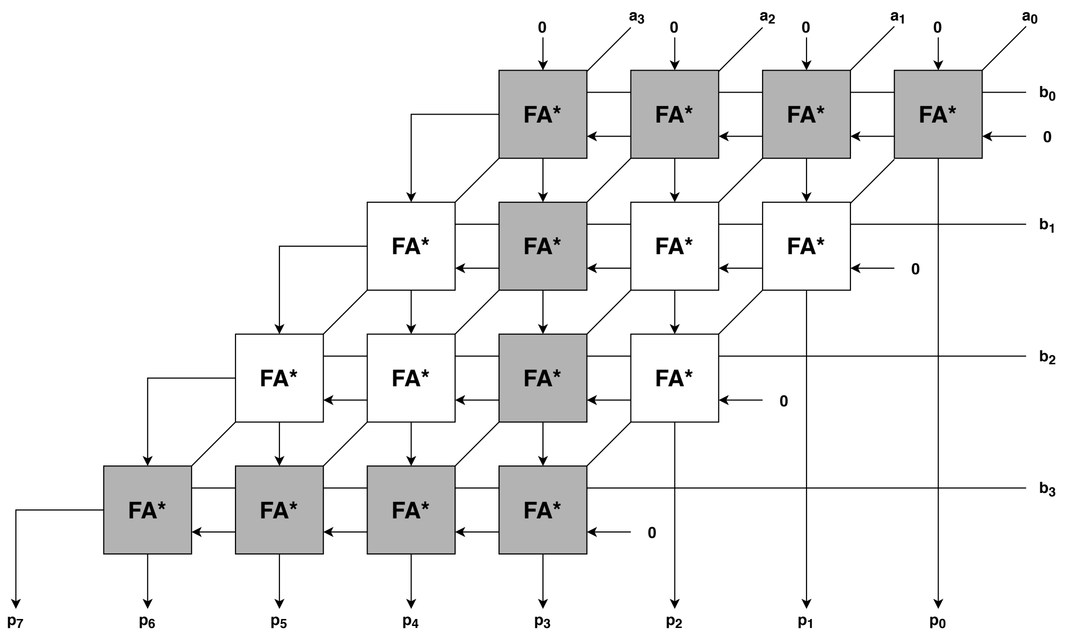


Figure 2.9: Parallel Multiplier with Carry Propagation

2.5.2 Parallel multiplier with Carry Save Adders

This type of multiplier is very similar to the previously described multiplier. The notable difference is that every row of full adders forms a carry-save adder instead of a ripple-carry adder. That means that the carry signals don't propagate inside the same level (row), but they are forwarded to the next level. In figure 2.10 the basic component of this type of multiplier is presented.

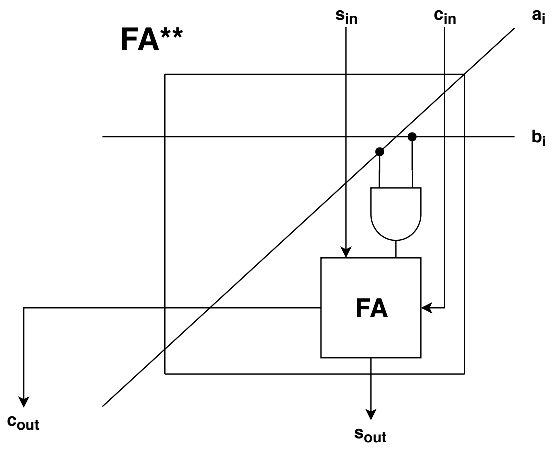


Figure 2.10: Circuit of FA**

The cell FA** is practically the same as the FA* the only difference being, that it receives the input carry signal from the previous level and gives the output carry signal to the next level. To better understand the differences, figure 2.11 features a 4x4 parallel multiplier using CS logic. The four most significant bits are given in CS notation so there is a 4-bit adder (last level), which modifies the result to a binary number. The total delay of this multiplier is closely depended on the type of adder used in the last level. When a ripple-carry adder is used the delay is $T = (n + m)T_{FA} + T_{AND}$. If a carry-look ahead adder is used, the delay is $T = (2 + \log_2 n)T_{FA} + T_{AND}$. In the example of the 4x4 multiplier the delays are equal, so the use of a carry-look ahead adder gives better results for greater n.

The way the parallel multiplier with carry-save adders is shown in figure 2.11 lacks efficiency. There is a better way to implement the 4x4 multiplier. Figure 2.12 exhibits this way and it is apparent that some FA** cells can be excluded. Instead of giving as input signals zeros some levels can be merged. With this implementation the total delay is $T = (m - 2 + n)T_{FA} + TAND$ as long as a carry ripple adder is used in the last level and the AND-gates are executed simultaneously. In case of a carry-look ahead adder in the last level, the delay is $T = (m - 2 + 2 + \log_2 n)T_{FA} + T_{AND} = (m + \log_2 n)T_{FA} + T_{AND}$

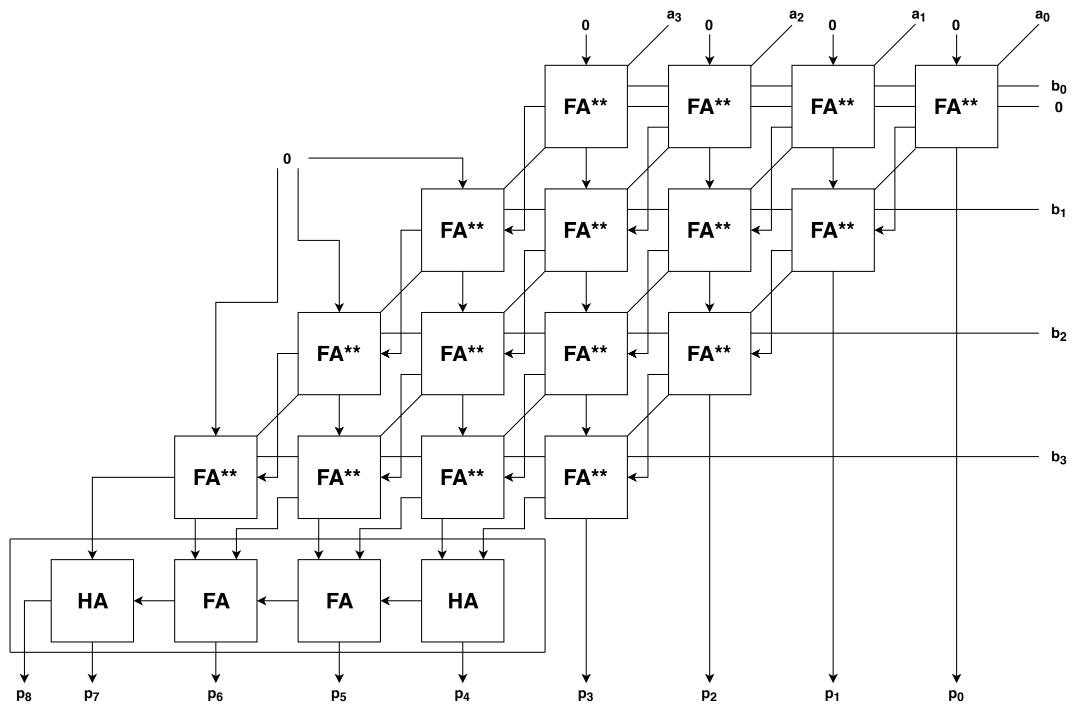


Figure 2.11: Parallel Multiplier using CS logic

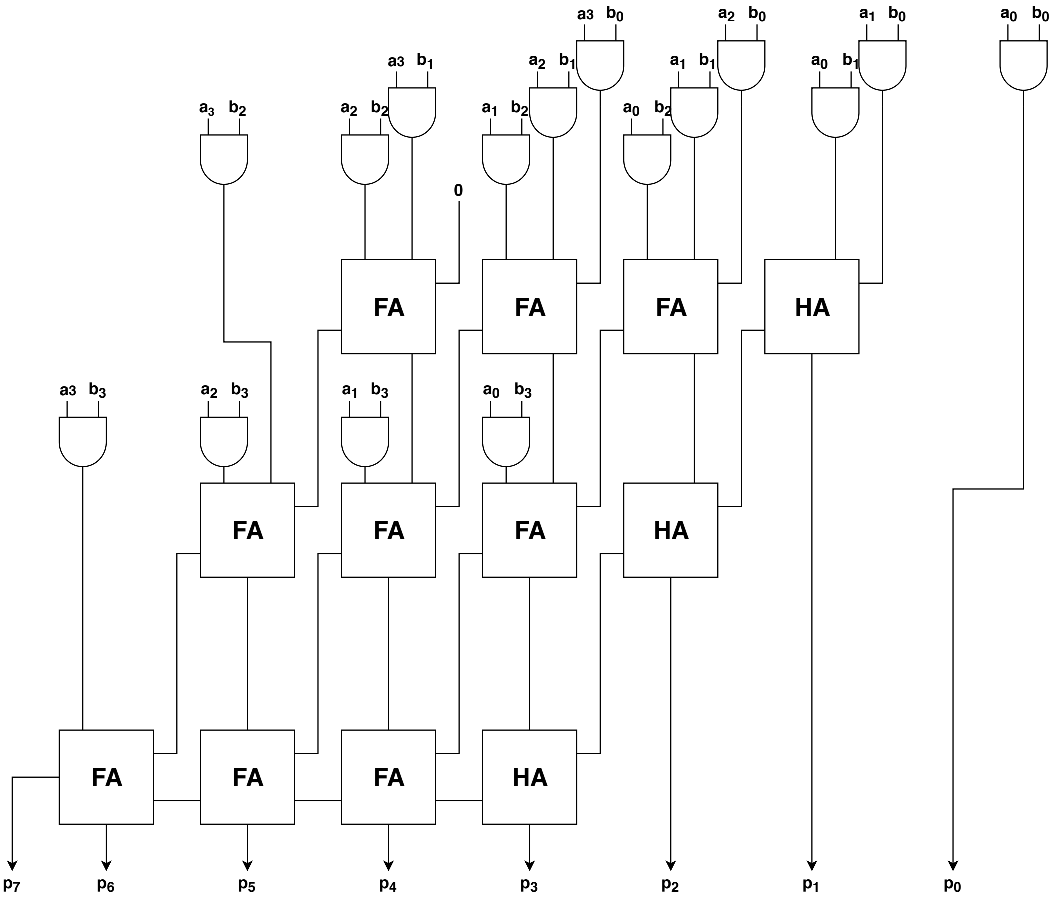


Figure 2.12: Parallel Multiplier using CS logic-Efficient Way

2.5.3 Wallace Tree Multiplier

The Wallace tree multiplier takes a different approach than the previously described multipliers. It uses the ability of the full adders to add three bits and give as output signals two bits (carry and sum). Thus the bits of the partial products are divided into groups of three and then again and again until the information is stored in carry-save notation. In figure 2.13 this division of a 8x8 Wallace tree multiplier is displayed. In this implementation only full adders are used. The delay is equal to the number of steps until the result is in carry-save notation. The total delay is just the delay of the steps plus the delay of the AND-gate and the delay of the carry-look ahead adder. So for this example $T = 7T_{FA} + T_{AND} + (2 + \log_2 15)T_{FA}$

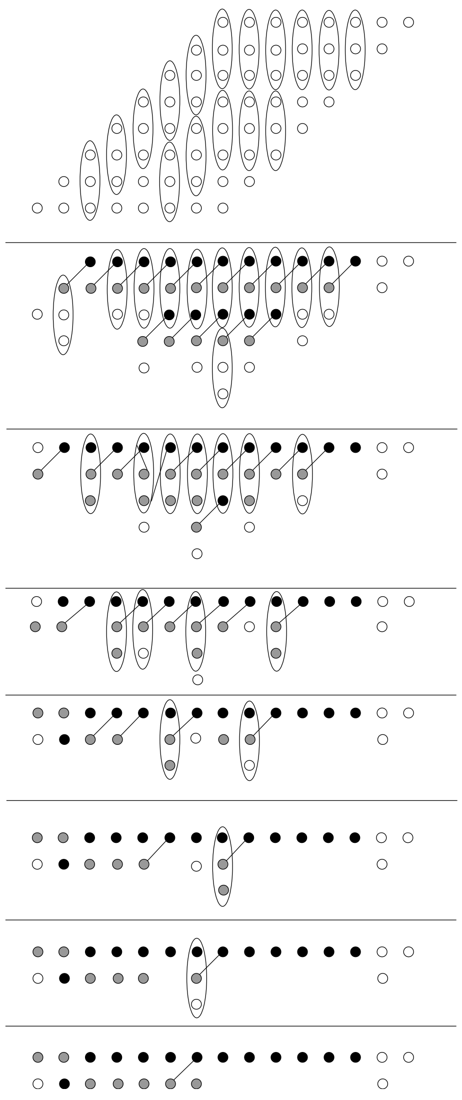


Figure 2.13: Bits division of a 8x8 Wallace Tree Multiplier

The steps of the figure 2.13 can be reduced if both full adders and half adders are used in the process. In the previous example the bits of the partial products were divided into groups of three as much it was allowed. Now by inserting of the half adders the algorithm is:

- The rows of partial products are divided into groups of three
- The result of each set of three rows is a set of two rows
- The resulting two rows consists of a row for the sum and a row for the carry.
- If there are remaining rows that cannot form a group of three, they are left alone.

In figure 2.14 an example of this method is featured. With the use of half adders and the alteration of the algorithm, the number of steps is reduced to 4 for an 8x8 multiplier. However, a 16x16 carry-look ahead adder is needed. So the total delay is $T = 4T_{FA} + T_{AND} + (2 + \log_2 16)T_{FA}$

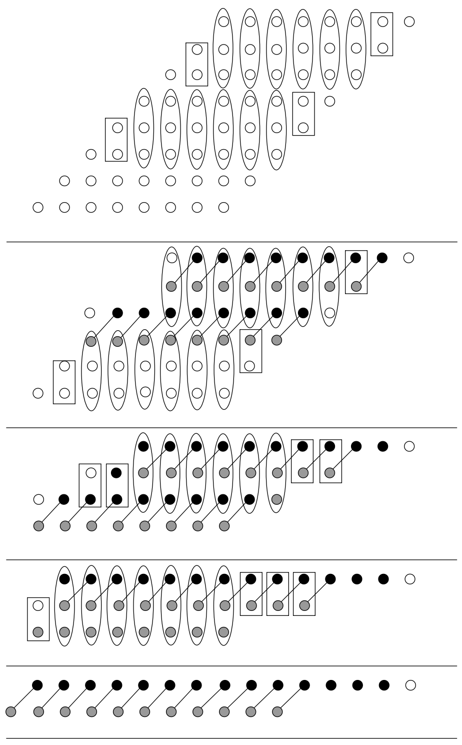


Figure 2.14: Rows division of a 8x8 Wallace Tree Multiplier

2.5.4 Modified Booth Algorithm using Wallace Tree

In subsection 2.3.2 the modified booth algorithm was described. In this subsection the implementation of the algorithm into circuit will be analyzed. Let A, B be two signed n-bit numbers in two's complement. As shown in the equation (2.3.2.1) $B = \sum_{j=0}^{n/2-1} b_j^{MB} 4^j$. In table 2.4 the radix-4 encoding was presented, but how are these encoded digits implemented into circuits? That's exactly what the table 2.8 shows.

Binary bits			Modified	Encoded Digit b_j^{MB}		
b_{2j+1}	b_{2j}	b_{2j-1}	Booth's Digit	sign = s_j	x1 = one_j	x2 = two_j
0	0	0	**0**	0	0	0
0	0	1	**+1**	0	1	0
0	1	0	**+1**	0	1	0
0	1	1	**+2**	0	0	1
1	0	0	**−2**	1	0	1
1	0	1	**−1**	1	1	0
1	1	0	**−1**	1	1	0
1	1	1	**0**	1 or 0	0	0

Table 2.8: Modified Booth Encoding Table

The logical equations that describe the table 2.8 are:

$$one_j = b_{2j-1} \oplus b_{2j} \tag{2.5.4.1}$$

$$two_j = (b_{2j+1} \oplus b_{2j}) \cdot \overline{one_j} \tag{2.5.4.2}$$

$$s_j = b_{2j+1} \tag{2.5.4.3}$$

The circuit that implements these logic equations is shown in figure 2.15.

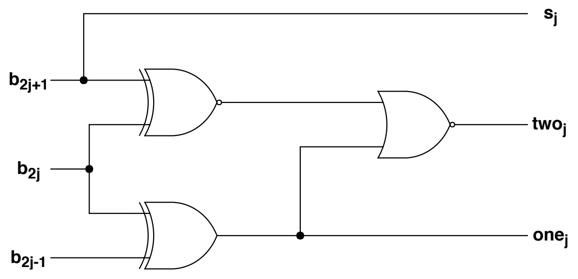


Figure 2.15: One,Two and s signals

So now that is clear how the modified booth encoding is implemented, the partial product generation will be explained.

$$P = A \cdot B = \sum_{j=0}^{n/2-1} A \cdot b_j^{MB} \cdot 2^{2j} = ct + \sum_{j=0}^{n/2-1} PP_j \cdot 2^{2j} \tag{2.5.4.4}$$

where ct is a correction term and PP_j the partial products

The correction term (ct) is needed for the correct implementation of the modified booth algorithm. In the figure 2.16 the steps of how the ct of an 8-bit multiplier is formed are shown.

- First step: The grey circles are the extra bit used for the case where A is multiplied by two ($two_j = 1$)

- Second step: The MSB of each partial product has a negative weight, so the relation $p + \bar{p} = 1$ is used. The grey circles are replaced with the black ($\bar{p}$) and a factor -1 is added ($-p = \bar{p} - 1$)

- Third step: The mathematical trick $(2 - 1) = 1$ is used, so $-1 = -2 + 1$.

- Fourth step: The subtractions created after the third step are performed

- Fifth step: Lastly, the blue circles that present the correction bits(n_j) are added. If the modified booth's digit has a negative weight ($b_j^{MB} = -1, -2$,i.e. $s_j = 1$), the correction bit is one in order to get to the 2's complement from the 1's complement. The correction bit is zero in case of positive weight. This will become more clear after the explanation of how the partial products are generated

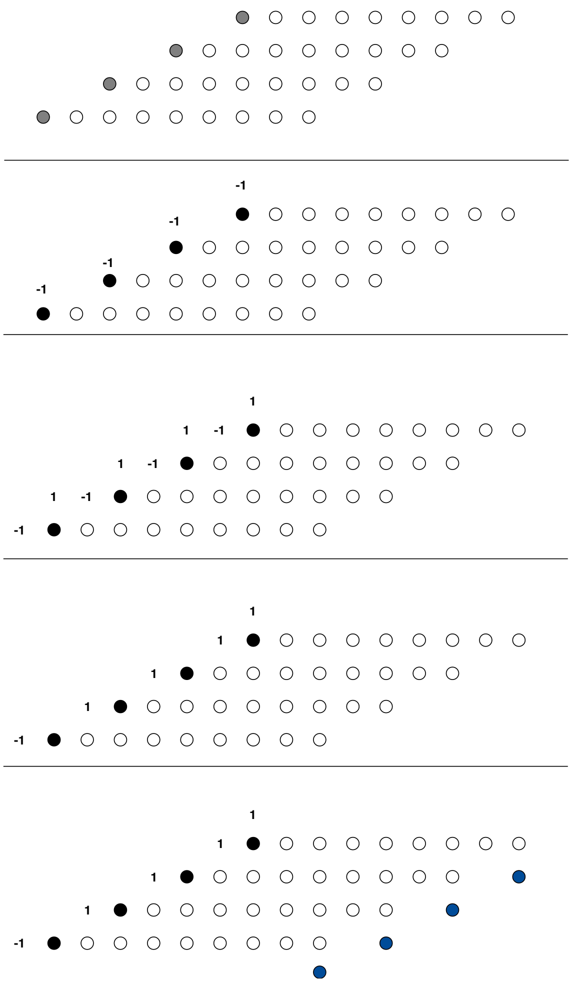


Figure 2.16: Partial products and Correction Term

Now negative weights are no longer an issue. The correction term is computed through the equation:

$$ct = \sum_{j=0}^{3}(\overline{p}_{j,8}2^{8+2j}) \ + \ \{-2^{15} + 2^{13} + 2^{11} + 2^{9} + 2^{8} + n_3 2^6 + n_2 2^4 + n_1 2^2 + n_0 2^0\}$$

where $p_{j,8}$ is the sign extension performed at the first step

Including the computation of the sign extension in the partial product generation, ct is given by the next equation, where The ct is in two's complement notation:

$$ct = \sum_{j=0}^{n/2-1}(n_j \cdot 2^{2j}) \ + \ 2^n(1 \ + \ \sum_{j=0}^{n/2-2}(2^{2j+1}) - 2^{n-1}) \tag{2.5.4.5}$$

The partial products are computed using the signals one_j, two_j and s_j via the next equations:

$$PP_j = \overline{p}_{j,n} \cdot 2^{n+2J} \ + \ \sum_{i=0}^{n-1}(p_{j,i} \cdot 2^{i+2j}) \tag{2.5.4.6}$$

$$p_{j,i} = ((a_i \ \oplus \ s_j) \cdot one_j) \ + \ ((a_{i-1} \ \oplus \ s_j) \cdot two_j) \tag{2.5.4.7}$$

$$p_{j,0} = ((a_0 \ \oplus \ s_j) \cdot one_j) \ + \ (s_j \cdot two_j) \tag{2.5.4.8}$$

$$\overline{p}_{j,n} = !(((a_{n-1} \ \oplus \ s_j) \cdot one_j) \ + \ ((a_{n-1} \ \oplus \ s_j) \cdot two_j)) \tag{2.5.4.9}$$

In the next example a partial product matrix with the correction term of a 8x8 multiplier using all the above is presented.

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
							1								
						1	$\overline{p}_{0,8}$	$p_{0,7}$	$p_{0,6}$	$p_{0,5}$	$p_{0,4}$	$p_{0,3}$	$p_{0,2}$	$p_{0,1}$	$p_{0,0}$
				1	$\overline{p}_{1,8}$	$p_{1,7}$	$p_{1,6}$	$p_{1,5}$	$p_{1,4}$	$p_{1,3}$	$p_{1,2}$	$p_{1,1}$	$p_{1,0}$		n_0
		1	$\overline{p}_{2,8}$	$p_{2,7}$	$p_{2,6}$	$p_{2,5}$	$p_{2,4}$	$p_{2,3}$	$p_{2,2}$	$p_{2,1}$	$p_{2,0}$		n_1		
1	$\overline{p}_{3,8}$	$p_{3,7}$	$p_{3,6}$	$p_{3,5}$	$p_{3,4}$	$p_{3,3}$	$p_{3,2}$	$p_{3,1}$	$p_{3,0}$		n_2				
									n_3						

In figure 2.17 the logical equation (2.5.4.7) is implemented and it is produced the featured circuit.

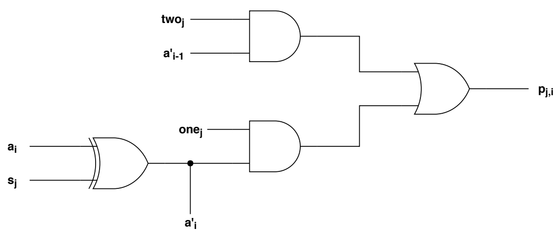


Figure 2.17: Partial Product Generator Unit

Lastly, in figure 2.18 all the components of the multiplier are being displayed. The multiplier consists of the partial products generators, the MB encoder, the CSA Wallace tree and the fast CLA adder.

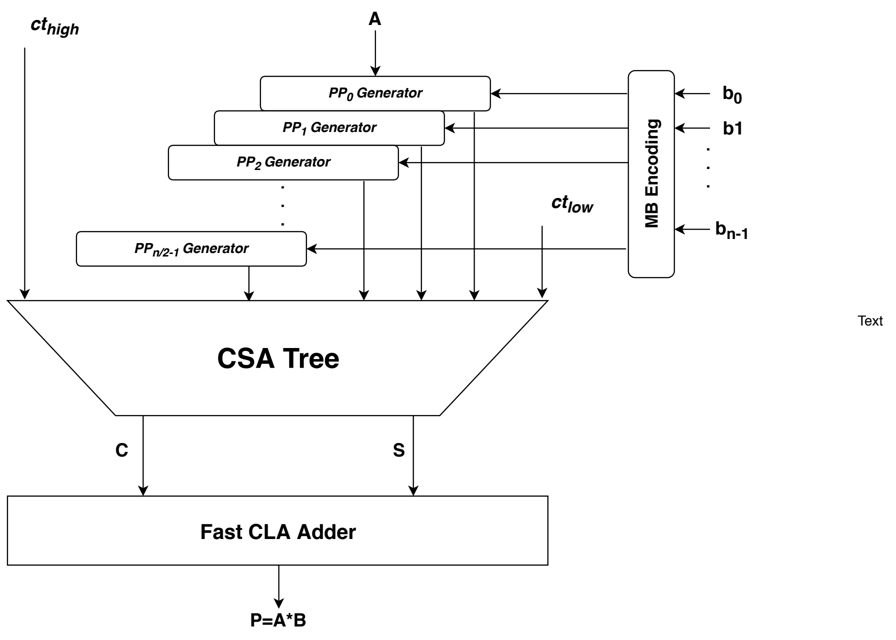


Figure 2.18: Modified Booth Multiplier

Chapter 3

Prior Work in the Field of Approximate Computing

3.1 Introduction

In the previous chapter we explained the basic theory that is needed to understand the approximation techniques developed in this diploma thesis. In this chapter the related, prior work that has been conducted in the field of approximate multipliers will be analyzed. The approximations that can be applied on a multiplier can be divided into two categories. The first category consists of approximations that are applied on the stage of accumulation (i.e. in the CSA Tree or other units that perform accumulation). Such approximation include approximate carry-save adders or approximate 4:2 compressors. In this diploma thesis the developed techniques are not of this category and therefore, no approximate adders and compressors will be analyzed. The second category consists of approximations that are applied in the stage of the partial products generation and the MB encoder. The stages were made clear in the previous chapter in figure 2.18. We further divide the approximation techniques of this category into four groups as it is shown in figure 3.1. In this chapter we will mention prior developed techniques belonging into these groups and some of them will be extended or combined with others to form the techniques that were developed during the research of this diploma thesis.

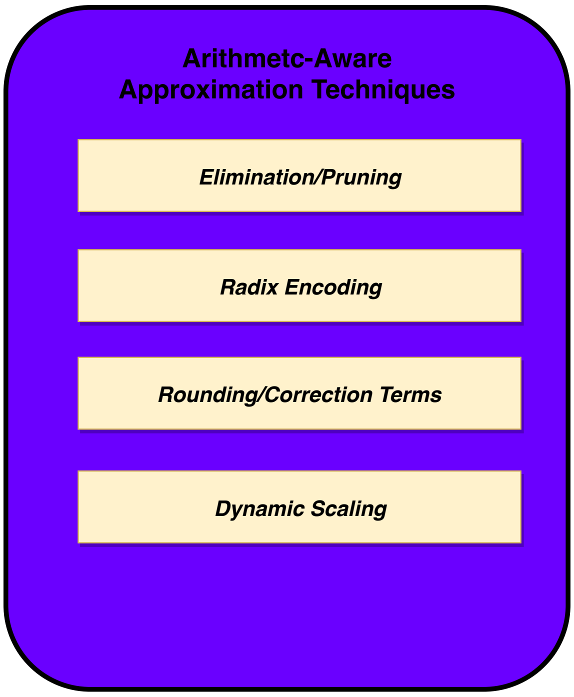


Figure 3.1: Categorization of the arithmetic-aware approximation techniques

3.2 Elimination/Pruning

In this section a new metric system and a technique based on elimination of some partial products will be presented.

3.2.1 Optimal Slope Ranking

A new design, approximate efficiency (AE), was introduced by Zhang *et al.* [32]. This new metric was used for the evaluation of the impact of each circuit node on its energy-delay-product (EDP). Taking advantage of this calculated AE a systematic approach named as optimal slope ranking (OSR) was produced. OSR prunes the nodes with a ranking list of AE. More specifically, the equations that present AE and EDP are:

$$EDP = f(Error) \tag{3.2.1.1}$$

Thus, the EDP of the circuits before and after pruning are shown as follows:

$$EDP_{before} = f(Error_{before}) \tag{3.2.1.2}$$

$$EDP_{after} = f(Error_{after}) \tag{3.2.1.3}$$

If two pruned nodes result in the same error, the node that leads to more reduction on EDP should be pruned first. AE is defined in equation (3.2.1.4)

$$AE = \frac{EDP_{after} - EDP_{before}}{Error_{after} - Error_{before}} = \frac{\Delta EDP}{\Delta Error} \tag{3.2.1.4}$$

At last the EDP reduction at error threshold can be expressed as follows:

$$\Delta EDP_{tot} = \sum \Delta EDP_{node} = \sum AE_{node} \Delta Error_{node} \tag{3.2.1.5}$$

3.2.2 Partial Product Perforation (PPP)

The partial product perforation method was proposed by Zervakis *et al.* [30]. In this technique, partial products are omitted, and thus, the depth of the accumulation tree is reduced. One possible downside is that error is increased exponentially while more partial products are excluded.

Let A,B be two n-bit numbers. The result of their multiplication is obtained after summing all the partial products Ab_i , where b_i is the ith bit of B. Thus

$$A \times B = \sum_{i=0}^{n-1} Ab_i 2^i,\ b_i \in \{0, 1\}. \tag{3.2.2.1}$$

The partial product perforation method omits the generation of k successive partial products starting from the jth one. The perforated partial products are not included in the accumulation tree, and hence n full adders can be eliminated. Applying the partial product perforation on the multiplication, the approximate result is produced:

$$A \times B|_{j,k} = \sum_{\substack{i=0,\\ i \notin [j,j+k)}}^{n-1} Ab_i 2^i,\ b_i \in \{0, 1\}. \tag{3.2.2.2}$$

It should be noted that $j \in [0, n-1]$ and $k \in [1, min(n-j, n-1)]$.

Respectively, when modified booth encoding (MBE) is used for partial product generation, the result of the approximate multiplication can be expressed:

$$A \times B|_{j,k} = \sum_{\substack{i=0,\\ i \notin [j,j+k)}}^{n/2-1} Ab_i^{MB} 4^i,\ b_i^{MB} \in \{0, \pm 1, \pm 2\}. \tag{3.2.2.3}$$

In the figure 3.2 an example is presented. The red circles are the bits that are not inserted in the accumulation tree. The equation (3.2.2.3) is used because of the modified booth encoding. For the example it was selected $j = 0$ and $k = 2$. The black circles represent the sign factors and the grey circles the inverted MSBs of the partial products.

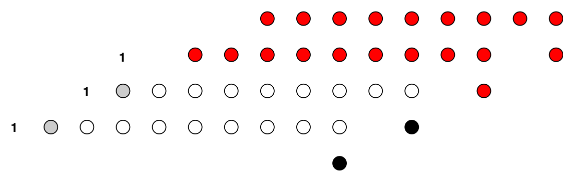


Figure 3.2: Applying PPP on a 8 × 8 multiplier using MBE

3.3 Radix Encoding

A wide range of approximations are based on radix encoding. These techniques lead to partial product reduction and therefore to energy efficient designs. The difference from the previous group (Elimination/Pruning) is that the partial products are not just eliminated but through approximations used on the stage of the encoding the number of the partial products generated is reduced. In this section such approximations techniques will be presented.

3.3.1 Approximate Hybrid High Radix Multipliers

High radix encodings lead to smaller accumulation trees. However high radix encodings require complex encoding and partial product genration trees. So the hybrid high radix encoding proposed by [13] and the performed approximations simplify the complexity of the aforementioned circuits.

Hybrid High Radix Encoding

Consider two n-bit numbers A and B. B is divided into two parts: The MSB part of $n-k$ bits and the LSB part of k bits. The configuration parameter, $k \geq 4$, is an even number, namely, $k = 2m : m \in \mathbb{Z}$, with $m \geq 2$. The MSB part is encoded using the radix-4(modified Booth) encoding. On the other hand the LSB part is encoded with the high radix-2^k encoding. The table 2.8 of the previous chapter features the radix-4 encoding. So B can be expressed as follows:

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i = \sum_{\substack{j=k/2,\\ k\geq 4}}^{n/2-2} y_j^{R4} 4^j + y_0^{R2^k} \tag{3.3.1.1}$$

where

$$y_j^{R4} = -2b_{2j+1} + b_{2j}b_{2j-1} \tag{3.3.1.2}$$

and

$$y_0^{R2^k} = -2^{k-1}b_{k-1} + 2^{k-2}b_{k-2} + \ldots + 2^1 b_1 + b_0 \tag{3.3.1.3}$$

As already mentioned the radix-4 encoding includes $(n-k)/2$ digits $y_j^{R4} \in \{0, \pm1, \pm2\}$, while $y_0^{R2^k} \in \{0, \pm1, \pm2, \pm3, \ldots, \pm2^{k-1} - 1, \pm2^{k-1}\}$ corresponds to the radix-2^k encoding. Overall, B is encoded with $(n-k)/2 + 1$ digits.

This high radix encoding is characterized by increased complexity, due to the high radix values of $y_0^{R2^k}$ that are not power of two, and thus, an approximate version was proposed by the authors of [13]. The MSB part will still be performed accurately using the radix-4 encoding. However, the LSB part will be approximately encoded. In particular all the values that are not power of two and the $k-4$ smallest powers of two will be rounded to the nearest of the four largest powers of two or zero, so that the sum of all the values of the approximate digit $\hat{y}_0^{R2^k}$ is zero. Therefore B is approximated as follows:

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i = \sum_{\substack{j=k/2,\\ k\geq 4}}^{n/2-2} y_j^{R4} 4^j + \hat{y}_0^{R2^k} \tag{3.3.1.4}$$

where

$$y_j^{R4} \in \{0, \pm 1, \pm 2\} \tag{3.3.1.5}$$

and

$$\hat{y}_0^{R2^k} \in \{0, \pm 2^{k-4}, \pm 2^{k-3}, \pm 2^{k-2}, \pm 2^{k-1}\} \tag{3.3.1.6}$$

From table 3.1 we extract the signals:

$$sign_j = b_{2j+1} \tag{3.3.1.7}$$

$$\times 1_j = b_{2j-1} \oplus b_{2j} \tag{3.3.1.8}$$

$$\times 2_j = (b_{2j+1} \oplus b_{2j}) \cdot \overline{(b_{2j-1} \oplus b_{2j})} = (b_{2j+1} \oplus b_{2j}) \cdot \overline{\times 1_j} \tag{3.3.1.9}$$

Input			**R4 Digit**	**Output**		
b_{2j+1}	b_{2j}	$b_{2j\text{-}1}$	y_j^{R4}	$sign_j$	$\times 2_j$	$\times 1_j$
0	0	0	0	0	0	0
0	0	1	1	0	0	**1**
0	1	0	1	0	0	**1**
0	1	1	2	0	**1**	0
1	0	0	-2	**1**	**1**	0
1	0	1	-1	**1**	0	**1**
1	1	0	-1	**1**	0	**1**
1	1	1	0	**1**	0	0

Table 3.1: Accurate Radix-4 Encoding Table

Table 3.2 presents the approximate radix-2^k encoding. As for the encoded signals that define the radix-2^k digit $\hat{y}_0^{R2^k}$, their Boolean equations are:

$$sign = b_{k-1} \tag{3.3.1.10}$$

$$\times 2^{k-4} = (\bar{b}_{k-2} \cdot \bar{b}_{k-3} \cdot \bar{b}_{k-4} + b_{k-2} \cdot b_{k-3} \cdot b_{k-4}) \cdot (b_{k-4} \oplus b_{k-5}) \tag{3.3.1.11}$$

$$\times 2^{k-3} = \bar{b}_{k-1} \cdot \bar{b}_{k-2} \cdot (\bar{b}_{k-3} \cdot b_{k-4} \cdot b_{k-5} + b_{k-3} \cdot \bar{b}_{k-4}) + b_{k-1} \cdot b_{k-2} \cdot (b_{k-3} \cdot \bar{b}_{k-4} \cdot \bar{b}_{k-5} + \bar{b}_{k-3} \cdot b_{k-4}) \tag{3.3.1.12}$$

$$\times 2^{k-2} = \bar{b}_{k-2} \cdot b_{k-3} \cdot (b_{k-1} + b_{k-4}) + b_{k-2} \cdot \bar{b}_{k-3} \cdot (\bar{b}_{k-1} + \bar{b}_{k-4}) \tag{3.3.1.13}$$

$$\times 2^{k-1} = \bar{b}_{k-1} \cdot b_{k-2} \cdot b_{k-3} + b_{k-1} \cdot \bar{b}_{k-2} \cdot \bar{b}_{k-3}. \tag{3.3.1.14}$$

R2^k Digit		**Output**				
$y_0^{R2^k}$	$\hat{y}_0^{R2^k}$	*sign*	$\times 2^{k\text{-}1}$	$\times 2^{k\text{-}2}$	$\times 2^{k\text{-}3}$	$\times 2^{k\text{-}4}$
$[0,\ 2^{k\text{-}5})$	0	0	0	0	0	0
$[2^{k\text{-}5},\ 2^{k\text{-}4}{+}2^{k\text{-}5})$	$2^{k\text{-}4}$	0	0	0	0	**1**
$[2^{k\text{-}4}{+}2^{k\text{-}5},\ 2^{k\text{-}3}{+}2^{k\text{-}4})$	$2^{k\text{-}3}$	0	0	0	**1**	0
$[2^{k\text{-}3}{+}2^{k\text{-}4},\ 2^{k\text{-}2}{+}2^{k\text{-}3})$	$2^{k\text{-}2}$	0	0	**1**	0	0
$[2^{k\text{-}2}{+}2^{k\text{-}3},\ 2^{k\text{-}1})$	$2^{k\text{-}1}$	0	**1**	0	0	0
$[\text{-}2^{k\text{-}1},\ \text{-}2^{k\text{-}2}\text{-}2^{k\text{-}3})$	$\text{-}2^{k\text{-}1}$	**1**	**1**	0	0	0
$[\text{-}2^{k\text{-}2}\text{-}2^{k\text{-}3},\ \text{-}2^{k\text{-}3}\text{-}2^{k\text{-}4})$	$\text{-}2^{k\text{-}2}$	**1**	0	**1**	0	0
$[\text{-}2^{k\text{-}3}\text{-}2^{k\text{-}4},\ \text{-}2^{k\text{-}4}\text{-}2^{k\text{-}5})$	$\text{-}2^{k\text{-}3}$	**1**	0	0	**1**	0
$[\text{-}2^{k\text{-}4}\text{-}2^{k\text{-}5},\ \text{-}2^{k\text{-}5})$	$\text{-}2^{k\text{-}4}$	**1**	0	0	0	**1**
$[\text{-}2^{k\text{-}5},\ 0)$	0	**1**	0	0	0	0

Table 3.2: Approximate Radix-2^k Encoding Table

In figure 3.3 an example is displayed, where the approximate radix-256 is applied on 16×16 multiplier. The circles represent the partial products from radix-4 encoding and the squares the partial product from the radix-256 encoding. Moreover the white circles and squares are the inverted MSBs of the partial products. There has been a sign extension seven times because the largest value of y_0^{R256} is $128 = 2^7$. Lastly, the black circles and squares stand for the sign factors.

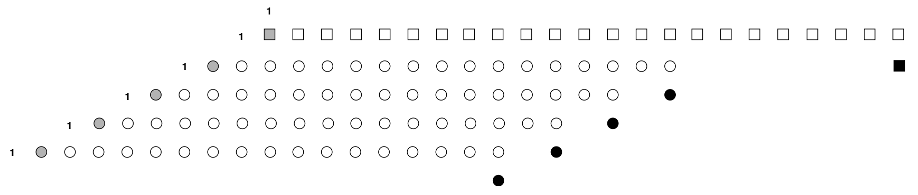


Figure 3.3: Applying radix-4 and radix-256 on a 16×16 multiplier

3.3.2 Other worth mentioning techniques

There are more techniques developed, that can't be presented all of them extensively as the hybrid high radix encoding was in the previous subsection. However there are more techniques that are worth mentioning.

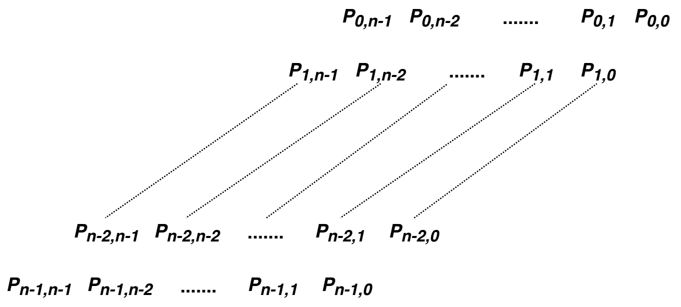
Such a case is an approximate radix-8 booth multipliers, that uses an approximate adder for producing $\pm 3A$, as suggested by Jiang *et al.* [8]. Liu *et al.* [18] designed approximate modified Booth encoders by transforming its K-Map, and also combined them with an approximate compressor. Recently Venkatachalam *et al.*[27] proposed an idea to alter the K-Map of partial product generation in order to simplify and make more parallel the circuit needed. The number of partial products remained the same. Through further approximation using OR-gates they managed to avoid the extra line of correction terms. Eventually the accumulation was implemented through OR-gates for the least significant part and exact full adders, half adders and 4-2 compressors for the most significant part.

3.4 Rounding/Correction Terms

The techniques belonging to this group reduce the number of the bits, that are inserted into the accumulation tree. They target usually the least significant bits of each partial product.

3.4.1 Truncation method

The truncation method was developed by Schulte and Swartzlander [26]. Consider two n-bit numbers A and B. The multiplication matrix is shown in figure 3.4. The truncation method or, as they named it, truncated multiplication uses only the $n+k$ most significant columns of the matrix.



$p_{0,n-1}$ $p_{0,n-2}$ .... $p_{0,1}$ $p_{0,0}$
$p_{1,n-1}$ $p_{1,n-2}$ .... $p_{1,1}$ $p_{1,0}$
$p_{n-2,n-1}$ $p_{n-2,n-2}$ .... $p_{n-2,1}$ $p_{n-2,0}$
$p_{n-1,n-1}$ $p_{n-1,n-2}$ .... $p_{n-1,1}$ $p_{n-1,0}$

Figure 3.4: Multiplication Matrix of two n-bit numbers

Truncated multiplication leads to two sources of error: reduction error and rounding error. Reduction error occurs because the $n-k$ least significant columns of the multiplication matrix are excluded from the computation of the product. Rounding error occurs because the product is rounded to n bits. To compensate for these two sources of error a correction constant is added to the $n+k$ most significant columns of the multiplication matrix. Figure 3.5 features a truncated multiplication matrix, where $C_{n+k-1}, C_{n+k-2}, \ldots, C_1, C_0$ is the correction constant. The value of the computed new product $\hat{P}$ is:

$$\hat{P} = P + E_{reduct} + E_{round} + C \tag{3.4.1.1}$$

where P is the exact product, E_{reduct} the reduction error, E_{round} the rounding error and C the correction constant. To minimize the the average error of the truncated multiplication, the correction is selected to be as close as possible to the additive inverse of the expected value of the sum of the two sources of error. Since the reduction and rounding error are both negative, the correction constant should be positive. For the estimation of the expected value of the reduction error it is considered that the probability of any bit a_i or b_j being one is 0.5. The positional weight of partial product bit $p_{j,i}$ is $2^{-2n+j+i}$ and $p_{j,i}$ is equal to 1 if and only if a_i and b_j are both one. Therefore, the expected value of $p_{j,i}$ is:

$$Expect[p_{j,i}] = \mu_{j,i} = -\frac{2^{-2n+j+i}}{4} \tag{3.4.1.2}$$

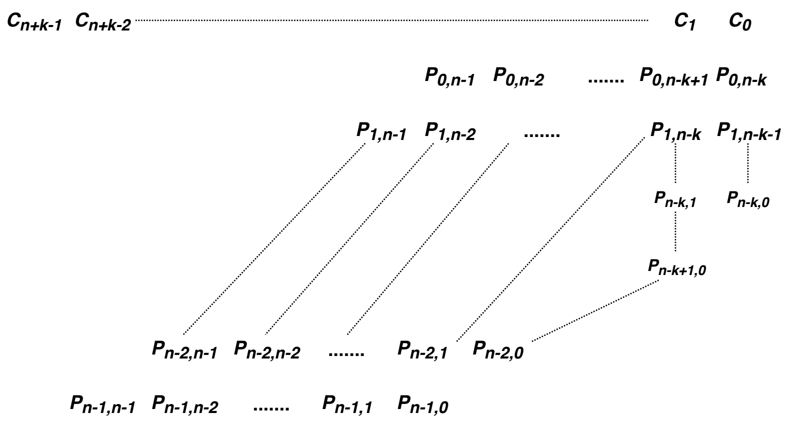

C_{n+k-1} C_{n+k-2} C_1 C_0
$P_{0,n-1}$ $P_{0,n-2}$ .... $P_{0,n-k+1}$ $P_{0,n-k}$
$P_{1,n-1}$ $P_{1,n-2}$ .... $P_{1,n-k}$ $P_{1,n-k-1}$
$P_{n-k,1}$ $P_{n-k,0}$
$P_{n-k+1,0}$
$P_{n-2,n-1}$ $P_{n-2,n-2}$ .... $P_{n-2,1}$ $P_{n-2,0}$
$P_{n-1,n-1}$ $P_{n-1,n-2}$ .... $P_{n-1,1}$ $P_{n-1,0}$

Figure 3.5: Truncated Multiplication Matrix of two n-bit numbers

Since all partial products bits in column q indices $j + i = q$, and there are $q + 1$ partial products bits in column q, the expected value of the reduction error is:

$$E_{reduct} = -\frac{1}{4}\sum_{q=0}^{n-k-1}((q+1)2^{-2n+q}) \tag{3.4.1.3}$$

To estimate the expected value of the rounding error, it is assumed that the probability of any product bit r_i being one is 0.5. If the products bits p_{n-k} to p_{n-1} are truncated, the expected value of the rounding error is:

$$E_{round} = -\frac{1}{2}\sum_{q=n-k}^{n-k-1}2^{-2n+q} = -2^{n-1}(1-2^{-k}) \tag{3.4.1.4}$$

The expected value of the total error is the sum of the aforementioned errors

$$E_{total} == -\frac{1}{4}\sum_{q=0}^{n-k-1}((q+1)2^{-2n+q}) - 2^{n-1}(1-2^{-k}) \tag{3.4.1.5}$$

The correction constant consists of $n + k$ bits. Thus, it is computed by the next equation:

$$C = -\frac{round(2^{n+k}\cdot E_{total})}{2^{n+k}} \tag{3.4.1.6}$$

where $round(x)$ indicates x is rounded to the nearest integer. Nevertheless this correction constant produces a non-zero component. An improvement to this method was introduced by King and Schwartzland [12] ,who came up with an idea to change between different correction constants in order to get rid of the non-zero component. A possible negative effect is the introduced delay due to the "decision" which correction constant should be used.

Another worth mentioning technique, which uses truncation, was developed by Zhang and He [31]. They proposed an approximate multiplier, where the partial product matrix is divided into two parts: a main part (MP) for an accurate accumulation and a truncated part (TP). The TP is further partitioned into TP_{major} and TP_{minor}. TP_{major} is the most significant column of TP and TP_{minor} is the part that is being estimated through a probabilistic approach.

3.4.2 Hybrid Partial Product Perforation-Rounding

The authors in [14] proposed a hybrid technique combining the partial product perforation of [30] with a truncation/rounding method.

Partial Product Perforation

The partial product perforation (PPP) technique was presented in a previous section, that featured the group of techniques about elimination or pruning. In [14] they used it as follows. Let A and B be two n-bit 2's complement binary numbers. They dismissed the generation of k successive partial products starting from the least significant ones. Therefore the k least significant modified Booth digits are not generated; namely, the $2k$ LSBs of B(including b_{-1}) are discarded. Thus, the product $A \times B$ is calculated approximately by the next equation:

$$A \times B|_k = \sum_{j=k}^{n/2-1} A \cdot b_j^{MB} \cdot 4^j \tag{3.4.2.1}$$

Partial Product Rounding

The idea is to discard the $m-1$ LSBs of A, and add a_{m-1} with the most significant remaing part (A_m), as follows:

$$A_m + a_{m-1} = \langle a_{n-1}, a_{n-2} \ldots a_m \rangle_{2'S} + a_{m-1} \tag{3.4.2.2}$$

The truncation of $m-1$ LSBs would lead to significant errors in the calculation. Therefore, the last remaining LSB (a_{m-1}) is added to A_m. The partial products with modified booth encoding are produced combining two cases.

In the case of $a_{m-1} = 0$, the inexact partial products ($\tilde{P}_j$) are calculated by
$\tilde{P}_j = (A_m + 0) \cdot b_j^{MB} = A_m \cdot b_j^{MB}$

In the case of $a_{m-1} = 1$, and using the relation $A_m + 1 = -\overline{A}_m$, the inexact partial products ($\tilde{P}_j$) are calculated by
$\tilde{P}_j = (A_m + 1) \cdot b_j^{MB} = (-\overline{A}_m) \cdot b_j^{MB} = \overline{A}_m \cdot (-b_j^{MB})$, where
$(-b_j^{MB}) = (-1)^{\overline{s}_j} \cdot (2 \cdot two_j + one_j)$. Using the relation $A_m^* = A_m \oplus a_{m-1}$ to form A_m or $\overline{A}_m$ the two cases are combined, Similarly $s_j^* = s_j \oplus a_{m-1}$ is used to form either b_j^{MB} or $-b_j^{MB}$. Therefore, the partial products are computed by $\tilde{P}_j = A_m^* \cdot b_j^{MB^*}$, where $b_j^{MB^*} = (-1)^{s_j^*} \cdot (2 \cdot two_j + one_j)$.

Partial product perforation and partial product rounding are combined to form the proposed technique called hybrid partial product perforation-rounding. The next equation describes this technique:

$$A \times B = \sum_{j=k}^{n/2-1} \tilde{P}_j \cdot 4^j = \sum_{j=k}^{n/2-1} A_m^* \cdot b_j^{MB^*} \cdot 4^j, \text{ where } k \in [0, n/2-1) \text{ and } m \in [0, n-1) \tag{3.4.2.3}$$

The correction term (ct), which was presented in the section "Modified Booth Multiplier" of chapter 2 , includes the '1's and the slightly different sign factors:

$$c_j^* = s_j^* \cdot (one_j + two_j)$$

In the example displayed in figure 3.6 $k = 3$ and $m = 4$ are chosen.

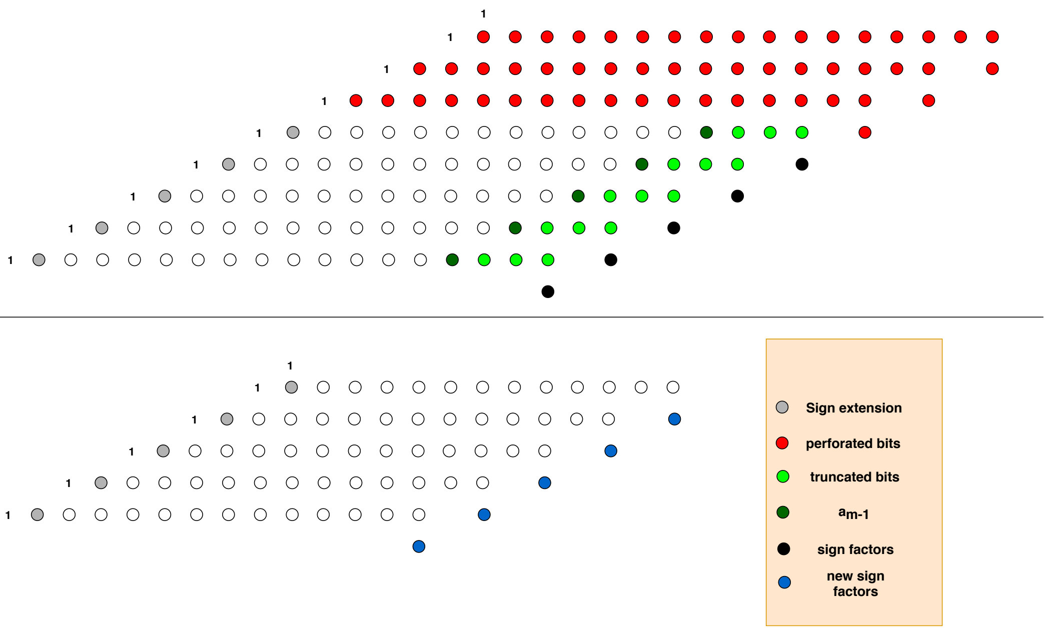


Figure 3.6: PR$|_{3,4}$

3.5 Dynamic Scaling

In this diploma thesis the techniques, that were developed, have more to do with the previously analyzed groups. Therefore, in this section two important techniques of this group will be mentioned.

The first was developed by Narayanamoorthy *et al.* [22]. They proposed an approximate multiplier that uses m sequential bits in an operand as segmented inputs. In [22] the authors introduce the static segment method (SSM) in order to fix the start point of a segment and by doing so they achieve scalable accuracy. A limitation of the above technique is the difficulty in scaling to higher inputs widths. As a result its benefits reduce respectably as the input size grows. Base on the fact that there more important bits than others Hashemi *et al.* [6] proposed to carefully select the range of bits for each operand of the multipliers. The bit selection is based on the use of two leading one detector(LOD) circuit blocks to locate the most significant 'one' in each operand (for example position t). Then depending on the accuracy required the following $k-2$ consecutive bits are selected. As for the rest an approximation is made using the expected value of a uniform distribution for values between [0,$2^{t-k+2}-1$], which is 2^{t-k+1}. Therefore, k bits are used and the rest are truncated. A general example of the approximation process is presented in figure 3.7.

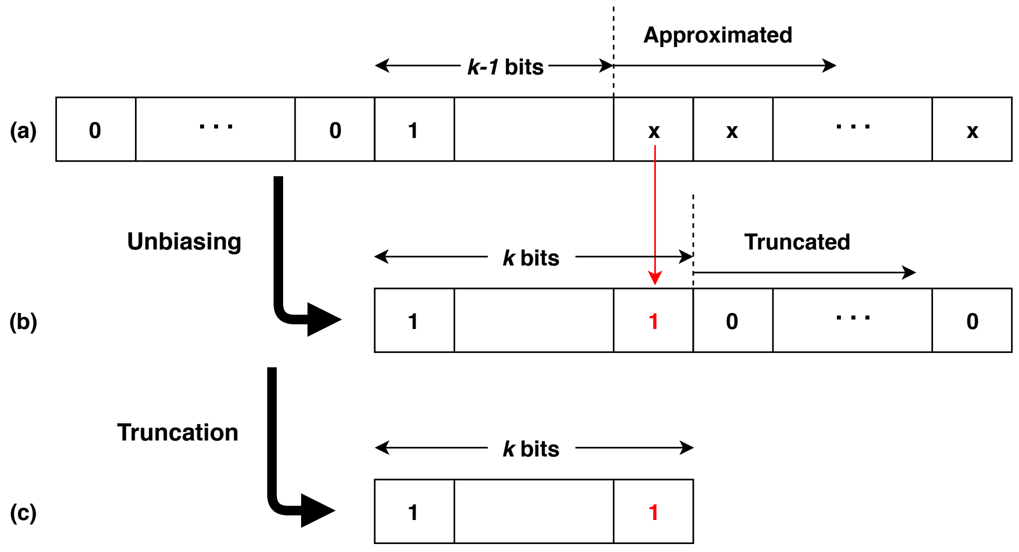


Figure 3.7: A general example of the approximate process. (a) Original number, (b) Number after unbiasing, (c) Final approximated input.

The aforementioned truncation technique belongs to the group of the truncation/rounding. A similar to this technique will be used in this diploma thesis in the next chapter, where all the developed multipliers will be presented.

Chapter 4

Proposed Approximate Techniques

4.1 Introduction

In this chapter all the work, that has been conducted in this diploma thesis, will be presented. There were developed five techniques:

1. Double High Radix Encoding
2. Double High Radix with perforation
3. High Radix with Correction
4. Perforation with Correction
5. Asymmetric Perforation and Rounding

After the theoretical part of each technique is presented, there will be a quick reference to the tools used for the simulations. In the end, the results, alongside with the conclusions, will be displayed.

4.2 Designs

As mentioned before, in this section the theory behind each technique will be analyzed.

4.2.1 Double High Radix Encoding

This technique is an extension of the previously presented hybrid high radix encoding, developed by the authors of [13].

In the proposed double high radix encoding, each operand is divided in two parts: the MSB part and the LSB part. For A the MSB part consists of n-m bits and the LSB part of m bits. Similarly, the MSB part of B comprises n-k bits and the LSB part k bits. The configuration parameters, $k, m \geq 4$, are even numbers, namely $k, m = 2l$: $l \in \mathbb{Z}$, with $l \geq 2$.

A is divided into A_1 and $x_0^{R2^m}$:

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i = A_1 + x_0^{R2^m} \qquad (4.2.1.1)$$

$$\text{where } A_1 = -a_{n-1}2^{n-1} + \sum_{\substack{j=m \\ m\geq 4}}^{n-2} a_i 2^i + a_{m-1}2^{m-1} \qquad (4.2.1.2)$$

$$\text{and } x_0^{R2^m} = -2^{m-1}a_{m-1} + 2^{m-2}a_{m-2} + \ldots + a_0 \qquad (4.2.1.3)$$

A_1 is the MSB part of A and $x_0^{R2^m} \in \{0, \pm 1, \pm 2, \pm 3, \ldots, \pm(2^{m\text{-}1}\text{-}1), \text{-}2^{m\text{-}1}\}$ corresponds to the radix-2^m encoding.

The MSB part of B is encoded using the radix-4 (modified Booth) encoding, while its LSB part is encoded with the high radix-2^k encoding.

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i = B_1 + y_0^{R2^k} \qquad (4.2.1.4)$$

$$\text{where } B_1 = \sum_{\substack{j=k/2 \\ k\geq 4}}^{n/2-1} y_j^{R4} 4^j \qquad (4.2.1.5)$$

$$\text{and } y_j^{R4} = -2b_{2j+1} + b_{2j} + b_{2j-1} \qquad (4.2.1.6)$$

$$\text{and } y_0^{R2^k} = -2^{k-1}b_{k-1} + 2^{k-2}b_{k-2} + \ldots + b_0 \qquad (4.2.1.7)$$

The radix-4 encoding includes $(n\text{-}k)/2$ digits $y_j^{R4} \in \{0, \pm 1, \pm 2\}$, while $y_0^{R2^k} \in \{0, \pm 1, \pm 2, \pm 3, \ldots, \pm(2^{k\text{-}1}\text{-}1), \text{-}2^{k\text{-}1}\}$ corresponds to the radix-2^k encoding. Overall, B is encoded with $(n\text{-}k)/2 + 1$ digits.

Next, the multiplication $A \times B$ is performed:

$$\begin{aligned} A \times B &= (A_1 + x_0^{R2^m}) \cdot (B_1 + y_0^{R2^k}) \\ &= (A_1 + x_0^{R2^m}) \cdot B_1 + (A_1 + x_0^{R2^m}) \cdot y_0^{R2^k} \\ &= A_1 \cdot B_1 + B_1 \cdot x_0^{R2^m} + A \cdot y_0^{R2^k} \qquad (4.2.1.8) \end{aligned}$$

The above double high radix technique is characterized by increased logic complexity, due to the high radix values of $x_0^{R2^m}$ and $y_0^{R2^k}$ that are not power of two, and thus, an approximate version is proposed. However, in order to retain high accuracy, the radix-4 encoding of the MSB of B is performed accurately. In particular, in the approximate encoding, all the values that are not power of two and the m-4 and k-4 smallest powers of two, respectively, are rounded to the nearest of the 4 largest powers of two or 0, so that the sum of all the values of the approximate digits $\hat{x}_0^{R2^m}$ and $\hat{y}_0^{R2^k}$ are 0. Only the 4 largest powers of two are kept, so that the radix-2^k/radix-2^m encoding circuit requires only about the double area in comparison with the accurate radix-4 encoder. Therefore, A and B are approximated as follows:

$$\tilde{A} = A_1 + \hat{x}_0^{R2^m} \tag{4.2.1.9}$$

$$\tilde{B} = B_1 + \hat{y}_0^{R2^k} \tag{4.2.1.10}$$

$$\text{where } \hat{x}_0^{R2^m} \in \{0, \pm 2^{m-4}, \pm 2^{m-3}, \pm 2^{m-2}, \pm 2^{m-1}\} \tag{4.2.1.11}$$

$$\text{and } \hat{y}_0^{R2^k} \in \{0, \pm 2^{k-4}, \pm 2^{k-3}, \pm 2^{k-2}, \pm 2^{k-1}\} \tag{4.2.1.12}$$

As a result the (4.2.1.8) is modified as follows:

$$\tilde{A} \times \tilde{B} = A_1 \cdot B_1 + B_1 \cdot \hat{x}_0^{R2^m} + A \cdot \hat{y}_0^{R2^k} \tag{4.2.1.13}$$

The logical equations of the radix-4 encoding, which is applied only to the MSB part of B are:

$$sign_j = b_{2j+1} \tag{4.2.1.14}$$

$$\times 1_j = b_{2j-1} \oplus b_{2j} \tag{4.2.1.15}$$

$$\times 2_j = (b_{2j+1} \oplus b_{2j}) \cdot \overline{(b_{2j-1} \oplus b_{2j})} \tag{4.2.1.16}$$

The accurate radix-4 encoding is displayed in the next table, which was also presented it the previous chapter and is the table 3.1

Input			**R4 Digit**	**Output**		
b_{2j+1}	b_{2j}	$b_{2j\text{-}1}$	y_j^{R4}	$sign_j$	$\times 2_j$	$\times 1_j$
0	0	0	0	0	0	0
0	0	1	1	0	0	**1**
0	1	0	1	0	0	**1**
0	1	1	2	0	**1**	0
1	0	0	-2	**1**	**1**	0
1	0	1	-1	**1**	0	**1**
1	1	0	-1	**1**	0	**1**
1	1	1	0	**1**	0	0

Accurate Radix-4 Encoding Table

The logic equations of the encoding signals that define the radix-2^k digit $\hat{y}_0^{R2^k}$ are:

$$sign = b_{k-1} \tag{4.2.1.17}$$

$$\begin{aligned}\times 2^{k-4} &= (\bar{b}_{k-2} \cdot \bar{b}_{k-3} \cdot \bar{b}_{k-4} + b_{k-2} \cdot b_{k-3} \cdot b_{k-4}) \\ &\cdot (b_{k-4} \oplus b_{k-5})\end{aligned} \tag{4.2.1.18}$$

$$\begin{aligned}\times 2^{k-3} &= \bar{b}_{k-1} \cdot \bar{b}_{k-2} \cdot (\bar{b}_{k-3} \cdot b_{k-4} \cdot b_{k-5} + b_{k-3} \cdot \bar{b}_{k-4}) \\ &+ b_{k-1} \cdot b_{k-2} \cdot (b_{k-3} \cdot \bar{b}_{k-4} \cdot \bar{b}_{k-5} + \bar{b}_{k-3} \cdot b_{k-4})\end{aligned} \tag{4.2.1.19}$$

$$\begin{aligned}\times 2^{k-2} &= \bar{b}_{k-2} \cdot b_{k-3} \cdot (b_{k-1} + b_{k-4}) \\ &+ b_{k-2} \cdot \bar{b}_{k-3} \cdot (\bar{b}_{k-1} + \bar{b}_{k-4})\end{aligned} \tag{4.2.1.20}$$

$$\times 2^{k-1} = \bar{b}_{k-1} \cdot b_{k-2} \cdot b_{k-3} + b_{k-1} \cdot \bar{b}_{k-2} \cdot \bar{b}_{k-3} \tag{4.2.1.21}$$

The above equations apply also for the radix-2^m digit $\hat{x}_0^{R2^k}$. The LSB part of both A and B is encoded in the same way. The only difference is that they have separate configuration parameters. Therefore, the same encoding is applied to both of them. Thus, the notation radix-2^k is used for this type of encoding. The next table is the table 3.2, which was presented in the previous chapter, and shows the radix-2^k encoding.

R2^k Digit		**Output**				
$y_0^{R2^k}$	$\hat{y}_0^{R2^k}$	$sign$	$\times 2^{k-1}$	$\times 2^{k-2}$	$\times 2^{k-3}$	$\times 2^{k-4}$
$[0, 2^{k-5})$	0	0	0	0	0	0
$[2^{k-5}, 2^{k-4}+2^{k-5})$	2^{k-4}	0	0	0	0	**1**
$[2^{k-4}+2^{k-5}, 2^{k-3}+2^{k-4})$	2^{k-3}	0	0	0	**1**	0
$[2^{k-3}+2^{k-4}, 2^{k-2}+2^{k-3})$	2^{k-2}	0	0	**1**	0	0
$[2^{k-2}+2^{k-3}, 2^{k-1})$	2^{k-1}	0	**1**	0	0	0
$[-2^{k-1}, -2^{k-2}-2^{k-3})$	-2^{k-1}	**1**	**1**	0	0	0
$[-2^{k-2}-2^{k-3}, -2^{k-3}-2^{k-4})$	-2^{k-2}	**1**	0	**1**	0	0
$[-2^{k-3}-2^{k-4}, -2^{k-4}-2^{k-5})$	-2^{k-3}	**1**	0	0	**1**	0
$[-2^{k-4}-2^{k-5}, -2^{k-5})$	-2^{k-4}	**1**	0	0	0	**1**
$[-2^{k-5}, 0)$	0	**1**	0	0	0	0

Approximate Radix-2^k Encoding Table

It should be noted that the notation DRAD$|_{2^m,2^k}$ is used to describe this technique. In the following example, presented in figure 4.1, $k = 8$ and $m = 8$ are chosen. The color grey indicates the sign extensions, the color black the sign factors and the color white the generated bits of the partial products. The circles represent the partial products generated by the factor $A_1 \cdot B_1$, the squares the partial product generated by the factor $A \cdot \hat{y}_0^{R2^k}$ and the triangles the partial product generated by the factor $B_1 \cdot \hat{x}_0^{R2^m}$.

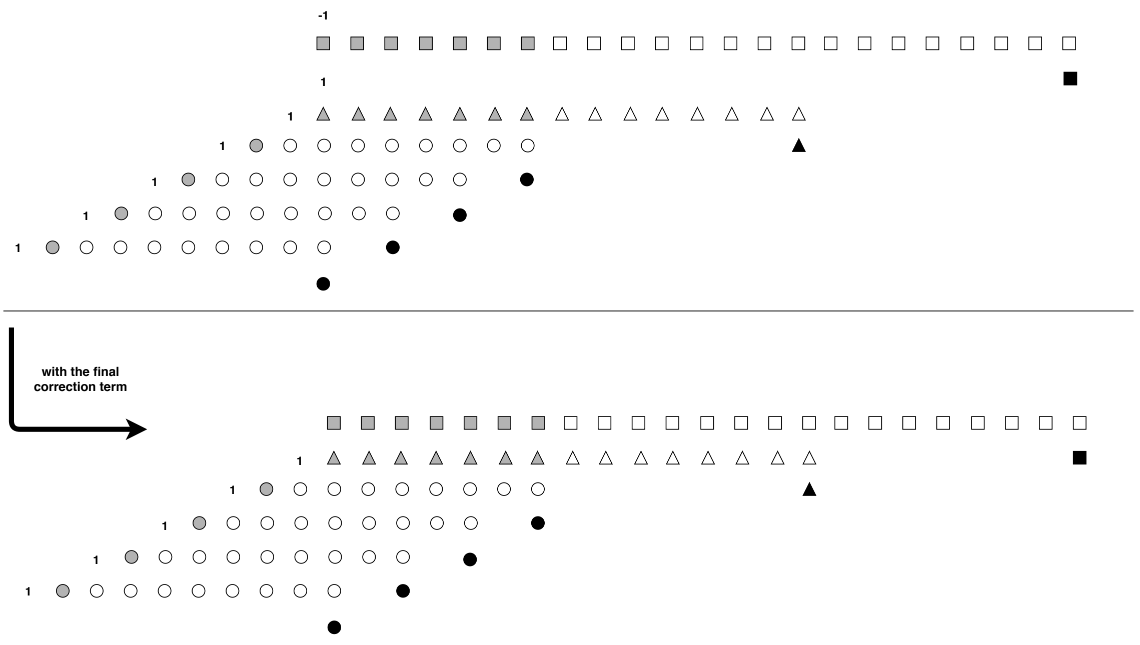


Figure 4.1: DRAD$|_{256,256}$

4.2.2 Double High Radix with Perforation

This technique is a follow up of the aforementioned double high radix encoding with just one modification. A certain factor of the equation (4.2.1.13) is eliminated, specifically $A\hat{y}_0^{R2^k}$. So Double High Radix with Perforation can be described by the following equation:

$$\tilde{A} \times \tilde{B} = A_1 \cdot B_1 + B_1 \cdot \hat{x}_0^{R2^m} \tag{4.2.2.1}$$

The equations (4.2.1.14), (4.2.1.15) and (4.2.1.16) are still used for the radix-4 encoding of the MSB of B. As for $\hat{x}_0^{R2^m}$ continues to be computed through the equations (4.2.1.17), (4.2.1.18), (4.2.1.19), (4.2.1.20) and (4.2.1.21) The notation DRADP$|_{2^m,2^k}$ is used to describe this technique. In figure 4.2 the previously displayed example in figure 4.1 is presented, but this time the partial product generated by the factor $A\hat{y}_0^{R2^k}$ is eliminated. As before, the circles stand for the partial products generated by the factor $A_1 \cdot B_1$ and the triangles for the partial product generated by the factor $B_1 \cdot \hat{x}_0^{R2^m}$. The color grey indicates the sign extensions, the color black the sign factors, the color white the generated bits of the partial products and the newly used color purple the inverse bit of the sign extension. The relation $1 + c = c\bar{c}$ was used to form the final correction term.

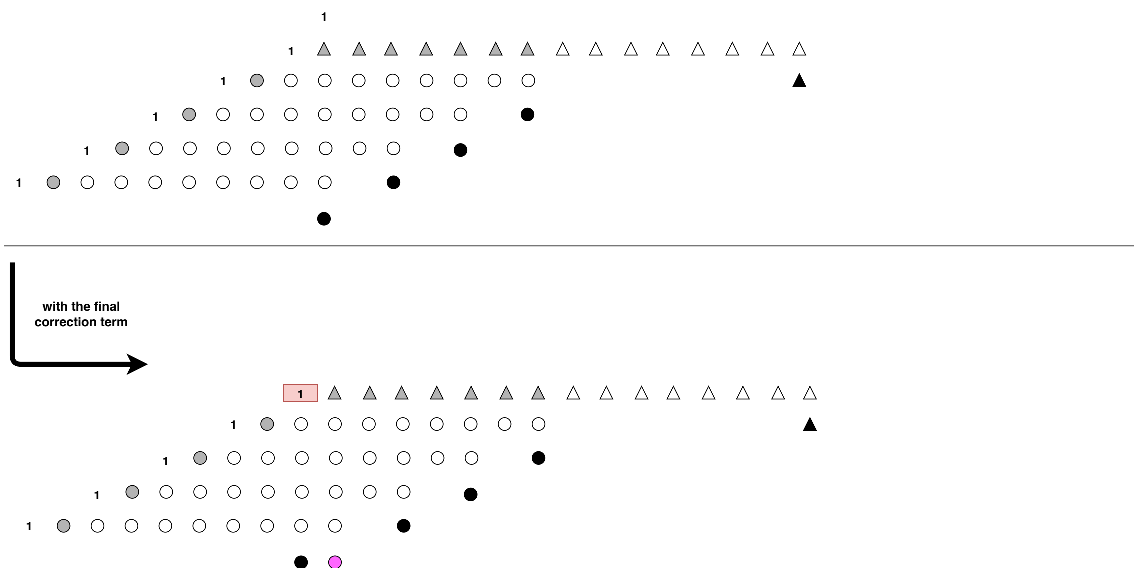


Figure 4.2: DRADP$|_{256,256}$

4.2.3 High Radix with Correction

Taking into consideration the double high radix encoding, High Radix with Correction was developed. The equations (4.2.1.4), (4.2.1.5),(4.2.1.6) and (4.2.1.7) are still being used for the encoding of B, but A isn't divided to MSB part and LSB part as before. Therefore $A \times B$ is computed as follows:

$$A \times B = A \cdot (B_1 \;+\; \hat{y}_0^{R2^k}) = A \cdot B_1 \;+\; A \cdot \hat{y}_0^{R2^k} \tag{4.2.3.1}$$

The partial product matrix generated by AB_1 is divided into the exact part (EP) and the approximate part (AP). (AP) is truncated, after an approximation is occured. Figure 4.3 shows the partial product matrix of 16×16 bit multipliers when $k = 6$, so a radix-64 encoding is applied to B. AP will be approximated as follows: The expected value of a uniform distribution for numbers in the interval $[0, 2^t - 1]$ is 2^{t-1}. The AP is directly combined with the selection of t. In the example of figure 4.3 we choose $t_1 = 8$, $t_2 = 6$, $t_3 = 4$, $t_4 = 2$. There isn't a t_0 because this approximation isn't applied to the partial product generated by the factor $A\hat{y}_0^{R2^k}$ of the equation (4.2.3.1). In figure 4.3 is also displayed the result of this approximation and truncation. The expected value is implemented with the logic equation:

$$y_i' = \;\times 1_i + \;\times 2_i \text{ with } i = 1...4 \tag{4.2.3.2}$$

$\times 1_i$ and $\times 2_i$ are computed with the equations (4.2.1.15) and (4.2.1.16). The extra "1" as a correction term is used as a form of rounding in order to reduce the total error.

It should be noted that the notation RADC$|_{2^k,t}$, where t is the equal to t_1.

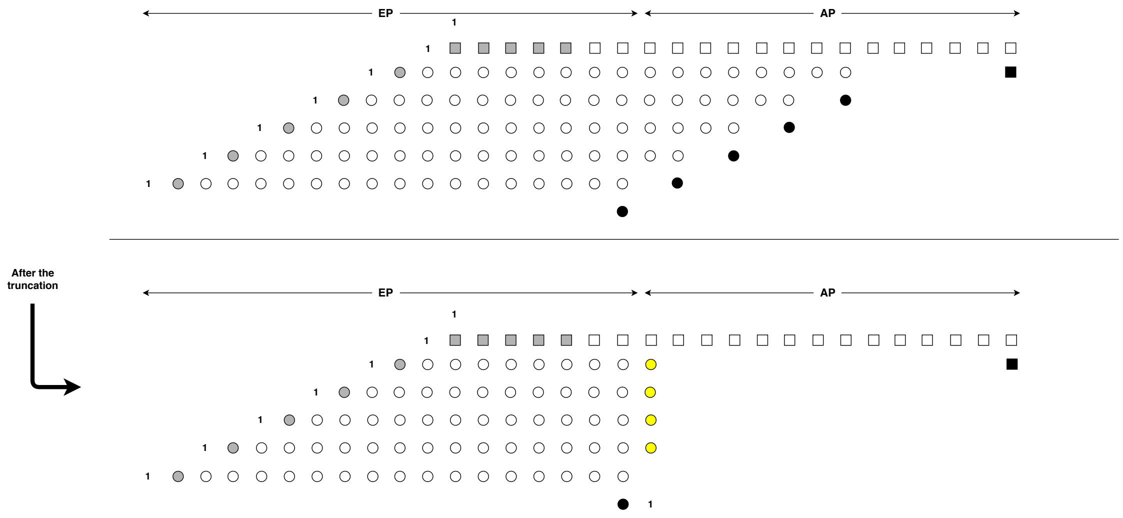


Figure 4.3: $\text{RADC}|_{64,8}$

In figure 4.3 the color grey indicates the sign extensions, the color black the sign factors and the color white the generated bits of the partial products, exactly as they did in previous examples. The extra color yellow stands for the bits y_i'. The circles represent the partial products generated by the factor $A \cdot B_1$ and the squares the partial product generated by the factor $A \cdot \hat{y}_0^{R2^k}$.

4.2.4 Perforation with Correction

The first idea was to to exclude the factor $A\hat{y}_0^{R2^k}$ from the equation (4.2.3.1), which describes the previous technique (High Radix with Correction). This would equivalent of eliminating at least the first 2 successive partial products because of the requirements: $k = 2l$ and $k \geq 4$. So finally this technique is a combination of Partial Product Perforation(PPP) as described in [30] and the previously described truncation method. In figure 4.4 a typical example is displayed, where the first 3 consecutive partial products are excluded and $t_3 = 8$ (Regarding the truncation method, the value of t_i for just the first partial product included in the matrix is given). The expected value is still computed via the logic equation:

$$y_i' = \times 1_i + \times 2_i \text{ with } i = 3...7 \tag{4.2.4.1}$$

$\times 1_i$ and $\times 2_i$ are still implemented with the equations (4.2.1.15) and (4.2.1.16). The partial product matrix is still divided into two parts: the AP part and the EP part. The extra "1" as a correction term is used as a form of rounding up in order to decrease the total error. The notation $\text{PERFOC}|_{k,t}$ is used to describe this technique ; t is equal to t_i, where i is the number of the first non perforated partial product.

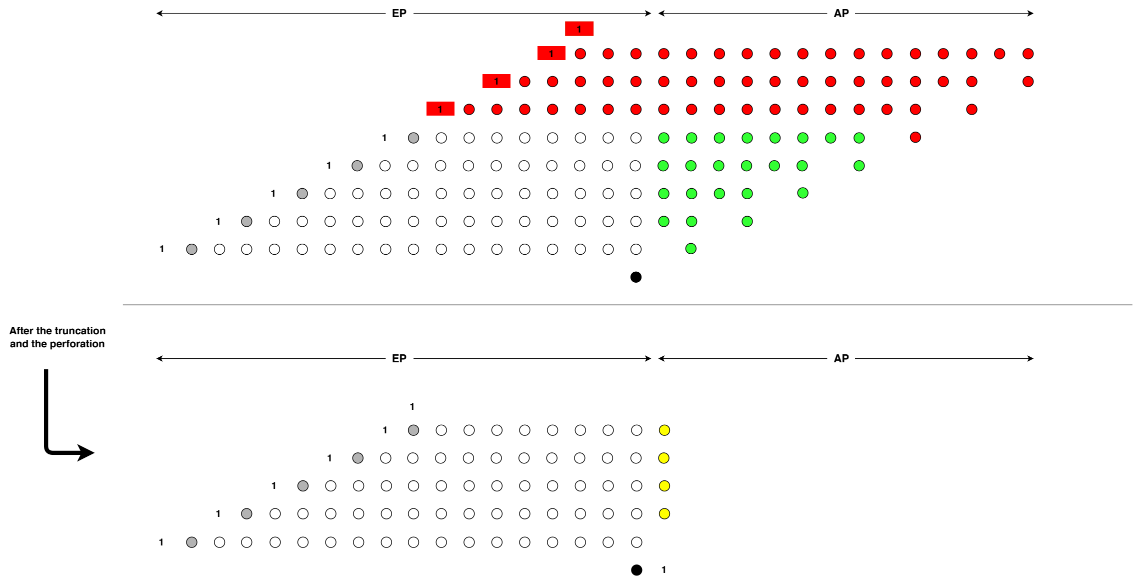


Figure 4.4: PERFOC$|_{3,8}$

In figure 4.4 the color grey represents the sign extensions, the color black the sign factors and the color white the generated bits of the partial products, exactly as they did in the previous example. Furthermore, the color yellow indicates the bits y_i', the color green the bits, which are being truncated, and the color red the partial products, that are being perforated.

4.2.5 Asymmetric Perforation and Rounding

The Asymmetric Perforation and Rounding technique is a variation of the hybrid partial product perforation-rounding technique, which was developed in [14]. The perforation of partial products is still applied as in the perforation with correction technique. The difference between this method and the described in [14] is the way the rounding is applied. Specifically, rounding is applied to the t_i-bit of each partial product (where i takes the value from the number of the first not perforated partial product to the number of the last partial product). Therefore, the $t_i - 1$ LSBs of A are truncated, and a_{t_i-1} is added to the most significant remaining part as the next equation shows:

$$A_{t_i} + a_{t_i-1} = \langle a_{n-1}, a_{n-2} \ldots\ a_{t_i} \rangle_{2'S} + a_{t_i-1} \tag{4.2.5.1}$$

In the case of $a_{t_i-1} = 0$, the approximated partial products $(\hat{P}_j)$ are computed by the next equation

$$\hat{P}_j = (A_{t_i} + 0)y_j^{R4} = A_{t_i} y_j^{R4} \tag{4.2.5.2}$$

where y_j^{R4} is calculated in (4.2.1.6)

In case of $a_{t_i-1} = 1$, and using the relation $A_{t_i} + 1 = -\bar{A}_{t_i}$, the approximated partial products are calculated by the next equation

$$\hat{P}_j = (A_{t_i} + 1)y_j^{R4} = (-\bar{A}_{t_i})y_j^{R4} = \bar{A}_{t_i}(-y_j^{R4}) \ , \tag{4.2.5.3}$$
$$\text{where } -y_j^{R4} = (-1)^{\bar{s}_j}(\times 2_j + \times 1_j)$$

s_j is equivalent to $sign_j$, and $sign_j$, $\times 1_j$ and $\times 2_j$ are calculated by the equations (4.2.1.14), (4.2.1.15) and (4.2.1.16),respectively.

The two cases are combined through the relation $A^*_{t_i} = A_{t_i} \oplus a_{t_i - 1}$ to form either A_{t_i} or $-\bar{A}_{t_i}$,depending on the value of $a_{t_1 - 1}$. Similarly the relation, $s^*_j = s_j \oplus a_{t_i-1}$, is used to form either y_j^{R4} or $-y_j^{R4}$. So the partial products are computed by $\hat{P} = A^*_{t_i} y_j^{R4*}$ where $-y_j^{R4} = (-1)^{\bar{s}_j}(\times 2_j + \times 1_j)$. As for the sign factors, the next logic equation is used to describe them.

$$c^*_j = s^*_j \wedge (\times 2_j \vee \times 1_j)$$

The asymmetry is shown in figure 4.5, where an example of this technique is presented. The logic behind the asymmetry is that the rounding can't be applied vertically without increasing the depth of the accumulation tree. Thus, it is applied as vertically as possible.

The notation APR$|_{k,t}$ is used to describe this technique ; t is equal to the value, that t_i would have gotten in the case of a perfectly vertical application of rounding, and i is the number of the first non perforated partial product.

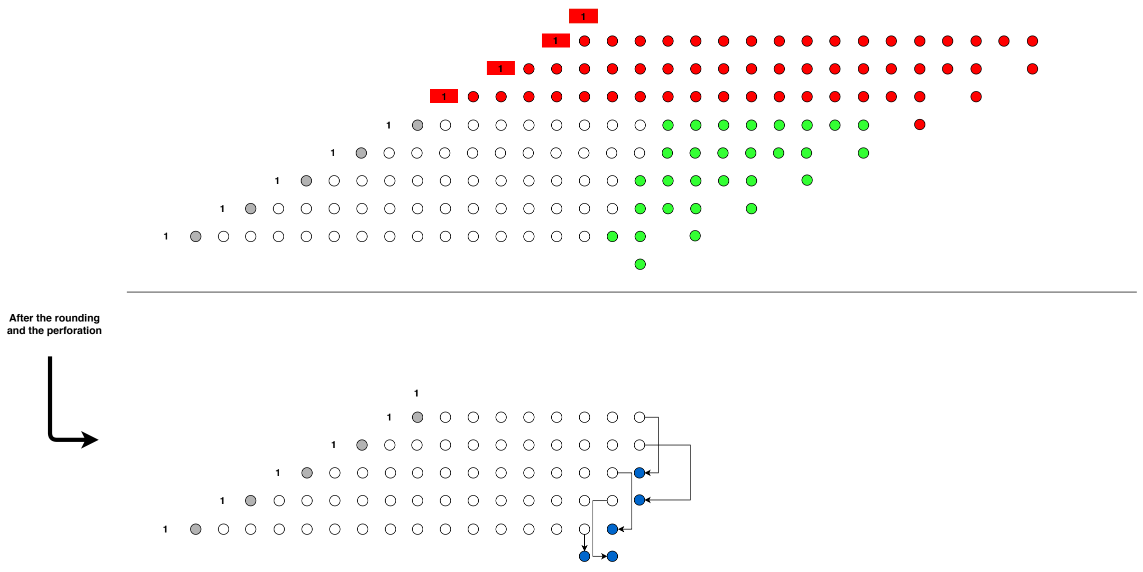


Figure 4.5: Partial Product Matrix of APR$|_{3,10}$

In figure 4.5 the color grey represents the sign extensions and the color white the generated bits of the partial products,exactly as they did in previous examples. Furthermore, the color blue stands for the newly formed sign factors c^*_j, the color green for the bits, which are being truncated, and the color red for the partial products, that are being eliminated.

4.3 Tools and Experimental Setup

All the multipliers to be compared, are implemented in *Verilog HDL*, synthesized using *Synopsys Design Compiler* and the *TSMC* 65-nm standard cell library, and simulated with *Mentor Graphics ModelSim*. The critical path delay and the area are reported by *Synopsys Design Compiler*, while the power consumption is measured with *Synopsys PrimeTime-PX* tool and the use of all the possible input combinations. All the designs are synthesized and simulated at $1V$, i.e., the nominal supply voltage. The procedure, that was followed, consists of the next steps:

1. The project is created via the command: **$create_project**

2. The Verilog code and the testbench are prepared.

3. The syntax of the source code and the testbench are checked via the commads **$make check_vlog** and **$make check_tb**,respectively.

4. In this step the rtl simulation is occurred via the command **$make rtl_sim**.Mentor Graphics ModelSim opens and the simulation begins. After that, with the help of matlab the functional correctness of the design is verified. Specifically, matlab and rtl simulation generate an output file of the results of 131070 multiplications. The two output files are compared and if they are identical, then the design is correct.

5. Firstly, all the parameters of the synthesis are set. One important parameter is the value of the clock period. Next, via the command **$make dcsyn** the design synthesis happens. This step is repeated until the critical path (smallest delay) is found. The command $make dcsyn uses the tool *Synopsys Design Compiler* and the TSMC 65-nm standard cell library in order to find out if the time constrains are violated and also computes the total area needed for the given parameters.

6. The command **$make sta** defines the clock period of the testbench to be the same with the clock period, which was used in the design synthesis. Furthermore, the Static Time Analysis(STA) provides a more accurate answer of whether the time constrains are met. Next, the command **$make gate_sim** is performed and a gate level simulation is occurred. Lastly, the command **$make power** computes the power consumption with the help of the *Synopsys PrimeTime-PX tool.*

7. The final step is to compute the error of the approximate multiplier. To achieve this, matlab is used, where the comparison of the exact and the approximate result leads to the computation of the average error.

Simulations were made for both, critical path delay and a more relaxed clock. In figure 4.6 the steps of the followed procedure are presented.

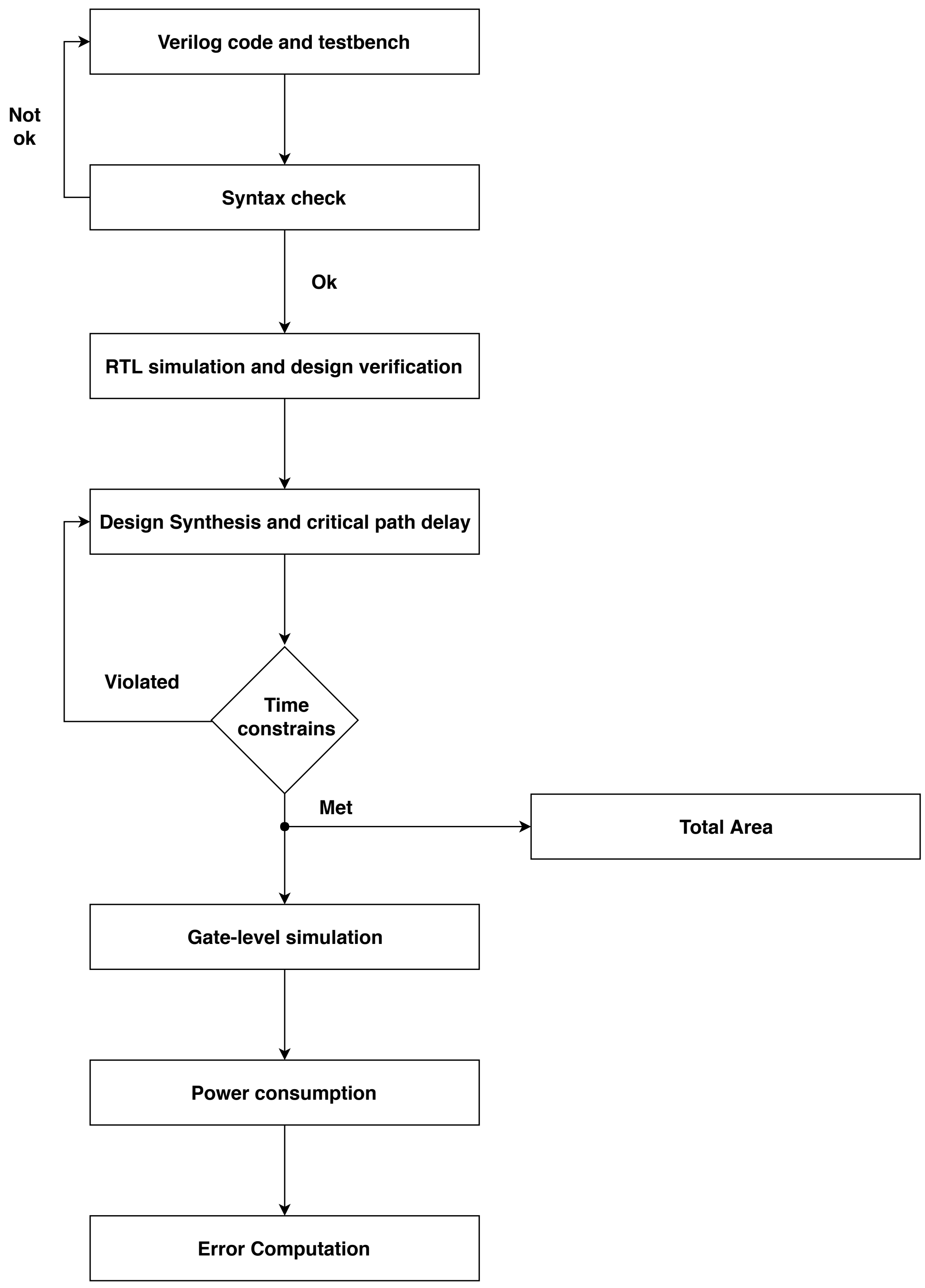


Figure 4.6: Flow chart of the followed procedure

4.4 Error Analysis

A critical issue in approximate computing designs is the error imposed due to the approximations and how it affects the final results. In [15], an error evaluation metric is proposed, being called mean relative error distance (*MRED*). *RED* is defined as the arithmetic difference between the accurate and the approximate product divided by the accurate product, while *MRED* is the average of *REDs* for a set of inputs (2^{2n} are all the possible input combinations for a $n \times n$ multiplier). The possibility of having *RED* smaller than 2% (*PRED*) is another important metric used in [9] and [18] for evaluating approximate radix multipliers.

Considering the multiplication of two n-bit numbers, A and B, with the $\tilde{P}$ being the approximate product and P the accurate, the *RED* is calculated by:

$$RED_{AB} = \frac{|P - \tilde{P}|}{|P|}$$

MRED is calculated by:

$$MRED = \frac{\sum RED_{AB}}{N}$$

where N the number of inputs.

PRED is given by:

$$PRED = \frac{pos}{N}$$

where *pos* the total sum of *REDs* being smaller than 2%.

4.5 Experimental Results

This section includes the evaluation of the developed techniques of this diploma thesis in terms of accuracy(error) and hardware (delay,area,power, and energy). All the simulations were made for a 16×16 multiplier ($n = 16$). Firstly, in table 4.1 all results of the double high radix encoding technique in critical path delay are displayed.

Multiplier	**Delay**(ns)	**Power**(μW)	**Area**(μm^2)	**Energy**($\mu W \cdot ns$)	**MRED**(%)	**PRED**%
DRAD$\|_{64,64}$	0.72	3376	2660	2430.72	0.15	99.19
DRAD$\|_{64,256}$	0.68	3500	2585	2380	0.34	97.98
DRAD$\|_{64,1024}$	0.64	3329	2302	2130.56	0.97	92.87
DRAD$\|_{256,64}$	0.69	3488	2574	2406.72	0.33	97.91
DRAD$\|_{256,256}$	0.66	3457	2470	2281.62	0.5	96.69
DRAD$\|_{256,1024}$	0.64	3029	2110	1938.56	1.11	91.62
DRAD$\|_{1024,64}$	0.67	3467	2475	2322.89	0.98	92.91
DRAD$\|_{1024,256}$	0.65	3106	2165	2018.9	1.12	91.70
DRAD$\|_{1024,1024}$	0.61	3022	1960	1843.42	1.65	86.75

Table 4.1: Total Results of DRAD$|_{2^m,2^k}$ in Critical Path Delay

In the next table 4.2 all results of the double high radix with perforation technique in critical path delay are displayed.

Multiplier	**Delay**(ns)	**Power**(μW)	**Area**(μm^2)	**Energy**$(\mu W \cdot ns)$	**MRED**(%)	**PRED**%
DRADP$\|_{64,64}$	0.68	3063	2335	2082.84	0.49	97.21
DRADP$\|_{64,256}$	0.63	2693	1975	1696.59	1.52	89.84
DRADP$\|_{64,1024}$	0.60	2468	1828	1480.80	4.84	60.68
DRADP$\|_{256,64}$	0.64	3005	2173	1923.2	0.65	95.93
DRADP$\|_{256,256}$	0.63	2251	1644	1418.13	1.65	88.59
DRADP$\|_{256,1024}$	0.57	2252	1643	1283.64	4.93	59.68
DRADP$\|_{1024,64}$	0.64	2404	1802	1538.56	1.25	90.96
DRADP$\|_{1024,256}$	0.58	2472	1734	1433.76	2.16	83.69
DRADP$\|_{1024,1024}$	0.55	1840	1333	1012	5.31	56.11

Table 4.2: Total Results of DRADP$|_{2^m,2^k}$ in Critical Path Delay

In the table 4.3 all results of the high radix with correction technique in critical path delay are presented.

Multiplier	**Delay**(ns)	**Power**(μW)	**Area**(μm^2)	**Energy**$(\mu W \cdot ns)$	**MRED**(%)	**PRED**%
RADC$\|_{64,6}$	0.72	3449	2748	2483.28	0.1	99.54
RADC$\|_{64,8}$	0.71	3429	2716	2434.59	0.15	99.35
RADC$\|_{64,10}$	0.68	3325	2523	2261	0.38	98.61
RADC$\|_{256,6}$	0.66	3494	2604	2306.04	0.33	98.21
RADC$\|_{256,8}$	0.64	3305	2418	2115.2	0.52	97.62
RADC$\|_{256,10}$	0.64	2893	2116	1832.32	1.45	95.49
RADC$\|_{1024,6}$	0.61	3380	2535	2061.8	1.13	92.77
RADC$\|_{1024,8}$	0.59	3260	2309	1923.4	1.94	91.21
RADC$\|_{1024,10}$	0.58	3031	2024	1757.98	5.57	85.47

Table 4.3: Total Results of RADC$|_{2^k,t}$ in Critical Path Delay

In the next table 4.4 all results of the perforation with correction technique in critical path delay are displayed. Lastly, in the table 4.5 all results of the asymmetric perforation and rounding technique in critical path delay are presented.

Multiplier	**Delay**(ns)	**Power**(μW)	**Area**(μm^2)	**Energy**$(\mu W \cdot ns)$	**MRED**(%)	**PRED**%
PERFOC$\|_{1,6}$	0.73	4011	3329	2928.03	0.04	99.82
PERFOC$\|_{1,8}$	0.73	3535	2939	2580.55	0.04	99.81
PERFOC$\|_{1,10}$	0.71	3803	3101	2700.13	0.05	99.76
PERFOC$\|_{1,12}$	0.72	2979	2466	2144.88	0.11	99.53
PERFOC$\|_{1,14}$	0.69	2669	2174	1841.61	0.38	98.58
PERFOC$\|_{2,6}$	0.69	3534	2786	2438.46	0.14	99.39
PERFOC$\|_{2,8}$	0.71	3096	2544	2198.16	0.15	99.35
PERFOC$\|_{2,10}$	0.68	3028	2478	2059.04	0.21	99.18
PERFOC$\|_{2,12}$	0.66	2754	2160	1817.64	0.48	98.41
PERFOC$\|_{2,14}$	0.66	3126	2342	2063.16	1.72	95.71
PERFOC$\|_{3,6}$	0.66	3005	2367	1983.3	0.45	95.57
PERFOC$\|_{3,8}$	0.66	2635	2129	1739.1	0.49	97.44
PERFOC$\|_{3,10}$	0.64	2428	1910	1553.92	0.68	96.88
PERFOC$\|_{3,12}$	0.63	1965	1519	1237.95	1.62	94.83
PERFOC$\|_{4,6}$	0.62	3158	2414	1957.96	1.52	90.14
PERFOC$\|_{4,8}$	0.60	2673	2114	1603.8	1.67	89.79
PERFOC$\|_{4,10}$	0.59	2241	1721	1322.19	2.47	88.37

Table 4.4: Total Results of PERFOC$|_{k,t}$ in Critical Path Delay

Multiplier	**Delay**(ns)	**Power**(μW)	**Area**(μm^2)	**Energy**$(\mu W \cdot ns)$	**MRED**(%)	**PRED**%
APR$\|_{1,8}$	0.75	3675	3188	2756.26	0.04	99.82
APR$\|_{1,10}$	0.73	3527	3021	2574.71	0.04	99.79
APR$\|_{1,12}$	0.73	3198	2756	2334.54	0.05	99.74
APR$\|_{1,14}$	0.7	3140	2548	2198	0.12	99.37
APR$\|_{1,15}$	0.69	2775	2281	1914.75	0.19	98.91
APR$\|_{2,8}$	0.71	3260	2703	2314.6	0.13	99.37
APR$\|_{2,10}$	0.7	3058	2606	2140.6	0.14	99.33
APR$\|_{2,12}$	0.69	2687	2197	1854.03	0.19	99.02
APR$\|_{2,13}$	0.68	2635	2142	1791.8	0.25	98.6
APR$\|_{2,14}$	0.68	2254	1836	1532.72	0.38	97.77
APR$\|_{2,15}$	0.66	2170	1712	1432.2	0.62	96.19
APR$\|_{3,6}$	0.68	2890	2436	1965.2	0.44	97.58
APR$\|_{3,8}$	0.67	3058	2519	2048.86	0.45	97.54
APR$\|_{3,10}$	0.65	2746	2183	1784.9	0.49	97.26
APR$\|_{3,12}$	0.64	2266	1787	1450.24	0.63	96.21
APR$\|_{3,13}$	0.64	1993	1590	1275.52	0.82	94.89
APR$\|_{3,14}$	0.63	1843	1385	1161.09	1.24	92.17
APR$\|_{3,15}$	0.60	2161	1694	1296.6	1.96	87.13
APR$\|_{4,6}$	0.64	2483	2019	1589.12	1.49	90.19
APR$\|_{4,8}$	0.61	2480	1949	1512.8	1.52	89.97
APR$\|_{4,10}$	0.61	1991	1574	1214.51	1.62	89.17
APR$\|_{4,12}$	0.59	1958	1575	1155.22	2.07	86.12

Table 4.5: Total Results of APR$|_{k,t}$ in Critical Path Delay

In the next tables 4.6,4.7,4.8,4.9 and 4.10 the results of each technique in a relaxed clock are presented. The chosen delay is $0.8ns$

Multiplier	**Delay**(ns)	**Power**(μW)	**Area**(μm^2)	**Energy**$(\mu W \cdot ns)$	**MRED**$(\%)$	**PRED**$\%$
DRAD$\|_{64,64}$	0.80	2474	2252	1979.2	0.15	99.19
DRAD$\|_{64,256}$	0.80	2180	1987	1744	0.34	97.98
DRAD$\|_{64,1024}$	0.80	1860	1651	1488	0.97	92.87
DRAD$\|_{256,64}$	0.80	2371	2075	1896.8	0.33	97.91
DRAD$\|_{256,256}$	0.80	2024	1770	1619.2	0.5	96.69
DRAD$\|_{256,1024}$	0.80	1788	1549	1430	1.11	91.62
DRAD$\|_{1024,64}$	0.80	2105	1850	1684	0.98	92.91
DRAD$\|_{1024,256}$	0.80	1899	1648	1519.2	1.12	91.70
DRAD$\|_{1024,1024}$	0.80	1666	1471	1332.8	1.65	86.75

Table 4.6: Total Results of DRAD$|_{2^m,2^k}$ in Relaxed Clock

Multiplier	**Delay**(ns)	**Power**(μW)	**Area**(μm^2)	**Energy**$(\mu W \cdot ns)$	**MRED**$(\%)$	**PRED**$\%$
DRADP$\|_{64,64}$	0.80	1928	1738	1542.4	0.49	97.21
DRADP$\|_{64,256}$	0.80	1540	1440	1232	1.52	89.84
DRADP$\|_{64,1024}$	0.80	1269	1177	1015.2	4.84	60.68
DRADP$\|_{256,64}$	0.80	1659	1532	1327.2	0.65	95.93
DRADP$\|_{256,256}$	0.80	1350	1267	1080	1.65	88.59
DRADP$\|_{256,1024}$	0.80	1092	1067	873.6	4.93	59.68
DRADP$\|_{1024,64}$	0.80	1409	1315	1127.2	1.25	90.96
DRADP$\|_{1024,256}$	0.80	1145	1114	916	2.16	83.69
DRADP$\|_{1024,1024}$	0.80	914	851	731.2	5.31	56.11

Table 4.7: Total Results of DRADP$|_{2^m,2^k}$ in Realxed Clock

Multiplier	**Delay**(ns)	**Power**(μW)	**Area**(μm^2)	**Energy**$(\mu W \cdot ns)$	**MRED**$(\%)$	**PRED**$\%$
RADC$\|_{64,6}$	0.80	2589	2290	2071.2	0.1	99.54
RADC$\|_{64,8}$	0.80	2486	2126	1988.8	0.15	99.35
RADC$\|_{64,10}$	0.80	2217	1930	1773.6	0.38	98.61
RADC$\|_{256,6}$	0.80	2116	1901	1692	0.33	98.21
RADC$\|_{256,8}$	0.80	1970	1742	1576	0.52	97.62
RADC$\|_{256,10}$	0.80	1823	1602	1458.4	1.45	95.49
RADC$\|_{1024,6}$	0.80	1653	1573	1322.4	1.13	92.77
RADC$\|_{1024,8}$	0.80	1615	1462	1292	1.94	91.21
RADC$\|_{1024,10}$	0.80	error is too big			5.57	85.47

Table 4.8: Total Results of RADC$|_{2^k,t}$ in Relaxed Clock

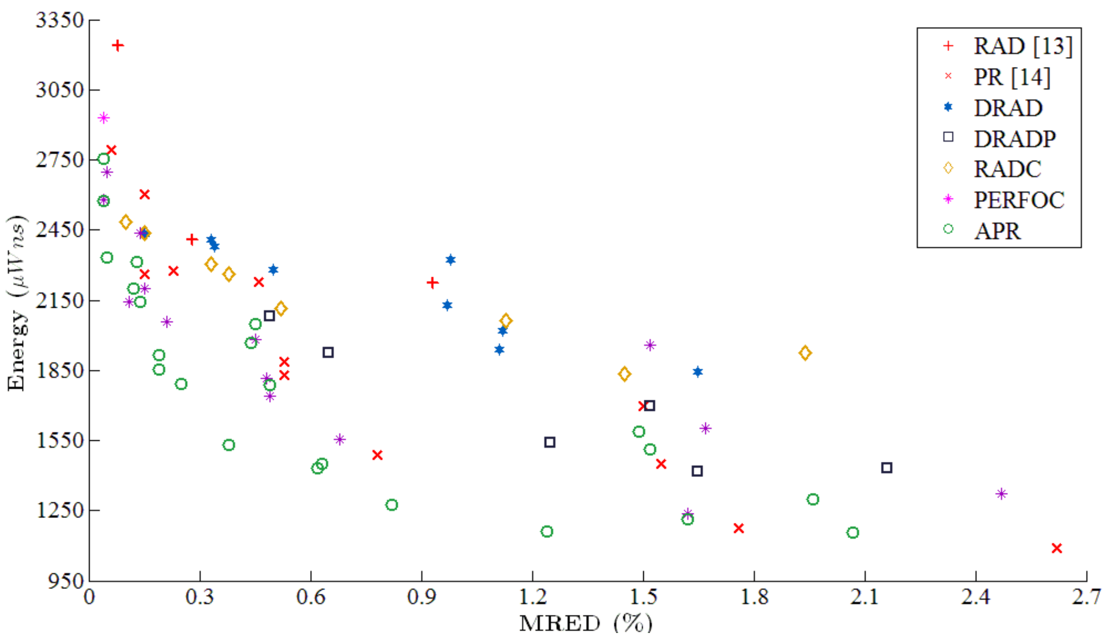
Multiplier	**Delay**(ns)	**Power**(μW)	**Area**(μm^2)	**Energy**$(\mu W \cdot ns)$	**MRED**(%)	**PRED**%
PERFOC$\|_{1,6}$	0.80	2818	2557	2254.4	0.04	99.82
PERFOC$\|_{1,8}$	0.80	2593	2380	2074.4	0.04	99.81
PERFOC$\|_{1,10}$	0.80	2486	2225	1988.8	0.05	99.76
PERFOC$\|_{1,12}$	0.80	2058	1892	1646.4	0.11	99.53
PERFOC$\|_{1,14}$	0.80	1789	1632	1431.2	0.38	98.58
PERFOC$\|_{2,6}$	0.80	2264	2117	1811.2	0.14	99.39
PERFOC$\|_{2,8}$	0.80	2097	1942	1677.6	0.15	99.35
PERFOC$\|_{2,10}$	0.80	1871	1707	1496.8	0.21	99.18
PERFOC$\|_{2,12}$	0.80	1661	1490	1328.8	0.48	98.41
PERFOC$\|_{2,14}$	0.80	1667	1550	1333.6	1.72	95.71
PERFOC$\|_{3,6}$	0.80	1856	1701	1484.8	0.45	95.57
PERFOC$\|_{3,8}$	0.80	1684	1533	1347.2	0.49	97.44
PERFOC$\|_{3,10}$	0.80	1448	1342	1158.4	0.68	96.88
PERFOC$\|_{3,12}$	0.80	1221	1129	976.8	1.62	94.83
PERFOC$\|_{4,6}$	0.80	1654	1506	1323.2	1.52	90.14
PERFOC$\|_{4,8}$	0.80	1413	1323	1130.4	1.67	89.79
PERFOC$\|_{4,10}$	0.80	1092	1094	873.6	2.47	88.37

Table 4.9: Total Results of PERFOC$|_{k,t}$ in Relaxed Clock

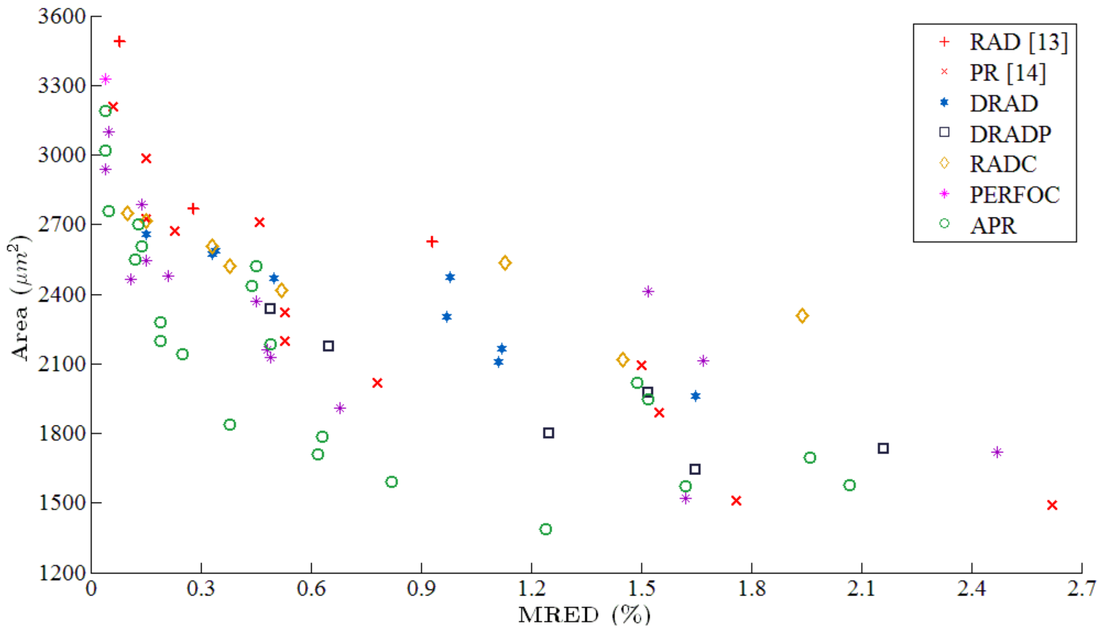
Multiplier	**Delay**(ns)	**Power**(μW)	**Area**(μm^2)	**Energy**$(\mu W \cdot ns)$	**MRED**(%)	**PRED**%
APR$\|_{1,8}$	0.80	2915	2611	2332	0.04	99.82
APR$\|_{1,10}$	0.80	2555	2354	2044	0.04	99.79
APR$\|_{1,12}$	0.80	2441	2214	1952.8	0.05	99.74
APR$\|_{1,14}$	0.80	2069	1885	1655.2	0.12	99.37
APR$\|_{1,15}$	0.80	1882	1738	1505.6	0.19	98.91
APR$\|_{2,8}$	0.80	2276	2068	1820.8	0.13	99.37
APR$\|_{2,10}$	0.80	2099	1927	1679.2	0.14	99.33
APR$\|_{2,12}$	0.80	1782	1659	1425.6	0.19	99.02
APR$\|_{2,13}$	0.80	1708	1537	1366.4	0.25	98.6
APR$\|_{2,14}$	0.80	1505	1395	1204	0.38	97.77
APR$\|_{2,15}$	0.80	1351	1250	1080.8	0.62	96.19
APR$\|_{3,6}$	0.80	1887	1804	1509.6	0.44	97.58
APR$\|_{3,8}$	0.80	1823	1711	1458.4	0.45	97.54
APR$\|_{3,10}$	0.80	1567	1487	1253.6	0.49	97.26
APR$\|_{3,12}$	0.80	1348	1271	1078.4	0.63	96.21
APR$\|_{3,13}$	0.80	1227	1154	981.6	0.82	94.89
APR$\|_{3,14}$	0.80	1108	1045	886.4	1.24	92.17
APR$\|_{3,15}$	0.80	1114	1094	891.2	1.96	87.13
APR$\|_{4,6}$	0.80	1551	1478	1240	1.49	90.19
APR$\|_{4,8}$	0.80	1332	1298	1065.6	1.52	89.97
APR$\|_{4,10}$	0.80	1159	1117	927.2	1.62	89.17
APR$\|_{4,12}$	0.80	981	1022	784.8	2.07	86.12

Table 4.10: Total Results of APR$|_{k,t}$ in Relaxed Clock

Combining the results in the tables 4.1, 4.2, 4.3, 4.4 and 4.5, the Pareto diagrams in the figure 4.7 are produced.



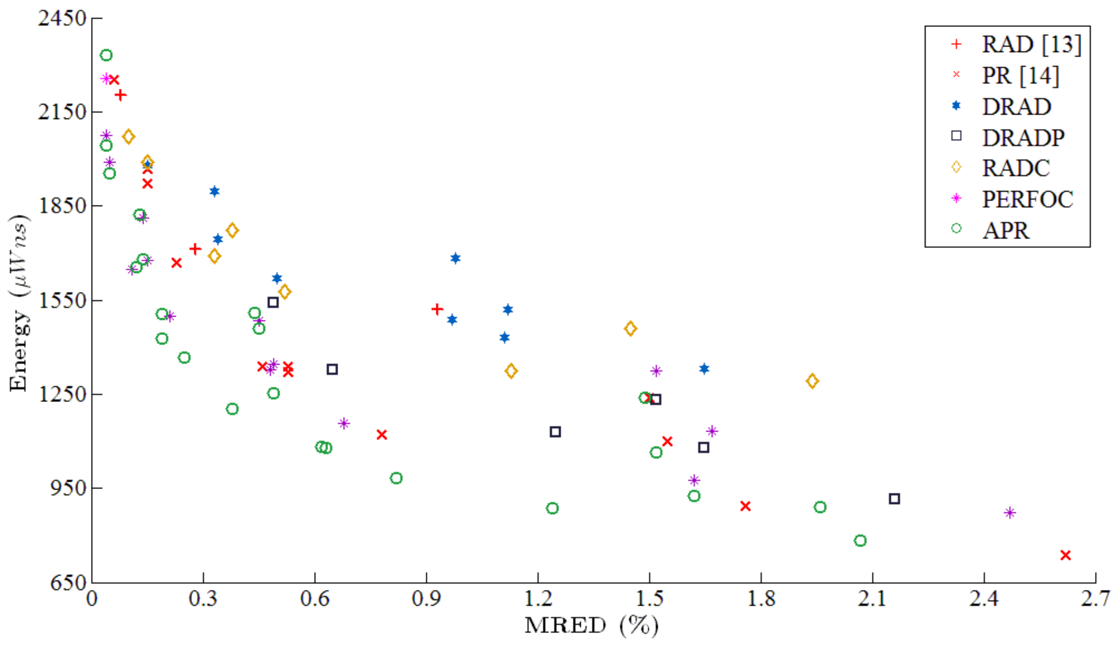
(a) Energy-Error



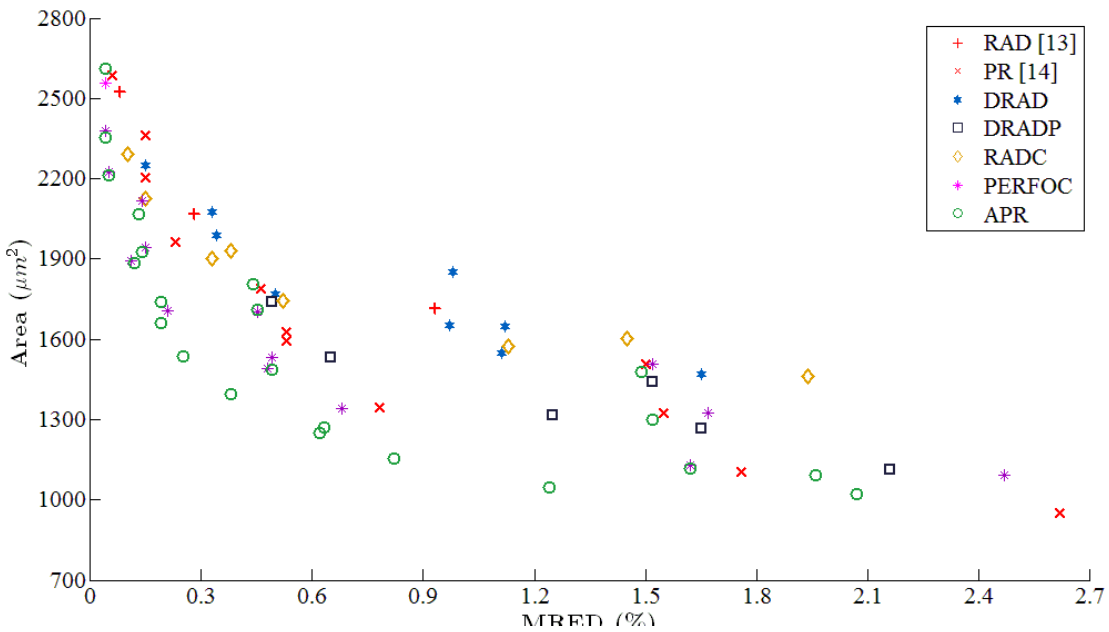
(b) Area-Error

Figure 4.7: Evaluation of the proposed approximate multipliers in Pareto diagrams, when synthesized and operating at their critical path delay.

Respectively, combining the results in the tables 4.6, 4.7, 4.8, 4.9 and 4.10 the Pareto diagrams in the figure 4.8 are produced.



(a) Energy-Error

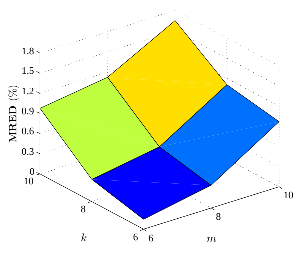


(b) Area-Error

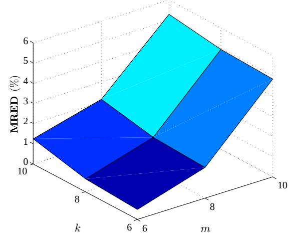
Figure 4.8: Evaluation of the proposed approximate multipliers in Pareto diagrams, when synthesized and operating at a relaxed clock.

Based on the above figures 4.7 and 4.8, it's easy to conclude that the best developed technique is the asymmetric perforation and rounding technique. It thrives in all categories (Energy and Area). The second best technique is the perforation with correction. In some cases, it is very close or even better than the APR technique. The others developed techniques are far behind due to lots of reasons. It seems that the approximations made, have an effect of high loss in accuracy without gaining much energy efficiency.

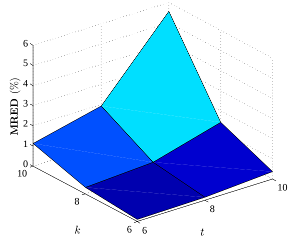
In the next figures 4.9 the diagrams present the changes in error when different values of the configurations parameters of each technique are applied.



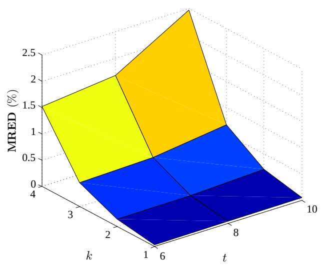
(a) DRAD$|_{2^k,2^m}$



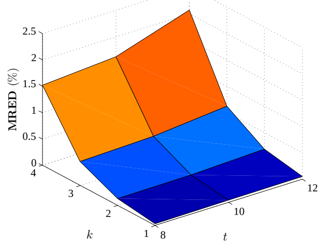
(b) DRADP$|_{2^k,2^m}$



(c) RADC$|_{2^k,t}$



(d) PERFOC$|_{k,t}$



(e) APR$|_{p,t}$

Figure 4.9: MRED variation w.r.t. approximation configuration parameters of 16×16 bit multipliers: (a) *Double High Radix* (b) *Double High Radix with Perforation* (c) *High Radix with Correction* (d) *Perforation with Correction* (e) *Asymmetric Perforation and Rounding*

Chapter 5

Conclusion and Future Work

The embedded and mobile nature of modern computing systems has led to an increased need for high performance and energy efficiency. As a result, since the failure of Dennard scaling, energy dissipation has become a first class concern in the design of integrated circuits. Towards this direction, approximate (or inexact) computing appears as an emerging and promising solution for energy-efficient systems design [5], exploiting the inherent error/noise resilience of various applications. More explicitly, perfect answers are often unnecessary (or do not exist) in a large number of application domains involving media processing, machine learning, data mining and statistics [2].

The topic of this diploma thesis was the exploration of new inexact techniques for energy-efficient approximate multiplication circuits. Therefore, five new methods of approximate multiplication were developed. The implementation of these methods was made in Verilog code. Next, the simulations for the configuration of the power consumption and error of each technique were performed with the help of Synopsys Design Compiler, Mentor Graphics ModelSim and Matlab. After all results were ready, the evaluation of the proposed multipliers was performed with the help of Energy-Error and Area-Error Pareto diagrams. The comparison between the multipliers was made, when they were synthesized and operating at their critical path delay. A second comparison between them was performed, when synthesized and operating at a relaxed clock. Finally, five diagrams were presented, where the error depending on the configuration parameters of each technique was displayed.

In this diploma thesis all the implementations were performed for a 16×16 multiplier. Future work could include the implementation of the developed techniques in order to form a 24×24 or a 32×32 multiplier. Generally, the implementation of these methods for a greater or lesser number of bits than 16 is pending. Another idea for future work could be the development of a technique similar to the one of high radix with correction. But instead of the used correction digits y_i', the rounding method of the asymmetric perforation and rounding technique could be applied.

Bibliography

[1] S. T. Chakradhar and A. Raghunathan. Best-effort computing: Re-thinking parallel software and hardware. In *Design Automation Conference*, pages 865–870, June 2010.

[2] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Design Automation Conference*, pages 1–9, May 2013.

[3] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Neural acceleration for general-purpose approximate programs. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 449–460, Dec 2012.

[4] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. Low-power digital signal processing using approximate adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124–137, Jan 2013.

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

[6] S. Hashemi, R. I. Bahar, and S. Reda. DRUM: A dynamic range unbiased multiplier for approximate applications. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 418–425, Nov 2015.

[7] H. Jiang, J. Han, and F. Lombardi. A comparative review and evaluation of approximate adders. In *Great Lakes Symposium on VLSI*, pages 343–348, May 2015.

[8] H. Jiang, J. Han, F. Qiao, and F. Lombardi. Approximate radix-8 booth multipliers for low-power and high-performance operation. *IEEE Transactions on Computers*, 65(8):2638–2644, Aug 2016.

[9] H. Jiang, J. Han, F. Qiao, and F. Lombardi. Approximate radix-8 booth multipliers for low-power and high-performance operation. *IEEE Transactions on Computers*, 65(8):2638–2644, Aug 2016.

[10] X. Jiao, Y. Jiang, A. Rahimi, and R. K. Gupta. Slot: A supervised learning model to predict dynamic timing errors of functional units. In *Design, Automation and Test in Europe*, pages 1183–1188, March 2017.

[11] M. Kamal, A. Ghasemazar, A. Afzali-Kusha, and M. Pedram. Improving efficiency of extensible processors by using approximate custom instructions. In *Design, Automation and Test in Europe*, pages 1–4, March 2014.

[12] E. J. King and E. E. Swartzlander. Data-dependent truncation scheme for parallel multipliers. In *Asilomar Conference on Signals, Systems and Computers*, pages 1178–1182, Nov 1997.

[13] V. Leon, G. Zervakis, D. Soudris, and K. Pekmestzi. Approximate hybrid high radix encoding for energy-efficient inexact multipliers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(3):421–430, March 2018.

[14] V. Leon, G. Zervakis, S. Xydis, D. Soudris, and K. Pekmestzi. Walking through the energy-error pareto frontier of approximate multipliers. *IEEE Micro*, 38(4):40–49, Jul-Aug 2018.

[15] J. Liang, J. Han, and F. Lombardi. New metrics for the reliability of approximate and probabilistic adders. *IEEE Transactions on Computers*, 62(9):1760–1771, Sept 2013.

[16] A. Lingamneni, C. Enz, K. Palem, and C. Piguet. Highly energy-efficient and quality-tunable inexact FFT accelerators. In *IEEE Custom Integrated Circuits Conference*, pages 1–4, Sept 2014.

[17] C. Liu, J. Han, and F. Lombardi. A low-power, high-performance approximate multiplier with configurable partial error recovery. In *Design, Automation and Test in Europe*, pages 1–4, March 2014.

[18] W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, and F. Lombardi. Design of approximate radix-4 booth multipliers for error-tolerant computing. *IEEE Transactions on Computers*, PP, 2017.

[19] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard. Quality of service profiling. In *ACM/IEEE International Conference on Software Engineering*, pages 25–34, May 2010.

[20] S. Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys*, 48(4), May 2016.

[21] A. Momeni, J. Han, P. Montuschi, and F. Lombardi. Design and analysis of approximate compressors for multiplication. *IEEE Transactions on Computers*, 64(4):984–994, Apr 2015.

[22] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim. Energy-efficient approximate multiplication for digital signal processing and classification applications. *IEEE Transactions on Very Large Scale Integration Systems*, 23(6):1180–1184, June 2015.

[23] R. Ragavan, B. Barrois, C. Killian, and O. Sentieys. Pushing the limits of voltage over-scaling for error-resilient applications. In *Design, Automation and Test in Europe*, pages 476–481, March 2017.

[24] K. M. Reddy, Y. B. N. Kumar, D. Sharma, and M. H. Vasantha. Low power, high speed error tolerant multiplier using approximate adders. In *International Symposium on VLSI Design and Test*, pages 1–6, June 2015.

[25] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. Enerj: Approximate data types for safe and general low-power computation. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 164–174, June 2011.

[26] M. J. Schulte and E. E. Swartzlander. Truncated multiplication with correction constant. In *IEEE Workshop on VLSI Signal Processing*, pages 388–396, Oct 1993.

[27] S. Venkatachalam, H. J. Lee, and S. Ko. Power efficient approximate booth multiplier. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2018.

[28] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Quality programmable vector processors for approximate computing. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, Dec 2013.

[29] Q. Xu, T. Mytkowicz, and N. S. Kim. Approximate computing: A survey. *IEEE Design Test*, 33(1):8–22, Feb 2016.

[30] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris, and K. Pekmestzi. Design-efficient approximate multiplication circuits through partial product perforation. *IEEE Transactions on Very Large Scale Integration Systems*, 24(10):3105–3117, Oct 2016.

[31] Z. Zhang and Y. He. A low-error energy-efficient fixed-width booth multiplier with sign-digit-based conditional probability estimation. *IEEE Transactions on Circuits and Systems II: Express Briefs*, pages 236–240, Feb 2018.

[32] Z. Zhang, Y. He, J. He, X. Yi, Q. Li, and B. Zhang. Optimal slope ranking: An approximate computing approach for circuit pruning. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2018.