Development of high performance hardware architectures for multimedia applications

Shafqat Khan

N° d'ordre : 4179 ANNÉE 2010

THÈSE / UNIVERSITÉ DE RENNES 1, FRANCE
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Traitement du Signal et Télécommunications

Ecole doctorale Matisse

présentée par

Shafqat KHAN

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

---

Développement d'architectures matérielles hautes performances pour les applications multimédia

Development of high performance hardware architectures for multimedia applications

Thèse soutenue à Lannion (France)
le Mercredi 29 Septembre 2010

devant le jury composé de :

Patrice QUINTON
Professeur des Universités
Université de Rennes 1 / président

Michel AUGUIN
Directeur de recherche CNRS
Université de Nice Sophia Antipolis / rapporteur

Christophe JÉGO
Maître de conférences
Télécom Bretagne / rapporteur

Philippe COUSSY
Maître de conférences
Université de Bretagne Sud / examinateur

Emmanuel CASSEAU
Professeur des Universités
Université de Rennes 1 / directeur de thèse

Daniel MÉNARD
Maître de conférences
Université de Rennes 1 / co-directeur de thèse

Résumé

Les besoins en puissance de calcul des processeurs sont en constante augmentation en raison de l'importance croissante des applications multimédia dans la vie courante. Ces applications requièrent de nombreux calculs avec des données de faible précision généralement issues des pixels. Le moyen le plus efficace pour exploiter le parallélisme de données de ces applications est le parallélisme dit de sous-mots (SWP pour *subword parallelism*). Les opérations sont effectuées en parallèle sur des données de faible précision regroupées ce qui permet d'utiliser au mieux les ressources disponibles dimensionnées pour traiter des mots. Dans cette thèse, la conception de différents opérateurs SWP pour les applications multimédia est proposée. Une bonne adéquation entre largeur des sous-mots et largeur des données manipulées permet une meilleure utilisation des ressources et conduit ainsi à améliorer l'efficacité de l'exécution de l'application sur le processeur. Les opérateurs arithmétiques de base développés sont ensuite utilisés dans un opérateur SWP reconfigurable. Ce dernier peut être configuré pour effectuer diverses opérations multimédia avec différentes largeurs de données. L'opérateur reconfigurable peut être utilisé comme unité spécialisée ou comme co-processeur dans un processeur multimédia afin d'en améliorer les performances. La vitesse interne des différentes unités de traitement est également améliorée en représentant les nombres en système redondant plutôt qu'en système binaire. Le système redondant permet entre autre d'augmenter la vitesse des opérations arithmétiques en évitant une propagation de retenue couteuse lors d'opérations d'addition. Les résultats obtenus montrent l'intérêt en terme de performances d'utiliser des opérateurs SWP lors de l'exécution d'applications multimédia.

Mots clés : applications multimédia, conception d'opérateurs VLSI, parallélisme dit de sous-mots (*SWP : subword parallelism*).

Abstract

The computational requirements of the processors are increasing tremendously with the increase in the complexity of applications. Among these applications, multimedia represents the class of applications which requires lot of computations on low precision pixels. These applications include motion estimation, discrete cosine transform, image filtering etc. The processing requirements of multimedia applications can be fulfilled by performing parallel computations on input pixel data. Subword parallelism (SWP) is one of the best options to exploit data level parallelism that exist in the applications. In SWP, rather than wasting the word oriented data path, parallel operations are executed on packed subwords. SWP increases the performance of the processor especially for multimedia applications with low precision pixel data. Coordination between pixel sizes in multimedia applications and subword sizes in SWP operators further increases the performance through a better resource utilization. In this thesis, reconfigurable SWP arithmetic operators are proposed for multimedia applications. In the proposed basic SWP operators, parallelism is obtained by using multimedia oriented subword sizes (8, 10, 12 or 16-bit) rather than classical subword sizes (8, 16 or 32-bit etc.). Compared to classical SWP operators, the multimedia SWP operator utilizes the available resources more efficiently when working on different video applications. SWP arithmetic operators are then used to design reconfigurable operators for multimedia applications. In the proposed reconfigurable operators, reconfiguration is provided at both the data size level and the operation level without any reconfiguration time overheads. These operators can perform a variety of basis as well as multimedia operations on different size pixel data. These operators can be used as co-processors to enhance the performance for multimedia applications. Along with parallelism, the internal computational speed of the different arithmetic units is improved by introducing the redundant number system in the SWP architectures. Redundant number system provides a carry propagation free addition which ultimately increases the speed of different arithmetic operations. The performance of SWP operators are verified on different multimedia kernels.

Key words : multimedia applications, VLSI design of operators, subword parallelism (*SWP : subword parallelism*).

Acknowledgements

I wish to acknowledge and thank all those people who supported me during my PhD work. It would not have been possible to complete this PhD thesis without the help and support of kind people around me.

First of all i would like to express my sincere gratitude to my advisor Prof. Dr. Emmanuel Casseau for his continuous support throughout my PhD work. I am extremely indebted for his time, suggestions and remarks that have always been constructive and productive. He guided me to approach the research problems and analyze the results in a logical way. It has been an honour for me to work with him. I would also like to thank Dr. Daniel Menard for all his help and support as my co-advisor. His constructive feedback and comments at various stages have been significantly useful in shaping the thesis up to the completion.

I take this opportunity to express my special gratitude towards all the member of IRISA/CAIRN Lab, Lannion France [22] who made my stay an outstanding experience. It is the moment of great pride for me that i have completed my PhD in this prestigious Lab. I would like to thanks the head of Lab, Prof. Dr. Olivier Sentieys for his generous help and kind attitude in the hour of need. I am also grateful to Dr. Arnaud Tisserand for his help on redundant number systems and reading the manuscript of our article in IJISCE'10 [51].

I am equally thankful to all the members of the jury who evaluated this work: Michel Auguin (Director of research CNRS, LEAT, Nice France), Christophe Jego (Assistant Professor at Telecom Bretagne, Brest France), Patrice Quinton (Director of ENS Cachan antenne de Bretagne France) and Philippe Coussy (Assistant Professor at LESTER Lorient France).

I also thank my fellow PhD students Tuong, Pasha, Manh, Jamal, Umer, Jullien, Renaud, Vingh, Naeem, Vivik, Karthick, Robin, Antoine, Cécile, Mahtab, and Hai-Nam for making a very good research and working environment in the Lab. I still regret for not being able to mention all those i met during my PhD.

Finally i would like to thank all my family members. I am forever indebted to my parents for their deep hearted prayers and never ending support. Their love and prayers helped me remain focused during my studies. The encouragement and support from my wife is always a source of inspiration and energy for me. This PhD thesis and our stay in France will always be memorable for us because our three kids (Maryam, Mubashir and Mudassir) are also born during this period.

Contents

Acknowledgements iii

List of Figures ix

List of Tables xii

Abbreviations xiv

Résumé étendu 1
- 0.1 Le parallélisme pour améliorer les performances . . . 1
  - 0.1.1 Le parallélisme au niveau des instructions . . . 2
  - 0.1.2 Le parallélisme au niveau des données . . . 2
- 0.2 Parallelisme de sous-mots (*subword paralelism* (SWP)) . . . 3
  - 0.2.1 Utilisation du concept SWP . . . 4
  - 0.2.2 Concept SWP et applications multimédia . . . 6
  - 0.2.3 Extensions multimédia dans les microprocesseurs à usage général 6
- 0.3 Besoins associés au concept SWP . . . 7
  - 0.3.1 Opérateurs SWP . . . 7
  - 0.3.2 Transferts de données entre les unités de traitement et la mémoire 8
  - 0.3.3 Disponibilité de données de faible précision . . . 9
- 0.4 Largeurs des sous-mots dans les architectures SWP . . . 9
  - 0.4.1 Parallélisme versus largeur des sous-mots . . . 9
  - 0.4.2 Largeurs des sous-mots supportées . . . 10
  - 0.4.3 Adéquation entre largeur des données et largeur des sous-mots . . 10
  - 0.4.4 Largeur de sous-mots conventionnelles . . . 11
  - 0.4.5 Largeurs de sous-mots orientées multimédia . . . 11
  - 0.4.6 Largeur des mots avec les opérateurs SWP . . . 13
- 0.5 Contributions et organisation du mémoire de thèse . . . 13
  - 0.5.1 Organisation de la thèse . . . 16

1 Subword Parallelism SWP in operator design 20
- 1.1 Parallelism for performance enhancement . . . 21
  - 1.1.1 Instruction level parallelism . . . 21
  - 1.1.2 Data level parallelism . . . 22
- 1.2 Multimedia Processing . . . 24

1.3 Subword parallelism SWP . . . 25
1.3.1 Utilization of data level parallelism in SWP . . . 26
1.3.2 SWP in multimedia application . . . 28
1.3.3 Multimedia extension in general purpose microprocessors . . . 29
1.3.4 SWP building block IPs . . . 31
1.4 SWP requirements . . . 32
1.4.1 SWP operators . . . 32
1.4.2 Data transfer between processing units and memory . . . 33
1.4.3 Availability of low precision data . . . 34
1.5 SWP instruction set . . . 34
1.5.1 Parallel ADD and SUB instruction . . . 34
1.5.2 SWP instructions to avoid overflow in MAX-2 . . . 35
1.5.3 MIX instruction in MAX-2 . . . 37
1.5.4 PERMUTE instruction in MAX-2 . . . 39
1.5.5 Memory instructions . . . 40
1.5.6 Assembly code with and without SWP instructions . . . 40
1.6 Subwords sizes in SWP architectures . . . 41
1.6.1 Parallelism Vs subword size . . . 41
1.6.2 Support for multiple subword sizes . . . 41
1.6.3 Coordination between data and subword size . . . 42
1.6.4 Classical subword sizes . . . 42
1.6.5 Multimedia subword sizes . . . 43
1.7 Word size in SWP operators . . . 44
1.8 Limitations of SWP . . . 47
1.9 Contributions and Organization of this Thesis . . . 48
1.9.1 Organization of thesis . . . 51
1.10 Conclusions . . . 54

2 Design of SWP basic operators 55
2.1 SWP operator design . . . 56
2.1.1 Complexity of SWP operators . . . 56
2.2 Add operator . . . 57
2.2.1 Classical SWP adder . . . 58
2.2.2 Multimedia SWP adder . . . 62
2.3 Multiply operator . . . 65
2.3.1 Classical SWP multiplier . . . 67
2.3.2 Multimedia SWP multiplier . . . 69
2.3.2.1 Dedicated PP generation units . . . 71
2.3.2.2 Generalize PP generation unit for SWP multimedia multiplier . . . 71
2.3.2.3 Addition of partial products for SWP multimedia multiplier . . . 75
2.3.2.4 Comparison of simple and SWP multimedia multiplier . . . 76
2.3.3 Analysis of SWP multipliers . . . 76
2.4 MAC operator . . . 80
2.4.1 Classical SWP MAC . . . 80
2.4.2 Multimedia SWP MAC . . . 81

2.5 Conclusions . . . 82

3 SWP in multimedia operations 83
3.1 Multimedia arithmetic operations . . . 84
3.1.1 Sum of absolute values of difference SAD . . . 84
3.2 Determination of absolute value of difference |a - b| . . . 86
3.2.1 Absolute value of difference : Method 1 . . . 86
3.2.1.1 SWP Absolute value of difference : Method 1 . . . 88
3.2.2 Absolute value of difference : Method 2 . . . 89
3.2.2.1 SWP Absolute value of difference : Method 2 . . . 91
3.2.3 Absolute value of difference : Method 3 . . . 92
3.2.3.1 SWP Absolute value of difference : Method 3 . . . 92
3.2.4 Comparison of SWP Absolute value of difference operators . . . 93
3.3 SWP SAD operator . . . 94
3.3.1 Comparison of simple and SWP SAD operator . . . 96
3.4 Sum of products (SOP) . . . 97
3.4.1 SWP sum of products . . . 99
3.4.2 Comparison of simple and SWP sum of products operator . . . 101
3.5 Sum of additions/subtractions . . . 102
3.5.1 SWP sum of additions/subtractions . . . 103
3.5.2 Comparison of simple and SWP $\sum(a \pm b)$ operators . . . 105
3.6 Conclusions . . . 106

4 Reconfigurable SWP operator for multimedia processing 107
4.1 Reconfigurable architectures . . . 108
4.1.1 Reconfiguration at interconnection level . . . 108
4.1.2 Reconfiguration at operator's level . . . 109
4.1.3 Reconfigurability using SWP . . . 109
4.2 SWP Reconfigurable multimedia operator . . . 110
4.2.1 Architecture of SWP Reconfigurable operator . . . 110
4.2.2 Connectivity of reconfigurable operator with other operators . . . 111
4.2.3 Building blocks of reconfigurable operator . . . 113
4.3 Basic SWP arithmetic units . . . 113
4.3.1 SWP ADD and SUB units . . . 114
4.3.2 SWP Absolute signed . . . 115
4.3.3 SWP multiplier unit . . . 116
4.3.4 SWP $|a - b|$ unsigned . . . 118
4.3.5 Accumulator unit . . . 118
4.4 Subword alignment and interconnection units . . . 118
4.4.1 Bit conversion units . . . 119
4.4.2 SWP subword adders units . . . 120
4.4.3 Multiplexer units . . . 121
4.5 Complex multimedia operations . . . 121
4.5.1 SWP $\sum(a \times b)$ operation . . . 122
4.5.2 SWP $\sum(|a - b|)$ operation . . . 123
4.5.3 SWP $\sum(a + b)$ signed operation . . . 124
4.5.4 Other complex operations . . . 125

4.6 Synthesis results . . . 126
4.6.1 Statistical power analysis . . . 128
4.7 Performance on multimedia applications . . . 130
4.8 Conclusion . . . 131

5 SWP using redundant representation 132
5.1 Number systems . . . 133
5.1.1 Binary number system . . . 133
5.1.2 Redundant number system . . . 134
5.2 Addition using BS number system . . . 136
5.2.1 Addition tables for BS numbers . . . 136
5.2.1.1 Addition table using direct method . . . 137
5.2.1.2 Addition table using internal barrow . . . 139
5.2.2 Addition of intermediate sum and carry digits . . . 139
5.3 Logic cell for BS digits addition . . . 140
5.3.1 Adder using BS logic cell . . . 143
5.3.2 Comparison of BS adder with other adder types . . . 143
5.4 Conversions between CB and BS numbers . . . 145
5.4.1 Conversions from CB to BS . . . 146
5.4.2 Conversions from BS to CB representation . . . 148
5.5 Multiplication using BS number system . . . 150
5.5.1 Comparison of CB and BS multiplier . . . 152
5.6 FSM based variable length BS adder . . . 153
5.6.1 FSM controller . . . 154
5.7 SWP using BS representation . . . 157
5.7.1 SWP adder using BS representation . . . 158
5.7.1.1 Comparison of SWP BS adder with SWP CB adder . . . 159
5.7.1.2 Comparison of simple and SWP BS adder . . . 160
5.7.2 SWP multiplier using BS representation . . . 160
5.7.2.1 Comparison of SWP BS multiplier with SWP CB multiplier 161
5.7.2.2 Comparison of simple and SWP BS multiplier . . . 162
5.8 SWP SAD using BS representation . . . 162
5.9 SWP BS conversions . . . 164
5.9.1 SWP CB to BS conversion . . . 165
5.9.2 SWP BS to CB conversion . . . 165
5.10 High speed reconfigurable multimedia operator . . . 166
5.10.1 Architecture of the operator . . . 167
5.10.2 Sum of products using reconfigurable operator . . . 168
5.10.3 Synthesis results . . . 170
5.10.3.1 Power analysis . . . 171
5.11 Conclusions . . . 173

6 Motion estimation using SWP operators 174
6.1 Motion estimation . . . 175
6.2 Search algorithms in motion estimation . . . 177
6.2.1 Full search algorithm . . . 177
6.2.2 Three step search algorithm . . . 178

6.2.3 Diamond search algorithm . . . 179
6.3 Cost functions . . . 180
6.3.1 Sum of absolute value difference SAD . . . 180
6.4 Motion estimation using SWP operators . . . 182
6.4.1 RAMs for search area image and current block . . . 183
6.4.2 RAM reading units . . . 183
6.4.3 SWP SAD computation unit . . . 184
6.4.4 SAD comparator unit . . . 185
6.4.5 State machine controller . . . 186
6.5 SWP ME using Full search algorithm . . . 189
6.5.1 Search time . . . 190
6.5.2 Synthesis results . . . 191
6.5.3 Comparison of simple and SWP ME operator . . . 193
6.6 SWP ME using Diamond search algorithm . . . 195
6.6.1 Search time . . . 197
6.6.2 Synthesis results . . . 198
6.6.3 Comparison of simple and SWP ME operator . . . 199
6.7 Comparison of FS and DS SWP ME operators . . . 200
6.8 SWP ME IP core . . . 201
6.9 Conclusions . . . 203

7 Conclusion and Perspectives **205**
7.1 Conclusion . . . 205
7.2 Future perspectives . . . 207

Bibliography **209**

Personal Publications **209**

List of Figures

1 Principe de l'additionneur SWP . . . 3
2 Implémentation de la boucle sur un processeur sans SWP . . . 4
3 Implémentation de la boucle sur un processeur avec SWP . . . 5
4 Opérateur SWP . . . 8
5 Utilisation de données 12 bits avec des sous-mots de largeur 16 bits . . . 12
6 Utilisation de données 12 bits avec des sous-mots de largeur 12 bits . . . 12
7 Block diagram of ROMA processor . . . 14

1.1 Architecture of SIMD processor . . . 23
1.2 Parallel subword ADD instruction . . . 26
1.3 Loop implementation on processor without SWP . . . 27
1.4 Loop implementation on processor with SWP . . . 28
1.5 Block diagram of SWP operator . . . 32
1.6 Parallel addition and subtraction in MAX-2 . . . 35
1.7 SAD operation using SWP instructions . . . 36
1.8 Finding of greater subwords values . . . 36
1.9 Different subword arrangements using *Mix* instruction . . . 37
1.10 Two dimension DCT transform . . . 38
1.11 Matrix transpose using *Mix* instruction . . . 38
1.12 Different permutations of subwords . . . 39
1.13 Classical subword sizes in SWP . . . 43
1.14 Multimedia oriented subword sizes in SWP . . . 44
1.15 Utilization of different word length processors for different multimedia oriented subword sizes . . . 46
1.16 Block diagram of ROMA processor . . . 49

2.1 Ripple carry adder using FA . . . 58
2.2 SWP Enabled Adder Architecture . . . 59
2.3 SWP adder architecture using group CLA . . . 60
2.4 Word and subword sizes supported by multimedia SWP adder . . . 62
2.5 SWP multimedia adder architecture . . . 63
2.6 swp multimedia adder using group CLA . . . 64
2.7 PPs generation using array of AND gates . . . 65
2.8 Booth recoding of multiplier bits . . . 66
2.9 Block diagram of vector multiplication . . . 67
2.10 SWP multiplication . . . 67
2.11 Arrangement of partial products for different subword sizes . . . 70
2.12 SWP partial product generation using dedicated units . . . 71

2.13 Valid bits of PP0 for different subword sizes . . . 72
2.14 Arrangement of PPs for 8-bit data size . . . 75
2.15 Area of SWP multipliers . . . 79
2.16 CP of SWP multipliers . . . 79
2.17 Power consumption of SWP MULTs . . . 79

3.1 Sum of absolute difference operator . . . 85
3.2 Absolute value of difference unit . . . 87
3.3 SWP Absolute value of difference operator . . . 88
3.4 Comparison and inversion of smaller input . . . 90
3.5 Absolute difference unit . . . 92
3.6 Comparison of SWP absolute difference operators . . . 93
3.7 SWP sum of absolute value difference SAD . . . 94
3.8 SWP subword adder unit for SWP SAD operator . . . 95
3.9 Pipelined architecture of sum of products operator . . . 98
3.10 SWP sum of product operator . . . 99
3.11 SWP subword adder unit for SWP SOP operator . . . 100
3.12 Sum of addition/subtraction operator . . . 102
3.13 SWP sum of addition/subtraction operator . . . 103
3.14 Arrangement of over flow bits for different subword sizes . . . 104
3.15 SWP subword adder unit for SWP $\sum(a \pm b)$ operator . . . 105

4.1 Reconfiguration at interconnection level . . . 108
4.2 SWP reconfigurable multimedia operator . . . 111
4.3 Inputs and outputs of SWP reconfigurable operator . . . 112
4.4 *SWP* $(a \pm b)$ operation using reconfigurable operator . . . 114
4.5 SWP absolute operation for signed subwords . . . 115
4.6 SWP product subword MSB and LSB part extractor . . . 117
4.7 SWP multiplication using reconfigurable operator . . . 117
4.8 Expansion of unsigned subwords . . . 119
4.9 Computation of SWP $\sum(a \times b)$ using reconfigurable multimedia operator 122
4.10 Computation of SWP $\sum|a - b|$ using reconfigurable multimedia operator 123
4.11 Computation of SWP $\sum(a + b)$ using reconfigurable multimedia operator 124
4.12 Synthesis at different clock periods . . . 127
4.13 Statistical power estimation . . . 128
4.14 Power consumption of SWP operations . . . 129

5.1 Block diagram of BS number addition . . . 136
5.2 Addition of BS numbers using addition Table 5.1 . . . 140
5.3 Addition of BS numbers using addition Table 5.2 . . . 140
5.4 Logic cell for BS digits addition . . . 142
5.5 Comparison of CB and BS adders . . . 144
5.6 Conversion between CB and BS numbers . . . 145
5.7 Conversion from BS to CB representation . . . 148
5.8 BS to CB conversion . . . 149
5.9 Multiplication using BS number system . . . 151
5.10 FSM based BS adder . . . 154
5.11 State diagram of adder . . . 155

5.12 SWP BS ADD architecture . . . 158
5.13 SWP adder for 8-digit subwords . . . 159
5.14 SWP BS SAD unit . . . 163
5.15 SWP BS $|a-b|$ unit . . . 164
5.16 SWP CB to BS conversion . . . 165
5.17 SWP BS to CB conversion . . . 166
5.18 Reconfigurable multimedia operator . . . 168
5.19 Sum of product computation using high speed reconfigurable operator . . . 169
5.20 Comparison of SWP operators using CB and BS representations . . . 171
5.21 Power consumption of operations . . . 172

6.1 Block matching in motion estimation algorithm . . . 175
6.2 Three step search algorithm . . . 178
6.3 Diamond search algorithm . . . 179
6.4 SWP SAD unit . . . 181
6.5 SWP motion estimation (ME) operator . . . 183
6.6 Block diagram of SWP SAD unit . . . 184
6.7 Block diagram of comparator unit . . . 185
6.8 State machine controller block . . . 186
6.9 Sequence of operations in the SWP ME operator . . . 188
6.10 Pixel values of search area image and current block . . . 190
6.11 8-bit motion estimation operator . . . 193
6.12 Comparison of five 8-bit ME operators and SWP ME operator using FS . . . 195
6.13 SWP ME operator using the diamond search algorithm . . . 196
6.14 Number of cycles required by FS and DS algorithms to find the best match 197
6.15 Comparison of five 8-bit ME operators and SWP ME operator using DS . . . 199
6.16 Comparison of SWP ME operator using different search algorithm . . . 200
6.17 Soft embedded processor . . . 202

List of Tables

1.1 TMS320C64x+ SWP instruction set . . . 30
1.2 Data sizes which can be manipulated by different DSPs offering SWP capabilities for arithmetic operations . . . 31
1.3 utilization of SWP operator for classical and multimedia subword sizes . . . 45

2.1 Internal control bits for different subword sizes . . . 59
2.2 Synthesis of classical SWP adders . . . 61
2.3 Internal control bits combinations for SWP multimedia adder . . . 63
2.4 Results of multimedia SWP adders . . . 64
2.5 PP generation using Booth recoding . . . 66
2.6 Results of classical SWP multiplier . . . 68
2.7 Results of multimedia SWP Multiplier . . . 76
2.8 Configurations of word and subwords sizes . . . 77
2.9 Synthesis results for SWP multipliers . . . 78
2.10 Results of classical SWP MAC . . . 81
2.11 Results of multimedia SWP MAC . . . 82

3.1 Method 1: Synthesis results of absolute difference operator . . . 89
3.2 Method 2: Synthesis results of absolute difference operator . . . 91
3.3 Method 3: Synthesis results of absolute difference operator . . . 93
3.4 Synthesis results of SAD operator . . . 96
3.5 Synthesis results of sum of products operator . . . 101
3.6 Synthesis results of $\sum(a \pm b)$ operator . . . 105

4.1 Synthesis on ASIC technologies . . . 126
4.2 Percentage reduction in number of cycles . . . 130

5.1 Addition table for the BS numbers . . . 137
5.2 Addition table for the BS numbers using internal barrow . . . 139
5.3 Synthesis results of BS adders . . . 143
5.4 Synthesis results of CB to BS digit converters . . . 147
5.5 BS to CB conversion rules . . . 148
5.6 Synthesis results of BS to CB converter . . . 149
5.7 Comparison of CB and BS multipliers . . . 152
5.8 Synthesis results of FSM based BS adders . . . 156
5.9 Synthesis results of simple and SWP BS adder . . . 160
5.10 Synthesis results of simple and SWP BS multiplier . . . 162

6.1 Cycles required by the SWP ME operator to find the best match using FS algorithm 191
6.2 Synthesis of SWP ME operator using FS algorithm 192
6.3 Synthesis of SWP ME operator using DS algorithm 198

Abbreviations

ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
AVX	**A**dvanced **V**ector e**X**tensions
BS	**B**arrow **S**ave
CP	**C**ritical **P**ath
CLK	**CL**oc**K**
CLB	**C**onfigurable **L**ogic **B**lock
CLA	**C**arry **L**ook **A**head
CSA	**C**arry **S**ave **A**dder
CB	**C**onventional **B**inary
DC	**D**esign **C**ompiler
DSP	**D**igital **S**ignal e**P**rocessing
DLP	**D**ata **L**evel **P**arallelism
DS	**D**iamond **S**earch
DCT	**D**iscrete **C**osine **T**ransform
DWT	**D**iscrete **W**avelet **T**ransform
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
FSM	**F**init **S**tate **M**achine
FA	**F**ull **A**dder
FS	**F**ull **S**earch
GPP	**G**eneral **P**urpose **P**rocessor
HA	**H**alf **A**dder
ILP	**I**nstruction **L**evel **P**arallelism
IP	**I**ntellectual **P**roperty
LSB	**L**east **S**ignificant **M**ultiple **B**it
LDSP	**L**arge **D**iamond **S**earch **P**attern

ME	**M**otion **E**stimation
MSB	**M**ost **S**ignificant **B**it
MV	**M**otion **V**ector
MAC	**M**ultiply and **AC**cumulate
MMX	**M**atrix **M**ath e**X**tension
MAX	**M**ultimedia **A**cceleration e**X**tension
MIMD	**M**ultiple **I**nstruction **M**ultiple **D**ata
MAD	**M**ean value of **A**bsolute **D**ifference
PE	**P**rocessing **E**lement
RISC	**R**educed **I**nstruction **S**et **C**omputer
ROM	**R**ead **O**nly **M**emory
RTL	**R**egister **T**ransfer **L**evel
RCA	**R**ipple **C**arry **A**dder
SWP	**S**ub **W**ord **P**arallelism
SAD	**S**um of **A**bsolute value **D**ifference
SOP	**S**um **O**f **P**roducts
SOC	**S**ystem **O**n **C**hip
SIMD	**S**ingle **I**nstruction **M**ultiple **D**ata
SPMD	**S**ingle **P**rogram **M**ultiple **D**ata
SSE	**S**tream **S**IMD e**X**tension
SDSP	**S**mall **D**iamond **S**earch **P**attern
TSS	**T**hree **S**tep **S**earch
TI	**T**exas **I**nstruments
VIS	**V**isual **I**nstruction **S**et
UMC	**U**nited **M**icroelectronics **C**orporation

Dedicated to my kids Maryam, Mubashir and Mudassir

Résumé étendu

Les besoins en puissance de calcul des processeurs sont en constante augmentation en raison de l'importance croissante des applications multimédias. Ces applications requièrent de nombreux calculs avec des données de faible précision (pixels). Un des moyens possibles pour répondre aux exigences de calcul des applications multimédias est d'utiliser pleinement les ressources disponibles du processeur. La plupart des processeurs fonctionnent avec des mots de données de grande largeur alors que dans les applications multimédia les calculs sont effectués sur des données de petite largeur. Aussi, avec des processeurs conventionnels, l'utilisation maximale des ressources (opérateurs, registres et autres éléments d'interconnexion) n'est pas possible lorsque les données sont de type multimédia. Pour améliorer les performances, une solution consiste à effectuer des calculs sur plusieurs données de faible précision en parallèle tout en conservant un chemin de données axé sur le mot processeur. Cela peut être possible en utilisant le concept de parallélisme dit *subword* (SWP : subword parallelism) [30] [31] [48]. En SWP, les opérations sont effectuées en parallèle sur des données de faible précision (*subwords* pour sous-mots) regroupées afin de ne pas gaspiller les ressources dimensionnées pour des mots. Une bonne adéquation entre les données manipulées et les dimensions des sous-mots est nécessaire pour utiliser efficacement les ressources disponibles.

0.1 Le parallélisme pour améliorer les performances

Les performances d'un processeur peuvent être mesurées à partir de la quantité de temps et les ressources nécessaires pour exécuter une tâche particulière. Si le processeur peut accomplir la tâche en un minimum de temps avec un minimum de ressources matérielles, alors ses performances seront élevées. Cependant, en pratique, il y a habituellement un compromis entre la vitesse et les ressources nécessaires. Le principe pour augmenter la vitesse du processeur est d'utiliser le parallélisme qui existe à différents niveaux [36, 39, 40]. Le parallélisme est une des techniques les plus anciennes et efficaces pour augmenter les performances d'un système informatique. Pour exploiter pleinement le parallélisme, des unités de traitement parallèles sont nécessaires pour effectuer des opérations en

parallèle. Ces unités de traitement parallèles permettent d'augmenter la vitesse mais la surface des unités augmente en conséquence. Le parallélisme peut être appliqué à différents niveaux. Les niveaux les plus importants sont le parallélisme d'instructions et le parallélisme au niveau des données.

0.1.1 Le parallélisme au niveau des instructions

Le parallélisme au niveau des instructions (ILP: Instruction Level Parallelism) est une mesure de nombre d'instructions dans un programme qui peuvent être exécutées simultanément. Toutefois, en raison des dépendances de données et de contrôle, il n'est pas toujours facile d'exécuter les instructions en parallèle. Il existe plusieurs techniques pour exploiter l'ILP. Ces techniques comprennent entre autre l'exécution spéculative, la prédiction de branchement, le pipeline d'instruction, etc.

0.1.2 Le parallélisme au niveau des données

Le parallélisme au niveau des données tire partie du parallélisme intrinsèque qui existe dans le calcul des données et qui peut généralement être appliqué chaque fois que les calculs sont réguliers et répétitifs. La principale condition du parallélisme de données est que les données doivent toujours être disponibles pour les calculs. Le processeur alloue des portions de données à différentes unités de traitement pour effectuer les calculs. Ce faisant, les calculs sont effectués en parallèle et les tâches sont accomplies en moins de temps. Il existe plusieurs architectures pour exploiter le parallélisme au niveau des données: SIMD (single instruction multiple data), MIMD (multiple instruction multiple data), etc.

Le moyen le plus efficace pour exploiter le parallélisme de données est le parallélisme *subword* (SWP : subword parallelism) [48, 59, 60]. Le principe permet d'augmenter les performances du processeur via le traitement en parallèle d'éléments de données. Le SWP est parfois appelé *small SIMD* [30]. Au lieu d'utiliser plusieurs unités de traitement distinctes, plusieurs opérations identiques sont effectuées en même temps sur un opérateur SWP. Un opérateur SWP effectue les mêmes opérations sur des éléments de données regroupés dans les registres d'entrée. En général, un opérateur SWP peut manipuler plusieurs largeurs d'élément de données (sous-mots). Ainsi, les ressources du processeur peuvent être utilisées plus efficacement lorsque la largeur des données manipulées est inférieure à la largeur des mots processeur. Ainsi le concept SWP peut être utilisé pour augmenter les performances des processeurs pour de nombreuses applications, notamment pour les applications multimédias.

0.2 Parallelisme de sous-mots (*subword paralelism* (SWP))

Avec le concept SWP, des calculs parallèles sont effectués sur des données de faible précision qui sont regroupées dans des registres dont la largeur est celle d'un mot [30] [48]. Ces données de faible précision sont des éléments appelés *subwords* (sous-mots). Dans le chemin de données, les unités de calcul effectuent plusieurs calculs sur plusieurs sous-mots regroupés. L'exécutions en parallèle est initiée par une seule instruction. Le nombre de sous-mots qui peuvent être regroupés dépend de la largeur de sous-mots sélectionnée. Une utilisation efficace des ressources du processeur, comme les opérateurs, les registres et les éléments d'interconnexion, est favorisée par le concept SWP. Les unités de calcul sont conçues pour mettre en oeuvre le concept SWP. La Figure 1 présente l'exemple de l'additionneur SWP (additionneur 64 bits ici). Cet additionneur est similaire à un additionneur classique qui accepte deux opérantes de 64 bits et qui produit un résultat sur 64 bits. En bloquant la propagation des retenues aux frontières des sous-mots, plusieurs additions de faible précision peuvent être effectuées.

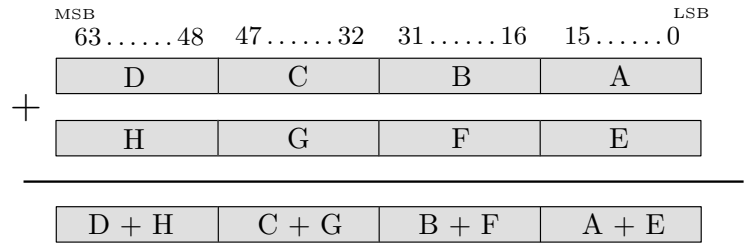

FIGURE 1: Principe de l'additionneur SWP

Ainsi, dans la Figure 1, quatre additions 16 bits sont effectuées en bloquant la propagation de la retenue des bits 15 à 16, 31 à 32 et 47 à 48. Les quatre additions sont effectuées simultanément sur des sous-mots de largeur 16 bits regroupés dans des registres de 64 bits. Cet additionneur prend en charge uniquement des sous-mots de largeur 16 bits. Plusieurs largeurs de sous-mots peuvent être cependant prises en compte. L'accélération du processeur dépend du nombre de sous-mots traités simultanément. Avec des données 16 bits, la vitesse de l'additionneur SWP présenté Figure 1 est quatre fois plus rapide a priori que celle d'un additionneur classique 64 bits. En pratique, la vitesse réelle d'un processeur SWP est généralement inférieure en raison d'un surcout dû à la mise en oeuvre du principe SWP, comme l'arrangement et les alignements des sous-mots avant les calculs.

0.2.1 Utilisation du concept SWP

Le concept SWP utilise le parallélisme au niveau des données. Généralement, lors de calculs parallèles, un morceau de code est exécuté à plusieurs reprises sur des données différentes. Prenons l'exemple de code qui suit en langage de haut niveau d'une boucle qui exécute le même calcul (addition) 100 fois. 'A', 'B' et 'C' sont trois tableaux qui peuvent stocker 100 éléments chacun. Chaque élément du tableau a une largeur de 16 bits. Les tableaux 'A' et 'B' contiennent les éléments d'entrée. Le tableau 'C' est utilisé pour stocker le résultat.

```
short A[100], B[100], C[100];
int i;
  for (i = 0; i < 100, i++)
    C[i] = A [i] + B[i];
  end for
```

Supposons que la machine sur laquelle ce code doit être exécuté est un processeur 64 bits (64 bits pour le chemin de données et les opérateurs, etc.). Si le processeur n'est pas de type SWP, il effectuera 100 additions en instanciant un opérateur *64-bit ADD* 100 fois. Au cours de chaque itération, les données d'entrée 16 bits des valeurs A [i] et B [i] sont stockées dans deux registres internes 64 bit X et Y (avec extension de signe si nécessaire). Puis l'opérateur *64-bit ADD* est utilisé pour calculer l'addition des valeurs contenues dans les registres X et Y. Enfin le résultat sur 64 bits (16 bits utiles) est stocké dans le registre Z. Le processus au cours de chaque itération est illustré Figure 2.

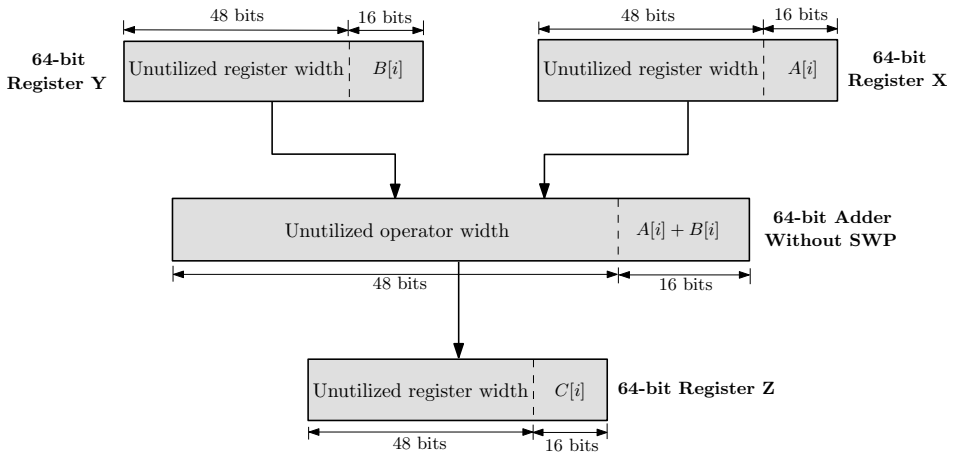

FIGURE 2: Implémentation de la boucle sur un processeur sans SWP

Comme illustré Figure 2, en raison de la faible précision des données d'entrée, les registres et l'opérateur ne sont pas pleinement utilisés. Au cours de chaque itération, au lieu

d'utiliser toute la largeur de 64 bits, seulement une largeur de 16 bits est réellement utilisée pour les deux registres et l'opérateur ADD. Cela se traduirait par une sous-utilisation du chemin de données du processeur et de l'unité de calcul.

Considérons maintenant que le processeur intègre le concept SWP. L'algorithme correspondant à la boucle considérée précédemment est ré-écrit comme indiqué ci-dessous:

```
short A[100], B[100], C[100];
int i;
  for (i = 0; i < 100, i+4)
    Packing of four 16-bit data from array A in Register X;
    Packing of four 16-bit data from array B in Register Y;
    Addition of X and Y using SWP enabled ADD operator;
    Store the four 16-bit results in register C;
  end loop;
```

Au cours de chaque itération, quatre données d'entrée 16 bits du tableau A sont regroupées dans le registre X et quatre données d'entrée 16 bits du tableau B sont regroupées dans le registre Y. L'opération d'addition est appliquée sur les registres X et Y, ce qui signifie que cette opération est appliquée simultanément sur quatre opérandes de 16 bits. Lors de chaque calcul, quatre résultats 16 bits sont stockés dans le registre Z. Le schéma correspondant à cette architecture est illustré par la Figure 3.

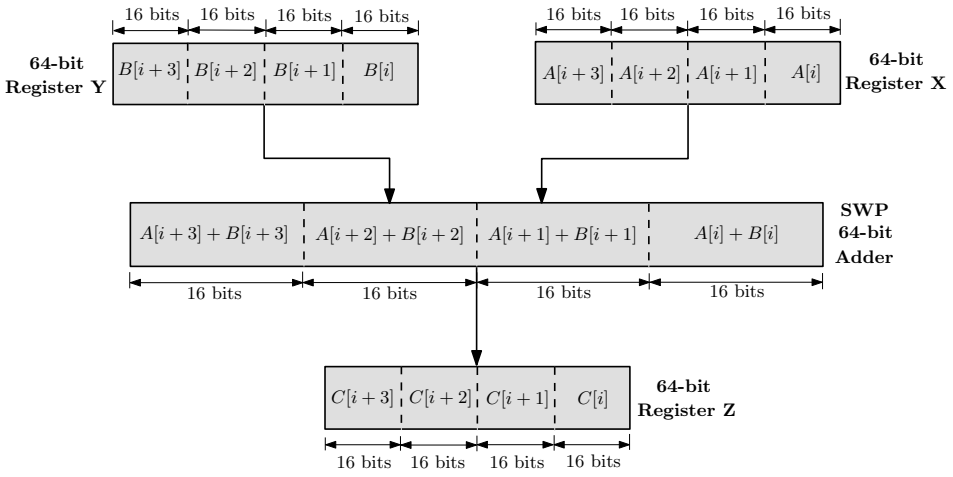

FIGURE 3: Implémentation de la boucle sur un processeur avec SWP

Comme illustré Figure 3, le taux d'utilisation du processeur est augmenté. Ainsi, au lieu de 100 instanciations de l'opérateur *64-bit ADD*, seulement 25 instanciations de l'opérateur *SWP 64-bit ADD* sont nécessaires. Les registres et les éléments d'interconnexion

sont également pleinement utilisés pour bénéficier au mieux du concept SWP. La rapidité du calcul est alors presque multipliée par quatre. En pratique, le regroupement et l'arrangement des sous-mots occupent du temps, ce qui entraîne une accélération légèrement inférieure à 4.

0.2.2 Concept SWP et applications multimédia

Les applications multimédia font appel en général à des calculs intensifs sur des données de faible précision, les pixels étant codés sur 8, 10, 12 ou encore parfois 16 bits. Différents algorithmes multimédia tels que l'estimation de mouvement, la transformée en cosinus discrète (DCT), la transformée en ondelettes discrète (DWT), etc. effectuent des calculs sur des données de largeurs différentes [7, 55]. La taille des pixels est très faible par rapport à la largeur des mots processeur. En effet, la plupart des processeurs sont optimisés pour effectuer des opérations sur des largeurs de mots de données qui valent 32 bits ou 64 bits voire 128 bits. Exécuter des applications multimédia sur de tels processeurs conduit à une sous-utilisation des ressources. Cette sous-utilisation des ressources du processeur peut être réduite en profitant du concept SWP lors de la conception du processeur.

Le concept SWP est une solution efficace pour accélérer les traitements multimédia car les algorithmes mis en oeuvre présentent un fort parallélisme de données avec des données de faible précision [30] [48]. Les pixels sont regroupés dans des registres de largeur égale à la largeur des mots et les calculs sont donc effectués en parallèle sur plusieurs pixels simultanément. Le taux de parallélisme obtenu dépend de la taille des pixels. Plus la taille des pixels est faible, plus grand sera le taux de parallélisme. Une bonne adéquation entre la taille des pixels et la largeur des sous-mots est évidement nécessaire pour avoir le meilleur taux d'utilisation des ressources possible.

0.2.3 Extensions multimédia dans les microprocesseurs à usage général

En raison de l'importance des applications multimédia tant par leur utilisation que par leur complexité grandissante, les processeurs à usage général (GPP) sont amenés à effectuer quantité de traitements multimédia. Pour améliorer les performances des GPP pour les applications multimédia, la plupart des GPP ont inclus un jeu d'instructions SWP en tant qu'extensions multimédia de leurs architectures [59] [60] [23] [83]. Ces extensions SWP permettent aux GPP d'utiliser plus efficacement leurs ressources matérielles lorsque que des données de faible précision sont utilisées. Les jeux d'instructions SWP de divers GPP bien connus sont:

- Matrix math extensions (MMX) et Stream SIMD d'Intel.
- AltiVec de Motorola PowerPC.
- Multimedia acceleration extensions MAX-1 et MAX-2 de PA-RISC.
- 3DNow! extension d'AMD architecture.
- Visual instruction set (VIS) d'UltraSPARC.
- MIPS digital media extensions (MDMX) de SGI MIPS.
- MVI de DEC Alpha.

En 2007, AMD a proposé un nouveau jeu d'instructions SWP (170 instructions) appelé SSE5 pour ISA x86-64. En 2008, Intel annonçait le développement d'AVX (Advanced Vector Extensions) avec 16 nouveaux registres SIMD 256 bits. Cet ensemble supporte également les opérations SIMD avec 3 opérandes. En 2009, AMD annonçait qu'elle supprimera le SSE5 et utilisera AVX, mais certaines instructions SSE5 perdureront dans le cadre de deux nouvelles extensions SWP appelées XOP et FMA4. Par ailleurs, plusieurs processeurs de traitement du signal récent (DSP) et processeurs multimédia supportent également le principe SWP pour améliorer les performances. Parmis les DSP avec des fonctions SWP figurent le TigerSHARC de Analog Devices, le TMS320C64x de Texas Instruments, etc.

0.3 Besoins associés au concept SWP

Les processeurs SWP permettent le traitement parallèle de données de faible précision et permettent d'accroître les performances globales du processeur. Lors de la conception d'un processeur SWP, plusieurs conditions doivent être remplies pour pouvoir effectuer les traitements correctement. Ces besoins comprennent évidement la disponibilité d'opérateurs SWP, un jeu d'instructions SWP, des procédures d'alignement et de regroupement des sous-mots, etc.

0.3.1 Opérateurs SWP

La conception d'un processeur SWP nécessite d'avoir à disposition des opérateurs SWP. Ces opérateurs effectuent des opérations parallèles sur des sous-mots regroupés [20, 28, 56]. Le nombre et la largeur des sous-mots pris en charge par l'opérateur SWP dépend des besoins. De ces besoins découle la largeur des différents sous-mots ainsi que la largeur des mots. Le schéma de principe d'un opérateur SWP est illustré Figure 4.

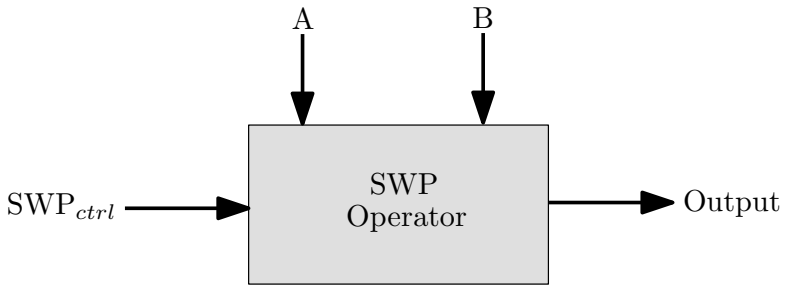

FIGURE 4: Opérateur SWP

Les entrées de l'opérateur SWP sont les opérandes A, B et les signaux de contrôle SWP (SWP$_{ctrl}$). Les entrées A et B contiennent les sous-mots regroupés. La largeur des sous-mots est sélectionnée à l'aide des signaux de contrôle SWP. L'opérateur SWP effectue des opérations sur plusieurs données en parallèle. Par exemple, si la largeur des sous-mots sélectionnée est 8 bits, alors l'opérateur SWP considère chaque vecteur d'entrée comme une combinaison de sous-mots de largeur 8 bits. Les opérations sont donc effectuées en parallèle avec des sous-mots de 8 bits. La sortie de l'opérateur SWP (le résultat) est également sous la forme de sous-mots regroupés. Selon le type d'opération, les largeurs des sous-mots en entrée et en sortie peuvent être différentes. Par exemple, avec l'opération multiplication, la largeur des sous-mots en sortie est le double de celle des sous-mots en entrée.

La complexité d'un opérateur SWP est généralement plus élevée que celle d'un opérateur classique car un opérateur SWP supporte des opérations en parallèle pour plusieurs largeurs de sous-mots alors qu'un opérateur classique effectue des opérations sur des données d'une seule largeur. Aussi, les opérateurs SWP requièrent des signaux de contrôle pour la sélection de la largeur de sous-mots. La complexité interne d'un opérateur SWP dépend du nombre et de la largeur des sous-mots. Moins il y a de largeurs différentes à supporter, plus l'opérateur sera simple. De même, s'il y a une relation mathématique (multiple) entre les différentes largeurs de sous-mots supportés, l'opérateur sera plus simple (pour plus de détails, voir le paragraphe 0.4).

0.3.2 Transferts de données entre les unités de traitement et la mémoire

Les transferts de données entre le processeur et la mémoire jouent un rôle très important dans la détermination de la vitesse globale d'un processeur. Comme illustré Figure 2, avec un processeur classique, un seul élément du tableau de données 16 bits est transféré en utilisant l'instruction *load* et *store*. Pour la même application, avec un processeur SWP, quatre éléments du tableau de données 16 bits sont transférés (Figure 3). Par

conséquent, lors de chaque transaction mémoire d'un processeur SWP, plus d'un élément de données est transféré en utilisant une seule instruction *load* et *store*. Cela conduit à une utilisation plus efficace de l'unité de mémoire. En pratique, les processeurs SWP doivent être en mesure de supporter à la fois des transferts de données de type mots et de type sous-mots. Par exemple, l'instruction MOVD permet de transférer 32 bits de données regroupées de la mémoire aux registres MMX et l'instruction MOVQ permet de transférer 64 bits de données regroupées.

0.3.3 Disponibilité de données de faible précision

La disponibilité de données de faible précision est partie intégrante du concept SWP. Si des données de faible précision ne sont pas disponibles, ce concept n'a pas d'intérêt. Généralement, dans des applications multimédia, une grande quantité de données de faible précision (pixels) est présente [10, 58]. Par conséquent, le concept SWP peut s'avérer très utile pour ces applications. A titre d'exemple, la technique SWP fonctionne efficacement pour les algorithmes d'estimation de mouvement vidéo, car ce type d'algorithmes doit être appliqué sur des millions de pixels [55, 77]. Toutefois, la technique SWP n'est pas utile pour les applications qui sont non parallèles par nature. L'exécution de ces applications sur des architectures SWP n'apportera pas d'amélioration des performances globales.

0.4 Largeurs des sous-mots dans les architectures SWP

Le sous-mot est une unité de moindre précision des données contenues dans un mot. En SWP, plusieurs sous-mots sont regroupés dans un registre de largeur égale à la largeur des mots, puis le traitement du mot entier est réalisé [98]. Dans un mot, les sous-mots peuvent être de largeurs différentes mais, dans le but de réduire la complexité, les sous-mots sont généralement de même largeur pour un calcul donné.

0.4.1 Parallélisme versus largeur des sous-mots

Le degré de parallélisme dans les architectures SWP dépend de la largeur des sous-mots. Plus la largeur des sous-mots est faible, plus élevé est le parallélisme. En général, le choix de la largeur des sous-mots résulte d'un compromis entre le parallélisme et la précision des données. Par exemple, une largeur de sous-mots de 8 bits peut s'avérer trop faible en terme de précision, alors que, lorsque l'on utilise un processeur 64 bits, une largeur

de sous-mots de 32 bits ne permet pas un parallélisme suffisant car on a alors seulement un parallélisme de deux opérations par instruction.

0.4.2 Largeurs des sous-mots supportées

En fonction des besoins, un processeur SWP peut supporter plus d'une largeur de sous-mots. Plus le nombre de sous-mots pris en charge est grand, plus grande est la complexité du processeur. Si la largeur des données en entrée correspond exactement à celle d'un des sous-mots, le rendement du processeur sera alors élevé. La largeur des sous-mots est sélectionnée à l'aide de signaux de contrôle. Ainsi, si la largeur des mots processeur est 32 bits alors la largeur des sous-mots peut être, par exemple, 8 bits, 10 bits, 12 bits et 16 bits. Une instruction permet alors d'effectuer des opérations sur quatre sous-mots de 8 bits ou sur trois sous-mots de 10 bits ou sur deux sous-mots de 12 bits ou sur deux sous-mots de 16 bits en parallèle ou sur un mot de 32 bits. En fonction de la valeur des signaux de contrôle SWP, l'opérateur SWP est configuré pour effectuer des opérations sur la largeur de sous-mots correspondante.

0.4.3 Adéquation entre largeur des données et largeur des sous-mots

Les performances d'un processeur SWP dépendent fortement de l'adéquation entre la largeur des données d'entrée et les largeurs de sous-mots supportées. Si la largeur des données est la même que la largeur des sous-mots, l'efficacité sera élevée. Une différence entre le largeur des données d'entrée et de la largeur sous-mots conduit à une sous-utilisation des ressources processeur. Le choix de la largeur des sous-mots dans les architectures SWP dépend en grande partie des applications pour lesquelles elles sont conçues. Dans le cas des processeurs à usage général (GPP), les calculs sont effectués pour une variété d'applications. Il n'est pas possible de prévoir exactement la largeur des données pour lesquelles le processeur doit fonctionner. Par conséquent, la largeur des sous-mots des opérateurs SWP dans les GPP n'est pas destinée à une catégorie particulière d'applications. A l'inverse, dans le cas de processeurs qui traitent d'applications spécifiques comme le multimédia par exemple, l'adéquation entre la largeur des données et la largeur des sous-mots peut être augmentée. En multimédia, de nombreux calculs sont exécutées sur des données de type pixels. Par conséquent, la largeur des sous-mots dans les processeurs multimédia doit être en adéquation, entre autre, avec les tailles des pixels. Un processeur multimédia hautes performances doit pouvoir supporter des sous-mots de toutes les tailles de pixels possibles. Dans le cadre de cette thèse, les largeurs des sous-mots sont caractérisées en deux catégories: *Classical subword sizes* et *Multimedia*

subword sizes (largeurs de sous-mots conventionnelles et largeurs de sous-mots orientées multimédia).

0.4.4 Largeur de sous-mots conventionnelles

Dans le cas des processeurs SWP existants [20, 28, 56, 57], un choix classique pour les largeurs des sous-mots est généralement 8, 16, 32 bits, etc. La raison de ce choix de largeurs est qu'il conduit à des opérateurs SWP moins complexes car la largeur d'un sous-mot donné est un multiple de la largeur du sous-mot de taille inférieure ($8 \times 2 = 16$, $16 \times 2 = 32$ et ainsi de suite). La complexité des opérateurs SWP est également moindre lorsque la largeur des sous-mots suit une relation arithmétique uniforme avec la largeur des mots processeur ($8 \times 8 = 64$, $16 \times 4 = 64$, $32 \times 2 = 64$ etc.). Les processeurs SWP qui sont basés sur des largeurs de sous-mots conventionnelles sont adaptés aux applications pour lesquelles la largeur des données d'entrée est basée sur l'octet. Toutefois, dans le cas d'applications multimédia, les tailles de pixels/données sont 8, 10, 12 ou parfois 16 bits et ne sont donc pas en adéquation avec les largeurs de sous-mots conventionnelles. Par conséquent, les opérateurs conventionnels SWP existants n'autorisent pas une pleine utilisation des ressources du processeur lorsque des applications multimédia sont considérées. A titre d'exemple, prenons en compte les applications multimédia liées aux images médicales. La taille des pixels est alors généralement de 12 bits. Supposons que ces applications sont exécutées sur un processeur SWP qui supporte les largeurs de sous-mots conventionnelles (8, 16, 32-bit, etc.). La largeur de sous-mots la plus appropriée disponible est donc 16 bits. Cependant, effectuer des calculs sur des pixels codés sur 12 bits avec des sous-mots de 16 bits conduit à une sous-utilisation des ressources du processeur tels que les registres, les unités de calcul, le chemin de données etc. Pour chaque ressource, les 4 derniers bits de poids fort ne sont pas nécessaires. Cette sous-utilisation des ressources du processeur est illustrée Figure 5 pour un processeur 64 bits.

Comme illustré Figure 5, en raison de la non disponibilité de sous-mots de largeur 12 bits, près de 25 % (16/64 = 0,25) des ressources du processeur ne sont pas utilisées . L'efficacité globale du processeur SWP est donc diminuée.

0.4.5 Largeurs de sous-mots orientées multimédia

Les largeurs de sous-mots orientées multimédia sont les largeurs qui sont en adéquation avec les tailles des pixels dans les applications multimédia modernes. Ces tailles sont habituellement 8, 10, 12 ou 16 bits. Les processeurs SWP conçus pour les applications multimédia doivent donc supporter des largeurs de sous-mots en correspondance. Les

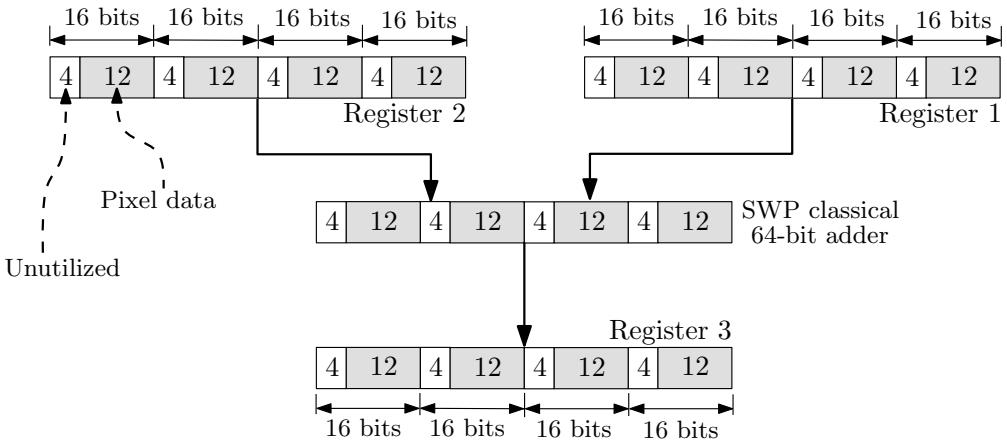

FIGURE 5: Utilisation de données 12 bits avec des sous-mots de largeur 16 bits

largeurs de sous-mots orientées multimédia (8, 10, 12 et 16 bits) ne sont pas des multiples entre elles et ne permettent pas d'avoir une relation arithmétique uniforme avec la largeur des mots processeur. Il en résulte une augmentation de la complexité de mise en oeuvre, mais au profit de l'efficacité de l'exécution de l'application sur le processeur. Par exemple, l'exemple précédent de calculs effectués sur des pixels codés sur 12 bits peut être exécuté sur un processeur SWP 64 bits qui supporte les largeurs de sous-mots orientées multimédia (8, 10, 12 et 16-bit) comme illustré Figure 6.

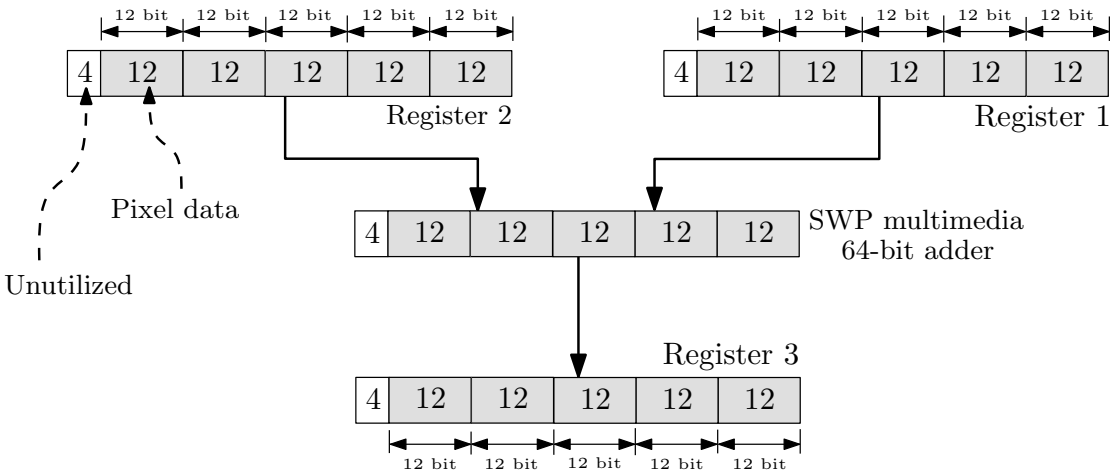

FIGURE 6: Utilisation de données 12 bits avec des sous-mots de largeur 12 bits

En raison de l'utilisation de largeurs de sous-mots orientées multimédia, les ressources sont utilisées de manière plus efficace. Les opérations sont effectuées sur cinq pixels 12 bits en parallèle. Avec l'opérateur SWP conventionnel, les opérations sont effectuées sur seulement quatre pixels 12 bits en parallèle. Par rapport au processeur SWP conventionnel, la sous-utilisation des ressources du processeur a été réduite de 25% à 6% (4 / 64 = 0,0625).

0.4.6 Largeur des mots avec les opérateurs SWP

En plus des largeurs des sous-mots, l'efficacité d'utilisation des ressources d'un opérateur SWP dépend aussi de la largeur des mots. La largeur des mots est choisie de manière à maximiser l'utilisation des ressources disponibles sachant qu'il y a plusieurs largeurs de sous-mots. Avec des largeurs de sous-mots conventionnelles (8, 16, 32-bit), il est facile de sélectionner la largeur des mots pour l'opérateur: il s'agit du plus petit commun multiple (PPCM) entre toutes les largeurs prises en charge. Les largeurs de mots appropriées sont donc 32-bit ou 64-bit etc. En utilisant ces largeurs de mots, l'utilisation de l'opérateur SWP sera maximale si les données d'entrée correspondent bien aux largeurs des sous-mots. Toutefois, avec des largeurs de sous-mots orientées multimédia, le PPCM entre toutes les largeurs de sous-mots n'est pas une option réaliste. Par exemple avec les largeurs de sous-mots 8, 10, 12 et 16 bits, le PPCM vaut 240 bits. Concevoir un circuit avec un chemin de données de 240 bits est trop couteux. Par conséquent, la largeur des mots doit être choisie autrement en cherchant à maximiser l'utilisation du processeur. Dans le cadre de cette thèse, pour la conception d'opérateurs SWP multimédia, nous avons choisi une longueur de mots de 40 bits. Cette valeur a été choisie en raison du bon compromis efficacité/complexité qu'elle procure. Ce choix sera justifié plus tard dans ce mémoire.

0.5 Contributions et organisation du mémoire de thèse

Le contexte de cette thèse repose sur le projet ROMA [29]. ROMA est synonyme de *Reconfigurable Operators for Multimedia Applications*. Ce projet a été lancé dans le cadre du programme ANR (Agence Nationale de la Recherche) *Architectures du Futur* (ANR-06-ARFU6-004-01)(2007-2010). Les partenaires du projet ROMA sont: l'IRISA [1], le CEA LIST [2], le LIRMM [3] et Thomson R&D France [4]. Dans les applications multimédia, le traitement de l'image est le défi majeur auquel les systèmes embarqués doivent faire face. Il s'agit de réaliser des calculs intensifs tout en répondant à des exigences en terme de puissance consommée. Des traitements de l'image au niveau pixel, comme le filtrage, la détection de bords, la corrélation au niveau pixel ou au niveau bloc, l'estimation de mouvement, etc. doivent être accélérés. Pour atteindre ces objectifs, le projet ROMA propose de développer un processeur reconfigurable, présentant une densité silicium élevée et une bonne efficacité énergétique, capable d'adapter sa structure aux

[1] Institut de Recherche en Informatique et Systèmes Aléatoires: http://www.irisa.fr/
[2] Laboratoire d'Intégration des Systèmes et des Technologies : http://www-list.cea.fr/
[3] Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier : http://www.lirmm.fr/
[4] Thomson Research & Development, France http://www.thomson.net

calculs qui peuvent être accélérés et/ou présentant un intérêt au niveau énergétique. Au contraire des précédentes tentatives de conception de processeurs reconfigurables, qui ont conduit à l'utilisation de réseaux d'interconnexions complexes entre opérateurs, le projet ROMA vise à concevoir une architecture pipeline à base d'opérateurs reconfigurables de granularité moyenne [29]. La Figure 7 présente l'architecture générale du processeur ROMA. Le processeur est principalement composé de blocs mémoire, d'opérateurs configurables, d'un réseau d'interconnexions entre blocs mémoire et opérateurs, d'un réseau d'interconnexions entre opérateurs, et d'un processeur de contrôle.

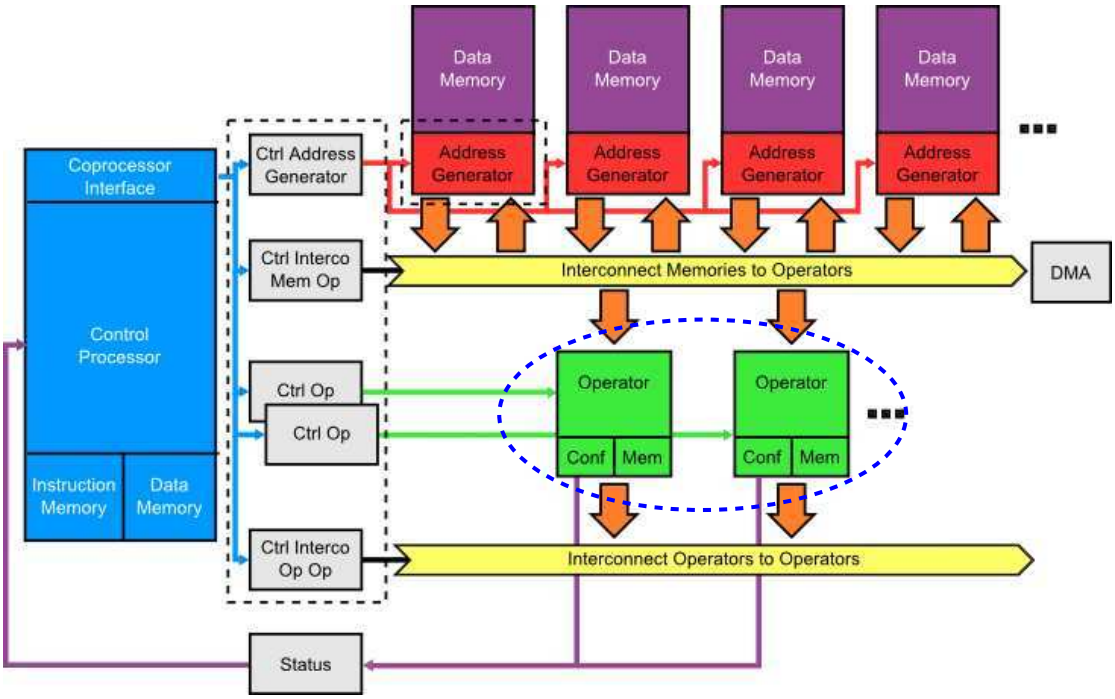

FIGURE 7: Block diagram of ROMA processor

Un bloc DMA sert d'interface pour accéder aux données provenant/à destination de l'extérieur du processeur, données alors stockées dans les blocs mémoires. Les opérateurs sont configurés pour effectuer un calcul donné via un code opération (op-code) délivré par le processeur de contrôle. Les flux de données sont également gérées par le processeur de contrôle. Un traitement est typiquement organisé de la manière suivante. Les données d'entrées stockées dans les blocs mémoire sont fournie à l'opérateur qui effectue les traitements sur ces données. Une fois les traitements effectués, le résultat peut être alors stocké dans un bloc mémoire ou envoyé à un autre opérateur.

Dans le cadre du projet ROMA, l'équipe lannionaise du laboratoire CAIRN de l'IRISA est impliqué dans le développement des opérateurs arithmétiques reconfigurables de ce processeur multimédia. Comme expliqué précédemment dans ce chapitre, le concept SWP peut être utilisé dans l'architecture des opérateurs pour améliorer les performances.

C'est dans ce contexte que s'inscrit directement cette thèse: concevoir des opérateurs multimédia reconfigurables hautes performances, intégrant le concept SWP. De nombreux opérateurs arithmétiques SWP ont été proposés dans la littérature. Cependant, les opérateurs proposés opèrent sur des largeurs de sous-mots conventionnelles ce qui aboutit à une sous-utilisation des ressources du processeur lorsque des applications multimédia sont exécutées sur ces opérateurs. Dans cette thèse, la conception de différents opérateurs SWP pour les applications multimédia est proposée. Une bonne adéquation entre largeur des sous-mots et largeur des données manipulées permet une meilleure utilisation des ressources disponibles et conduit ainsi à améliorer l'efficacité de l'exécution de l'application sur le processeur. Ces opérateurs arithmétiques de base sont ensuite utilisés dans un opérateur SWP reconfigurable. Ce dernier peut être configuré pour effectuer diverses opérations multimédia avec différentes largeurs de données sans surcout temporel de reconfiguration. La vitesse interne des différentes unités de traitement est également améliorée en représentant les nombres en système redondant plutôt qu'en système binaire. Le système redondant permet entre autre d'augmenter la vitesse des opérations arithmétiques en évitant une propagation de retenue couteuse lors d'opérations d'addition. Une architecture matérielle pour l'estimation de mouvement est également développée sur la base des opérateurs SWP proposés. Les résultats montrent l'intérêt en terme de performances d'utiliser des opérateurs SWP lorsque lors de l'exécution d'applications multimédia. Les contributions faites au cours de ce travail de thèse sont données ci-dessous.

- **SWP for multimedia operator design [52]:** Le but est d'introduire la conception d'opérateurs arithmétiques de base intégrant le concept SWP. Différents algorithmes sont utilisés pour la conception des opérateurs de base comme l'addition, la soustraction, la multiplication et le produit-accumulation. Les algorithmes qui requièrent un minimum de ressources sont analysés. Les largeurs de sous-mots conventionnelles et orientées multimédia sont considérées. Les surcouts d'intégration du concept SWP sont analysés.

- **SWP multimedia operator design [53]:** L'objectif est de mettre en évidence la sous-utilisation des ressources du processeur SWP lorsqu'il n'y a pas une bonne adéquation entre largeur de sous-mots et largeur des données manipulées. Différentes architectures d'unités arithmétiques pour les applications multimédia sont proposées en vue d'améliorer les performances globale du processeur. L'efficacité de ces architectures est évaluée sur différentes technologies cibles.

- **Reconfigurable SWP Operator for Multimedia Processing [50]:** La conception d'un opérateur SWP reconfigurable pour des applications multimédia est

présentée. Cet opérateur peut être utilisé pour effectuer diverses opérations multimédia sur différentes largeurs de données. La reconfiguration est réalisée au niveau des opérations exécutées et au niveau de la largeur des sous-mots. Les surcouts de configuration sont réduits en vue d'obtenir de bonnes performance.

- **Reconfigurable Operator Based Multimedia Embedded Processor [69]:** L'objectif est de présenter un processeur multimédia reconfigurable pour lequel les couts d'interconnexion sont réduits. En effet, la plupart des processeurs reconfigurables souffre généralement d'un temps de reconfiguration non négligeable, d'un cout d'interconnexion élevé et ne répond pas aux contraintes de faible puissance consommée. La conception d'un processeur reconfigurable, à base d'opérateurs reconfigurables de granularité moyenne, conçu pour les applications multimédias est présentée. L'architecture est flexible et évolutive. Les opérateurs peuvent être configurés en terme de la fonction qu'ils implémentent et de la largeur des données manipulées.

- **High speed reconfigurable SWP operator for multimedia processing using redundant data representation [51]:** Le but est de concevoir un opérateur SWP reconfigurable dédié aux applications multimédia en utilisant le système redondant pour représenter les nombres. Le concept SWP permet d'améliorer l'efficacité des traitements en considérant plusieurs données en parallèle. Par ailleurs, la vitesse des différentes unités arithmétiques est augmentée en évitant une propagation de retenue couteuse dans l'opération d'addition en utilisant le système redondant plutôt que le système binaire pour représenter les nombres. Ces deux avantages sont combinés dans l'opérateur reconfigurable proposé.

0.5.1 Organisation de la thèse

Le travail présenté dans cette thèse est divisé en sept chapitres. Un aperçu des chapitres est donné ci-dessous.

Chapitre 1 - Mise en oeuvre du concept SWP dans la conception d'opérateurs (Subword Parallelism SWP in operator design):

Ce chapitre a été rédigé avec l'intention de présenter au lecteur les termes et concepts utilisés dans les chapitres suivants de la thèse. Le chapitre commence par décrire le concept de parallélisme pour l'amélioration des performances. Les différents niveaux auxquels le parallélisme peut être appliqué sont examinés. Le parallélisme dans le cadre du traitement multimédia est également expliqué. Un aperçu des différentes architectures

disponibles pour le traitement des applications multimédia est donné. Le parallélisme de sous-mots (SWP pour Sub-Word Parallelism) est l'une des techniques utilisée afin d'exploiter le parallélisme au niveau des données présent dans différentes applications. Les avantages de l'utilisation du concept SWP pour améliorer les performances des processeurs lorsque l'on travaille sur les applications multimédia sont donnés. Différents processeurs à usage général (GPP) qui contiennent des instructions SWP sont également présentés. Les besoins relatifs à l'intégration du concept SWP dans l'architecture d'un processeur sont développés. Ces besoins consistent en la disponibilité d'opérateurs et d'instructions SWP et le fait d'avoir à manipuler des données de faible précision. Pour clarifier le concept, quelques instructions SWP et leurs fonctionnalités sont expliquées à l'aide d'exemples. Les performances du processeur SWP dépendent du choix de la largeur des sous-mots. L'effet d'une utilisation de sous-mots de largeurs conventionnelles ou de largeurs orientées multimédia sur l'efficacité des ressources du processeur est mis en évidence. Une bonne adéquation entre largeur des sous-mots et largeur des données manipulées permet d'améliorer les performances du processeur. La largeur de mots des opérateurs SWP est sélectionnée sur la base d'un bon compromis efficacité/complexité de ressources mises en oeuvre. Quelques limitations des processeurs SWP sont également données.

Chapitre 2 - Conception d'opérateurs SWP de base (SWP basic operators design):

Le contenu de ce chapitre est basé sur nos publications [52] et [53]. L'objectif de ce chapitre est la conception d'opérateurs arithmétiques SWP de base qui permettent d'effectuer des calculs élémentaires sur différentes données en parallèle. Ces opérateurs sont destinés à être ensuite utilisés dans un processeur de traitement d'applications multimédia afin d'en améliorer les performances. Les opérateurs concernés sont l'addition, la soustraction, la multiplication et le produit-accumulation. Ces opérateurs SWP sont conçus pour des largeurs de sous-mots conventionnelles et orientées multimédia et leurs performances sont comparées. Dans le cas des opérateurs SWP avec des largeurs de sous-mots conventionnelles, moins de ressources sont nécessaires en raison des relations arithmétiques uniformes entre largeur du mot et largeurs des sous-mots. Toutefois, l'efficacité des opérateurs avec des largeurs de sous-mots orientées multimédia sont meilleures. Pour chaque opération élémentaire, le concept SWP est appliqué sur différents algorithmes afin d'identifier l'algorithme qui conduit à une augmentation minimale des ressources. Comparée aux autres opérations de base, l'opération de multiplication requiert classiquement beaucoup de ressources matérielles. C'est également le cas lorsque le concept SWP est utilisé. Toutefois, un multiplieur SWP particulièrement efficace est proposé. Son architecture est basée sur le multiplieur SWP proposé dans [56] pour des largeurs de sous-mots conventionnelles. Les performances du multiplieur dédié aux données de

type multimédia sont analysées pour différentes largeurs de mots et de sous-mots. La surface, le chemin critique et la puissance consommées sont comparés.

Chapitre 3 - Le concept SWP dans les opérations multimédia (SWP in multimedia operations):

Dans ce chapitre, les différentes opérations nécessaires aux applications multimédias sont intégrées en utilisant les opérateurs arithmétiques SWP de base du chapitre 2. Ces opérations multimédia comprennent la somme de valeurs absolues de différences (SAD) utilisée pour l'estimation de mouvement, la somme de produits (SOP) utilisée pour la transformée en cosinus discrète (DCT). D'autres opérations d'usage général comme la somme d'additions / de soustractions ($SWP\sum(a \pm b)$) sont également intégrées en utilisant des opérateurs SWP. L'opérateur SWP qui réalise la valeur absolue de différences $|a - b|$ utilisée dans l'opérateur SAD est intégré pour différents algorithmes. Pour chaque opérateur décrit dans ce chapitre, les performance sont analysées ainsi que les couts additionnels consécutifs à la mise en oeuvre du concept SWP sur des données de largeurs orientées multimédia.

Chapitre 4 - Opérateur SWP reconfigurable pour applications multimédia (Reconfigurable SWP operator for multimedia processing):

Le contenu de ce chapitre est basé sur nos publications [50] et [69]. Dans ce chapitre, un opérateur SWP reconfigurable dédié aux applications multimédia est proposé. Cet opérateur élimine le besoin d'un temps de reconfiguration et autorise une reconfiguration au niveau des opérations exécutées et au niveau de la largeur des données manipulées. Un grande variété d'opérations multimédia peut être effectuée sur des données de différentes largeurs. L'exécution d'une opération donnée est réalisée via des signaux de contrôle qui permettent de sélectionner la largeur de sous-mots appropriée et d'activer les unités nécessaires à cette opération. Les unités arithmétiques utilisées dans l'opérateur reconfigurable intègrent le concept SWP. L'opérateur reconfigurable est synthétisé sur différentes technologies cibles et les résultats sont analysés. Comparé à un opérateur traditionnel SWP utilisé dans différents circuits DSP, l'opérateur reconfigurable proposé exécute plusieurs opérations multimédia en moins de cycles. Cet opérateur peut être utilisé comme unité spécialisée ou comme co-processeur dans un processeur multimédia afin d'en améliorer les performances.

Chapitre 5 - Opérateur SWP et représentation redondante (SWP using redundant representation):

Le contenu de ce chapitre est basé sur notre publication [51]. Le concept SWP permet d'augmenter les performances d'un opérateur en traitant en parallèle plusieurs données. Toutefois, la vitesse interne des différentes unités de traitement est aussi un facteur important. La vitesse interne des unités arithmétiques peut être augmentée en évitant une propagation de retenue couteuse dans les opérations d'addition si le système redondant plutôt que le système binaire est utilisé pour représenter les nombres. Avec le système redondant, les nombres sont représentés par des chiffres plutôt que par des bits. Chaque nombre peut être représenté par différentes combinaisons de chiffres redondants. Cette redondance dans la représentation du nombre aide à la réalisation d'additions très rapides sans propagation de retenue lourde en temps. Le concept SWP et la représentation des nombres en système redondant sont combinés dans ce chapitre pour la conception d'un opérateur reconfigurable pour applications multimédia. Les opérateurs arithmétiques proposés conduisent à une bonne utilisation des ressources via le concept SWP et à une grande vitesse de traitement grâce à l'utilisation du système redondant. Le surcout dû à l'utilisation du système redondant est analysé pour différentes technologies cibles.

Chapitre 6 - Estimation de mouvements à base d'opérateurs SWP (Motion estimation using SWP operators):

L'estimation de mouvement (ME) est couramment utilisée dans les applications multimédia pour l'opération dite de compression vidéo. Avec l'algorithme d'estimation de mouvement, le bloc courant qui doit être transmis est comparé avec les différents blocs d'une trame de référence et la meilleure correspondance est recherchée. Cette opération requiert beaucoup de calculs sur des données de faible précision (les données d'entrée sont des pixels) et le concept SWP présente donc un grand intérêt. L'algorithme ME est intégré en utilisant des opérateurs SWP permettant de considérer différentes tailles de pixels. Le processus de comparaison de blocs à l'aide de l'opérateur SWP nécessite moins de temps qu'une exécution sur des opérateurs classiques. Au cours de ces expériences, différents algorithmes de recherche sont utilisés, à savoir la recherche exhaustive et la recherche en diamants. Plusieurs tailles d'images de références (48x48, 32x32, 16x16) et tailles de blocs (16x16, 8x8) sont considérées.

Chapitre 7 - Conclusions:

Un résumé des points importants développés dans les différents chapitres de cette thèse est donné. Les travaux réalisés sont rappelés et plusieurs perspectives de travail sont proposées.

Chapter 1

Subword Parallelism SWP in operator design

The computational requirements of the processors are constantly increasing due to the fast growing needs of multimedia applications. These applications are computationally hungry with low precision pixel data. One of the possible ways to meet the computational requirements of multimedia applications is to fully utilize the available resources of the processor. Most of the processors work on words of data where as in multimedia applications the computations are performed on small size pixel data. Therefore with the word oriented conventional processors, maximum utilization of resources (operators, registers and other interconnection elements) is not possible for multimedia data. To improve the performance, the processor needs to perform parallel computations on low precision pixel data without wasting the word oriented datapath. This can be possible using subword parallelism (SWP) [30] [31] [48]. In SWP, parallel operations are performed on low precision packed data rather than wasting the word size resources. For this purpose SWP enable hardware units are required in processor design. These units perform parallel operations on low precision data items called subwords. Greater the coordination between pixel data and subword sizes, higher will be the resource utilization which ultimately increases the performance. In this chapter SWP technique is explained to increase the performance of processor through data level parallelism.

This chapter is organized as follows: Section 1.1 gives an overview of need for parallelism and describes different levels at which the parallelism can be exploited. Section 1.2 presents different processing methods available for the execution of multimedia applications. Section 1.3 explains the use of subword parallelism (SWP) to enhance the performance of processor for multimedia applications. Section 1.4 describes different requirements for using SWP in processor design. Section 1.6 explains the significance of

using different subword sizes in SWP processor design. Section 1.7 describes the effects of using different word sizes in the designing of SWP operators. Section 1.9 gives the contributions of this thesis and it also gives the brief summary of work which will be presented in next chapters of this thesis. Finally the chapter is concluded in Section 1.10.

1.1 Parallelism for performance enhancement

The performance of processor can be measured by the amount of time and resources required to perform any particular task. If the processor accomplishes the task in minimum time with minimum hardware resources then its performance will be high. However in practice there is usually a trade off between speed and resource requirement. One of the method to increase the speed of processor is to utilize the parallelism that exists at different levels [36, 39, 40]. Parallelism is one of the oldest and most important techniques to increase the performance of computing system. To fully exploit the parallelism, the parallel processing units are required which can perform parallel operations on input data. These parallel processing units ultimately increases the speed but the area also increases accordingly. Based upon the requirements, the parallelism can be applied at different levels. Main levels at which parallelism can be applied are instruction level parallelism and data level parallelism.

1.1.1 Instruction level parallelism

Instruction level parallelism ILP is a measure of how many instructions in a program can be executed simultaneously. The goal of compiler and processor is to identify and take the advantage of ILP as much as possible by executing them in parallel. Due to data and control dependencies it is not always easy to execute the instructions in parallel. There are several techniques used to exploit ILP.

Out of order execution : In this technique, the parallelism at instruction level is achieved by executing the instructions out of order. If the data required by one instruction is produced by another instruction then these two instructions can not be executed in parallel. To overcome this dependency, instructions are executed out of order. By doing so those instructions are executed first which do not have any dependencies. In this way the maximum instructions can be executed in parallel.

Register renaming : Register renaming is one of the techniques to increase the ILP. If two or more instructions try to use the same registers then these instructions can not be executed in parallel. To resolve this dependency the registers are renamed temporarily so that the instructions can be executed in parallel.

Speculative execution : In the speculative execution the instructions are executed without being sure that execution of these instructions will be required or not. If the speculation comes out to be true then the processor continue the execution of successive instructions. However if the speculation comes out to be wrong then the program counter will jump back to original location.

Branch prediction : Branch prediction is used to resolve the control dependencies and increase the ILP. In this technique, branch instruction is predicted and the execution of instructions is done on the basis of this prediction. Latter when the branch is actually resolved, if the prediction comes out to be true then the execution continues otherwise the program counter will jump back.

Instruction pipelining : Instruction pipelining is also used to increase the ILP. Several instructions are executed in parallel in different pipeline stages. Execution of multiple instructions can be partially overlapped.

1.1.2 Data level parallelism

In data level parallelism, same computations are performed on several data elements simultaneously. Data level parallelism takes advantage of the intrinsic parallelism that exists in the computation of data and generally can be applied whenever there are regular, repetitive computations. The main requirement of data level parallelism is that excessive amount of data should always be available for computation. The processor allocates some portions of data to different processing units for computations. By doing so, parallel computations are performed and the tasks are completed in less time. There are several ways to exploit data level parallelism. Some of them are given below.

Single instruction multiple data (SIMD) : SIMD is one of the most popular architecture to exploit the data level parallelism. In SIMD processors, the operation specified in a single instruction is applied on multiple data elements and executed by multiple operators at the same time. Therefore single instruction is required to perform parallel operations on data elements.

An overview of SIMD processor architecture is shown in Figure 1.1. The control unit broadcast the input instruction to all the processing elements (PEs) simultaneously. All the PEs performs same computations on different data elements.

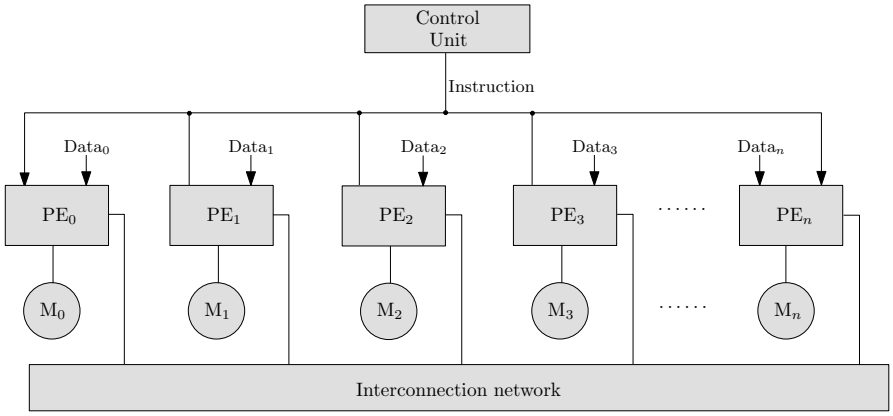

FIGURE 1.1: Architecture of SIMD processor

The speed-up of SIMD processor is determined by the number of data elements which can be processed in parallel. Greater the number of PEs higher will be the parallelism. SIMD processor requires the availability of multiple PEs and their interconnection network in the underlying hardware architecture.

Multiple instruction multiple data (MIMD) : In MIMD architecture, multiple processing elements simultaneously execute different instructions on different data. The control unit allocates different instructions to different PEs. The PEs performs required operations on input data and gives the results. This type of architecture is used when the heterogeneous operations are required on different data elements. The MIMD architectures are more complicated compared to SIMD processors, as the control unit has to allocate different instructions to different PEs. These PEs perform the tasks in different times, therefore the additional control signals are required to maintain the synchronism between PEs. Based on the availability of hardware resources, both SIMD and MIMD architecture can also utilize ILP that exist in the application.

The most efficient way to exploit data level parallelism is subword parallelism (SWP) [48, 59, 60]. It increases the performance of processor through parallel processing of data items. SWP is some time called as small SIMD [30]. In SWP instead of using separate PEs, multiple operations are performed using a single SWP enabled operator. This operator performs same operations on the data elements which are packed in input registers. Based upon input data sizes, these operators can perform operations on different size data elements. Using SWP, the processor resources are fully utilized even though the input data size is less than word size of processor. SWP can be utilizes to increase the performance of processors for many applications especially for multimedia applications. In multimedia application excessive amount of pixel data is usually available for computation. Therefore the data level parallelism can be exploited to perform

parallel computations on these pixel data. In SWP there is a trade off between number of operations which can be executed in parallel and the execution time of application. Larger the number of parallel operations by SWP operator, lesser will be the execution time.

1.2 Multimedia Processing

Multimedia processing involves the processing of digital video, image, graphics etc. In multimedia applications the computations are performed on low precision pixel data. These applications are very computation hungry and require lot of processing. To execute these applications, dedicated processors are required which can perform the multimedia oriented computations very efficiently on pixel data [73] [80]. In the beginning, general purpose processors (GPP) were not designed specifically for multimedia applications. However with the passage of time these applications become so ubiquitous that most of time GPP has to deal with some sort of multimedia processing. On average almost 90% of GPP clock cycles are spend to execute multimedia data. Due to these growing trends, GPP has also included many features which supports multimedia processing. These extra features increase the performance of GPP when working on multimedia applications. The well establish architectures available for the processing of multimedia applications are given below.

- **Super scalar processor** A super scalar architecture is a uniprocessor that can execute two or more scalar operations in parallel [84]. Super scalar architecture is used in most general-purpose processors. This architecture exploits instruction-level parallelism (ILP). The dependence between the instructions is either handled statically or dynamically [38]. The static approach relies on the compiler to pack independent instructions (from an execution schedule) and the hardware to execute them in parallel. In the dynamic approach, the instruction schedule is done dynamically and dependencies are tracked by the hardware. The static super scalar architecture is more likely to suit a multimedia processor. This approach has been used in many DSPs. A more general limitation of super scalar processors is the limits of ILP available in the application.

- **Multiprocessor** Multiprocessor architecture can be used for the processing of multimedia applications. At the higher level multimedia processing can be seen as single-program-multiple-data (SPMD). In SPMD one procedure is generally applied to a large data set. For example, the motion estimation procedure in MPEG encoder is applied to all macro blocks in a frame. This forms a parallel

execution schedule. Thus multiprocessing can be used for the parallel processing of different algorithms in multimedia applications.

- **Vector processor** Vector architectures are cost effective solution for applications with high data level parallelism. They were originally designed for scientific applications, such as weather forecasting and physics simulations. Multimedia application also matches the data parallel nature of these applications. Therefore the vector processors can also be used for multimedia processing. In multimedia application excessive amount of data level parallelism is available and the vector processor exploit this parallelism very efficiently.

  Vector processor can be implemented in two possible ways. The first way to implement vector processor is based on having one or relatively few pipelined functional units. Vector elements are processed in a pipelined fashion. These processors are called pipelined vector processors. In the second way the vector processor is implemented by replicating the functional units and achieves parallelism by processing all elements of the vector at the same time. When processing all the vectors in parallel, the interconnection network is also required between the functional units. In order to fulfill the data requirements of all the functional units, multiple paths from memory to processor are required. These type of processors are called parallel vector processors. SWP is also a form of vector processing which utilizes data level parallelism [30, 59]. In the next section SWP technique will be discuss in detail.

1.3 Subword parallelism SWP

In SWP, parallel computations are performed on low precision data elements which are packed in word size registers [30] [48]. These low precision data elements are called subwords. As a result of SWP, the same data path and computation units perform more than one computation on multiple subwords packed in word size registers. Parallel executions on packed subwords are initiated by single instruction. The number of subwords which can be packed in word size register depends upon the selected subword size. Operations on packed subwords make efficient use of the processor resources like operators, registers and interconnection elements etc. To perform operations on packed subwords, computation units are required which contain SWP capability. Figure 1.2 shows an example of adder with SWP capability. This adder is similar to conventional adder which accepts two 64 bits numbers and produce 64 bit sum. By blocking the carry chain at subword boundaries we can perform multiple low precision additions on subword data.

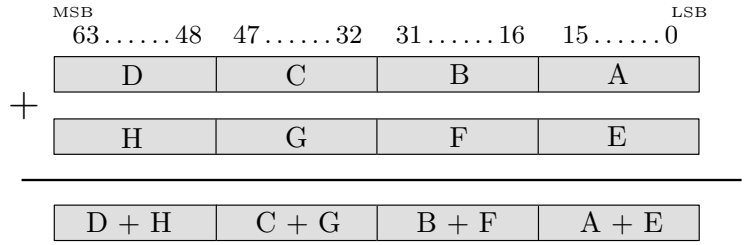

FIGURE 1.2: Parallel subword ADD instruction

In Figure 1.2, four 16-bit additions are performed by blocking the propagation of carry from bit 15 to 16, from bit 31 to 32 and from bit 47 to 48. These four additions are performed simultaneously on 16-bit subwords packed in 64-bit word registers. This adder supports only 16-bit subword size. Based on the requirements SWP operator can be implemented which supports multiple subword sizes. The speed-up attained by SWP processor depends on the number of subwords which are processed simultaneously. On 16-bit data, the speed of SWP adder shown in Figure 1.2 is four times faster than the conventional 64-bit adder. The actual speed-up of SWP processor is usually less due to the SWP overheads like subword arrangement and alignments before the computations.

1.3.1 Utilization of data level parallelism in SWP

SWP utilizes data level parallelism of application. Generally in data parallel computation one piece of code is executed several times for different data. In some cases the input data for each iteration is available in advance where as in some other cases input data for next iterations is calculated by previous iterations. Let us consider the following high level language loop which executes the same computation (addition) 100 times. A, B and C are three arrays that can store 100 elements each. Each element of array is of 16-bit data width. Arrays 'A' and 'B' contain input data elements. Array 'C' is used to store the result.

```
short A[100], B[100], C[100];
int i;
  for (i = 0; i < 100, i++)
    C[i] = A [i] + B[i];
  end for
```

Let us assume that the machine on which this code is to be executed is 64-bit processor (64-bit data path and operators etc.). As the processor do not contain SWP capability therefore it will perform 100 add operations by instantiating *64-bit ADD* operator 100

times. During each iteration, 16-bit input data values A[i] and B[i] are stored in two internal 64-bit registers X and Y. Then *64-bit ADD* operator is used to compute the addition of 16-bit values stored in registers X and Y. Finally 16-bit result is stored in register Z. The process during each iteration is shown in Figure 1.3.

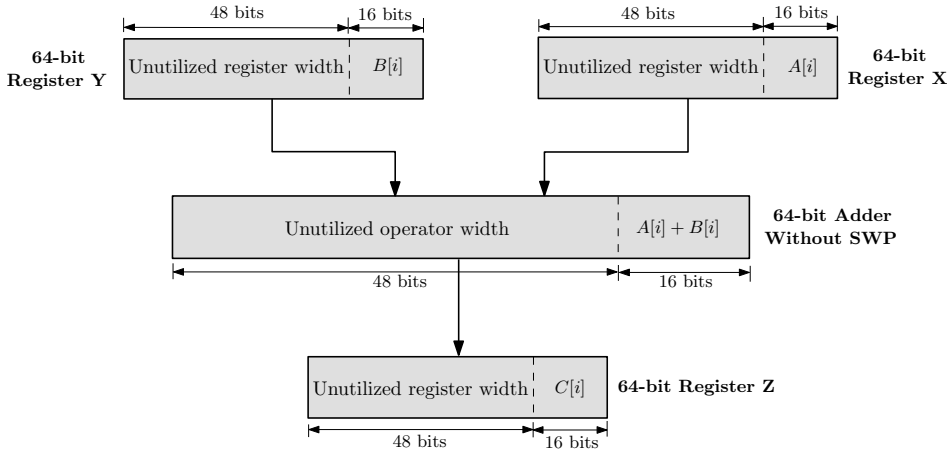

FIGURE 1.3: Loop implementation on processor without SWP

As shown in Figure 1.3, due to small precision of input data, the registers and operator data width are not fully utilized. During each iteration instead of utilizing full width of 64 bits, only 16-bit data width is utilized for both registers and ADD operator. This would result in the under utilization of processor datapath and computation units. Now let us consider that this processor contain SWP capability and can perform the computation on subwords packed in processor registers. The algorithm for above loop on SWP capable processor is shown below:

```
short A[100], B[100], C[100];
int i;
  for (i = 0; i < 100, i+4)
    Packing of four 16-bit data from array A in Register X;
    Packing of four 16-bit data from array B in Register Y;
    Addition of X and Y using SWP enabled ADD operator;
    Store the four 16-bit results in register C;
  end loop;
```

During each iteration on SWP capable processor, four 16-bit input data values from array A are packed in register X and four 16-bit input data values from array B are packed in register Y. ADD operation is the applied on register X and Y which means that operation is applied simultaneously on four 16-bit data values from each input array.

In each computation, four 16-bit results are stored in register Z. The block diagram of this architecture is shown in Figure 1.4.

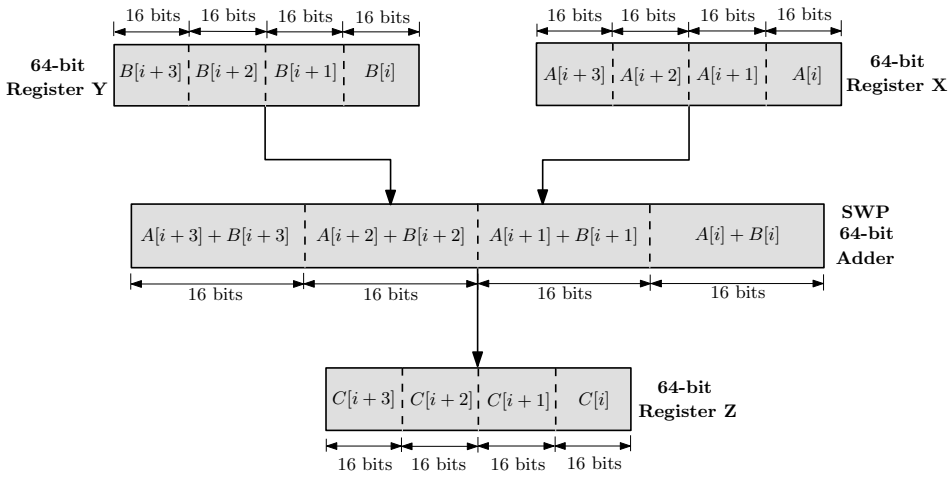

FIGURE 1.4: Loop implementation on processor with SWP

As shown in Figure 1.4 the utilization of processor has been increased. Instead of 100 instantiations of *64-bit ADD* operator, SWP capable processor requires only 25 instantiations of *SWP 64-bit ADD* operator. With SWP operator, the registers and other interconnection elements are also utilized to maximum extend. Therefore by the use of SWP, the speed of processor is almost increased by four times. The packing and arrangement of subwords also consumes some time, which results in actual speed-up that is slightly less than 4. Before any computation the subwords are packed in word size registers and are arranged in required order. If the compiler is friendly with SWP architectures, these SWP overheads can be reduced. The speed-up in SWP architecture occurred because of parallel computations on packed subwords.

1.3.2 SWP in multimedia application

Multimedia applications are normally computationally intensive with low precision pixel data such as 8, 10, 12 or sometimes 16 bits. Different multimedia algorithms like motion estimation, discrete cosine transform (DCT), discrete wavelet transform (DWT) etc. perform extensive computations on pixels of different sizes [7, 55]. The size of pixel data is very small compared to the word size of processor. Most of the processors are optimized to perform operations on words of data which are 32-bit or 64-bit or 128-bit etc. wide. Executing multimedia applications on word oriented processors results in under utilization of processor's resources. Based on pixel size only few percentage of hardware resources are consumed in each clock cycle. This under utilization of processor

resources can be minimized by introducing SWP capability in the processor design which needs to perform on multimedia applications.

SWP is an efficient and flexible solution for accelerating multimedia processing because the algorithms exhibit a great deal of data parallelism on lower precision data [30] [48]. Due to the availability of excessive pixel data, multimedia applications are inherently suitable for SWP architectures. Using SWP, low precision pixels are packed in word size registers and parallel computations are performed on multiple pixels simultaneously. By doing this, processor can achieve more parallelism rather than wasting the word size datapath and register sizes. This will increase the overall efficiency of processor when working on multimedia applications. The amount of parallelism achieved depends upon the size of pixel data. Smaller the pixel size, higher will be the parallelism. The coordination between pixel size and subword size of SWP processor also increases the resource utilization of processor.

1.3.3 Multimedia extension in general purpose microprocessors

Due to the huge growth in multimedia applications, excessive amount of multimedia processing is also required in general purpose processors (GPP) as well. To enhance the performance of GPP for multimedia applications, most of the GPP have included SWP instruction set as multimedia extensions to their architectures [59] [60] [23] [83]. These SWP extensions allow GPP to utilize word size hardware resources to maximum extend even when it is not processing high precision data. When executing multimedia applications in GPP, the extended SWP instructions sets are used to operate on subwords (pixels) in parallel way. The SWP instruction set extensions in various well known GPPs include:

- Matrix math extensions (MMX) and Stream SIMD in Intel.
- AltiVec in Motorola PowerPC .
- Multimedia acceleration extensions MAX-1 and MAX-2 in PA-RISC.
- 3DNow! extension in AMD architecture.
- Visual instruction set (VIS) in UltraSPARC.
- MIPS digital media extensions (MDMX) in SGI MIPS.
- MVI in DEC Alpha.

In 2007, AMD proposes new SWP instruction set (170 instructions) called SSE5 to x86-64 ISA. In 2008 Intel announces the development of AVX (Advanced Vector eXtensions) with 16 new 256-bit SIMD registers. It also supports SIMD operations with 3 operands. In 2009, AMD announces that it will discard SSE5 and use AVX, but some SSE5 instructions will survive as part of two new SWP extensions called XOP and FMA4. Several recent digital signal processors (DSP) and multimedia processors also support SWP for performance enhancements. DSP with SWP capability include tigerSHARC from Analog Devices [42], TMS320C64x from Texas Instruments [41] etc. Media processors which are specially designed for multimedia rather than general purpose processing also include SWP which provides low overhead parallelism.

Instructions	b_{in1}	b_{in2}	b_{out}	k	T_{lat}	T_{th}
`ADD4`	8	8	8	4	1	1
`ADD2`	16	16	16	2	1	1
`ADD`	32	32	32	1	1	1
`MPY4`	8	8	16	4	4	1
`MPY2`	16	16	32	2	4	1
`MPY2IR`	16	32	32	2	4	1
`MPYx`	16	16	32	1	1	1
`MPYxI`	16	32	48	1	4	1
`MPYxIR`	16	32	32	1	4	1
`MPY32`	32	32	32	1	4	1

TABLE 1.1: TMS320C64x+ SWP instruction set

To illustrate the different SWP instructions available in recent DSP, the TMS320C64x+ instruction set has been analyzed. The different instructions available for addition and multiplication are summarized in Table 1.1. The terms b_{in1}, b_{in2} represent respectively the input word-length and b_{out} corresponds to the operation output word-length. The instruction latency is expressed through T_{lat} and the instruction throughput T_{th} is equal to 1 cycle for each instruction. This means that a new instruction can be started on the operator (functional unit) at each cycle. For the addition, three instructions manipulating different word-lengths are available. One, two or three additions can be executed on respectively 32, 16 or 8-bit data. For the multiplication, seven instructions manipulating different word-lengths are available. For some instructions (`MPY2IR`, `MPYxIR`), the multiplication output word-length is lower than the word-length required to store the multiplication results. Thus, some bits of the multiplication result are eliminated. The suffix `R` indicates that the rounding mode is used for the cast of the operator output data. Two instructions (`MPYxI`, `MPYxIR`) manipulate the same input word-length but the results are stored on 48 or 32 bits. In the first case, two registers are needed to store

the multiplication output and in the second case only one register is needed but some output bits have to be eliminated. In this last case, the following operations using this result take less time.

To reduce the code execution time, some recent fixed point processors exploit the data-level parallelism by providing SWP capabilities [70, 71]. An operator (multiplier, adder, shifter) of N word length is split to execute k operations in parallel on sub-word of N/k word-length. This technique can accelerate the code execution time up to a factor k . Thus, these processors can manipulate a wide diversity of data types as shown in Table 1.2 for several recent DSPs [71].

Processor	**Data Types (bits)**
TMS320C64x (T.I.) [41]	8, 16, 32, 40, 64
TigerSHARC (A.D.) [42]	8, 16, 32, 64
SP5, UniPhy (3DSP) [1]	8, 16, 24, 32, 48
CEVA-X1620 (CEVA) [43]	8, 16, 32, 40
ZSP500 (LSI Logic) [100]	16, 32, 40, 64

TABLE 1.2: Data sizes which can be manipulated by different DSPs offering SWP capabilities for arithmetic operations

In [26], SWP technique has been used to implement a CDMA (code-division multiple access) synchronization loop in the TigerSharc DSP [42]. The SWP capabilities offer the opportunity to achieve an average 6.6 MAC per cycle with two MAC units.

1.3.4 SWP building block IPs

Most of the modern synthesis tools contain SWP or SIMD building block IPs in their libraries. Based on the requirements, these IPs can be used in any hardware architecture design. For instance in Synopsys Design Compiler [87], different functions are available to implement SWP arithmetic IPs. The *DWF_dp_simd_add* functions implement a configurable SIMD adder [88]. It allow to either add arguments 'a' and 'b' as full-width vectors (for example, one 32-bit addition) or to add smaller partitions of 'a' and 'b' using multiple parallel adders (for example, two 16-bit additions or four 8-bit additions). Similarly *DWF_dp_simd_mult* functions implement a configurable SIMD multiplier. The instantiation of *DWF_dp_simd_add* and *DWF_dp_simd_mult* building block IPs is given below.

```
sum <= DWF_dp_simd_add(a, b, no_confs, conf);

product <= DWF_dp_simd_mult(a, b, no_confs, conf);
```

Where a and b are two input operands. Argument *no_confs* specifies the number of possible configurations. Argument *conf* dynamically selects one configuration. Configuration with parameter *conf* has 2^{conf} partitions of size $\{\text{width}/2^{conf}\}$. Where *width* is the size of input operands. If the *width* of input operand is 32-bit and *conf* argument is selected as 2 then *DWF_dp_simd_add* and *DWF_dp_simd_mult* function implements 32-bit SWP adder and multiplier respectively with subword size of 8-bit. Operators with other classical subword sizes can be implemented by selecting different values of input arguments. However these classical SWP IPs are more generic and have less efficiency compared to dedicated SWP operators designed for multimedia applications with pixel oriented subword sizes.

1.4 SWP requirements

SWP processors gives parallel processing of low precision data and increase the overall performance of processor. When SWP capability is introduced in processor's design then there are several requirements which need to be fulfill for its smooth processing. These requirements includes SWP enabled operators, SWP instruction set, subwords alignment and regrouping procedures, subword data movement between memory and processing units etc.

1.4.1 SWP operators

For designing a processor with SWP capability, the basic requirement is SWP enable computational operators. These operators perform parallel operations on packed subwords [20, 28, 56]. The number and size of subwords supported by SWP operator depends upon the requirement. Based on the requirements, SWP operators perform computations on different subword sizes as well as on word size data. The block diagram of SWP operator is shown in Figure 1.5.

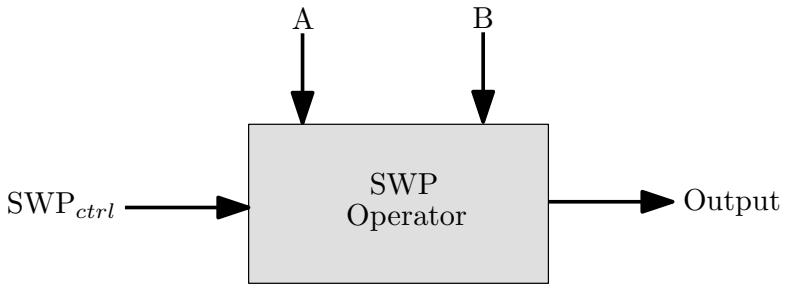

FIGURE 1.5: Block diagram of SWP operator

The inputs to SWP operator are operands A, B and SWP control signals (SWP_{ctrl}). Inputs A and B contains packed subwords. The subword size in SWP operator is selected with the help of SWP control signals (SWP_{ctrl}). Based upon selected subword size, SWP operator perform parallel operations on subword size data. For instance if the selected subword size is 8-bit, then SWP operator considers each input vector as the combination of 8-bit subword sizes. Parallel operations are performed on 8-bit subword sizes and the results are obtained corresponding to each subword computation. The output of SWP operator is also in the form of packed subwords. Based on the operation type, the size of input and output subword sizes can be different. For instance in multiplication operation, the size of output subword sizes are doubled than the input subword sizes.

The complexity of SWP operator is usually higher than simple operator because SWP operator supports parallel operations on different subword size data. Where as the simple operator perform operations on word size data only. SWP operators requires additional control signal for subword size selection. These control signals directs the SWP operator about the size of subwords packed in the input data vector. The internal complexity of SWP operator depends upon the number and size of supported subword sizes. Smaller the number of supported subword sizes lesser will be the complexity. If the supported subword sizes have uniform relationship with word size of operator then the complexity of SWP operator will be less (for details see section 1.6).

1.4.2 Data transfer between processing units and memory

Transfer of data between processor and memory plays very important role in determining the overall speed of a processor. As shown in Figure 1.3, a processor without SWP capability requires only one 16-bit array element to be transferred by using a load and store instruction. However for the same application, a SWP enable processor requires four 16-bit array elements to be transferred (Figure 1.4). Therefore during each memory transaction of a SWP capable processor, more than one data element needs to be transferred using a single load and store instruction. This would result in a more efficient use of the memory unit as well. For efficient use of SWP, the data and address lines between memory and processing units should be able to support both word and subwords data transfer. Multimedia extensions in GPPs contain different instruction to transfer packed data between memory and processing units. For instance the MOVD instruction transfer 32-bit of packed data from memory to MMX registers and vice versa. Similarly MOVQ instruction transfer 64-bit of packed data in MMX multimedia extension.

1.4.3 Availability of low precision data

Another requirement of SWP processor is the availability of low precision data. If the low precision data is not available then the parallel processing of subwords cannot increase the overall performance of processor. Usually in multimedia applications there is an excessive amount of low precision pixel data available [10, 58]. Therefore the SWP is very useful for multimedia applications. As an example, SWP technique works very efficiently for video motion estimation algorithm because this algorithm has to be applied on millions of pixels which are stored as low precision data [55, 77]. However SWP technique is not useful for the applications which are either not parallel in nature or which do not contain excessive amount of data level parallelism. Executing these applications on SWP architectures will not make any substantial effect on the overall performance enhancement.

1.5 SWP instruction set

In addition to other requirements, processors that uses SWP capability also need extended instruction set in order to cope with subwords of data. These extended instruction set invokes the parallel computations on subwords and also the supporting operations needed by SWP [23, 59, 60, 83]. These operations include alignment of data before or after certain parallel operations, arrangement and regrouping of subwords within the registers for parallel computation, contraction of data to smaller data width, expansion of data to larger data width, the ability to move multiple subwords between processor registers and memory etc [31]. In extended instruction sets, new primitives are also introduced in order to fulfill the needs while operating on subwords of data. For instance MAX-2 is an extension to 64-bit PA-RISC microprocessor. MAX-2 uses 16-bit subword size data [59, 60]. By using MAX-2 extended instructions, four parallel 16-bit operations can be performed by PA-RISC microprocessor. MAX-2 extension includes different instructions to perform parallel operations on subwords.

1.5.1 Parallel ADD and SUB instruction

MAX-2 uses *hadd* and *hsub* instructions for parallel addition and subtraction of subwords respectively [59]. These instructions perform parallel operations on subwords and store the results in resultant subwords. Addition and subtraction using *hadd* and *hsub* instructions is shown in Figure 1.6.

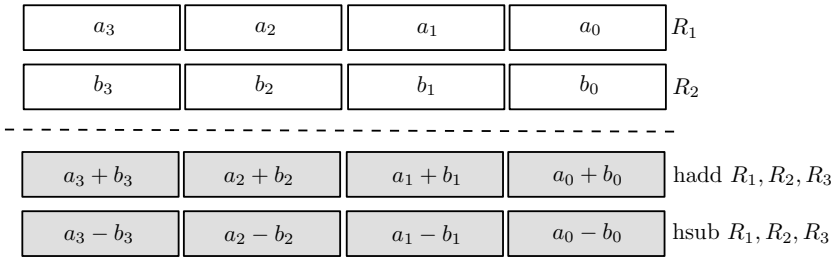

FIGURE 1.6: Parallel addition and subtraction in MAX-2

Before using *hadd* and *hsub* instructions, one has to make sure that no overflow will occur and the result of addition and subtraction should not exceed the subword size.

1.5.2 SWP instructions to avoid overflow in MAX-2

Overflow occurs when the result of any operation on subwords can not be represented by allocated number of bits in resultant subword. This situation happens mostly in SWP arithmetic operations. To avoid this overflow MAX-2 uses saturation arithmetics instructions [59]. These instructions includes:

hadd, us (A, B, C);
hsub, us (A, B, C);
hadd, ss (A, B, C);
hadd, ss (A, B, C);

For instance *hsub, us (A, B, C)* instruction is used to compute the difference of unsigned subwords stored in two input registers A and B. However if the resultant difference is negative then it is clipped to zero. Saturation arithmetics instructions are helpful to compute different operations required in multimedia applications as well.

SAD operation using SWP instructions Sum of absolute value of difference (SAD) is one of the most frequently used operation in multimedia applications [94, 101]. It is used in motion estimation algorithm for block matching. SAD operation is given by Equation 1.1.

$$\text{SAD} = \sum_{i=0}^{N-1} |a_i - b_i| \tag{1.1}$$

SAD is applied on pixel data packed in input registers. It can be performed in MAX-2 using saturation arithmetic SWP instructions. This process is shown in Figure 1.7.

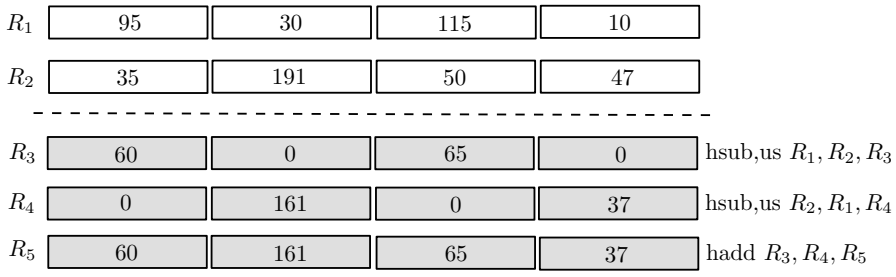

FIGURE 1.7: SAD operation using SWP instructions

The unsigned pixels from current frame and reference frame are stored in two registers R_1 and R_2. Each of these registers contains four pixels. The difference of two pixels can be positive or negative number. *hsub,us(R_1, R_2, R_3)* instruction gives only the positive difference between the corresponding subwords. If the difference between the subwords is negative, it is clipped to zero. By using *hsub,us* instruction two times with operand switched, the absolute value of difference between the subwords can be obtained. These absolute values of difference are then added using *hadd(R_3, R_4, R_5)* to obtain final values in one register. The packed subwords are then added to obtained SAD value.

Greater number finding using SWP instructions In certain applications it is often required to find greater of two input numbers. In SWP, the operations are performed on subwords therefore it is necessary to find the larger value for each subword compared to the corresponding subword in other operand. For this purpose, multiple relational operators are required which are equal to the number of subwords packed in the input registers. However using saturation arithmetic SWP instructions, this operation can be performed easily on subwords. The process is shown with the help of example in Figure 1.8.

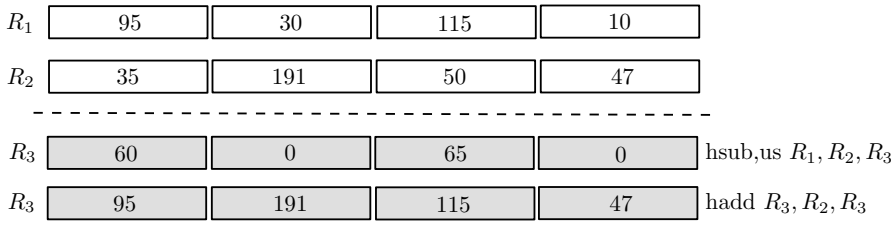

FIGURE 1.8: Finding of greater subwords values

In the first step, the subwords of register R_2 are subtracted from register R_1 using hsub,us(R_1, R_2, R_3) SWP instruction. If the result of any of the subword subtraction is negative the resultant subword in register R_3 will be zero. In the second step, the subwords of register R_3 are added to R_2 using hadd(R_3, R_2, R_3)

SWP instruction. Finally the register R_3 will contain those subwords whose values are greater than corresponding subwords values in other input register.

1.5.3 MIX instruction in MAX-2

Mix instruction is introduced in MAX-2 for the regrouping of subwords which is usually required in different SWP computations [59]. *Mix* instruction takes the subwords from two input registers and interleaves the alternative subwords in the resultant register [85]. Some of the arrangements and regrouping of subwords that are commonly needed in a SWP capable processor are shown in Figure 1.9.

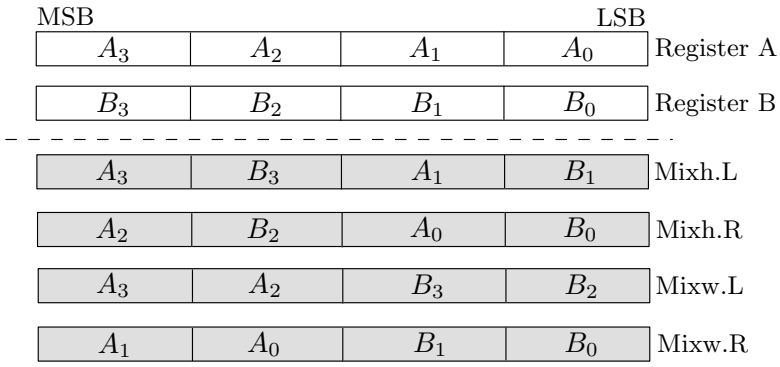

FIGURE 1.9: Different subword arrangements using *Mix* instruction

'A' and 'B' are two input registers which contain four subwords each. In the first rearrangement, subwords from input registers are combined in resultant registers by interleaving the alternate subwords starting from LSB using *Mixh,L* instruction. In the second rearrangement same type of regrouping is done starting from MSB using *Mixh,R* instruction. These two arrangements are useful when we need to perform certain operations on even or odd subwords of certain registers. However some times certain operations need to be performed on successive subwords or on word size data. For this purpose *Mixw,L* and *Mixw,R* instructions are used. *Mixw,L* instruction takes the alternate pairs of subwords from two input registers starting from LSB and combined them in resultant register. Similarly the *Mixw,R* instruction performs the same job starting from MSB.

Matrix transpose using MIX instruction *Mix* instruction is helpful in performing matrix transpose [59] which is required in discrete cosine transform (DCT) and in other video applications as well. Transpose operation is used when certain operation needs to be applied first on rows and then on columns of (n × n) image. Instead of applying operation on rows and then on columns, it is easy to apply

operation on rows followed by the transpose operation and then again apply the operation on rows [3]. This will simplify the computation process. Transpose operation used in the computation of two dimensional DCT transform (2-D DCT) is shown in Figure 1.10.

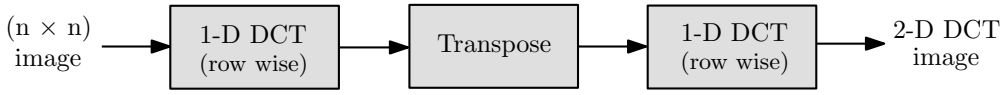

FIGURE 1.10: Two dimension DCT transform

Therefore the transform operation reduces the complexity of two dimensional DCT operation by performing the same operation twice. In SWP processors, transpose operation can be performed using few *Mix* instructions. For the transpose of (n × n) matrix only nlog(n) *Mix* instructions are required. The transpose of (4 × 4) matrix using *Mix* instructions is shown in Figure 1.11.

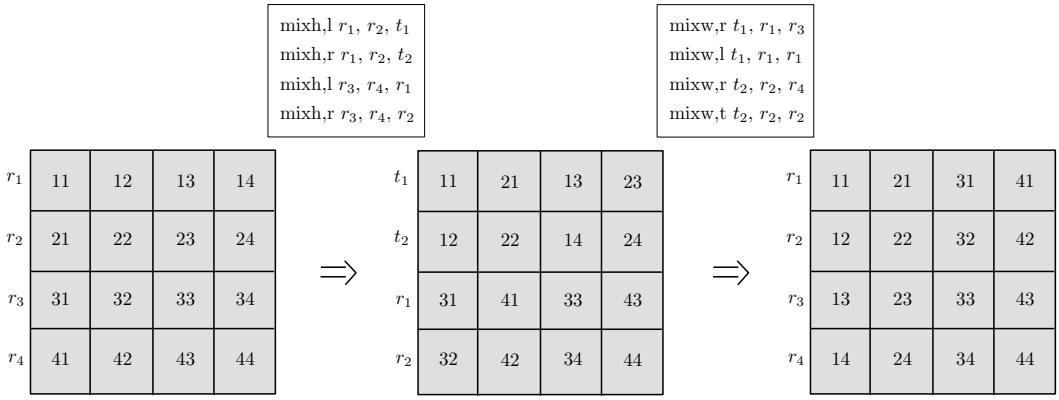

FIGURE 1.11: Matrix transpose using *Mix* instruction

r_1, r_2, r_3 and r_4 are input registers. Each register contain four packed subwords/pixels. t_1 and t_2 are the temporary registers used to store the intermediate results. Each *Mix* instruction takes the subwords of two registers mix them and gives results in output register. In the first step the subwords of registers r_1, r_2, r_3 and r_4 are mixed using four *Mix* instructions and the results are stored in registers t_1, t_2, r_1 and r_2. In the second step the subwords of registers t_1, t_2, r_1 and r_2 are mixed using four *Mix* instructions. At the output we get the transpose of input matrix. For (4 × 4) matrix it requires eight *Mix* instructions to transpose. Without using *Mix* instruction this task needs more than 20 instructions.

1.5.4 PERMUTE instruction in MAX-2

Permute instruction gives the different permutations of subwords stored in the input register [59] [60]. *Permute* instruction is helpful when the arrangement of subwords needs to be changed within the register for certain computation. Some of the different subwords permutations which can be obtained using *Permute* instructions are given in Figure 1.12.

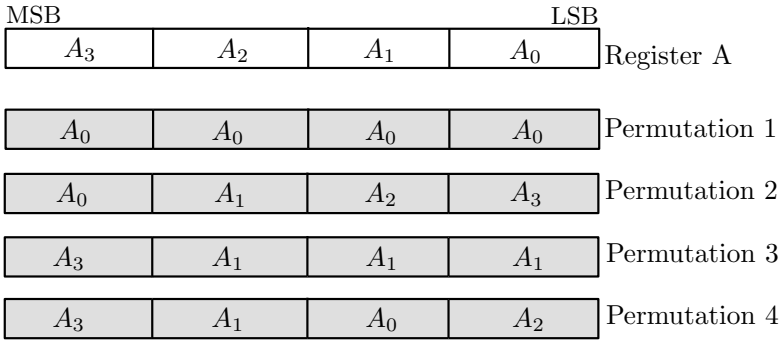

FIGURE 1.12: Different permutations of subwords

In permutation 1, subword A_0 is replicated in resultant register. This permutation is useful when A_0 is a single scaler which needs to perform certain operations with multiple subwords packed in a register. Therefore using the *Permute* instruction multiple copies of subword are obtained in single register and then the required operations are performed in parallel. In permutation 2, the order of subwords is reversed. This permutation is useful when switching from big-endian to little-endian format or vice versa is required. *Permute* instruction also allows to replicate any specific subword for specific number of times. For instance in permutation 3, the subword A_1 is replicated three times. Similarly, Permutation 4 provides the rearrangement of subwords in particular order.

In addition to these instructions, there are several other instructions which are used to support operations on packed subwords in different processors with multimedia extensions. For example in MMX [60], *unpack* instruction is used to expand smaller subwords to larger data width (8-bit to 16-bit subwords, 16-bit to 32-bit subwords etc.). Similarly *pack* instruction is use for contraction of subwords to smaller data width. Other most commonly used instructions in multimedia extension sets are *PshL, PshR* (parallel shift left and right), *Pcmp* (parallel subword compare), *Padd* (parallel addition) and *Psub* (parallel subtraction) etc.

1.5.5 Memory instructions

As in SWP, computations are performed on packed subwords therefore it is necessary to load/store multiple subwords from/to the memory [60]. In multimedia applications, usually the pixels are stored in the memory in a symmetrical manner. Therefore the memory can be accessed in a predictable way which will help to reduce the cache misses. A single *load* instruction in SWP processor loads multiple packed subwords in the internal register of the processor. This will reduce the number of instructions required to load the data in the processor. Without SWP single load instruction bring single data item in the processor. Similarly the *store* instruction in SWP processor also stores multiple subwords in the memory. Therefore in multimedia applications, SWP help to use the memory more efficiently.

1.5.6 Assembly code with and without SWP instructions

To see the effect of using SWP instructions, we can execute the array addition algorithm (explained in Section 1.3) by using simple and SWP instructions. Without SWP capabilities, the assembly language instructions for GPP to execute the computations shown in Figure 1.3 are given below:

```
01.       ldi 99, Ri;              Initialize counter to 99
02.  Loop:lds,ma 1(Raddrx),Rx;     Load one 16-bit number in register Rx
03.       lds,ma 1(Raddry),Ry;     Load one 16-bit number in register Ry
04.       add Rx, Ry, Rz;          ADD operator without SWP
05.       addibf,<-1, Ri, loop;    Decrement counter
```

As shown above, 64-bit ADD operator is instantiated 100 times to perform required computations on a processor without SWP capability. Now let us perform the same computations on SWP processor (Figure 1.4) using SWP instruction set. SWP instructions required to perform these computations are given below.

```
01.       ldi 24, Ri;               Initialize counter to 24
02.  Loop:ldds,ma 4(Raddrx),Rx;     Load four 16-bit numbers in register Rx
03.       ldds,ma 4(Raddry), Ry;    Load four 16-bit numbers in register Ry
04.       hadd Rx, Ry, Rz;          ADD operator with SWP capability
```

```
05.       addibf,<-1, Ri, loop;    Decrement counter
```

Using SWP load instructions (*ldds,ma*), four 16-bit data items are loaded in the internal registers simultaneously. In the next step add operation is also performed simultaneously on four packed subwords using *SWP ADD* operator. Due to the use of SWP, only 25 instantiations of *SWP ADD* operator are required.

1.6 Subwords sizes in SWP architectures

Subword is a lower precision unit of data contained within a word. In SWP, multiple subwords are packed into a word size register and then the processing of whole word is done [98]. The subwords in a single word can be of different sizes but in order to reduce the complexity usually subwords are of same size.

1.6.1 Parallelism Vs subword size

The degree of parallelism in SWP architectures depends on the size of subword. Smaller the subword size higher will be the parallelism. In general the selection of subword size is a trade off between parallelism and data precision. For instance in a processor of word size 64-bits, subword size of 8-bit do not give sufficient precision and on the contrary subword size of 32-bits do not give sufficient parallelism as it gives parallelism of two operations per subword instruction only. Hence the best choice left for subword size is 16-bit. Therefore any data element which is less than 16-bit is either zero padded (unsigned) or sign extended (signed) to 16-bit before the computation.

1.6.2 Support for multiple subword sizes

Based upon the requirements, SWP processor can support more than one subword size. Greater the number of supported subword sizes, higher will be the complexity of SWP processor. If the input data size accurately matches with one of the supported subword size then the efficiency of SWP processor will be high. Depending upon the input data size, the required subword size is selected with the help of SWP control signals. These control signals directs the SWP operators to perform operations on one particular subword size. If the word size of processor is 32 bits then some useful subword sizes are 8 bits, 10 bits, 12 bits and 16 bits. Hence an instruction operates on either four 8-bit subwords or three 10-bit subwords or two 12-bit subwords or two 16-bit subwords in

parallel or on one 32-bit word obviously. The SWP control signals corresponding to each subword size will be different. For 8, 10, 12 and 16-bit subword sizes the corresponding control signals SW_8, SW_{10}, SW_{12} and SW_{16} can be used. These control signals selects one of the subword size. When any of these control signal is active, the SWP operator reconfigure itself to perform operations on corresponding subword size.

1.6.3 Coordination between data and subword size

The performance of SWP processor highly depends upon the coordination between input data size and the supported subword size. If the data size is same as subword size then the efficiency of SWP processor will be high. Difference in data and subword sizes results in the under utilization of processor resources. The selection of subword size in SWP architectures highly depends on the applications for which they are designed. In general purpose processors (GPPs), computations are performed for variety of applications. Hence it is not possible to exactly predict the size of data on which the processor has to operate. Therefore the subword sizes of SWP operators in GPP are not meant for any particular class of applications. However in the processors which deal with specific type of applications like multimedia etc., the coordination between data sizes and subword sizes can be increased. In multimedia applications extensive computations are performed on pixel data. Therefore the subword sizes in multimedia processors should be in coordination with pixel sizes. For high performance, multimedia processor should contain the subword support for all the possible pixel sizes. Based on the scope of this thesis, the subword sizes can be categorize as *Classical subword sizes* and *Multimedia subword sizes*.

1.6.4 Classical subword sizes

In the existing SWP capable processors [20, 28, 56, 57], the classical choices for subword sizes are usually 8, 16, 32 bits etc.(power of 2). The reason behind the selection of these subword sizes being the less complexity of SWP operators, especially when subword sizes are multiple of the smallest subword size ($8 \times 2 = 16$, $16 \times 2 = 32$ and so on). The complexity of classical SWP operators is also less because most of the times the supported subword sizes have uniform arithmetic relation with word size of processor ($8 \times 8 = 64$, $16 \times 4 = 64$, $32 \times 2 = 64$ etc.). The SWP processors which are based on classical subword sizes are suitable for general purpose applications in which the input data sizes are byte oriented. However for any particular class of applications, these subword sizes results in under utilization of processor resources. For instance in most modern multimedia applications, pixel/data sizes are 8, 10, 12 or sometimes

16 bits which are not in coordination with classical subword sizes. Therefore existing classical SWP operators do not give full utilization of processor resources when working on multimedia applications. As an example consider certain multimedia applications related to medical images. In these images the pixel size is usually 12-bit. Let us assume that these applications are executed on a SWP processor that supports only classical subword sizes (8, 16, 32-bit etc.). The most appropriate available classical subword size for these applications is 16-bit. However performing computations on 12-bit pixels using subword size of 16-bit results in the under utilization processor resources such as registers, computational units, datapath etc. For each subword computation the resources corresponding to last 4 bits (16 - 12 = 4) will not be utilized. This under utilization of processor resources is shown in Figure 1.13 for 64-bit processor.

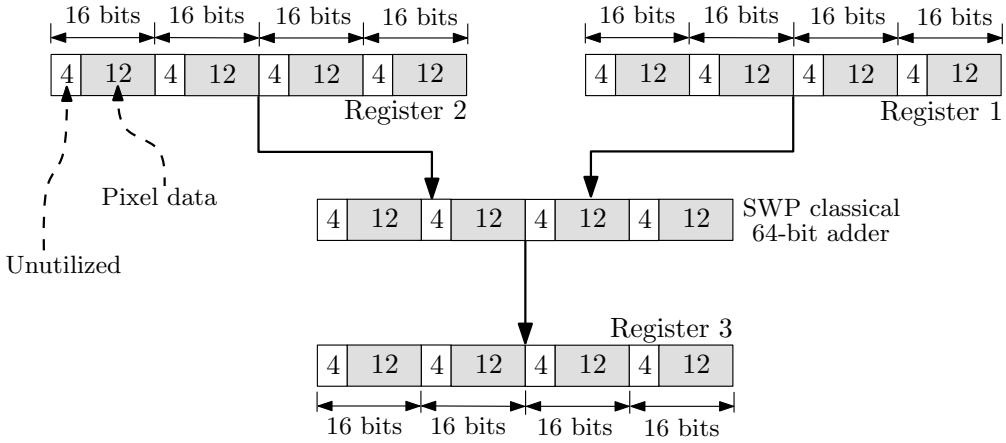

FIGURE 1.13: Classical subword sizes in SWP

As shown in Figure 1.13, due to non availability of 12-bit subword size in the SWP processor's architecture, almost 25% (16/64 = 0.25) of processor resources have not been utilized. This would result in the overall decrease in the efficiency of SWP processor.

1.6.5 Multimedia subword sizes

Multimedia oriented subword sizes are those sizes which are in coordination with pixel sizes in modern multimedia applications. Pixel data in multimedia applications are usually 8, 10, 12 or 16 bits. In SWP processors which are designed for multimedia applications, supported subword sizes should also be in coordination with pixel data. The efficiency of SWP processor can be increased for multimedia applications if the SWP operators are designed based on multimedia oriented subword sizes rather than classical subword sizes. The multimedia subword sizes (8, 10, 12 and 16bits) do not have any uniform arithmetic relationship with the word size of processor resulting in the increase of implementation complexities but at the same time these operators increases the speedup

of processor especially for multimedia applications. For instance, the multimedia computations example shown in Figure 1.13 can be executed on a SWP processor which supports multimedia oriented subword sizes (8, 10, 12 and 16-bit) rather than classical subword sizes as shown in Figure 1.14.

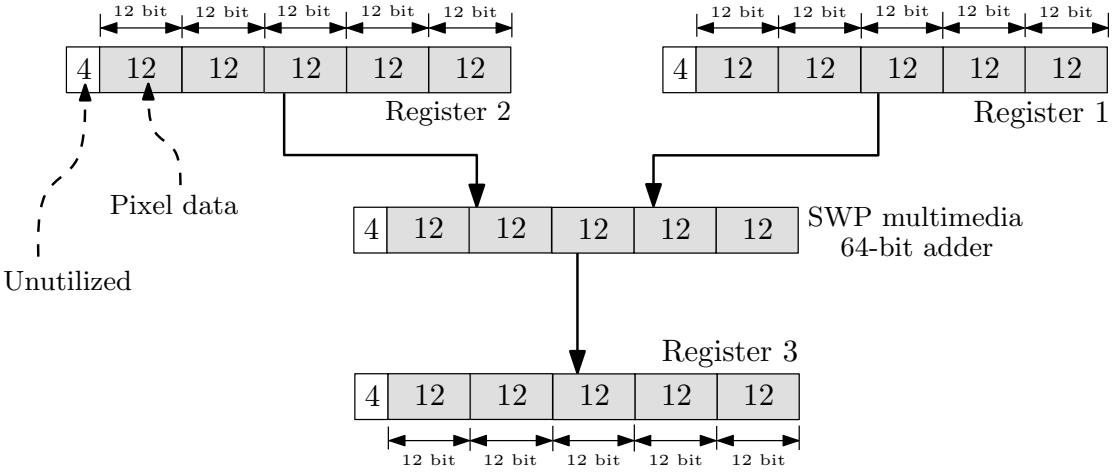

FIGURE 1.14: Multimedia oriented subword sizes in SWP

As shown in Figure 1.14, due to the use of multimedia oriented subword sizes, the resources corresponding to each packed subword are fully utilized. Parallel operations are performed on five 12-bit pixels simultaneously. Where as in classical SWP operator due to the under utilization of resources, parallel operations are performed on four 12-bit pixels only. Compared to the classical SWP processor, the under utilization of processor's resources has been reduced from 25% to 6% (4/64 = 0.0625).

1.7 Word size in SWP operators

Along with subword sizes, the resource utilization in SWP operator also depends on the word size. The word size is chosen in such a way to increase the utilization of available resources when working on different subword sizes. In classical subword sizes (8, 16, 32-bit), it is easy to select the word size of operator as the least common multiple (LCM) of all the supported subword sizes. For classical subword sizes (8, 16, 32-bit), the appropriate word sizes are 32-bit or 64-bit etc. By using these word sizes, the utilization of SWP operator will be maximum when operating on different classical subword sizes. However in case of multimedia oriented subword sizes, the LCM of all the supported subword sizes is not a feasible option for word size. For instance for multimedia oriented subword sizes (8, 10, 12, 16-bit), the LCM comes out to be 240-bit. Which is not practical for multimedia processors. Therefore the word size for multimedia oriented

SWP operator is selected in such a way to maximize the utilization of processor when working on different pixel sizes.

In this thesis, for the designing of multimedia SWP operators we have chosen word length of 40-bit. This 40-bit is chosen because it gives good efficiency/complexity trade off and ensures better resource utilization with different multimedia oriented pixel sizes. The utilization ratio of classical and multimedia SWP operator when working on different pixel sizes is shown in Table 1.3.

	Classical SWP Operator (Word size = 32-bit) (Subword size = 8, 16-bit)		**Multimedia SWP Operator** (Word size = 40-bit) (Subword size = 8, 10, 12, 16-bit)	
	Operations	**Utilization(%)**	**Operations**	**Utilization(%)**
a(8) OP b(8)	4	100	5	100
a(10) OP b(10)	2	62	4	100
a(12) OP b(12)	2	75	3	90
a(16) OP b(16)	2	100	2	80
a(32) OP b(32)	1	100	1	80
a(40) OP b(40)	0	X	1	100

TABLE 1.3: utilization of SWP operator for classical and multimedia subword sizes

As shown in Table 1.3, the word length of classical SWP operator is chosen as 32-bit and it can support subword sizes of 8, 16 and 32 bits. On the other hand the word size of multimedia SWP operator is 40-bit and it can support multimedia oriented subword sizes of 8, 10, 12, 16 bits. The utilization ratio shows the percentage of SWP operator resources utilized to perform certain arithmetic operation on specified pixel sizes. Higher the utilization ratio, better will be the performance of SWP operator. When the pixel size is 8-bit, the utilization of both classical and multimedia operator is maximum (100%). When the pixel size is 10-bit, the utilization ratio of multimedia SWP operator is 100% because it utilizes all its data width. However the utilization ratio of classical SWP operator is only 62% for 10-bit pixel sizes. This decrease in utilization ratio occurs because the classical SWP operator uses 16-bit subword sizes to operate on 10-bit pixel sizes. Due to this reason, the number of parallel operations performed by SWP multimedia operator on 10-bit pixels are also more (4 operations) than classical SWP operator (2 operations). Similarly for other pixel sizes the utilization ratio and number of parallel operations for both classical and multimedia SWP operator are shown in Table 1.3. On average when working on multimedia pixel sizes, the utilization ratio and the number of parallel operations performed by SWP multimedia operator are more compared to classical SWP operator. The reason being the appropriate selection of word and subword sizes in SWP multimedia operator. SWP multimedia operator uses subword sizes which are in coordination with pixel data sizes in multimedia applications. Moreover the word size of SWP multimedia operator gives good efficiency/complexity

trade off when working on different pixel sizes. Figure 1.15 shows the utilization of different word length operators for different multimedia oriented subword sizes.

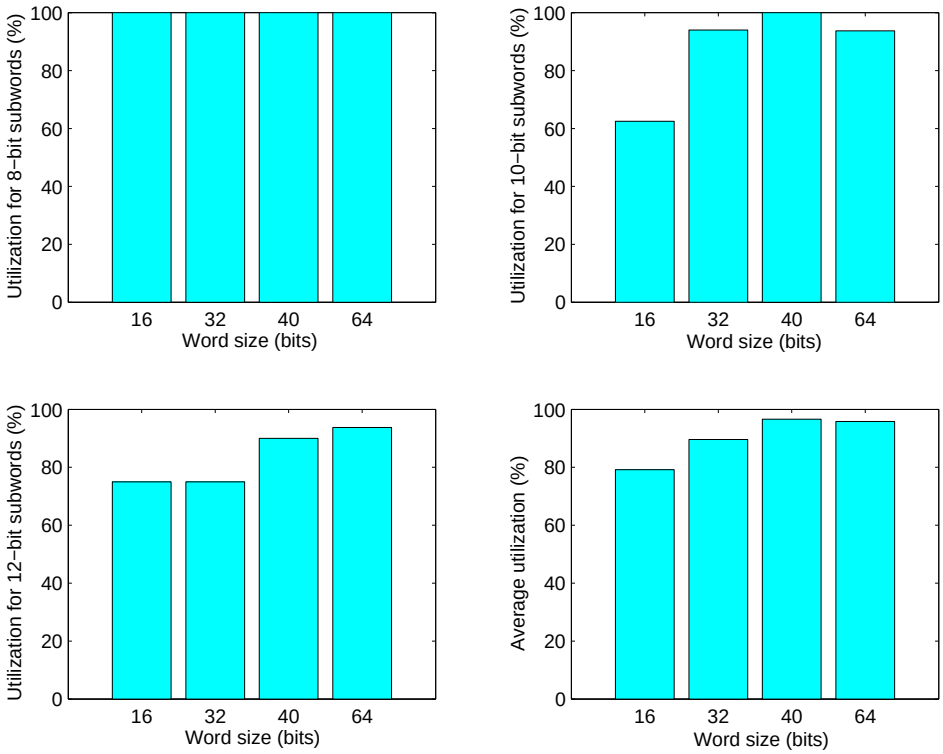

FIGURE 1.15: Utilization of different word length processors for different multimedia oriented subword sizes

As shown in Figure 1.15, for 8-bit subword size, the utilization on all 16, 32, 40 and 64-bit machines are same (100%). For 10-bit subword size, the utilization of 16, 32, 40 and 64-bit processors are 63%, 94%, 100% and 94% respectively. Similarly for 12-bit subword size, the utilization of 16, 32, 40 and 64-bit processors are 75%, 75%, 90% and 94% respectively. On average for most commonly used multimedia pixel sizes (8, 10 and 12-bit) the percentage utilization of 40-bit word size processor is greater than other word sizes. For 16, 32, 40 and 64-bit word processor, the average utilization for multimedia oriented subword sizes are 79%, 90%, 97% and 96% respectively. The other reason for the selection of 40-bit word size is that it gives maximum utilization (100%) for most of the pixel sizes which are used frequently in multimedia applications. SWP Processor with 32-bit and 64-bit word size gives 100% utilization only when pixel size is 8-bit. On the other hand, the SWP processor with 40-bit word size gives 100% utilization for 8-bit and 10-bit pixel sizes.

1.8 Limitations of SWP

SWP increases the performance of processors through parallel processing of subwords. The SWP processor increases the resource utilization of processor when working on low precision pixel data of multimedia applications. However there are certain limitations of SWP architectures. Most of these limitations are highlighted when the applications which are not inherently parallel are executed on SWP architectures. Some of the limitations are given below.

- **Scalability** Scalability of SWP processor is complex task compared to conventional processor which operates on words of data. In SWP processors, arithmetic operators are designed in such a way to operate on subwords of different sizes. Designing of these SWP operators is complex compared to simple operators. Therefore the simple scalability of SWP processor is not possible. To scale up the SWP processor or to increase the number of supported subword sizes, different processing units needs to be designed at subword level.
- **Optimization of SWP instruction set** SWP processors used specialized instruction sets to enhance the performance for low precision pixel data. Sometimes these instructions change the order of the subwords or combine the subwords packed in different registers in required order. These type of instructions are specific to SWP processing. The compiler of general purpose processor requires more efforts to optimize the SWP instructions compared to the conventional instructions. To attain the maximum advantage of SWP, the compiler of GPP should be friendly with SWP instruction set.
- **SWP overheads** The overheads of including SWP capability in processor's architecture is the management and alignment of subwords. Before performing the computation, the subwords are arranged within the registers. Sometimes the subwords are either expanded or contracted for alignment purposes. Due to these overheads, the SWP architectures normally require more resources compared to simple designs. However the speed enhancement achieved through the parallel processing of subwords undermines the effects of these overheads.
- **Memory Misalignment** SWP processor performs computations on subwords of data. For parallel processing, multiple subwords are accessed from memory. In the memory the data is usually stored as words of data at each location. To access any particular subword it is necessary to address the whole word in the memory. If the subword data is properly aligned at each memory location then in one memory reference multiple subwords can be accessed. For this purpose the subword data needs to be aligned in the memory.

The advantages of using SWP in multimedia applications is much more compared to the its overheads. Due to this reason most of the processors which have to deal with multimedia applications incorporate SWP capability. Different synthesis tools like Synopsys, Altera etc. also contain SWP operators in their libraries. These operators perform parallel operations on subwords. However most of these operators are based on classical subword sizes.

1.9 Contributions and Organization of this Thesis

The context of this thesis is based upon the ROMA project [29]. ROMA stands for *Reconfigurable Operators for Multimedia Applications*. This project was started under the *Architectures du Futur*(ANR-06-ARFU6-004-01) program of ANR (Agence Nationale de la Recherche). In multimedia applications, image processing is the major challenge embedded systems have to face. It is computationally intensive with power requirements to meet. Image processing at pixel level, like image filtering, edge detection, pixel correlation or at block level such as motion estimation have to be accelerated. To achieve these objectives, the ROMA project proposes to develop a reconfigurable processor, exhibiting high silicon density and power efficiency, able to adapt its computing structure to computation patterns that can be speed-up and/or power efficient. On the contrary of previous attempts to design reconfigurable processors, which have focused on the definition of complex interconnection network between simple operators, the ROMA project aims to design a pipeline-based low-power coarse grain reconfigurable operators to avoid traditional overhead, in reconfigurable devices, related to the interconnection network [29]. Due to the requirement of expertise in various design fields, a consortium was formed to work on different parts of ROMA project. This consortium includes four well known research Labs of France: IRISA [1], CEA LIST [2], LIRMM [3] and Thomson R&D France [4]. An overview of different blocks of the ROMA processor are shown in the Figure 1.16. The main parts of the processors are a set of reconfigurable operators, memory banks, interconnect between memory banks and operators, interconnect between operators, and a control processor. A DMA is used to access external data through the memory banks.

In this processor, the data flow and instruction flow to different units are managed using control modules. The data for the computations is provided to the different arithmetic operators using the memory banks. Depending on the instruction operation code

[1]Institut de Recherche en Informatique et Systemes Aleatoires: http://www.irisa.fr/
[2]Laboratoire d'Intégration des Systèmes et des Technologies : http://www-list.cea.fr/
[3]Laboratoire d'Informatique de Robotique et de Microelectronique de Montpellier : http://www.lirmm.fr/
[4]Thomson Research & Development, France http://www.thomson.net

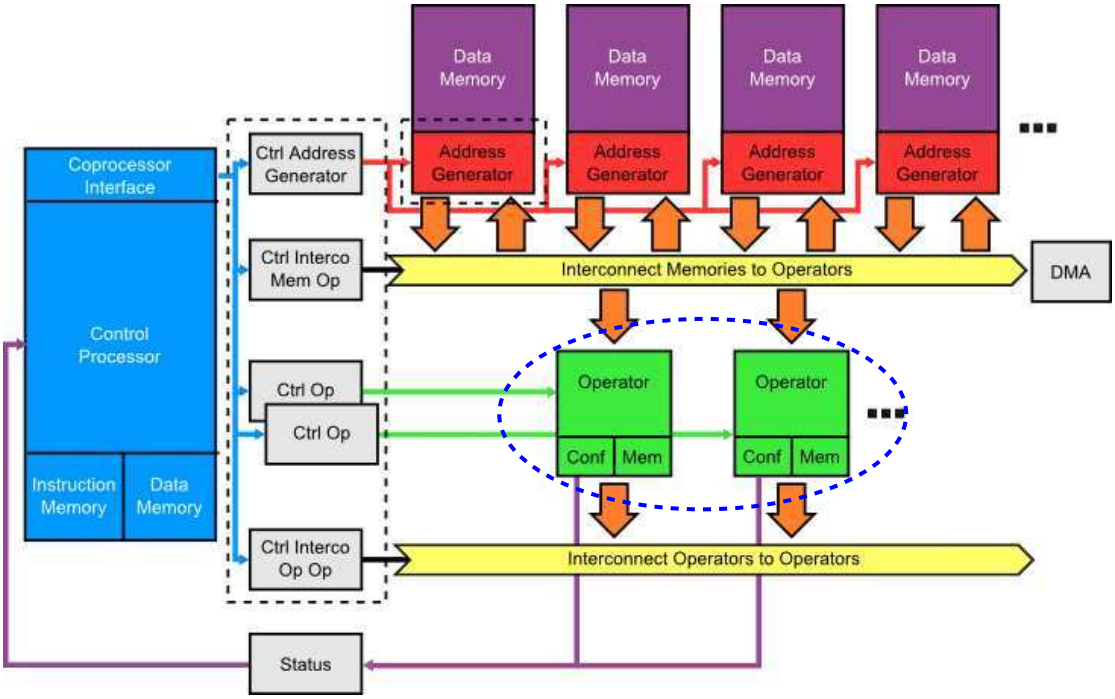

FIGURE 1.16: Block diagram of ROMA processor

(op-code), the control units generate activation signals for the required units. After performing the computations, the results are stored back in the memory banks or are propagated to another configurable operator. Instead of using complex interconnection network, the reconfigurability is provided inside the different operator's architecture. Under the umbrella of the ROMA project, IRISA/CAIRN lab is involved in the development of the reconfigurable arithmetic operators for the multimedia processor. As explained in this chapter, SWP capability can be introduced in the architecture of the operators to improve the performance. It is the context of this thesis. Numerous SWP arithmetic operators have been proposed in literature to perform parallel computations. However all these operators works on classical subword sizes which results in the under utilization of processor resources for multimedia applications. In this thesis, the design of different multimedia SWP operators are proposed using subword sizes which are in coordination with pixel sizes in multimedia applications. Due to this coordination, the processor utilizes the available resources more efficiently. Which ultimately increases the overall performance of the processor. Proposed SWP basic arithmetic operators are then used to design reconfigurable SWP operator for multimedia applications. This operator can be reconfigured to perform variety of multimedia operations on different pixel size data without any reconfiguration time overheads. Along with parallelism, the internal speed of different processing units is also improved by using SWP technique on redundant number system rather than binary number system. Redundant number system increases the speed of different arithmetic operations through carry propagation

free addition property. Hardware architecture for motion estimation algorithm is developed using the proposed multimedia SWP operators. Results shows that due to the use of multimedia oriented SWP operators, the performance of processor is increased when working on multimedia applications. Contributions which are made during this thesis work are given below.

- **SWP for multimedia operator design [52]:** The purpose is to introduce SWP capabilities in the design of basic arithmetic operators. Different algorithms are used for the designing of basic operators like ADD, SUB, MULT and MAC. Algorithm which requires minimum resources for SWP capability are analyzed. Both classical and multimedia oriented subword sizes are considered. The overheads for incorporating SWP in operator design are analyzed.

- **SWP multimedia operator design [53]:** The aim is to highlight the non coordination between classical subword sizes and the pixel sizes in modern multimedia applications. Due to this non coordination, the under utilization of processor resources occurs when working on pixel data. Architectures of different multimedia oriented basic arithmetic units are proposed to enhance the overall performance of processor. The efficiency of these operators are evaluated on different target technologies.

- **Reconfigurable SWP Operator for Multimedia Processing [50]:** The goal is to implement reconfigurable multimedia SWP operator. This operator can be used to perform variety of multimedia operations on different size pixel data. Reconfigurability is provided at both operation level and the data size level. Reconfiguration overheads are reduced to ensures better performance. Due to parallel processing of pixel data, the proposed operator can perform different multimedia operations in less time compared to other operators.

- **Reconfigurable Operator Based Multimedia Embedded Processor [69]:** The objective is to limit the interconnection overheads in the design of reconfigurable multimedia processor. Previous reconfigurable processors suffer from reconfiguration overheads and do not meet low power constraints. The design of a reconfigurable processor based on a coarse-grain granularity tailored for multimedia applications is presented. The architecture is flexible and scalable. Coarse-grain operators can be optimized in term of the function they implement, the data word-length and the parallelism speed-up.

- **High speed reconfigurable SWP operator for multimedia processing using redundant data representation [51]:** The purpose is to design SWP reconfigurable multimedia operator using redundant data representation. Multimedia

oriented SWP increase the performance through parallel processing of subwords. Where as the internal speed of different arithmetic units are increased through carry propagation free addition on redundant data representation rather than binary data representation. The advantages of both parallelism and high speed arithmetic operations are combined in proposed reconfigurable operator design. The power consumed by reconfigurable operator while performing different multimedia operations are also analyzed. The overheads corresponding to the use of redundant number system are compared with the overheads of operators based on binary number system.

1.9.1 Organization of thesis

The work presented in this thesis is divided into seven chapters. Here is an outline of chapters.

Chapter 1 - Subword Parallelism SWP in operator design:

The chapter is written with an intent to initiate the terms and concepts to the reader which are used in the subsequent parts of this thesis. The chapter starts by describing the concept of parallelism for the performance enhancement. Different levels at which the parallelism can be applied are explored in detail. Parallelism in the context of multimedia processing is also explained. An overview of different architectures which are available for the processing of multimedia applications is given. Subword parallelism (SWP) is one of the techniques which are used to exploit data level parallelism that exist in different applications. Advantages of using SWP for enhancing the performance of processor when working on multimedia applications are given in detail. Different general purpose processors (GPP) which contain SWP extended instruction set for multimedia processing are also given. The basic requirements for incorporating SWP capability in the architecture of processor are elaborated in detail. These requirements include SWP operators, SWP instruction set, availability of low precision data etc. To clarify the concept, few SWP instructions and their functionality are explained with the help of examples. Selection of subword sizes plays vital role in the performance of SWP processor. The effect of using classical and multimedia oriented subword sizes on the resource utilization of processor is highlighted. Higher coordination between pixel sizes and subword sizes increases the performance of processor. Word size of SWP operators are selected on the basis of efficiency/ complexity trade off and ensure better resource utilization. Few limitations of SWP processors are also given. At the end the contributions and the

organization of this thesis are given.

Chapter 2 - SWP basic operators design:

The contents of this chapter are based on our publication [52] and [53]. The objective of this chapter is to design basic arithmetic SWP operators which can perform parallel computations on different size pixel data. These operators can then be used in any processor to enhance the performance for multimedia applications. Different SWP arithmetic operators which are designed in this chapter includes ADD, SUB, MULT and MAC. These SWP operators are designed for both classical and multimedia oriented subword sizes and their performances are compared. Classical subword sizes requires less resources compared to multimedia oriented subword sizes due to uniform arithmetic relation between word and subword sizes. However the performance of SWP multimedia operators is better due to high resource utilization. SWP technique is applied on different algorithms which are used to design arithmetic operators. For each operator, the algorithm which requires minimum resource increase to upgrade it with SWP capability is highlighted. Compared to other basic operators, the multiplication operation consumes lot of hardware resources. Introducing SWP capability in multiplication operator also requires more resources. However in this chapter multimedia oriented SWP multiplier is proposed which consumes less resources for SWP capability. Its architecture is based upon SWP multiplier [56] proposed for classical subword sizes. The performance of multimedia oriented SWP multiplier is analyzed by implementing it for different word and subword sizes. The area, critical path and power analysis is performed. The SWP basic arithmetic operators designed in this chapter are then used in the designing of multimedia operator.

Chapter 3 - SWP in multimedia operations:

In this chapter, different operations which are required in the multimedia applications are implemented using SWP basic arithmetic operators explained in Chapter 2. These multimedia operations include sum of absolute value difference (SAD) for motion estimation, sum of product (SOP) for discrete cosine transform (DCT) etc. Other general purpose operations like sum of additions/subtractions ($SWP\sum(a \pm b)$) are also implemented using SWP operators. SWP absolute difference unit |*a* - *b*| used in SAD operator is implemented using different algorithms. These algorithms are then compared on SWP platform before using in SAD operator. SWP SAD operator performs the motion estimation computations in less time compared to conventional word oriented operator. Similarly the performances of other SWP multimedia operators are also compared. The overheads for incorporating pixel oriented SWP capability in multimedia operators are

also analyzed. In all these operators, hardware resources are fully utilized due to better coordination between pixels and subword sizes which ultimately increases the performance of processor.

Chapter 4 - Reconfigurable SWP operator for multimedia processing:

The contents of this chapter are based on our publication [50] and [69]. In this chapter reconfigurable multimedia operator is proposed. This operator eliminates the need of reconfiguration time and provides the reconfigurability at both operation level and data size level. Variety of multimedia operations can be performed on different size pixel data. The inputs to reconfigurable operator consist of word size vectors which contain packed subwords. To perform any particular operation, the control unit selects the appropriate subword size and activates the units which need to perform computations. Arithmetic units which are used in reconfigurable operator contain multimedia oriented SWP capability. These units perform operations on packed subwords in parallel. The activated units reconfigure itself for different pixel size data using SWP control signals. Proposed reconfigurable operator is synthesized on different technologies and the results are analyzed. Compared to the conventional SWP operator used in different DSP chips, the proposed reconfigurable operator performs different multimedia operations in less number of cycles. This operator can be used as dedicated unit or co processor in any multimedia processor to enhance the performance.

Chapter 5 - SWP using redundant representation:

The contents of this chapter are based on our publication [51]. SWP increases the performance through parallel processing of data. However the internal speed of different processing units also plays very important role in the overall efficiency of processor. The internal speed of arithmetic units can be increased by using carry propagation free addition on redundant number system rather than binary number system. In redundant system, numbers are represented by digits rather than bits. Each number can be represented by different combinations of redundant digits. This redundancy in representing the number help to realize high speed carry propagation free addition. Therefore to achieve parallelism along with high speed operations, multimedia oriented SWP capability is introduced in redundant number system for the first time. These arithmetic operators provide high resource utilization through multimedia SWP and high speed though the use of redundant number systems. The overheads of high speed SWP operators are analyzed on different target technologies. Reconfigurable SWP operator using

the redundant number system is proposed for performing different multimedia operations.

Chapter 6 - Motion estimation using SWP operators:

Motion estimation (ME) is most commonly used operation for video compression in multimedia applications. In ME algorithm the current block which needs to be transmitted is compared with different blocks in reference frame and the best match is searched. The block matching process requires lot of computations on pixel data. Conventional operators consume excessive time to find the best match in ME algorithm. To analyze the performance of proposed SWP operators, ME algorithm is implemented using SWP operators on different pixel size data. SWP operators perform parallel computations on pixel data and accomplish the block matching process in less time compared to conventional operators. During these experiments, different search algorithms are used which includes full search and diamond search algorithms. ME algorithm is applied on variety of Search area image sizes (48×48, 32×32, 16×16) and block sizes (16×16, 8×8). Results show that SWP operator can perform ME computations more efficiently compared to conventional operators.

Chapter 7 - Conclusions:

A brief summary of the important points developed in different chapters of this thesis is given. We first look back and summarize what we have achieved, and we then look ahead to outline what can be accomplished next.

1.10 Conclusions

This chapter elaborates the need of SWP to increase the performance of processor for multimedia applications. Different parameters which effect the performance of SWP processor are discussed in detail. Although SWP has already been used in many processor designs, but due to the use of classical subword sizes the maximum utilization of processor resources cannot be achieved when working on modern multimedia applications. To overcome this non coordination, multimedia oriented subword sizes are introduced in SWP architectures. These subword sizes are in coordination with pixel sizes and increases the performance of processors. In the next chapter the SWP basic operators are designed using classical as well as multimedia oriented subword sizes.

Chapter 2

Design of SWP basic operators

For designing SWP processor, the basic requirement is SWP arithmetic operators to execute parallel operations on subwords. These operators perform the same operation on all the subwords packed in a word size input registers. The size of subword data is specified by SWP control signals. Based on the requirements, SWP operator can support multiple subword sizes. The complexity of SWP operator depends upon the number and the size of subwords supported by the operator. In this chapter we will discuss the designing of SWP basic operators in detail. The contents of this chapter are based on our publications [52] and [53].

The rest of this chapter is organized as follows: Section 2.1 gives an overview of SWP operator design. Section 2.2 describes the designing of SWP ADD operator using classical and multimedia oriented subword sizes. SWP ADD operators using different algorithms are then compared with their simple versions. Section 2.3 explains some of the algorithms which can be used in the designing of multiplication operator. The architecture of SWP multiply operator using classical subword sizes are described in detail. The overheads for using SWP capability in multiplier architecture are also discussed. This section also explain the architecture of SWP multiplier using multimedia oriented subword sizes. Different steps like generation of partial products, addition of partial products etc. are discussed in detail. Finally the analysis of SWP multimedia multipliers is presented using different word and subword sizes. Section 2.4 describe the implementation of SWP MAC operator using classical and multimedia oriented subword sizes. The area and speed overheads for using SWP in operator design are also discussed. Finally the chapter is concluded in Section 2.5.

2.1 SWP operator design

For performing any computation in SWP processor, SWP enable basic arithmetic operators are required which include add, subtract and multiply. Other application specific operator units like multiply and accumulate unit (MAC), sum of absolute difference (SAD) etc. can be added to specialize the processor to a specific domain like multimedia. By SWP enable operator we mean the operator which can perform parallel operation on subwords packed into word size registers. SWP enable operators are designed in such a way that they can operate on data of different subword sizes and also on data of word size depending upon the requirement.

2.1.1 Complexity of SWP operators

SWP enable operators are more flexible compared to simple operators so it is obvious that most of the time their implementation requires more resources compared to their simple versions. However in this chapter, efforts are made to design SWP enable operators with minimum resource increase. The complexity of SWP operators depends upon two main factors.

- **Number of supported subword sizes** The complexity of SWP operator increases with the increase in the number of subword sizes supported by the operator. For each subword size, SWP operator has dedicated control logics which increases with the increase in the number of subword sizes. For instance, the complexity of SWP operator which supports four subword sizes will be high compared to the SWP operator which supports only two subword sizes. Greater the number of supported subword sizes, higher will be the flexibility of the operator. The SWP operator with multiple subword sizes is more robust as it gives better performance for different data sizes.

- **Subword size** The complexity of SWP operator also depends upon the subword sizes supported by SWP operator. If the word size of the operator is multiple of subword sizes then the complexity of SWP operator will be less. For instance in 64-bit processor, some of the subword sizes are 4, 8, 16, 32-bit. The boundaries of these subword sizes overlaps with each other which reduces the overall control logics required to manage different subwords. In this work, these subword sizes are called *classical subword sizes* as explained in section 1.6.5 of Chapter 1. However if the word size of the operator is not a multiple of subword sizes then the complexity of SWP operator will be high. This occurs when we want to design the SWP operators for any particular class of applications like multimedia etc.

In multimedia applications, the efficiency of SWP operator will be high when the supporting subword sizes are in coordination with pixel sizes. Pixel sizes in multimedia applications are 8, 10, 12 or sometimes 16-bit. By using these subword sizes the SWP operators utilizes the available resources more efficiently when working on multimedia applications. However due to non uniformity in subword sizes, the complexity of SWP operator also increases. These subword sizes are termed as *multimedia subword sizes*. In this chapter SWP operators are designed using both classical and multimedia oriented subword sizes.

Since the introduction of SWP in processor's designs, work has been done on the implementation of SWP enable operators which help to increase the performance of SWP processors. In most of these works the operator's architectures are based upon low precision classical data sizes (8, 16, 32-bit etc.) rather than multimedia oriented pixel sizes (8, 10, 12 or 16-bit). These operators improve the performance of SWP processors to some extent. In [28] SWP enabled adders is proposed that introduces SWP capability by breaking the carry chain of carry look ahead adder using carry propagate and carry generate signals. In [56] an efficient architecture for SWP enabled multiplier is proposed. This multiplier implements SWP capability with the minimum increase in area resources and critical path. It uses simple AND operation for generation of partial products and then arrange them efficiently for each subword size. In [19] variable precision multiplier is proposed for FPGA platform. It uses minimum FPGA resources for implementing variable precision multiplier. In [20] a 64-bit fixed point vector multiply and accumulate (MAC) unit is proposed which supports multiple precision (one 64x64, two 32x32, four 16x16 or eight 8x8) operations. Its design is based on shared segmentation method. However all these SWP operators are designed using classical subword sizes (8, 16, 32-bit etc.) which results in the under utilization of processor resources for multimedia applications.

In the next sections classical as well as multimedia SWP enable basic operators are designed and their performance are compared with their simple versions. Simple and SWP versions of each operator is implemented using different algorithms. Both classical and multimedia oriented word sizes and subword sizes are considered. In multimedia oriented SWP operators, subword sizes of 8, 10, 12 and 16 bits are chosen based on data sizes of current multimedia applications.

2.2 Add operator

Adder is one of the basic operators which is required in every processor design. It is used to add signed as well as unsigned binary numbers. The signed and unsigned

addition process is same except in signed number addition, the unequal vectors are sign extended before the addition. The adders are used in the implementation of several other arithmetic operators like subtraction, multiplication etc. Therefore the efficient implementation of addition algorithm improves the performance of other arithmetic operations as well. Different adder architectures have been proposed to increase the performance of addition process. These adders include ripple carry adder RCA, carry look ahead (CLA), group CLA, carry save adder (CSA) etc. The basic units which are used in all these architectures are full adder (FA) and half adder (HA). FA is used to add two input bits and a carry bit from previous bit location. Where as the HA is used to add two bits only. These basic units are used at each bit location in adder architectures. The use of FAs in RCA is shown in Figure 2.1.

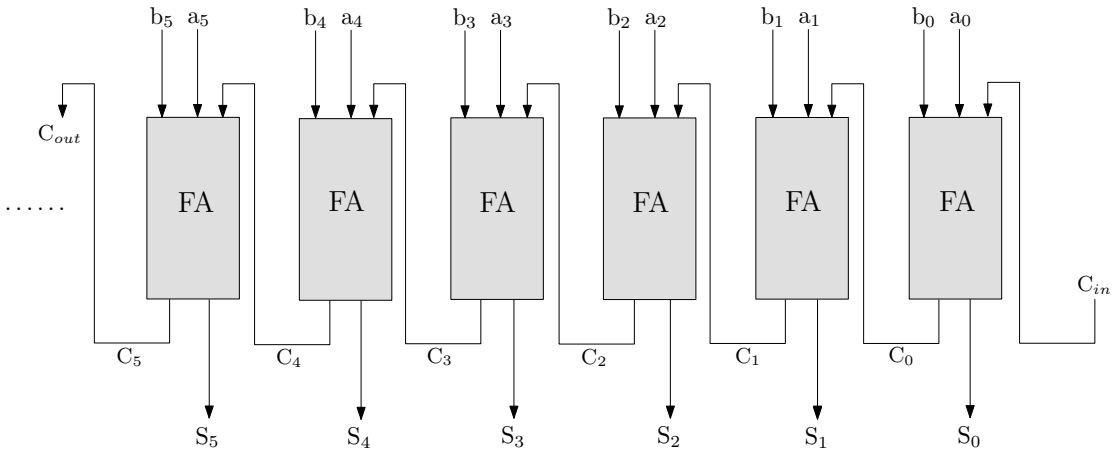

FIGURE 2.1: Ripple carry adder using FA

SWP increases the flexibility of adder by allowing the same operator to be used for different data length operations. Simple ADD operator performs the addition of two input vectors where as the SWP adder performs addition of several subwords packed in a word size register. There are several ways to introduce the SWP in basic ADD architectures using classical and multimedia oriented subword sizes.

2.2.1 Classical SWP adder

SWP technique can easily be applied to conventional adder architectures. Figure 2.2 shows a 16-bit adder with SWP capability. This adder is similar to a conventional adder which accepts two 16 bits numbers and produce 16 bit sum and 1 bit carry.

By blocking/unblocking of the carry chains that propagate the carry, addition of subwords of different sizes such as 4, 8, 12 or 16 can be obtained. For example by blocking the carry to propagate from bit 3 to bit 4, from bit 7 to bit 8 and from bit 11 to 12, the

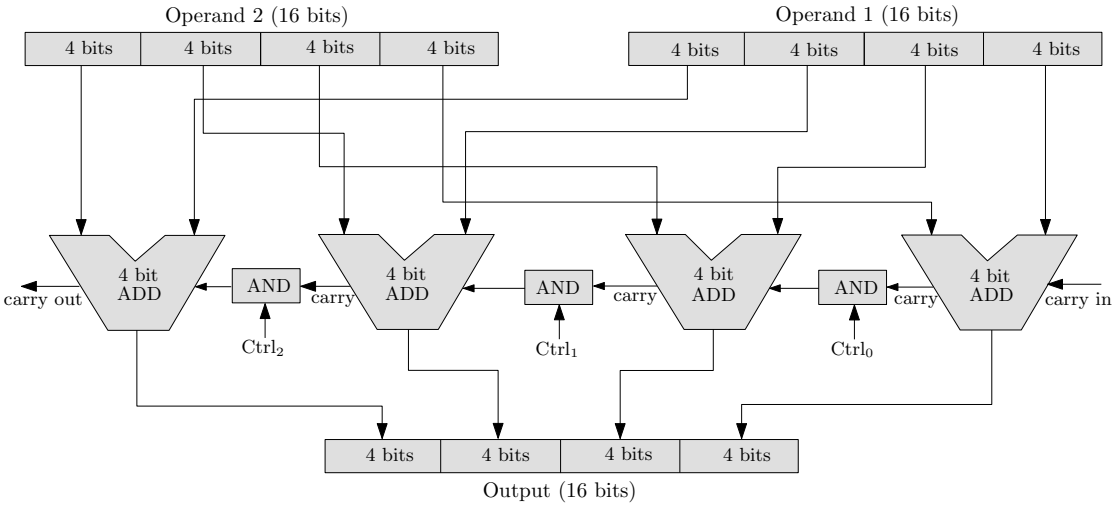

FIGURE 2.2: SWP Enabled Adder Architecture

adder is able to perform four 4-bit additions simultaneously. According to the requirement any other combinations of adder sizes can be obtained. The blocking or unblocking of carries at various bit location is performed using subword control signals (SWP_{ctrl}). These external subword control signals are used to generate the internal control bits ($Ctrl_0$, $Ctrl_1$ ...) which blocks the carry propagation at required bit locations. The internal control bit combinations used in the experiments for obtaining adders of different subword sizes are shown in the table 2.1.

Ctrl$_2$ (bit)	**Ctrl$_1$** (bit)	**Ctrl$_0$** (bit)	**Operations**
0	0	0	Four 4-bit Adders
1	0	1	Two 8-bit Adders
1	1	1	One 16-bit Adder

TABLE 2.1: Internal control bits for different subword sizes

The four 4-bit adders in figure 2.2 can be of any type such as ripple carry adder RCA, carry look ahead adder CLA, Group CLA etc. In RCA addition at any bit location waits for carry to be propagated from the previous bit location. In CLA carry inputs for all bit locations are calculated prior to the addition. In CLA carry logic gets complicated with the increase in word length. Therefore in order to reduce the complexity, CLA is implemented in groups of 4-bits [97]. The SWP adder using group CLA adders is shown in Figure 2.3.

As shown in Figure 2.3, group/block size of 4-bit is used in the implementation of *SWP 16-bit Group CLA* adder architecture. The boundaries of 4, 8, 16-bit subword sizes are synchronize with 4-bit block size. To reduce the carry propagation time, the carry for

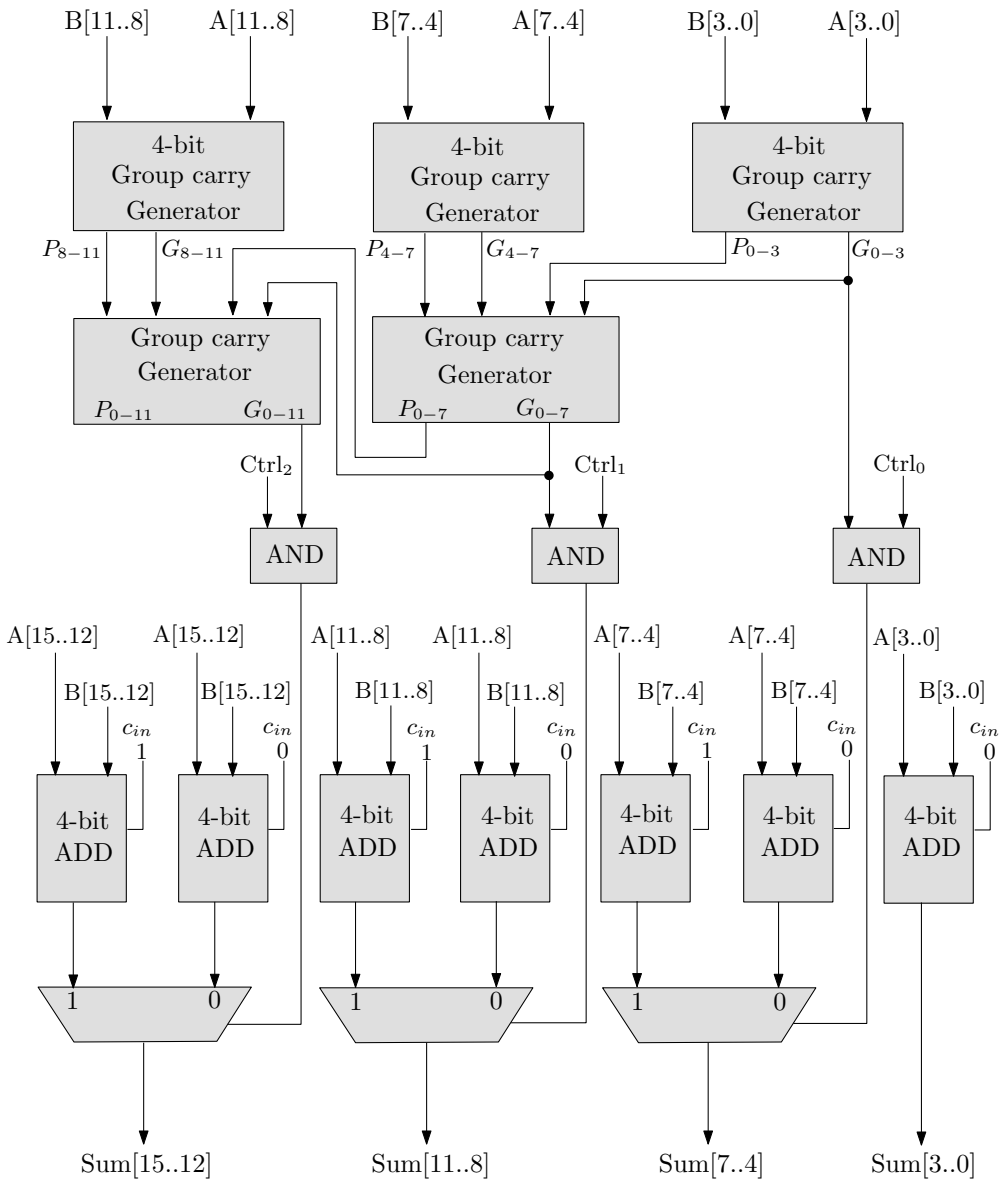

FIGURE 2.3: SWP adder architecture using group CLA

next block is generated using *4-bit group carry generator* units. These units generate the input carries for each addition block. The addition of blocks between subword boundaries are performed in parallel with carry generation logic. The *4-bit ADD* blocks generate the sum for both '0' and '1' input carries. Based on the outcome of corresponding carry generator unit and the selected subword size, the final sum of each block is selected using multiplexers. The select lines for the multiplexers are generated by ANDing the control signals ($Ctrl_0$, $Ctrl_1$, $Ctrl_2$) corresponding to the selected subword sizes with the output of *4-bit group carry generator* units. Due to the use of CLA carry generation logic, the addition blocks do not wait for the propagated carry. The speed of *SWP 16-bit Group CLA* adder is more than *SWP 16-bit RCA* due to the reduction in carry rippling effect.

The complexity of *SWP 16-bit Group CLA* is less than *SWP 16-bit CLA* because of the implementation of carry generation logic in groups of 4 bits rather than whole word.

In order to analyze the area and speed, simple and SWP enabled adders are synthesized to ASIC standard cell HCMOS9GP 130nm (CORE9GPLL 4.1 low leakage standard cell library from ST Microelectronics) and 90nm (fsd0t-a standard performance low voltage threshold cell library from UMC) technology using Synopsys [87] Design Vision and to FPGA (Xilinx Virtex II) using Mentor Graphics [34] Precision RTL tool. Table 2.2 shows the comparison of simple and classical SWP enable adder while using RCA, CLA and group CLA types of adders.

	90nm CMOS ASIC			**130nm CMOS ASIC**			**FPGA VirtexII**		
	Nand Gates	**CP** (ns)	**Gates X CP** (norm)	**Nand Gates**	**CP** (ns)	**Gates X CP** (norm)	**CLBs**	**CP** (ns)	**CLBs X CP** (norm)
	16-bit Ripple Carry Adder (RCA)								
Simple	118	1.07	1	114	3.20	1	34	12.9	1
SWP	**123**	1.05	**1.02**	**122**	3.05	1.01	30	13.8	0.94
Overhead	4 %	-2 %	2 %	7 %	-5 %	2 %	-12 %	7 %	6 %
	16-bit Carry Look Ahead (CLA) Adder								
Simple	168	0.65	0.87	175	1.41	0.68	33	12.1	0.91
SWP	218	0.87	1.51	238	**1.54**	1.01	44	8.85	0.89
Overhead	30 %	34 %	74 %	36 %	9 %	49 %	33 %	-27 %	-3 %
	16-bit Group Carry Look Ahead (CLA) adder								
Simple	153	0.92	1.12	149	1.87	0.76	32	8.82	0.64
SWP	165	**0.86**	1.13	165	2.07	**0.94**	**29**	**8.75**	**0.58**
Overhead	8 %	-7 %	1 %	11 %	11 %	23 %	-9 %	-1 %	-10 %

TABLE 2.2: Synthesis of classical SWP adders

The SWP adder in these experiments is able to perform either four 4-bit additions or two 8-bit additions or one 16-bit addition simultaneously (Table 2.1). On the other hand simple adder is able to perform only one 16-bit addition. This adder can be used for the addition of signed as well as unsigned numbers. Area (in terms of 2 input NAND gates or CLBs) and critical path (CP) for each type of implementation are shown. In SWP designs, the CP is the longest path between input and output irrespective of the selected subword size. Usually this path is obtained when performing the operation on subwords of highest available size (in this case 16-bit). Positive overhead indicates the additional resources (area or CP) required in terms of percentage of simple design resources to implement SWP capability. Negative overhead indicates that SWP designs require less area or CP. SWP designs with smallest area cost or smallest CP are shown in bold.

On ASIC platform *SWP 16-bit RCA* gives good results in terms of area. However from CP point of view, *SWP 16-bit Group CLA* and *SWP 16-bit CLA* gives good results on 90nm and 130nm technology respectively. On FPGA platform *SWP 16-bit Group CLA* shows good results in term of both area and CP. In order to find the overall efficiency of each design, the product of total NAND gates (or CLBs) consumed and CP is also calculated. This product is normalized based on *16-bit simple RCA* results. A smaller value of this product term indicates a high efficiency of the design and vice versa. The SWP design with the highest efficiency on each target technology is shown in bold. Efficiency of the *SWP enabled 16-bit Group CLA adder* shows that it gives good results on 130nm ASIC as well as on FPGA. In fact group CLA reduces the carry calculation logic by considering the groups of 4-bits only compared to conventional CLA one. On 90nm ASIC, *SWP enabled RCA* gives the best efficiency but due to the inherent carry rippling characteristics of RCA its CP is high compared to other SWP designs.

2.2.2 Multimedia SWP adder

Instead of classical subword sizes multimedia oriented SWP adder supports different subword sizes that are related to multimedia applications. Word and subword sizes which are supported by multimedia SWP operator are shown in Figure 2.4.

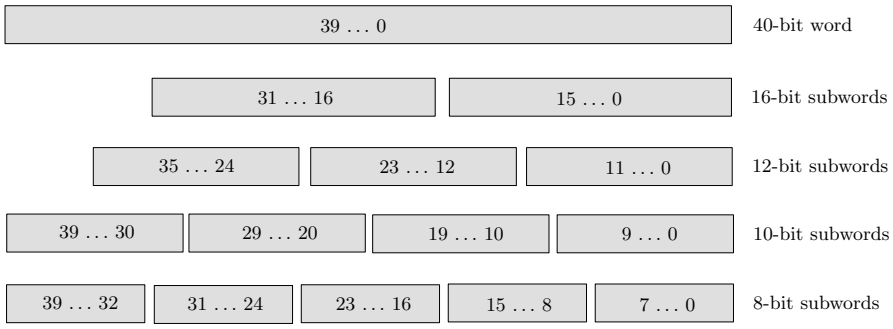

FIGURE 2.4: Word and subword sizes supported by multimedia SWP adder

Multimedia SWP operator supports 8, 10, 12 and 16-bit subword sizes. Based upon the available pixel sizes, the subword size is selected with the help of SWP control signals. Word size of 40-bit is chosen because it gives good efficiency/complexity trade off and ensures better resource utilization with different multimedia oriented pixel sizes. Simple 40-bit and SWP enable adders are implemented using the same methodology explain in previous subsection. This SWP adder can be used in any processor that has to deal with multimedia applications. Simple 40-bit adder can add two 40-bit signed/unsigned numbers. SWP enable 40-bit adder can be used to perform either five 8-bit additions or four 10-bit additions or three 12-bit additions or two 16-bit additions or one 40-bit addition of signed/unsigned numbers.

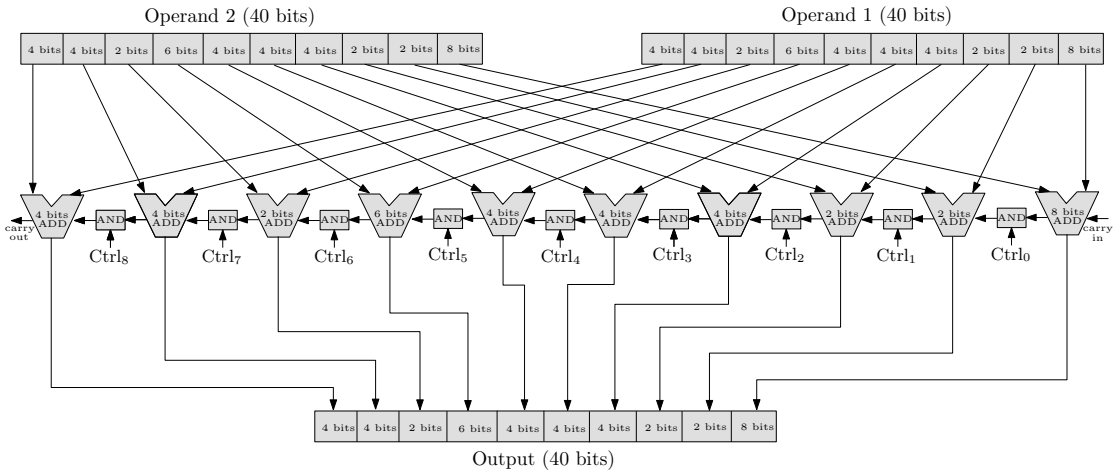

FIGURE 2.5: SWP multimedia adder architecture

Figure 2.5 shows the architecture of SWP multimedia adder architecture. Based on the requirements, the adder blocks between subwords control logics can be of any type. Corresponding to each selected subword size, the internal control bits $Ctrl_0$ to $Ctrl_8$ either block or unblock the carry transmission at various bit locations. The bit values of these control signals corresponding to each selected subword size are shown in Table 2.3.

$Ctrl_8$ (bit)	**$Ctrl_7$** (bit)	**$Ctrl_6$** (bit)	**$Ctrl_5$** (bit)	**$Ctrl_4$** (bit)	**$Ctrl_3$** (bit)	**$Ctrl_2$** (bit)	**$Ctrl_1$** (bit)	**$Ctrl_0$** (bit)	**Operations**
1	0	1	0	1	0	1	1	0	Five 8-bit Adders
1	1	0	1	0	1	1	0	1	Four 10-bit Adders
0	1	1	0	1	1	0	1	1	Three 12-bit Adders
1	0	1	1	1	0	1	1	1	Two 16-bit Adders
0	1	1	1	1	1	1	1	1	One 40-bit Adder

TABLE 2.3: Internal control bits combinations for SWP multimedia adder

Based on the results of SWP 16-bit adders (section 2.2.1), RCA and Group CLA algorithms are used in the implementation of multimedia oriented SWP adder. *SWP 40-bit RCA* and *SWP 40-bit group CLA* can be optimized for area and speed respectively. The architecture of *SWP 40-bit group CLA* is shown in Figure 2.6.

As shown in Figure 2.6, the input carries for addition blocks are generated by *group carry generator* units. The multimedia oriented subword sizes (8, 10, 12 and 16-bit) are not exact multiple of word size (40-bit). Therefore the complexity of SWP multimedia operator is high compared to SWP classical operator. Instead of using constant 4-bit carry generators, the bit width of carry generation units are synchronize with the addition block sizes between the subword boundaries. For 8-bit subword size, the subword boundaries lies at bit location 7, 15, 23, 31 and 39. Similarly for other subword sizes

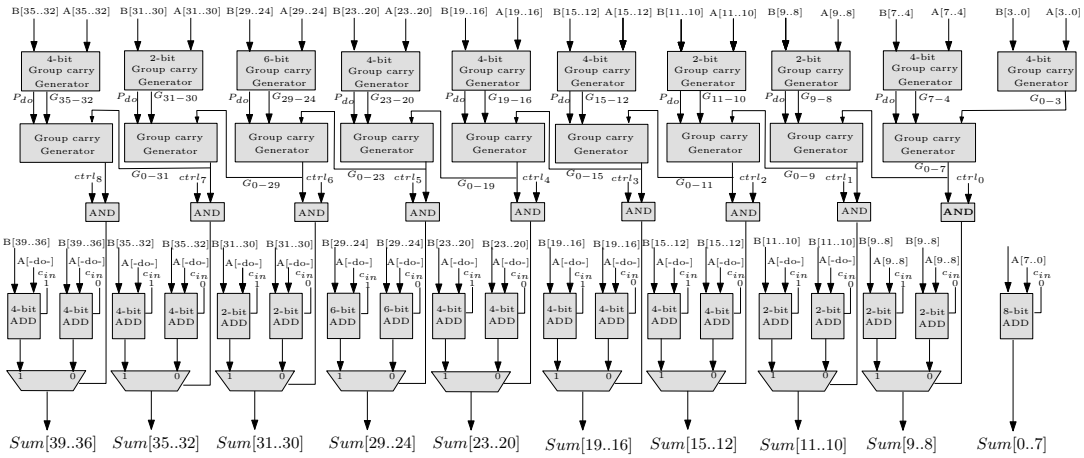

FIGURE 2.6: swp multimedia adder using group CLA

the subword boundaries lies at corresponding bit locations. Due to this synchronization, the size of carry generation blocks ranges between 2 to 6 bits. This variable group size reduces the complexity of overall *SWP 40-bit group CLA* adder.

	90nm CMOS ASIC			**130nm CMOS ASIC**			**FPGA VirtexII**		
	Nand Gates	**CP** (ns)	**Gates X CP** (norm)	**Nand Gates**	**CP** (ns)	**Gates X CP** (norm)	**CLBs**	**CP** (ns)	**CLBs X CP** (norm)
40-bit Ripple Carry Adder (RCA)									
Simple	291	2.69	1	281	8.10	1	82	25.2	1
SWP	**345**	4.31	1.90	**347**	10.1	1.54	**78**	27.6	1.04
Overhead	19 %	60 %	90 %	23 %	25 %	54 %	-5 %	10 %	4 %
40-bit Group Carry Look Ahead (CLA) adder									
Simple	372	2.31	1.10	372	4.11	0.67	74	15.7	0.56
SWP	444	**1.92**	**1.09**	463	**4.52**	**0.92**	81	**15.2**	**0.60**
Overhead	19 %	-17 %	-1 %	24 %	10 %	37 %	9 %	-3 %	7 %

TABLE 2.4: Results of multimedia SWP adders

Table 2.4 shows the area and CP results of SWP multimedia ADD operator. Compared to the 16-bit adder in subsection 2.2.1, the area and CP overheads are high. The reason for this increase is that the *SWP 40-bit adder* can support more subwords sizes (8, 10, 12, 16-bit) of multimedia nature compared to the *SWP 16-bit adder* which supports less number of classical subword sizes (4 and 8). Again the *Group CLA SWP adder* gives higher efficiency on all target technologies compared to the *SWP RCA adder*. Gate count of the *SWP RCA* is less but high CP reduces its overall efficiency.

2.3 Multiply operator

Multiplication is the mathematical operation of scaling one number by another. It is one of the basic operations in elementary arithmetics. Binary multiplier operator is used to multiply two input binary numbers. The input binary numbers are called multiplicand and multiplier and they can be signed or unsigned numbers. The main steps in the multiplication process are generation of partial products (PPs) and addition of PPs [37, 54]. There are several methods for the generation of PPs but using array of AND gates and Booth recoding are most famous methods.

Array of AND gates for PPs generation

The simplest way of generating PPs is by ANDing each bit of the multiplicand with all the bits of the multiplier. ANDing with '1' generate the same multiplier bit and ANDing with '0' bit generate '0' PP bit. In this method numbers of partial products are equal to the number of bits in multiplier. The generation of PPs using array of AND gates is shown in Figure 2.7.

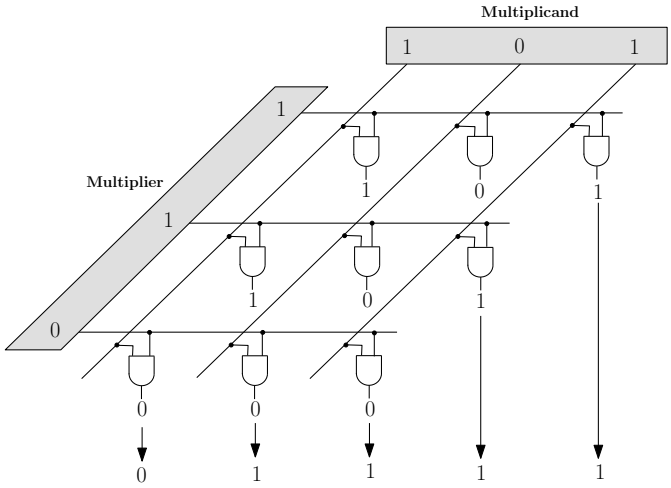

FIGURE 2.7: PPs generation using array of AND gates

As shown in Figure 2.7, the complexity of this method is very less. For the multiplication of signed 2's complement numbers, the 2's complement of last partial product needs to be taken before the addition of PPs. Moreover the PP vectors are either sign extended (signed multiplication) or zero padded (unsigned multiplication) before the addition. To simplify the procedure of sign extension or zero padding, the most significant bit (MSB) of each partial product is inverted and binary 1 is added at each MSB location. These binary 1s can be combined in one correction vector which is added to partial products.

Booth recoding for PPs generation In Booth recoding instead of considering single bit of multiplier, two bits are considered each time for the generation of PP which reduces the number of PPs to half. This algorithm uses string properties for the recoding of multiplier bit vector. From left to right each pair is observed for string property along with the higher order bit of the previous pair. Based on the bit combinations, each pair is recoded to either -2, -1, 0, 1 or 2. For least significant pair, the initial bit is considered as '0'. The recoding of bit pairs is shown in Figure 2.8.

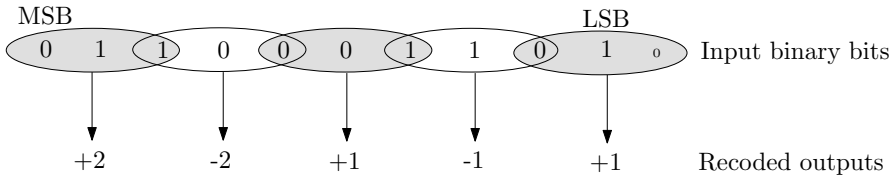

FIGURE 2.8: Booth recoding of multiplier bits

Based on recoded output values, PPs are generated either by simple copying of multiplicand bits (when recoded output is +1) or by left shifting of multiplicand bits (when recoded output is +2) or by taking 2's complement of multiplicand bits (when recoded output is -1) or by left shifting after taking 2's complement of multiplicand bits (when recoded output is -2). Due to the recoding process the number of PPs are reduced to half. The Booth recoding process and corresponding PP generation is shown in Table 2.5.

$\mathbf{X}_{2i+1}$	$\mathbf{X}_{2i}$	$\mathbf{X}_{2i-1}$	**Recoded**	**Generated partial product**
0	0	0	0	Copy all zeros
0	0	1	+1	Copy multiplicand bits
0	1	0	+1	Copy multiplicand bits
0	1	1	+2	Shift left by one
1	0	0	-2	2ś complement and then shift left
1	0	1	-1	2ś complement and then copy
1	1	0	-1	2ś complement and then copy
1	1	1	0	Copy all 0's

TABLE 2.5: PP generation using Booth recoding

After the generation of PPs by any of the above methods, they are added using adder trees. Efficient addition of PPs increases the overall performance of multiplication operation. This process is shown in Figure 2.9.

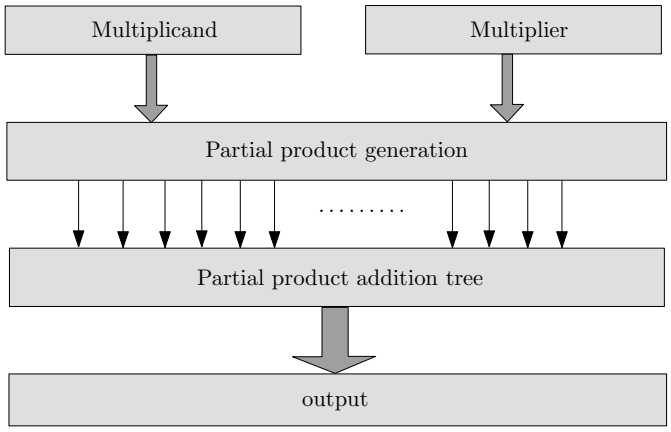

FIGURE 2.9: Block diagram of vector multiplication

Simple multiplier performs the multiplication of two input numbers where as SWP multiplier performs the parallel multiplication of subwords packed in word size input registers. The complexity of SWP multiplier depends upon the number and sizes of subwords.

2.3.1 Classical SWP multiplier

A SWP enabled multiplier can perform a simultaneous multiplication of subwords packed in a word. Figure 2.10 shows the example of a classical SWP enabled multiplication in which the 16-bit inputs are each portioned into four 4-bit subwords. The four pairs of subwords are multiplied to produce four 8-bit product subwords, which can be concatenated to produce a 32-bit result.

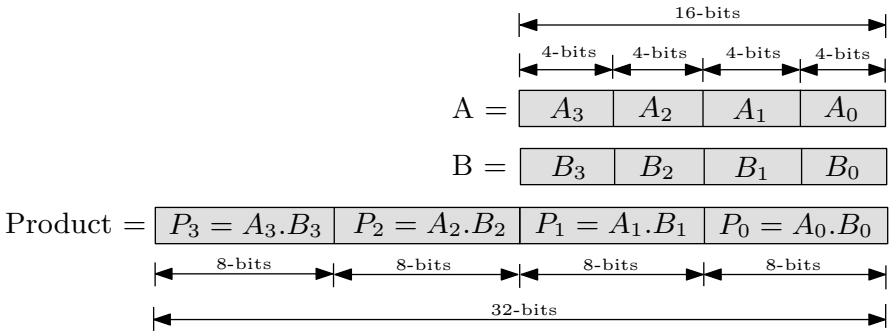

FIGURE 2.10: SWP multiplication

In SWP multiplier the computations are performed while taking into account different size subwords. The PPs in SWP multiplier can be generated either by array of AND gates or Booth recoding method. The PPs are generated corresponding to selected subword size. The PP bits generated for one subword size may not be valid for other subword size.

Due to the different arrangement of subwords, the simple method of array of AND gates for the generation of PPs is efficient compared to the other more complicated methods.

After generation of PPs, inversion of MSBs and computations of correction vectors corresponding to selected subword size are calculated based on signed or unsigned multiplication. In SWP multiplier as there are several parallel multiplications therefore the correction vectors are computed separately for each multiplication block. The generated PPs are added either by using compression trees and final carry propagate adder or we can allow the synthesis tool to implement any optimized adder. However in SWP multipliers care must be taken to isolate the product of subword blocks from each other.

	90nm CMOS ASIC			**130nm CMOS ASIC**			**FPGA VirtexII**		
	Nand Gates	**CP** (ns)	**Gates X CP** (norm)	**Nand Gates**	**CP** (ns)	**Gates X CP** (norm)	**CLBs**	**CP** (ns)	**CLBs X CP** (norm)
16-bit Multiplier									
Simple	2500	2.46	1	1801	6.21	1	188	16.0	1
SWP	4400	**2.63**	1.88	3040	**6.28**	1.71	381	**16.5**	2.09
Overhead	76 %	7 %	88 %	69 %	1 %	71 %	103 %	3 %	109 %
16-bit Booth Multiplier									
Simple	1748	3.81	1.08	1540	8.46	1.16	195	21.7	1.41
SWP	2950	4.03	1.93	2363	8.26	1.75	296	22.7	2.23
Overhead	69 %	6 %	79 %	53 %	-2 %	50 %	52 %	5 %	59 %
16-bit Schulte Multiplier									
SWP	**2485**	2.83	**1.14**	**1751**	6.88	**1.08**	**220**	16.7	**1.22**
Overhead @ Mult	-1 %	15 %	14 %	-3 %	11 %	8 %	17 %	4 %	22 %

TABLE 2.6: Results of classical SWP multiplier

Table 2.6 shows the area and CP results of simple and classical SWP enabled 16-bit multipliers. The product term Gates x CP is normalized based on the *simple 16-bit Multiplier* results. In these experiments SWP enable multipliers can perform either four 4-bit multiplications or two eight bit multiplications or one 16-bit multiplication simultaneously. Simple multiplier can perform only one 16-bit multiplication. Both *16-bit Multiplier* and *16-bit Schulte Multiplier* can be used for the multiplication of signed as well as unsigned numbers. *16-bit Booth Multiplier* can be used for the multiplication of signed numbers only. In *16-bit Multiplier* PPs are generated by simple AND operations of multiplier and multiplicand bits. In *16-bit Booth Multiplier*, PPs are generated by using booth recoding scheme. In both *16-bit Multiplier* and *16-bit Booth Multiplier*, SWP capability is incorporated by implementing the multiplier in blocks of subword size and finally concatenating the output of each block. Logic is shared between blocks during

logic synthesis. The *16-bit Schulte Multiplier* is efficient implementations of SWP enable multiplier for multimedia applications and was proposed in [56]. Schulte multiplier also generates PPs by simple AND operation but it does not require detection and suppression of carries across subword boundaries (for more details on Schulte multiplier see [56]). These three multiplier architectures have been chosen for their good matching with SWP technique. In all the multipliers, instead of implementing compression trees the synthesis tool is allowed to do the optimized addition of PPs. Results show that the area overhead for implementing SWP capability in multipliers is more than 50% except for SWP Schulte multiplier which gives similar area compared to simple version of *16-bit Multiplier*. In fact Schulte multiplier is dedicated to SWP and takes benefits from conveniently compatible subword sizes. For word length of 16-bit, it is designed to perform parallel multiplications on four 4 x 4, two 8 x 8 and one 16 x 16. Due to high compatibility of subword sizes (4 and 8-bit) with word size (16-bit), it is convenient to obtained and arrange PPs corresponding to each subword size [56]. On both ASIC and FPGA platforms, SWP Schulte multiplier gives good results in terms of area. For all the SWP designs, increase in CP is small compared to their simple versions. On both ASIC and FPGA platforms, *SWP 16-bit Multiplier* gives good results in terms of CP but CP of *Schulte multiplier* is also not too far from it. Efficiency of *Schulte multiplier* compared to other multipliers indicates that it is the most efficient classical SWP multiplier design.

2.3.2 Multimedia SWP multiplier

Multimedia SWP multiplier supports the subword sizes which are in coordination with pixel sizes in multimedia applications. Simple and a SWP enabled version of a 40-bit multiplier are implemented which are able to perform signed as well as unsigned multiplications. SWP multimedia multiplier can perform either five 8-bit multiplications or four 10-bit multiplications or three 12-bit multiplications or two 16-bit multiplications or obviously one 40-bit multiplication. In the simple version, PPs are generated by ANDing of multiplicand bits with each multiplier bit. In the SWP version PPs are generated by ANDing the multiplier bits with only those bits of multiplicand which corresponds to the selected subword size. Basically the PPs are generated like the *16-bit Schulte multiplier* explained previously. However in the *16-bit Schulte multiplier* the selected subword sizes (4 and 8) are conveniently compatible with the word size (16-bit). In the *SWP 40-bit multiplier*, the increased numbers of multimedia oriented subword sizes (8, 10, 12 and 16) are not so easily compatible with the operator word size (40-bit). Due to different arrangements of PPs matrix, usually the PPs bits generated for the multiplication of one subword size data is not valid for other subword size data. For instance the arrangement of PPs bits for the subword size of 8-bit are different from the

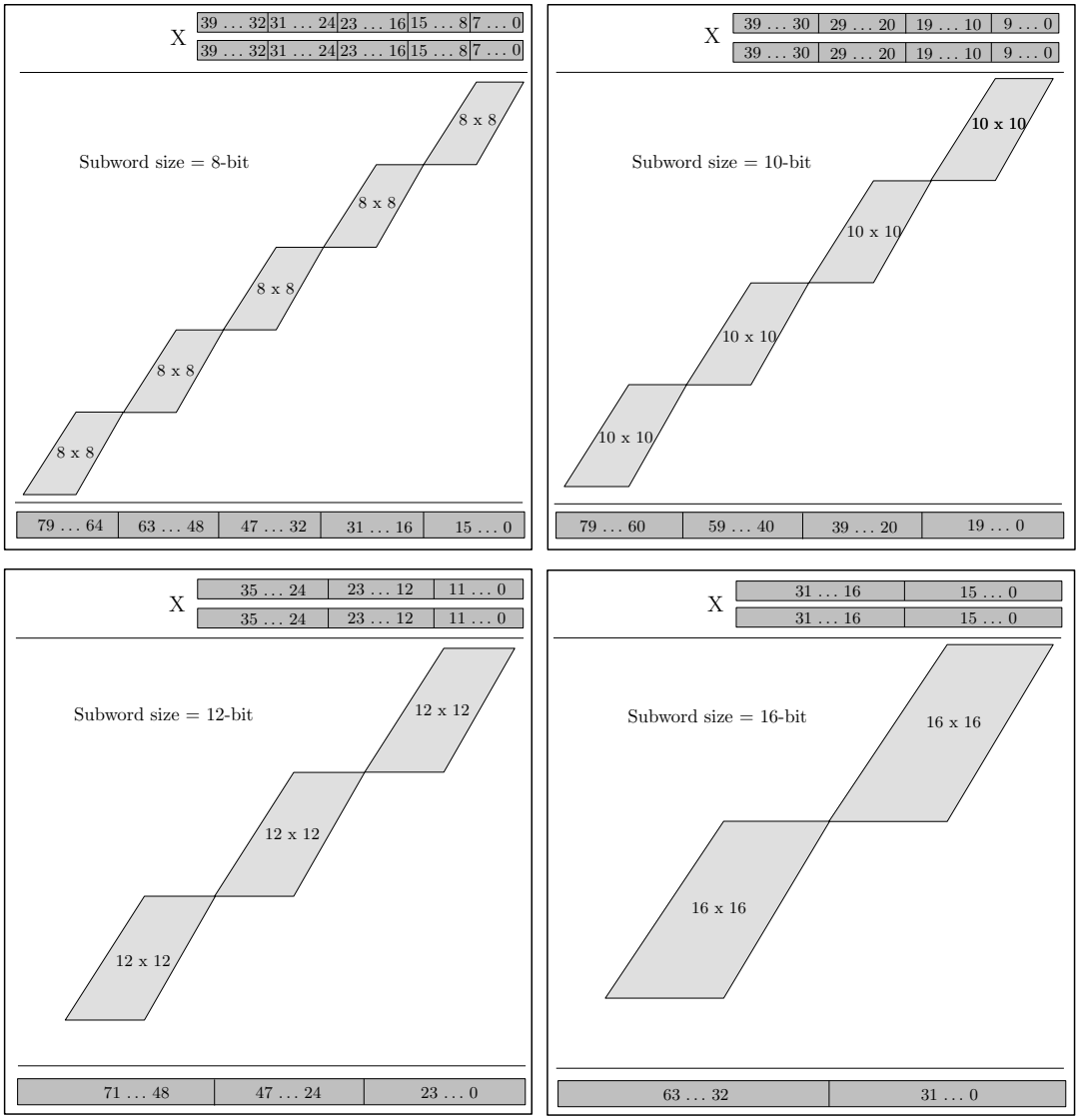

FIGURE 2.11: Arrangement of partial products for different subword sizes

arrangement of PPs bit for subword size of 10-bits. In SWP multimedia multiplier, PPs are generated for each bit of multiplier vector. However for each selected subword size only those PP bits are generated which are required. The portion of PPs corresponding to each selected subword sizes are shown in Figure 2.11.

As shown in Figure 2.11, the PPs corresponding to different selection of subword size are different. The unused PP bits are 0. For instance in case of 8-bit subword size, five (8x8) partial product arrays are used. Similarly for other subword sizes the corresponding size PP arrays are used.

2.3.2.1 Dedicated PP generation units

In SWP multimedia multiplier, the simplest way to generate PPs is to generate PPs vector separately for each subword size. By using this approach, the dedicated hardware units are required for the generation of PPs for each subword size. Based upon the selected subword size, the PPs output of corresponding unit is used. This process is shown in Figure 2.12.

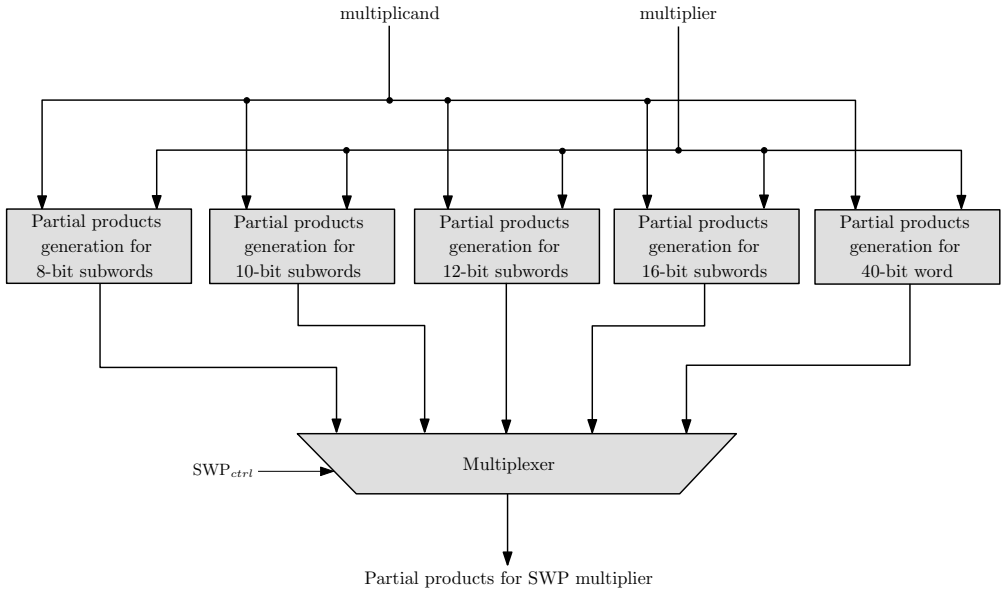

FIGURE 2.12: SWP partial product generation using dedicated units

The drawback of using the architecture shown in Figure 2.12 is that large amount of hardware resources are required. The reason being that dedicated PP generation units are used for each subword size. The number of these units will be equal to the number of subword sizes supported by the SWP multiplier. To overcome this draw back of excessive hardware resource requirements, SWP multiplier is designed in such a way that same PP generation hardware can be used for different selection of subword size. By using this approach, the hardware used for the generation of PPs for 8-bit subword size is also used for the generation of PPs for 10, 12, 16 or 40-bit subword sizes. For this purpose generalize partial product generation unit is required.

2.3.2.2 Generalize PP generation unit for SWP multimedia multiplier

Generalize PP generation unit generate partial products for different selection of subword sizes. Based upon the selected subword size the input multiplicand vector is updated before the generation of partial product. This updated multiplicand vector is then used instead of normal multiplicand vector for the generation of PPs. In this way the PP

generation hardware remains same for different selection of subword sizes. For instance in case of 8-bit subword size, the first eight PPs ($PP_0 \ldots PP_7$) requires multiplicand vector bits ranging from 0 to 7. Therefore for the generation of these PPs, the update in multiplicand vector is done in such a ways that multiplicand bits $(40 \ldots 8)$ are made 0. Similarly for the next eight PPs ($PP_8 \ldots PP_{15}$) only multiplicand bits (8 to 15) are required. Therefore the updated vector contains only these bits and rest of the bits are made 0. In the same way the multiplicand vector is updated for other PPs also. After this updating, the PPs are generated by simple AND operation irrespective of selected subword size. Using this method same PP generation hardware is used for different selection of subword sizes. This result in the reduction of hardware resources of SWP multimedia multiplier.

In our implementation of SWP multimedia multiplier, the generated PPs bits remain valid for each selected subword size multiplication. For this purpose the PPs matrix is divided into two parts. First part contains those bits of PPs which remains same irrespective of different subword size selection and are generated by direct ANDing of multiplier bits with multiplicand bits. Second part contains those PPs bits whose values changes for different selection of subword sizes and are generated indirectly by ANDing multiplier bits with modified multiplicand vector. Modified multiplicand vector changes its bit values corresponding to each selected subword size and hence the PPs bits updates automatically for each selected subword size.

Let us consider the generation of PP_0 which is the partial product corresponding to bit location 0 of multiplier vector. Bits of PP_0 which remains valid for different selection of subword sizes are shown by filled boxes in Figure 2.13. Unfilled boxes represent '0's.

Bit Location	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Subword size 8-bit																																									
Subword size 10-bit																																								
Subword size 12-bit																																								
Subword size 16-bit																																								
40-bit multiplication																																								

FIGURE 2.13: Valid bits of PP0 for different subword sizes

As shown in the Figure 2.13, $PP_0(7 \ldots 0)$ bits remains same irrespective of subword size selection. Therefore these PP bits are generated directly by ANDing multiplicand $(7 \ldots 0)$ bits with the 0th bit of multiplier vector.

$PP_0\ (7 \ldots 0) = \text{multiplicand}(7 \ldots 0)\ .\ \text{multiplier}(0);$

However the rest of the bits $PP_0(39 \ldots 8)$ changes their values corresponding to different subword size selection. These variable bits of PP_0 are generated indirectly by four modified multiplicand vectors (mod_vec_8, mod_vec_8_10, mod_vec_8_10_12 and mod_vec_8_10_12_16). Modified vectors bit values changes corresponding to each selected subword size. SW_8, SW_{10}, SW_{12} and SW_{16} are each one bit SWP control signals. The control signal corresponding to selected subword size will be '1' and the rest are '0'. For 8-bit selected subword size, SW_8 control signal will be '1' and SW_{10}, SW_{12}, SW_{16} are '0'. To illustrate the the generation of PPs, let us consider the generation of bits for partial product PP_0 using the modified multiplicand vectors. For this purpose first of all the bit values of four modified multiplicand vectors are obtained using multiplicand vector and SWP control signals as given below.

mod_vec_8 = multiplicand . $\overline{(SW_8)}$;
mod_vec_8_10 = multiplicand . $\overline{(SW_8 \quad + \quad SW_{10})}$;
mod_vec_8_10_12 = multiplicand . $\overline{(SW_8 \quad + \quad SW_{10} \quad + \quad SW_{12})}$;
mod_vec_8_10_12_16 = multiplicand . $\overline{(SW_8 \quad + \quad SW_{10} \quad + \quad SW_{12} \quad + \quad SW_{16})}$;

For different selection of subword sizes, the values of these modified multiplicand vectors are different. The bit values of modified multiplicand vectors for different selection of subword sizes are given below.

When selected subword size = 8-bit :

$SW_8 = 1; \quad SW_{10} = 0; \quad SW_{12} = 0; \quad SW_{16} = 0;$
mod_vec_8 $(9 \ldots 8) = 0$;
mod_vec_8_10$(11 \ldots 10) = 0$;
mod_vec_8_10_12 $(15 \ldots 12) = 0$;
mod_vec_8_10_12_16 $(39 \ldots 16) = 0$;

When selected subword size = 10-bit :

$SW_8 = 0; \quad SW_{10} = 1; \quad SW_{12} = 0; \quad SW_{16} = 0;$
mod_vec_8 $(9 \ldots 8)$ = multiplicand $(9 \ldots 8)$;
mod_vec_8_10$(11 \ldots 10) = 0$;
mod_vec_8_10_12 $(15 \ldots 12) = 0$;
mod_vec_8_10_12_16 $(39 \ldots 16) = 0$;

When selected subword size = 12-bit :

$SW_8 = 0$; $SW_{10} = 0$; $SW_{12} = 1$; $SW_{16} = 0$;
mod_vec_8 (9 ... 8) = multiplicand (9 ... 8);
mod_vec_8_10(11 ... 10) = multiplicand (11 ... 10);
mod_vec_8_10_12 (15 ... 12) = 0;
mod_vec_8_10_12_16 (39 ... 16) = 0;

When selected subword size = 16-bit :

$SW_8 = 0$; $SW_{10} = 0$; $SW_{12} = 0$; $SW_{16} = 1$;
mod_vec_8 (9 ... 8) = multiplicand (9 ... 8);
mod_vec_8_10(11 ... 10) = multiplicand (11 ... 10);
mod_vec_8_10_12 (15 ... 12) = multiplicand (15 ... 12);
mod_vec_8_10_12_16 (39 ... 16) = 0;

For word size multiplication :

$SW_8 = 0$; $SW_{10} = 0$; $SW_{12} = 0$; $SW_{16} = 0$;
mod_vec_8 (9 ... 8) = multiplicand (9 ... 8);
mod_vec_8_10(11 ... 10) = multiplicand (11 ... 10);
mod_vec_8-10_12 (15 ... 12) = multiplicand (15 ... 12);
mod_vec_8_10_12_16 (39 ... 16) = multiplicand (39 ... 16);

By using modified multiplicand vectors, the generation of partial products is automatically adopted for selected subword size. Therefore at this point, PPs are generated irrespective of selected subword sizes. Using modified multiplicand vectors, bits of PP_0 for different subword sizes are generated using following equations.

PP_0 (7 ... 0) = multiplicand(7 ... 0) . multiplier(0);
PP_0 (9 ... 8) = mod_vec_8 (9 ... 8) . multiplier(0);
PP_0 (11 ... 10) = mod_vec_8_10 (11 ... 10) . multiplier(0);
PP_0 (15 ... 12) = mod_vec_8_10_12 (15 ... 12) . multiplier(0);
PP_0 (39 ... 16) = mod_vec_8_10_12_16(39 ... 16) . multiplier(0);

Similarly for the generation of other PPs, modified multiplicand vectors are used accordingly. Therefore PPs bits are generated for different subword sizes using the same PPs generation hardware. Bit inversions and addition of correction vectors are done based upon the selected multiplication type (signed/unsigned) and the subword size. The arrangement of PPs for an unsigned multiplication with a subword size of 8-bits is shown in figure 2.14.

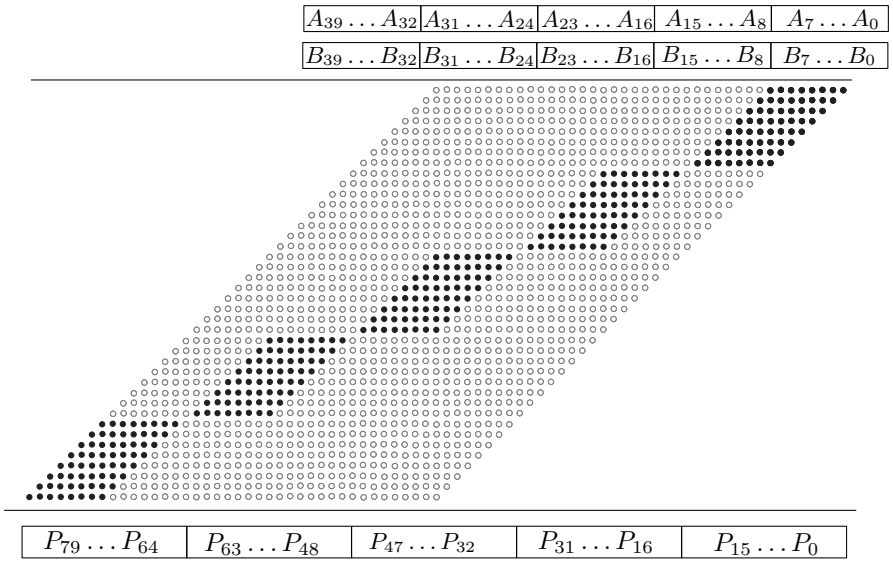

FIGURE 2.14: Arrangement of PPs for 8-bit data size

Unfilled circles represent unused (0's) PP bits whereas filled circles represent used PP bits for a subword size of 8-bits. Each block of filled circles represents one 8x8 multiplication block. In the case of a signed multiplication, to avoid the sign extension, the MSBs of the filled portion of each PP are inverted and 1's are added at bit location 8, 24, 40, 56 and 72 as correction vector. As in a signed multiplication, the last PP is 2's complement so in order to obtain the 2's complement, bits of last PPs of each multiplication block are inverted and '1s' are added at LSB of each block. Addition of PPs and correction vector is done using optimized adders of the synthesis tool. PPs for other subword sizes (10, 12, 16 or 40) are arranged in a same fashion. However the location of sign extension inversion bits, correction vector bits and the PPs whose 2's complement needs to be taken are different for different subword sizes. In the case of a subword size of 12, the last four PPs are all filled with zeros (unused). Similarly, for a subword size of 16, the last eight PPs are all filled with zeros.

2.3.2.3 Addition of partial products for SWP multimedia multiplier

The generated PPs are added using adder trees. At each level of tree, the adders add the partial products in parallel. For each subword size the product subwords are represented by twice number of bits. For 8-bit subword size, each product subword consists of 16 bits. These 16 bits are enough to represent the product of two 8-bit subwords hence no overflow will occur. Similarly for other subword sizes, the product subwords have twice data lengths. For word size multiplication of 40-bits the product consists of 80 bits.

2.3.2.4 Comparison of simple and SWP multimedia multiplier

SWP multimedia multiplier performs several subword multiplications in parallel. On the other hand simple 40-bit multiplier can perform multiplication of two 40-bit numbers and generate 80-bit product. Therefore it is obvious that due to SWP controls, SWP multimedia multiplier will require more resources compared to simple multiplier. However by using the efficient schemes for partial product generation and addition, the overheads of SWP multipliers can be reduced.

	90nm CMOS ASIC			**130nm CMOS ASIC**			**FPGA VirtexII**		
	Nand Gates	**CP** (ns)	**Gates X CP** (norm)	**Nand Gates**	**CP** (ns)	**Gates X CP** (norm)	**CLBs**	**CP** (ns)	**CLBs X CP** (norm)
Simple	14518	6.07	1	10532	14.0	1	917	19.7	1
SWP	15099	7.38	1.26	11081	15.0	1.13	1505	21.4	1.78
Overhead	4 %	22 %	26 %	5 %	7 %	13 %	64 %	9 %	78 %

TABLE 2.7: Results of multimedia SWP Multiplier

Table 2.7 shows the area and CP results of simple and multimedia SWP enable 40-bit multipliers. Due to the efficient technique for the generation and arrangement of PPs corresponding to each subword size, the area and CP overheads are less. Area and CP overheads of *SWP multimedia 40-bit Multiplier* are little more than the area and CP overheads of the *16-bit classical Schulte Multiplier* (Table 2.6). However this small increase in overheads is of less importance when considering the fact that the SWP *40-bit multimedia Multiplier* can support five different multimedia oriented subword sizes (8, 10, 12, 16 and 40) compared to the *16-bit classical Schulte Multiplier* which can support only three different subword sizes (4, 8 and 16). SWP overhead in term of efficiency (gate x CP) shows very good results on ASIC platforms. For an ASIC implementation, the *SWP multimedia multiplier* requires only 5% more area compared to simple one. CP increase is 22% and 7% on 90nm and 130nm ASIC technologies respectively. In a FPGA implementation the resources are CLBs rather than standard cells [44, 74]. Therefore an area overhead of 64% on a FPGA platform represents the increase in CLBs rather than chip area precisely.

2.3.3 Analysis of SWP multipliers

Keeping in view the complexity of multipliers (compared to other basic arithmetic operators), SWP multimedia multipliers (signed/unsigned) with different configurations of word lengths (WL) and subword sizes are implemented in order to analyze the increase in

area, CP and power consumption for each configuration. For these experiments, WL and subword sizes are selected that pertains to multimedia applications. These SWP multipliers can perform both signed as well as unsigned multiplications and are designed in the same way as the 40-bit SWP multiplier explained in section 2.3.2. Different data length multiplication operations which can be performed by SWP multipliers corresponding to each configuration are shown in table 2.8.

Config. No.	**Word Length**	**Supported (Sub)Words size operations**
1	16	(Two 8 x 8) **or** (One 16 x 16)
2		(Two 8 x 8) **or** (One 10 x 10) **or** (One 16 x 16)
3		(Two 8 x 8) **or** (One 12 x 12) **or** (One 16 x 16)
4	24	(Three 8 x 8) **or** (Two 10 x 10) **or** (One 24 x 24)
5		(Three 8 x 8) **or** (Two 12 x 12) **or** (One 24 x 24)
6	30	(Three 8 x 8) **or** (Two 12 x 12) **or** (One 30 x 30)
7		(Three 10 x 10) **or** (Two 12 x 12) **or** (One 30 x 30)
8	32	(Four 8 x 8) **or** (Three 10 x 10) **or** (Two 12 x 12) **or** (One 32 x 32)
9		(Four 8 x 8) **or** (Three 10 x 10) **or** (Two 16 x 16) **or** (One 32 x 32)
10	36	(Four 8 x 8) **or** (Three 10 x 10) **or** (One 36 x 36)
11		(Four 8 x 8) **or** (Three 12 x12) **or** (One 36 x 36)
12		(Four 8 x 8) **or** (Three 10 x 10) **or** (Two 16 x 16) **or** (One 36 x 36)
13		(Four 8 x 8) **or** (Three 12 x 12) **or** (Two 16 x 16) **or** (One 36 x 36)
14	40	(Five 8 x 8) **or** (Four 10 x 10) **or** (Three 12 x 12) **or** (One 40 x 40)
15		(Five 8 x 8) **or** (Four 10 x 10) **or** (Three 12 x 12) **or** (Two 16 x 16) or (One 40 x 40)

TABLE 2.8: Configurations of word and subwords sizes

Each SWP multiplier can perform multiplication operation on WL as well as on subword length data. For instance the SWP multiplier shown in the configuration 2 can perform either two 8-bit multiplications or one 10-bit multiplication or one 16-bit multiplication. For each WL different number and sizes of subwords are selected for implementation. Within the group of SWP multipliers with same WL, the operations on subwords supported by one configuration can be supported by other configuration. For instance configuration 3 (two 8 x 8, one 12 x 12 and one 16 x 16) can also perform the operations on subword data lengths offered by configuration 2 (two 8 x 8, one 10 x 10 and one 16 x 16). However for this purpose the user has to expand the 10-bits inputs to 12-bits (sign extension for signed multiplication or zero padding for unsigned multiplication) and arrange the input subwords data in accordance with the configuration being used. On the contrary configuration 2 cannot perform operation on the subwords sizes offered by configuration 3 because a 12-bit multiplication cannot be performed on 10-bit operator. Same is the case for other configuration groups.

Conf. No.	Word (WL) (bits)	Supported Subword sizes (bits)	90nm CMOS ASIC				130nm CMOS ASIC				FPGA VirtexII		
			Nand Gates	CP (ns)	Gates X CP	Power (mW)	Nand Gates	CP (ns)	Gates X CP	Power (mW)	CLB	CP (ns)	CLBs X CP
1	**16**	**8**	2320	2.55	**5916**	4.9	1666	6.89	**11479**	8.55	173	16.2	**2802**
2		**8, 10**	2432	2.77	6737	4.2	1787	6.67	11919	7.18	211	16.6	3503
3		**8, 12**	2439	2.86	6976	4.5	1794	6.76	12127	7.72	203	16.5	3350
4	**24**	**8, 10**	5378	3.99	**21297**	9.9	3966	9.97	39541	15.4	499	17.4	8683
5		**8, 12**	5386	4.13	22244	10.1	3938	9.12	**35915**	16.4	478	17.3	**8269**
6	**30**	**8, 10**	8349	5.40	45085	15.5	6094	11.9	72519	22.3	795	18.9	15026
7		**10, 12**	8340	5.13	**42784**	16.5	6092	11.6	70667	23.6	779	18.9	**14723**
8	**32**	**8, 10, 12**	9651	5.72	55204	14.3	7095	13.7	97202	20.0	944	19.9	18786
9		**8, 10, 16**	9630	5.59	**53832**	15.6	7033	12.9	**90726**	21.8	925	19.9	**18408**
10	**36**	**8, 10**	12182	5.76	**70168**	20.5	8787	13.4	**117746**	30.9	1184	20.2	23917
11		**8, 12**	12156	6.30	76583	21.5	8728	14.3	124810	32.4	1151	20.1	**23135**
12		**8, 10, 16**	12256	6.17	75620	17.6	8884	13.9	123488	25.5	1190	20.2	24038
13		**8, 12, 16**	12211	6.17	75342	18.5	8812	14.0	123368	27.0	1171	20.1	23537
14	**40**	**8, 10, 12**	14918	6.84	**102039**	21.5	10930	16.2	177066	30.2	1492	20.5	**30586**
15		**8,10,12,16**	15099	7.38	111431	17.9	11082	15.0	**166230**	24.6	1505	21.4	32207

TABLE 2.9: Synthesis results for SWP multipliers

Table 2.9 shows the area, CP and the power consumption results of the SWP multipliers with table 2.8 configurations. Area, CP and power consumption are mainly dependent upon the WL. However within the same WL group, slight increase or decrease in area, CP and power consumption occurs due to the variation in the number and the size of the supported subwords. As the WL increases, the area, CP and power consumption on both ASIC and FPGA platform increases accordingly. Results show that area, CP and power consumption varies slightly for different configurations having the same WL but different subword sizes. The phenomenon can be seen in every group. Let us take the group with WL of 24-bits which consists of configuration 4 and 5. In this group, area increases slightly from 5378 to 5386 nand gates, CP also increases slightly from 3.96ns to 4.13ns and power consumption increases from 9.9 to 10.1mwatt on 90nm technology. The reason being that within most of the groups, the number of available multiplication operations remains same. For instance in the group with WL of 24-bits, configuration 4 can perform six operations (three 8-bit operations or two 10-bit operations or one 16-bit operation) and configuration 5 can also perform six operations (three 8-bit operations or two 12-bit operations or one 16-bit operation). However there are some groups in which different configurations supports different number of operations but even then area, CP and power consumption does not vary tremendously within the group. Furthermore within the same WL group, the area, the CP and the power consumption do not varies largely with the increase or decrease of number of subword sizes supported by SWP multiplier. This can be seen in configuration group with WL of 40-bit. In this group, configuration 14 supports three subword sizes (8, 10 and 12) and configuration 15 support

four subword sizes (8, 10, 12 and 16). However there is only a small difference in area and CP of these two configurations on all the target technologies. The main reason is that the SWP control logic is small compared to the multiplication's dedicated logic. Within each group, the configuration with higher efficiency (gate x CP) is shown in bold letters.

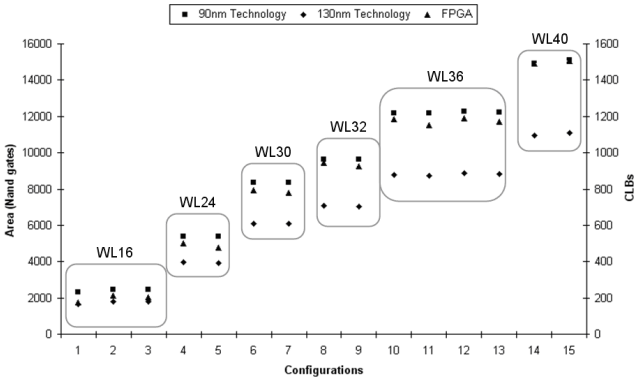

FIGURE 2.15: Area of SWP multipliers

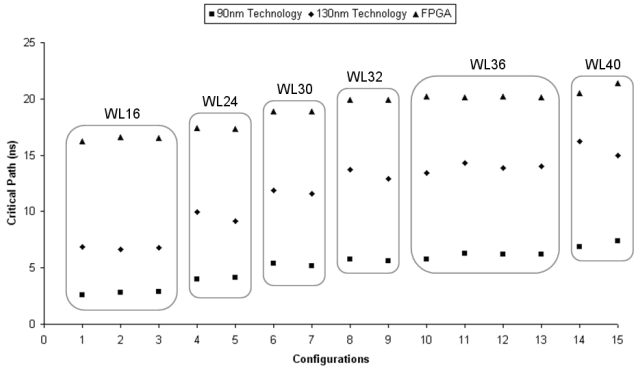

FIGURE 2.16: CP of SWP multipliers

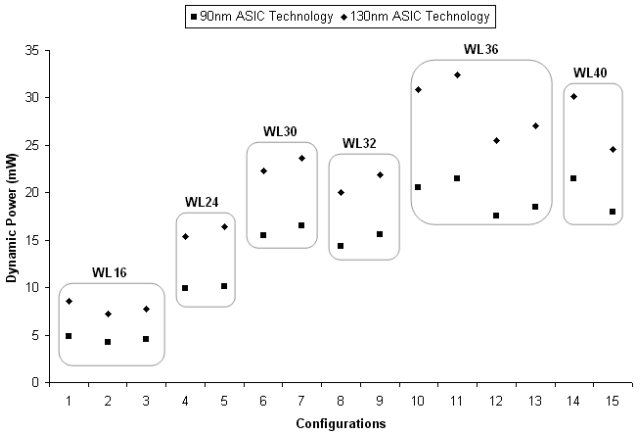

FIGURE 2.17: Power consumption of SWP MULTs

Figure 2.15 shows the increase in area of the SWP multipliers for the different configurations. Figure 2.16 shows the increase in CP for the same set of configurations. Figure 2.17 shows the increase in the power. As said before, graphs show that within each WL group, area, CP and power consumption change slightly with the variation in number or sizes of subwords selected. However moving from one WL group to next, area increases more rapidly compared to CP and power consumption as shown by the slope of the curves.

2.4 MAC operator

The multiply accumulate (MAC) unit represents one of the most frequently used block in multimedia and digital signal processing applications. The multiplier in a MAC is used to multiply two N-bit numbers and generate a 2N-bit result. The adder is used to add the result of the multiplier with a 2N-bit addend [2, 27, 64].

2.4.1 Classical SWP MAC

The generation of the PPs for classical SWP MACs can be done either by AND operations or booth recoding or with the Schulte algorithm. Instead of using one extra adder for the accumulation purpose, the addition of the PPs and addend are combined. Optimization of the adder is done by the synthesis tool. The SWP enable MAC in these experiments is capable of performing either four 4-bit multiplications and four 8-bit additions or two 8-bit multiplications and two 16 bit additions or one 16-bit multiplication and one 32 bit addition. On the other hand simple MAC is capable of performing one 16-bit multiplication and one 32-bit addition.

Table 2.10 shows the area and CP results. Product term (gates x CP) is normalized based on the *simple MAC* results. Both *MAC* and *MAC using Schulte Multiplier* are able to perform MAC operation on signed as well as unsigned numbers. *MAC using Booth Recoding* is able to perform MAC operation on signed numbers only. Multiplier in *MAC* generates PPs by AND operations. On an ASIC platform the area overhead for implementing SWP capability is more than 80% except for the *Schulte MAC* in which the area overhead is almost zero on both 90nm and 130nm technologies. On the contrary, CP overhead is less than 10% except for *Schulte MAC* in which CP overhead is 27% and 15% on 90nm and 130nm technologies respectively. On the FPGA platform the CLB overhead of the *Schulte MAC* is also less compared to the other implementations. On ASIC platforms, the classical *SWP Schulte MAC* gives good results in terms of area and the *SWP MAC* gives good results in terms of CP. On FPGA platform, the *SWP Schulte*

	90nm CMOS ASIC			130nm CMOS ASIC			FPGA VirtexII		
	Nand Gates	**CP** (ns)	**Gates X CP** (norm)	**Nand Gates**	**CP** (ns)	**Gates X CP** (norm)	**CLBs**	**CP** (ns)	**CLBs X CP** (norm)
MAC									
Simple	2733	2.51	1	1919	6.42	1	187	15.5	1
SWP	5261	**2.70**	2.07	3760	**6.84**	2.09	435	16.4	2.46
Overhead	92 %	8 %	107 %	96 %	7 %	109 %	133 %	6 %	146 %
MAC (Booth)									
Simple	2010	4.06	1.19	1718	8.81	1.23	211	23.5	1.71
SWP	3625	4.38	2.31	3090	8.68	2.18	353	24.5	2.98
Overhead	80 %	8 %	94 %	80 %	-1 %	77 %	67 %	4 %	74 %
MAC (Schulte)									
SWP	**2684**	3.19	**1.25**	**1927**	7.38	**1.15**	**218**	**16.0**	**1.2**
Overhead @ MAC	-2 %	27 %	25 %	1 %	15 %	15 %	17 %	3 %	20 %

TABLE 2.10: Results of classical SWP MAC

MAC gives good results in terms of both area and CP. Overall area and CP overhead of the MAC operation have been increased compared to the 16-bit multiplier explained in the section 2.3.1. This increase occurs because of the addition of the 2N-bit addend which also requires SWP capability. Efficiency shows that the 16-bit classical *Schulte based MAC* is still the best SWP MAC architecture for classical subword sizes.

2.4.2 Multimedia SWP MAC

Multimedia oriented 40-bit simple and SWP enable MAC operators are also implemented using the simple and SWP 40-bit multipliers explained in Section 2.3.2. These MACs are able to operate on signed as well as unsigned numbers. The simple version of the 40-bit MAC can perform one 40-bit multiplication and one 80-bit addition. The SWP version of the 40-bit MAC can perform either five 8-bit multiplications and five 16-bit additions or four 10-bit multiplications and four 20-bit additions or three 12-bit multiplications and three 24-bit additions or two 16-bit multiplications and two 32-bit additions or one 40-bit multiplication and one 80-bit addition.

Table 2.11 shows the area and CP results of simple and SWP enable multimedia 40-bit MACs. On an ASIC platform, maximum area overhead for implementing SWP capability is 3% only. This occurs because of the use of an efficient SWP multiplier which generates PPs efficiently (see table 2.7). Maximum CP overhead on an ASIC platform is 23%. On a FPGA platform, CLBs and CP overheads are 61% and 12%

	90nm CMOS ASIC			130nm CMOS ASIC			FPGA VirtexII		
	Nand Gates	**CP (ns)**	**Gates X CP** (norm)	**Nand Gates**	**CP (ns)**	**Gates X CP** (norm)	**CLBs**	**CP (ns)**	**CLBs X CP** (norm)
Simple	15163	6.14	1	10967	14.3	1	958	19.7	1
SWP	15573	7.55	1.26	11289	16.7	1.20	1543	22.0	1.8
Overhead	3 %	23 %	26 %	3 %	17 %	20 %	61 %	12 %	80 %

TABLE 2.11: Results of multimedia SWP MAC

respectively. Due to the use of CLBs as implementation resources [74], these overheads values are not in full coordination with ASIC overheads. This shows that ASIC resources (standard cells) well suit SWP designs rather than FPGA resources (CLBs).

2.5 Conclusions

This chapter describes the designing of SWP basic operators using classical as well as multimedia oriented subword sizes. Coordination between pixel sizes of multimedia applications and subword sizes in SWP operators can increase the efficiency of multimedia processors through better resource utilizations. As there exist no uniform arithmetic relationship between pixel sizes and processor word sizes therefore sometimes the implementation overheads of multimedia oriented SWP operators are little higher compared to classical SWP operators. But at the same time the extents to which these multimedia SWP operators increase the efficiency of overall processor easily undermine these small overheads. When used in any multimedia processor, these SWP operators provide speedup along with flexibility for multimedia applications. In the next chapter we will discuss the architectures of more complex operators which are required in multimedia applications. These operations include sum of absolute value difference operator, sum of product etc.

Chapter 3

SWP in multimedia operations

In Chapter 2 we have presented that how the SWP capability can be introduced in the architecture of basic operators like ADD, SUB, MULT, MAC etc. The overheads for incorporating classical and multimedia oriented SWP in basic operator design are also discussed. This chapter explores the implementation of different operations which are most commonly used in modern multimedia applications. The simple as well as SWP versions of these operations are implemented using the basic blocks presented in Chapter 2. Multimedia oriented SWP capabilities are introduced in different operator's designs. The operator blocks presented in this chapter will further be used for the implementation of reconfigurable multimedia operator design.

The rest of this chapter is organized as follows: Section 3.1 gives the brief overview of different operations which are required in multimedia applications. It also explain the requirement of sum of absolute values of differences (SAD) operation and its utility in multimedia applications. Section 3.2 presents different methods for the computation of absolute value of difference operation. Multimedia oriented SWP capability is also introduced in these methods. Section 3.3 describes simple as well as SWP SAD architectures and their comparisons are also presented. Section 3.4 describes the implementation of sum of product (SOP) operation and its utilization in multimedia applications. Simple and SWP versions of SOP operator are also presented along with their comparisons. Section 3.5 describes simple and SWP sum of additions/subtractions operation which is commonly used arithmetic operation. Finally we conclude the chapter in Section Section 3.6.

3.1 Multimedia arithmetic operations

In addition to basic arithmetic operators, customized operators are also required in multimedia processor design. Efficient implementation of these customize operators increases the performance of overall processor when working on different multimedia applications [6]. Most of the times the implementation of these multimedia operations involve the efficient utilization of SWP basic arithmetic operators. These customized multimedia operations include:

- **Sum of absolute values of differences SAD** is one of the most commonly used operation in multimedia applications (image/video compression). SAD is used for block matching in motion estimation algorithm.
- **Sum of products SOP** is used in different transform like discrete cosine transform DCT, discrete wavelet transform DWT etc. This operator involves the multiplication and the subsequent accumulation of products.
- **Sum of additions/subtractions** is used in the accumulation of sums or differences between pixels. Due to the generic nature of this operation, it is used in most of the multimedia applications.

In the next sections we will discuss the implementation of each of these multimedia operators in details. Multimedia SWP capability is introduced in the design of these operators to increase the parallelism which ultimately improves the performance of processor.

3.1.1 Sum of absolute values of difference SAD

SAD is most commonly used operation in motion estimation algorithms. SAD operator computes the sum of absolute values of differences between current frame pixels and reference frame pixels. For any particular block size, the absolute values of differences are accumulated to obtain final SAD value. SAD value indicates the resemblance between current frame block and reference frame block. Smaller SAD value corresponds to high resemblance between the blocks. In block matching, SAD values are calculated for different blocks in search window and the block with minimum SAD is selected as best possible match of current block. In any multimedia processor, SAD computations are used very frequently. Therefore the implementation of optimized SAD unit plays very important role in the overall performance of processor. SAD computations are given by Equation 3.1

$$\text{SAD} = \sum_{i=0}^{N-1} |a_i - b_i| \tag{3.1}$$

Where N is the number of pixels in block. SAD operation can be divided into two main blocks; determination of absolute value of difference |a - b| and the accumulation of absolute values. The pipelined architecture of SAD operator is shown in Figure 3.1.

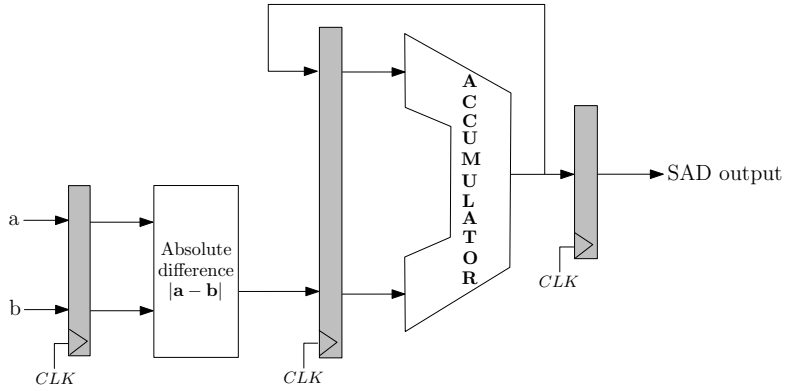

FIGURE 3.1: Sum of absolute difference operator

'a' and 'b' are the input pixels. |*a* - *b*| unit computes absolute value of difference between current pixels and reference pixels. These differences are accumulated recursively using *Accumulator* unit, until the SAD for whole block is obtained. The input and output data sizes of SAD operator depend upon the size of pixels and the size of blocks as well. For 8-bit pixel sizes, both inputs 'a' and 'b' are 8-bit unsigned numbers. The block size determines the data width at the output of SAD operator. If the block size is (16×16), it implies that there will be accumulation of 256 absolute values. To add 256 values, 8 extra bits ($2^8 = 256$) are required in the worst case to avoid any overflow. Therefore for a block size of (16×16) the data width at the output of SAD operator should be equal to (input pixel size + 8) bits. However the output data width of SAD operator can be reduced at the cost of some precision loss.

Determination of absolute values of difference |*a* - *b*| plays very important role in the overall SAD architecture. Efficient implementation of this unit ensures the better performance of overall SAD operator. In the following section we will discuss few methods for the implementation of |*a* - *b*| unit. Multimedia oriented SWP capability will also be introduce in |*a* - *b*| unit.

3.2 Determination of absolute value of difference |a - b|

|*a* - *b*| unit computes the absolute value of difference between two pixels. Pixels are usually stored as unsigned binary numbers therefore this operation is applied on unsigned data. |*a* - *b*| is computationally intensive operation because of the involvement of absolute value operation. Direct implementation of absolute value operation requires lot of hardware resources. Several methods have been proposed to implement the |*a* - *b*| operation without directly implementing the absolute value hardware [93, 94, 101, 102]. These methods tries to avoid the absolute operation by implementing it using other less expensive hardware resources like adders, subtractors etc. Indirect implementation of |*a* - *b*| unit increases the performance of overall SAD unit. Few of these methods for the computation of absolute difference value are discussed here and their performances are analyzed. These methods were proposed for single pixel values, however we will introduce multimedia oriented SWP capability in the computation of |*a* - *b*| operation. Using SWP capability several absolute difference values are calculated in parallel. Latter SWP |*a* - *b*| unit with high efficiency will be used in SWP SAD operator design.

3.2.1 Absolute value of difference : Method 1

This method was proposed in [93] for the computation of |*a* - *b*| on unsigned pixels. In this method |*a* - *b*| operation is computed using addition and inversion operations. This method is based on following Equations 3.2 and 3.3.

$$|a-b| = (a+\overline{b})+1 \quad \text{if} \quad a > b \tag{3.2}$$

$$|a-b| = \overline{(a+\overline{b})}+0 \quad \text{if} \quad a \leq b \tag{3.3}$$

Where 'a' and 'b' are input pixel values. $\overline{b}$ and $\overline{(a+\overline{b})}$ are bit inverted version or 1's complement of b and (a + $\overline{b}$) respectively. Equations 3.2 and 3.3 can be proved mathematically. The hardware realization of these equations for input pixel size of 8-bit is shown in Figure 3.2.

The eight bits of input pixel 'a' are represented by $(a_0 \ldots a_7)$ and the eight bits of input pixel 'b' are represented by $(b_0 \ldots b_7)$. In the first step bits of pixel 'b' are inverted using eight single bit inverters. Then *8-bit Adder* is used to add 'a' and '$\overline{b}$' vectors. The output of this adder are sum bits $(s_0 \ldots s_7)$ and the *carry* bit. The next operation is defined by the status of generated *carry* bit.

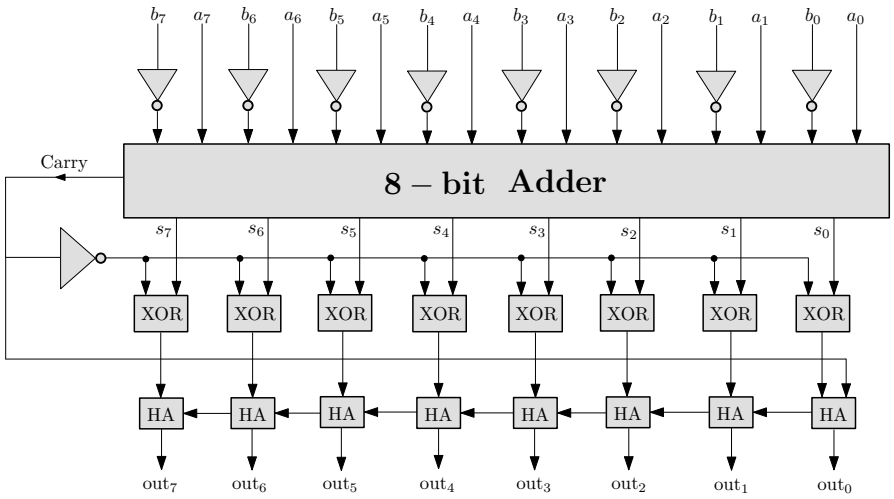

FIGURE 3.2: Absolute value of difference unit

- **Generated carry is '0'** If the *carry* bit is '0' it implies that $(a \leq b)$. It can be proved mathematically using following equations.

  $$b > a$$

  $$-a + b > 0$$

  $$2^n - 1 - a + b > 2^n - 1$$

  $$\overline{a} + b > 2^n - 1$$

  $$\overline{a} + b \geq 2^n$$

  The last step is possible because we are dealing with natural, non-fractional, numbers. The maximum value of $(\overline{a} + b)$ is $2 \times (2^n - 1) = 2^{n+1} - 2$. This is a (n + 1) bit number. The most-significant bit, with weight 2^n, is computed as the carry-out of the n-bit addition. Thus checking whether $\overline{a} + b \geq 2^n$ means checking whether the addition of the bit inverted 'a' and the operand 'b' produces a 'carry' out [102]. Thus if the *carry* bit is '0' it implies that $(a \leq b)$. Hence the implementation of $|a - b|$ is done using Equation 3.3. The *carry* bit is inverted to '1' and is used to take 1's complement of sum bits $(s_0 \ldots s_7)$. This 1's complement operation is realized in hardware using exclusive-OR (XOR) gates.

- **Generated carry is '1'** If the *carry* bit is '1' it implies that $(a > b)$. So the implementation of $|a - b|$ is done using Equation 3.2. The *carry* bit is inverted to '0' and is given at input of each *XOR* gate. When one of the input of *XOR* gate is '0', it will simply transfer the second input at the output. Due to this property of *XOR* gate, the sum bits $(s_0 \ldots s_7)$ are obtained at the output of *XOR* gate array without any inversion.

In both cases generated *carry* bit is added to the output of *XOR* gate array. As *carry* is a single bit value which is added to 8 bits, therefore half adders (HA) are used to perform this addition. At LSB the output of *XOR* gate is added to *carry* bit and out_0 bit is generated. At rest of the bit locations output of *XOR*s gates are added to the carry generated by the HA at previous bit location to generate outputs (out_1 ... out_7). This method implements the absolute difference $|a$ - $b|$ operation using inverters, addition block, array of XOR gates and array of HAs.

3.2.1.1 SWP Absolute value of difference : Method 1

SWP absolute value difference *SWP* $|a$ - $b|$ operator is used to compute multiple |a - b| operations in parallel. The inputs to *SWP* $|a$ - $b|$ operator are two input vectors which contain pixels from current block and reference block. SWP_{ctrl} signals are used to indicate the size of selected subwords. Based upon the selected subword size, the *SWP* $|a$ - $b|$ operator determines the boundaries of pixels packed in input vectors. The block diagram of *SWP* $|a$ - $b|$ operator is shown in Figure 3.3.

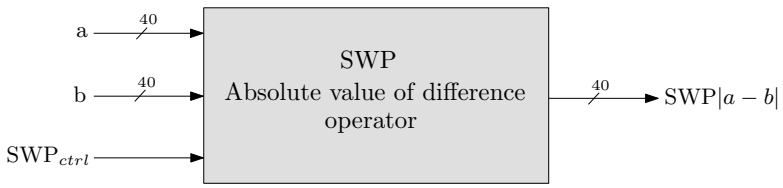

FIGURE 3.3: SWP Absolute value of difference operator

The implementation of *SWP* $|a$ - $b|$ operator is based on the method explained in previous section 3.2.1 for the computation of $|a$ - $b|$ on 8-bit pixels. *SWP* $|a$ - $b|$ operator supports multimedia oriented subword sizes of 8, 10, 12, 16-bit. The input vectors 'a' and 'b' which contain the packed pixels are of 40-bit data length. This word size of 40-bit is used as it gives good efficiency trade off with different multimedia oriented pixel sizes. Based upon the selected subword size, *SWP* $|a$ - $b|$ operator can perform either five |a(8bit) - b(8bit)| or four |a(10bit) - b(10bit)| or three |a(12bit) - b(12bit)| or two |a(16bit) - b(16bit)| or one |a(40bit) - b(40bit)| operations simultaneously. When the selected subword size is 8-bit, each 40-bit input vector is considered as five 8-bit packed pixels. Absolute value of difference operation is applied simultaneously on all five packed pixels. At the output five 8-bit absolute values of differences are obtained which are also packed in 40-bit register. Due to the subtraction operation no overflow occurs. Therefore 8 bits are sufficient to store the results of |a - b| operation on each subword. Similarly for other subword sizes absolute values of differences are obtained corresponding to selected subword size.

To analyze the performance of absolute difference operators, the comparison of simple and SWP $|a - b|$ operators are done on different target technologies. Simple $|a - b|$ operator can perform only one |a(40-bit) - b(40-bit)| operation. Where as the SWP $|a - b|$ operator can perform multiple parallel operations on subwords of different sizes. Using this method, the synthesis results of implementing simple and *SWP* $|a - b|$ operation on both ASIC and FPGA platforms are shown in Table 3.1.

	90nm CMOS ASIC			**130nm CMOS ASIC**			**FPGA VirtexII**		
	Nand Gates	**CP** (ns)	**Gates X CP**	**Nand Gates**	**CP** (ns)	**Gates X CP**	**CLB**	**CP** (ns)	**CLBs X CP**
Simple	1582	4.2	6644	1602	8.8	14098	211	14.4	3038
SWP	2072	4.4	9117	1971	9.4	18527	245	14.9	3651
Overhead	31 %	5 %	37 %	23 %	7 %	31 %	16 %	3 %	20 %

TABLE 3.1: Method 1: Synthesis results of absolute difference operator

As shown in table 3.1, the SWP $|a - b|$ operator consumes more resources compared to simple $|a - b|$ operator. This increase in resources occurs because SWP $|a - b|$ operator performs multiple parallel operations compared to simple $|a - b|$ operator which can perform only one operation. On 90nm ASIC technology SWP $|a - b|$ consumes 31% and 5% more area and CP compared to simple $|a - b|$ operator. Similarly on 130nm ASIC technology, SWP $|a - b|$ consumes 23% and 7% more area and CP compared to simple $|a - b|$ operator. On FPGA platform, SWP $|a - b|$ consumes almost 16% and 3% more area and CP compared to simple version. The product term (gates x CP) is normalized to simple $|a - b|$ operator. Compared to (a - b) operator, $|a - b|$ operator consumes almost 30% to 40% more area and 40% to 50% more CP on different target technologies.

3.2.2 Absolute value of difference : Method 2

This method [102] also implements the absolute operation using indirect method. The main steps in the computation of $|a - b|$ operation are following.

- **Find smaller operand** In the first step smallest of two input operand is determined. There are several ways to find the smaller operand. However in this method the CLA *carry generation* logic is used to find the smaller number.
- **Invert the smaller operand** In the second step the smaller of two operands is inverted. This inversion can be done by bit inversion. As the bit inversion is not exactly equal to the inversion of number. Therefore to nullify the unwanted effects of bit inversion, correction vector is used.

- **Addition** In the third step the greater operand is added to the inverted version of smaller operand. The correction vector produced in the second step is also added to the sum. This addition can be done by using any optimized adder.

Using these three steps, the absolute value of difference operation $|a - b|$ can be realized in hardware without the direct implementation of absolute operation. This process uses comparator, inverter and adder units for the implementation of $|a - b|$. The first two steps of comparison and inversion can be combined and implemented using the architecture shown in Figure 3.4.

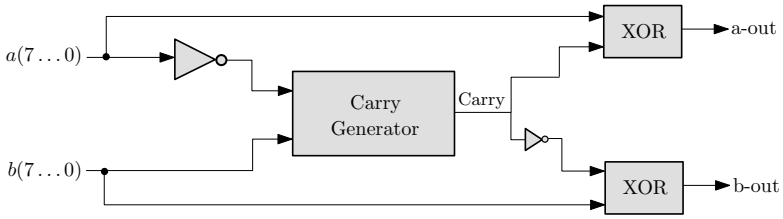

FIGURE 3.4: Comparison and inversion of smaller input

'a' and 'b' are two 8-bit input numbers. To determine smaller input, CLA *carry generator* circuitry is used because it can calculate the output carry efficiently. CLA *carry generator* block do not calculate the carry output by actually adding two input numbers. It only determines whether the carry will be generated or not when 'b' and bit inverted version of 'a' are added. Based on the value of carry output the smaller of two numbers can be determined.

- **When generated *carry* is '1'** If the generated *carry* is '1', it implies that b > a. So in this case smaller input 'a' is inverted. This inversion is done using *XOR* gate arrays. In the Figure 3.4, each *XOR* block contain the array of two input *XOR* gates. The number of gates in this array is equal to the number of bits in each input. One array of *XOR* gates is connected to input 'a'. In this array, one of the input of each *XOR* gate is connected to generated *carry* signal. The other array of *XOR* gates is connected to input 'b'. In this aray, one of the input of each *XOR* gate is connected to $\overline{carry}$ signal. When the generated *carry* is '1', the *XOR* gate array corresponding to input 'a' will generate inverted version of 'a' at output (*a-out*). As *carry* is '1', therefore *XOR* gate array corresponding to input 'b' will generate the same signal 'b' at the output (*b-out*).

- **When generated *carry* is '0'** When the generated *carry* is '0', the whole phenomenon will be reversed. If the *carry* generated is '0', it implies a > b. So smaller input 'b' will be inverted. In this case the *XOR* gate array corresponding to input

'a' will generate same output. Where as the *XOR* gate array corresponding to input 'b' will generate inverted version of 'b' at output.

In the final step *a-out* and *b-out* vectors are added along with correction vector. Correction vector is required because the bit inversion performed in the second step is not exactly equal to the negation of number. The bit inversion of any number 'X' is given by Equation 3.4.

$$\overline{X} = 2^n - 1 - X \tag{3.4}$$

In order to make bit inverted version ($\overline{X}$) equal to the negation of number (-X), the ($-2^n + 1$) term must be added to the right hand side of Equation 3.4. This is done with the help of correction vector. In practice correction vector is calculated for whole block of image and added once only.

3.2.2.1 SWP Absolute value of difference : Method 2

Using the method shown in Figure 3.4, SWP version of $|a - b|$ is also implemented. Instead of using single value, the absolute values of differences are determined for all the packed subwords. The subword boundaries are defined by SWP control signals. These signals direct the SWP $|a - b|$ unit about the size of pixels (8 or 10 or 12 or 16-bit) packed in the 40-bit input registers. Based on selected subword size, multiple $|a - b|$ operations are performed in parallel by *SWP* $|a - b|$ operator. The synthesis results of implementing simple and *SWP* $|a - b|$ operation on both ASIC and FPGA platforms are shown in Table 3.2.

	90nm CMOS ASIC			**130nm CMOS ASIC**			**FPGA VirtexII**		
	Nand Gates	**CP** (ns)	**Gates X CP**	**Nand Gates**	**CP** (ns)	**Gates X CP**	**CLB**	**CP** (ns)	**CLBs X CP**
Simple	1710	3.20	5472	1967	7.5	14753	220	12.9	2838
SWP	2231	3.46	7719	2301	8.2	18868	264	14.1	3722
Overhead	30 %	8 %	41 %	17 %	9 %	28 %	20 %	9.3 %	31 %

TABLE 3.2: Method 2: Synthesis results of absolute difference operator

As SWP $|a - b|$ operator perform multiple operations, therefore it requires more resources compared to the simple $|a - b|$ operator. The increase in resources also occurs because SWP operator performs operations on packed subwords which require arrangement and alignment before the computation.

3.2.3 Absolute value of difference : Method 3

This method is based on the calculation of absolute value of difference using comparator and subtractor units. In this method both (a - b) and (b - a) are computed in parallel with comparator. Based on the outcome of comparator, either (a - b) or (b - a) is selected as output. (a - b) is selected as output when (b < a) and (b - a) is selected as output when (a < b). The comparison of two input numbers can be done either by CLA carry generation method explained in previous section 3.2.2 or the synthesis tool can be allowed to implement any efficient comparison scheme available in the library. The block diagram of $|a$ - $b|$ using this method is shown in Figure 3.5.

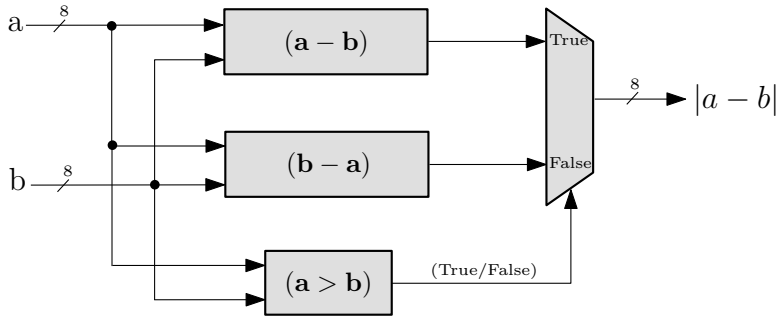

FIGURE 3.5: Absolute difference unit

As shown in the Figure 3.5, this method is very simple. The output of subtractors is selected with the help of 2 to 1 multiplexer.

3.2.3.1 SWP Absolute value of difference : Method 3

Multimedia oriented SWP capability can be introduced in the architecture of $|a$ - $b|$ unit shown in Figure 3.5. For this purpose absolute difference is calculated for each subword in the input registers. The comparator unit compares the corresponding subwords in input registers. The absolute difference output for each subword is selected based on the output of comparator units. Table 3.3 shows the synthesis results on ASIC and FPGA platform for the implementation of simple and SWP $|a$ - $b|$.

Although SWP $|a$ - $b|$ operator requires more resources compared to simple $|a$ - $b|$ but due to parallel operations on multiple subwords, the overall performance of SWP $|a$ - $b|$ operator is better than simple $|a$ - $b|$ operator. Moreover the SWP $|a$ - $b|$ operator can support five different subword sizes (8, 10, 12, 16 and 40-bit). Where as the simple $|a$ - $b|$ operator can perform operations on 40-bit data only.

	90nm CMOS ASIC			130nm CMOS ASIC			FPGA VirtexII		
	Nand Gates	**CP** (ns)	**Gates X CP**	**Nand Gates**	**CP** (ns)	**Gates X CP**	**CLB**	**CP** (ns)	**CLBs X CP**
Simple	1313	3.87	5081	1012	8.8	8906	168	13.6	2285
SWP	1602	4.34	6952	1326	9.6	12730	229	15.2	3481
Overhead	22 %	12 %	37 %	31 %	9 %	43 %	36 %	11 %	52 %

TABLE 3.3: Method 3: Synthesis results of absolute difference operator

3.2.4 Comparison of SWP Absolute value of difference operators

Comparison of all the three implementations of *SWP* |*a* - *b*| are done in order to select the best possible operator for further use in multimedia operator design. Figure 3.6 shows the area and critical path (CP) comparisons of three implementations of *SWP* |*a* - *b*| operator explained so for.

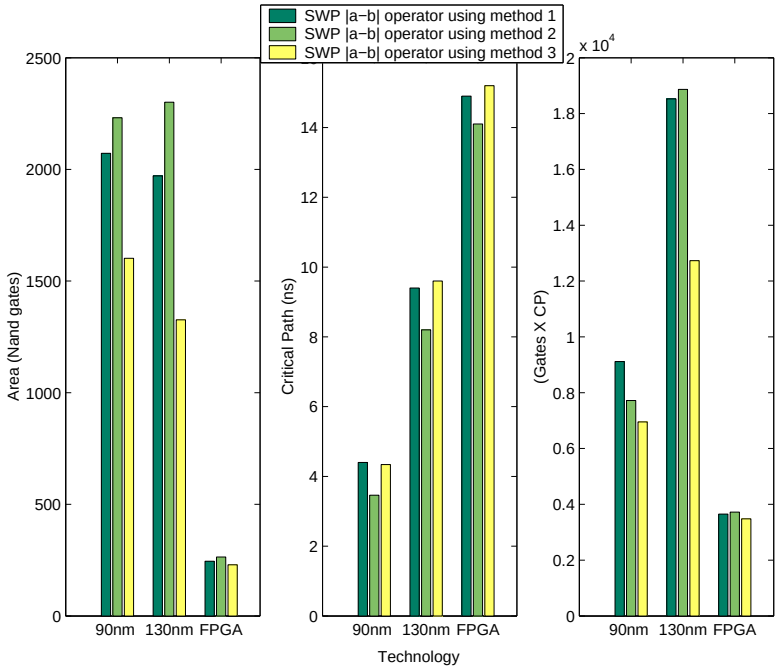

FIGURE 3.6: Comparison of SWP absolute difference operators

As shown in Figure 3.6, each method consumes different area and CP for the implementation of *SWP* |*a* - *b*| operator. This variation in area and CP corresponds to the different methods of implementation. On each target technology, *Method 3* gives minimum area and *Method 2* gives minimum CP. *Method 2* gives minimum CP because it uses CLA carry generation scheme for the determination of smaller operand. However due to the use of CLA carry logic area increases accordingly. *Method 3* consumes minimum area as it uses optimized comparator operator from the library of synthesis tool.

On all the target technologies the efficiency (gates x CP) of *Method 3* is better than other methods. Based on area and CP constraints, any of these operators can be use for the computation of SWP absolute difference. However due to the better efficiency, we will use *Method 3* for SWP absolute difference implementation in multimedia operator design.

3.3 SWP SAD operator

SWP |*a* - *b*| units explained in the previous section can be used to design *SWP SAD* operator. This operator performs parallel operations on pixels from current frame and reference frame and generates the SAD value. The level of parallelism depends upon the selected subword size. For different subword sizes, *SWP SAD* operator perform operations on multiple pixels in each cycle and compute the overall SAD value of block in less time compared to *simple SAD* operator. In *SWP SAD* design multimedia oriented subword sizes (8, 10, 12 and 16 bits) are considered rather than classical subword sizes (8, 16, 32 bits etc.). For 8-bit subword size *SWP SAD* unit perform SAD operations on five pixels in each clock cycle. Therefore the overall SAD operation of each block is almost five times faster than *simple SAD* unit which perform only one operation in each clock cycle. Overall speed up of SWP operator is not exactly equal to five times because some cycles are also required for arrangement and alignments of subwords. The pipelined architecture of the proposed *SWP SAD* operator is shown in Figure 3.7.

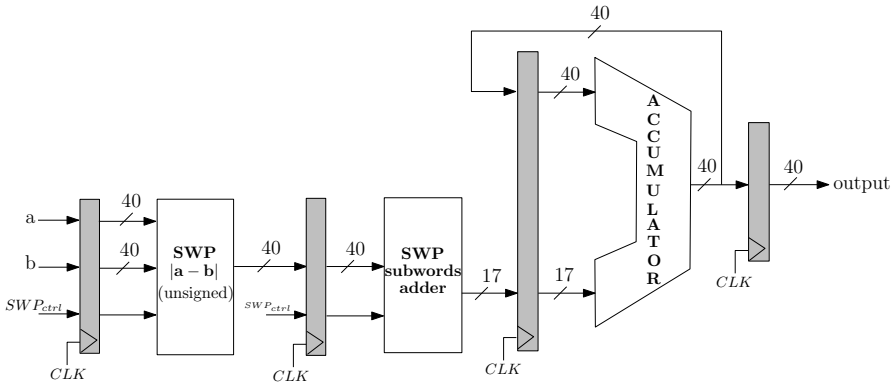

FIGURE 3.7: SWP sum of absolute value difference SAD

As shown in Figure 3.7, the inputs to *SWP SAD* operator are two 40-bit input vectors and SWP control signals (SWP_{ctrl}). The output is a 40-bit SAD value. SWP SAD operator consists of three main units.

- **SWP |a - b| unit :** This unit computes the absolute values of differences on packed subwords. In this design, the architecture of *SWP* |*a* - *b*| unit explained

in Section 3.2.3 (*Method 3*) is used as it gives better overall efficiency (gates x CP) compared to other methods. However based on the required timing and area constraints, the *SWP* |*a* - *b*| architectures explained in Section 3.2.1 and 3.2.2 can also be used. The output of *SWP* |*a* - *b*| unit is in the form of packed subwords within 40-bit word register. These packed subwords are given to *SWP subword adder* unit.

- **SWP subword adder unit :** Based upon selected subword size (SWP_{ctrl}), *SWP subword adder* unit perform the addition of subwords packed in 40-bit register. The output of *SWP subword adder* unit is a single value. For 8-bit subword size *SWP subword adder* unit performs the addition of five 8-bit packed subwords. Similarly for other subword sizes the packed subwords are added to get single value at the output. The architecture of *SWP subword adder* unit is shown in Figure 3.8.

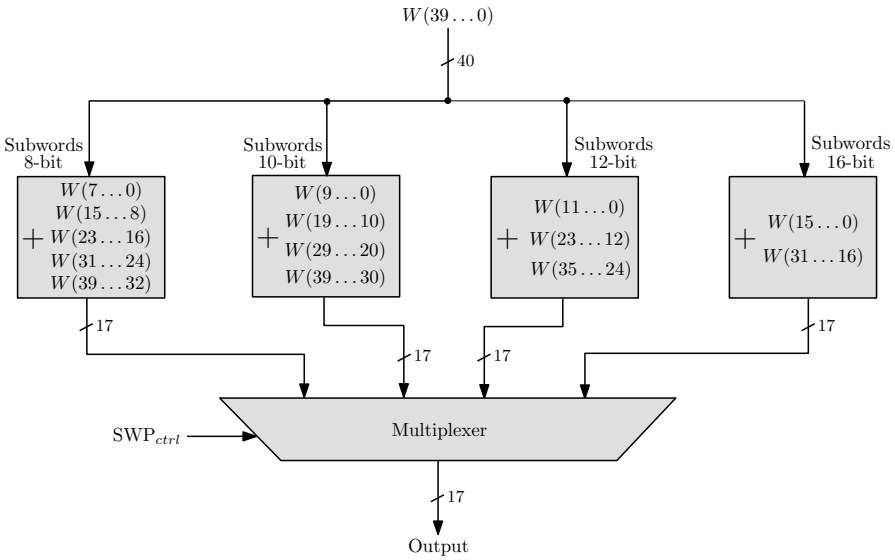

FIGURE 3.8: SWP subword adder unit for SWP SAD operator

The bit width at the output of *SWP subword adder* unit should be selected in such a way that overflow should not occur for any selected subword size. This can be ensured if the output bit width of *SWP subword adder* unit is selected on the basis of largest subword size (16-bit). This will ensure that no overflow will occur for smaller subword sizes as well. In case of 16-bit subword size, the input to *SWP subword adder* unit are two 16-bit |a - b| values packed in one register. *SWP subword adder* unit adds these 16-bit |a - b| values and generate 17-bit value. Therefore the maximum bit-width requirement at the output of *SWP subword adder* unit is 17 bits. For other smaller subword sizes (8, 10 and 12 bits), the output bit-width requirements of *SWP subword adder* unit is less than 17 bits. When the output of any addition block in *SWP subword adder* unit is less than 17

bits, it is zero padded before giving it to multiplexer unit. The multiplexer selects the output on the basis of selected subword size (SWP_{ctrl}). The 17-bit output of *SWP subword adder* unit is given to accumulator unit

- **Accumulator unit :** *Accumulator* unit is used to accumulate recursively the 17-bit output generated by *SWP subword adder* unit in each clock cycle. As the accumulation is done in each clock cycle therefore the bit growth will also occur at the output of accumulator. The output of accumulator is 40-bit wide which results in a 23 guard bits (40 - 17 = 23) to avoid any overflow. Therefore when the maximum subword size of 16-bit is selected then the operator can perform the accumulation of at least 2^{23} values without any overflow. For other smaller subword sizes the number of values which can be accumulated without any overflow increases further.

3.3.1 Comparison of simple and SWP SAD operator

The comparison of simple and SWP operators explained above can be done on the basis of computation time and hardware resources. Computation time is the time required to compute the SAD of whole block of image. For (16x16) image block with pixel size of 8 bits, SWP SAD unit requires 55 cycles to compute SAD value of whole block. On the other hand simple SAD operator requires 259 cycles to compute the SAD value of (16x16) blocks. Therefore by using SWP SAD operator the numbers of cycles have been reduced by almost 80%. Maximum parallelism is achieved for smallest subword size of 8-bit. For other subword sizes, the number of cycles required for the computation of SAD increases accordingly.

The cost for parallelism is the increase in hardware resources of SWP SAD unit. However due to the efficient design these SWP overheads are very less compared to the speed-up achieved by SWP. Table 3.4 shows the synthesis results of simple and SWP SAD units on different target technologies.

	90nm CMOS ASIC			**130nm CMOS ASIC**			**FPGA VirtexII**		
	Nand Gates	**CP** (ns)	**Gates X CP**	**Nand Gates**	**CP** (ns)	**Gates X CP**	**CLB**	**CP** (ns)	**CLBs X CP**
Simple	3197	4.1	13107	4049	9.2	37250	204	14.7	2999
SWP	4368	4.4	19219	5128	9.7	49742	262	15.3	4009
Overhead	37 %	7 %	47 %	27 %	6 %	34 %	28 %	5 %	34 %

TABLE 3.4: Synthesis results of SAD operator

As shown in Table 3.4, SWP SAD operator requires more area and CP resources compared to simple SAD operator. The main reason for this increase is that SWP SAD operator performs parallel operations compared to simple operator. On 90nm ASIC, 130nm ASIC and FPGA platforms, SWP SAD operator consumes 37%, 27% and 28% more area compared to simple SAD operator. The major contribution for this area increase is *SWP subword adder* unit. This unit consumes almost 20% to 25% area of SWP SAD operator. The registers and other glue logic corresponding to *SWP subword adder* unit also consumes some area. This unit is not required in simple SAD operator. On all target technologies the CP overheads of SWP SAD operator are less than 10%.

3.4 Sum of products (SOP)

The sum of products (SOP) operation is most commonly used in multimedia applications. SOP operation is given by Equation 5.9

$$\text{dot product} = \sum_{i=0}^{N-1} (a_i \times b_i) \tag{3.5}$$

Discrete cosine transform (DCT) is one of the most familiar algorithms which utilize SOP operation. DCT is used in multimedia applications for video compression. DCT shifts the image from time domain to frequency domain. In DCT, product of pixel values and coefficients are accumulated to get values of image components in frequency domain [12, 63]. The most general form of DCT is given by Equation 3.6.

$$X(K) = \alpha(k) \sum_{n=0}^{N-1} u(n) \frac{cos(2n+1)}{2N} K\pi \tag{3.6}$$

Where

$$\alpha(0) = \sqrt{\tfrac{1}{N}}$$

$$\alpha(k) = \sqrt{\tfrac{2}{N}} \qquad 1 \leq \text{k} \leq \text{N-1}$$

Where u(n) is the input signal and X(K) is the transformed output. As shown in the Equation 3.6, the DCT is a computationally complex operation. Different algorithms have been proposed [3], [9], [7], [12] to reduce the complexity of DCT. But the computations of SOP terms remain there in all the algorithms. Efficient implementation of SOP

increases the performance of processor for multimedia applications. The SOP operation involve both multiplication and accumulation operations. Multiplication requires lot of hardware resources compared to addition. Therefore it is obvious that hardware resources required for SOP operator will be higher than SAD operator. The pipelined architecture of simple SOP operator is shown in Figure 3.9.

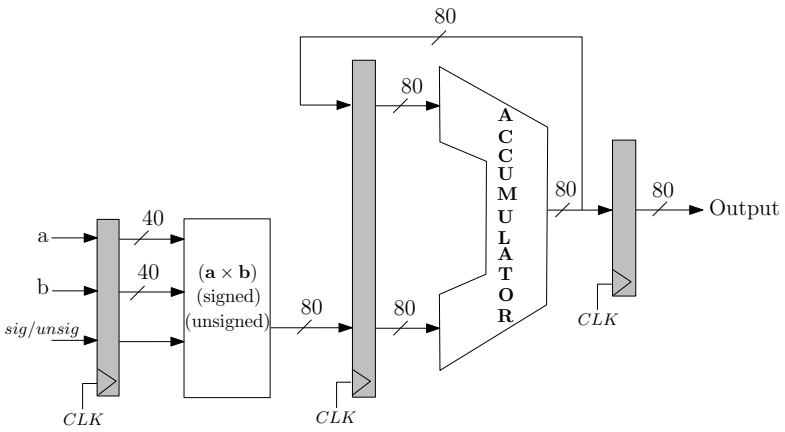

FIGURE 3.9: Pipelined architecture of sum of products operator

'a' and 'b' are two 40-bit inputs. *sig/unsig* is a single bit input control signal for the selection of signed or unsigned operations. The process of SOP can be divided into two main blocks.

- **Multiplication :** In the first pipelined stage the *(a × b)* unit computes the product of two 40-bit numbers 'a' and 'b' and generate 80-bit product. This multiplication can be done by using any of the algorithms like Booth recoding, array of AND gates etc. (explained in Section 2.3 of Chapter 2). In the implementation of SOP operator, we have used an array of AND gates for the generation of partial products. This method can be used for the multiplication of signed as well as unsigned numbers. On the other hand methods like Booth recoding are dedicated for the multiplication of signed 2's complement binary numbers only. Compared to other complex methods, multiplication using array of AND gates is simple and can easy be extended to SWP architectures. Based upon the input control signal *sig/unsig*, the multiplication is performed on either signed 2's complement numbers or on unsigned numbers.

- **Accumulation :** In the next pipeline stage the product generated by *(a × b)* unit is accumulated recursively by using *Accumulator* unit. The data width at the output of an *Accumulator* unit can be increased based upon the requirement of number of values to be accumulated.

3.4.1 SWP sum of products

The *SWP SOP* operator performs parallel operations on the subwords packed in input registers. In *SWP SOP* operator multimedia oriented subword sizes of 8, 10, 12 and 16-bits are considered rather than classical subword size. These subword sizes are in coordination with pixel sizes in most multimedia applications. The pipelined architecture of *SWP SOP* operator is shown in Figure 3.10.

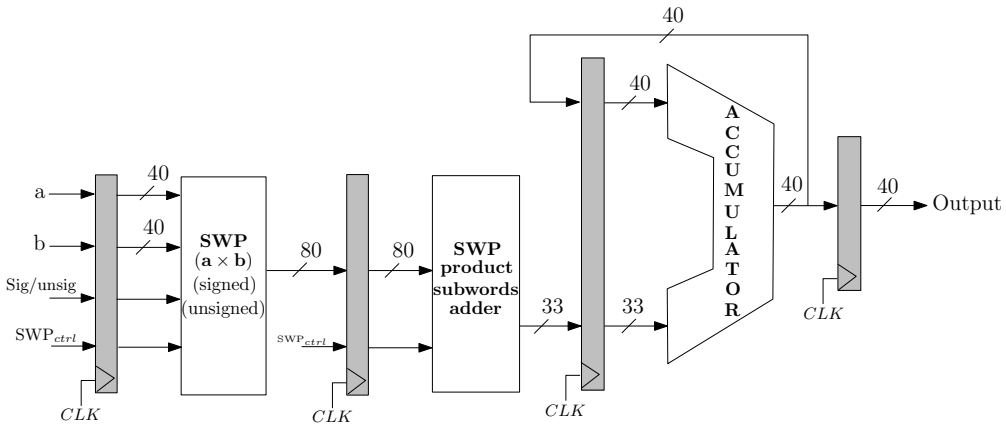

FIGURE 3.10: SWP sum of product operator

The inputs to *SWP SOP* operator are two 40-bit vectors and control signals SWP_{ctrl} and *sig/unsig*. The 40-bit input vectors contain packed subwords. Each 40-bit input vector contain either five 8-bit subwords or four 10-bit subwords or three 12-bit subwords or two 16-bit subwords. SWP_{ctrl} signals are used to select subword size as per the requirements. To differentiate between signed and unsigned operations *sig/unsig* control signal is used. This control signal indicates whether the packed subwords are signed 2's complement numbers or unsigned numbers. The main blocks in the architecture of *SWP SOP* operator are given below.

- **SWP (a × b) :** It is used to multiply the subwords packed in the input registers. The internal architecture of this unit is based upon the SWP multimedia multiplier operator explained in Section 2.3.2 of Chapter 2. This SWP multiplier is selected because it requires minimum hardware resources to incorporate multimedia oriented SWP capability. The design of *SWP (a × b)* is based on SWP multiplier prposed in [56]. However the SWP multiplier used in our *SWP SOP* operator supports multimedia oriented subword sizes rather than classical subword sizes. It gives better performance when working on different pixel sizes. Based upon the selected subword size (SWP_{ctrl}) and number format (*sig/unsig*), the signed or unsigned multiplication is performed on packed subwords. This unit generate 80-bit product at the output. The 80-bit output contains product subwords. Due to the

multiplication operation, the size of each product subword is double than the size of corresponding input subword. For 8-bit subword size, each product subword consists of 16-bit. Similarly for 10-bit subword size, each product subword is 20-bit wide and so on. The product subwords are given to *SWP product subwords adder* unit.

- **SWP product subwords adder :** This unit performs the addition of packed product subwords. This addition is done in the same manner as explained in Section 3.3 for SAD operation. However in *SWP SOP* operator, the inputs to *SWP product subwords adder* unit are product subwords (16, 20, 24, 32-bit) rather than simple subwords (8, 10, 12, 16-bit). Therefore the number of bits required to represent the output of *SWP product subwords adder* unit also increases accordingly. The additions performed by *SWP product subwords adder* unit corresponding to different selection of subword size are shown in Figure 3.11.

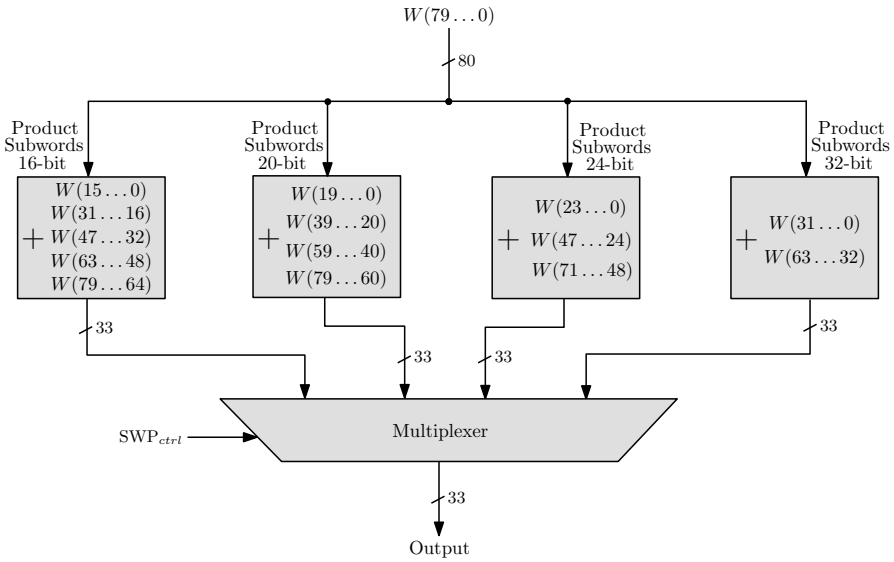

FIGURE 3.11: SWP subword adder unit for SWP SOP operator

To ensure the addition of product subwords without any overflow, data size at the output of *SWP product subwords adder* unit is selected on the basis of maximum product subword size. Maximum product subword size is 32-bit which is generated when two 16-bit subwords are multiplied. As shown in Figure 3.11, when the selected subword size is 16-bit the 80-bit product register contains two 32-bit product subwords. These 32-bit subwords are added by *SWP product subwords adder* unit to generate 33-bit output without any overflow. For other smaller subword sizes, the bit width requirements at the output of *SWP product subwords adder* unit are less. For 8, 10, 12-bit subword sizes, the output bit width requirements of *SWP product subwords adder* unit are 17, 21, 25 bits respectively. By using

bit width corresponding to maximum size subword, the probability of bit overflow for all the smaller subwords is also avoided. On the basis of selected subword size (SWP_{ctrl}), the multiplexer selects the output of *SWP product subwords adder* unit. In the next stage, the 33-bit output of *SWP product subwords adder* unit is given to *Accumulator* unit.

- **Accumulator :** This unit accumulates the 33-bit outputs from *SWP product subwords adder* units recursively in each clock cycle. As the input to accumulator is always single value rather than packed subwords, therefore SWP capability is not required in *Accumulator* design. The output of the accumulator is 40-bit value. When the maximum subword size (16-bit) is selected the accumulator can accumulates ($2^7 = 128$) values without any overflow. For other smaller subword sizes the number of values which can be accumulated increases further.

3.4.2 Comparison of simple and SWP sum of products operator

For different block sizes, *SWP SOP* unit computes the sum of products in less time compared to *simple SOP* operator. This decrease in time occurs because *SWP SOP* operator performs multiple parallel operations in each clock cycle. For (16x16) image size, the *SWP SOP* operator requires 80%, 74%, 65% and 49% less cycles to compute sum of products when the pixel sizes are 8, 10, 12 and 16-bit respectively. This decrease in cycles occutr because *SWP SOP* operator fully utilize the data path, registers, operating units etc. even though it is working on low precision pixel data.

Due to the multiple parallel operations, the hardware requirements of *SWP SOP* operator is usually higher than *simple SOP* operator. However the efforts are made to increase the parallelism along with minimum resource increase. Table 3.5 shows the synthesis results of simple and multimedia oriented *SWP SOP* operators.

	90nm CMOS ASIC			**130nm CMOS ASIC**			**FPGA VirtexII**		
	Nand Gates	**CP** (ns)	**Gates X CP**	**Nand Gates**	**CP** (ns)	**Gates X CP**	**CLB**	**CP** (ns)	**CLBs X CP**
Simple	17220	6.1	105042	13847	14.1	195243	1031	19.9	20517
SWP	20665	7.2	148788	16061	17.3	277855	1903	21.7	41295
Overhead	20 %	18 %	42 %	16 %	23 %	41 %	85 %	9 %	95 %

TABLE 3.5: Synthesis results of sum of products operator

SWP SOP operator consumes more resources compared to *SWP SAD* operator because of the involvement of multiplication operation. Multiplication is more costly operation compared to absolute difference or addition operation. However due to the use of efficient

methods for partial product generation and addition in SWP multiplier unit, the area and CP overheads of *SWP SOP* operator are not much higher than *simple SOP* operator. On ASIC platform, the area and CP overheads are between 15% to 25% only. These overheads also include the area and CP consumed by *SWP product subwords adder* unit and corresponding interconnection cost. On FPGA platform, due to the use of CLBs rather than gates, area and CP overheads are 85% and 9% respectively.

3.5 Sum of additions/subtractions

This operation is required in the situation when the recursive accumulation of pixels sum or difference is needed in certain computations like DCT, discrete wavelet transform (DWT), finite impulse response (FIR) filters, infinite impulse response filters (IIR) filters etc. The sum of addition/subtraction operation is given by Equation 3.7.

$$\text{Sum of addition/subtraction} = \sum_{i=0}^{N-1} (a_i \pm b_i) \tag{3.7}$$

The hardware architecture for *sum of additions/subtractions* unit is shown in Figure 3.12.

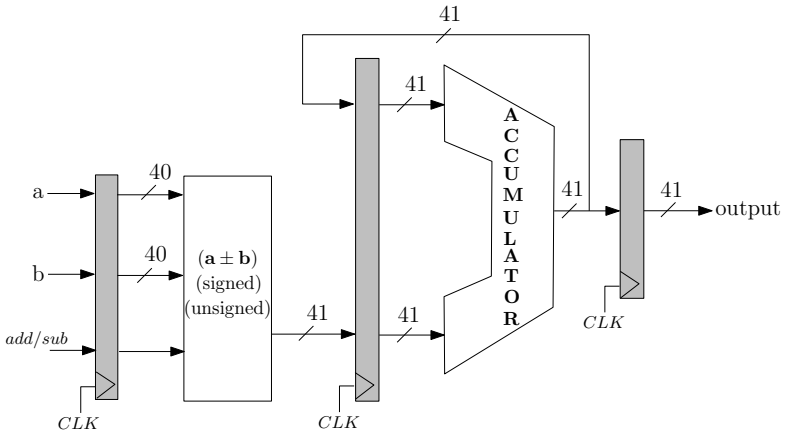

FIGURE 3.12: Sum of addition/subtraction operator

'a' and 'b' are two 40-bit input numbers (signed/unsigned). The input control signal *add/sub* is used to select addition or subtraction operation. Based upon selected operation, $(a \pm b)$ unit either performs addition or subtraction. From the hardware point of view, the subtraction process is almost similar to addition process. In subtraction process, the 2's complement of the operand which needs to be subtracted is taken before the addition. To avoid any overflow, the output of $(a \pm b)$ unit is represented by 41 bits. Due to the allocation of one extra bit, overflow can not occur even though when

both inputs 'a' and 'b' gets maximum values. The output of $(a \pm b)$ unit is accumulated recursively by using *Accumulator* unit. In each clock cycle, the sum of additions or subtractions is obtained at the output of operator.

3.5.1 SWP sum of additions/subtractions

By using multimedia oriented SWP in $\sum(a \pm b)$ architecture we can perform the same operation on multiple subwords in parallel. By doing so all the operator's resources are fully utilized even though the size of pixels is less than the word size of operator. Keeping in view the utility of $\sum(a \pm b)$ operation, SWP version of this operator is also designed. The architecture of $SWP\sum(a \pm b)$ operator is shown in Figure 3.13.

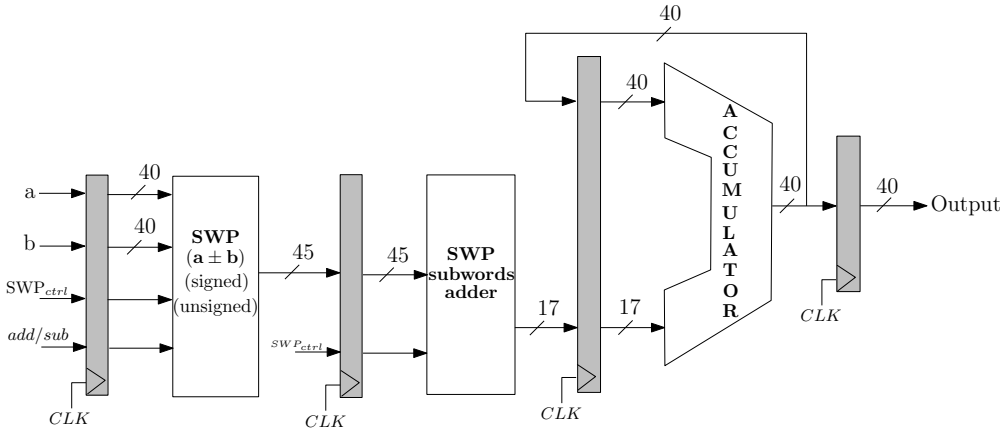

FIGURE 3.13: SWP sum of addition/subtraction operator

'a' and 'b' are two 40-bit input vectors which contain packed subwords. $SWP\sum(a \pm b)$ mainly consists of three main units.

- **SWP (a ± b) :** This unit is used to compute sum or difference of subwords packed in two input registers. This unit can perform either five $\{a(8bit) \pm b(8bit)\}$ or four $\{a(10bit) \pm b(10bit)\}$ or three $\{a(12bit) \pm b(12bit)\}$ or two $\{a(16bit) \pm b(16bit)\}$ or one $\{a(40bit) \pm b(40bit)\}$ operations. Its architecture is based on SWP adders explained in section 2.2 of Chapter 2. Based upon the design constraints, the adders/subtractors between the subword control logic can be of any type. In SWP (a ± b) unit, optamized adder/subtractor from library are used between control logic. There are two external control signals to this unit which are SWP_{ctrl} and *add/sub* signals. SWP_{ctrl} signals are used for subword size selection and *add/sub* control signal is used to select the desire operation. In case of addition or subtraction, overflow of result is avoided by using one extra bit. However in SWP

operator, one extra bit is allocated to each resultant subword to avoid any overflow. Therefore the number of the overflow bits is equal to the number of subwords packed in the register. For different subword sizes the numbers of subwords packed in word size registers are different. Therefore the number of overflow bits requirement will be different for different subword sizes. The arrangement of overflow bits corresponding to different subword sizes is shown in Figure 3.14.

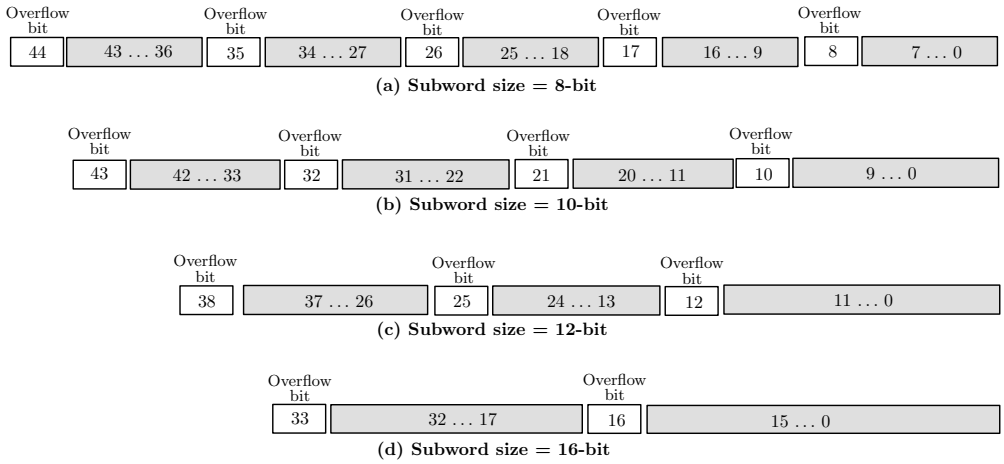

FIGURE 3.14: Arrangement of over flow bits for different subword sizes

Maximum number of subwords are packed in word size register when the selected subword size is minimum. In this case five subwords (maximum number of subwords) are packed in 40-bit register when the selected subword size is 8-bit (minimum subword size). Therefore five overflow bits are required for 8-bit subword size. For other subword sizes the number of overflow bits required are less. For 10, 12 and 16-bit subword sizes four, three, two overflow bits are required respectively. Due to these overflow bits, the output of *SWP(a ± b)* unit consists of 45 bits instead of 40 bits. The 45-bit output of *SWP (a ± b)* unit is selected based upon the smallest subword size. The output of *SWP(a ± b)* unit is given to *SWP subwords adder* unit.

- **SWP subwords adder :** This unit adds the packed subwords and generates single value at the output. As the output of *SWP(a ± b)* unit consist of 45 bits. Therefore each subword is represented by (selected subword size + 1) bits. The addition of these subwords using *SWP subwords adder* unit is shown in Figure 3.15.

  As shown in Figure 3.15, to avoid any overflow, 18 bits are allocated at the output of *SWP subwords adder* unit. These bits are selected on the basis of maximum subword size of 16-bit. The 18-bit output of *SWP subwords adder* unit is given to *Accumulator* unit.

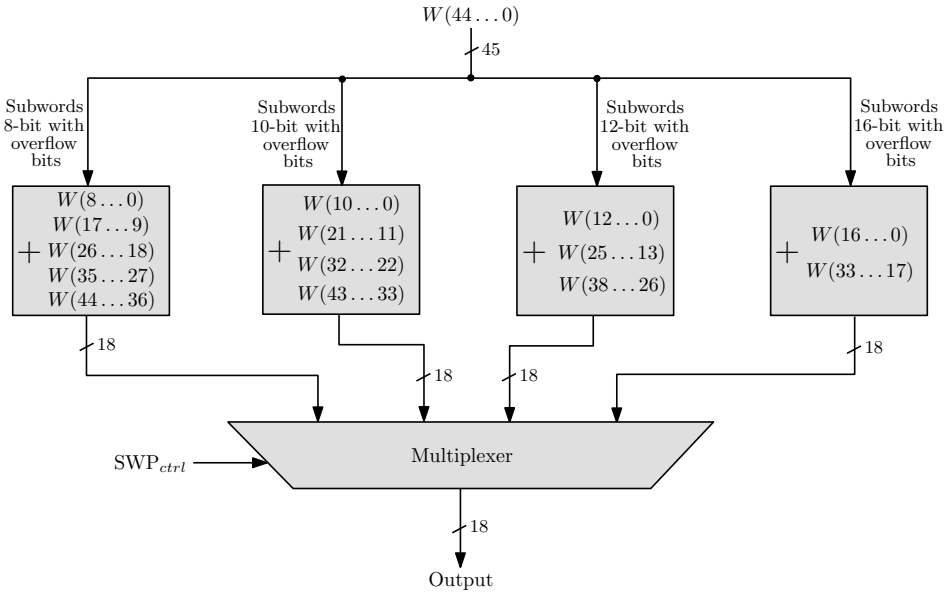

FIGURE 3.15: SWP subword adder unit for SWP $\sum(a \pm b)$ operator

- **Accumulator :** This unit performs the recursive accumulation of 18-bit data in each clock cycle. The output of accumulator consists of 40 bits. For maximum subword size of 16-bit, *Accumulator* unit can perform at least 2^{22} accumulations without any overflow.

3.5.2 Comparison of simple and SWP $\sum(a \pm b)$ operators

Due to parallelism, SWP $\sum(a \pm b)$ operator performs more operations compared to simple $\sum(a \pm b)$ operator. The actual speed up achieved will depend upon selected subword size. Smaller the subword size, higher will be the parallelism. Table 3.6 shows the synthesis results of simple and SWP $\sum(a \pm b)$ on both ASIC and FPGA platforms.

	90nm CMOS ASIC			**130nm CMOS ASIC**			**FPGA VirtexII**		
	Nand Gates	**CP** (ns)	**Gates X CP**	**Nand Gates**	**CP** (ns)	**Gates X CP**	**CLB**	**CP** (ns)	**CLBs X CP**
Simple	2370	2.45	5807	2789	5.95	16595	204	14.1	2876
SWP	3697	2.79	10315	3963	6.84	27107	291	15.2	4423
Overhead	56 %	14 %	78 %	42 %	15 %	63 %	43 %	8 %	54 %

TABLE 3.6: Synthesis results of $\sum(a \pm b)$ operator

In SWP $\sum(a \pm b)$ operator, resources are consumed by *SWP* $(a \pm b)$ unit, *SWP subwords adder* unit, *Accumulator* unit and interconnection network. These units consume almost 24%, 27%, 9% and 40% area respectively. Table 3.6 shows the area and CP overheads of

SWP operator. Although the SWP $\sum(a \pm b)$ operator consume more resources compared to simple $\sum(a \pm b)$ operator but its efficiency is much better compared to simple operator due to the parallel processing of subwords.

3.6 Conclusions

In this chapter different operations which are most commonly required in multimedia applications are discussed. Simple and SWP version of each of these operators are designed and the comparisons are made on different target technologies. Most of the time SWP operators require more resources compared to the simple operators. However the efficiency of SWP operators is higher because they can process multiple data items in each clock cycle. By using multimedia oriented SWP capability, we can increase the efficiency of operator many times at the cost of only small increase in hardware resources. This efficiency increase depends upon the selected subword size. Smaller subword sizes offer more parallelism compared to larger subword sizes. Keeping in view the performance of SWP operators, the small area and CP overheads are of less importance. Based upon the SWP operators discussed so for, the architecture of reconfigurable multimedia operator will be discussed in next chapter. This operator can perform different basic as well as multimedia operations on multiple subword size data.

Chapter 4

Reconfigurable SWP operator for multimedia processing

For performance enhancement, reconfigurable processors have to overcome the overheads of reconfigurations such as complexity of interconnection network and reconfiguration time. In the processors dealing with multimedia applications these overheads can be reduced by providing the reconfigurability inside the processing units rather than at interconnection level. Due to low precision data nature of multimedia applications, reconfiguration at operator level also provides additional speedup through parallel execution of low precision data. In this chapter pipelined architecture of reconfigurable coarse grain subword parallel (SWP) operator is presented for multimedia applications. This operator not only eliminates the need of reconfiguration time but also provide the reconfigurability at both data size level (different pixel data sizes) and at operation level (different multimedia oriented operations). This ensures the better utilization of processor resources and reduces the reconfiguration overheads significantly. The contents of this chapter are based on our publications [50] and [69].

The rest of this chapter is organized as follows: Section 4.1 gives the brief overview of reconfigurations at different levels in the processor's design and their overheads. In section 4.2, pipelined architecture of coarse grain reconfigurable SWP operator is presented. In section 4.3, architectures of all the basic SWP units which are used in the construction of reconfigurable operator are described. In section 4.4 different units for the arrangement and alignment of subwords and the interconnection units which are used in reconfigurable SWP operator are presented. Section 4.5, elaborates different multimedia operations which can be performed using reconfigurable SWP operator. In section 4.6, synthesis results of implementing proposed reconfigurable operator on both ASIC and FPGA platforms are summarized. Section 4.7 contains the discussion on the

performance of our reconfigurable operator compared to operators in state of the art DSP chips. Finally we conclude the chapter in section 4.8.

4.1 Reconfigurable architectures

Reconfiguration is one of the most efficient ways to increase the efficiency of processor. In the reconfiguration process, the processor adopt itself dynamically for the new set of tasks [5], [106], [86]. Using reconfigurable processors, several tasks can be performed on the same hardware merely by changing its configuration. Reconfigurable computing is intended to fill the gap between hardware and software, achieving potentially much higher performance than software, while maintaining a higher level of flexibility than hardware [18]. There are several levels at which the reconfiguration can be done during the processing.

4.1.1 Reconfiguration at interconnection level

Multimedia applications are normally computationally intensive with low precision pixel data such as 8, 10, 12 or sometimes 16-bits. Traditional reconfiguration at interconnection level [21], [81], [92] tries to increase the flexibility and efficiency of processor through the reconfiguration of interconnection network at bit level for new set of applications without concentrating on the internal reconfigurability of different processing units. Thus, for each new application, interconnection network reconfigures itself and processing units remain almost idle during reconfiguration process [4], [79], [96]. An overview of reconfiguration at this level is shown in Figure 4.1.

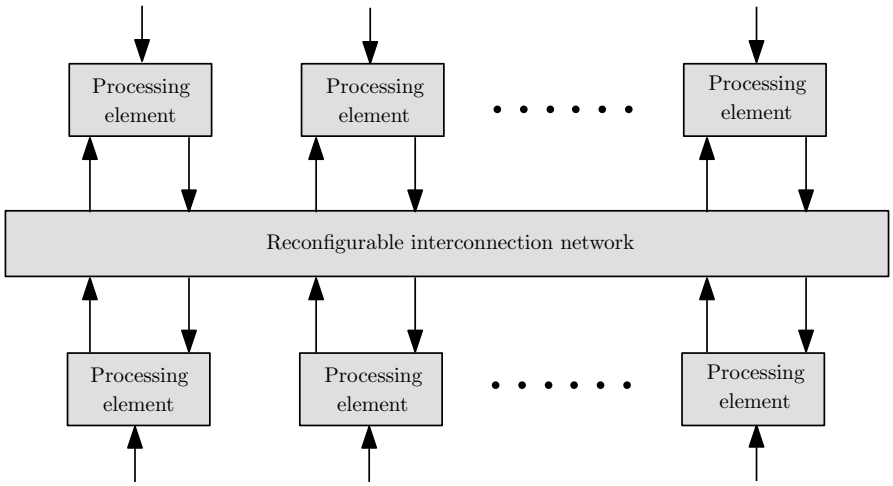

FIGURE 4.1: Reconfiguration at interconnection level

This lack of focus on the internal reconfiguration of processing units for new set of data and applications results in the under utilization of processor's resources such as arithmetic operators, datapath, registers etc. Due to low precision data in multimedia applications, these traditional reconfiguration methods increase the complexity of interconnection network and do not effectively contribute too much to speed up the overall processor efficiency.

4.1.2 Reconfiguration at operator's level

In this chapter, coarse grain reconfigurable pipelined operator is proposed for multimedia applications. This operator increases the resource utilization through reconfiguration at both data size selection level (subword size) and at operation level. Reconfigurations at operator's level not only reduce the complexity of interconnection network but also increase the utilization of processor resources for multimedia applications. The operator can be reconfigured to operate on different data sizes through the use of subword parallelism (SWP). In SWP low precision pixel data (subwords) are packed into word size registers and the same operation is performed in parallel on all the packed subwords. This parallel execution of subwords increases the utilization of processor resources and thus improves the efficiency [30, 98]. Compared to existing methods [20, 28, 56], multimedia oriented pixel data sizes are considered rather than conventional data sizes.

4.1.3 Reconfigurability using SWP

SWP techniques has been carried out on the basic arithmetic operators (ADD, SUB, MULT etc.) to increase the performances of processors. These basic SWP operators perform parallel operations on subwords which are conveniently compatible with the word size of processor. By doing this, processor can achieve more parallelism rather than wasting the word size datapath and register sizes when operating on low precision data [16, 26]. Subwords can be of any size depending upon the application data. Conventionally the word size of processor is multiple of subword sizes which helps to reduce the complexity of SWP operators. For instance, some of the conventional subword sizes for 64-bit processors are 8, 16 and 32 bits.

These conventional subword sizes increase the performance of general purpose processors due to the availability of data which is either of one byte length or some multiple of byte length. Several basic SWP operators have already been proposed. In [28], SWP adder operator is proposed which performs addition of conventional subword size data. In [56] [19] [11] [20] efficient SWP enabled multiplier and MAC operators are presented for different platforms. All these basic SWP operators are designed for conventional subword

sizes. However in multimedia applications, the input data (pixels) for computations is 8, 10, 12 or sometimes 16-bits. These multimedia data sizes are not in coordination with existing processor's subword sizes resulting in the under utilization of processor resources.

4.2 SWP Reconfigurable multimedia operator

In the proposed reconfigurable SWP operator, multimedia oriented subword sizes are considered rather than conventional subword sizes. This operator can be reconfigured to operate on variety of multimedia oriented subword data sizes. Unlike a normal reconfigurable processor, no reconfiguration time is required for reconfiguring the operator to new subword size operation. At operation level the operator can be reconfigured to perform variety of multimedia oriented arithmetic operations on subword data. These operations include addition, subtraction, multiplication, absolute value, sum of absolute difference (SAD), sum of product etc. Without even increasing the complexity of interconnection network, reconfiguration is provided at both data size level and operation level. This reconfigurable operator can be used as a processing unit in any multimedia based pipelined processor. In this chapter, architecture of this reconfigurable operator is proposed and its performance is analyzed by implementing it on different target technologies.

4.2.1 Architecture of SWP Reconfigurable operator

To perform parallel computations on low precision multimedia oriented pixel data, a 40-bit reconfigurable operator has been designed. The architecture of this operator is shown in figure 4.2.

Reconfigurable SWP operator can perform operations on word size operands (40-bits) as well as on subwords (8 or 10 or 12 or 16-bits) packed in word size registers. Word size of 40-bits is chosen because it gives good efficiency/complexity trade off and ensures better resource utilization with different multimedia oriented pixel sizes. Control bits used to select the subword size are communicated to all the units which contain SWP capabilities. To clarify the schematic, these control bits are not shown in Figure 4.2. When selected subword size is 8-bits then each input is considered as five 8-bit subwords packed in a 40-bit word. Hence on each 8-bit configuration, the SWP operator performs five same 8-bit basic operations in parallel. Basic operations are addition, subtraction, absolute value and multiplication. For subword size of 10-bits, the SWP operator performs four 10-bit operations in parallel, and three 12-bit operations or two 16-bit operations for

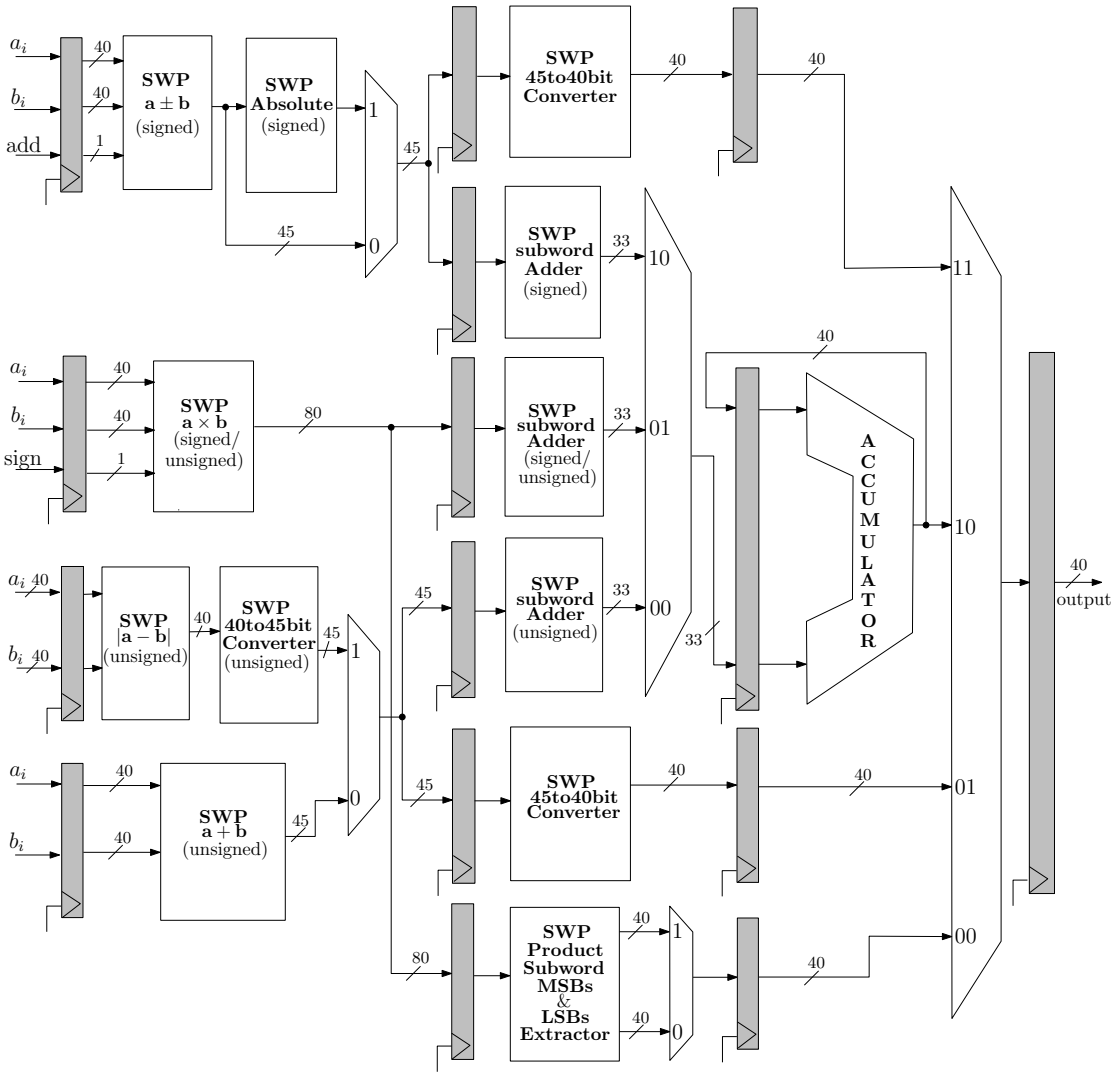

FIGURE 4.2: SWP reconfigurable multimedia operator

subword sizes of 12-bits or 16-bits respectively. In other words, the operator can be configured for both the computation it executes and the size of data. Based on the requirements, reconfigurable SWP operator can perform operations on signed as well as unsigned data formats and gives the results in required format. Based on the selected operation the output of reconfigurable operator can be in the form of subwords or single accumulated value.

4.2.2 Connectivity of reconfigurable operator with other operators

In SWP reconfigurable operator, the length of input and output data vectors is limited to word length (40-bits) to ensure better connectivity with other processor's operators. The input and output ports of SWP reconfigurable operator are shown in Figure 4.3.

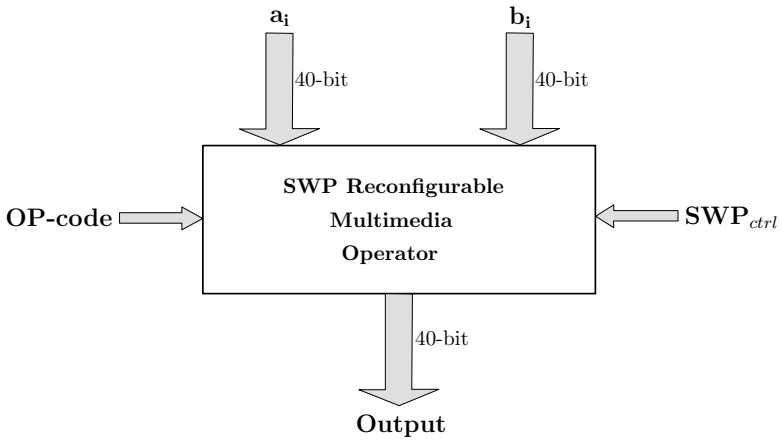

FIGURE 4.3: Inputs and outputs of SWP reconfigurable operator

As shown in Figure 4.3, the inputs and output of SWP reconfigurable operator consists of following signals.

- **Input vectors $\mathbf{a}_i$ and $\mathbf{b}_i$** These are two 40-bit input vectors which contains the data to be processed. Based on selected subword size, the SWP reconfigurable operator considers each input vector as either five 8-bit pixels or four 10-bit pixels or three 12-bit pixels or two 16-bit pixels.
- **Operation code (Op-code)** The *OP-code* contains the information about the operation which needs to be performed on input data. The control unit activates different arithmetic and interconnection units in accordance with selected *OP-code* to perform required operation.
- **Subword control signals (SWP_{ctrl})** These signals are used to select subword size in SWP reconfigurable operator. Based upon SWP_{ctrl} signals, the arithmetic operations are performed on either 8, 10, 12 or 16-bit subword sizes.
- **Output vector** The output of SWP reconfigurable operator consists of 40-bit vector which contains resultant subwords. However if the result of certain operations requires more than word data length then it can be obtained at output in multiple clock cycles.

To increase the operating frequency and throughput, the three stage pipelined architecture is used in the implementation of SWP reconfigurable operator. SWP operator gives high precision results with either no or minimum number of bit loss while performing different multimedia operations. For the operations which involve the recursive accumulation of results, overflow due to bit growth is avoided by allocating guard bits in accordance with 40-bit word length.

4.2.3 Building blocks of reconfigurable operator

The primary blocks for SWP reconfigurable operator consists of SWP arithmetic units. However for subword arrangement and alignment purpose, additional units are also required in the architecture of SWP reconfigurable operator. This reconfigurable operator is designed to perform basic as well as complex multimedia operations. Therefore to select appropriate inputs for different arithmetic units, interconnection elements are also required. Mainly the reconfigurable SWP operator consists of following building blocks.

- SWP basic arithmetic units
- MSBs and LSBs extraction units
- Accumulator unit
- Bit conversion units
- SWP subword adders units
- Multiplexer units

These units are connected in such a way that variety of multimedia oriented basic and complex SWP operations can be performed such as SAD for motion estimation algorithm, sum of products for DCT algorithm etc. SWP reconfigurable operator is designed in such a way that for performing any particular computation only the required blocks are activated by controller. This will reduce the unwanted switching activity at the ports, which ultimately reduce the overall power consumption of the operator.

4.3 Basic SWP arithmetic units

Basic SWP arithmetic units are the main processing elements in reconfigurable operator design. SWP capability is incorporated in these units with the minimum increase in area and speed overheads. These units can perform basic arithmetic operations on input data of different subword sizes and formats (signed/unsigned). These operations include SWP $(a \pm b)$ signed, SWP (abs) signed, SWP $(a \times b)$ signed/unsigned, SWP $|a - b|$ unsigned, SWP (a + b) unsigned. Word size 40-bit operations can also be performed by basic SWP arithmetic units. For complex multimedia operation combination of different basic SWP arithmetic units are used along with other glue logic.

4.3.1 SWP ADD and SUB units

SWP (a±b) signed operator is used to perform addition or subtraction of signed subword data. Addition or subtraction operation is selected with the input control signal. Subtraction is same as addition except in subtraction two's complement of operand which needs to be subtracted is taken before addition. Its architecture is based upon the SWP multimedia adder operator explained in section 2.2.2 of chapter 2. Based upon the selected subword size, internal carry control bits at different subword boundaries are made either '0' (breaking carry chain) or '1' (continuing carry chain). The adders between the carry control logic can be ripple carry adder RCA or carry look ahead adder CLA or Group CLA or carry save adder CSA etc. However for SWP designs group CLA gives better results compared to other adders. Most modern synthesis tools contain highly optimized adders which are used to meet design constraints. In our implementation, instead of implementing adder at gate level, efficient adders available in the library are used between the control logic and their results are found to be almost similar to Group CLA adders in SWP architectures. For subword size of 8-bits, *SWP (a±b)* unit performs five addition or subtraction operations. The output of each subword operation requires one extra bit to avoid any overflow resulting in an overall vector length of 45-bits internally. In case the reconfigurable operator is used for addition or subtraction operations only then 40-bit result can be obtained at the output by throwing away one bit from each resultant subwords obtained from *SWP (a ± b)* unit. The computation of *SWP (a ± b)* and its routing to operator's output is highlighted in Figure 4.4.

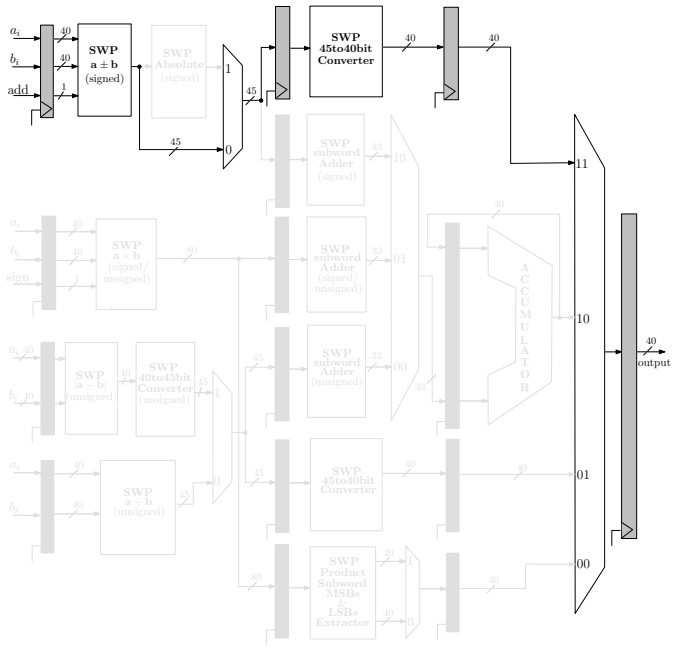

FIGURE 4.4: *SWP (a ± b)* operation using reconfigurable operator

The 45-bit output of *SWP* $(a \pm b)$ unit is reduced to 40-bit using *SWP 45 to 40-bit converter* unit (see section 4.4.1). By using the appropriate control signals, the *4 to 1 multiplexer* passes the resultant 40-bit vector to the output of operator. The lossless results can also be obtained at output at the cost of minor decrease in parallelism. For this purpose input pixels are bit extended and higher order subword size is selected for performing computations on lower size pixel data (e.g operations on 8-bit pixels using 10-bit subword size operator) to avoid any overflow. To preserve the accuracy of operations involving the accumulation of results, lossless 45-bit results from *SWP* $(a \pm b)$ unit are used inside the reconfigurable operator.

4.3.2 SWP Absolute signed

SWP Absolute signed unit is used to perform the absolute operation on signed subword data. The absolute operation gives the magnitude of signed number. If the number is positive then the absolute operation gives the same input as output. However, if the number is negative then the absolute operation gives the positive magnitude of the number at the output. In 2's complement numbers, the MSB of the number has negative weight. Therefore the operation performed by *SWP Absolute signed* unit depends upon the value of MSB of each subword. If the MSB of the subword is '1' then two's complement of corresponding subword is taken. However if the MSB of subword is '0' then the absolute operator will generate the same input at the output. This process is shown in Figure 4.5 for subword size of 8-bit.

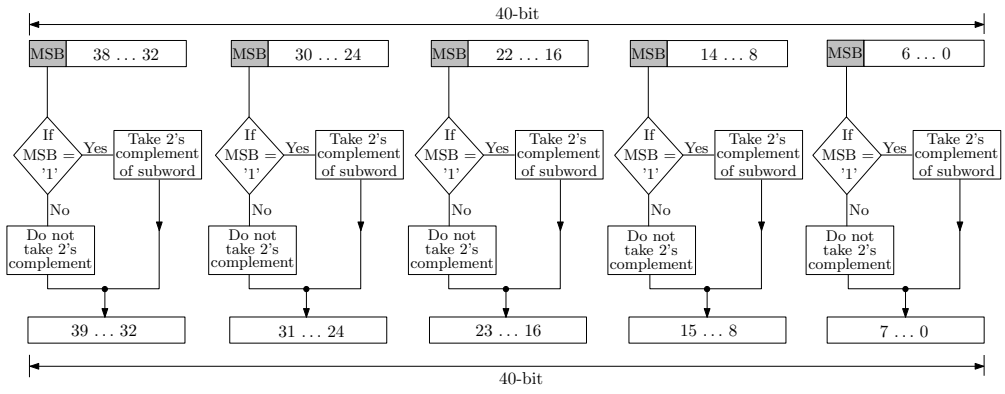

FIGURE 4.5: SWP absolute operation for signed subwords

The input and output subword data sizes of *SWP Absolute signed* unit are same. The resultant subwords from *SWP Absolute signed* unit can be used internally or it can be obtained directly at the output of reconfigurable operator.

4.3.3 SWP multiplier unit

SWP (a × b) signed/unsigned unit is used to perform SWP multiplication of signed as well as unsigned data. The detail architecture for the generation and addition of partial products (PPs) for this unit has already been explained in Section 2.3.2 (Multimedia SWP multiplier) of Chapter 2. The implementation of SWP multiplier unit is based upon an extension of SWP multiplier proposed in [56]. Compared to other SWP multipliers which are based on algorithms like Booth recoding etc., SWP multiplier proposed in [56] gives good efficiency as it does not require any detection and suppression of carries at subword boundaries. The SWP multiplier proposed in [56] supports only classical subword sizes (8, 16 and 32-bits etc.). However in our implementation of SWP multiplier for reconfigurable operator, the multimedia oriented subword sizes of 8, 10, 12 and 16-bits are considered which do not have any uniform arithmetic relation with word size (40-bit) of SWP operator. To reduce the hardware, PPs for different selection of subword sizes are generated using generalized PP generation unit. Bit inversions and addition of correction vectors are done based upon the selected multiplication type (signed/unsigned) and the subword size. On ASIC technology, compared to simple multiplier, the area and critical path overhead for incorporating multimedia oriented SWP capability in multiplier architecture is approximately 5% and 14% respectively.

- **SWP MSBs and LSBs extraction unit :** Like the inputs, the output data of all the basic SWP units can be represented by subwords packed in 40-bit registers except for *SWP (a × b) unit* whose output consists of subwords packed in 80-bit register. The packed subwords in 40-bit register can be obtained at the output of reconfigurable operator through the use of appropriate control bits. However the packed subwords in 80-bit register from *SWP (a × b) unit* can not be obtained at output at one time because output data length is limited to 40-bits. To overcome this limitation, 80-bit product is divided into 40-bit MSBs and LSBs parts. This function is performed by *SWP product subword MSB & LSB extractor* unit. Based upon the selected subword size this unit extracts MSBs and LSBs parts of 80-bit product subwords. For 8-bit subword size, this process is shown in Figure 4.6.

  For subword size of 8-bits, *SWP product subword extractor* unit extracts (71 . . . 64, 55 . . . 48, 39 . . . 32, 23 . . . 16, 7 . . . 0) and (79 . . . 72, 63 . . . 56, 47 . . . 40, 31 . . . 24, 15 . . . 8) 40-bit parts of product. Based on the requirement one of these parts is selected using multiplexer unit. Hence the complete product is obtained at the output of reconfigurable operator in the form of subword LSBs and MSBs in successive clock cycles. For other subword sizes the extraction of MSBs and LSBs parts are done accordingly. The *SWP product subword MSB & LSB extractor* unit is also useful when performing computations on fixed point numbers. Based on the

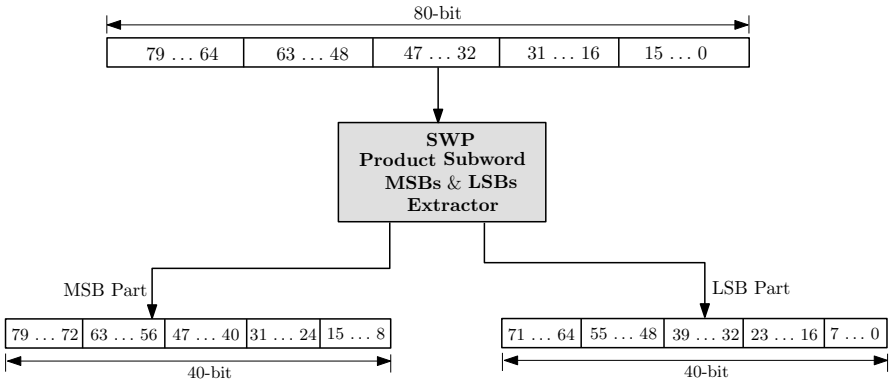

FIGURE 4.6: SWP product subword MSB and LSB part extractor

requirements, only MSB part of resultant subwords can be retained which would result in the reduction of precision. Similarly for high precision results in fixed point algorithms, LSB part of resultant subwords can be retained.

The arithmetic and interconnection units of SWP reconfigurable operator which are used to obtain the product subwords at the output are highlighted in Figure 4.7.

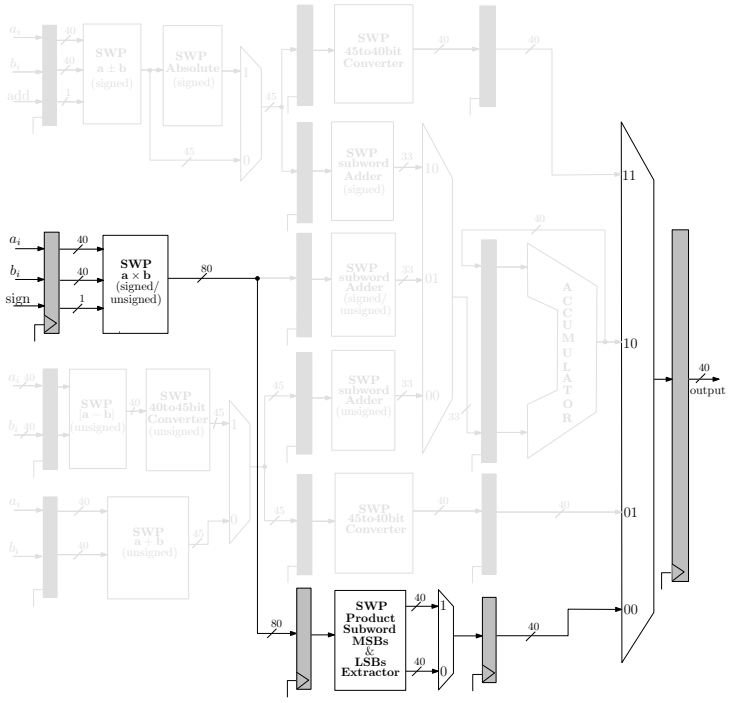

FIGURE 4.7: SWP multiplication using reconfigurable operator

As shown in Figure 4.7, the control unit activates all the highlighted units to perform multiplication operation. The *2 to 1 multiplexer* unit at the output of *SWP product subword MSBs & LSB extractor* unit is used to select either LSBs or MSBs part of product subwords. This 40-bit part is then routed to the output by using '00' control

signals for *4 to 1 multiplexer* unit. Therefore in successive clock cycles, whole 80-bit product subwords are obtained at the output of reconfigurable operator.

4.3.4 SWP $|a-b|$ unsigned

SWP $|a-b|$ *unsigned* unit is used to perform absolute difference of unsigned subwords. This unit is required because usually the pixels are stored as unsigned data. The basic architecture of this unit is based upon the *SWP absolute difference* unit explained in Section 3.2.3 of chapter 3 (Method 3). To avoid the absolute operation, this unit either calculates $a-b$ (when $a > b$) or $b-a$ (when $b > a$). One other reason to avoid the absolute operation is that the result of $(a-b)$ can be either positive or negative and in unsigned format it is not possible to represent negative number. Therefore *SWP* $|a-b|$ *unsigned* operation is obtained without actually implementing absolute calculation hardware. Due to subtraction operation bit overflow is not possible hence 40-bits are sufficient enough to store the subwords of any size. Similarly *SWP* $(a+b)$ *unsigned* unit is used to perform the addition of unsigned subwords. To avoid any internal overflow 45-bits are allocated to store resultant subwords from *SWP* $(a+b)$ *unsigned* unit.

4.3.5 Accumulator unit

This unit is used to accumulates recursively the outputs generated by basic arithmetic operators. Before giving to *accumulator* unit, the subwords are added by using *SWP subword adder* units. The input to *accumulator* unit is 33-bit number and the output of the *accumulator* is 40-bit number. The output is fed back for the recursive accumulation in each clock cycle. The number of values which can be accumulated without any precision loss depends upon the selected subword size. Smaller the subword size, higher will be accumulations without any precision loss.

4.4 Subword alignment and interconnection units

In addition to basic SWP arithmetic units, SWP reconfigurable operator also requires certain units for subword arrangements and alignments. These units arrange the subwords in proper order so that the desire computations can be performed in parallel. In order to share the arithmetic units for different multimedia operations, different interconnection units are used. These units include multiplexer units, register units etc. The controller generates appropriate control signals to activate the required units corresponding to each operation.

4.4.1 Bit conversion units

When working on subwords of data, it is required to align the subwords in the registers before any computation. For this purpose *Bit conversion units* are used in the architecture of SWP reconfigurable operator. These units either extend or contract the subwords packed in the registers. There are two types of bit conversion units which are used in SWP reconfigurable operator.

- **SWP 40 to 45-bit converter :** This unit expand the 40-bit input vector to 45-bit. It aligns the 40-bit output of $SWP\ |a-b|$ *unsigned* unit with 45-bit output of $SWP\ (a+b)$ *unsigned* unit. In this conversion each subword is expanded by one bit. As the input subwords are unsigned numbers, therefore each subword is expanded by '0' bit. If the subwords are signed 2's complement numbers then the MSB of each subword is bit extended instead of '0' padding. The number of expanded bits depends upon the number of subwords packed in the input register. For different selection of subword sizes, there are different numbers of subwords packed in the register. The process of subwords expansion for 8-bit unsigned subwords is shown in Figure 4.8.

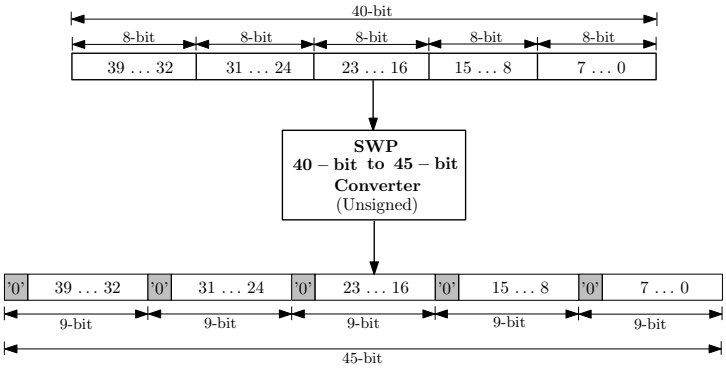

FIGURE 4.8: Expansion of unsigned subwords

As shown in Figure 4.8, for 8-bit subword size the 40-bit vector is expanded to 45-bit vector. The arrangement of expanded bits will be different for different selection of subword sizes. For 8, 10, 12 and 16-bit subwords, the number of expanded bits are 5, 4, 3 and 2 bits respectively. As the subword size increases the number of expanded bits reduces but the output vector length remains 45-bit. The unused bits are set to '0'.

- **SWP 45 to 40-bit converter :** This unit contracts the 45-bit input vector to 40-bit vector. Each subword within the input register is contracted by one bit. This contraction is required because the output of SWP reconfigurable operator

is 40-bit. Contraction from 45-bit to 40-bit is only required when the output of basic arithmetic units are directly required at the output. As a result of this contraction, there is a slight loss of precision. However this contraction is not required for the operations which perform the accumulation of results generated by basic arithmetic units. Therefore no precision will be lost for accumulation based operations. These operations include $\sum(a \pm b)$, $\sum(a \times b)$, $\sum|a - b|$ etc.

4.4.2 SWP subword adders units

The inputs to *SWP subword adder* unit are resultant subwords from different basic SWP arithmetic units. Based upon the selected subword size, the *SWP subword adder* unit separates the N_{sa} subwords x_i packed in the input register and then performs the addition of these subwords. The expression of the *SWP subword adder* output z_{sa} is equal to

$$z_{sa} = \sum_{i=0}^{N_{sa}-1} x_i \tag{4.1}$$

For alignment purpose before the addition, *SWP subword adder* unit performs either sign extension (signed subwords) or zero padding (unsigned subwords) of subwords depending upon the selected data format. The output of *SWP subword adder* unit consists of 33-bit. This 33-bit data length is selected based upon the worst case of 16-bit subword size for $\sum(a \times b)$ operation with no bit loss (For details see Section 4.5.1 of Chapter 3). For other subword sizes and operations, the data length requirements at the output of *SWP subword adder* units are less. There are three *SWP subword adders* units used in SWP reconfigurable operator design.

- **45-bit SWP subword adder (unsigned)** The input to this unit is 45-bit vector which contain unsigned packed subwords. These unsigned subwords are added and extended by zero padding to 33-bit output.
- **45-bit SWP subword adder (signed)** The input to this unit is 45-bit vector which contain signed packed subwords. These signed subwords are added and bit extended to 33-bit output.
- **80-bit SWP subword adder (signed)** This unit is used to add the product subwords generated by *(a × b)* unit. The input of this unit is 80-bit vector which contain product subwords. These subwords are added to generate 33-bit output.

4.4.3 Multiplexer units

Multiplexers are used to provide appropriate data to basic SWP arithmetic units. Based upon the operation need to be performed, controller unit generates appropriate control *(ctrl)* signals for the multiplexers and enable signals for registers. There are three types of multiplexer units which are used in SWP reconfigurable operator design.

- Two to one Multiplexer
- Three to one Multiplexer
- Four to one Multiplexer

As shown in Figure 4.2, three units of *2 to 1 multiplexers* , one unit of *3 to 1 multiplexer* and one unit of *4 to 1 multiplexer* are used. For performing any particular operation if the output of certain multiplexer unit is not required, it is disabled to reduce the switching activity.

4.5 Complex multimedia operations

For certain multimedia applications more complex operations are required. For instance the operation required in the computation of SAD is given by

$$\text{SAD} = \sum_{i=0}^{N-1} |a_i - b_i| \tag{4.2}$$

Similarly the multiplication-accumulation or dot product operation required in the computation of discrete cosine transform DCT algorithm is given by

$$\text{DOTP} = \sum_{i=0}^{N-1} (a_i \times b_i) \tag{4.3}$$

Rather than subwords which sometimes provide loss of bit, these operations produce lossless single accumulated value at the output of reconfigurable operator. When the accumulated value is small, it is either bit extended or zero padded to output data size of 40-bits. To perform these complex operations *SWP subword adder* units and *accumulator* unit are used in addition to basic SWP arithmetic units. As an example, consider the computation of SWP $\sum(a \times b)$, SWP $\sum |a - b|$ and SWP $\sum(a + b)$ with 8-bit subword size using SWP reconfigurable operator shown in Figure 4.2.

4.5.1 SWP $\sum(a \times b)$ operation

Sum of product $\sum(a \times b)$ operation can be performed on different subword size data using SWP reconfigurable multimedia operator shown in Figure 4.2. To perform $\sum(a \times b)$ operation, the controller unit generates signals to activate the required units. All the remaining units are disable to reduce the switching activity. The arithmetic units and other blocks which are activated to perform $\sum(a \times b)$ computations are highlighted in Figure 4.9.

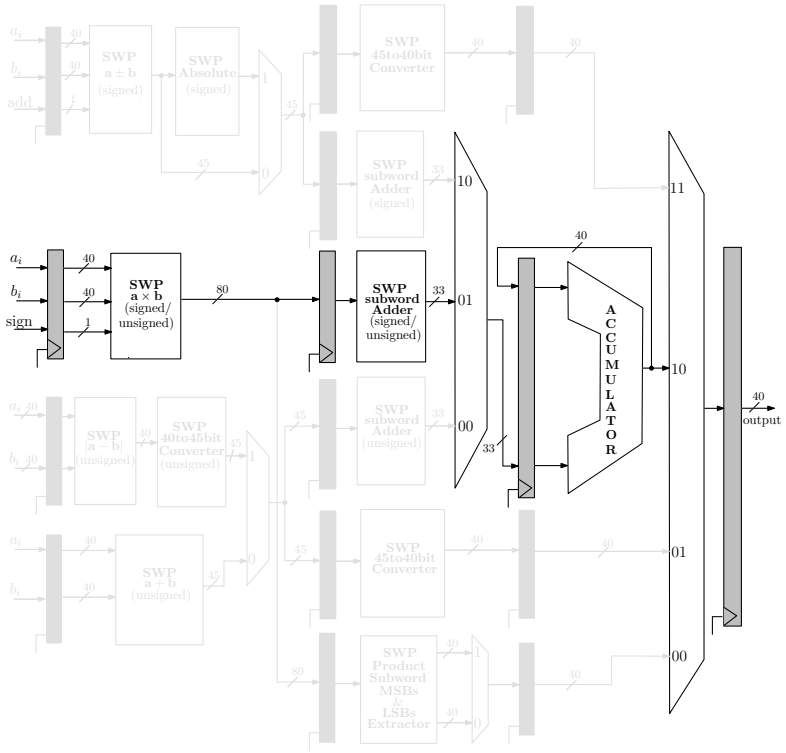

FIGURE 4.9: Computation of SWP $\sum(a \times b)$ using reconfigurable multimedia operator

The control unit generate appropriate signals for the selection of subword size and multiplication type (signed/ unsigned). In the beginning, SWP $(a \times b)$ unit produces 80-bit product value. This product value is used as input to a *SWP subword adder* unit. For 8-bit selected subword size, *SWP subword adder* unit considers the 80-bit input as five 16-bit subword products (without loss of bit) and adds them to generate a 33-bit value. This 33-bit value from *SWP subword adder* unit is obtained at the output of *3 to 1 multiplexer* unit using '01' select line signals. These select line signals are generated by controller unit for *3 to 1 multiplexer*. At each clock cycle the *accumulator* accumulates 33-bit value with the previous values to generate 40-bit $\sum(a \times b)$ term. This 40-bit $\sum(a \times b)$ term can be obtained at the output of the reconfigurable operator through *4*

to 1 multiplexer unit. The control unit generate '10' select line signals for this multiplexer unit. As the inputs to *accumulator* are single values instead of packed subwords, therefore SWP capability is not required for the *accumulator* unit implementation.

4.5.2 SWP $\sum(|a-b|)$ operation

SAD is commonly used operation in multimedia applications. SWP reconfigurable operator can be used to perform SAD operation. The arithmetic units and other hardware units which are activated for performing SAD computations are shown in Figure 4.10.

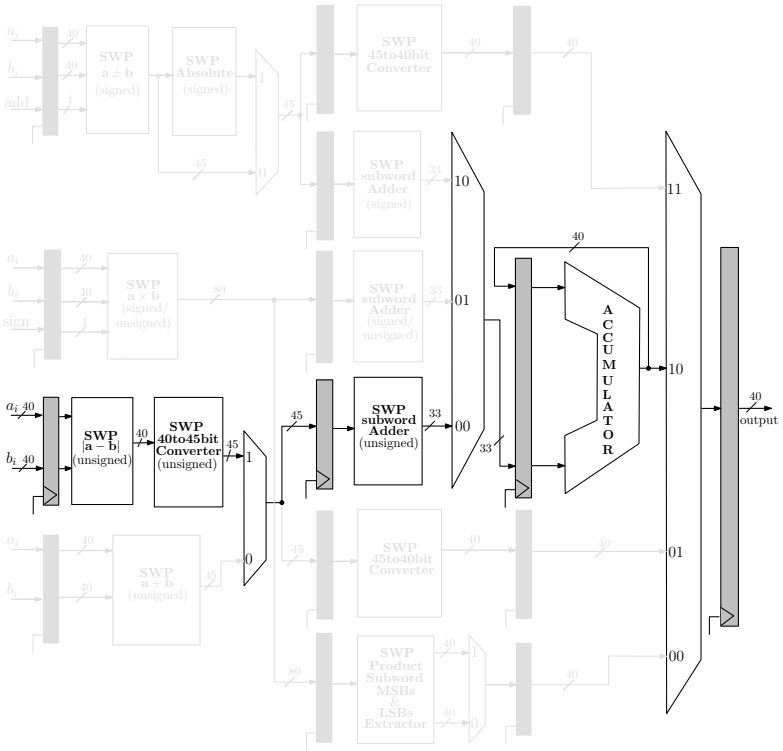

FIGURE 4.10: Computation of SWP $\sum |a-b|$ using reconfigurable multimedia operator

The highlighted units are activated by the controller for the computation of SAD. SWP $|a-b|$ unit produces the absolute difference for each packed subword. If the selected subword size is 8-bit, then the 40-bit output of SWP $|a-b|$ unit contains five 8-bit absolute value subwords. For alignment purpose, this 40-bit output is expanded to 45-bit using *40 to 45 bit converter*. This conversion aligns the subwords with the other input (from *SWP (a + b)* unit) of 45-bit multiplexer. The output of multiplexer is given to *SWP subword adder* unit in the next pipeline stage. *SWP subword adder* unit adds the subwords and generate 33-bit output. For 8-bit subword size, the actual requirement at the output of *SWP subword adder* unit is only 11 bits (addition of five 8-bit numbers). Remaining bits (33 - 11 = 22 bits) are all 0's and will be used as guard bits in next pipeline

stage. The 33-bit output from *SWP subword adder* unit is given to *3 to 1 multiplexer* unit. The controller generates '00' for the select lines of this multiplexer. The output of *3 to 1 multiplexer* is given to *Accumulator* unit for recursive accumulation. For final selection, the accumulated output is given to *4 to 1 multiplexer* unit. Controller selects the input corresponding to select line '10' as multiplexer output. As a result final SAD value is obtained at the output of this multiplexer. Effectively, for 8-bit subword size we can perform at least 2^{32} accumulations of absolute differences without any overflow.

4.5.3 SWP $\sum(a+b)$ signed operation

$\sum(a+b)$ operation is usually required to accumulate the pixel values in multimedia applications. In SWP reconfigurable operator, this operation can be performed on signed as well as on unsigned data. The arithmetic and interconnections units which are activated to perform SWP $\sum(a+b)$ operation on signed data are highlighted in Figure 4.11.

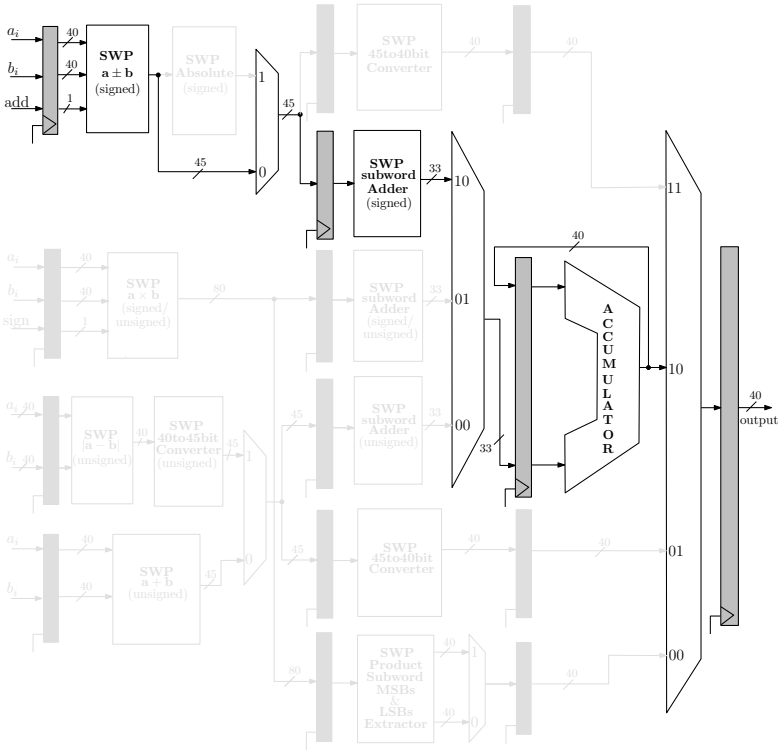

FIGURE 4.11: Computation of SWP $\sum(a+b)$ using reconfigurable multimedia operator

SWP $(a \pm b)$ unit is used to add the subwords packed in word size input registers. The controller unit generates the appropriate signals for the selection of subword size. To perform the addition operation, the 'add' signal is made '1' by control unit. For performing subtraction operation this 'add' signal is made '0' by controller. The output

of SWP $(a \pm b)$ unit consists of 45-bit. These 45 bits are allocated instead of 40 bits to avoid any overflow. For 8-bit subword size, 45 bits at the output of SWP $(a \pm b)$ unit contains five 9-bit (subword bits (8) + overflow bit (1) = 9-bit) subwords. These 45 bits are then given to *2 to 1 multiplexer* unit. The control unit generate '0' control signal for this multiplexer to obtain the result of *SWP (a + b)* operation at the output of MUX. In the next pipelined stage, the 45-bit output is given to *SWP subword adder (signed)* unit. This unit adds the packed subwords and generates 33-bit result. This 33-bit sum is obtained at the output of *3 to 1 multiplexer* using '10' control signals at select lines. In the next pipelined stage, the recursive accumulation of 33-bit results is done using *Accumulator* unit. The 40-bit output from the *Accumulator* unit is obtained at the output of *4 to 1 multiplexer* unit using '10' control signals for the select lines.

4.5.4 Other complex operations

In addition to basic arithmetic computations, SWP reconfigurable multimedia operator can perform variety of operations on pixel data. The results of these operations can be obtained at the output of operator using appropriate control signals. The complex SWP operations which can be performed using this reconfigurable operator are listed below:

- $\sum(a \times b)$ signed
- $\sum(a \times b)$ unsigned
- $\sum |a - b|$ signed
- $\sum |a - b|$ unsigned
- $\sum |a + b|$ signed
- $\sum(a + b)$ signed
- $\sum(a + b)$ unsigned
- $\sum(a - b)$ signed

For performing any these operations, the arithmetic operators, register units, multiplexer units and other datapath units are enabled by control unit. The control unit activates only those units which are required to perform certain operations in desire computation. Based upon the requirements, any combination of above mentioned operations can also be obtained such as

- $\sum(a \times b) + \sum |a - b|$ signed

- $\sum |a+b| + \sum |a-b|$ signed
- $\sum |a \times b| + \sum (a+b)$ unsigned
- etc.

For the complex operations which involve the accumulation of results generated by basic units, the output word-length depends upon the number of values needed to be accumulated. In the worst case when performing $\sum(a \times b)$ operation on 16-bit subwords, the output of the *SWP* $(a \times b)$ unit consists of two 32-bit subwords. As the *accumulator* is 40-bit wide, the extra eight bits are used as guard bits to avoid any overflow. Therefore the reconfigurable operator can perform at least 256 (2^8) accumulations of worst data length product terms. For other operations and smaller subword data sizes, the numbers of guard bits are greater and thus the number of accumulations which can be performed increases further without any overflow. For example in $\sum(a \times b)$ operation on 8-bit pixel sizes, the product subwords are 16-bit wide therefore the reconfigurable operator can perform (2^{24}) accumulations without any bit loss.

4.6 Synthesis results

To analyze the area, speed and power consumption, overall reconfigurable operator design is synthesized to ASIC standard cell 130nm and 90nm technology using Synopsys *Design Compiler* and to FPGA (Xilinx Virtex II) using Mentor Graphics *Precision RTL* tool. The area, speed and power consumption have been measured. Table 4.1 shows the results obtained for the two ASIC technologies.

Technology (CMOS)	**Clock period(ns)**	**NAND Gates**	**Power** (mW)	**Gates** $\times$ **CP** $\times$ **Power**
90 nm	6.0	29980	6.63	1192604
130 nm	10.0	31126	6.93	2157031

TABLE 4.1: Synthesis on ASIC technologies

To analyze the overall efficiency, product of gates, clock period (CP) and consumed power is also computed. Smaller value of this product term indicates higher efficiency. In order to get the best possible clock frequency for reconfigurable operator, synthesis are performed for different clock periods on each ASIC technology as shown in figure 4.12.

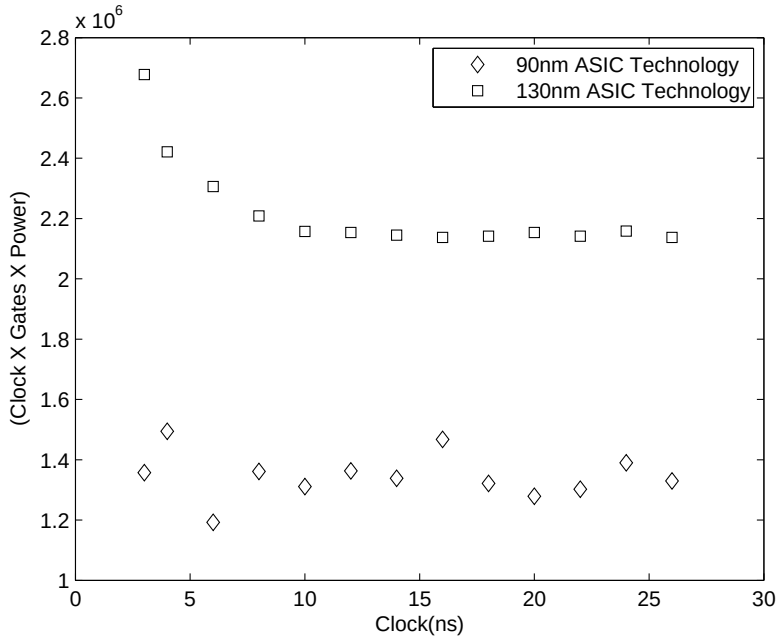

FIGURE 4.12: Synthesis at different clock periods

As shown in Figure 4.12, on 130nm ASIC technology, the product of gates, clock period and consumed power decreases with the increase in the clock period. After 10ns, the product term remains almost constant. Therefore on 130nm technology, clock period of 10ns gives good trade off between the efficiency and frequency of reconfigurable operator. On 90nm ASIC technology, the decrease in the product term is not very even with clock frequency. However the maximum efficiency (minimum product term) is obtained at the clock period of 6ns.

Table 4.1 shows the synthesis results for those clock frequencies which give highest efficiency (minimum gate×CP×power) on each target technology while meeting all the design constraints. However as per the requirements of processor, the reconfigurable operator can be used up to the minimum clock period of 3ns. On 90nm technology, despite of using lower clock period (6ns), the area consumed and power consumption are less compared to 130nm technology. After the analysis, it is found that the unit which consumes maximum design resources is the SWP multiplier. Although the multiplier architecture is based on [56] which is known to be much more efficient for SWP implementation than conventional multiplier architectures, it consumes almost 51% of total area and 40% of total power. The blocks like signed arithmetic basic units, unsigned arithmetic basic units and subword adder units consumes respectively 11%, 10%, 12% of total area and 12%, 20%, 12% of total power. Each of the remaning design blocks like bit conversion units, MSB and LSB extractor units, accumulator units and multiplexer units consumes almost less than 2% of total area and power. On the FPGA Virtex II

platform, 2800 CLBs are required and the critical path is equal to 17.4 ns. Actually area and speed overheads for implementing SWP capability are less on ASIC technology compared to FPGA technology. The reason is that in FPGA implementation resources are CLBs rather than gates as in ASIC. Therefore ASIC resources better suit the SWP designs.

4.6.1 Statistical power analysis

The power consumed by the reconfigurable operator will be different while performing different multimedia operations. This occurs because while performing any particular operation, only those units are enabled which are required to perform certain computations. All the remaining units are disabled to reduce the switching activity. Statistical power estimation is carried out to find out the power consumed by SWP reconfigurable operator while performing different operations on input data. This process is shown in Figure 4.13.

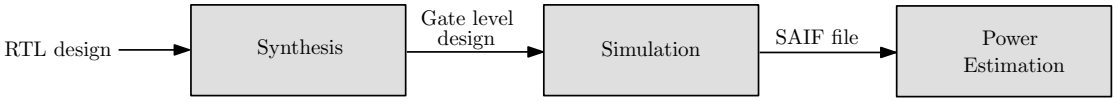

FIGURE 4.13: Statistical power estimation

Statistical power estimation gives accurate power consumption compared to the probabilistic power estimation. In the probabilistic power estimation, the power consumption is estimated on the basis of probability of switching activity at the nodes. Where as in statistical method the switching activity at each node is monitored while performing actual operations on test vectors. As shown in the Figure 4.13, the process of statistical power estimation can be divided into three main steps.

- **Step 1 : Generation of gate level net-list** In the first step gate level net-list is generated from RTL design file. This can be done by using any synthesis tool. In our experiments we have used *Design Compiler* tool from Synopsys for this purpose. The *Design Compiler* generates the gate level net list by using the gate components from its library.
- **Step 2 : Generation of switching activity file (SAIF)** In the second step, the switching activity file is generated by using the simulation tool. In our experiments we have used *ModelSim* simulation tool for this purpose. The gate level net list generated in the first step is given as input to *ModelSim*. The special libraries provided by Synopsys are also linked with simulator so that the simulator can recognize the gate level components used by Synopsys tool. Then the simulations

are performed using different test vectors and the switching activity of all the nodes at gate level are monitored and stored in a switching activity file (SAIF). In our experiments, the switching activity files corresponding to different operations performed by SWP reconfigurable operator on random vectors are generated and stored. These SAIF files are then used to estimate the power consumed by SWP reconfigurable operator while performing different multimedia operations.

- **Step 3 : Estimation of Power** In this step the switching activity file (SAIF) generated in Step 2 is given to *Design Compiler* tool for the estimation of statistical power. The *Design Compiler* tool estimates the power on the basis of switching activity at each node while performing different operations.

The percentage power consumed by SWP operator to perform different SWP operations on 130nm ASIC technology is shown in the Figure 4.14. To make the comparison fair, input data vectors and subword size remains same in the calculation of power for all operations. Subword size of 8-bits and clock period of 10ns are used during these experiments.

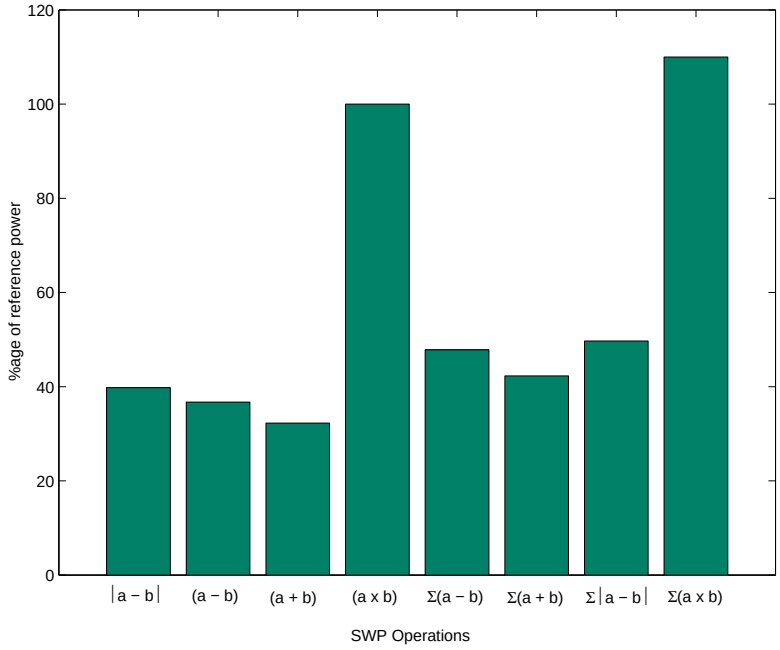

FIGURE 4.14: Power consumption of SWP operations

In Figure 4.14, power consumed by SWP multiplier operation is taken as reference (100%). All other operations consume some percentage of reference power. The power consumed by each operation depends upon the number and the type of units which are activated to perform certain computations. The computations which involve the multiplication of subwords consume more power compared to other operations. Obviously the power consumed by the complex operations which involve the accumulation

is more compared to the operations which involve only basic SWP arithmetic units. Additional power is mainly due to subword adder units. Maximum power consumed by reconfigurable operator is 110% of reference power when performing $\sum(a \times b)$ operation.

4.7 Performance on multimedia applications

Proposed reconfigurable operator can be used in any pipelined processor to enhance the performance especially for multimedia applications. At present time, reconfigurable operator has been synthesized, and performance assessment can be given at the operator level. The SAD kernel used in motion estimation and DCT kernel used in video compression algorithms are good candidates at this granularity level. For comparison, state-of-the-art Texas Instruments (TI) TMS320C64x DSP architecture is used. The processing unit of the TI DSP is made up of two clusters. Each cluster consists of four functional units along with one multiplier and two arithmetic and logic units. This architecture provides SWP capabilities based on 8, 16 and 32 bit data. For a fair comparison, one reconfigurable operator is considered for our processor and one cluster is considered for the TI DSP. For 16-bit pixels, the number of cycles N_{cycles} required to compute the SAD applied to 16 by 16 image blocks is 128 for both implementations. N_{cycles} is 60 and 64 for our operator and TI DSP based solution respectively when 8-bit pixels are considered. For 10 and 12 bit pixels, the granularity in term of data size of our operator allows the number of cycles to be reduced. N_{cycles} required to compute SAD of 16 by 16 image blocks are reduced by 50% and 25% for 10 and 12 bit pixels respectively. Similarly for the computation of DCT of 16 by 16 image block, N_{cycles} are reduced by 52% and 33% when considering the pixel sizes of 10 and 12 bits respectively. This occurs because of the better utilization of datapath and arithmetic units in our reconfigurable SWP operator for multimedia applications. Compared to each cluster of the DSP chip, the percentage reduction in number of cycles (N_{cycles}) of our reconfigurable operator on different multimedia kernels (SAD, DCT and discrete wavelet transform DWT [72]) when applied on a (16x16) image size is shown in Table 4.2.

	Pixel size (bits)			
	8	**10**	**12**	**16**
%age reduction for SAD	0%	50%	25%	0%
%age reduction for DCT	0%	52%	33%	0%
%age reduction for DWT	0%	58%	38%	0%

TABLE 4.2: Percentage reduction in number of cycles

In addition to computation cycles shown in Table 4.2, TI DSP also requires loop control cycles when performing different multimedia operations. However these loop control cycles are not considered. With our reconfigurable operator, no extra control cycle is required. In practice, processing is spread on two clusters with TI DSP so N_{cycles} is divided by two. In our case, as per the requirements several reconfigurable SWP operators can be used in the processor's design to further increase the efficiency through parallel processing.

4.8 Conclusion

Reconfigurability at data size level and at operation level in the arithmetic operator design improves the performance of processors for several multimedia and DSP applications. By introducing reconfigurability at this level, the complexity of interconnection network is reduced to high extent. The operator can reconfigure itself to perform different multimedia operations on different data sizes without any need of reconfiguration time. Support of subword sizes that are in coordination with pixel sizes in multimedia applications further enhance the performance though better resource utilization. When used in any multimedia processor, this reconfigurable operator provides speedup along with flexibility for multimedia applications.

Chapter 5

SWP using redundant representation

In the previous chapters the designing of different multimedia operators have been discussed. These operators are designed by using multimedia oriented subword parallelism (SWP) capability on binary number system. Although the binary number system is widely used number system but as far as the designing of arithmetic operators is concerned, it has speed limitations. The speed of binary based operators cannot be increased beyond certain limits. The reason for this speed limitation is the carry propagation in the binary based addition operation which subsequently effects other operations as well. To overcome this limitation and to increase the speed of different arithmetic units, operators are designed using redundant number system. Redundant number system ensures the carry propagation free addition which increases the speed of different arithmetic units. In this chapter different basic and multimedia operators are designed using redundant number system and their performances are compared with the binary operators. To enhance the performance of processor for multimedia applications, SWP is also used on redundant numbers for the designing of different operators. In SWP redundant operators, multimedia oriented subword sizes are considered rather than classical subword sizes. The contents of this chapter are based on our publication [51].

The rest of this chapter is organized as follows: Section 5.1 gives the overview of binary and redundant number system. The limitations of binary binary adders are also discussed. Section 5.2 describes the process of carry propagation free addition using redundant number system. Section 5.3 presents the architecture of logic cell used in the addition of redundant numbers. Redundant adders based on this logic cell are also compared with binary adders. Section 5.4 describes the procedure for the conversions between redundant and binary number system. The hardware resources required for

this process are also explained. Section 5.5 explain the multiplication of numbers using redundant number system. The comparison of redundant multiplier is done with binary multipliers to highlight the speed effects. Section 5.6 present the architecture of redundant adder using different resources. The use of each resource is controlled by using finite state machine (FSM) controller. Section 5.7 describe the implementation of redundant operators using multimedia oriented SWP capability and their performance are compared with SWP binary operators. Section 5.9 presents the use of SWP in different number conversion units. In Section 5.8, the architecture of a borrow save SWP SAD unit and its advantages over a conventional binary SWP SAD unit are also explained. The pipelined architecture of a coarse grain reconfigurable SWP operator using a redundant representation is presented in Section 5.10. The synthesis results of the proposed reconfigurable operator for multimedia processing and its performance compared to the state of the art DSP chips are also presented. Finally we conclude the chapter in Section Section 5.11.

5.1 Number systems

Number system provides the set of symbols by which the numerical value of number can be represented. There are different systems available to represent the numbers such as binary number system, octal number system, decimal number system, hexadecimal number system, redundant number system etc. These number systems have their own advantages in particular domain. For instance the decimal or base 10 number system is used in our calculations because of its ease in interpretation. On the other hand modern computer architectures are based upon binary number system due to the ease of computations using computer hardware. In the processor design, the efficiency of different arithmetic operators is highly based on the number system used. The internal architecture of operator designed for two different number systems will be different. In this section we will discuss the significance of binary and redundant number system in the context of addition operation. Addition is directly or indirectly used in almost all the arithmetic operations. Therefore any number system which gives the efficient implementation of addition process ultimately ensures the high performance of almost all the arithmetic operations.

5.1.1 Binary number system

In binary number system, the value of the number is represented using bits. In radix-2 each bit can take either 0 or 1 value. By using the combination of bits the value of any number can be represented. For instance decimal number 56 can be represented in

binary format with 111000 bits combination. Due to the simplicity of bit representation, the arithmetic operators like addition, subtraction, multiplication etc. are designed in hardware using binary number system. These operators give better efficiency while using minimum hardware resources. However there are certain limitations of conventional binary based arithmetic operators due to which the speed of these operators cannot increase beyond certain limits. Latter in this chapter we will use CB for conventional binary representation.

In conventional binary (CB) adders, the propagation of carry at any stage of addition is the major limitation to increase the speed of adder or subtractor. The carry signal has to propagate from least significant bit (LSB) to most significant (MSB). Therefore the delay due to carry propagation increases with the increase in input vector length. Different adder architecture based upon CB number system has been proposed to increase the speed of addition while limiting carry propagation to minimum level. These adders include carry look ahead adder (CLA), conditional sum adder (CSA) etc. These adder scheme tries to reduce the propagation of carry through different methods [97]. However in all these CB adders the propagation of carry remains there up to some extent at any stage of addition. For instance in CLA the computation of carry for each location is based upon the bit values at all the previous bit locations. Therefore to perform the addition at any bit location the adder has to wait for the computation of carry. This delay due to carry logic circuitry increases with the increase of vector length. Similarly in conditional sum adder, to break up the carry chain the input vectors to be added are divided into small blocks of 4 or 8 bits etc. These blocks are added in parallel for '0' as well as for '1' carry input. Based upon the actual carry output of each block, the final sum of next block is selected. So the carry chain has been broken but still the delay due to carry chain within the blocks remains exist. Therefore it can be concluded that in the adder using CB number system, it is impossible to totally eliminate the effect of carry propagation. However the carry propagation can be reduced to some extent using different techniques.

5.1.2 Redundant number system

To overcome carry propagation problem in the adder architectures, we have to switch to another number system which is called redundant number system. Redundant or barrow save number system allows to perform the addition with maximum one digit carry propagation irrespective of size of input vectors. Latter in this paper we will use BS for borrow save or redundant number system. In barrow save (BS) number system, the numbers are represented by digits rather than bits. In conventional radix-r number system each digit can take r possible values ranging from 0 to r-1. For instance in radix-2

binary number system each bit can take one of the two possible values that are either 0 or 1. Similarly for radix-10 decimal number system each digit can take any value from range 0 to 9. However in radix-r BS number system each digit can take more than r values. One of the most famous BS number system is signed digit number system [8]. In signed digit BS system each digit can take either positive or negative digit values. In radix-r signed digit BS number system each digit D can take values from the range given in Equation 5.1.

$$-a \leq D \leq +a \qquad \text{where} \qquad \lceil \frac{r-1}{2} \rceil \leq a \leq r-1 \tag{5.1}$$

By using different values of r, the digit set for radix-r sign digit BS numbers can be obtained. For instance in radix-2 BS representation [25, 37], numbers are represented using digits from the digit set $\{-1, 0, 1\}$. Digit -1 is denoted by $\overline{1}$ in this chapter. Each number is represented by the combination of any of these digits. For example $\{1\}_{-CB}$ can be represented by digit combinations $\{01\}_{-BS}$ or $\{1\overline{1}\}_{-BS}$. The numerical value of any number x in BS representation can be calculated using Equation 5.2.

$$\sum_{i=0}^{n-1} x_i \times 2^i \qquad \text{where} \qquad x_i \in \{-1, 0, 1\} \tag{5.2}$$

Using the BS representation, some numbers have several representations; this is a *redundant* number system. For instance the number 9, denoted by $\{1001\}_{-CB}$ in CB, has several BS representations: $\{101\overline{1}\}_{-BS}$ and $\{11\overline{1}\overline{1}\}_{-BS}$ and $\{1001\}_{-BS}$. This redundancy in representing the same number using different combinations of signed digits provides ground for carry propagation free addition [45]. Using BS number system the addition of digits at any rank depends upon the input digits at the same rank and the input digits at the one previous rank only. Therefore addition of two numbers can be performed in constant time irrespective of size of vectors to be added [46, 95]. As the addition is involve in almost all the arithmetic operations like subtraction, multiplication, division etc. Therefore using carry propagation free adders, high speed arithmetic operations can be performed. Redundant number system is different from residue number system which is commonly used in cryptography algorithms. In residue number system, the input numbers are broken into smaller numbers. These smaller factors of bigger numbers are then used in arithmetic operations to increase the speed. Operating on smaller number reduce the propagation of carries to certain extend. Whereas in redundant number systems focus is made to eliminate the complete propagation of carry.

5.2 Addition using BS number system

To overcome the delay due to the carry logic, a carry propagation free addition algorithm is used in our proposed multimedia operator. Carry propagation free adders perform the addition on data represented in the BS format rather than the CB format. In BS numbers system, the addition is done in two steps. In the first step the input vectors are added to generate intermediate sum $s_i \in \{-1, 0, 1\}$ and intermediate carry $c_i \in \{-1, 0, 1\}$ signals. These intermediate sum and carry signals are arranged as $\{s_i, 2c_i\}$ that is carry left shifted by one rank. In the second step s_i and c_{i-1} are added to generate final sum $z_i \in \{-1, 0, 1\}$. The intermediate sum and the carry signals are generated in such a way that the final addition of s_i and c_{i-1} do not generate any carry. This process is shown in Figure 5.1 for the addition of 8-digit BS numbers.

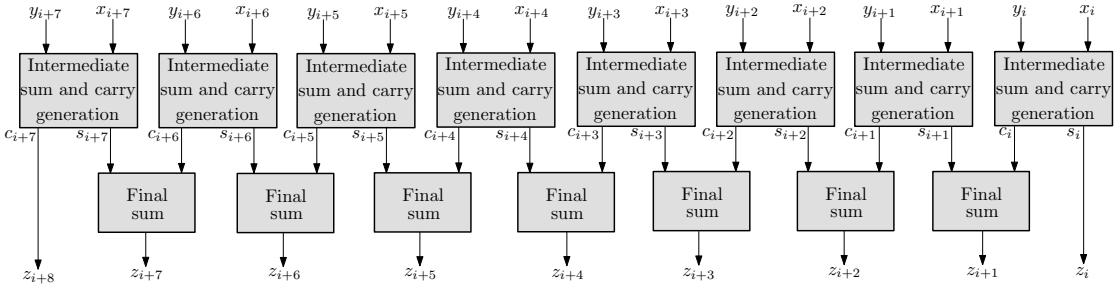

FIGURE 5.1: Block diagram of BS number addition

At any rank i, the intermediate sum s_i and intermediate carry c_i signals are generated based upon the input BS digit values at rank i and at the neighbouring lower order rank i-1. As shown in Figure 5.1 the intermediate sum and carry generator at all the ranks works in parallel without waiting for any carry from previous location. The final adders also operate in parallel without the generation or propagation of carry. The intermediate sum and carry signals are generated with the help of addition tables.

5.2.1 Addition tables for BS numbers

An operation is defined by the table which contains all the possible combinations of inputs and the corresponding outputs. For CB numbers the truth table of full adder is a typical example of addition table for the addition of two input bits and a carry bit. However in BS adders inputs are digits rather than bits. So the addition table is constructed to generate intermediate sum s_i and carry c_i digits corresponding to different combinations of input digits. There are several addition table available for BS digit addition [90] [89] [46] but we will discuss only two to highlight their importance.

5.2.1.1 Addition table using direct method

Let x_i and y_i be the two digits which needs to be added. x_{i-1} and y_{i-1} are the two input digits at next lower rank position i-1. s_i and c_i are the intermediate sum and carry signals generated at rank i respectively. Table 5.1 shows the addition table for intermediate sum and carry signals generated against different input digit values.

Type	$\mathbf{x_i}$ (digit)	$\mathbf{y_i}$ (digit)	($\mathbf{x_{i-1}}$, $\mathbf{y_{i-1}}$) (digits)	$\mathbf{c_i}$ (digit)	$\mathbf{s_i}$ (digit)
1	1	1	Dont care	1	0
2	1	0	Both are non negative	1	$\bar{1}$
	0	1	Otherwise	0	1
3	0	0	Dont care	0	0
4	1	1	Dont care		
	$\bar{1}$	1			
5	0	$\bar{1}$	Both are non negative	0	$\bar{1}$
	$\bar{1}$	0	Otherwise	$\bar{1}$	1
6	$\bar{1}$	$\bar{1}$	Dont care	$\bar{1}$	0

TABLE 5.1: Addition table for the BS numbers

All possible combinations of input digits are covered in Table 5.1. Intermediate signals are generated in such a way that both s_i and carry c_{i-1} signals can never be 1 or $\bar{1}$ at the same time. When one of intermediate signal s_i or c_{i-1} is 1 then the options left for other intermediate signal are either 0 or $\bar{1}$. Similarly when one of intermediate signal s_i or c_{i-1} is $\bar{1}$ then the options left for other intermediate signal are either 0 or 1. If the intermediate sum and carry signals are generated on this principle then the addition of s_i and c_{i-1} in the second step never generates carry and hence no propagation occurs. In fact the realization of this principle is possible only by using BS number system in which the same number can be represented by different combinations of digits. So whenever there is the violation of above principle, the redundancy property is used to represent the same number by some other combination of digits which fulfils the principle. Latter in this discussion we will call this principle as *principle of carry propagation free addition*. Table 5.1 uses the same principle and ensures no generation of carry. The intermediate sum s_i and carry c_i signals are generated on the basis of x_i and y_i inputs. However when there is the probability of violation of *principle of carry propagation free addition* then the input digits at next lower rank x_{i-1} and y_{i-1} are also taken into account for the determination of intermediate sum s_i and carry c_i signals. On this basis the input digit combinations in Table 5.1 can be divided in six different types.

- **Type 1, 3, 4, 6 :** In all these types the digit value of input signals x_i and y_i are such there is no probability of violation of *principle of carry propagation free addition* because s_i is equal to '0'. Therefore in these types, the intermediate

output signals $\{c_i, s_i\}$ are generated without taking into account the input digit values at next lower rank (x_{i-1} and y_{i-1}).

- **Type 2 :** In Type 2, one of input digit x_i or y_i is 1 and other is 0. In this case there is the danger of violation of *principle of carry propagation free addition* because s_i is equal to '1'. Therefore to avoid this danger the input digit values x_{i-1} and y_{i-1} at next lower rank are also taken into account for the generation of intermediate sum s_i and carry c_i digits. The input digits at x_{i-1} and y_{i-1} are analyzed to determine whether the carry will be generated at (i-1) rank or not. If there is a possibility of carry generation at (i-1) rank then we will use redundancy to represent the intermediate sum (s_i) and carry (c_i) digits so that the carry propagation free addition can be obtained. As in BS numbers system 1 can be represented either by digits $\{01\}_{-BS}$ or by $\{1\bar{1}\}_{-BS}$. When there is a possibility of positive carry (digit 1) generation at rank (i-1) then the intermediate output signals (c_i, s_i) takes values $(1, \bar{1})$. Positive carry at (i-1) rank will be generated if both x(i-1) and y(i-1) are 1 or one of them is 1 and other is 0. When there is a possibility of negative carry ($\bar{1}$) generation at rank (i-1) then the intermediate output signals (c_i, s_i) takes values $(0, 1)$. Negative carry at (i-1) rank will be generated if both x(i-1) and y(i-1) are $\bar{1}$ or one of them is $\bar{1}$ and other is 0. If there is no possibility of carry generation at rank (i-1) then intermediate output signals (c_i, s_i) can take either values $(1, \bar{1})$ or $(0, 1)$.

- **Type 5 :** The input digits in this type are almost of same nature as Type 2 except that the values of input digits are different in both types. In this type one of input digit x_i or y_i is $\bar{1}$ and other is 0. So there is also a danger of violation of *principle of carry propagation free addition* because s_i is equal to $\bar{1}$. To avoid this danger the input digits value at the next lower rank (x_{i-1} and y_{i-1}) are also taken into account for the generation of intermediate sum (s_i) and carry (c_i) digits. When there is a possibility of positive carry (digit 1) generation at rank (i-1) then the intermediate output signals (c_i, s_i) takes values $(0, \bar{1})$. When there is a possibility of negative carry ($\bar{1}$) generation at rank (i-1) then the intermediate output signals (c_i, s_i) takes values $(\bar{1}, 1)$. If there is no possibility of carry generation at rank (i-1) then intermediate output signals (c_i, s_i) can take either values $(0, \bar{1})$ or $(\bar{1}, 1)$.

Therefore using BS number system, the intermediate sum and carry signal (c_i, s_i) depends only on the inputs x_i, y_i and x_{i-1}, y_{i-1} digit values. The intermediate sum and carry signals at each rank can be generated in parallel.

5.2.1.2 Addition table using internal barrow

This is the another type of addition table used for the addition of BS numbers [90]. The end objectives of this addition table are also same, that is to perform carry propagation free addition. In this addition table internal barrow signal is generated before the generation of intermediate sum and carry as shown in Table 5.2.

Inputs		**Internal barrows**		**Outputs**	
$\mathbf{x_i}$ (digit)	$\mathbf{y_i}$ (digit)	$\mathbf{b_{i+1}}$ (digit)	$\mathbf{b_i}$ (digit)	$\mathbf{c_{i+1}}$ (digit)	$\mathbf{s_i}$ (digit)
$\bar{1}$	$\bar{1}$	$\bar{1}$	$\bar{1}$	0	$\bar{1}$
$\bar{1}$	$\bar{1}$	$\bar{1}$	0	0	0
$\bar{1}$	0	$\bar{1}$	$\bar{1}$	0	0
$\bar{1}$	0	$\bar{1}$	0	1	$\bar{1}$
0	0	0	$\bar{1}$	0	$\bar{1}$
0	0	0	0	0	0
1	$\bar{1}$	$\bar{1}$	$\bar{1}$	1	$\bar{1}$
1	$\bar{1}$	$\bar{1}$	0	1	0
1	0	0	$\bar{1}$	0	0
1	0	0	0	1	$\bar{1}$
1	1	0	$\bar{1}$	1	$\bar{1}$
1	1	0	0	1	0

TABLE 5.2: Addition table for the BS numbers using internal barrow

In Table 5.2, intermediate sum (s_i) and carry (c_i) signals are generated in such a way that the intermediate sum (s_i) signal is restricted to $\{0, \bar{1}\}$ digits and intermediate carry (c_i) signal is restricted to $\{0, 1\}$. This restriction is equivalent to the condition in the previous addition table (section 5.2.1.1) that s_i and c_{i-1} can never be both 1 or $\bar{1}$ at the same time. Due to this restriction, no carry will be generated when the addition of intermediate sum and carry is done. In this addition table intermediate sum and carry signals are obtained in two steps. In the first step barrow signal is generated. The barrow signal at (i+1) rank depends upon the input digit values at ith rank. Barrow signal b_{i+1} is generated on the basis of x_i, y_i and b_i signals. Similarly the intermediate sum s_i and carry c_{i+1} are also generated on the basis of x_i, y_i and b_i signals. The advantage of using this addition table is that single bit value can be used internally to represent intermediate sum and carry signals as the intermediate sum and carry signal are restricted to $0, \bar{1}$ and $0, 1$ digits respectively.

5.2.2 Addition of intermediate sum and carry digits

Using any of the addition tables discussed in sections 5.2.1.1 and 5.2.1.2, the intermediate sum (s_i) and carry (c_i) digits can be generated. After the generation, the intermediate sum (s_i) and carry (c_i) digits vectors are arranged in two rows. This arrangement is

done in such a way that carry signal c_i is left shifted by one rank resulting in $\{2c_i, s_i\}$. After this arrangement the intermediate sum (s_i) and carry (c_{i-1}) are added at each rank to obtain final sum vector. If intermediate sum (s_i) and carry (c_i) are generated using any of the addition tables shown in Table 5.1 and Table 5.2 then no carry will be generated in this final addition process. The addition of BS numbers using addition Table 5.1 is shown in Figure 5.2.

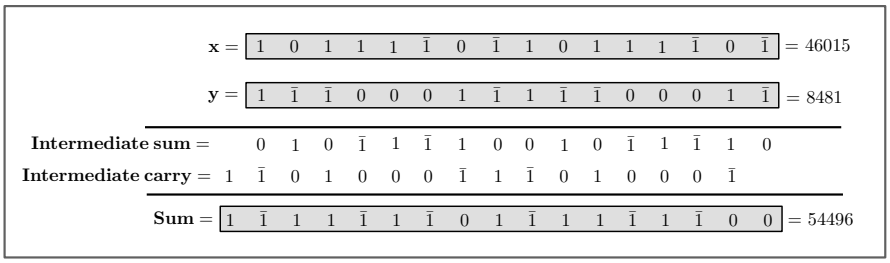

FIGURE 5.2: Addition of BS numbers using addition Table 5.1

The addition of same BS numbers can be performed using addition Table 5.2. This addition is shown in Figure 5.3.

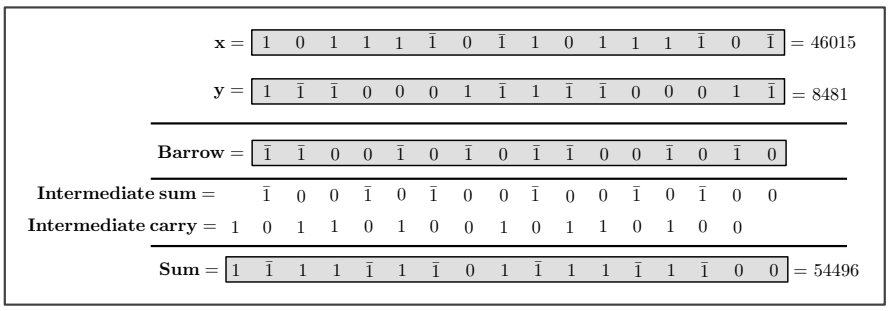

FIGURE 5.3: Addition of BS numbers using addition Table 5.2

As shown in Figure 5.2 and 5.3, in both cases no carry is generated in the addition of intermediate sum and carry vectors. Although the values of intermediate sum and carry digits are different for two addition tables but the final sum is same. Like the generation of intermediate sum and carries, the addition of s_i and c_{i-1} can also be done in parallel at all ranks. Therefore by using BS adder scheme we can perform parallel addition at all the ranks without any propagation of carry.

5.3 Logic cell for BS digits addition

The addition process explained in Section 5.2 can be realized in hardware using logic cell for BS digit addition. This logic cell operates in the same manner as the full adder cell works in CB number system. The inputs to full adder are two input bits at rank i

and carry bit from previous rank i-1. The full adder generates sum and output carry. In the logic cell for BS numbers, the inputs are two input digits which needs to be added [89]. Internally the logic cell at any rank uses digits from the one lower order rank only. The output of the logic cell is the final sum digit. For the addition of two BS numbers, separate logic cell is used at each rank. The architecture of logic cell for BS addition depends upon two main factors.

- **Addition table :** The basic architecture of any addition logic cell is based upon the addition table used. In case of BS addition there are many addition tables available [90] [89] [46]. Although these addition tables generate same final sum but the architecture of logic cells based upon these addition tables are different. Each of these addition cells takes different hardware resources. The logic cell explain in this section for the addition of two BS digits is based upon the addition table shown in Table 5.1.
- **Binary encoding of BS digits :** The architecture of logic cell also depends upon the encoding scheme used to represent digits in bit format. Radix-2 BS digit set consists of $\{\bar{1}, 0, 1\}$ digits. In binary based computers architectures, only bits 0 and 1 are understandable. Therefore before performing certain operations, we have to encode BS digits in bits. Each digit in the set $\{\bar{1}, 0, 1\}$ is encoded by two bits that are most significant bit (MSB) and least significant bit (LSB). For the encoding of BS digit x_i, the encoded MSB and LSB binary bits are represented by x_{i+} and x_{i-} respectively. One possible way of encoding digit set $\{\bar{1}, 0, 1\}$ in binary bits is $\{10, 00, 11\}$. Other encodings of digits in binary bits are also possible. However we will use this encoding scheme in the logic cell explained here.

Figure 5.4 shows the logic cell for the addition of BS digits [89]. The inputs of the logic cell consists of BS digits x_i and y_i. The BS digit x_i is represented in bit encoded form as x_{i+} and x_{i-}. Similarly the input BS digit y_i is represented in bit encoded form as y_{i+} and y_{i-}. Bits $\overline{x_{i+}}$ and $\overline{y_{i+}}$ are inverted versions of bits x_{i+} and y_{i+} respectively. The signals p_i and u_i are internal signals generated by the logic cell at each rank and are utilized by the logic cell at the next higher rank. For instance p_{i-1} and u_{i-1} are the signals generated by logic cell at rank i-1 and are utilized by logic cell at rank i. These internal signals are different from the conventional carry signal in binary addition which ripples through out the whole input vector. The output of logic cell is the sum digit z_i at rank i. This sum digit is encoded in bit form as z_{i+} and z_{i-}. Equations corresponding to different signals used in the architecture of logic cell are given below.

$$x_{id} = x_{i+} + x_{i-}$$

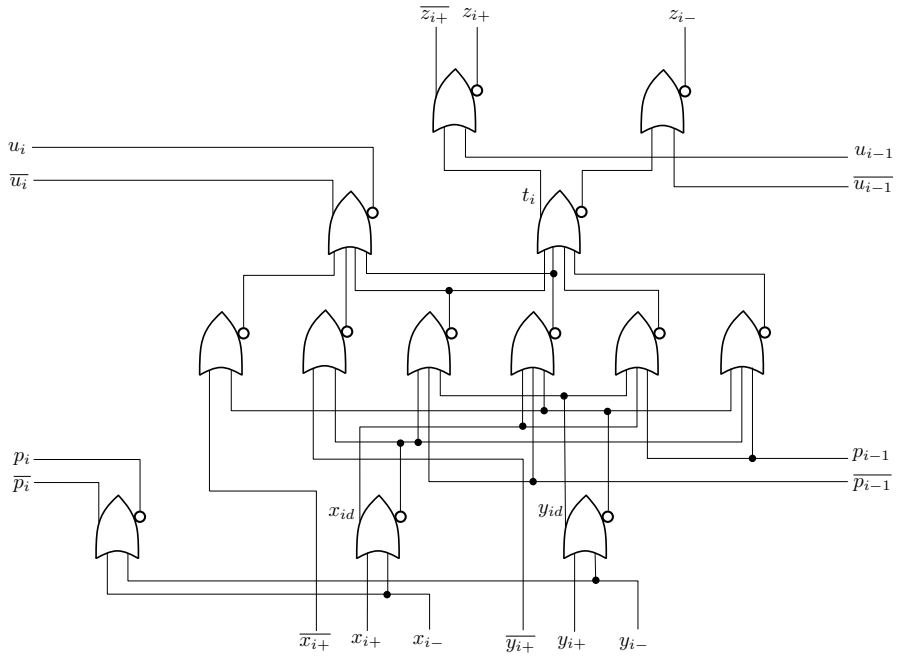

FIGURE 5.4: Logic cell for BS digits addition

$$y_{id} = y_{i+} + y_{i-}$$

$$p_i = \overline{x_i} \cdot \overline{y_i}$$

$$u_i = \overline{x_{id} \cdot \overline{y_{id}} \cdot p_{i-1} + \overline{x_{id}} \cdot y_{id} \cdot p_{i-1} + x_{i+} \cdot y_{id} + x_{id} \cdot y_{i+}}$$

$$t_i = x_{id} \cdot \overline{y_{id}} \cdot p_{i-1} + \overline{x_{id}} \cdot y_{id} \cdot p_{i-1} + x_{id} \cdot y_{id} \cdot \overline{p_{i-1}} + \overline{x_{id}} \cdot \overline{y_{id}} \cdot \overline{p_{i-1}}$$

$$z_{i+} = \overline{t_i} \cdot \overline{u_{i-i}}$$

$$z_{i-} = t_i \cdot u_{i-i}$$

The logic cell shown in Figure 5.4 consists of 13 OR/NOR gates. The output generated by this logic cell depends only on the two input digits and the internal signals generated by the logic cell at next lower rank. Therefore by using BS logic cell addition of digits at each rank is done in constant time. As the addition of the digits at all the ranks is performed in parallel therefore the complete addition of BS numbers is also obtained in constant time irrespective of word length. However this speed does not come without any cost. The complexity of logic cell shown in Figure 5.4 is more than full adder cell used in CB number system. Due to this complexity the implementation area of BS logic cell is more than CB full adder cell. Moreover the encoding of BS digits in bit format requires more number of bits than the CB representation. This increase in bits ultimately increases the implementation area and other hardware resources. Therefore the carry propagation free addition using BS representation is obtained at the cost of

increase in some hardware. But the speed of processing units plays such a vital role in the overall performance that most of the times these overheads are acceptable.

5.3.1 Adder using BS logic cell

Logic cell shown in Figure 5.4 is used to implement BS adders of different data sizes. The inputs to the BS adder are the two BS numbers which needs to be added. The adder length of 40, 64 and 128-digit are selected for implementation. The output of each BS adder is the sum vector of corresponding digit length. After the implementation, the area, critical path (CP) and power consumption of three different size BS adders will be analyzed.

Data width (digits)	**90nm CMOS ASIC**				**130nm CMOS ASIC**				**FPGA VirtexII**		
	Nand Gates	CP (ns)	Power (mW)	Gates X CP X Power	Nand Gates	CP (ns)	Power (mW)	Gates X CP X Power	CLBs	CP (ns)	CLBs X CP
40	589	0.26	1.5	230	689	0.67	2.4	1108	98	6.2	608
64	949	0.26	2.3	568	1113	0.67	4.0	2983	158	6.2	980
128	1910	0.26	4.7	2334	2244	0.67	8.1	12178	318	6.2	1972

TABLE 5.3: Synthesis results of BS adders

Table 5.3 shows the synthesis results of the BS adders on different target technologies. As the length of input vectors increases from 40-digit to 128-digit the area of BS adder also increases accordingly. The reason for this increase is obvious that for larger width adders more BS logic cells are required and hence more area will be consumed. The probabilistic power consumption of BS adders on two ASIC technologies also increases with the increase of data width. This increase in power is related to the number of gates. Therefore as the number of gates increases the power consumption also increases. The most interesting part of these synthesis results are the critical path (CP) of different width adders. On all target technologies, the CP remains same irrespective of the width of input vectors. For instance on 90nm ASIC technology, the CP remains 0.26ns for 40-digit, 64-digit and 128-digit adder. Similarly on other technologies the CP also remains constant. The CP remains constant even the length of adder is further increased. The reason for this constant CP is the use of BS logic cell which perform the addition in constant time irrespective of the size of input vectors.

5.3.2 Comparison of BS adder with other adder types

To elaborate the advantages of using BS adders, the performance of BS adder is compared with other conventional binary (CB) adders. There are different types of CB adders

available but only three adders are selected to compare with BS adder. These CB adders are ripple carry adder (RCA), Group carry look ahead adder (CLA) and the adder using the synthesis tool [97]. RCA and Group CLA are most familiar type of adders in CB number system. Group CLA implements CLA in the groups of 8-bit to reduce the carry logic complexity. The adder using the synthesis tool is the optimized CB adder available in the library of synthesis tool. Based upon the requirement synthesis tool uses these adders to meet the design constraints. In this comparison, CB and BS adders are used to add input vectors of 40 bits/digits. In the 40-digit BS adder, logic cell shown in Figure 5.4 is used for addition. Figure 5.5 shows the area, CP and power comparison of BS adder with CB adders.

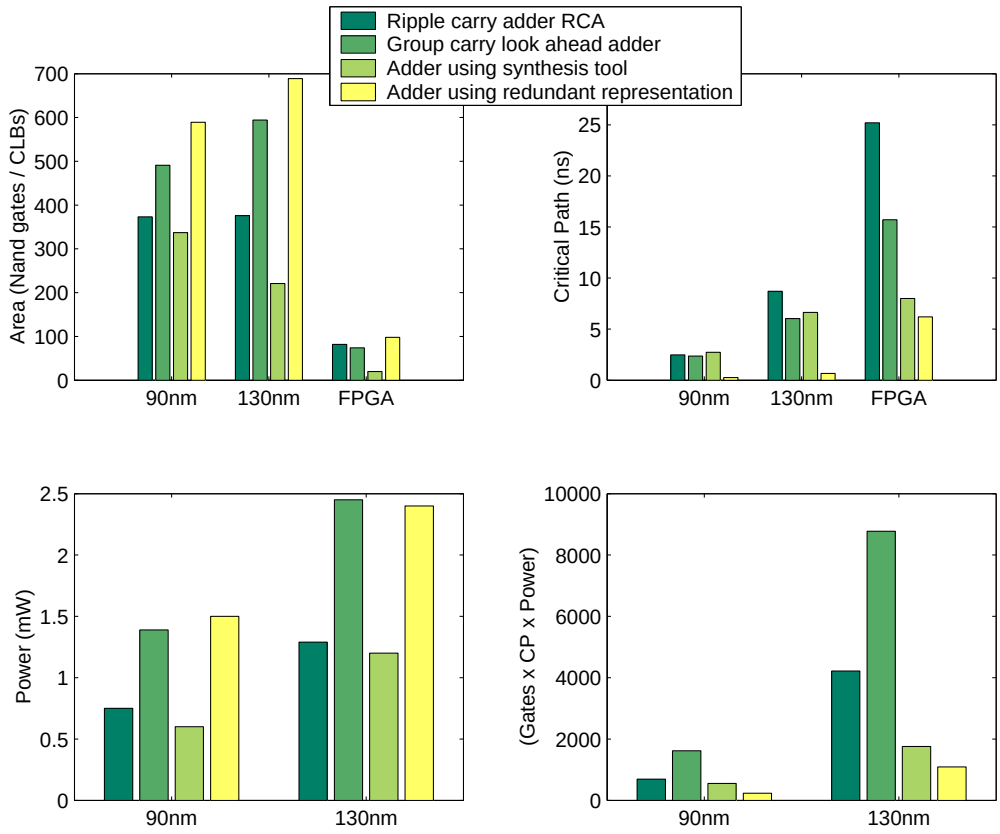

FIGURE 5.5: Comparison of CB and BS adders

As shown in Figure 5.5, the CP of BS adder is much less compared to the CB adders. On 90nm ASIC technology, the CP of BS adder is only 10%, 11% and 9% of the CP consumed by CB adders RCA, group CLA and synthesis tool adder respectively. On 130nm ASIC technology, the CP of BS adder is only 8%, 11% and 10% of the CP consumed by CB adders RCA, group CLA and synthesis tool adder respectively. On FPGA technology the resources are CLBs rather than gates so results are not in full coordination. However on FPGA the BS adder consumes almost 25%, 39% and 78% of CP consumed by CB adders RCA, group CLA and synthesis tool adder respectively. This reduction in CP

occurs due to the use of BS number system which performs addition in constant time. The decrease in CP due to the use of BS number system is even more prominent if we increase the size of adder. The reason being that CP of BS adder remains constant irrespective of input vector size but the CP of CB adders increases with the increase of input vector sizes.

The area consumed by BS adder is more compared to the area of CB adders. On ASIC technologies, the BS adder consumes almost 60%, 20% and 75% more area compared to RCA, Group CLA and synthesis tool CB adders respectively. There are two reasons for this area increase. The first one is the complexity of BS logic cell which is more than CB full adder cell. The second reason is the binary encoding of digits which uses twice number of bits compared to conventional binary numbers. The probabilistic power consumption of each adder is in accordance with number of gates utilized. The total efficiency of adder on ASIC technologies can be computed by taking the product of gates, CP and power consumption. Smaller value of this product term indicates high efficiency. On 90nm ASIC technology, the product term of BS adder is 69%, 86% and 58% less than the product terms of binary RCA, Group CLA and synthesis tool adder respectively. Similarly on 130nm ASIC technology, the product term of BS adder is 74%, 88% and 38% less than the product terms of binary RCA, Group CLA and synthesis tool adder respectively. This increase in the efficiency of BS adder occurs because of the reduction in CP. After analyzing these results, it is found that BS adders perform high speed additions at the cost of some area and power increase. Moreover the CP of BS adders are constant irrespective of vector lengths.

5.4 Conversions between CB and BS numbers

In most of the processors the standard format for representing the data is a CB number system. Therefore to perform operations on BS numbers we have to convert the input binary numbers into BS numbers before starting the computations. After required computations the results in BS format are converted back to CB representation [75, 105]. This process is shown in Figure 5.6.

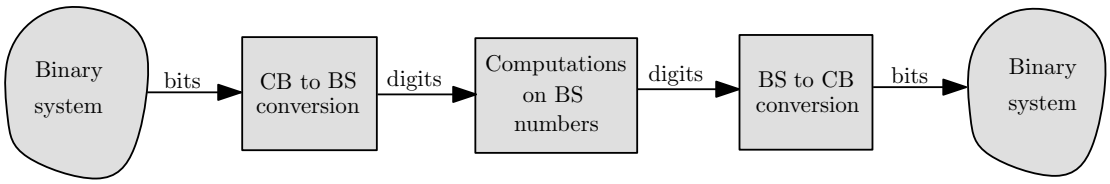

FIGURE 5.6: Conversion between CB and BS numbers

In practice once input CB data is converted to BS format, multiple required operations are performed on BS data before converting back to CB representation.

5.4.1 Conversions from CB to BS

In this conversion CB numbers are converted to BS representation. Each bit of CB number is converted to corresponding digit from digit set $\{\bar{1}, 0, 1\}$. The CB number can be converted to several different BS numbers which represents the same CB number. Based upon the requirements, any of the BS number can be used for onward BS computations. In CB number system there are basically two types of number formats that are *unsigned CB numbers* and *Signed 2's complement CB numbers*. As in these two types, bits have different weights so the conversion strategy for signed and unsigned numbers will be different.

- **Conversion from unsigned CB to BS number :** In unsigned CB numbers each bit has positive weight and the value of unsigned CB number is given by Equation 5.3.

$$\sum_{i=0}^{n-1} x_i \times 2^i \qquad \text{where} \qquad x_i \in \{0, 1\} \tag{5.3}$$

  The simplest way to convert unsigned CB number to equivalent BS number is to simply convert each bit to a digit of same value. CB bit 0 is converted to digit 0 and CB bit 1 is converted to digit 1. For unsigned CB numbers no computation is required to convert them to the equivalent BS representation as both have the same value. As there is no negative weight bit location in unsigned number so no $\bar{1}$ will appear in corresponding BS number. The value of converted BS digit number is given by equation 5.4.

$$\sum_{i=0}^{n-1} x_i \times 2^i \qquad \text{where} \qquad x_i \in \{\bar{1}, 0, 1\} \tag{5.4}$$

  The conversion describe above is one possible conversion from unsigned CB number to BS number. However other conversions can also be possible which involve $\bar{1}$ digits as well. But for unsigned numbers the combined weights of all $\bar{1}$ digits are always less than the combined weights of all 1 digits. This will ensure that the net value of BS number is always positive.

- **Conversion from signed 2's complement CB to BS number :** In signed 2's complement CB numbers, the MSB has negative weight. The value of CB number

represented in 2's complement format can be positive or negative. If MSB is 1 then the value will be negative otherwise if MSB is 0 then the value will be positive. The value of signed CB number is given by Equation 5.5.

$$-x_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} x_i \times 2^i \qquad \text{where} \qquad x_i \in \{0, 1\} \tag{5.5}$$

Due to the negative weight of MSB the conversion process will be different from unsigned numbers. If the MSB of signed 2's complement CB number has bit value of 1 then it is converted to digit value $\bar{1}$ in BS number. However if the MSB of signed 2's complement CB number has bit value 0 then it remains digit value 0 in BS number. The conversion for rest of the bits {(MSB - 1) ...0} is same as unsigned numbers that is CB bits 1 and 0 are converted to BS digits 1 and 0 respectively. In this conversion method $\bar{1}$ digit can appear only at the most significant position in BS number depending on the value of number. However based upon the requirements, the conversion can also be done in such a way that multiple $\bar{1}$ digits appear through out the BS number.

Like the BS addition, this conversion process is independent of vector length and can be performed in constant time. The conversion form CB (signed/unsigned) to BS number system is implemented on different target technologies using the methods explained above. This converter performs the conversion of signed as well as unsigned CB numbers to BS digits. The results are shown in table 5.4.

Data width (bits)	**90nm CMOS ASIC**				**130nm CMOS ASIC**				**FPGA VirtexII**		
	Nand Gates	CP (ns)	Power (mW)	Gates X CP X Power	Nand Gates	CP (ns)	Power (mW)	Gates X CP X Power	CLBs	CP (ns)	CLBs X CP
40	3	0.03	0.003	0.0003	3	0.12	0.008	0.0029	1	5.1	5.1
64	3	0.03	0.003	0.0003	3	0.12	0.008	0.0029	1	5.1	5.1
128	3	0.03	0.003	0.0003	3	0.12	0.008	0.0029	1	5.1	5.1

TABLE 5.4: Synthesis results of CB to BS digit converters

As shown in Table 5.4, the conversion process is implemented for different size vectors. The area, CP and power consumption of CB to BS converter remains same for all the vector sizes. The small area, CP, power consumed by the CB to BS conversion corresponds to conversion of signed CB numbers to BS numbers. For unsigned CB numbers, the area, CP, power requirement is almost zero as there is no need of checking MSB and binary 0s and 1s becomes digits 0s and 1s in BS numbers. The results show that CB to BS conversion consumes almost negligible resources.

5.4.2 Conversions from BS to CB representation

After performing high speed computations on BS numbers, they are converted back to CB numbers using *BS to CB converter* [75, 99, 105]. Conversion from BS to CB number is more complex compared to the conversion from CB to BS. BS to CB conversion is done in two steps. In the first step positive and negative weight binary numbers are constructed directly from BS number. Positive weight binary number is constructed from all the positive digits in BS number and the location of $\bar{1}$ and 0 digits are filled with 0 bits. Similarly the negative weight binary number is constructed from all the negative digits in BS number and the location of 1 and 0 digits are filled with 0 bits. In the second step the negative weight binary number is subtracted from positive weight binary number. The value of resulting binary number is the equivalent to the value of BS number. This process is shown in Figure 5.7.

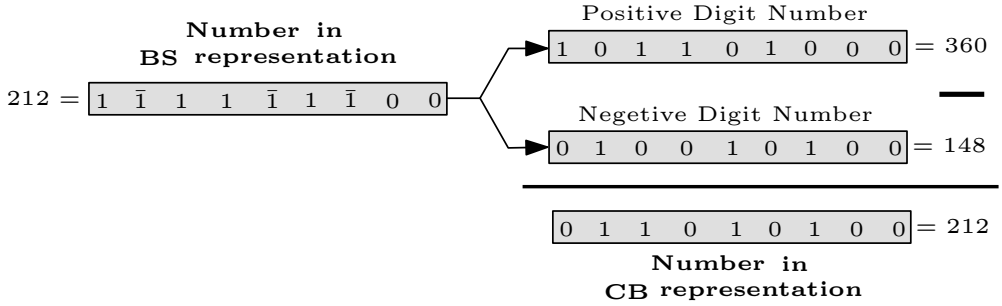

FIGURE 5.7: Conversion from BS to CB representation

An n-digit BS number can take values in the range of $\{-(2^n - 1) \ldots + (2^n - 1)\}$. For instance 8-digit BS number can take values ranging from 255 to -255. In CB number system -255 cannot be represented using 8 bits. Therefore in order to avoid any overflow (n+1) bits can be used to represent equivalent CB number.

Another conversion method [91] is based upon the direct conversion from BS to CB numbers. In this method the truth table (shown in Table 5.5) is constructed for all the possible combinations and the output CB bits are generated accordingly.

Digits	**Binary encoded**		**Carry input**	**Binary output**	**Carry output**
x_i	x_{i+}	x_{i-}	c_i	b_i	c_{i+1}
0	0	0	1	1	1
0	0	0	0	0	0
1	0	1	1	0	0
1	0	1	0	1	0
$\bar{1}$	1	1	1	0	1
$\bar{1}$	1	1	0	1	1

TABLE 5.5: BS to CB conversion rules

x_i is the input BS digit which needs to be converted. x_{i+} and x_{i-} are the bit encoded versions of input BS digit. c_i is the carry input at rank i. The truth table defines the CB bit output b_i and carry output c_{i+1} for each combination of input BS digit x_i and carry input c_i. The carry output generated at rank i is used as input for conversion at rank i+1. Therefore at each rank the equivalent bit value is determined on the basis of input BS digit and carry input. From the truth table in Table 5.5, the logic equations for the conversion of BS to CB numbers are given in Equations 5.6 and 5.7.

$$b_i = x_{i-} \text{ XOR } c_i \tag{5.6}$$

$$c_{i+1} = x_{i+} + \bar{x}_{i-} \cdot c_i \qquad \text{Where} \quad c_0 = 0 \tag{5.7}$$

As an example let us consider the conversion from BS to CB number using the logic equations shown above. The conversion from BS to CB number using this method is shown in Figure 5.8.

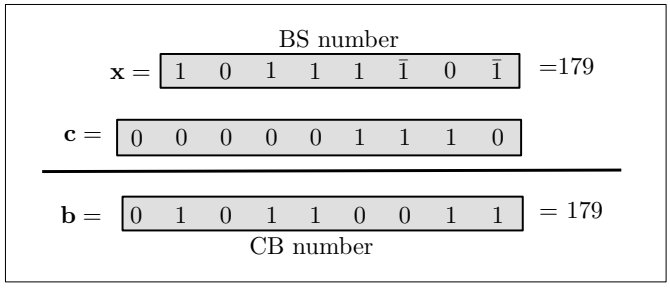

FIGURE 5.8: BS to CB conversion

As shown in the Figure 5.8 at each digit rank, the output consists of equivalent CB bit and the generated carry bit. This carry bit is used as input to find the equivalent bit value for the next rank digit. The advantage of using this method is that the binary bits are obtained directly from BS digits. BS to CB conversion explained above has been implemented on different target technologies for three different vector lengths. BS to CB converter is implemented for vector length of 40-digit, 64-digit and 128-digit. The area, CP and power consumption on different target technologies are shown in Table 5.6.

Data width (digits)	**90nm CMOS ASIC**				**130nm CMOS ASIC**				**FPGA VirtexII**		
	Nand Gates	CP (ns)	Power (mW)	Gates X CP X Power	Nand Gates	CP (ns)	Power (mW)	Gates X CP X Power	CLBs	CP (ns)	CLBs X CP
40	418	2.8	0.7	819	301	6.8	1.3	2661	21	8.0	168
64	664	4.4	1.1	3214	477	10.8	2.1	10818	33	9.0	297
128	1320	8.7	2.2	25265	946	21.4	4.3	87051	65	11.7	761

TABLE 5.6: Synthesis results of BS to CB converter

As shown in Table 5.6, the area, CP and power consumption of BS to CB converter increases with the increase of input vector length. The resources consumed by BS to CB converter are more compared to the CB to BS converter. The reason being the difference in the complexity of two processes. However these conversions are only required at the input and outputs of BS computations. In the beginning once converted to BS number system, numerous high speed computations are performed on BS digits as per the requirements. Then at the end the resultant BS digits are converted back to CB bits. Therefore the high speed attained due to multiple operations on BS data is much more compared to the resources consumed by these converters at input and output.

5.5 Multiplication using BS number system

Multiplication is one of the most important operation which is used in most of the applications. Multiplication involves the addition of partial products which can be performed with high speed using carry propagation free adders. This will increase the overall speed of multiplication process. The multiplication process using the BS number system consists of four main steps.

- **Conversion from CB to BS representation** In this step both input binary vectors multiplicand and multiplier are converted to BS digit vectors using the method explained in Section 5.4. Both inputs, multiplier and multiplicand can be unsigned or signed 2's complement numbers. After conversion the vectors are in BS representation and consist of digits rather than bits.

- **Generation of partial products** In this step the partial products are generated using BS digit vectors. Corresponding to each digit of multiplier vector partial product vector is generated. Based on the digit set $\{\bar{1}, 0, 1\}$ three types of partial product vectors are possible. If the multiplier digit is $\bar{1}$ the the partial product vector is the inversion of multiplicand digit vector. Inversion of digit vector means 1s are converted to $\bar{1}s$ and $\bar{1}s$ are converted to 1s and 0s remains same. If the multiplier digit is 0 then the corresponding partial product vector consists of all 0 digits. If the multiplier digit is 1 then the partial product vector is the copy of multiplicand digit vector. The generation of partial product using BS digits is simpler compared to the CB numbers. In the multiplication of CB numbers, the generation of partial products for signed and unsigned numbers is slightly different. For signed multiplication, 2's complement of last partial product is taken. Moreover in CB multiplication each partial product is either zero padded (unsigned multiplication) or sign extended (signed multiplication) before the addition. However

in BS number system the distinguish between signed and unsigned input vectors is done only at the time of CB to BS digit conversion. After conversion, the partial products for all the BS digits are generated in the same manner without any zero padding or sign extension. This reduces the complexity of partial product generation process.

- **Addition of partial products** In this step, the generated partial products are added using high speed carry propagation free adders explained in Section 5.2. This addition is done using adder trees. The addition of partial product at each level is done in parallel. This addition is completed in constant time irrespective of the size of the partial product. Due to this fast addition of partial products the overall speed of multiplication also increases.

- **Conversion from BS to CB** In this step the output of BS multiplier is converted back to CB representation using the method explained in Section 5.4.

The multiplication process explained above is shown with the help of an example in Figure 5.9.

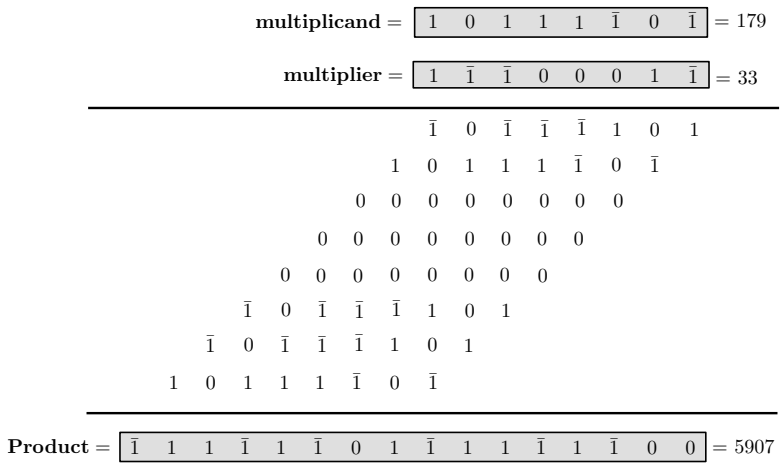

FIGURE 5.9: Multiplication using BS number system

Figure 5.9 shows the multiplication of two 8-digit numbers using BS data representation. Eight partial products are generated corresponding to each digit of multiplier vector. These eight partial products are arranged in such a way that each successive partial product is left shift by one digit rank. In the next step these partial products are added using BS adder tree. As there are eight partial products so there are $log_2(8) = 3$ levels of addition. Within each level all the additions are performed in parallel in constant time. After three level of addition the product is obtained which is in BS digit form. Based on the requirement, this product can be converted back to CB representation. The data length of product in CB form will be 16 bits.

5.5.1 Comparison of CB and BS multiplier

Comparison of BS multiplier explained in this section is done with CB multiplier. For this comparison 40-bit/digit length is selected. CB multiplier takes two 40-bit vectors as input and generates 80-bit result. In CB multiplier partial products are generated by ANDing each multiplier bit with all the bits of multiplicand vector. The addition of partial products is done using tree of binary adders. On the other hand the BS multiplier takes two 40-digit numbers as input and generate 80-digit product. Table 5.7 shows the synthesis results of CB and BS multiplier when implemented on different target technologies.

Number system	**90nm CMOS ASIC**				**130nm CMOS ASIC**				**FPGA VirtexII**		
	Nand Gates	**CP** (ns)	**Power** (mW)	**Gates X CP X Power**	**Nand Gates**	**CP** (ns)	**Power** (mW)	**Gates X CP X Power**	**CLBs**	**CP** (ns)	**CLB X CP**
CB	14518	6.1	17.2	1523229	10532	14.0	23.3	3435538	917	19.7	18065
BS	31268	2.5	37.0	2892290	26330	5.5	58.2	8428233	1862	11.9	22158

TABLE 5.7: Comparison of CB and BS multipliers

As shown in Table 5.7, the CP of 40-digit BS multiplier is less than the CP of 40-bit CB multiplier. The reason being that BS multiplier uses high speed adder tree to add the partial products. On the other hand CB multiplier uses binary adder tress whose CP increases with the increase of vector length. On ASIC technology, the CP of BS multiplier is almost 60% less than the CP of CB multiplier. On FPGA platform, the CP of BS multiplier is almost 40% less than the CP of CB multiplier. Due to this reduction in CP, the BS multiplier performs multiplications in less time compared to CB multiplier. The overhead for this high speed is the increase in area and power consumption. On all target technologies, the CB multiplier consumes almost 50% of the area and power consumed by BS multiplier. Due to the increase in the area and the power of BS multiplier, the product term (area×CP×power) of CB multiplier is 53% and 41% of BS multiplier on 90nm and 130nm ASIC technologies respectively. The increase in gate count in BS multiplier occurs because the logic cell for the addition of BS digits is more complex compared to binary full adder cell. Also due to the encoding of each digit by two bits, the number of bits increases which ultimately increases the area. But due to high speed requirements in different applications, these overheads have less importance. Moreover the CP of CB multiplier is greater, therefore to obtained the same throughput as BS multiplier, we need two or three CB multiplier units in parallel which increases the area accordingly.

5.6 FSM based variable length BS adder

Based on the requirements, the hardware resources can be shared in the designing of operators. In these operators same hardware resources are used multiple times and the operations are performed with minimum design resources while meeting the timing constraints. The number and type of resources is determined by the area and timing constraints. If the area constraints are flexible then more number of resources can be used to accomplish the task in less time. Similarly if the timing constraints are not very strict then less number of resources can be used to perform the task.

In this section the BS adders are designed using different resource sets. Different resources which are required to perform addition operation using BS representation are *CB to BS digit converter*, *BS adder* and *BS to CB converter*. Multiple units of each these resources can be used in the design. Data size of each of these resources also depends upon the design constraints. Larger size units usually consume more area compared to the smaller units. Some times larger size units are beneficial in the sense that they complete the task in less time compared to the multiple usage of the small size unit. However it depends upon the actual design constraints which determine the size and the number of units required. Finite state machine (FSM) controller is used to control the sequence of operations performed by these units. This controller enables each resource at appropriate time. After the completion of the task the resource is disabled. Depending on the requirement controller can activate the resources more than one time.

The BS adder explained in this section can be used as different size adder. This adder is designed with minimum resources so that the implementation area can be minimized. It utilizes the available resources and produces the sum at the output. This adder can perform addition on 8, 10, 12, 16 and 32-bit data. Therefore the same adder is used as either one 8-bit adder or one 10-bit adder or one 12-bit adder or one 16-bit adder or one 32-bit adder. The input consists of two 32-bit CB numbers and *data size* control signal. The *data size* control signal directs the operator about the size of adder required. For instance *data size* signal directs the operator that one 8-bit adder is required or one 10-bit adder is required etc. When this adder is used as 8-bit adder the input bits from $(7 \ldots 0)$ are used only. The rest of the bits $(31 \ldots 8)$ in 32-bit input vectors are ignored. However when the adder is used as 32-bit adder then the whole 32 bits of input vectors are used. Figure 5.10 shows the hardware resources required to perform these additions.

As shown in Figure 5.10, the resource set consist of different units. These units are utilized to obtain the BS adders of different sizes. FSM controller controls the operation of different units. The details of resources used in this design are given below.

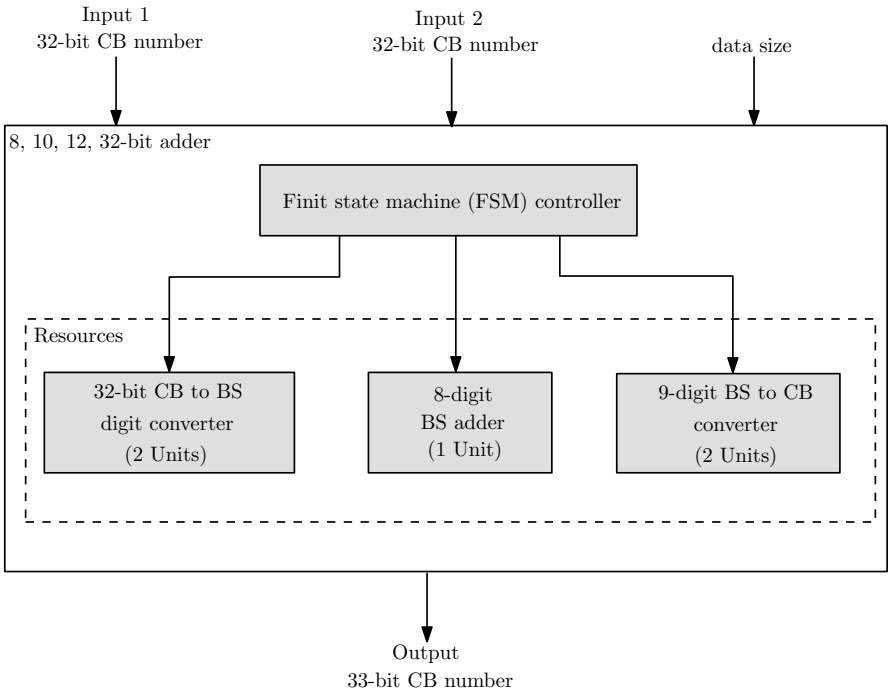

FIGURE 5.10: FSM based BS adder

- **32-bit CB to BS digit converter** Two units of *32-bit CB to BS digit converter* are used in this adder design. These units are used to convert input CB numbers to BS numbers. The input to each of these units is 32-bit CB numbers. The output of these units consists of 32-digit BS numbers. This resource consumes very small area irrespective of input vector length. Therefore instead of using it for small size data it is used for maximum data size of 32-bit.

- **8-digit BS adder** One unit of *8-digit BS adder* is used. This unit is used to add two 8-digit BS numbers and generates 9-digit result. One extra digit is used to avoid any overflow. This unit uses carry propagation free addition on BS digits. For the addition of larger vectors (10, 12, 16 and 32-digits), this unit is used multiple times.

- **9-digit BS to CB converter** One unit of *9-digit BS to CB converter* is used. This unit is used to convert the BS digits into CB bits. It takes 9 digits as input and generates 9-bits. As the output of *8-digit BS adder* are 9 digits therefore the size of *BS to CB converter* is also chosen as 9-digit.

5.6.1 FSM controller

The functions of different units used in the design of this adder are controlled with the help of FSM controller. It controls all the operations and activates the units which are

required at in particular states. Depending on the *data size* control bits, the FSM controller activates the units corresponding to the size of the adder required. The sequence of operations performed for different size adders are shown in Figure 5.11.

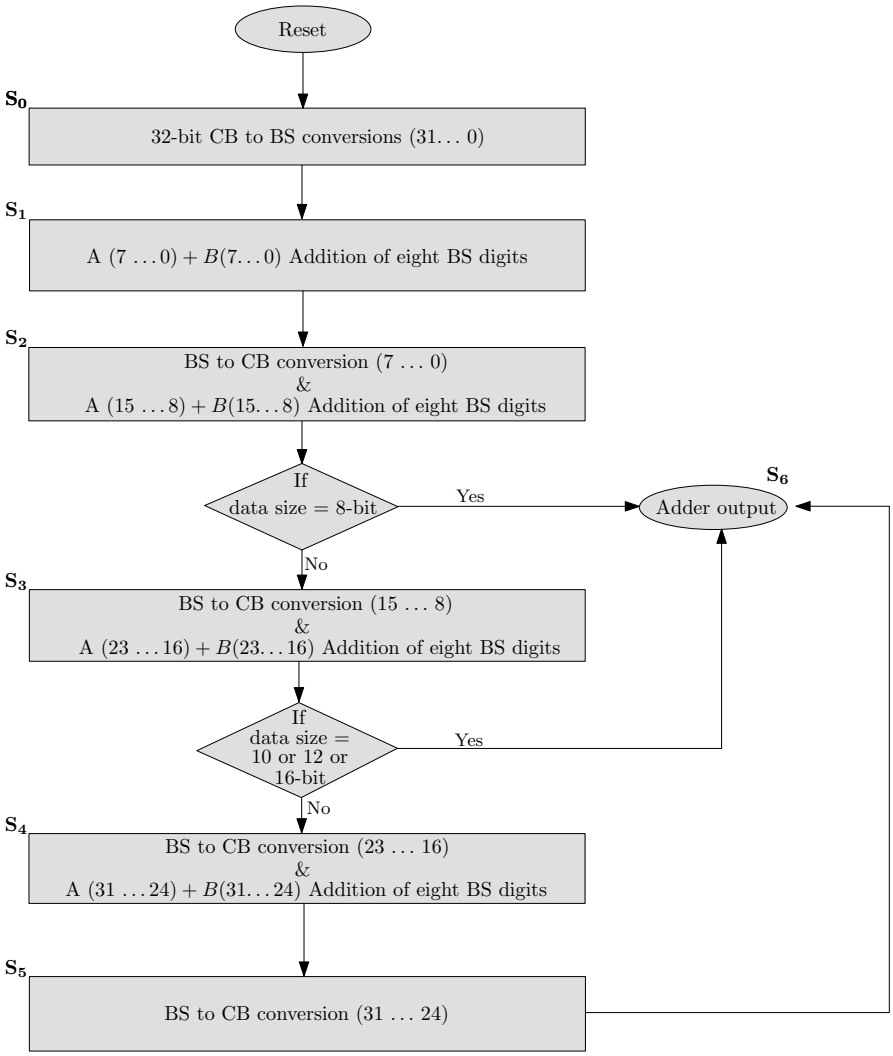

FIGURE 5.11: State diagram of adder

As shown in Figure 5.11, the finite state machine (FSM) controller for the adder consists of different states. In each state certain units are enabled which needs to perform operations. The controller starts with state S_0 and activates the *32-bit CB to BS digit converter* units. These units convert input CB bits to BS digits. After this conversion controller enters in state S_1 and activates *8-digit BS adder* unit. This unit adds the 8 digits inputs ranging from (7 ... 0). The output is the 9-digit sum. In the next state S_2 the *9-digit BS to CB converter* unit is activated. This unit converts 9-digit data from BS adder to CB format. In state S_2 the unit *8-digit BS adder* is free and data for this unit is also available for computation. So *8-digit BS adder* unit is also activated in state S_2 to add 8-digits ranging from (15 ... 8). After state S2, the controller checks *data size*

control signal about the size of the adder required. If the 8-bit adder is required then the controller gives the 9-bit sum at output and goes into initial state S_0 to wait for next operation. However if the size of the adder required is greater than 8-bit then controller enters into state S_3. In the state S_3 both *8-digit BS adder* unit and *9-digit BS to CB converter* unit are activated in parallel. *8-digit BS adder* unit performs the addition of input digits ranging from (23 ... 16) and the *9-digit BS to CB converter* unit performs the conversion of digits ranging from (15 ... 8). After state S_3 the data size control signal is checked to determine whether the required size of addition has been performed or not. If the required size of adder is either 10, 12 or 16-bit then the controller gives the sum bits at output and goes into initial state S_0. However if the size of the adder required is more than 16-bit then this process is repeated in states S_4 and S_5. After state S_5 the sum of two 32-bits is obtained at the output.

The advantage of using this adder is that addition of different size data is performed using only one adder unit of 8-digit size. Depending upon the requirement, the resource set of adder can be increased which will reduce the addition time. However the implementation area increases with the increase of resources. These adders are synthesized to different target technologies. Table 5.8 shows the synthesis results of FSM based adder with different resource sets.

Res. Set	**90nm CMOS ASIC**				**130nm CMOS ASIC**				**FPGA VirtexII**		
	Nand Gates	**CP (ns)**	**Power (mW)**	**Eff.**	**Nand Gates**	**CP (ns)**	**Power (mW)**	**Eff.**	**CLBs**	**CP (ns)**	**Eff.**
RS1	1495	0.90	0.88	8288	1726	2.62	1.5	47482	88	4.4	2710
RS2	2316	1.07	1.14	14125	2632	2.88	1.98	75044	121	12.2	7381

TABLE 5.8: Synthesis results of FSM based BS adders

Resource set *RS1* in Table 5.8 consist of one *32-bit CB to BS digit converter*, one *8-digit BS adder* and one *9-digit BS to CB converter*. The area, CP and power consumption of complete adder using these resources are shown in Table 5.8. The number of cycles required to add two numbers depends upon the size of numbers needs to be added. For instance if this adder is used to add two 8-bit numbers using resource set *RS1* then it requires 4 clock cycles. Similarly for the addition of 32-bit numbers it requires 7 clock cycles. Now if we increase the resources by little amount then the number of cycles required to obtain the sum decreases accordingly. For this purpose, in resource set *RS2* instead of *9-digit BS to CB converter*, we use *32-digit BS to CB converter* unit. The FSM controller will activate the units in accordance with new resources. Due to this small increase in resources the number of cycles required to compute the sum of 32-bit numbers have been reduced from 7 to 5 clock cycles. However as we increase the resources the implementation area, CP and power consumption also increases accordingly. On

different target technologies, the efficiency of adder using resource sets *RS1* and *RS2* are also shown. This efficiency is computed by taking the product of area, power, CP and cycles required to add two 32-bit numbers. Smaller value of this product indicates high efficiency. On all technologies, *RS1* has high efficiency because of less area, CP and power consumption. However the adder using resource set *RS2* consumes less cycles to compute the addition of two numbers. Based upon the implementation constraints any number and size of units can be used to meet the performance requirements.

5.7 SWP using BS representation

In the previous chapters, the design of a SWP based multimedia operator is described. The performance of the operator was improved through the use of SWP on multimedia oriented pixel sizes without focusing on the internal speed of the different processing units (ADD, SUB, MULT etc.). These operators use SWP on CB data which has the inherited carry propagation feature at any stage of the arithmetic operation resulting in an overall limited speed for the different multimedia operations. In this section, we emphasize on the designing of a multimedia operators which not only uses multimedia oriented subword sizes to increase the resource utilization but also increases the speed of the different processing units through the use of BS number system. The carry propagation free addition property of BS number systems increases the speed of the different basic operations which ultimately increases the speed of multimedia operations like SAD, DCT etc. To our best knowledge, it is the first time redundant number system is applied to SWP design.

To increase the parallelism as well as the speed of processing unit, SWP capability is introduced in the operator's design that are working on BS data representation. By doing so the operator performs parallel computations on subwords which are represented in BS data format. In the designing of SWP BS operators, multimedia oriented subword sizes (8, 10, 12 and 16-bit) are considered rather than classical subword sizes (8, 16 and 32-bit etc.). These multimedia oriented subword sizes are in coordination with pixel sizes in modern multimedia applications. Due to this coordination the utilization of processor resources increases. The word size of 40-bit is chosen because it gives good efficiency for different multimedia pixel sizes.

5.7.1 SWP adder using BS representation

The most commonly used operations in multimedia applications are addition, subtraction, absolute value, multiplication, SAD $\sum(a-b)$ for motion estimation, sum of products $\sum(a \times b)$ for discrete cosine transform DCT etc. Almost all these operations require addition/subtraction at some stage of their internal computation. Therefore efficient adder scheme can increase the overall performance of all the arithmetic units in the multimedia operator.

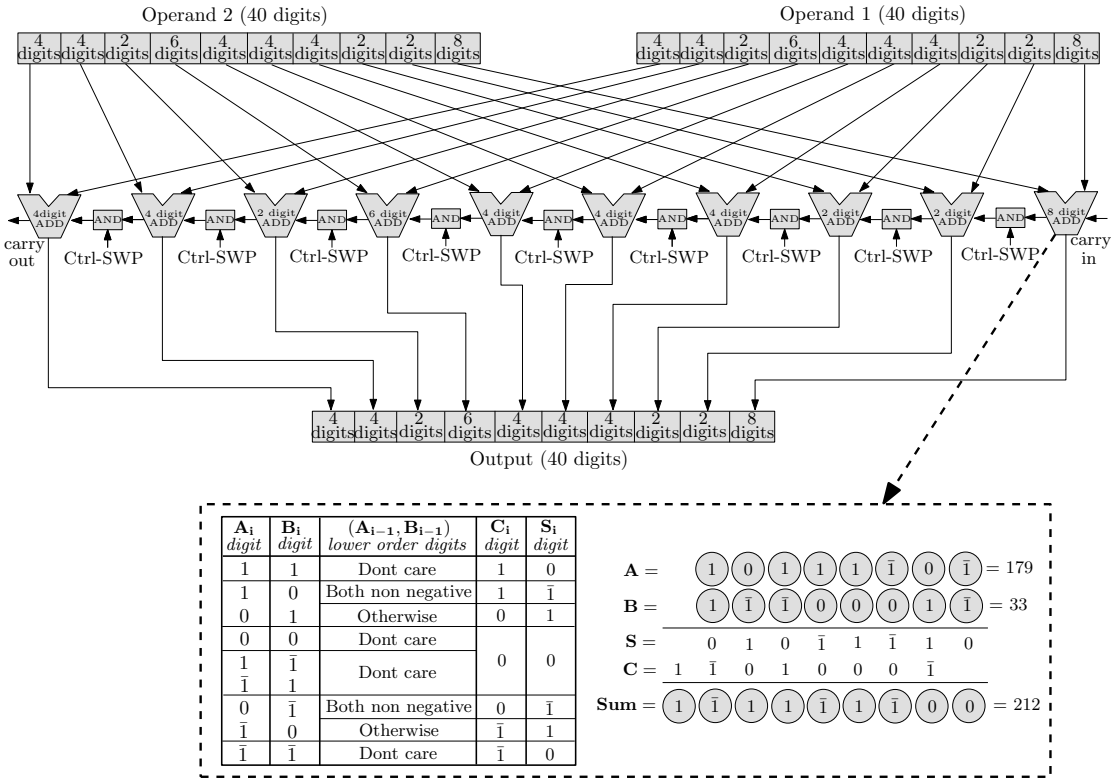

FIGURE 5.12: SWP BS ADD architecture

The *SWP BS adder* is used to perform the SWP addition of subword data in the BS representation. The *SWP BS adder* can perform either five 8-digit additions or four 10-digit additions or three 12-digit additions or two 16-digit additions or one 40-digit addition. The input to *SWP BS adder* consists of two 40-digit vectors and SWP control signal (SWP_{ctrl}). The SWP_{ctrl} indicate the operator about the size of subword digits which are packed in the input vectors. Along with parallelism, the speed of addition is increased by adding the numbers using BS adders. The architecture of *SWP BS adder* is based upon the breaking of adder chain at subword boundaries. Based upon the selected subword size, the adder chain at subword boundaries are either break or continued as shown in Figure 5.12.

To increase the speed of the addition, carry propagation free adders are used on BS data between the control logics. There are several ways to add BS digits [45, 89] without propagation of the carry (see section 5.2.1). In our *SWP BS adder*, the addition of BS digits is performed using the addition Table 5.1. Using BS adders, the propagation of the carry is avoided and a parallel addition can be performed in a constant time irrespective of the word length of the vectors to be added.

- **Overflow control digits in SWP BS adder** To avoid any overflow, one extra digit must be allocated to each resultant subword. The total number of extra digits required to prevent the overflow for different selection of subword sizes depends upon the maximum number of subword sizes which can be packed in word size register. In our case maximum number of five 8-digit subwords can be packed in word size register of 40-digit. Therefore five extra digits are required which can avoid the overflow for all the selected subword sizes. Allocation of extra digits for subword size of 8-digit is shown in 5.13.

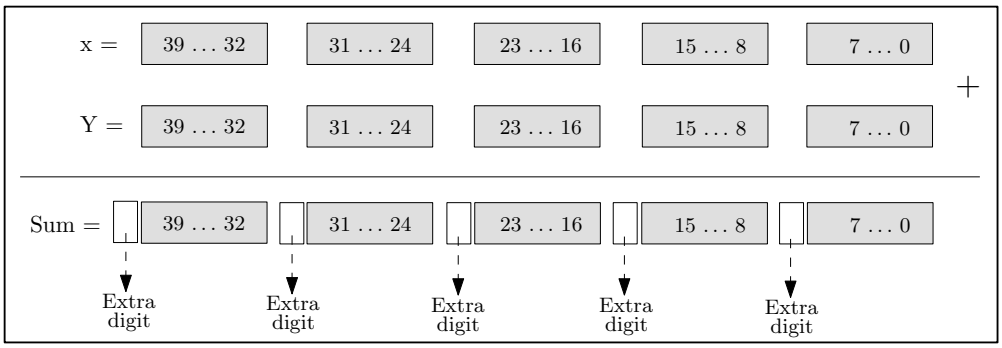

FIGURE 5.13: SWP adder for 8-digit subwords

Hence the addition of two 8-digit subwords results in 9-digits. Due to these extra digits the complete output of SWP redundant adder consists of 45 digits. For other subword sizes, the extra digits are allocated at their respective subword boundaries.

5.7.1.1 Comparison of SWP BS adder with SWP CB adder

Without considering the BS conversions, the critical path (CP) of the SWP adder on the BS representation is less compared to the CP of a SWP adder on CB data of same word and subword sizes (multimedia oriented). Compared to a CB SWP adder, the CP of the BS SWP adder is almost 51% less for 8-bit subwords up to 85% less for 16-bit subwords. Although the BS conversions also consume some critical path but its value is very small compared to the CP saved while performing several arithmetic operations. The area and dynamic power overhead for this high speed are only 29% and 16% respectively.

5.7.1.2 Comparison of simple and SWP BS adder

To analyze the performance, both simple and SWP BS adder are synthesized to different target technologies. Simple 40-digit BS adder can perform the addition of 40-digit BS numbers. Where as SWP BS adder can perform multiple parallel add operations based upon selected subword size. Due to this parallelism extra hardware resources are required. Table 5.9 shows the synthesis results of simple and SWP BS adder.

	90nm CMOS ASIC				130nm CMOS ASIC				FPGA VirtexII		
	Nand Gates	CP (ns)	Power (mW)	Gates X CP X Power	Nand Gates	CP (ns)	Power (mW)	Gates X CP X Power	CLBs	CP (ns)	CLBs X CP
Simple	589	0.26	1.5	230	689	0.66	2.4	1091	98	6.2	608
SWP	645	0.3	1.5	290	730	0.7	2.6	1329	112	6.2	694
Overhead	10%	15%	0%	26%	6%	6%	8%	22%	14%	0%	14%

TABLE 5.9: Synthesis results of simple and SWP BS adder

As shown in table 5.9, the area and CP of SWP BS adder is slightly more than simple 40-digit adder. On ASIC technology, SWP BS adder requires almost 10% and 15% more area and CP respectively compared to the simple 40-digit BS adder. On FPGA technology, SWP BS adder requires 14% more area compared to simple 40-digit BS adder. The power consumption of simple and SWP design on both ASIC technologies is almost same. The (area×CP×power) product term of SWP BS adder is 26% and 22% more than simple 40-digit adder on 90nm, 130nm ASIC technologies respectively. This small increase in the resources occurs due to the parallelism provided by SWP adder.

5.7.2 SWP multiplier using BS representation

Multiplier is one of the most important basic processing unit in the design of any arithmetic operator. SWP BS multiplier can perform several multiplications in parallel on BS data representation. Use of SWP will increase the parallelism and the operations on BS data will increase the speed of multiplication process. In SWP BS multiplier, the multiplication of BS digit subwords are performed in the same manner describe in Section 5.5. Partial products are generated for each digit of multiplier vector. However for each selected subword size only those partial product digits are generated which are required.

- **Partial product generation unit for SWP BS multiplier :** The architecture of partial product generation unit for SWP BS multiplier is based upon the partial product generation hardware for SWP binary multiplier explained in Section 2.3.2

of Chapter 2. However in SWP BS multiplier the digits are used instead of bits which results in the simplicity of PP generation as well. In SWP BS multiplier, partial product generation unit generate partial products for different selection of subword sizes. The arrangement of subwords within the input registers is different for different subword sizes. Therefore the arrangement of partial product blocks will also be different corresponding to the different selection of subword sizes. Before the generation of each partial product, the multiplicand vector is updated in accordance with selected subword size. For different selection of subword sizes, the multiplicand vector gets different values. This updated multiplicand vector is then used to generate partial products. The partial product generation hardware remains same irrespective of selected subword size. Therefore instead of using dedicated partial product generation hardware for each subword size, the same partial product generation hardware is used for different subword sizes. The distinction between signed and unsigned subwords is made at the time of the CB to BS conversion therefore no sign extension, zero padding or correction vector is required for signed/unsigned PPs.

- **Addition of partial products for SWP BS multiplier :** The generated partial products are added using high speed BS adder trees. At each level of tree, these adders add the partial products in parallel. For each subword size the product subwords are represented by twice number of digits. For 8-digit subword size, each product subword consists of 16 digits. These 16 digits are enough to represent the product of two 8-digit subwords hence no overflow will occur. Similarly for other subword sizes, the product subwords have twice data lengths. For word size multiplication of 40-digits the product consists of 80 digits.

5.7.2.1 Comparison of SWP BS multiplier with SWP CB multiplier

Due to the use of high speed BS adders, the SWP BS multiplier computes the product of subwords in less time compared to SWP CB multiplier. Compared to the SWP CB multiplier of same word and subword sizes, the SWP BS multiplier computes the product in 40% less time (for 8-bit subwords) up to 51% less time (for 16-bit subwords). The area and dynamic power overhead for this high speed are 40% and 49% respectively. This area increase occurs because the binary encoding of each BS digit requires at least two bits which ultimately increases the overall area of SWP BS multiplier.

5.7.2.2 Comparison of simple and SWP BS multiplier

SWP BS multiplier performs several subword multiplications in parallel. On the other hand simple BS 40-digit multiplier can perform multiplication of two 40-digit numbers and generate 80-digit product. Therefore it is obvious that due to SWP controls, SWP BS multiplier will require more resources compared to simple BS multiplier. However by using the efficient schemes for partial product generation and addition, the overheads of SWP multipliers can be reduced. Simple as well as SWP BS multipliers are synthesized on different target technologies and the results are shown in Table 5.10.

	90nm CMOS ASIC				**130nm CMOS ASIC**				**FPGA VirtexII**		
	Nand Gates	CP (ns)	Power (mW)	Gates X CP X Power	Nand Gates	CP (ns)	Power (mW)	Gates X CP X Power	CLBs	CP (ns)	CLBs X CP
Simple	31268	2.5	37.0	2892290	26330	5.5	58.2	8428233	1862	11.9	22158
SWP	31517	3.2	38.3	3862723	26361	7.4	59.8	11665270	2418	13.4	31434
Overhead	1%	28%	4%	34%	1%	35%	3%	38%	30%	13%	42%

TABLE 5.10: Synthesis results of simple and SWP BS multiplier

As shown in the Table 5.10, SWP BS multipliers requires little more resources compared to simple 40-digit BS multiplier. On 90nm ASIC technologies, SWP BS multiplier requires almost 1%, 28% and 4% more area, CP and power respectively. On 130nm ASIC technologies, SWP BS multiplier requires almost 1%, 35% and 3% more area, CP and power respectively. Similarly on FPGA technology, SWP BS multiplier requires 30% and 13% more area and CP respectively. The (area×CP×power) product term of SWP BS multiplier is 34%, 38% and 42% more than simple BS multiplier on 90nm, 130nm ASIC and on FPGA technologies respectively. The small increase in resources for SWP BS multiplier occurs because SWP multiplier supports more parallel operations.

5.8 SWP SAD using BS representation

Sum of absolute value differences (SAD) is one of the most commonly used operation in video applications for motion estimation etc. SAD operation is used in this section to explain that how computations are handled in our multimedia operator using the basic arithmetic units explained so for. SAD operation is given by Equation 5.8.

$$\text{SAD} = \sum_{i=0}^{N-1} |a_i - b_i| \tag{5.8}$$

As the SAD operation is normally applied to low precision pixel data in multimedia applications, therefore subword parallelism SWP can enhance the performance of a SAD unit. In one computation round, rather than calculating single SAD computation, SWP allow to compute SAD computations on multiple pixels data. The main functions in the calculation of the SAD are finding of the absolute values of the difference and their accumulation. Both these functions involve the addition/subtraction process. To improve the speed of SWP SAD operator, carry propagation free BS adders are used instead of binary adder. These BS adders perform the addition in constant time irrespective of word length. Therefore, the use of both SWP along with BS adders increases throughput and speed of SAD operator. A pipelined architecture of the SWP SAD operator using BS representation is shown in Figure 5.14.

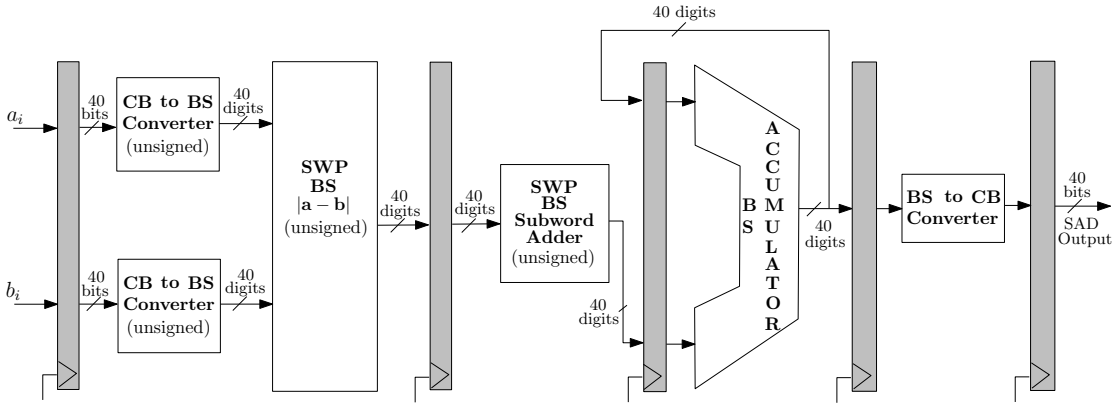

FIGURE 5.14: SWP BS SAD unit

A dedicated control signal, not shown in Figure 5.14 is used to select the subword size for the SWP units. After converting numbers from CB to BS representation, a *SWP BS* $|a-b|$ *unsigned* unit is used to compute the absolute value of the difference of the packed subwords in the BS format. To achieve maximum advantages of the carry propagation free addition, this unit implements the absolute value operation using SWP BS adders/subtractors. The *SWP BS* $|a-b|$ *unsigned* unit either calculates $a-b$ (when $a > b$) or $b-a$ (when $b > a$) on BS subword data. Therefore in order to get more advantage (high-speed) of carry propagation free addition/subtraction, *SWP BS* $|a-b|$ *unsigned* operation is implemented using BS subtractors instead of implementing absolute value calculation directly as shown in figure 5.15.

Due to subtraction operation overflow is not possible hence 40 digits are sufficient to store the resultant subwords from the *SWP BS* $|a-b|$ *unsigned* unit. The comparator unit (*SWP* $(a > b)$) generate boolean values equal to the number of selected subwords packed in word size input registers. The maximum values are generated when selected subword size is minimum. For 8-bit subword size, five boolean values (maximum) are generated as there are five packed subwords. For other larger subword sizes less number

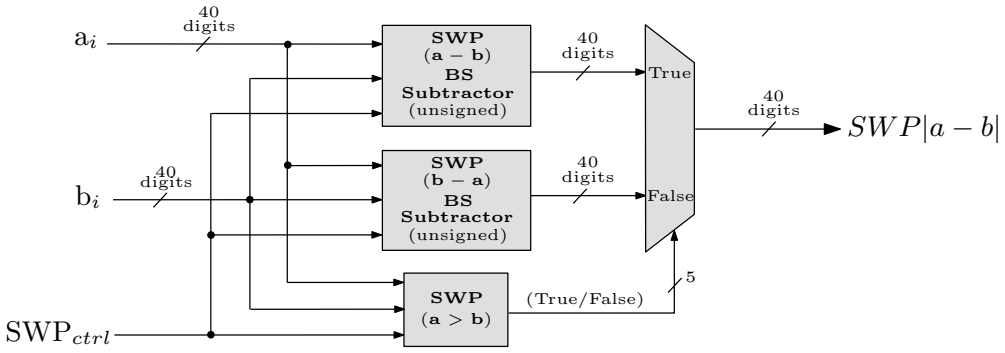

FIGURE 5.15: SWP BS $|a-b|$ unit

of comparator outputs are used. The output of *SWP BS* $|a-b|$ unit is in the form of packed subwords. The *SWP BS subword adder* unit adds the subwords packed in one register to obtain a single value. Based upon the selected subword size, the *SWP BS subword adder* unit separates the subwords packed in its input register and then performs the addition of these subwords using a BS adder.

To obtain the SAD value in the BS format, the output of the *SWP subword BS adder* unit is accumulated recursively using the *BS accumulator*. All these units ($|a-b|$, subword adder and accumulator) involve additions/subtraction of subwords which are performed on BS digits rather than CB bits which increases the speed of the overall SWP SAD operation. Finally the BS SAD output is converted to its CB representation using *BS to CB Converter* unit. Due to the use of fast BS adders, the BS pipelined SWP SAD unit can operate at 50% faster frequency (for 8-bit subwords) upto 70% faster frequency (for 16-bit subwords) compared to a CB SWP SAD unit. The area and power overheads of this speed enhancement are only 31% and 30% respectively.

5.9 SWP BS conversions

Conversion units are required at the input and output of SWP operator. At the input these conversion units convert CB bits to BS digits so that the arithmetic operations can be performed on BS data format. At the output the BS data is converted back to CB format. As in SWP arithmetic operators, the operations are performed on subwords, therefore the multimedia oriented SWP capability is also required in the conversion units. SWP conversion units perform parallel conversions on all the subwords which are packed in input registers. These conversions are based on the methods explained in section 5.4. Due to parallel conversions on multiple subwords, SWP conversion units require slightly more resources compared to simple conversion units.

5.9.1 SWP CB to BS conversion

SWP CB to BS conversion unit is used to convert the CB subword bits into BS subword digits. Based upon the selected subword size, the conversion is performed on 8, 10, 12 or 16-bit subwords. For different subword sizes there are different numbers subwords packed in the word size register. Therefore the number and the location of MSB's of packed subwords will be different for different subword sizes. For 8-bit subword size, there are five MSBs, as there are five 8-bit subwords in 40-bit register. The conversion of signed CB subwords to BS subwords using SWP CB to BS conversion unit is shown in Figure 5.16.

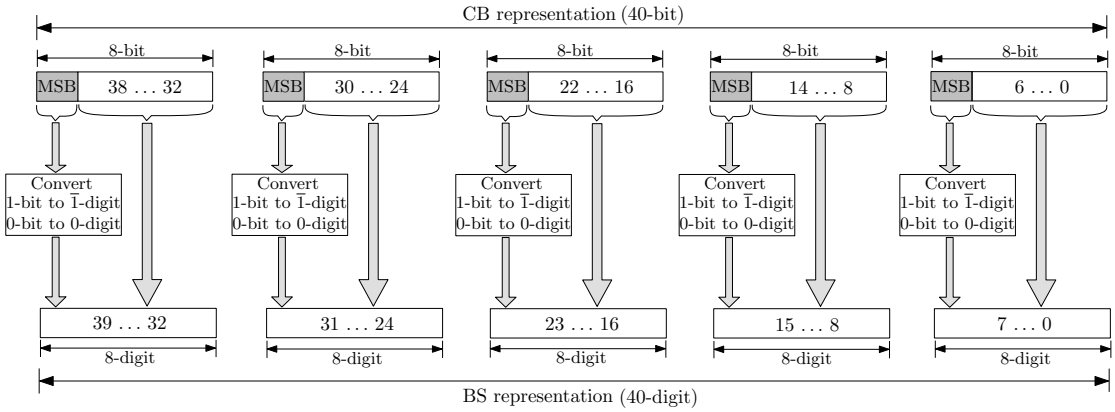

FIGURE 5.16: SWP CB to BS conversion

As shown in figure 5.16, except the MSB of each subword the remaining bits of all the subwords gets the same value in BS representation. The MSB of each subword is monitored for '0' or $\overline{1}$ conversion. For other subword sizes, the subword bits are converted to digits accordingly.

5.9.2 SWP BS to CB conversion

SWP BS to CB conversion unit is used to convert the BS subword digits to CB subword bits. The input to SWP BS to CB conversion unit is 40-digit vector. Which contain either five 8-digit subwords or four 10-digit subwords or three 12-digit subwords or two 16-digit subwords or one 40-digit word. Based on selected subword size, SWP BS to CB conversion unit converts the packed subwords from BS to CB format. For subword size of 8-bit, this parallel conversion process is shown in Figure 5.17.

As shown in figure 5.17, each subword is converted to positive and negative weight binary numbers. For each subword size, the negative weight number is subtracted from positive

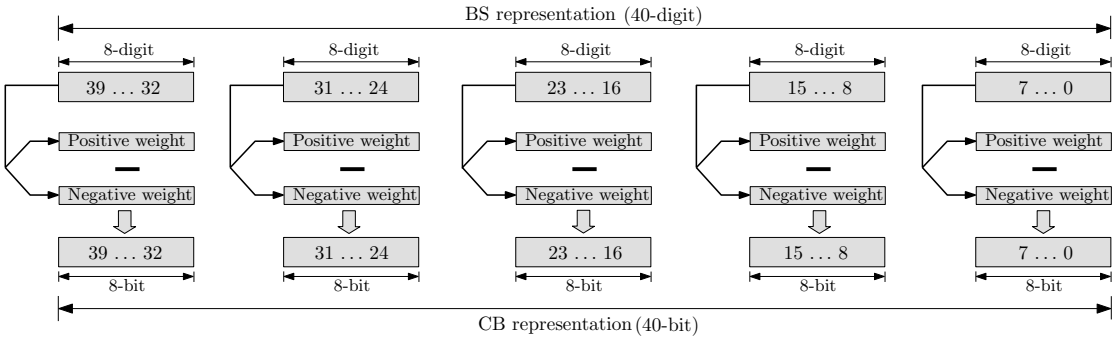

FIGURE 5.17: SWP BS to CB conversion

weight number to get subwords in CB format. For other subword sizes, the conversion is applied on corresponding subword sizes.

Depending on the output of operator, either simple (for single value) or SWP (for packed subwords) *BS to CB converter* unit is required at the output of arithmetic operator. The overhead of this conversion is very small compared to the overall computations performed by the operator. For instance, in the SWP reconfigurable multimedia operator explained in Section 5.10, the *SWP BS to CB converter* unit only consumes 3%, 6% and 4% of total area, CP and dynamic power respectively.

5.10 High speed reconfigurable multimedia operator

For better performance and efficiency, high-speed reconfigurable computation units are required in processor design. However the reconfiguration overheads like interconnection cost and reconfiguration time reduce the benefits of reconfigurable processors. At the same time within the arithmetic operators, the speed of operations on binary data cannot be increased beyond certain limits because of the inherited carry propagation at any stage of the addition. In this section to address reconfiguration and computation time issues, a high-speed reconfigurable operator is proposed for multimedia applications. This operator provides reconfigurability at both the operation level (different multimedia oriented operations) and at the data size level (different pixel data sizes) through the use of multimedia oriented subword parallelism (SWP). Reconfiguration at this level does not increase the complexity of the interconnection network as well as no reconfiguration time is required. For better resource utilization, multimedia oriented subword sizes (8, 10, 12 or 16 bits) are considered rather than existing conventional subword sizes (8, 16 and 32 bits etc.)[11, 19, 20, 28, 56]. To increase the speed of different processing units, arithmetic operations are performed using a redundant or borrow save representation rather than the conventional binary (CB) representation. The BS representation allows

to use a carry propagation free addition in the arithmetic units. Compared to CB adders, carry propagation free adders increase the overall speed of the reconfigurable operator when performing different multimedia operations like sum of absolute value difference SAD, discrete cosine transform DCT etc. Moreover the SWP overheads when performing on BS data are less compared to CB data. For multimedia applications, this operator ensures reconfigurability with high resource utilization along with high-speed operations. The proposed multimedia operator can be used as a dedicated core (co-processor) to speedup multimedia processing [107]. For multimedia applications, the main processor transfers control to co-processor which will perform the computations on pixel data more efficiently compared to conventional computational operators.

5.10.1 Architecture of the operator

Our reconfigurable multimedia operator can perform different basic and complex multimedia operations on data of different sizes (8, 10, 12, 16, 40 bits). Maximum parallelism with high resource utilization is attained through the use of SWP with these multimedia oriented subword sizes. Along with parallelism high-speed multimedia operations are performed through the use of BS representation. Efficient results are obtained for three stage pipelined architecture of this operator and is shown in Figure 5.18. This operator is made up of the basic units which we have presented above.

Control bits used to select the subword size are communicated to all the units which contain SWP capabilities. To clarify the schematic, these control bits are not shown in Figure 5.18. Through the use of this control word, this multimedia operator can be configured for both the computation it executes and the size of the data. In the beginning all the input data vectors are converted from CB to BS representation so that all the arithmetic computations can be performed in BS. The reconfigurable SWP operator can perform operations on signed as well as unsigned data values and gives the results in the required format. Based on the selected operation, the output of the reconfigurable operator can be in the form of a subwords or single accumulated value. After performing computations on BS values, the output is converted back to CB representation. As different operations produce different data length outputs, therefore different data length simple and SWP *BS to CB converter* are used in the operator design. Like the inputs, the output data of all the basic SWP units can be represented by subwords packed in 40-bit registers except for *SWP BS ($a \times b$) unit* whose output consists of subwords packed in a 80-digit register. These 80 digits can be further converted to 80 bits using *BS to CB converter*. As the output data length is limited to 40 bits, therefore the 80-bit product is divided into 40-bit MSB and LSB parts using *SWP product subwords LSB and MSB extractor* unit. Hence the complete product is obtained at the output of the

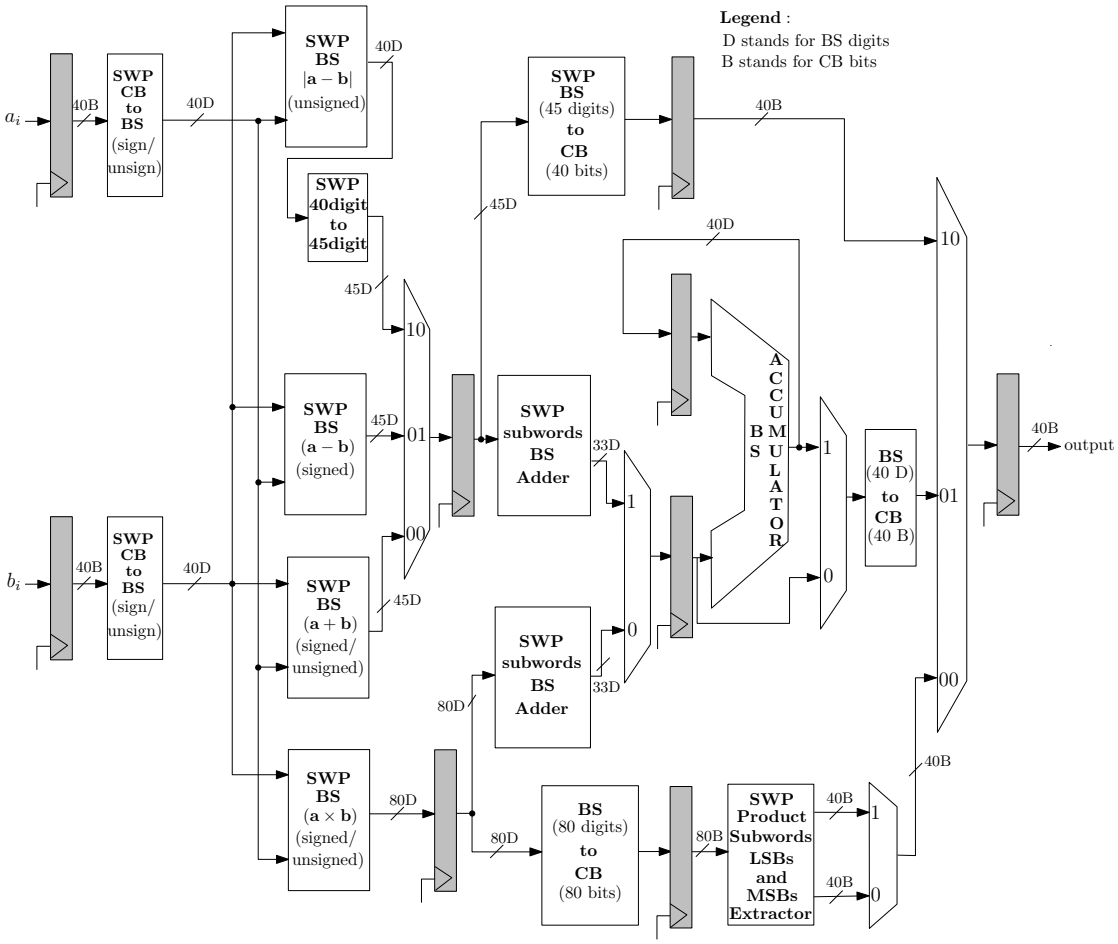

FIGURE 5.18: Reconfigurable multimedia operator

reconfigurable operator in the form of subwords LSBs and MSBs in two successive clock cycles.

In addition to basic arithmetic operations (signed/unsigned), the reconfigurable operator can perform multimedia operations like SAD for motion estimation, dot product for DCT, $\sum(a+b)$ signed/unsigned, $\sum(a-b)$ signed/unsigned etc. Based upon the requirements, any combination of these operations can also be obtained such as $\sum(a \times b) + \sum(a+b)$ etc. Rather than subwords which sometimes provide loss of bit, these operations produce lossless single accumulated value at the output of the reconfigurable operator. If the accumulated value from the *SWP BS accumulator* is small, it is zero padded to 40 digits.

5.10.2 Sum of products using reconfigurable operator

As an example, consider the computation of the multiplication-accumulation used for dot product is given by Equation 5.9.

$$\text{dot product} = \sum_{i=0}^{N-1} (a_i \times b_i) \tag{5.9}$$

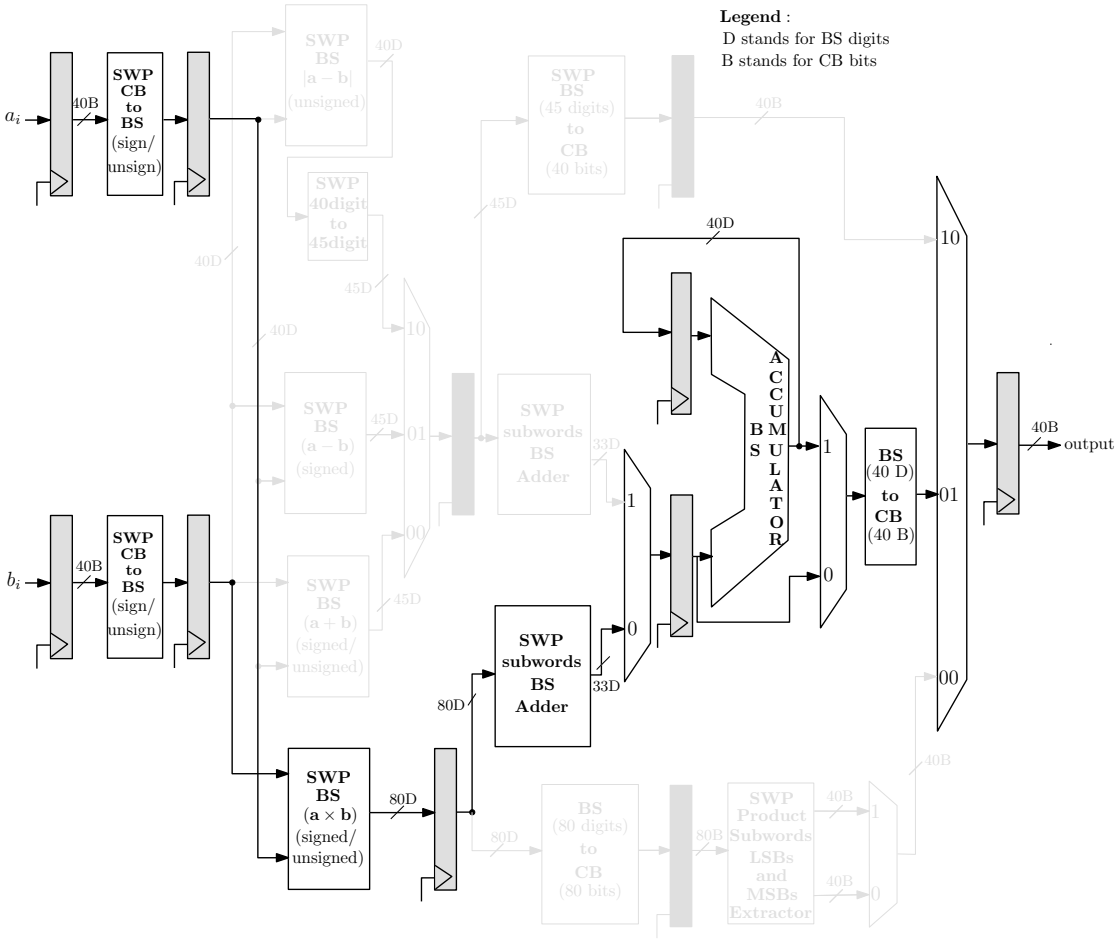

FIGURE 5.19: Sum of product computation using high speed reconfigurable operator

The computation of sum of product using high speed SWP reconfigurable operator is shown in Figure 5.19. The inputs are two 40-bit values and the selected subword size which is assumed to be 8 bits for this case. Hence each input vector contains five 8-bit packed subwords. First of all the input CB values are converted to their corresponding BS representation using *SWP CB to BS* units. Then the *SWP BS (a×b)* unit produces a 80-digit product value in the form of five 16-digit product subwords. These packed product subwords are added together using the *SWP subword BS adder* unit and generate a 33-digit value. This 33-digit data length is selected based upon the worst case of 16-bit subword size for $\sum(a \times b)$ operation with no digit loss. For other subword sizes and operations, the data length requirements at the output of the *SWP subword BS adder* units are less. At each clock cycle the *BS accumulator* accumulates the 33-digit value with the previous values to generate a 40-digit $\sum(a \times b)$ term at the output. The input to the *BS accumulator* is a 33-digit value and the output of the *BS accumulator* is a 40-digit

accumulated value resulting in extra seven digits. These extra seven digits are used as guard digits to avoid overflow. For other operations and smaller subword data sizes, the numbers of guard digits are greater and thus the number of accumulations which can be performed increases further. At the output, 40-digit sum of product value is converted to binary representation using *BS to CB* unit. Similarly other multimedia operations can be performed by SWP reconfigurable BS operator by activating the required units.

5.10.3 Synthesis results

For the analysis of the area, speed and power consumption, the overall SWP reconfigurable BS operator has been synthesized to ASIC standard cell 130nm and 90nm technologies using Synopsys *Design Compiler*. The area, speed and power consumption have been obtained for both target technologies.

In order to get the best possible clock frequency for the reconfigurable operator, synthesis were performed for different clock periods on each ASIC technology and the results are considered for those clock frequencies which give a high efficiency (smallest product of gates, clock period and consumed dynamic power). On both target technologies, due to the use of the carry propagation free adder, the clock frequencies that give maximum efficiency are higher compared to the same SWP multimedia operator using the CB representation (clock periods are 6ns on 90nm ASIC technology and 10ns on 130nm ASIC technology). The maximum design resources are consumed by the SWP BS multiplier which consumes almost 50% to 65% of total area and 55% to 60% of total power on different target technologies. Due to this reason we were able to increase the flexibility of the reconfigurable operator by adding other arithmetic operators without increasing the area to a larger extent. As a result the operator can perform variety of multimedia operations depending upon the requirements.

Figure 5.20 shows the comparison of the area, clock period and dynamic power of the SWP multimedia operators when using CB and BS representations. On all target technologies, the clock frequency of the SWP operator using BS representation increases due to the use of high-speed BS arithmetic units. The area overhead for this high-speed is mainly due to the redundant arithmetic units, conversion units and additional glue logics used in BS operators. The area of CB operator is less than BS operator. However to achieve the throughput of BS operator, multiple CB operators are required in parallel. Which ultimatly increases the area accordingly. Therefore to target the high throughput constraints, the area requirements of parallel CB operators are more compared to the BS operator. The probabilistic dynamic power overheads of BS operator correspond to the increase in the number of gates. On both ASIC technologies, the energy (power×

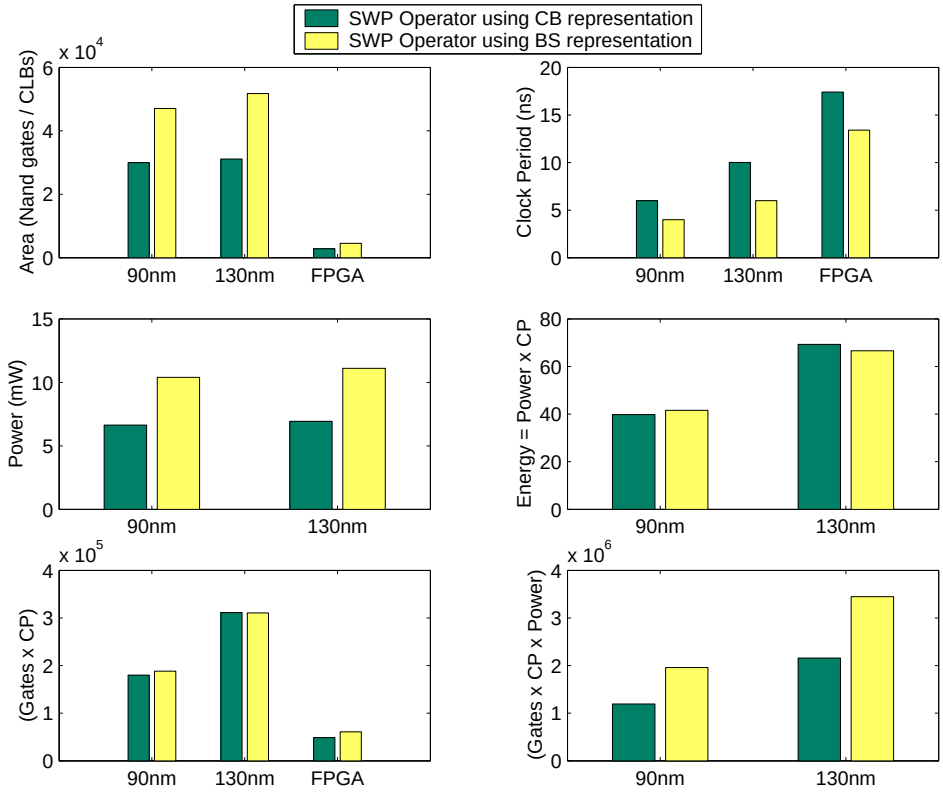

FIGURE 5.20: Comparison of SWP operators using CB and BS representations

clock period) requirements of SWP CB and SWP BS operators are almost same with the minor difference of less than 5%. With almost same energy consumption, the overall speed of SWP BS operator is almost 33% and 40% faster than SWP CB operator on 90nm and 130nm ASIC technologies respectively. On different target technologies, the (gates×CP) product of both CB and BS operators are almost same. With the same (gates×CP) product value, the BS operator provides high speed operations at the cost of some area increase. On both ASIC technologies, the product of area, clock period and dynamic power of both CB and BS SWP operators are also compared. The value of this product term is more for SWP operator using BS representation because it requires more area and consumes more power.

5.10.3.1 Power analysis

To perform any particular multimedia operation, only the required units are enabled. All the remaining units are disabled to reduce the switching activity. On the 130nm ASIC technology, the percentage of total power consumed by the SWP reconfigurable BS operator (with 8-bit selected subword size and clock period of 6ns) to perform different SWP operations is shown in the Figure 5.21. During these experiments, statistical power

is estimated by monitoring the switching activity on each node while performing different operations on random test vectors.

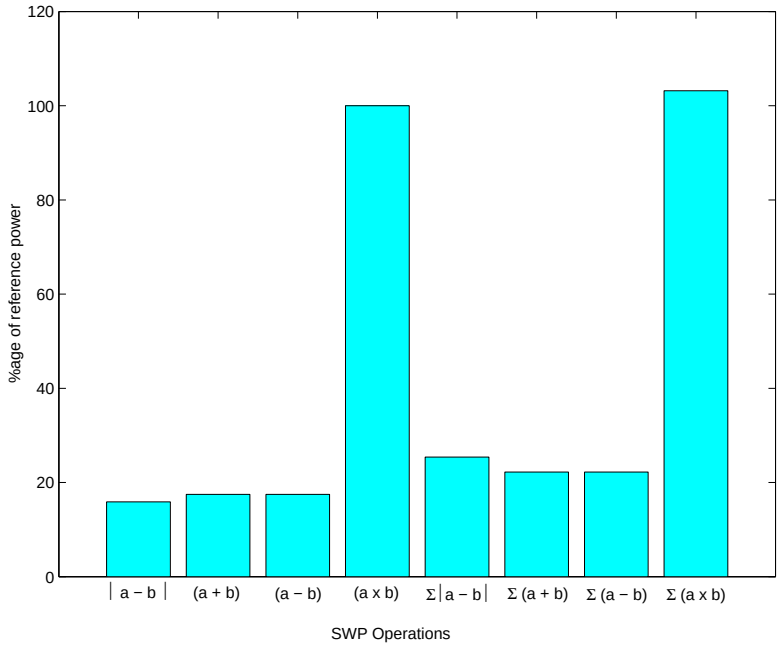

FIGURE 5.21: Power consumption of operations

We assume reference power (100%) is consumed by the more complex basic operator i.e the multiplier $(a \times b)$. All the SWP operations consume some percentage of reference power based upon the power consumption of the arithmetic units needed to be enabled for a particular operation. Obviously the power consumed by the complex operations which involve the accumulation is slightly larger compared to the operations which involve basic SWP arithmetic units. Additional power is mainly due to subword adder units. Maximum power consumed by the reconfigurable operator is 104% of the reference power when performing $\sum(a \times b)$ operation.

Compared to the SWP reconfigurable operator using binary representation (Figure 4.14 in Chapter 4), the reconfigurable BS operator consumes more power. This increase in power corresponds to increase gate count in BS architectures. Due to the difference in the internal architectures of processing units, the percentage power consumed by different operations in CB and BS reconfigurable operators is different. The percentage of reference power consumed by addition, subtraction and absolute difference based operations (a $\pm$ b, |a - b|, $\sum(a \pm b)$, $\sum |a-b|$ etc.) is different in both BS and CB operators. In CB operator, these operations consume almost 40% to 50% of reference power. Whereas in BS operator these operations consume almost 20% to 30% of reference power. This small variation occur mainly because of the internal architecture of arithmetic units in both operators and also because of the fact that reference power in both operators are

different. As the BS multiplier consumes more power therefore the percentage of power consumed by other operations is only few percent of reference power.

On average, the area overhead due to the use of BS number system in different units like ADD, SUB, MULT, absolute value of difference etc. is between 40% to 50%. Extra area is required due to the increase in the number of registers required for BS operator. Compared to the CB operator, the BS operator requires almost twice number of registers as each digit is represented by two bits. However the corresponding speed increase for basic SWP BS units varies between 60% to 80%. Due to these high-speed basic arithmetic units, the computation time of multimedia operations is also reduced by the same percentage when operating on different pixel sizes.

5.11 Conclusions

Efficient reconfiguration along with high-speed arithmetic units improves the performance of processors for several multimedia applications. The benefits of both parallelism and high-speed computation can be combined in the operator design by using multimedia oriented SWP on BS representation. In SWP, supported subword sizes that are in coordination with the pixel size of multimedia applications can further improves the performance through better resource utilization. Our work in this chapter shows that the speed of almost all the SWP arithmetic units used in the operator design can be improved by using the barrow save representation rather than conventional binary representation. The cost for this speed enhancement is the increase in area and power consumption. Due to this increase the overall product of area, CP and power also increases accordingly. However the speed of the processing units plays such an important role in modern multimedia applications that these overheads can be tolerated to some extend.

Chapter 6

Motion estimation using SWP operators

In the previous chapters different multimedia oriented SWP operators have been described. These operators perform operations on pixel data and increase the efficiency of processor through a better resource utilization. In this chapter the SWP operating units are used to design the dedicated hardware architecture for motion estimation algorithm. Motion estimation is selected because of its computational complexity and its importance in almost all the video compression standards. Instead of using classical subword sizes, multimedia oriented pixel sizes are used in our SWP motion estimation operator. The performance of SWP operators is analyzed by using it with different search algorithms of motion estimation.

The rest of this chapter is organized as follows: Section 6.1 gives the brief overview of motion estimation algorithms used in video compression. Section 6.2 presents different search algorithms which are used to find the best match in motion estimation. Section 6.3 describes the cost functions which are used to compare the blocks in the motion estimation algorithm. Section 6.4 presents the implementation of motion estimation using SWP operators. Each block of the *SWP ME* operator is described in detail. Section 6.5 and Section 6.6 describe the working of the *SWP ME* operator when using full search and diamond search algorithms respectively. Different hardware requirements and search time requirements are also discussed in detail. Section 6.7 elaborates the comparison of the *SWP ME* operator when using different search algorithms. Section 6.8 describes the use of *SWP ME* operator as co-processor IP using MicroBlaze soft processor core environment. Finally we conclude the chapter in Section 6.9.

6.1 Motion estimation

To efficiently utilize the video channel bandwidth, video compression is one of the most powerful technique. In video compression, temporal redundancies between the frames are removed and minimum possible information is sent on the channel for the transmission of video images. To remove these temporal redundancies motion estimation is one of the most efficient tool [62, 103]. Now a days motion estimation is used in most of the video compression algorithms. In motion estimation, candidate block in the current frame which needs to be transmitted is compared with different blocks in the reference frame and best match is searched. Instead of transmitting whole block, the difference between the candidate block in the current frame and best match block in the reference frame is transmitted [33, 47]. The location of the best match block in the reference frame is measured in terms of motion vector. At the receiving end motion vector indicates the location of best matched block in the reference frame. This process is called motion estimation which allows to transmit the video information using very small amount of data transfer [68].

Block matching is the most popular motion estimation technique. It works on the blocks of image rather than on individual pixels. The process of block matching used for motion estimation is shown in Figure 6.1.

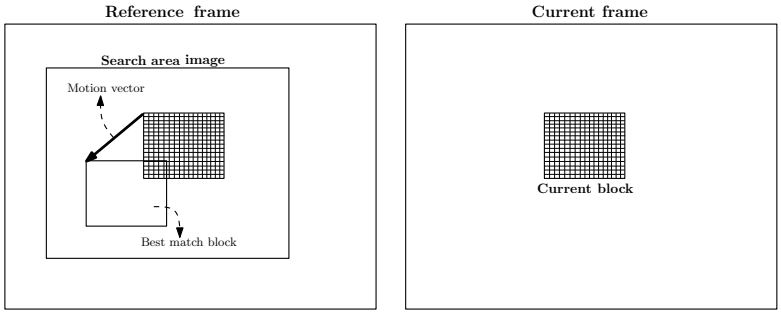

FIGURE 6.1: Block matching in motion estimation algorithm

As shown in Figure 6.1, *current block* is the candidate block in *current frame* which needs to be transmitted. *Current block* is compared with the blocks in *reference frame* using search algorithm. Search algorithm gives best match block that gives minimum cost function output in comparison process. When best match block is found its location is represented by motion vector. Different terminologies which are related to the motion estimation and are used frequently in this chapter are given below.

- **Current frame :** It refers to the current video frame which needs to be transmitted through video channel. For instance PAL and SECAM system recommends the frame rate of 25 frames per second.

- **Current block :** It refers to the block in current frame whose best match needs to be found using motion estimation algorithm. The dimension of current block is measured in terms of number of pixels. Based on the requirements the current block can be of any size like (16×16), (8×8) etc. The block size highly effects the performance of motion estimation algorithm.
- **Reference frame :** It refers to the previous frame in which the best match for current block is searched. Reference frame is not always fixed through out the communication. Reference frame is updated after certain period of time to increase its resemblance with the current frame. Greater the resemblance between current frame and reference frame, easier will be for search algorithm to find the best match.
- **Search area image :** In order to reduce the search time, the best match for current block is searched only in a particular window of reference frame and not in the whole reference frame. This window is called as search area image. Based on the requirements this search area image can be of any size. For instance for (16×16) current block the search area can be (48×48) or (64×64) etc.
- **Block base address :** It refers to the address of particular block in search area image. Base address of a block indicates the address of top left pixel of the block. For instance in (48×48) search area image there are numerous (16×16) blocks which are indicated by their base addresses.
- **Motion vector :** When the block matching algorithm is applied and the best match for the current block is found in the reference frame then its location is indicated by motion vector (MV). MV indicates both horizontal and vertical coordinates of matched block in search area image. The center of search area window is considered to have zero MV coordinates. MV usually gives the coordinates of top left corner of matched block.

The overall performance of motion estimation algorithm depends on two main factors:

- Search algorithm
- Cost function

Efficiency of both these units has enormous impact on the overall performance of motion estimation process [13, 17]. In the next sections we will explore different available search algorithms and cost functions and their efficiency and accuracy is analyzed. After the analysis, appropriate search algorithms and cost functions are chosen for the hardware implementation of motion estimation using our SWP multimedia operators.

6.2 Search algorithms in motion estimation

The search algorithm for finding best match plays vital role in the efficiency and accuracy of motion estimation [33, 66]. Search algorithm tries to find best possible match of current block in the reference frame with minimum possible resources and time. There are different algorithms available for finding best match in motion estimation. Some of these algorithms are given below:

- Full search algorithm
- Three step search algorithm
- Diamond search algorithm

6.2.1 Full search algorithm

Full search (FS) is the most accurate search algorithm which finds the match of current block in reference frame with high level of precision [65, 67, 78]. In FS the current block is compared with all the possible blocks in search area of reference frame. The number of possible blocks in search area depends upon the size of block and the size of search area image as well. For instance if the block size is (16x16) and search area image size is (48x48), then the number of possible (16x16) candidate blocks in (48x48) search area are given by Equation 6.1.

$$(p+1)^2 \;=\; (32+1)^2 \;=\; 1089 \tag{6.1}$$

Where

$$\text{p} \;=\; \text{Search area image size} \;-\; \text{Current block size} \;=\; 48 \;-\; 16 \;=\; 32$$

So in FS, each (16x16) current block has to compare with 1089 blocks in (48x48) search area window to find the best match. Although the FS algorithm gives very accurate match but due to the large number of comparisons it requires exhaustive computations which consumes lot of time. Compared to other search algorithms, the complexity of FS is less because the search pattern is always known in advance. In most other search algorithms, the search pattern depends upon the outcome of previous comparisons. The computations in FS are independent of any previous results. Therefore these computations can be performed in parallel to increase the speed. High speed can be achieved by exploiting the parallelism at both pixel level using SWP and at block level using multiple comparison units.

6.2.2 Three step search algorithm

To reduce the search time, three step search (TSS) algorithm compares only few blocks of search area image with the current block [49, 61]. In TSS algorithm, search process starts with nine search points centered around the zero motion vector (MV) point. Each of these search points are separated from next point by four pixel locations. The cost function (sum of absolute difference SAD or mean value of absolute difference MAD etc.) is calculated at each search point. For each calculation of cost function the top left corner of current block is coincide with the search point under consideration. The value of cost function indicates the difference between two blocks. Smaller the cost function value, lesser will be the difference between two blocks and vice versa. Out of the nine search points, the search point with minimum difference will be the center point for second search step. In the second search step, search pattern consists of nine search points centered around the minimum difference point of first step. These search points are separated from each other by two pixel locations. The cost function is calculated at each search point and the search point with minimum difference will be considered as the center point for next search step. Similarly in the third search step the cost function is calculated at nine search points separated by one pixel location and centered around the minimum difference point of second step. In this search step the point with minimum difference is considered as best match point. The coordinates of best match point are the required MV coordinates which are transmitted. The search steps of TSS are shown in Figure 6.2.

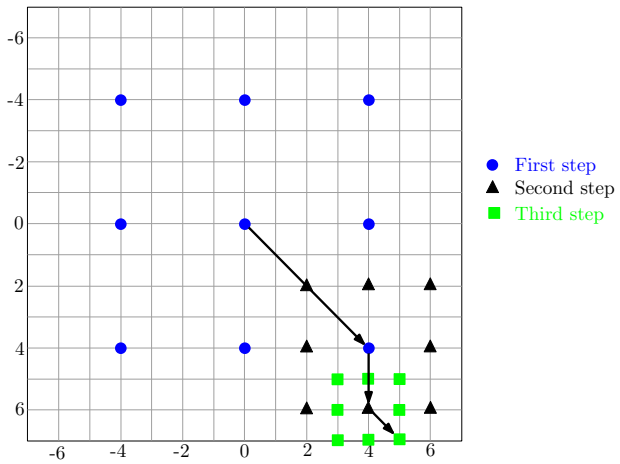

FIGURE 6.2: Three step search algorithm

In TSS the number of search points are very less compared to FS algorithm. Therefore the TSS algorithms finds MV in less time compared to FS. However the accuracy of TSS is less compared to FS because the search is carried out at few points rather than at all the points in search area image. For larger search area the number of search steps in

TSS can be increased to increase the accuracy of search process [76]. The complexity of TSS is higher than FS because the center point for next search pattern is based on the minimum search point in previous step. Where as in FS search pattern is always fixed.

6.2.3 Diamond search algorithm

To increase the accuracy of search process, diamond search (DS) was proposed [108]. The search patterns in DS are selected in such a way that the search process should not be trapped in wrong directions. In DS there are two types of search patterns that are large diamond search pattern (LDSP) and small diamond search pattern (SDSP). LDSP consists of nine search points and SDSP consists of five search points. Both LDSP and SDSP are in the shape of diamond. Search process starts with LDSP centered around the zero MV. Cost function is calculated at all the nine search points of LDSP. The search point with the minimum difference (smaller value of cost function) determines the location of center point of the search pattern in the next step. If the minimum difference comes at any point other than the center point of LDSP then in the next step the search patterns remains LDSP. This process continues until the minimum difference comes at the center point of LDSP. When the minimum difference point comes at center of LDSP then the search pattern is switched from LDSP to SDSP centered around the minimum difference point. In SDSP cost function is calculated at all the five search points and the coordinates of search point with minimum difference value is considered as final MV. The search pattern for DS is shown in figure 6.3.

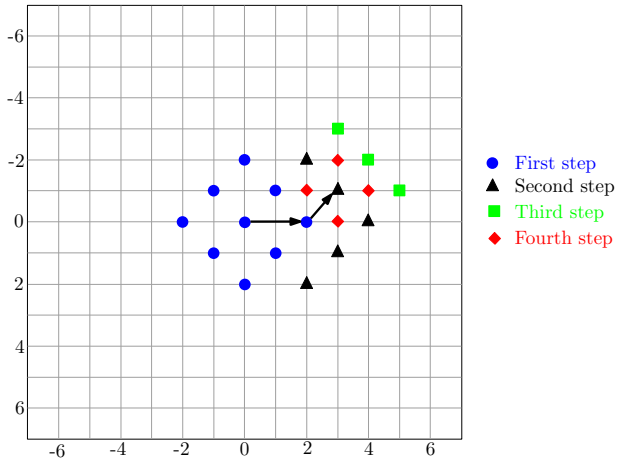

FIGURE 6.3: Diamond search algorithm

The number of search points in DS are less compared to FS. Therefore DS requires less time to find MV compared to FS. The accuracy of DS is higher than TSS but less than FS. The accuracy of DS algorithm is high because of the use of two different diamond shape search patterns (LDSP and SDSP). The number of iterations of LDSP

are determined at run time and totally depends on the features of current block and search area image. In the final step SDSP is used to narrow down the focus for more accurate MV determination. Unlike the FS and TSS, the time required to find MV using DS algorithm is not fixed. The search time in DS depends upon the number of iteration of LDSP. Search time will be smaller if the minimum difference point comes quickly at the center of LDSP and vice versa. The complexity of DS is high compared to FSS and TSS because of different search patterns used and also because of unknown number of iterations of LDSP.

6.3 Cost functions

A Cost function is used to compare and estimates the difference between current block and the blocks in the search area image. Smaller value return by the cost function indicates the high similarity between the blocks and vice versa. There are different cost functions available to estimate the differences between the blocks. These cost functions are based on different mathematical functions which tries to find the differences as accurately as possible [15]. Some of the cost functions are Sum of absolute value difference (SAD), Mean absolute value difference (MAD), sum of absolute transformed differences (SATD) [82] etc. However in our SWP operator for motion estimation, we will use SAD as cost function due its simplicity and efficiency.

6.3.1 Sum of absolute value difference SAD

SAD is the most commonly used cost function in motion estimation because of its simplicity. The mathematical expression for the computation of SAD on image blocks is given by Equation 6.2.

$$\mathrm{SAD} = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |\mathrm{I_n(k+i,\ l+j)} \ - \ \mathrm{I_{n-1}(k+i+u,\ l+j+v)}| \tag{6.2}$$

Where M and N represents the horizontal and the vertical size of block respectively. For instance the value of both M and N is 16 for the block size of (16×16). The current block is compared with the block in search area window whose location is represented by (u,v). I_n(k+i, l+j) is a pixel at (k+i, l+j) in the current frame and I_{n-1}(k+i+u, l+j+v) is a pixel at location(k+i+u, l+j+v) in the reference frame. In SAD, the comparison of blocks is based upon the values of pixels in both current block and reference image block. SAD operator computes the absolute values of differences between the corresponding

blocks. The computation of absolute value of difference at any pixel location depends only on the pixels at that particular location in both blocks. Due to the non dependence of computation, the absolute value of difference computations can be made in parallel inside the blocks using subword parallelism SWP. Absolute values are then added to give the overall difference between two blocks. The pipelined architecture of the SWP SAD unit which will be used in our SWP motion estimation operator is shown in Figure 6.4.

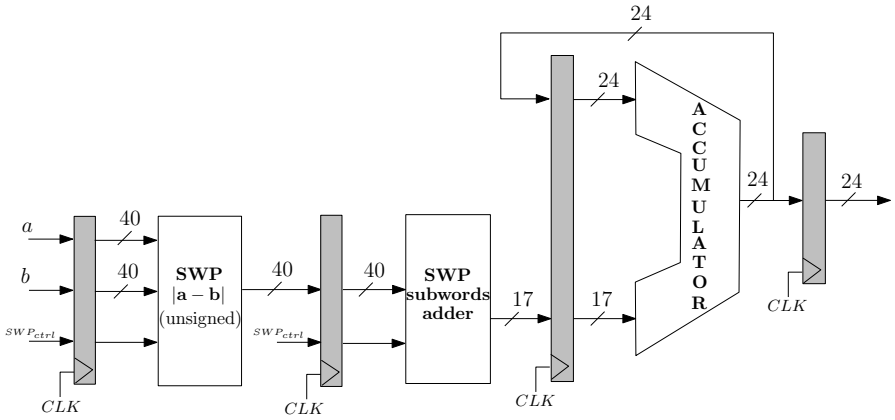

FIGURE 6.4: SWP SAD unit

The detail of each building block of SWP SAD unit shown in Figure 6.4 have already been given in Section 3.3 of Chapter 3. SWP SAD operator perform computations on pixels which are packed in the word size (40-bit) input registers. Instead of working on a single pixel, SWP SAD unit operates on all the packed pixels in parallel using subword parallelism. Smaller the size of the selected subword higher will be the parallelism. For SAD calculation on block size of (16x16), the SWP SAD operator requires approximately 52 clock cycles for subword size of 8-bit. In each clock cycle instead of working on single pixel SWP SAD unit perform computations on five 8-bit pixels in parallel.

The output of SWP SAD unit consist of 24 bits. These 24 bits are selected based upon the maximum current block and subword sizes used. In our *SWP motion estimation* operator, we have considered maximum block size of (16×16) and maximum subword size of 16-bit. For 16-bit subword size, *SWP* $|a$ - $b|$ unit performs absolute difference computations on two 16-bit pixels in parallel in each clock cycle. The packed absolute difference values are then added using *SWP subwords adder* unit and generate 17-bit result. These 17-bit results are accumulated recursively by the accumulator. Without SWP, SAD computations on (16×16) block require the accumulation of 256 absolute difference values. But due to the use of SWP the number of these accumulations are reduced by the degree of parallelism. For maximum subword size of 16-bit, the accumulations are reduced by the factor of 2 (256/2 = 128). For other smaller subword sizes

these accumulations reduces further due to more parallelism. To add 128 values, accumulator requires 7 guard bits ($2^7 = 128$) to avoid any overflow. Therefore for maximum subword and block sizes, the total number of bits required at the output of accumulator is 24 bits ($17 + 7 = 24$). These bits are sufficient to represent the SAD values of other smaller subword sizes (8, 10, 12-bit) and also for other smaller block sizes. However based upon the requirements, bit width at the output of SWP SAD unit can further be increased when more larger block sizes are considered.

6.4 Motion estimation using SWP operators

SWP ME operator is design to perform the motion estimation computation for different multimedia oriented pixel sizes. This operator increases the performance of processor through better resource utilization. *SWP ME operator* can operate on 8, 10, 12 or 16-bit pixel sizes. These pixel sizes are chosen based upon the modern multimedia applications. Different SWP units used in the design of *SWP ME operator* performs the computations on pixels packed in 40-bit word size registers. The word size of 40-bit is chosen as it gives the better efficiency trade off with different multimedia oriented pixel sizes. The inputs to *SWP ME operator* are *current block* from current frame and the *search area image* from reference frame. The *current block* and *search area image* are stored in input random access memories (RAMs). SWP SAD is used as cost function to determine the match between the current block and the blocks in search area image. *SWP ME operator* gives *motion vector* and *global minimum SAD value* as the output. The *SWP Ctrl* is used to select the subword size based on the size of input pixels. The selected subword size is communicated to different units which utilizes SWP capabilities. Two search algorithms (full search and diamond search) are used to analyze the performance of proposed *SWP ME operator*. These search algorithms are chosen based upon their accuracy and computational requirements. The block diagram of *SWP ME operator* is shown in Figure 6.5.

As shown in figure 6.5, there are different blocks which contributes to the overall computations of motion estimation algorithm. The internal architecture of each block is pipelined to increase the speed of ME operation. Clock (CLK) signal and synchronous reset (Reset) signal is communicated to all the required blocks. In the next subsections the functionality of each block is explained.

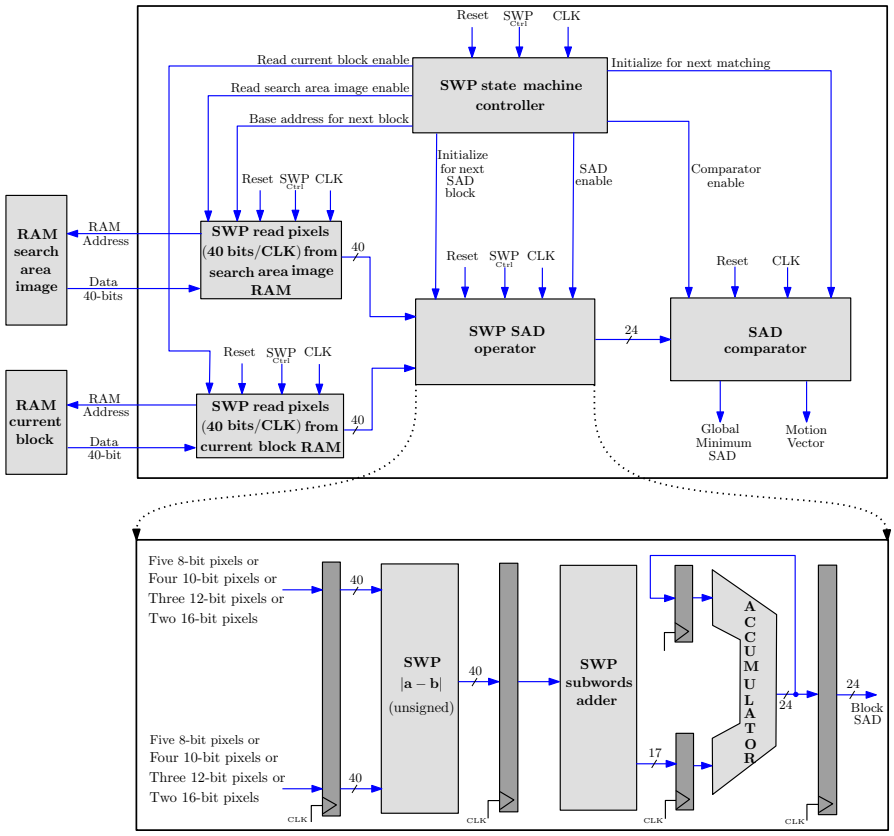

FIGURE 6.5: SWP motion estimation (ME) operator

6.4.1 RAMs for search area image and current block

The input to *SWP ME operator* are pixels from search area image and pixels from current blocks. These pixels are stored in two input RAMs that are *RAM search area image* and *RAM current block*. Based upon the requirements both RAMs can be of any appropriate size. For instance some of the appropriate sizes which are commonly used in video standards are (48×48) for search area image and (16×16) for current block. The output port of each RAM has data width of 40 bits. The output from RAMs consists of pixels packed in 40-bit word length. Based on the selected subword size, the 40-bit output data from each RAM consist of either five 8-bit pixels or four 10-bit pixels or three 12-bit pixels or two 16-bit pixels. The input to each RAM is the RAM address from where the data is required to be read. The data in the RAM at the location mentioned by RAM address is loaded on the output port.

6.4.2 RAM reading units

RAM reading units are used to read the pixel data stored in search area and current block RAMs. RAM reading units generates the input addresses for the RAMs. The

controller unit enables or disable RAM read units based on the requirements. In each clock cycle reading unit generate the address of next pixels required for computation. For instance for subword size of 8-bit, five pixels (40 bits) are read from RAM in each clock cycle. So the RAM reading unit generates RAM address with the increment of five pixels in each clock cycle. These five pixels from search area RAM and current block RAMs are given to SAD computation unit in each clock cycle. Once activated, the RAM read units are enabled until all the pixels from the required blocks are read. For (16×16) block size and (48×48) search area size, each RAM read units read 256 pixels for SAD computations on the blocks. For current block RAM, the entire (16×16) RAM is read for the SAD computations of (16×16) blocks. However for search area RAM, the reading unit requires the base address of next (16×16) block to be read for SAD computation. Starting from the base address, the reading unit for search area image generate the addresses for (16×16) block. As the 40-bit output of RAMs contains different number of pixels for different subword sizes (five 8-bit pixels, four 10-bit pixels, three 12-bit pixels and two 16-bit pixels). Therefore for different subword sizes, the reading units requires different number of cycles to read (16×16) pixel blocks. For 8-bit subword size, maximum number of pixels (five pixels) are read in each clock cycle hence it requires minimum number of clock cycle to read entire (16×16) block. For other subword sizes the number of cycles required to read (16×16) block increases corresponding to the number of pixels read in one clock cycle.

6.4.3 SWP SAD computation unit

SWP SAD computation unit is used to compute SAD on pixel data. The inputs to this unit are *CLK*, *Reset*, *SWP ctrl* (SWP control signal), *SAD enable*, *Initialize for next block SAD* and two 40-bit vectors. The input and output signals of *SWP SAD* unit are shown in Figure 6.6.

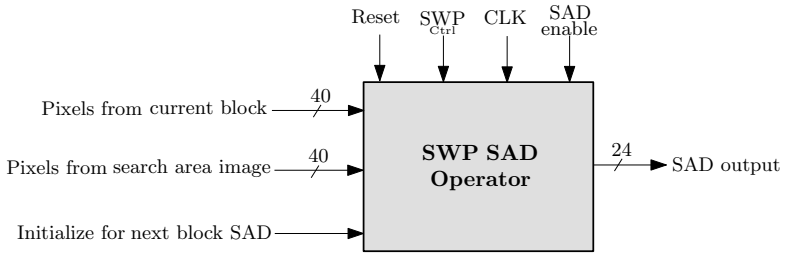

FIGURE 6.6: Block diagram of SWP SAD unit

The controller unit enables the *SWP SAD* unit when the SAD computations are required on pixels. In each clock cycle the 40-bit vectors from current block RAM and search area RAM are given to SWP SAD unit. These 40-bit vectors contain the packed pixels.

SWP ctrl signal directs the *SWP SAD* unit about the size of pixels stored in input 40-bit vectors. *SWP SAD* unit computes the SAD ($\sum |a-b|$) on the pixels packed in its input vectors. The partial SAD values are accumulated until all the pixels in the input blocks are finished. The pipelined architecture of SWP SAD unit is shown in Figure 6.4. The block size in ME determines the number of pixels on which SAD computations are applied. For instance for (16×16) block size, the *SWP SAD* unit computes the SAD on (16×16) current block and (16×16) block taken from search area image. Before each (16×16) SAD computation, the controller directs the *SWP SAD* unit to get ready for new SAD computation by sending *Initialize for next block SAD* signal. On receiving this signal the *SWP SAD* unit clears all the data in its pipelined registers and get ready for new SAD computation on new (16×16) blocks. As the input 40-bit vectors contain different number of pixels for different selected subword sizes. Therefore the number of cycles required to compute the SAD of (16×16) blocks also depends upon the selected subword size. The minimum cycles are required when selected subword size is 8-bit and the maximum cycles are required when selected subword size is 16-bit.

6.4.4 SAD comparator unit

SWP SAD unit computes the SAD on current block and the block in the search area window indexed by the base address. After required number of clock cycles *SWP SAD* unit gives the SAD value of complete block. This SAD output value is compared with the previously obtained minimum SAD value using *SAD comparator* unit. The inputs to this unit are *Current SAD value*, *Initialize for next matching*, *Comparator enable*, *CLK* and *Reset*. The block diagram of *SAD comparator* unit is shown in Figure 6.7.

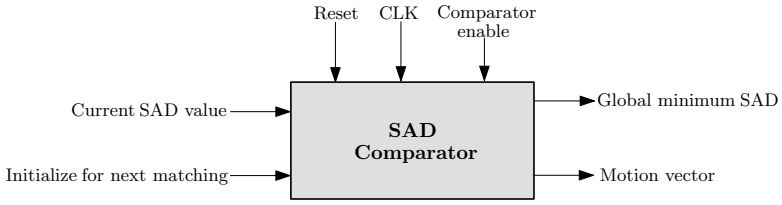

FIGURE 6.7: Block diagram of comparator unit

After the comparison, if the current SAD value is less than previously obtained minimum SAD value then the minimum SAD value is updated by assigning current SAD value to minimum SAD value. However if the minimum SAD value is less than current SAD value then the minimum SAD value remains the same. The comparator unit is enabled by the controller after each computation of SAD on particular block size. For instance for (16×16) block size the comparator unit is activated after each computation of SAD on (16×16) blocks. The minimum SAD value is stored internally after each comparison.

After the required number of comparisons which is determined by search algorithm, the comparator unit gives the *global minimum SAD value*. The location of block in search area image corresponding to the global minimum SAD value is indicated by *motion vector*. For block size of (16×16) and search area size of (48×48), the FS algorithm requires 1089 comparisons before giving *global minimum SAD value*. Other search algorithms like TSS and DS etc. computes less number of SAD values and ultimately requires less number of comparisons. The controller send *Initialize for next matching* signal to *SAD comparator* before the start of block matching for new current block. On receiving this signal, *SAD comparator* unit initializes the minimum SAD value register internally for the next block matching. Normally the initial value of this register is set at maximum which is then compared with actual current SAD values.

6.4.5 State machine controller

Controller unit generate different control signals for different modules in ME operator. It enables, disable or initialize the modules at the appropriate time. The controller unit is implemented using synchronous finite state machine (FSM). The state machine performs the transition between the states on clock edge only. In each state, controller generates enable signals for different units which need to be active to perform particular operations. After the completion of certain operations, the controller unit generates signals to initialize the unit for next operation. The control signals generated by FSM controller are shown in Figure 6.8.

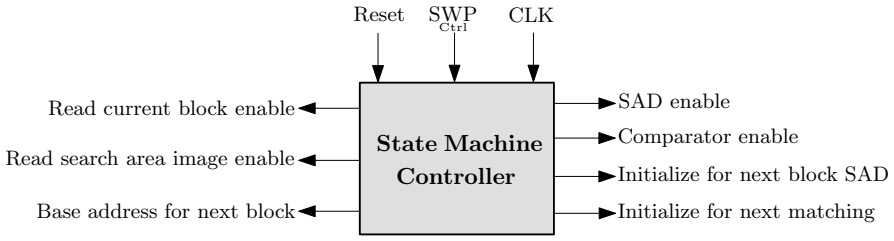

FIGURE 6.8: State machine controller block

As shown in the Figure 6.8, the FSM controller generates following control signals.

- **Read current block enable :** When this signal is activated it enables the reading of pixels from current block RAM. This signal remains active until the entire current block is read and the pixels are provided to SAD computation unit in successive clock cycles. When the read operation is over the controller disable the *Read current block enable* signal and the reading process is finished.

- **Read search area image block enable :** When this signal is activated, the pixels are read from search area image at the particular location indicated by the base address. This control signal is active in synchronize with *Read current block enable* signal so that the pixels from both current block and search area blocks are available for the computations at the same time. When all the pixels from search area block are read successfully then the controller disable the *Read search area image block enable* signal.

- **Base address for next block :** Depending on the search algorithm, the controller generates the base address of next block to be read from search area image. For instance in full search algorithm all the possible blocks in search area image are compared with the current block. Therefore after each computation of SAD on the blocks, the controller generates base address of next block in search area in the increment of one pixel location. This process continues until the entire search area image is scanned in both horizontal and vertical directions. For other search algorithm, the controller generates the base address in accordance with the search pattern used.

- **SAD enable :** This control signal enables the SWP SAD unit. When the pixels are ready for SAD computations, the controller activates the SWP SAD unit using *SAD enable* signal. During the entire computations of SAD on any particular block, this control signal remains active. When the SAD computations for the block are finished, controller disable the *SAD enable* signal.

- **Initialize for next block SAD :** This control signal is sent by the controller to *SWP SAD* unit before the start of new SAD computations. On receiving this signal, the *SWP SAD* unit initializes all its internal registers to zero value and get ready for the next SAD computations. Without this initialization, the unwanted values in the internal registers of *SWP SAD* unit can result in incorrect SAD value.

- **Comparator enable :** This control signal enables the comparator unit. After the SAD computation on each block, the controller activates the comparator unit to compare the current SAD value with the previously obtained minimum SAD value.

- **Initialize for next matching :** This control signal is sent by the controller to *SAD comparator* unit before the start of comparisons for new current block. On receiving this control signal, *SAD comparator* unit initializes the internal register which is used to store the temporary minimum SAD value. Without this initialization the first SAD value of the new current block is compared with the minimum SAD value corresponding to previous current block. Which would result in incorrect global minimum SAD value and motion vector.

The controller activates different units of *SWP ME* operator in appropriate sequence. The controller makes sure that input data should be ready before the activation of any particular arithmetic unit. The sequence in which the controller activates different units is shown in the Figure 6.9. The controller remains in the same state for required number of clock cycles until the activated unit finished its task. Whenever the transition between the states is required it is done at the positive edge of the clock.

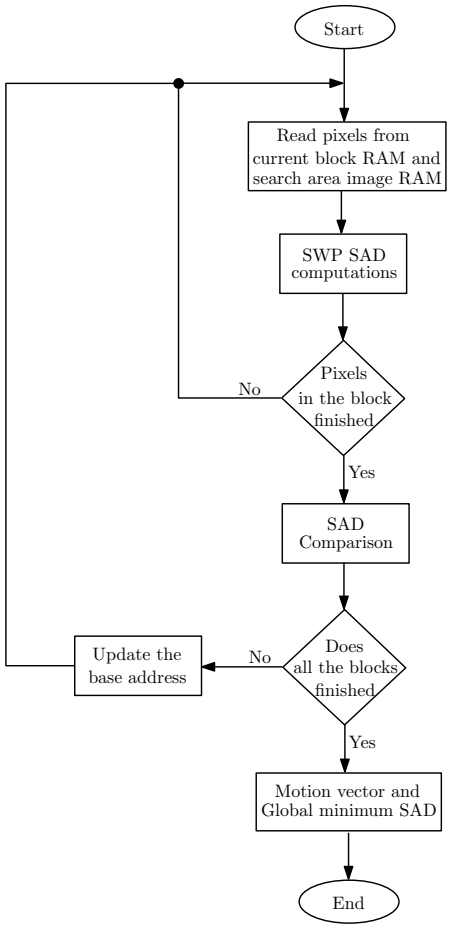

FIGURE 6.9: Sequence of operations in the SWP ME operator

As shown in Figure 6.9, after the start the controller activates the read RAMs units. Then the *SWP SAD* unit is activated to perform computation on the pixels. In each clock cycle based upon the selected subword size, the read unit reads particular number of pixels and *SWP SAD* unit performs computations on these pixels. This process continues until all the pixels in the block are finished. The number of cycle required for this operation depends upon the block size and the selected subword size. For block size of (16×16) and subword size of 8-bit, this loop is repeated 52 times ($256/5 \approx 52$). In practice the clock cycles due to initial pipeline latency, control cycles and subword alignments are also required. After the completion of SAD computations on the block, the comparator unit is activated which compares the SAD values for minimum SAD.

This sequence of operations continues with new base addresses until the best match of current block is found in search area image. The exact number of times this sequence is repeated depends upon the search algorithm. For instance in full search with block size of (16×16) and search area image of (48×48), this sequence is repeated 1089 times. After this the motion vector is given at the output ports of *SWP ME* operator.

6.5 SWP ME using Full search algorithm

The performance of *SWP ME* operator explained in Section (Section 6.4) can be analyzed by using it with different search algorithms. In this section the *SWP ME* operator is applied to find the motion vectors while using full search (FS) algorithm. FS algorithm provides good platform for analyzing the *SWP ME* operator's performance. FS algorithm requires exhaustive computations and gives highly accurate match of current block in the search area image. Due to the high computational requirements of FS, parallel architecture of different arithmetic and data transfer units of *SWP ME* provides sufficient speedup. During these experiments, motion estimation is applied on different pixel sizes through the selection of different subword sizes (8, 10, 12 and 16-bit). Current block size and search area image size plays very important role in the overall efficiency of motion estimation algorithm. These parameters determines the number of computations required for SAD computations and ultimately for overall motion vector determination. Different current block sizes of (16×16), (8×8) and search area image sizes of (48×48), (32×32) and (16×16) are considered during these experiments. *SWP ME* operator find the best match of current block in search area image of different sizes. The search time required to find best match and the hardware resource requirements corresponding to different block sizes and search area sizes are analyzed. Complexity of search algorithm determines the overall complexity of block matching in motion estimation. The complexity of FS is less which results in the simplicity of controller design that controls different operations. The impact of search algorithm complexity on the overall resource requirements in our *SWP ME* operator will also be analyzed.

Using our *SWP ME* operator, the FS algorithm is applied on famous *Lena image*. First of all the *Lena image* is reduced to the search area image size using MATLAB [1] and is used as search area image. In order to find the accuracy, random block of current block size is cropped from the *Lena image* and is used as current block. For this case the minimum SAD value corresponding to best match block will be zero. The pixel values corresponding to search area (48×48) and current block (16×16) are shown in Figure 6.10.

[1] Matrix laboratory : http://www.mathworks.com/

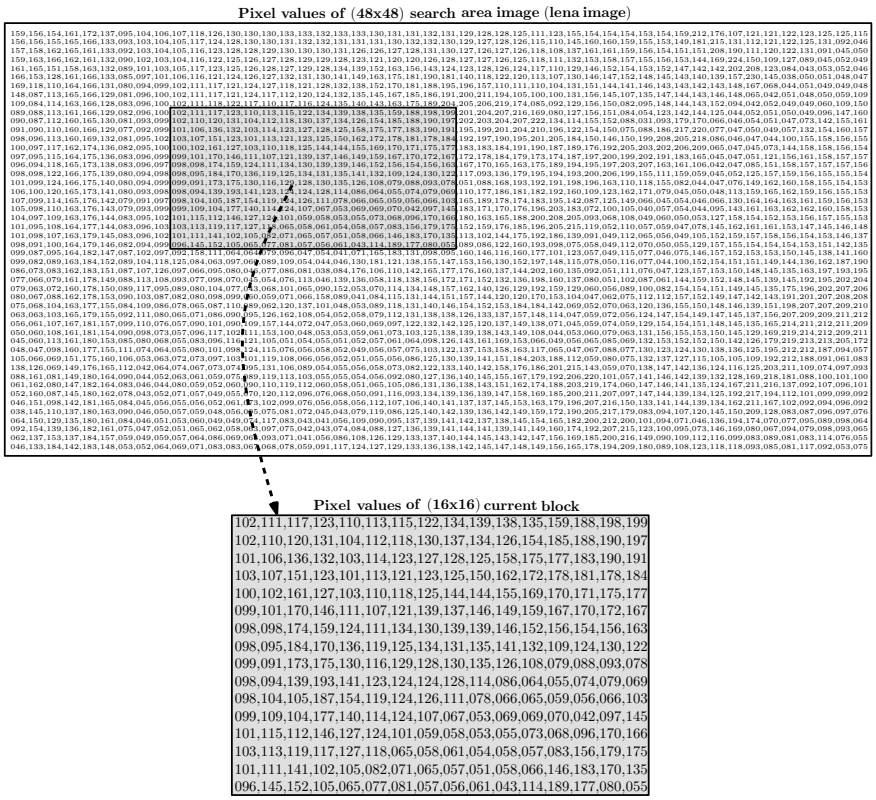

FIGURE 6.10: Pixel values of search area image and current block

Both search area image and current block are given as input to *SWP ME* operator. The performance parameters like search times, chip area, critical path and power consumption are analyzed. The comparison of *SWP ME* is done with simple ME architecture to highlight the advantages of using multimedia oriented SWP in ME design.

6.5.1 Search time

Search time (ST) is the total number of clock cycles required to find best match in search area. In FS algorithm, the current block is compared with each and every possible block in the search area image. Therefore the search time will be high compared to other search algorithms. In *SWP ME* operator the search time also depends upon the selected subword size. For smaller subword sizes, the search time is less. In case of smaller subword sizes, more number of pixels are packed in word size registers and hence more parallel SAD operations can be performed. For 8-bit subword size five 8-bit parallel SAD operations can be performed. Where as for 16-bit subword size only two 16-bit SAD operations can be performed in parallel using SWP ME operator. Due to this increased parallelism for smaller subwords, the overall search time decreases. For other greater subword sizes the search time increases accordingly.

Simulations are performed to find the time required by the *SWP ME* operator to find the best match using FS algorithm. *ModelSim tool* [34] is used as simulation tool. These simulations are performed for different selection of subword sizes, current block sizes and search area image sizes. The number of cycles required by the *SWP ME* operator to find the best match are shown in Table 6.1.

Search area image size	**Current block image size**	**Subword size** (bits)	**Cycles required for block matching**
(48x48)	**(16x16)**	8	77320
		10	77320
		12	112168
		16	147016
(32x32)	**(16x16)**	8	20519
		10	20519
		12	29767
		16	39015
(16x16)	**(8x8)**	8	1863
		10	1863
		12	2511
		16	3159

TABLE 6.1: Cycles required by the SWP ME operator to find the best match using FS algorithm

These cycles include pixel load cycles, SAD computation cycles, comparison cycles, control cycles etc. For all current blocks and search area sizes, the number of cycles required for smaller subword sizes are less compared to the number of cycles required for larger subword sizes. For all the subword sizes, the search time of *SWP ME* operator have been reduced compared to the simple ME operator. For current block size of (16x16) and search area size of (48x48), the simple ME operator requires 316070 cycles to find best match using FS algorithm. When the selected subword size is 8-bit or 10-bit, *SWP ME* operator requires almost 76% less cycles compared to simple ME operator. For 12-bit and 16-bit subword sizes, the *SWP ME* operator requires almost 65% and 53% less cycles respectively. This reduction in cycles occurs due to the parallel processing of packed pixels using SWP. For each set of block sizes, the number of cycles required for 8-bit and 10-bit subword sizes are same. The reason being that due to the selected current block size, the *SWP SAD* operator requires almost same number of cycles to process one row of pixels. However for other selection of current block sizes, 8-bit subword size will requires less number of cycles.

6.5.2 Synthesis results

To analyze the area, speed and power, *SWP ME* operator shown in Figure 6.5 is synthesized to ASIC technology (130nm and 90nm) and to FPGA (Xilinx Virtex II). The

controller unit in *SWP ME* enables or disables different units corresponding to the FS algorithm. Synthesis is performed while considering different block sizes and search area images. In order to precisely analyze the effect of block size and search area image size on the *SWP ME* operator, external RAMs used to store current block and search area images (shown in Figure 6.5) are not considered in synthesis process. As the area and other resources consumed by these RAMs increases with the increase of image size which can over shadow the effect of actual resource increase by *SWP ME* operator with the increase of block size or search area image size. The primary effect due to the change in block size or search area image size is the increase or decrease in the number of cycles required to find the best match (Section 6.5.1). Obviously when the search area image or current block size increases the *SWP ME* operator has to perform more computations so the search time increases accordingly. However the small change in the area, critical path (CP) and power consumption also occurs due the change in block sizes and search area image sizes. Table 6.2 shows the synthesis results of SWP ME operator for different block sizes and search area image sizes using full search algorithm.

Search area	**Current block**	**90nm CMOS ASIC**			**130nm CMOS ASIC**			**FPGA VirtexII**	
		Nand Gates	**CP (ns)**	**Power (mW)**	**Nand Gates**	**CP (ns)**	**Power (mW)**	**CLBs**	**CP (ns)**
(48x48)	**(16x16)**	5890	3.57	1.97	5765	5.82	2.2	1637	11.2
(32x32)	**(16x16)**	5734	3.56	1.92	5628	5.79	2.1	1593	10.7
(16x16)	**(8x8)**	5492	3.56	1.8	5387	5.8	2.1	1527	10.4

TABLE 6.2: Synthesis of SWP ME operator using FS algorithm

Each unit in the *SWP ME* operator consumes certain amount of resources. The maximum resources are consumed by the *SWP SAD* unit. *SWP SAD* unit almost consumes 64% of total area. Other units like *RAM reading*, *comparator* and *controller* almost consumes 11%, 9% and 12% of total area respectively. Compared to other units, the CP of the *SWP SAD* unit is more and the overall CP of *SWP ME* operator is mainly due to the *SWP SAD* unit.

As the search area image or current block size decreases, the area, CP and power either remains same or decreases slightly (less than 10%) on both ASIC and FPGA platforms. For performing ME computations on different block sizes, the internal architecture of *SWP SAD* unit remains same which constitute major portion of the *SWP ME* operator (64% of total area). Therefore, most of the design resources consumed by *SWP ME* operator remains same irrespective of block sizes. The slight decrease in resources occurs because for the smaller block sizes and search area sizes the controller has to load less number of blocks of smaller size for SAD computations. However there is not a substantial decrease in area, CP and power consumption due to the decrease in block sizes. Due

to the internal pipelining of different units used in SWP ME, the CP does not change too much with the increase of block size or search area image size. On ASIC technologies, the small variations in power consumption for different block sizes are mainly related to the small increase in the number of gates. To analyze the total effect of block sizes and search area sizes, we have to take into account the search time corresponding to different block sizes. The search time reduces with the decrease in either search area, current block or subword size.

6.5.3 Comparison of simple and SWP ME operator

To analyze the advantages of using SWP in ME operator design, the performance of *SWP ME* operator is compared with simple ME operator. *Simple ME* operator uses operating units of fixed size like 8-bit or 16-bit etc. It can perform operations on specified pixel size for which it is designed. For instance *simple 8-bit ME* operator can perform operations on pixels of 8-bit size only. Any multimedia application with pixel sizes smaller than 8-bit can also be executed on *simple 8-bit ME* operator. But for this purpose the pixels have to be extended to 8-bit before being processed. However the pixel sizes greater than 8-bit cannot be processed on *simple 8-bit ME* operator. More over *simple 8-bit ME* operator operates on only one 8-bit pixel in each clock cycle. The architecture of *simple 8-bit ME* operator is shown in Figure 6.11.

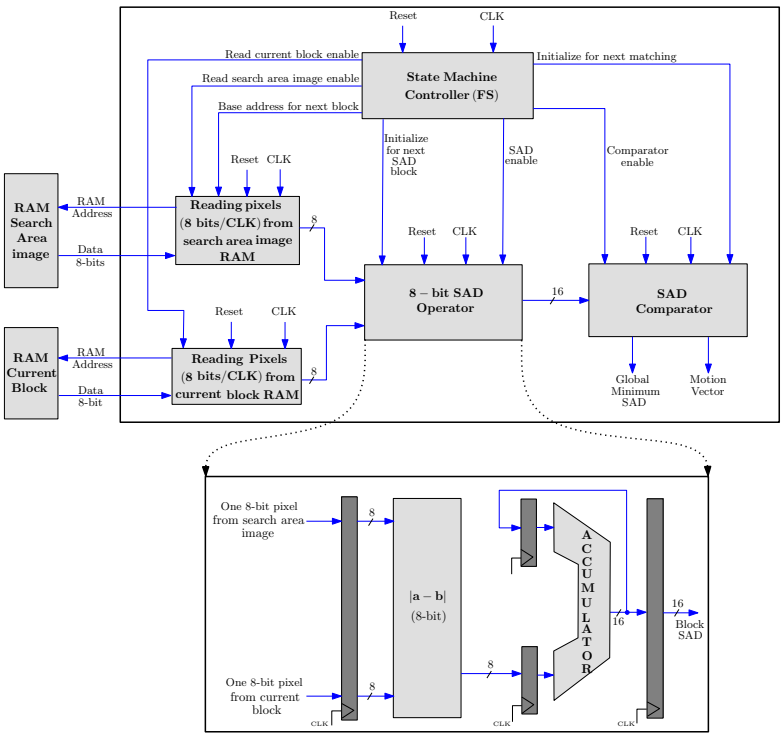

FIGURE 6.11: 8-bit motion estimation operator

As shown in Figure 6.11, *simple 8-bit ME* operator reads one 8-bit pixel from current block and search area image in each clock cycle. These 8-bit pixels are given to *8-bit SAD* unit which computes absolute value of difference and forward it for accumulation. This process continues until the whole block is finished. For (16×16) block size this process is repeated 256 times as there are 256 pixels in (16×16) block. As there are numerous (16×16) blocks in search area image so this SAD calculation is repeated for each block. In (48×48) search area there are 1089 (16×16) blocks which needs to be compared in FS algorithm. Therefore the SAD computation of block is repeated 1089 times.

On the other hand *SWP ME* operator can perform on multimedia oriented pixel sizes (8, 10, 12 and 16-bit) through different selection of subword sizes. When subword size of 8-bit is selected, the *SWP ME* operator performs on five 8-bit pixels in parallel. Similarly for 10, 12 and 16-bit subwords, *SWP ME* operator perform parallel operations on four 10-bit, three 12-bit or two 16-bit pixels respectively. For comparison purpose the performance of five *simple 8-bit ME* operators (shown in Figure 6.11) are compared with *SWP ME* operator. Five *simple 8-bit ME* operators are chosen because *SWP ME* operator can also performs on five 8-bit pixels in parallel when the selected subword size is 8-bit. Although our *SWP ME* operator can perform on 10, 12 and 16-bit pixel sizes as well but for this comparison we take its capability to perform on 8-bit pixels only. During this comparison the current block size of (16×16) and search area size of (48×48) is used. FS is used as search algorithm. Figure 6.12 shows the area , CP, power and efficiency comparison of five *simple 8-bit ME* operators with *SWP ME* operator.

As shown in Figure 6.12, on average five *simple 8-bit ME* operators consumes almost twice area compared to single *SWP ME* operator. SWP ME operator consumes less area due to the efficient utilization of operating units and data path. For instance instead of five memory accesses the *SWP ME* operator requires only single memory access to read five 8-bit pixels. Same is the case for other units as well. The CP of both *SWP ME* operator (for 8-bit subword size) and *simple 8-bit ME* operator are almost the same as internally both are working on 8-bit data size. The CP of the *SWP ME* operator is slightly more because of SWP overheads related to subwords. The power consumption of five *simple 8-bit ME* operators is more compared to the *SWP ME* operator because of high gate count. (Area × CP) and (Area × CP × power) products of *SWP ME* operator are less than five *simple 8-bit ME* operators. On 90nm and 130nm ASIC technologies, (Area × CP × power) product term is 71% and 69% less than five *simple 8-bit ME* operators respectively. On FPGA platform, the (CLBs × CP) product of *SWP ME* is almost 38% less than five *simple 8-bit ME* operators.

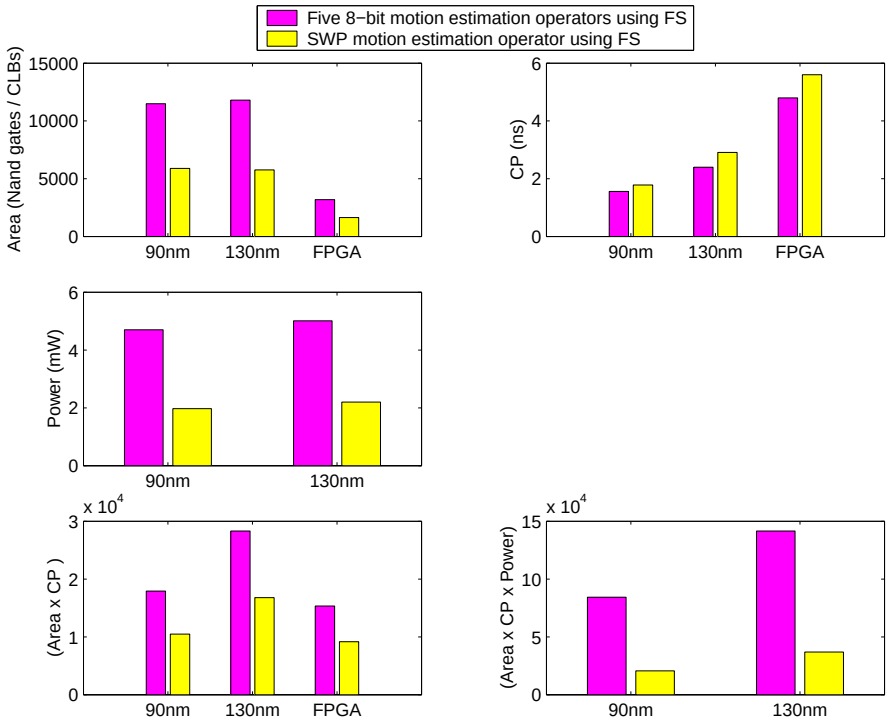

FIGURE 6.12: Comparison of five 8-bit ME operators and SWP ME operator using FS

The number of cycles required to find best the match using five *simple 8-bit ME* operators is almost 15% less than *SWP ME* operator. The reason being that *SWP ME* operator has to handle other pixel sizes (10, 12 and 16-bit) as well. Therefore some cycles are consumed on the alignment and arrangement of subwords corresponding to different pixel sizes. On the other hand *simple 8-bit ME* operator is a dedicated unit which cannot perform on other multimedia oriented pixel sizes. Overall performance results shows that compared to simple operators *SWP ME* operator provides flexibility as well as the efficiency in the computations of motion estimation.

6.6 SWP ME using Diamond search algorithm

In this section *SWP ME* operator explained in section 6.4 is used to find the motion vector using diamond search (DS) algorithm. As explained earlier DS algorithm gives the best match in less time compared to full search algorithm. However the accuracy of DS algorithm is less compared to FS algorithm. As far as the architecture of *SWP ME* operator for DS algorithm is concerned it is slightly different from *SWP ME* operator for FS algorithm. In FS algorithm, the base address of next block in search area image is always known in prior. Therefore controller does not wait for any input to decide about next base address. Where as in *SWP ME* operator for DS algorithm, the next block

in search area image which needs to be compared with current block depends on the outcome of previous comparison. The previous comparison decides whether the search pattern remains in LDSP or shifted to SDSP. Therefore after each iteration of LDSP, the comparator unit has to declare the minimum SAD point in LDSP and is intimated to control unit. On the basis of the output of comparator, control unit decides to remains in LDSP or jump to SDSP. Hence the control unit generates the base address of next block in search area accordingly. Therefore in DS algorithm controller has to do more job compared to FS algorithm. The block diagram of SWP ME operator using DS search algorithm is shown in Figure 6.13.

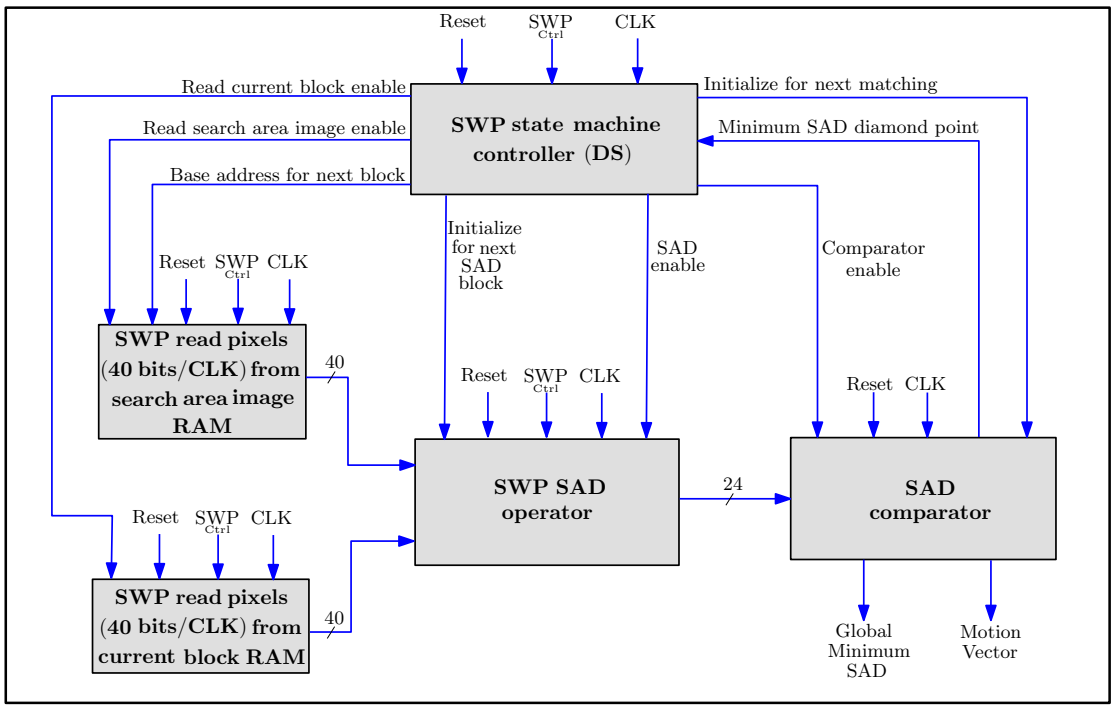

FIGURE 6.13: SWP ME operator using the diamond search algorithm

As shown in Figure 6.13, the *SAD comparator* unit finds out the minimum SAD point of the diamond and send its coordinates to controller. The controller checks whether the minimum SAD point is located at the center of diamond or not. If it is located at the center of diamond then the controller jumps to SDSP and generates the base addresses of next blocks accordingly. However if the minimum SAD point is not located at the center of diamond then the controller remains in LDSP and generate the base addresses of next blocks. Due to these extra tasks, the controller unit of *SWP ME* operator for DS algorithm is complex compared to FS algorithm. The other units like *RAM for current block and search area image*, *RAM read units*, *SWP SAD operator* and *SAD comparator* etc. operates in the same manner as it was for FS algorithm.

6.6.1 Search time

In DS algorithm search time (ST) required to find the match of current block in search area image is less compared to FS algorithm. In FS algorithm the search time is always fixed because the current block has to compare with all the blocks in search area image. Where as in DS algorithm search time is not always fixed and it varies for different search area images and current blocks. In these experiments the *Lena image* whose pixel values are shown in Figure 6.10 are used as search area image. *SWP ME* operator using DS algorithm is applied to find the best match of current block. Different sizes of current blocks and search area images are used in these experiments. Simulations are performed to find the search time required by *SWP ME* operator when using DS as search algorithm. The comparison of search time corresponding to the different selection of subword sizes while using FS and DS algorithms is shown in Figure 6.14.

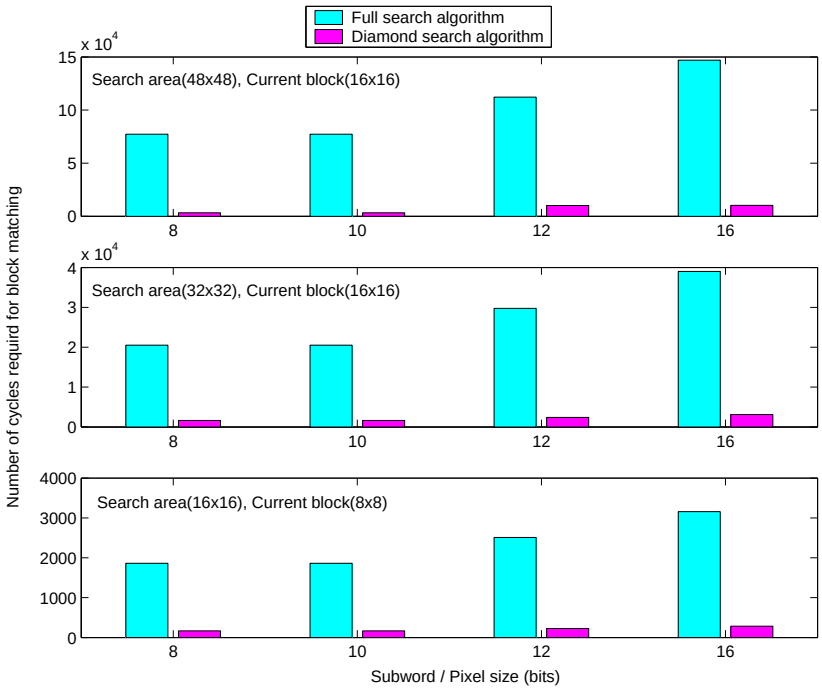

FIGURE 6.14: Number of cycles required by FS and DS algorithms to find the best match

For different subword sizes, the search time of DS algorithm is less compared to FS algorithm when applied on the same search area image and current block. In these experiments the results shows that to find the best match, DS algorithm almost requires 90% to 95% less time compared to FS algorithm using *SWP ME* operator. However this reduction in search time occurs at the cost of accuracy. For all block sizes, 8-bit subword size requires less search time compared to other larger subword sizes because it offers more parallelism. Cycles for 8-bit subword size and 10-bit subword size are same

because for the specified current block sizes both subword sizes requires same number of cycles. For the same size search area image and current block, if the pixel values are changed then the search time of FS algorithm remains same but for DS algorithm the search time varies in accordance with required search steps.

6.6.2 Synthesis results

To analyze the area, CP and power consumption, *SWP ME operator for DS algorithm* is synthesize on both ASIC and FPGA platforms. During this synthesis different current block sizes and search area image sizes are considered. The synthesis results of *SWP ME* operator for DS algorithm is shown in table 6.3.

Search area	**Current block**	**90nm CMOS ASIC**			**130nm CMOS ASIC**			**FPGA VirtexII**	
		Nand Gates	**CP (ns)**	**Power (mW)**	**Nand Gates**	**CP (ns)**	**Power (mW)**	**CLBs**	**CP (ns)**
(48x48)	**(16x16)**	7157	3.56	2.4	7009	5.82	2.7	2057	11.0
(32x32)	**(16x16)**	6980	3.56	2.3	6844	5.79	2.6	2018	10.3
(16x16)	**(8x8)**	6763	3.56	2.25	6624	5.79	2.5	1948	10.1

TABLE 6.3: Synthesis of SWP ME operator using DS algorithm

For all pairs of current block and search area sizes, the area of *SWP ME operator using DS algorithm* is higher than the area of *SWP ME operator using FS algorithm*. On average the area increase from FS to DS algorithm is approximately 20%. This increase in area corresponds to the extra complexity of the controller in the *SWP ME* operator when using the DS algorithm. Due to this increase in area, power consumption also increases by small amount. The CP is based on the pipelined architecture of SWP arithmetic units which remains same for both FS and DS *SWP ME* operators. With the decrease of the current block and search area sizes, the gate area, CP and consumed power decrease slightly in the same manner as it was for FS *SWP ME* operator. On average, moving from search area image of (48x48) and current block size of (16x16) to search area image of (16x16) and current block size of (8x8), there is only 5% to 7% decrease in area. This decrease in area corresponds to the smaller block sizes and smaller search area image sizes. The CP for different current blocks and search area pairs remains almost same due to the internal pipelined architectures of the different processing units. As there is a only small variation in the number of gates for different pairs, so power consumption also varies slightly accordingly.

6.6.3 Comparison of simple and SWP ME operator

Like FS, comparison is also made between *SWP ME* and five *simple 8-bit ME* operators when using DS algorithm. Figure 6.15 shows the area , CP, power and efficiency comparisons.

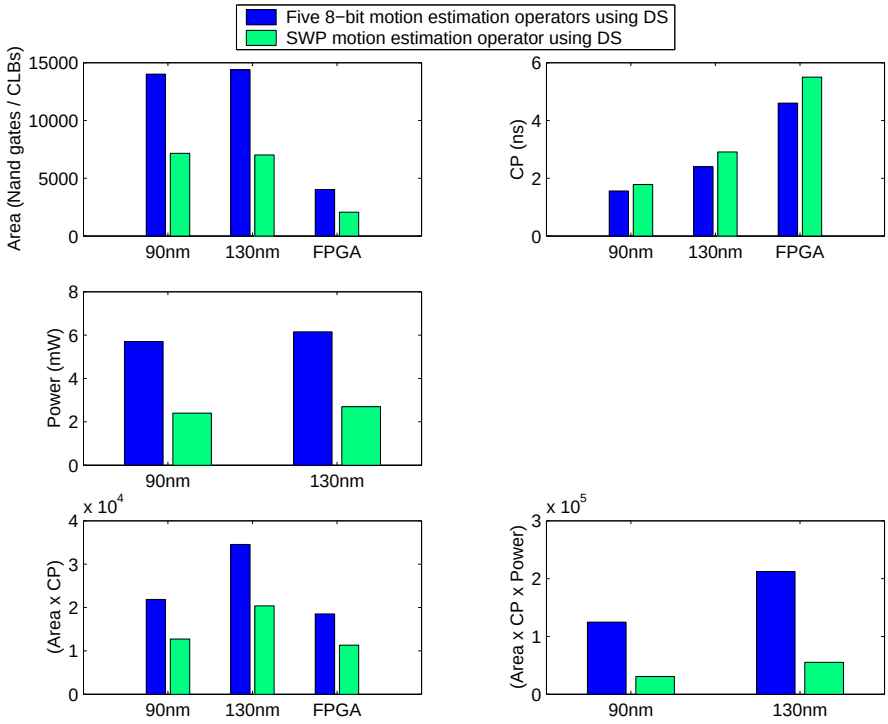

FIGURE 6.15: Comparison of five 8-bit ME operators and SWP ME operator using DS

As shown in Figure 6.15, compared to FS algorithm the area of both simple and SWP ME operator increases slightly due to the use of DS algorithm. On all target technologies, the *SWP ME* operator only consumes almost 50% of the area consumed by five *simple 8-bit ME* operators. The reduction in area occurs because of the efficient use of resources. The CP of the *SWP ME* operator (for 8-bit subword size) is slightly more than *simple 8-bit ME* operator due to the arrangement and alignment of subwords corresponding to SWP architectures. Due to the high gate count and power consumption of five *simple 8-bit ME* operators, the (Area $\times$ CP $\times$ power) product of *SWP ME* operator is almost 70% less on both ASIC technologies. On FPGA platform, the (Area $\times$ CP) product of *SWP ME* operator is almost 40% less than five *simple 8-bit ME* operators. The architecture of the *SWP ME* operator is slightly complicated compared to replication of simple operators. However at the same time the SWP operator provides a better resource utilization and flexibility through parallel operations on different size pixel data.

6.7 Comparison of FS and DS SWP ME operators

In this section the performance of the *SWP ME* operator is compared when using two different search algorithms. The use of different search algorithms like FS and DS do not make any major difference in the hardware requirements of the *SWP ME* operator because internally almost the same operating units are used irrespective of the search algorithm. The only difference lies in controller part which generates and activates the units according to different search algorithm. However the search time corresponding to different search algorithms can make major difference in the overall efficiency of ME computations using *SWP ME* operator. Figure 6.16 shows the area, CP, power and overall efficiency comparison of *SWP ME* operator when using FS and DS algorithm. For this comparison current block size of (16×16) and search area image size of (48×48) are used.

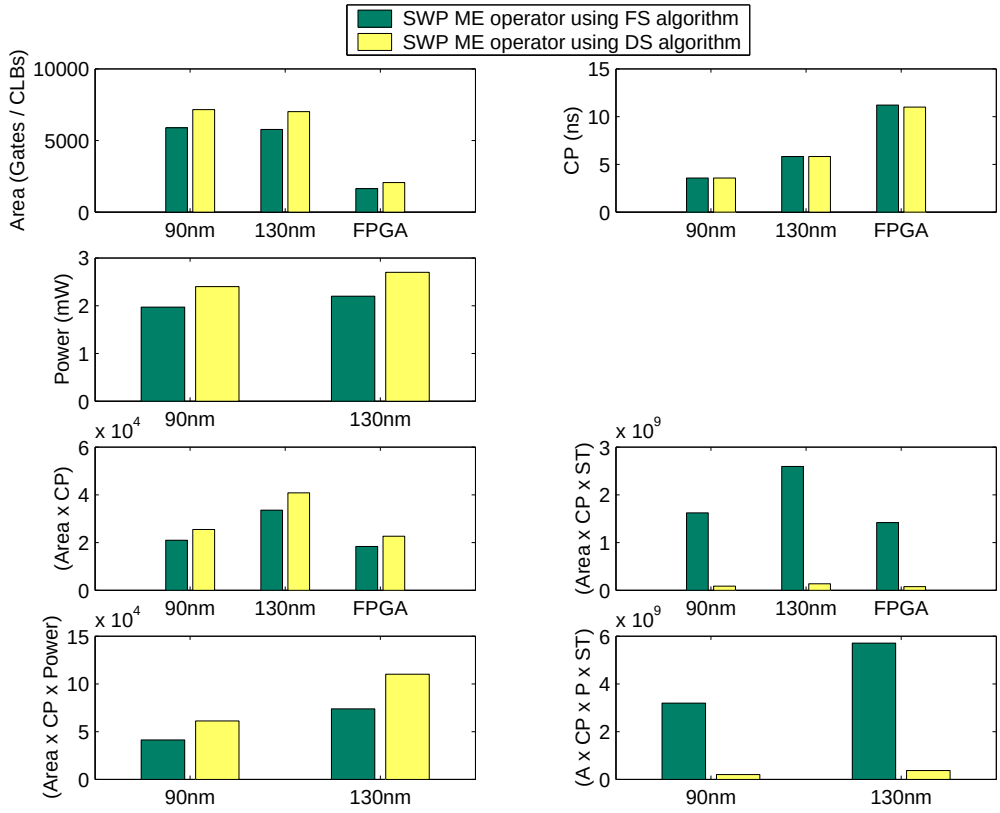

FIGURE 6.16: Comparison of SWP ME operator using different search algorithm

As shown in Figure 6.16, the area of the *SWP ME operator using the DS algorithm* is slightly more than the *SWP ME operator using the FS algorithm*. This small increase in area is due to the extra control logic required in DS to decide about the diamond switching and finding the base address of the next block. CP of the *SWP ME* operator when using FS and DS are almost same because of using same internal operators for

ME calculations. The power consumption increases slightly when we move from FS algorithm to DS algorithm due to the small increase in gate count. Hence there is not a very tremendous difference between the area, CP and power consumption of *SWP ME* operator when using FS and DS algorithms. On both ASIC and FPGA technologies, the product of area and CP for FS algorithm is 15% to 20% less than DS algorithms. In order to analyze the combined effect of hardware resources, the product of area, CP and power is plotted for both FS and DS algorithms. On both ASIC technologies this product is almost 20% to 30% less when using FS algorithm.

The search time (ST) of the FS and DS algorithms makes major difference in determining the overall efficiency of the motion estimation. This difference in search time (ST) can be highlighted by analyzing the (area×CP×search time) and (area×CP×power×search time) *product terms* corresponding to the FS and DS algorithms while using *SWP ME* operator. These *product terms* gives the over all efficiency of the motion estimation process when working on different search algorithms. A smaller value of these *product terms* indicates a high efficiency. *product terms* of FS algorithm are much larger than DS algorithm. The main reason being the large number of clock cycles required to find best match in FS algorithm. *product terms* of *SWP ME* operator when using DS algorithm are almost 90% to 95% less than FS algorithm [2]. In most of the practical video standards due to high search time requirements of FS algorithm, it is not mostly used. However as far the accuracy of the motion estimation is concern the FS algorithm always has an edge over other search algorithms.

6.8 SWP ME IP core

To analyze the performance, the proposed *SWP ME* operator is used as IP core in embedded processor environment. For this purpose, the Xilinx Embedded Development Kit (EDK) [24] is used that enables us to design a complete custom embedded processor system for implementation in a Xilinx FPGA device. Under the umbrella of EDK, the hardware portion of embedded processor is designed using Xilinx Platform Studio (XPS) and the software applications developed in C/C++ language are created and verified using Software Development Kit (SDK). In our implementation, we have targeted *XC3SD1800A* FPGA device from *Spartan-3A DSP* family. MicroBlaze [104] is used as a processor type which is a 32-bit RISC Harvard architecture soft processor core with a rich instruction set optimized for embedded applications. With the MicroBlaze soft processor, we have complete flexibility to select the combination of peripheral, memory and interface features that will give us the exact system we need on a single FPGA

[2] High value is also due to the size (48×48) see Figure 6.14

device. By using MicroBlaze, the soft embedded processor core environment created for our implementation is shown in Figure 6.17.

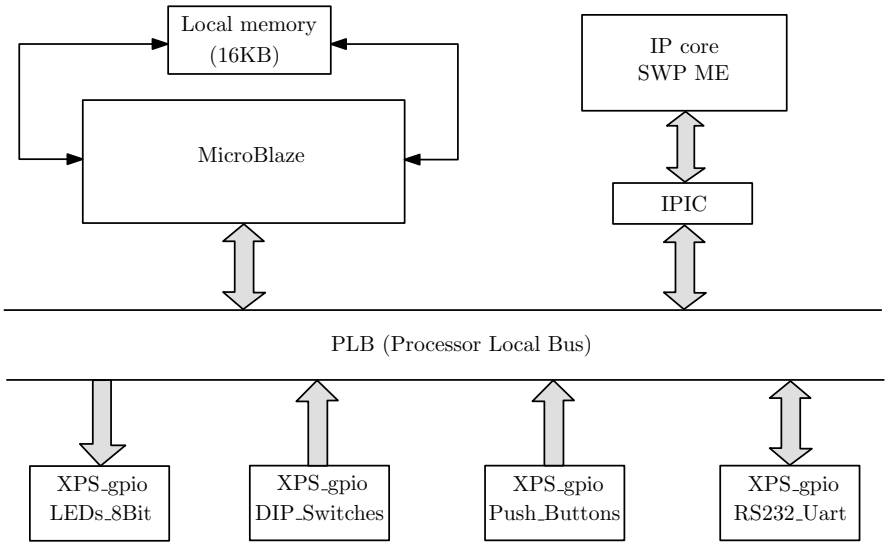

FIGURE 6.17: Soft embedded processor

In the soft processor configuration, based on the requirements different peripherals and memory elements are selected using XPS. Local memory of 16KB is selected for MicroBlaze. However for larger applications, the processor can use double data rate synchronous dynamic random access (DDR-SDR) memory available on Spartan-3A DSP board which is hard wired to FPGA device. All the peripherals communicate with MicroBlaze using high speed processor local bus (PLB). The general purpose inputs and output (XPS_gpio) devices like LEDs, DIP switches, push buttons, Universal asynchronous receiver and transmitter (UART) etc. are connected on PLB. The functionality of all these peripherals are controlled by MicroBlaze processor. LEDs on the board are used to monitor the status of certain output ports. The inputs to certain hardware modules can be provided through DIP switches or push buttons. UART is used to transmit and receive the data to/from the output terminal using RS-232 port. The custom IP core (*SWP ME* operator) is integrated to MicroBlaze processor using IPIC (intellectual property interconnect) signals. These signals provides the interface which are much easier to work with when compared to operating on the PLB protocols directly. The software applications for underlying hardware platform are developed in C/C++ language using SDK tool. These application contains input data which is used by different peripherals and processor's elements. When the software application is executed on hardware platform, the input pixels from reference and current frames are provided to *SWP ME* IP through MicroBlaze. The MicroBlaze processor receives the input pixels using RS232_UART peripheral. The pixel data is then forwarded to *SWP ME* IP core for efficient computation of motion estimation algorithm. Based upon the pixel

sizes in multimedia application, the subword size is selected by DIP switches which are connected directly to the inputs of the *SWP ME* IP core. After performing the computations, *SWP ME* IP core gives the motion vector coordinates to MicroBlaze which directs them to output terminal using RS232_UART.

Due to the use of multimedia oriented SWP, the MicroBlaze processor along with *SWP ME* co-processor perform the motion estimation computations in less time compared to conventional operators. The actual speed-up depends upon the size of pixels in multimedia application under consideration. Results shows that for different block sizes, the proposed SWP co-processor provides the practical speed-up of 4.7, 3.6, 2.7 and 1.8 when the input pixel sizes are of 8, 10, 12 and 16-bit respectively. The practical speed-up for different subword sizes are slightly less than theoretical speed-up (5, 4, 3 and 2 for 8, 10, 12 and 16-bit pixel sizes respectively) due to different SWP overheads. These overheads includes packing/unpacking of pixels in word size registers, arrangement and alignment of subwords before and after certain computations etc. However even with these small overheads, SWP operator gives better speed-up compared to conventional operators. For FS and DS algorithms, the *SWP ME* IP core along with communication interface requires 1878 and 2197 CLBs respectively. Due to the efficient utilization of hardware resources, the proposed IP core can be used in any multimedia processor to enhance the performance.

6.9 Conclusions

This chapter presented the implementation of motion estimation algorithms using SWP operators. The proposed SWP motion estimation operator operates on multimedia oriented pixel sizes using different selections of subword sizes. The *SWP ME* operator utilizes the data level parallelism and increase the performance through parallel processing of pixels. Performance of *SWP ME* operator is analyzed on two different search algorithms that are full search and diamond search algorithms. Parallel architecture of *SWP ME* operator is used to carry out the intensive computations of highly accurate full search algorithm. On the other hand *SWP ME* operator is also employed to perform motion estimation using diamond search algorithm which is well known for its efficiency and accuracy. The performance of *SWP ME* operator on these two search algorithms are compared and their advantages and drawbacks are highlighted. Despite the search algorithm, the performance of *SWP ME* operator is also evaluated for different selections of current block sizes as well as search area images. The size of these two parameters plays a vital role in motion estimation algorithms. The effect of increase and decrease in the block sizes on the overall performance of *SWP ME* operator is discussed in detail.

After the analysis it is found that compared to simple motion estimation operator, *SWP ME* operator gives high efficiency through parallel processing as well as the flexibility by operating on different pixel sizes of multimedia nature.

Chapter 7

Conclusion and Perspectives

7.1 Conclusion

This thesis describes the implementation of reconfigurable arithmetic operators for multimedia applications. The performances of these operators are improved by exploiting the data level parallelism that exist in multimedia applications. As explained in Chapter 1, subword parallelism (SWP) is used to perform parallel operations on subwords which are packed in word size registers. Smaller the subword size higher will be the parallelism. The overheads for incorporating SWP capability include subwords alignment and arrangement before the computations. Compared to the SWP advantages these overheads are small but are to be taken into account. Classical use of subword sizes in SWP results in the under utilization of processor resources when working on multimedia applications. In this thesis reconfigurable arithmetic operators are proposed which are based upon multimedia oriented subword sizes. These subword sizes are in coordination with pixel sizes in multimedia applications. Due to this coordination, the processor utilizes the available resources more efficiently.

In the beginning of this thesis, SWP capability is introduced in the architectures of basic arithmetic units. For comparison purpose, these operators are designed using both classical and multimedia oriented subword sizes. The overheads for incorporating SWP capability in the operator's design are analyzed in Chapter 2. Due to the non uniform nature of pixel sizes, the complexity of multimedia SWP operators are little more compared to classical SWP operators. However due to the efficient utilization of the resources, the performance of multimedia oriented SWP operators is much higher when working on different video applications. In Chapter 3, proposed SWP basic operators are used to design different application specific operators which are commonly required in the multimedia domain. These operations include sum of absolute difference (SAD)

for motion estimation, sum of product (SOP) for different transforms etc. Based upon the selected subword size, these dedicated SWP operators execute parallel operations on multiple pixels.

In Chapter 4, the pipelined architecture of the reconfigurable SWP operator is presented. In conventional reconfigurable processors, the interconnection network reconfigures itself for new set of application without focusing on the internal architectures of the operating units. The proposed operator can reconfigure itself at both operation level and data size level without any reconfiguration time overheads. Using this operator, variety of multimedia operations can be performed on different sizes of pixel data. To perform any operation, the control unit activates only those SWP units which need to perform certain computations on input pixel data. This will reduce the overall power consumption of the operator when performing different multimedia operations. Performance of the reconfigurable operator is analyzed on different target technologies. The proposed reconfigurable operator can be used in any processor to enhance the performance for multimedia applications.

SWP increases the performance of the processor through parallel processing of data; however the internal computation speed of different arithmetic units also plays an important role in the overall performance of the processor. To increase the speed as well as the parallelism in different operations, for the first time multimedia oriented SWP capability is introduced in arithmetic operators using redundant number system. This number system provides a carry propagation free addition of input vectors. In redundant number system, the addition of numbers is performed in constant time irrespective of input data length. As the addition process is used in almost all the arithmetic operations (MULT, SUB, MAC etc.), therefore any speed enhancement in the addition algorithm results in the overall increase in the speed of different operations. In redundant system, the numbers are represented by digits rather than bits which causes some increase in area. However the speed-up attained at the cost of this small increase in area is of vital interest. By using SWP on redundant number system, parallelism as well as the high speed computations are obtained in arithmetic operator's design. These high speed SWP arithmetic operators are then used to design reconfigurable operator for multimedia applications. This operator can perform different multimedia operations at high speed compared to binary SWP operator. In Chapter 5, the performance comparison of binary and redundant SWP operator is also presented.

Motion estimation is one of the most commonly used algorithm in multimedia applications. Based upon its computational intensity, SWP operators are used to design a dedicated hardware architecture for the computation of motion estimation in Chapter 6. The SWP motion estimation operator performs block matching on different multimedia

oriented pixel sizes. The performance of this operator is analyze on different search algorithms (full search, diamond search) using different block sizes. Due to the parallel computations, the SWP ME operator performs computations much more efficiently compared to simple operators. The actual speed-up attained depends upon the subword/pixel sizes. The overheads for this performance enhancement are also highlighted in Chapter 6. Results shows that multimedia oriented SWP capability in the arithmetic operators increases the overall efficiency of the processor to a larger extent.

7.2 Future perspectives

This work opens various directions of research in multimedia processing. Some of them are discussed below.

This thesis focuses on the development of operators for multimedia processing. However the design of compiler which can further improve the performance of the application that is executed on the processor is out of the scope of this thesis. In future, work can be done to design the compiler which provides friendly environment for multimedia oriented SWP operators. The compiler must be able to parallelize the data to maximum extend. The arrangement and alignment overheads of the SWP operator can be distributed at compiler level as well. To improve the performance, the compiler of the SWP processor contains the intelligence to manage the subwords to maximum extent before giving them to the processing units. By using multiple SWP operators, the compiler can optimize the performance using both instruction level and data level parallelism. By doing so compiler can issue multiple instructions for parallel computations on the data using SWP operators.

The reconfigurable operators proposed in Chapter 4 and Chapter 5 can be extended to perform other required operations on low precision data. However it will depend upon the type of applications for which the operator needs to be designed. The SWP motion estimation block presented in Chapter 6 can be extended to design dedicated co-processors for other multimedia computations. These co-processors perform the required operation more efficiently by using parallel processing on input data. In the co-processors environment, the scheduling of appropriate tasks to the processors plays an important role. The algorithms for the scheduling and allocation [14, 32, 35] of tasks to the SWP co-processors needs to be explored to further enhance the performance.

Work can be done to improve the interface between memory and SWP operators so that the required data should be available in minimum clock cycles. For this purpose the arrangement and alignment of subwords in the memory needs to be focused. For

the performance enhancement, the memory structure and the data link between the processing units and the memory should support the transaction of multiple subwords in a single memory reference. The data must also be addressable at the subword-level (for example to perform delay operation as in a FIR filter). For this purpose, work can be done to provide efficient interface between SWP operators and memory.

In this thesis, beside binary SWP operators the SWP technique is also introduced on the redundant number system which gives very promising results. This idea can further be extended by designing operators using the combination of SWP and other efficient number systems. The subword sizes can also be tailored according to data sizes in targeted applications.

Personal Publications

[1] **S. Khan**, E. Casseau and D. Menard. "Reconfigurable SWP Operator for Multimedia Processing", In *Proceedings of the International Conference on Application-specific Systems, Architectures and Processors(ASAP'09)*, Pages 199-202, Boston, USA, 2009.

[2] **S. Khan**, E. Casseau and D. Menard. "SWP for multimedia operator design", In *Proceedings of the 2nd Colloque Nationale of GDR SoC-SIP*, Paris, France, 2008.

[3] **S. Khan**, E. Casseau and D. Menard, "SWP multimedia operator design", In *Proceeding of 5th international sciences of electronics, technologies of information and telecommunications conference (SETIT)*, Hammamet, Tunisia, March 2009.

[4] D. Menard, E. Casseau, **S. Khan**, O. Sentieys, S. Chevobbe, S. Guyetant and R. David. "Reconfigurable Operator Based Multimedia Embedded Processor", In *Proceedings of the International Workshop on Applied Reconfigurable Computing: Architectures, Tools and Applications (ARC'09)*, Pages 39-49, Karlsruhe, Germany, 2009.

[5] **S. Khan**, E. Casseau and D. Menard. "High speed reconfigurable SWP operator for multimedia processing using redundant data representation", In *International journal of information science and computer engineering*, Vol. 1, No. 1 Pages 45-52, Australia, 2010.

[6] E. Casseau, **S. Khan** and B. Le Gal, "Multimode architecture design", In *Proceeding of Design and Architectures for Signal and Image Processing Workshop (DASIP'07)*, Grenoble, France November 2007.

[7] B. Le Gal, E. Casseau and **S. Khan**, "HLS Design Flow for Multimode IP Generation Under Multiple Constraints", In *Proceeding of 14th IEEE International Conference on Electronics, Circuits and Systems (ICECS'07)*, Marrakech, Morocco, December 2007.

Bibliography

[1] 3DSP. SP-5 Fixed-point Signal Processor Core. 3DSP Corporation, Irvine, Calif, USA, July 1999.

[2] A. Abdelgawad and M. Bayoumi. High Speed and Area-Efficient Multiply Accumulate (MAC) Unit for Digital Signal Processing Applications. In *Proceedings of the IEEE International Symposium on Circuits and Systems ISCAS*, pages 3199–3202, May 2007.

[3] L. V. Agostini, I. S. Silva, and S. Bampi. Pipelined fast 2D DCT architecture for JPEG image compression. In *Proceedings of the 14th Symposium on Integrated Circuits and Systems Design*, pages 226–231, Pirenopolis, 2001.

[4] H. Amano. A Survey on Dynamically Reconfigurable Processors. In *Institute of Electronics, Information and Communication Engineers (IEICE) Transactions on Communications*, pages 3179–3187, 2006.

[5] H. Amano, Y. Hasegawa, S. Tsutsumi, T. Nakamura, T. Nishimura, V. Tanbunheng, A. Parimala, T. Sano, and M. Kato. MuCCRA chips: Configurable Dynamically-Reconfigurable Processors. In *Proceedings of the IEEE Asian Solid-State Circuits Conference (ASSCC'07)*, pages 384–387, 2007.

[6] N. B. Amor, Y. L. Moullec, J. P. Diguet, J. L. Philippe, and M. Abid. Design of a Multimedia Processor Based on Metrics Computation. In *Advances in Engineering Software*, volume 36, pages 448–458, 2005.

[7] A. B. Attitalah, P. Kadionik, F. Ghozzi, P. Nouel, N. Masmoudi, and P. Marchegay. Implementation of Loeffler Algorithm on Stratix DSP compared to Classical FPGA Solutions. In *Proceedings of the International Symposium on Communications, Control and Signal Processing (SCCSP)*, Morocco, 2006.

[8] A. Avizienis. Signed-Digit Number Representations for Fast Parallel Arithmetic. *IRE Transactions on Electronic Computers*, 10:389–400, 1961. Reprinted in E. Swartzlander (ed.), Computer arithmetic, vol.II, IEEE Computer Society Press, 1990.

[9] M. Bousselmi, M. S. Bouhlel, N. Masmoudi, and L. Kamoun. New parallel architecture of the DCT and its inverse for image compression. In *Proceedings of the the 7th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, volume 1, pages 345–348, 2000.

[10] C. Brunelli, F. Garzia, and J. Nurmi. A coarse-grain reconfigurable architecture for multimedia applications featuring subword computation capabilities. In *Proceedings of the Journal of real-time image processing*, volume 3, pages 21–32, March 2008.

[11] C. Brunelli, P. Salmela, J. Takala, and J. Nurmi. A Flexible Multiplier for Media Processing. In *Proceedings of the IEEE workshop on Design and Implementation*, pages 70–74, Nov 2005.

[12] K. Bukhari, G. Kuzmanov, and S. Vassiliadis. DCT and IDCT Implementations on Different FPGA Technologies. In *Computer Engineering Lab, Delft University of Technology, Netherlands*.

[13] A. M. Campos, F. J. B. Merelo, M. A. M. Peirot', and J. A. C. Esteve. Integer-pixel motion estimation H.264/AVC accelerator architecture with optimal memory management. In *Proceedings of the International Journal of microprocessors and microsystems*, pages 68–78, 2008.

[14] E. Casseau, S. Khan, and B. Le Gal. Multimode architecture design. In *Proceeding of Design and Architectures for Signal and Image Processing Workshop (DASIP'07)*, Grenoble, France, November 2007.

[15] S. Chatterjee and A. Chakrabarti. Parallel Hardware Design for Motion Estimation. In *Proceedings of the International Journal of Recent Trends in Engineering*, volume 1, pages 653–657, May 2009.

[16] M. O. Cheema and O. Hammami. Customized SIMD Unit Synthesis for System on Programmable Chip - A Foundation for HW/SW Partitioning with Vectorization. In *Proceedings of the IEEE Design Automation conference*, pages 54–60, Jan 2006.

[17] J. Choi, N. Togawa, M. Yanagisawa, and T. Ohtsuki. VLSI Architecture for a Flexible Motion Estimation with Parameters. In *Proceedings of the 15th International Conference on VLSI Design (VLSID.02)*, 2002.

[18] K. Compton and S. Hauck. Reconfigurable Computing: A Survey of Systems and Software. In *Proceedings of the ACM Computing Surveys*, volume 34, page 171Ű210, June 2002.

[19] P. Corsonellol, S. Perri, M. A. Iachinol, and G. Cocorullo. Variable Precision Arithmetic Circuits for FPGA Based Multimedia Processors. In *IEEE Transactions on very large scale integration (VLSI) systems*, volume 12, September 2004.

[20] A. Danysh and D. Tan. Architecture and Implementation of a Vector/SIMD Multiply-Accumulate Unit. In *Proceedings of the IEEE Computer Society*, volume 54, pages 284–293, March 2005.

[21] R. David, D. Chillet, S. Pillement, and O. Sentieys. DART A Dynamically Reconfigurable Architecture dealing with Next Generation Telecommunications Constraints. In *Proceedings of the Reconfigurable Architecture Workshop (RAW 02)*, April 2002.

[22] IRISA (Institut de Recherche en Informatique et Systèmes Aléatoires). [online] available: https://www.irisa.fr.

[23] K. Diefendorff. Altivec extension to powerpc accelerates media processing. In *Proceedings of the IEEE Micro*, March 2000.

[24] Xilinx Embedded Development Kit (EDK). [online]. available: http://www.xilinx.com/ise/embedded/edk_docs.html.

[25] M. D. Ercegovac and T. Lang. *Digital Arithmetic*. Morgan Kaufmann, 2003.

[26] D. Esftathiou, J. Fridman, and Z. Zvonar. Recent developments in enabling technologies for the software defined radio. In *Proceedings of the IEEE Communication Magazine*, pages 112–117, August 1999.

[27] A. A. Farooqui and V. G. Oklobdzija. General data-path organization of a MAC unit for VLSI implementation of DSP processors. In *Proceedings of the IEEE International Symposium on Circuits and Systems ISCAS'98*, volume 2, pages 260–263, 1998.

[28] A. A. Farooqui, V. G. Oklobdzija, and F. Chechrazi. 64-Bit Media Adder. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, Orlando, May 1999.

[29] ROMA (Reconfigurable Operators for Multimedia Architectures) Project. [online] available: https://roma.irisa.fr.

[30] J. Fridman. Sub-Word Parallelism in Digital Signal Processing. In *IEEE signal processing magazine*, pages 27–35, March 2000.

[31] J. Fridman. Data alignment for sub-word parallelism in DSP. In *Proceedings of the IEEE workshop on signal processing systems*, pages 251–260, March 2002.

[32] B. Le Gal, E. Casseau, and S. Khan. HLS Design Flow for Multimode IP Generation Under Multiple Constraints. In *Proceeding of the 14th IEEE International Conference on Electronics, Circuits and Systems (ICECS'07)*, Marrakech, Morocco, December 2007.

[33] M. Ghanbari. The cross search algorithm for motion estimation. In *Proceedings of the IEEE Transaction Communication*, volume COM-38, pages 950–953, July 1990.

[34] Mentor Graphics. [online]. available: www.mentor.com.

[35] R. Gupta and F. Brewer. High Level Synthesis: A Retrospective. In *High-Level Synthesis from Algorithm to Digital Circuit, Editted by P. Coussy and A. Morawiec, Published by Springers*, pages 13–28, 2008.

[36] S. Gupta, R. Gupta, N. Dutt, and A. Nicolau. SPARK: A Parallelizing Approach to the High-Level Synthesis of Digital Circuits. Springer; 1st edition, May 2004.

[37] A. Guyot, Y. Herreros, and J. M. Muller. JANUS, an On-line Multiplier/Divider for Manipulating Large Numbers. In *Proceedings of the 9th IEEE Symposium on Computer Arithmetic*, pages 106–111, 1989.

[38] J. L. Hennessy and D. A. Patterson. Computer a quantitative approach architecture. In *3rd Edition*. Morgan Kaufmann.

[39] H. C. Hunter and J. H. Moreno. A new look at exploiting data parallelism in embedded systems. In *Proceedings of international conference on Compilers, architecture and synthesis for embedded systems*, pages 159–169, 2003.

[40] IEEE Std 1666-2005 IEEE Standard SystemC Language Reference Manual. [online]. available: http://www.systemc.org. March 2006.

[41] Texas Instruments Incorporated. TMS320C64x Technical Overview. Texas Instruments, Dallas, Tex, USA, February 2000.

[42] Analog Device Incorporation. TigerSHARC Hardware Specification. Analog Device, December 1999.

[43] CEVA Incorporation. CEVA-X1620 Datasheet. CEVA, San Jose, Calif, USA, 2004.

[44] 2008 Update Overview International Technology Roadmap for Semiconductors. [online]. available: www.itrs.net/links/2008itrs/update/2008_update.pdf.

[45] G. Jaberipur and B. Parhami. Constant-time addition with hybrid-redundant numbers: Theory and implementations. In *Proceedings of the Integration, the VLSI Journal*, volume 41, pages 49–64, 2008.

[46] G. Jaberipur, B. Parhami, and M. Ghodsi. An efficient universal addition scheme for all hybrid-redundant representations with weighted bit-set encoding. In *Proceedings of the The Journal of VLSI Signal Processing*, volume 42, pages 149–158, 2006.

[47] J. R. Jain and A. K. Jain. Displacement measurement and its application in interframe image coding. In *Proceedings of the IEEE Transaction on Communications*, volume COM-29, pages 1799–1808, December 1981.

[48] M. D. Jennings and T. M. Conte. Subword extensions for video processing on mobile systems. In *Proceedings of the IEEE Concurrency*, volume 6, pages 13–16, July-Sept 1998.

[49] X. Jing and L. P. Chau. An Efficient three-step search algorithm for block motion estimation. In *Proceedings of the IEEE Transactions on multimedia*, volume 6, pages 435–438, 2004.

[50] S. Khan, E. Casseau, and D. Menard. Reconfigurable SWP operator for multimedia processing. In *Proceedings of the International Conference on Application-specific Systems, Architectures and Processors(ASAP'09)*, pages 199–202, Boston, USA, 2009.

[51] S. Khan, E. Casseau, and D. Menard. High speed reconfigurable SWP operator for multimedia processing using redundant data representation. In *Proceedings of the International journal of information science and computer engineering*, volume 1, pages 45–52, Australia, 2010.

[52] S. Khan, E. Casseau, and D. Menard. SWP for multimedia operator design. In *Proceedings of the 2nd Colloque Nationale of GDR SoC-SIP*, Paris, France, June 2008.

[53] S. Khan, E. Casseau, and D. Menard. SWP multimedia operator design. In *Proceedings of the Proceeding of 5th international sciences of electronics, technologies of information and telecommunications conference (SETIT)*, Hammamet, Tunisia, March 2009.

[54] P. Kitsos, G. Theodoridis, and O. Koufopavlou. An efficient reconfigurable multiplier architecture for Galois field. In *Proceedings of the Microelectronics Journal*, pages 975–980, 2003.

[55] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro. Motion compensated interframe coding for video conferencing. In *Proceedings of the Nat. Telecommunication Conference*, pages 5.3.1–5.3.5, New Orleans, Nov 1981.

[56] S. Krithivasan and M. J. Schulte. Multiplier Architectures for Media Processing. In *Proceedings of the Thirty seventh Asilomar Conference on signals, systems and computers*, volume 2, pages 2193–2197, 2003.

[57] S. Krithivasan, M. J. Schulte, and J. Glossner. A subworld-parallel multiplication and sum-of-squares unit. In *Proceedings of the IEEE computer society annual symposium on VLSI*, pages 273–274, Feb 2004.

[58] M. Lanuzza, S. Perri, P. Corsonello, and M. Margala. A New Reconfigurable Coarse-Grain Architecture for Multimedia Applications. In *Proceedings of the Second NASA/ESA Conference on Adaptive Hardware and Systems AHS*, pages 119–126, 2007.

[59] R. B. Lee. Subword Parallelism with MAX-2. In *IEEE Computer Society*, volume 16, pages 51–59, August 1996.

[60] R. B. Lee. Multimedia extensions for general-purpose processors. In *IEEE Workshop on Signal Processing Systems SIPS - Design and Implementation*, pages 9–23, Nov 1997.

[61] R. Li, B. Zeng, and M. L. Liou. A new three step search algorithm for block motion estimation. In *Proceedings of the IEEE Transaction on Circuits and Systems for Video Technology*, volume 4, pages 438–442, August 1994.

[62] T. Li, S. Li, and C. Shen. A novel configurable motion estimation architecture for high-efficiency MPEG-4/H.264 encoding. In *Proceedings of Asia and South Pacific Design Automation Conference*, pages 1264–1267, 2005.

[63] Z. Li, S. Peng, H. Ma, and Q. Wang. A Reconfigurable DCT Architecture for Multimedia Applications. In *Proceedings of the Congress on Image and Signal Processing*, pages 360–364, 2008.

[64] Y. Liao and D. B. Roberts. A high-performance and low-power 32-bit multiply-accumulate unit with single-instruction-multiple-data (SIMD) feature. In *Proceedings of the IEEE Journal of solid-state circuits*, pages 926–931, July 2002.

[65] Y-C. Lin and S-C. Tai. Fast full-search block-matching algorithm for motion compensated video compression. In *Proceedings of the International Conference on Pattern Recognition (ICPR '96)*, pages 914–921, 1996.

[66] L. K. Liu and E. Feig. A block based gradient descent search algorithm for block motion estimation in video coding. In *IEEE Transaction on Circuits and Systems for Video Technology*, volume 6, pages 419–422, August 1996.

[67] H. Loukil, A. B. Atitallah, F. Ghozzi, M. A. Ben Ayed, and N. Masmoudi. A Pipelined FSBM Hardware Architecture for HTDV-H.26x. In *Proceedings of the International journal of electrical and electronics engineering*, pages 128–135, 2008.

[68] R. Meagher, M. Sushmitha, M. E. Rizkalla, P. Salama, and M. E. Sharkawy. VHDL Design for Real Time Motion Estimation Video Applications. In *Proceedings of the Journal of Signal Processing Systems*, pages 339–348, Dec 2008.

[69] D. Menard, E. Casseau, S. Khan, O. Sentieys, S. Chevobbe, S. Guyetant, and R. David. Reconfigurable Operator Based Multimedia Embedded Processor. In *Proceedings of the International Workshop on Reconfigurable Computing: Architectures, Tools and Applications*, pages 39–49, Karlsruhe, Germany, 2009.

[70] D. Menard and O. Sentieys. DSP Code Generation with Optimized Data-Word Length Selection. In *Proceedings of 8th International Workshop on Software and Compilers for Embedded Systems (SCOPES'04)*, Amsterdam, Netherlands, Sep 2004.

[71] Daniel Menard, Daniel Chillet, and Olivier Sentieys. Floating-to-Fixed-Point Conversion for Digital Signal Processors. In *EURASIP Journal on Applied Signal Processing*, pages 1–19, Hindawi Publishing Corporation, 2006.

[72] M. Nagabushanam, C. P. Raj, and S. Ramachandran. Design and implementation of parallel and pipelined distributive arithmetic based discrete wavelet transform IP core. In *Proceedings of the European Journal of Scientific Research*, volume 35, pages 378–392, 2009.

[73] J. Oliver, V. Akella, and F. Chong. Efficient orchestration of sub-word parallelism in media processors. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 225–234, 2004.

[74] Coregen [Online]. [online]. available: www.xilinx.com/ipcenter/coregen/.

[75] D. S. Phatak, T. Goff, and I. Koren. Constant-Time Addition and Simultaneous Format Conversion Based on Redundant Binary Representations. In *Proceedings of the IEEE Transactions on computers*, volume 50, pages 1267–1278, 2001.

[76] L. M. Po and W. C. Ma. A novel four-step search algorithm for fast block motion estimation. In *IEEE Transaction on Circuits and Systems for Video Technology*, volume 6, pages 313–317, June 1996.

[77] A. Puri, H. M. Hang, and D. L. Schilling. An efficient block matching algorithm for motion compensated coding. In *Proceedings of the IEEE International Conference Acoustic, Speech and Signal Processing*, pages 1063–1066, 1987.

[78] N. Roma and L. Sousa. A New Efficient VLSI Architecture for Full Search Block Matching Motion Estimation. In *Proceedings of the Eleventh international conference on very large scale integration of systems on chip*, volume 218, pages 253–264, 2001.

[79] Y. Saito, T. Sano, M. Kato, V. Tunbunheng, Y. Yasuda, and H. Amano. A Real Chip Evaluation of MuCCRA-3: A Low Power Dycamically Reconfigurable Processor Array. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'09)*, pages 283–286, 2009.

[80] R. Sangireddy and A. K. Somani. On-Chip Adaptive Circuits for Fast Media Processing. In *Proceedings of the IEEE transactions on circuits and systems*, volume 53, pages 946–950, 2006.

[81] T. Sano, Y. Saito, and H. Amano. Configuration with Self-Configured Datapath: A High Speed Configuration Method for Dynamically Reconfigurable Processors. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'09)*, pages 112–118, July 2009.

[82] M. G. Sarwer, L. M. Po, and Q. M. J. Wu. Fast sum of absolute transformed difference based 4 x 4 intra-mode decision of H.264/AVC video coding standard. In *Journal of Signal Processing: Image Communication*, volume 23, pages 571–580, 2008.

[83] M. S. Schmookler, M. Putrino, C. Roth, M. Sharma, A. Mather, J. Tyler, H. V. Nguyen, M. N. Pham, and J. Lent. A low-power, high-speed implementation of a PowerPC TM microprocessor vector extension. In *Proceedings of 14th IEEE Symposium on Computer Arithmetic*, pages 12–19, 2002.

[84] J. P. Shen and M. H. Lipasti. Modern processor design fundamentals of superscalar processors. In *Edition 1.* McGraw-Hill Science/Engineering/Math, 2002.

[85] Z. J. Shi. Subword Permutations with MIX Instructions. In *Proceedings of 39th Asilomar Conference on Signals, Systems and Computers*, pages 1637–1641, Nov 2005.

[86] I. Skliarova and A. B. Ferrari. Design and implementation of reconfigurable processor for problems of combinatorial computations. In *Proceedings of the Euromicro Symposium on Digital Systems Design*, pages 112–119, June 2001.

[87] Synopsys. [online]. available: www.synopsys.com.

[88] Synopsys. [online]. available: www.synopsys.com/dw/doc.php/doc/dwf/.

[89] N. Takagi, H. Yasuura, and S. Yajima. High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree. In *IEEE Transactions on Computers*, volume C-34, pages 789–796, 1985.

[90] M. Thornton. A signed binary addition circuit based on an alternative class of addition tables. In *Proceedings of the Computer and Electrical Engineering*, volume 29, pages 303–315, 2003.

[91] M. Thornton. The conversion algorithm and implementation between carry-save and binary sign-digit representations. In *Proceedings of the Asian Journal of information technology*, volume 10, pages 901–906, 2005.

[92] V. M. Tuan, N. Katsura, H. Matsutani, and H. Amano. Evaluation of a multicore reconfigurable architecture with variable core sizes. In *Proceedings of the Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'09)*, pages 1–8, July 2009.

[93] J. Vanne, E. Aho, T. D. Hämäläinen, and K. Kuusilinna. A High-Performance Sum of Absolute Difference Implementation for Motion Estimation. In *IEEE transaction on circuits and systems for video technology*, volume 16, July 2006.

[94] S. Vassiliadis, E. A. Hakkennes, J. S. S. M. Wong, and G. G. Pechanek. The sum-absolute-difference motion estimation accelerator. In *Proceedings of the 24th Euromicro Conference*, volume 2, pages 559–566, 1998.

[95] A. K. Verma and P. Ienne. Improved use of the carry-save representation for the synthesis of complex arithmetic circuits. In *Proceedings of the International conference on computer aided design ICCAD'04*, pages 791–798, 2004.

[96] M. Vorbach and R. Becker. Reconfigurable processor architectures for mobile phones. In *Proceedings of the International Parallel and Distributed Processing Symposium IPDPS*, pages 6–12, April 2003.

[97] J. Wakerly. Digital Design. In *3rd Edition*, Upper Saddle River, NJ, 2000. Prentice Hall.

[98] A. Wang and A. Chandrakasan. A 180-mV subthreshold FFT processor using a minimum energy design methodology. In *Proceedings of the Solid-State Circuits, IEEE Journal*, volume 40, pages 310–319, January 2005.

[99] G. Wang. The conversion algorithm and implementation between carry-save and binary sign-digit representation. In *Proceedings of the Asian journal of information technology*, pages 901–906, 2005.

[100] S. Wichman and N. Goel. The Second Generation ZSP DSP. LSI Logic Corporation, Milpitas, Calif, USA, 2002.

[101] S. Wong, B. Stougie, and S. Cotofana. Alternatives in FPGA-based SAD Implementations. In *Proceedings of the IEEE international conference on field-programmable technology (FPT)*, pages 449–452, Hong Kong, 2002.

[102] S. Wong, S. Vassiliadis, and S. Cotofana. A Sum of Absolute Differences Implementation in FPGA hardware. In *Proceedings of the 28th Euromicro Conference on Multimedia and Communications*, pages 183–188, July 2002.

[103] B. F. Wu and T. L. Yu. Efficient hierarchical motion estimation algorithm and its VLSI architecture. In *Proceedings of the IEEE Transactions on very large scale integration (VLSI) systems*, volume 16, pages 1385–1398, October 2008.

[104] S. Xu and H. Pollitt-Smith. A multi-microblaze based soc system: From systemc modeling to fpga prototyping. In *Proceedings of the 19th IEEE/IFIP International Symposium on Rapid System Prototyping*, pages 121–127, 2008.

[105] S. M. Yen, C. S. Laih, C. H. Chen, and J. Y. Lee. An efficient redundant-binary number to binary number converter. In *Proceedings of the IEEE Journal of Solid-State Circuits (JSSC)*, volume 27, pages 109–112, 2002.

[106] S. Yeo, T. Roh, and J. Kim. High Energy Efficient Reconfigurable Processor for Mobile Multimedia. In *Proceedings of the International Conference on Circuits and Systems for Communications (ISSCC'08)*, pages 618–622, June 2008.

[107] X. Zhang and X. Shen. A Power-Efficient Floating-Point Co-processor Design. In *Proceedings of the International Conference on Computer Science and Software Engineering*, pages 75–78, 2008.

[108] S. Zhu and K. K. Ma. A new diamond search algorithm for fast block matching motion estimation. In *Proceedings of the IEEE Transaction on Image Processing*, volume 9, pages 287–290, Feb. 2000.

Titre. Développement d'architectures matérielles hautes performances pour les applications multimédia

Résumé. Les besoins en puissance de calcul des processeurs sont en constante augmentation en raison de l'augmentation croissante de la complexité des applications. Les applications multimédia requièrent de nombreux calculs avec des données de faible précision généralement issues des pixels. Un moyen efficace pour exploiter le parallélisme de données de ces applications est le parallélisme dit de sous-mots (subword parallelism (SWP)). Les opérations sont effectuées en parallèle sur des données de faible précision regroupées ce qui permet d'utiliser au mieux les ressources disponibles dimensionnées pour traiter des mots. Dans cette thèse, la conception de différents opérateurs SWP pour les applications multimédia est proposée. Des opérateurs arithmétiques de base sont d'abord développés puis utilisés dans un opérateur SWP reconfigurable. Ce dernier peut être configuré pour effectuer diverses opérations multimédia avec différentes largeurs de données. L'opérateur reconfigurable peut être utilisé comme unité spécialisée ou comme co-processeur dans un processeur multimédia afin d'en améliorer les performances. Les résultats obtenus montrent l'intérêt d'utiliser des opérateurs SWP lors de l'exécution d'applications multimédia.

Title. Development of high performance hardware architectures for multimedia applications

Abstract. The computational requirements of the processors are increasing tremendously with the increase in the complexity of applications. Among these applications, multimedia represents the class of applications which requires lot of computations on low precision pixels. Subword parallelism (SWP) is one of the best options to exploit data level parallelism that exist in the applications. In SWP, rather than wasting the word oriented data path, parallel operations are executed on packed subwords. SWP increases the performance of the processor especially for multimedia applications with low precision pixel data. In this thesis, reconfigurable SWP arithmetic operators are proposed for multimedia applications. SWP arithmetic operators are then used to design reconfigurable operators for multimedia applications. These operators can perform a variety of basis as well as multimedia operations on different size pixel data. These operators can be used as co-processors to enhance the performance for multimedia applications. The performance of SWP operators are verified on different multimedia kernels.