

**Evolutionary Strategies for the High-Level Synthesis  
of VLSI-Based DSP Systems for Low Power**

*A Thesis submitted to the University of Wales  
For the Degree of Doctor of Philosophy*

M S Bright

Circuits and Systems Research Group  
Division of Electronic Engineering  
School of Engineering  
Cardiff University

October 1998

## DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any other degree.

Signed .....(candidate)

Date .....

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed .....(candidate)

Date .....

I hereby give consent for my thesis, if accepted, to be available for photocopy and for inter-library loan, and for the title and summary to be made available to other organisations.

Signed .....(candidate)

Date .....

## **Acknowledgements**

I would like to thank my supervisor, Dr. T. Arslan, for all the patience and assistance he has shown me over the course of my undergraduate and postgraduate studies. He ensured that the research was enjoyable as well as fruitful.

Thanks also to all of my colleagues in the Circuits and Systems Research Group for their assistance and support.

I also thank my family and friends who have helped me immensely during the research and the preparation of this thesis, and throughout the whole of my studies.

To Louise, for her unlimited support, understanding and patience.

## Summary

Power consumption of signal-processing microcircuit systems has become increasingly important in recent years, primarily due to the explosion in demand for portable systems where battery-lifetime is a major marketing point. In addition, the thermal dissipation levels of microcircuits are beginning to affect reliability and future development. These factors have elevated the importance of power in the design process.

The consideration of power adds another degree of complexity to the design process, hence the requirement for power-conscious tools to support low-power system design. Considering power as a high-level parameter promises to maximise power reductions. However, the complex nature of the high-level design process is significantly increased with the consideration of power consumption; hence the interest in the application of heuristic optimisation techniques.

This thesis presents a high-level tool for the power-conscious design of digital signal processing systems in CMOS technology. The tool consists of a specially tailored genetic algorithm with embedded high-level design properties. The embedding of these techniques required extensive modification to standard genetic algorithm operations to instil the algorithm with the ability to explore the low-power solution space efficiently. Problem specific genetic mutation and crossover operations were developed to incorporate high-level design transformations. To guide the search, power estimation strategies were examined and implemented within the fitness evaluation framework.

The prototype system was extended to incorporate additional genetic techniques to improve the efficiency and results of the exploration. The power-estimation module was also enhanced to obtain estimations based on practical circuit synthesis techniques.

The performance of the enhanced tool is illustrated with benchmark circuits, illustrating the tool's ability to reduce the power of practical signal processing examples. The multi-objective properties of the genetic algorithm are exploited to present design information illustrating the trade-off between area and power while keeping throughput constant. Alternative search techniques were developed and compared with the genetic algorithm. The results illustrate the superiority of the genetic algorithm in obtaining the lowest power solution and presenting trade-off information.

# Contents

<i>Declaration</i> .....	<i>ii</i>
<i>Acknowledgements</i> .....	<i>iii</i>
<i>Summary</i> .....	<i>iv</i>
<i>Contents</i> .....	<i>v</i>
<i>List of Figures</i> .....	<i>ix</i>
<i>List of Tables</i> .....	<i>xiii</i>
<i>Abbreviations</i> .....	<i>xiv</i>
<i>Nomenclature</i> .....	<i>xvi</i>
<b>Chapter 1 – Introduction</b> .....	<b>1</b>
1.1 Motivation for Low-Power VLSI .....	1
1.2 Requirements for Low-Power Computer Aided Design Tools .....	5
1.3 Overview of Thesis .....	9
1.4 Publications Arising from This Research.....	11
<b>Chapter 2 – Synthesis for Low Power VLSI</b> .....	<b>13</b>
2.1 High-Level Synthesis.....	14
2.1.1 Behavioural Level Problem Representation .....	18
2.2 Power Dissipation in CMOS Devices .....	21
2.3 Power Reduction.....	24
2.4 High-Level Power Reduction.....	30
2.4.1 Architectural Level Power-Reduction .....	30
2.4.2 Behavioural Level Optimisation.....	35
2.5 Power Reduction Using Transformations .....	37
2.5.1 Retiming.....	39
2.5.2 Pipelining.....	40
2.5.3 Automatic Pipelining.....	41
2.5.4 Loop Unfolding.....	41
2.5 Summary .....	44
<b>Chapter 3 - Power Analysis</b> .....	<b>46</b>
3.1 Power Dissipation in CMOS Devices .....	48
3.2 Transistor Level Analysis .....	49
3.3 Logic Level Analysis.....	50
3.4 Architectural Level Analysis .....	54
3.5 Behavioural Level Analysis .....	59
<b>Chapter 4 – Genetic Algorithms</b> .....	<b>64</b>
4.1 Overview of Genetic Algorithms .....	64
4.2 Standard Genetic Algorithm Implementation.....	67
4.2.1 Solution Representation.....	67

4.2.2 Fitness Assessment .....	68
4.2.3 Reproduction Operators .....	69
4.2.4 Solution Selection .....	71
4.2.5 Stopping Criteria .....	73
4.3 Non-Standard Genetic Algorithms .....	74
4.4 Applications of Genetic Algorithms .....	75
4.5 Application of GAs to the Problem of Low-Power Synthesis .....	78

## **Chapter 5 - The Genetic Algorithm..... 80**

5.1 Problem Representation.....	82
5.1.1 Chromosome Representation .....	83
5.1.2 Input Data Format .....	87
5.2 Population Initialisation.....	88
5.3 Selection Procedure .....	89
5.3.1 Implementation of Selection Procedure within GALOPS .....	91
5.4 Genetic Mutation Operators .....	92
5.4.1 Retime Transformation as Mutation .....	96
5.4.2 Back Retime Transformation as Mutation .....	98
5.4.3 Pipeline Transformation as Mutation.....	99
5.4.4 Automatic Pipeline Transformation as Mutation .....	104
5.4.5 Unfolding Transformation as Mutation .....	106
5.4.6 Application of Mutation Operators.....	110
5.5 Genetic Crossover Operators.....	113
5.5.1 Problem-Specific Crossover Operator.....	115
5.5.2 Implementation of Crossover Operator .....	118
5.6 Fitness Evaluation .....	121
5.6.1 Supply Voltage Estimation .....	124
5.6.2 Capacitance Estimation.....	130
5.6.3 Power Estimation Used as Fitness.....	135
5.7 Benchmark Designs .....	136
5.7.1 3rd Order Finite Impulse Response Filter (FIR3) .....	137
5.7.2 8th Order Finite Impulse Response Filter (FIR8).....	137
5.7.3 2nd Order Lattice Filter (LAT2).....	138
5.7.4 8th Order Avenhaus Filter .....	139
5.8 Genetic Algorithm Operating Parameters.....	143
5.9 Implementation and Results .....	149
5.9.1 Optimisation without Unfolding .....	149
5.9.2 Optimisation with Unfolding .....	157
5.9.3 Conclusions.....	160

## **Chapter 6 - Enhanced Fitness Estimation..... 162**

6.1 Scheduling and Functional Unit Allocation .....	163
6.1.1 Scheduling Algorithms .....	166
6.2 Overview of Estimation of Hardware Resources .....	170
6.3 Initial Estimates of Lower Bounds on Functional Units.....	171

6.4 Improved Estimate of Lower Bounds on Functional Units .....	174
6.4.1 ASAP/ALAP Scheduling.....	174
6.4.2 The Relaxed Estimation Technique .....	177
6.5 Examples of Lower Bound Estimation.....	183
6.6 Estimation of Lower Bounds on Registers.....	186
6.7 Hardware Unit Models.....	190
6.8 Capacitance Estimation.....	191
6.8.1 Switched Capacitance of Functional Units .....	192
6.8.2 Switched Capacitance of Registers.....	193
6.9 Summary of Enhanced Fitness/Power Estimation Module.....	194

## **Chapter 7 – Modifications to the Prototype GALOPS . 197**

7.1 Enhanced Mutation Operator (Remove-Pipeline).....	197
7.2 The Elitism Mechanism for Chromosome Selection.....	200
7.2.1 Experimental Analysis of Effect of Elitism .....	201
7.3 Selection of Operator Application Rates.....	204
7.3.1 Taguchi Optimisation of GA Application Rates.....	210
7.4 Ranking Selection Scheme .....	218
7.4.1 Experimental Analysis of Effect of Ranking.....	224
7.5 Summary .....	225

## **Chapter 8 – Results..... 227**

8.1 Benchmark Circuits .....	227
8.2 GALOPS Operating Parameters .....	229
8.3 Results for Ten Benchmark Designs .....	229
8.3.1 8TAPFIR .....	231
8.3.2 AVEN8DI.....	233
8.3.3 AVEN8PA.....	234
8.3.4 DCST .....	235
8.3.5 BIQUAD3 .....	236
8.3.6 GMLAT4.....	238
8.3.7 ELLIP5 .....	239
8.3.8 LMS5.....	240
8.3.9 VOLTERRA.....	241
8.3.10 ORTH2LAT .....	242
8.4 Discussion.....	243
8.5 Conclusions.....	246

## **Chapter 9 - Multi-Objective Search Space Exploration..... 249**

9.1 Pareto-Surface .....	251
9.2 Pareto Surface Generation in GALOPS .....	254
9.3 Area and Power Trade-off for Benchmark Designs .....	255
9.3.1 8TAPFIR .....	256
9.3.2 AVEN8DI.....	257
9.3.3 AVEN8PA.....	258

9.3.4 DCST .....	259
9.3.5 BIQUAD3 .....	260
9.3.6 GMLAT4.....	261
9.3.7 ELLIP5 .....	261
9.3.8 LMS5.....	262
9.3.9 VOLTERRA.....	263
9.3.10 ORTH2LAT .....	264
9.4 Discussion.....	264
9.5 Conclusion .....	266

**Chapter 10 – Development of Alternative Search Techniques..... 267**

10.1 Gradient Search .....	268
10.1.1 <i>Implementation and Results of Gradient Search</i> .....	269
10.2 Simulated Annealing .....	272
10.2.1 <i>Implementation and Results of Simulated Annealing</i> .....	275
10.3 Conclusion .....	280

**Chapter 11 – Conclusions and Future Work ..... 283**

11.1 Primary Features of This Work.....	283
11.2 System Implementation.....	285
11.3 Discussion.....	286
11.4 Conclusions.....	291
11.5 Future Work.....	293

References.....	297
-----------------	-----

Appendix A

## List of Figures

Figure 1.1 Power Consumption Trends in Intel Processors.....	4
Figure 2.1 Example Data Flow Graph.....	19
Figure 2.2 Example Illustrating Critical Path.....	20
Figure 2.3 Typical CMOS Digital Circuit Under Activation.....	22
Figure 2.4 Relationship between VDD and delay.....	23
Figure 2.5 Example of Two Different Logic Designs with the Same Function.....	27
Figure 2.6 Scheduling and Allocation for Low-Power.....	32
Figure 2.7 Use of Slower Low Power Functional Units on Non- Time-Critical Operations.....	33
Figure 2.8 Use of Transformations to Reduce Supply Voltage.....	36
Figure 2.9 Example of Retiming Transformation.....	39
Figure 2.10 Example of Pipelining Transformation.....	40
Figure 2.11 Example of Automatic Pipeline Transformation.....	41
Figure 2.12 3 Steps of the Unfolding Process.....	43
Figure 2.13 Example of the 3 Steps of the Unfolding Transformations.....	44
Figure 3.1 The Dual-Bit-Type Model.....	56
Figure 3.2 RTL Template for Behavioural Level Power Analysis.....	60
Figure 3.3 Interconnect Capacitance Model [Chan95].....	62
Figure 4.1 Pseudo-Code for a Basic GA.....	67
Figure 4.2 Example of a Mutation Operation.....	69
Figure 4.3 Example of a Crossover Operation.....	70
Figure 4.4 Roulette Wheel Creation.....	72
Figure 5.1 Overview of GALOPS – Low Power GA-Based Synthesis Tool.....	82
Figure 5.2 GALOPS Chromosome Template.....	84
Figure 5.3 Example Chromosome Representation.....	85
Figure 5.4 Example GALOPS Input File.....	88
Figure 5.5 Plot of Solutions within a Search Space.....	89
Figure 5.6 Effect of Linearisation on Fitness Values.....	90
Figure 5.7 Chromosome Selection Procedure.....	91
Figure 5.8 DFG Mutation Process.....	93
Figure 5.9 Example Mutation Operation Using A Retime Transformation.....	95
Figure 5.10 Pseudo-Code for the Retime Mutation Algorithm.....	96
Figure 5.11 Four Stages of the Retime Mutation Algorithm.....	98
Figure 5.12 Example of the Back Retime Transformation.....	98
Figure 5.13 Pseudo-Code of the Back Retime Transformation.....	99
Figure 5.14 Pseudo-Code for the Pipeline Mutation Algorithm.....	100
Figure 5.15 Example for Checking Pipeline Constraints.....	101
Figure 5.16 Identification of Cutset Points in Pipeline Mutation.....	103

Figure 5.17 Pseudo-Code for ‘Cutset’ Delay Insertion .....	104
Figure 5.18 Pseudo-Code for the Automatic Pipeline Mutation Algorithm.....	105
Figure 5.19 Step 1 Of Unfolding Transformation.....	107
Figure 5.20 Pseudo-Code for Step 2 of the Unfolding Transformation.....	107
Figure 5.21 Pseudo-Code for Step 3 of the Unfolding Transformation.....	108
Figure 5.22 Application of Mutation Operators within GALOPS .....	112
Figure 5.23 DFG Functionality Corruption due to the Crossover Process .....	114
Figure 5.24 Identification of Transformations for Crossover Operation.....	116
Figure 5.25 Effect of Crossover on Two Chromosomes .....	117
Figure 5.26 Pseudo-Code for Crossover Transformation Operation.....	119
Figure 5.27 Fitness Evaluation of New Design.....	123
Figure 5.28 Pseudo-Code for Recursive Path Search Algorithm.....	125
Figure 5.29 Start Point Of Critical Path.....	126
Figure 5.30 Example of Determination of Critical Path.....	127
Figure 5.31 Pseudo-Code for Supply Voltage Calculation.....	129
Figure 5.32 Interconnect Capacitance Model [Chan95].....	134
Figure 5.33 Capacitance Estimation Process .....	135
Figure 5.34 3rd Order FIR Filter .....	137
Figure 5.35 8th Order FIR Filter .....	138
Figure 5.36 2nd Order Lattice Filter.....	139
Figure 5.37 8th Order Avenhaus Filter – Direct Form Representation (AVEN8PA).....	140
Figure 5.38 8th Order Avenhaus Filter – Parallel Form .....	142
Figure 5.39 Effect of Varying Application Rates on Power Reduction of Benchmark Designs.....	145
Figure 5.40 Effect of Varying Application Rates on Power Reduction of Benchmark Designs (Batches 2-7).....	147
Figure 5.41 Effect of Varying Application Rates on Convergence Rate.....	148
Figure 5.42 Illustration of Pipeline Stages in Optimised 3rd Order FIR Filter.....	151
Figure 5.43 Processed DFG of 8th Order Avenhaus Filter – Direct Form.....	154
Figure 5.44 Plot of Best Fitness throughout Evolution from Average of Multiple Runs.....	155
Figure 5.45 Best GA Performance for AVEN8DI Design.....	156
Figure 5.46 GA Performance and Effect of Unfolding on AVEN8DI Design .....	159
Figure 6.1 Example Scheduling Operation.....	164
Figure 6.2 Illustration of Design Space with Upper and Lower Bounds .....	170
Figure 6.3 Example of Unfeasible Schedule Generated by Absolute Lower Bounds on Functional Units.....	173
Figure 6.4 Pseudo-code for ASAP Scheduling Algorithm .....	175

Figure 6.5 Pseudo-Code for ALAP Scheduling Algorithm .....	176
Figure 6.6 ASAP and ALAP Scheduling of 3 Tap FIR Filter DFG.....	176
Figure 6.7 Illustration of Schedule Generated Using Crude Lower Bounds as Limits on Functional Units .....	179
Figure 6.8 Pseudo-code for List-Scheduling Algorithm.....	180
Figure 6.9 Slack Times for 3 Tap FIR Filter.....	181
Figure 6.10 Pseudo-Code for Lower Bounds Estimation of Functional Units .....	183
Figure 6.11 Estimates of Number of Multipliers for Range of DFGs Generated for Low-Power Exploration of 8th Order Avenhaus Filter .....	185
Figure 6.12 Estimates of Number of Adders for Range of DFGs Generated for Low-Power Exploration of 8th Order Avenhaus Filter .....	186
Figure 6.13 Determination of Minimum Lifetime of a Variable E .....	188
Figure 6.14 Pseudo-Code for Estimation of Minimum Number of Registers.....	189
Figure 6.15 Register Accesses in an Example DFG.....	193
Figure 6.16 Overview of Enhanced Area and Capacitance Estimation Procedures .....	194
Figure 7.1 Reversible Exploration.....	198
Figure 7.2 Pseudo-Code for Remove-Pipeline Mutation.....	199
Figure 7.3 Application of Remove-Pipeline Mutation .....	200
Figure 7.4 Effect of Increasing Elitism Application on Power Consumption of Benchmark Designs .....	202
Figure 7.5 Effect on GA Iteration Length of Increasing Elitism Application.....	203
Figure 7.6 Example Graph Showing the Effect of Parameter A.....	210
Figure 7.7 Plot of S/N Ratios for AVEN8DI DFG .....	216
Figure 7.8 Plot of S/N Ratios for AVEN8PA DFG.....	217
Figure 7.9 Domination of Fitness by Super-Fit Individual Chromosome .....	220
Figure 7.10 Effect of Ranking on Simple Population Representation.....	221
Figure 7.11 Assignment of Rank to a Population of Individuals .....	223
Figure 7.12 Pseudo-Code for the Ranking Algorithm.....	223
Figure 8.1 GA Performance for 8TAPFIR Non-Unfolded .....	233
Figure 8.2 GA Performance for AVEN8DI Non-Unfolded.....	234
Figure 8.3 GA Performance for AVEN8PA .....	235
Figure 8.4 GA Performance for DCST Unfolded .....	236
Figure 8.5 GA Performance for BIQAUD3 .....	237
Figure 8.6 GA Performance for GMLAT4 Non-Unfolded.....	239
Figure 8.7 GA Performance for ELLIP5 .....	240
Figure 8.8 GA Performance for LMS5 Non-Unfolded.....	241
Figure 8.9 GA Performance for VOLTERRA .....	242
Figure 8.10 GA Performance for ORTH2LAT Unfolded Design.....	243
Figure 8.11 Power Reduction and Area Increase for Ten Benchmark Designs.....	244

Figure 9.1 Example Pareto-Surface for 2-Dimensional Optimisation .....	252
Figure 9.2 Identification of Pareto-Points in GALOPS .....	254
Figure 9.3 Example of different DFGs with the Same Power and Area Characteristics .....	256
Figure 9.4 Pareto-Surface for 8TAPFIR .....	256
Figure 9.5 Pareto-Surface for AVEN8DI .....	257
Figure 9.6 Pareto-Surface for AVEN8PA .....	258
Figure 9.7 Pareto-Surface for DCST .....	259
Figure 9.8 Pareto-Surface for BIQUAD3 .....	260
Figure 9.9 Pareto-Surface for GMLAT4 .....	260
Figure 9.10 Pareto-Surface for ELLIP5 .....	261
Figure 9.11 Pareto-Surface for LMS5 .....	262
Figure 9.12 Pareto-Surface for VOLTERRA .....	263
Figure 9.13 Pareto-Surface for ORTH2LAT .....	264
Figure 9.14 Typical Pareto-Surface Chart .....	265
Figure 10.1 Illustration of an Optimisation Process .....	273
Figure 10.2 Pseudo-Code for Simulated Annealing Algorithm .....	275
Figure 10.3 Comparison of GA and SA Search .....	278
Figure 10.4 Pareto-Surface Information Generated with SA and GA .....	280

## List of Tables

Table 5.1 Architectural Level Components .....	132
Table 5.2 Application Rates for Analysis .....	144
Table 5.3 Results without Unfolding of Prototype GALOPS .....	150
Table 5.4 Results with the Application of Unfolding .....	157
Table 6.1 Estimates on Minimum Number of Functional Units for Benchmark DFGs .....	184
Table 6.2 Switched Capacitance Estimations of Hardware Units .....	191
Table 7.1 Mode Power-Consumption for Benchmark Designs with Varying Elitism Parameter .....	202
Table 7.2 Example Orthogonal Taguchi Array .....	207
Table 7.3 Set of Results Produced from Taguchi Experiments .....	208
Table 7.4 Signal-To-Noise Ratio Analysis for Experiment Results .....	209
Table 7.5 Possible Application Rates of Each Operator .....	211
Table 7.6 Taguchi Array for Optimisation of Genetic Operator Application Rates .....	213
Table 7.7 Taguchi Results for AVEN8DI DFG .....	214
Table 7.8 Taguchi Results for AVEN8PA .....	215
Table 7.9 S/N Ratio Analysis for AVEN8DI DFG .....	216
Table 7.10 S/N Ratio Analysis for AVEN8PA DFG .....	216
Table 7.11 Selected Application Rates from Taguchi Analysis of Two DFG Designs .....	217
Table 7.12 Selected Application Rates for GALOP System .....	218
Table 7.13 Mode Power Consumption Obtained with FPS and Ranking Selection Schemes .....	224
Table 8.1 Overall Power Reduction and Area Increase for Benchmark Designs .....	230
Table 10.1 Comparison of Results Obtained with Gradient Search and GALOPS .....	270
Table 10.2 Comparison of Results Obtained with SA and GA .....	278

## Abbreviations

<b>ALAP</b>	As Late As Possible
<b>ANSI</b>	American National Standards Institute
<b>ASAP</b>	As Soon As Possible
<b>ASIC</b>	Application Specific Integrated Circuit
<b>AVEN8DI</b>	8 <sup>th</sup> Order Avenhaus Filter Direct Form
<b>AVEN8PA</b>	8 <sup>th</sup> Order Avenhaus Filer Parallel Form
<b>BIQUAD3</b>	3 <sup>rd</sup> Order Bi-Quadratic Filter
<b>BP</b>	Break Point
<b>CAD</b>	Computer Aided Design
<b>CMOS</b>	Complementary Metal Oxide Semiconductor
<b>CP</b>	Critical Path
<b>CPL</b>	Complementary Pass-Gate Logic
<b>DARPA</b>	Defence Advanced Research Projects Agency
<b>DBT</b>	Dual Bit Type
<b>DCST</b>	Discrete Sine/Cosine Transform
<b>DCT</b>	Discrete Cosine Transform
<b>DFG</b>	Data Flow Graph
<b>DSP</b>	Digital Signal Processing
<b>ELLIP5</b>	5 <sup>th</sup> Order Elliptic Wavelet Filter
<b>FDS</b>	Force Directed Scheduling
<b>FIR</b>	Finite Impulse Response
<b>FIR3</b>	3 <sup>rd</sup> Order FIR Filter
<b>FIR8</b>	8 <sup>th</sup> Order FIR Filter
<b>FPS</b>	Fitness Proportionate Selection
<b>FSM</b>	Finite State Machine
<b>GA</b>	Genetic Algorithm
<b>GALOPS</b>	Genetic Algorithm for Low Power Synthesis
<b>GMLAT4</b>	4 <sup>th</sup> Order Gray-Markel Lattice Filter
<b>HLT</b>	High Level Transformation
<b>I/O</b>	Input/Output
<b>IC</b>	Integrated Circuit
<b>IEEE</b>	Institute of Electrical and Electronic Engineers
<b>IIR</b>	Infinite Impulse Response
<b>ILP</b>	Integer Linear Programming
<b>LAT2</b>	2 <sup>nd</sup> Order Lattice Filter
<b>LMS5</b>	5 <sup>th</sup> Order Least Mean Square Algorithm
<b>LSB</b>	Least Significant Bit
<b>MIPS</b>	Millions of Instructions Per Second
<b>MOGA</b>	Multi-Objective GA
<b>MSB</b>	Most Significant Bit
<b>NDS</b>	Non Dominated Solution
<b>NP</b>	Non-Polynomial

<b>ORHT2LAT</b>	2 <sup>nd</sup> Order Orthogonal Lattice Filter
<b>PDA</b>	Portable Digital Assistant
<b>PFA</b>	Power Factor Approximation
<b>PGA</b>	Parallel Genetic Algorithm
<b>RAM</b>	Random Access Memory
<b>RCS</b>	Resource Constrained Scheduling
<b>RTL</b>	Register Transfer Level
<b>SA</b>	Simulated Annealing
<b>SPA</b>	Stochastic Power Analysis
<b>TCS</b>	Time Constrained Scheduling
<b>TRCS</b>	Time and Resource Constrained Scheduling
<b>US</b>	United States
<b>UWN</b>	Universal White Noise
<b>VHDL</b>	Very High Speed Integrated Circuit High-level Description Language
<b>VLSI</b>	Very Large Scale Integration
<b>VOLTERRA</b>	Volterra Filter
<b>WWW</b>	World Wide Web

# Nomenclature

## Chapter 2

$I_{SC}$	Short-circuit current
$I_{DY}$	Dynamic switching current
$V_{DD}$	Supply voltage
$I_{DD}$	Supply current
$C_L$	Load capacitance
$V_{in}$	Input voltage
$V_{out}$	Output voltage
$P_{average}$	Average power
$P_{switching}$	Switching power
$P_{short-circuit}$	Short-circuit power
$P_{leakage}$	Leakage power
$C$	Capacitance
$f$	Frequency
$k$	Switching activity
$C^*$	Switched Capacitance
$L$	Inductor
$N$	Unfolding factor
$U \rightarrow V$	Connect from node U to V in a DFG
$D$	Delay node in a DFG
$T$	Execution time of a DFG
$\lceil \cdot \rceil$	Ceiling function

## Chapter 3

$V_{DD}$	Supply voltage
$C^*$	Switched Capacitance
$P_{average}$	Average power
$P_s(x)$	Signal Probability
$P_t(x)$	Transition Probability
$f$	Frequency
$C_i$	Capacitance of node i
$P_t(X_i)$	Transition probability of node i

## Chapter 4

$Num$	Number of gene in a chromosome
$Fnc$	Function of gene in a chromosome
$DEL$	Delay node in a DFG
$linear\_fitness$	Linear fitness of a solution
$fitness$	Fitness of a solution
$min(fitness)$	Minimum fitness in a population
$max(fitness)$	Maximum fitness in a population
$M$	Number of solutions in a population
$N$	Unfolding factor
$L$	Number of nodes in a DFG

$V_{DD}$	Supply voltage
$C^*$	Switched Capacitance
<b>Chapter 6</b>	
$abs\_minRi$	Absolute lower bound on resources of type i.
$add\_min$	Estimate of minimum number of add resources
$C\_step$	Control step
$\lceil \cdot \rceil$	Ceiling function
$C_{functional\_units}$	Capacitance of functional units in datapath
$C_i$	Capacitance of resource of type i
$C_{interconnect}$	Capacitance of interconnect in datapath
$Cp$	Critical Path length
$C_{registers}$	Capacitance of registers in datapath
$C_{single\_register}$	Capacitance of a single register
$C_{total}$	Total capacitance of datapath
$E$	Variable passed between two nodes in a DFG
$I$	Resource or node type
$k$	Number of signals feeding node or resource
$m$	Number of nodes in a DFG
$max\_Ri$	Upper bound on number of resources of type i
$min\_C\_step$	Minimum length of critical path
$mult\_min$	Estimate of minimum number of multiplier resources
$n$	Number of connection nets in a DFG which require a variable to be stored
$N$	Number of operations in the DFG
$Ni$	Number of operations of type i in the DFG
$Ri$	Number of hardware resources of type i
$S_{ALAP}$	ALAP time of operation S
$S_{ASAP}$	ASAP time of operation S
$slack\_time$	The mobility of an operation in a DFG
$t_{lifemin}$	Minimum lifetime of a variable
$Total_{control\_steps}$	Maximum number of control steps in a schedule
$X$	Number of control steps in a schedule
<b>Chapter 7</b>	
$S/N$	Signal to noise ratio of Taguchi experiment result
<b>Chapter 10</b>	
$T$	Temperature
$k$	Boltzmann's constant
$\partial E$	Change in energy between two states
$p(\partial E)$	Probability of selection of a solution

## **Chapter 1 – Introduction**

### **1.1 Motivation for Low-Power VLSI**

The design of VLSI (Very Large Scale Integration) devices has historically concentrated on providing higher levels of performance while reducing cost. The increasing development of smaller feature sizes has enabled increased integration levels to produce designs with millions of transistors, enabling existing designs to be implemented with reduced area. This reduction in area can translate to a smaller die size, reducing packaging costs and increasing the fabrication yield. Often, reduced design area is exploited to enable more features to be incorporated on the same size VLSI device ('system on a chip'), thus increasing performance without significantly increasing the packaging cost of the device. Improvements to the VLSI fabrication process have enabled an increase in wafer and die size, further increasing the amount of transistors that can be integrated onto a single device. In addition to increased integration the clock frequencies of VLSI devices are continually increasing in order to squeeze the maximum possible performance out of these highly complex designs.

One effect of increasing integration levels and clock-frequencies is the increased power consumption (and heat dissipation) of VLSI devices. In recent years the importance of power consumption as a VLSI implementation parameter has increased significantly due to several main factors.

One of the major commercial trends to have influenced the importance of power consumption is the explosion in the portable electronics market [Sheng92, Chan94a, Chan92]. Personal computers, mobile phones, pagers and Personal Digital

Assistants (PDAs) are examples of portable systems that have achieved significant commercial success. The portable system share of the computing and communication market is only expected to increase in the future [Singh95, Yeap96]. Future developments in portable systems are expected to see the development of Portable Communication Systems [Sheng92, Snyder94], integrating communication and computation facilities in a single unit.

For portable electronic systems the continuous operating time provided by a single battery charge is a major factor in commercial success. A short operating time will remove the main feature of the system, its portability, and hence reduce its attractiveness to consumers. The operating lifetime of a portable system is dependent on both its power consumption and the battery capacity. While battery technology has increased considerably since the development of portable systems it is not expected to offer significant advances in the near future [Singh95, Chuang98, Bella95a]. Providing longer operating times through increased battery capacity means using a larger, heavier battery. This is undesirable as the weight of portable systems is also of major concern to consumers. Therefore, the increase of operating time of a portable electronic system requires reduction of its power requirements.

In addition to increasing the operating time of existing portable systems, there is an increasing expectation from consumers for these systems to offer the same features and performance as their non-portable versions, increasing the burden on the already lean power-budget [Gary96]. Therefore, power reduction is not only required for today's portable systems, but also to enable the development of the next generation of portable systems with increased functionality.

While the VLSI devices consume a significant portion of the overall power budget, power reduction of portable systems has targeted all areas of the system, such as the display [Ajluni95] and communication modules [Mcgrath95, Sheng92, Larson98]. The expansion in functions required in portable systems (multimedia, voice-mail, etc.) has increased the requirement for dedicated Digital Signal Processors (DSPs). The advantage of DSPs over generic microprocessors is that they can be designed for low-power operation for a specific task, leading to significant reductions in power [Burd96, Arslan95, Lyon93].

The reduction of the power burden in portable systems is not the only driving force behind low power VLSI design. Increased power consumption has led to devices that dissipate a considerable amount of heat, which subsequently leads to an increase in the packaging costs of the device. The requirement for heat sinks, fans and heat management systems significantly adds to the cost of the device, affecting its commercial viability. Reduction of power and associated heat dissipation can reduce device cost and hence increase its commercial success [Bursky95]. In addition, high temperature devices can have a significantly reduced reliability, leading to an increased probability of device failure [Singh95, Bella95a, Rabaey96, Yeap96].

Recently, power consumption and heat dissipation have been identified as a potential limitation to increased levels of integration and clock frequencies. Power is beginning to become a limiting factor in the design of the next generation of high-performance devices [Szek98, Chuang98, Pedram95].

Power reduction has been targeted as being of paramount importance by the VLSI industry. The US Advanced Research Projects Agency has initiated a program to develop the technology and design infrastructure required for low power VLSI,

with the aim of producing power savings of orders of magnitude – from 100 milliwatts/MIPS to 10 microwatts/MIPS [Lem94, Fried94]. Companies such as Toshiba [Kunii95], Motorola [Gary94], IBM [Ikeda95], Intel [Guerra98, Intel98] and DEC [Gowan98] have implemented schemes for the development of low-power devices and portable systems; providing additions to their existing product range and new systems to satisfy the demand for higher performance at lower energy. Figure 1.1 illustrates the power trends in the Intel family of processors [Intel98, Bella95a, Tiwari98].

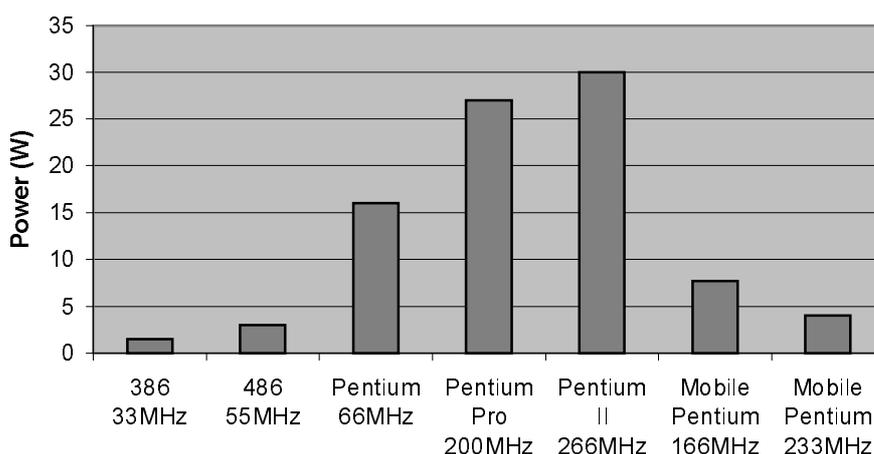


Figure 1.1 Power Consumption Trends in Intel Processors

As processor performance increased (due in part to increased integration levels and higher clock speeds) power dissipation initially increased. However, recognition of the power restraints on portable systems led to the development of the Intel Mobile Processor family for portable systems. The Mobile processors have benefited from reduced feature sizes,  $0.35\mu\text{m}$  and  $0.25\mu\text{m}$ , which enabled lower supply voltages (less than 2V). In addition the processors were designed for low-

power operation, incorporating power saving techniques such as different standby levels depending on application requirements. As the figure illustrates the Pentium 66MHz has a power consumption of 16 Watts compared to the Mobile Pentium 233MHz power consumption of 4 Watts which offers significantly increased processing power. Similar design schemes have led to the development of low-power versions of the Motorola PowerPC (PowerPC 603 [Gary94]) and the DEC Alpha (DEC Alpha 21264 [Gowan98]). These design examples illustrate the benefit of targetting power as an objective parameter in the design process, leading to reduced power consumption while preserving the trend towards increased computational throughput.

The factors discussed in this chapter highlight the reasons why power consumption has become an important parameter in VLSI devices. For portable and high-performance systems it has joined the traditional parameters of area and speed as an objective parameter for VLSI devices [Bella95, Bella95b].

### 1.2 Requirements for Low-Power Computer Aided Design Tools

The requirement for low power VLSI devices significantly increases the complexity of the VLSI design and fabrication process. Amendments to the fabrication process (such as process scaling) can offer reductions in power consumption but they require costly modifications [Singh95]. Voltage reduction through feature size reduction (voltage scaling) takes advantage of reduced feature sizes requiring lower electric fields and hence lower voltages to obtain the same throughput. While voltage scaling has been used extensively (witness the industries

move to a 3.3V standard) it is not expected to provide the required power reductions while maintaining the increase in throughput [Meng98, Singh95, Landman94a].

This has led to the increasing recognition that the incorporation of power consumption as a standard design parameter is expected to return the largest benefits with the smallest increases in cost [Singh95, Roy95, Horo94]. Effectively this means adopting a design for power strategy.

The inclusion of power as a design parameter increases the burden on the VLSI design engineer who is dealing with increasingly complex devices with more demanding implementation and design-time requirements. Computer Aided Design (CAD) tools are frequently used in VLSI design to deal with the sheer complexity of the devices. Traditional CAD tools typically contain area and speed optimisation but the consideration of power consumption increases the optimisation task from 2 dimensions (area and speed) to 3 dimensions (area, speed and power). This considerably increases the complexity of the optimisation task; in addition, the three objectives are typically in competition i.e. low power design is a multi-objective task which requires trade-offs to be made between area, speed and power. Therefore there is a requirement for CAD tools which can consider all three parameters to enable the VLSI designer to produce designs that meet area, speed and power specifications. These CAD tools must incorporate techniques for power manipulation (to reduce power) and power estimation (to provide feedback to determine the effect of power modifications).

There are a variety of techniques that target low power VLSI at all levels of the design process, from floorplanning and layout to high-level algorithm

modification. High-level consideration of power as an initial design parameter is expected to provide the greatest power reduction in terms of return on investment [Rabaey96, Singh95, Bella95b, Chan98]. Lower level techniques, such as those at the circuit or logic level, tend to have a limited, local impact on power consumption as they operate within constraints specified by higher levels of the design process. High-level power design does not operate within such constraints so it has greater freedom to manipulate the design to explore the low power solution space.

While commercial CAD tools are beginning to address the need for low power design very few currently offer high-level power estimation and even fewer offer optimisation [Coudert96, Landman94a]. There is a requirement for increased development of high-level CAD tools that consider power as an objective parameter [Macii97]. The disadvantage of high-level power reduction is that high-level VLSI design is already a complex problem; many of the tasks are NP-complete, requiring heuristic algorithms to determine satisfactory solutions to the problems. The complexity of the high-level design problem is compounded by the addition of power as an objective. A high-level low power CAD tool requires the ability to handle the complex solution space to produce low power designs.

The ability to handle complex solution spaces is a requirement at many levels of the VLSI design process. The difficulty of determining good VLSI design solutions, in a reasonable amount of time has resulted in significant research into the application of heuristic optimisation techniques, both to improve the design process and the resulting designs. Simulated Annealing (SA) is one such technique, a heuristic algorithm inspired by the physical process of cooling, it has been successfully used within the complex VLSI routing process [Kirk83, Vecchi83] and

VLSI synthesis [Neil]. Tabu Search [Reeves95, Glover98] is a heuristic technique which explores the solution space by building upon information gained from previous exploratory moves. They have only recently begun to be used in VLSI design optimisation problems at the physical design level.

Genetic Algorithms (GAs) are a search and optimisation technique inspired by the processes of natural evolution. They have been successfully applied to areas of VLSI physical design automation, solving complex problems and improving on results obtained with other techniques such as SA. This proven track record in VLSI design, together with their proven optimisation capabilities in such complex problems, makes them a good choice for investigation as the basis of a low power design system.

For the purposes of this thesis there is only scope to perform an indepth investigation of one of the afore mentioned techniques. The evidence of the success of GAs in solving complex, multi-objective VLSI problems has led to their selection as the optimisation technique investigated in this thesis.

The objective of this research is to investigate the combination of high-level power exploration with artificial intelligence search techniques. This thesis presents the development and analysis a low-power CAD tool that uses high-level algorithmic modifications for power reduction. The tool incorporates a GA that manipulates high-level algorithmic descriptions, incorporating modified genetic search techniques, to explore the low-power solution space.

The tool targets the power reduction of DSP devices implemented as Application Specific Integrated Circuits (ASICs) in CMOS technology. The importance of low-power DSP devices, in particular those developed for specific

functions, was discussed in section 1.1. ASIC DSPs offer the potential to execute complex signal processing functions with greater efficiency and lower-power consumption than general purpose processors. CMOS is chosen, as it is the most commonly used VLSI technology today. This is primarily because of its excellent scaling qualities and its low power consumption when not switching.

### 1.3 Overview of Thesis

The thesis is divided into 11 chapters. This chapter has presented the requirement for low power VLSI devices and the associated need for CAD tools to assist the design engineer in achieving low power targets.

Chapter 2 discusses the problem of high-level power synthesis. Traditional high-level VLSI synthesis is reviewed to present its advantages relating to design exploration and modification at early levels of the design process. The contributing factors of power dissipation in CMOS devices are discussed before presenting an overview of techniques that target low-power VLSI design. The discussion presented in chapter 2 concludes that although low power is best tackled at all levels of the design process, incorporating all techniques into a unified framework, high-level low power design offers the greatest benefits in terms of power reduction. Techniques for high-level power reduction are discussed before presenting the technique used in this thesis: high-level behavioural transformation. The technique incorporates high-level transformations to manipulate the power consumption of high-level algorithms. The transformations incorporated into the CAD tool presented in this thesis are also presented.

Chapter 3 examines the other requirement of a low power CAD tool: power estimation. Power estimation techniques are reviewed at all levels of the design process to illustrate the trend of high-level power estimation having to trade accuracy for speed. High-level power estimation techniques, though still in their infancy, are discussed within the context of feedback to a high-level power estimation system.

Chapter 4 presents the concepts of the GA search and optimisation technique. Examples of the technique to solve VLSI engineering problems, including power reduction, are discussed. The chapter discusses the complexity of high-level behavioural transformation for low power, illustrating the requirement for efficient tools, such as a GA, that can deal with the complex solution space inherent in low power design.

Chapter 5 contains the development of the low-power design tool, incorporating high-level power design and estimation into a genetic framework. The prototype design is tested with a small set of benchmark circuits to illustrate its effectiveness in reducing the power consumption of high-level signal processing applications.

Chapter 6 presents modifications to the tools' power estimation module to provide more accurate feedback during the search and modification process. The improved techniques incorporate high-level synthesis methods and high-level power analysis techniques.

Chapter 7 targets the optimisation of the GA processes. Various modifications to the standard GA are discussed and analysed to determine their effects on the low power tool are beneficial.

Chapter 8 presents ten benchmark signal-processing designs with the results obtained by the developed tool for power reduction. The results illustrate the ability of the tool to reduce the power consumption of the high-level designs.

Chapter 9 discusses the concept of multi-objective optimisation, where conflicting parameters such as area and speed are required to be concurrently optimised. The GA can be used to generate invaluable information to the VLSI designer illustrating a range of trade-offs that can be made. The chapter includes pareto-surface charts for each of the ten benchmark designs, illustrating the area-power trade-offs that can be made.

Chapter 10 compares the GA technique with other search and optimisation techniques, to illustrate its effectiveness in producing low power designs and exploring the solution space to present design trade-off information.

Chapter 11 presents conclusions and future work.

### 1.4 Publications Arising from This Research

1. M.S. Bright and T. Arslan, “A Genetic Framework For The High-Level Optimisation Of Low Power VLSI DSP Systems”, IEE Electronics Letters, 20th June 1996, Vol. 32, No. 13, pp. 1150-1151
2. T. Arslan, E. Ozdemir E., M.S. Bright and D.H. Horrocks, “Genetic Synthesis Techniques for Low-Power Digital Signal Processing Circuits”, IEE Colloq. On Digital Synthesis, London, UK, 15th Feb 1996, pp.7/1-7/5
3. M. S. Bright and T. Arslan, “A Genetic Algorithm for the High-Level Synthesis of DSP Systems for Low Power”, Proc. IEE/IEEE Conf. on

Genetic Algorithms in Engineering Systems, Innovations and Applications (GALESIA '97), Glasgow, UK, 2-4 Sept. 1997, pp. 174-179

4. M. S. Bright and T. Arslan, “Transformational-Based Synthesis of VLSI Based DSP Systems for Low Power Using a Genetic Algorithm”, IEEE Int. Symposium on Circuits and Systems, ISCAS 98, Monterey CA, 31 May – 3 June 1998
5. M. S. Bright and T. Arslan, “Low-Power High-Level DSP System Methodologies and Techniques: Impact on CAD”, IEE UK Low-Power Forum, Sheffield UK, Sept. 16<sup>th</sup>-17<sup>th</sup> 1998, pp. 7.1-7.5
6. M. S. Bright and T. Arslan, “Supply Voltage Reduction Through High-Level Design Techniques”, IEE UK Low-Power Forum, Sheffield UK, Sept. 16<sup>th</sup>-17<sup>th</sup> 1998, pp. 10.1-10.5
7. M. S. Bright and T. Arslan, “Multi-Objective Design Strategies for High-Level Low-Power Design of DSP Systems”, IEEE Int. Symposium on Circuits and Systems, ISCAS 99, Florida
8. M. S. Bright and T. Arslan, “Synthesis Of Low-Power DSP Systems Using A Genetic Algorithm”, IEEE Trans. On Evolutionary Computation (In Preparation)

## **Chapter 2 – Synthesis for Low Power VLSI**

The complexity of VLSI devices has increased greatly since their inception, with modern devices typically containing millions of transistors. Exploitation of these advances in VLSI technology requires advanced design tools to handle the increased complexity. In addition, VLSI designers are being asked to produce designs with higher performance and zero defects, while reducing the design time. CAD tools have been, and will continue to be, instrumental in achieving these goals [Casa80, Micheli90]. The addition of objective parameters such as area, power, etc. further increases the burden on the VLSI design engineer, increasing the need for effective CAD tools.

A typical ASIC design methodology [Gajski94] starts with a set of device requirements developed in conjunction with designers and the marketing department. From this specification a team of designers produce an architectural-level specification of the chip which resolves the requirements specification into high-level functional blocks. This is then passed onto logic and layout designers who convert each block into a logic- or circuit-level schematic. VLSI schematic capture tools can then be used to simulate and synthesise the design down to the device level. In this typical design methodology the CAD tools are not used until the architectural level layout has been specified.

One of the most important developments of recent years in VLSI design has been the advent of high-level synthesis systems and CAD tools. High-level synthesis enables the design to be described purely in a behavioural form i.e. a

specification of its desired function. CAD tools that employ high-level synthesis methodologies can then be used to synthesise the design down to the device level, a complete automation of the VLSI design and fabrication process.

This chapter discusses high-level synthesis and low-power design techniques. The discussion on high-level design concludes that high-level synthesis offers considerable advantages as part of the VLSI design process. Similarly, power reduction at the high-level has the greatest effect on overall power consumption. Therefore, high-level synthesis with low-power objectives offers the combination of these advantages. Therefore, the work presented in this thesis considers power reduction as part of the high-level synthesis process.

This chapter describes the properties of high-level synthesis and high-level CAD tools in section 2.1. Section 2.2 outlines the main factors contributing to CMOS power as background for the following discussion on low-power design techniques. Section 2.3 presents an overview of techniques that have been developed to reduce the power consumption of VLSI devices. Section 2.4 specifically discusses high-level power reduction techniques including those used in the tool presented in this thesis. The techniques use high-level behavioural transformations to reduce power consumption at the algorithmic level. Section 2.5 presents the specific set of high-level transformations used within the tool.

### **2.1 High-Level Synthesis**

The term ‘high-level synthesis’ is usually used to describe the process of turning an abstract behavioural description of a desired function or algorithm into an architectural-level specification. The behavioural-description describes the input and output behaviour of the algorithm in terms of operations and data transfers,

separate from any VLSI implementation details. The architectural level maps the operations and data onto functional units and registers, effectively producing a block diagram of the VLSI device.

High-level synthesis is a relatively new addition to traditional VLSI synthesis techniques. In the 1970s interest in CAD and design synthesis was mostly at lower levels of the design process, such as automation of layout and routing. In the 1980s interest in high-level synthesis techniques increased partly due to the increasing complexity of VLSI devices, resulting in extensive work on high-level synthesis techniques [McFar90].

The tasks of high-level synthesis can be conveniently grouped into three main steps [Micheli94]:

1. Compile the behavioural specification into an internal representation.  
The internal representation is a convenient format for the exploration of the solution space as well as the execution of the high-level synthesis tasks.
2. Perform optimising transformations on the internal representation of the behavioural specification. Depending on the CAD tool, the transformations are used to produce an optimal design or to provide a range of alternative designs to the designer so that the optimal can be selected.
3. Perform the high-level synthesis tasks such as scheduling and allocation to produce the layout of the hardware structure [Micheli94].

The tool presented in this thesis concentrates on the application of the first two steps to produce optimal high-level algorithms for low-power implementation.

During this process the tool performs many of the high-level synthesis tasks of step 3 to estimate characteristics of the hardware structure. The system does not actually produce hardware structures as it is intended as the first step in a more comprehensive high-level synthesis system. There are a number of steps that can be applied during step 3 to reduce power consumption, as discussed in section 2.4. The tool in this thesis concentrates on the reduction of power in step 2. The presented system produces optimised high-level designs to enable the subsequent application of high-level procedures which optimise power during step 3.

In recent years there has been a trend to automating synthesis at higher and higher levels of the design process due to the advantages of high-level synthesis. High-level synthesis is seen as one of the main techniques to reduce the design-cycle time of complex VLSI devices. A long design cycle can render a project obsolete by the time it reaches the marketplace. This is especially so in the case of military applications where new technology is required to be available as soon as possible. The US Defence Advanced Research Projects Agency (DARPA) has recognised that the development of high-level synthesis tools is one of the key requirements in improving the ASIC design process [Madis96, Madis96a]. The automation of high-level synthesis tasks will produce designs more quickly [McFar83]; therefore, designs have greater chance of meeting their market window and hence maximising profits in commercial applications.

The increased complexity of ASIC applications is requiring greater concentration on the verification of the design, to ensure it is error free and matches the specification, at all levels of the design process. High-level synthesis makes it possible to verify the device's function far earlier in the design process than lower level tools. The behavioural descriptions are less complex than their corresponding

architectural-level descriptions, making design simulations and any necessary re-designs faster to implement. Once the high-level behavioural specification has been verified high-level synthesis presents the ability to use automated CAD tools throughout the rest of the design process. As these tools use proven techniques they are less prone to introducing errors into the design, reducing or eliminating the time required for costly redesigns [Camp90, McFar90].

One of the most important advantages offered by high-level synthesis is the ability to explore the design space at very early stages in the design process. Decisions made at high-levels have the greatest affect on the VLSI device's implementation parameters such as area and power [Micheli90, Bentz97, McFar90, Guerra98, Sato94]. A good high-level synthesis tool presents alternative designs in a reasonable amount of time so that the effect of high-level decisions can be explored and high-level trade-offs performed [McFar90]. One of the main aims of a high-level CAD tool is to produce an optimal design. Earlier high-level tools produced sub-optimal designs that required considerably larger area than those produced by manual techniques [McFar83, McFar90]. Even with advanced high-level synthesis tools, manual optimisation may still be required to extract the maximum possible performance [Gajski94]. The continuing development of high-level CAD tools is aimed at optimising both the devices produced and the time taken to perform the synthesis process with the aim of outperforming manual synthesis and optimisation techniques.

A number of high-level CAD synthesis tools have been developed to take advantage of the properties described in the previous paragraphs. The CATHEDRAL system [Guerts91] specifically targets the synthesis of DSP devices, applying high level transformations to explore the solution space. Other systems

such as FLAMEL [Trick87], HYPER [Rabaey90, Reese94, Chan95, Bentz] and MIMOLA [Bhask90] also apply high-level transformations during the synthesis process to obtain designs optimal in terms of desired parameters such as area, speed, power, etc. The high-level transformations are used to modify the characteristics of the behavioural descriptions, changing the VLSI implementation characteristics [Walker89]. These tools are academic based research projects; high-level synthesis tools have only recently become of interest to the commercial market [Gajski94]. In-house, high-level CAD tools have been used to develop practical devices and commercially available high-level synthesis tools are in development. One of the key reasons for the increase in commercial interest is the extra VLSI device criterion required to be optimised during the synthesis process, such as operation with reduced power consumption [Walker94]. The experience of academic tools and the growing commercial acceptance of high-level synthesis illustrates its importance as a VLSI design technique; not only to harness the power available in today's multi-million transistor devices but also to produce designs with the required optimal operation characteristics.

### *2.1.1 Behavioural Level Problem Representation*

As discussed previously, step 1 in high-level synthesis is to represent a behavioural description or algorithm of the problem in a suitable format. High-level languages such as VHDL [Ash95, Bhask90], VERILOG [Gajski94] and even PASCAL [Trick87] have been used to describe high-level algorithms. VHDL is an IEEE recognised standard and, as such, has wide support in high-level synthesis. Whichever method is used to describe the behavioural algorithm, it is usually transformed into an internal representation format within the high-level synthesis

tool. The typical format for internal representation is the Data Flow Graph (DFG) [Casa80, McFar90, Rabaey90, Gela93, Sriv95 and Park98]. The properties of a DFG are described in this section.

A DFG is a directed graph that represents a complete iteration of the signal-processing algorithm. An example of a simple DFG, illustrating a 2<sup>nd</sup> order Finite Impulse Response (FIR) filter operation, is shown in Figure 2.1.

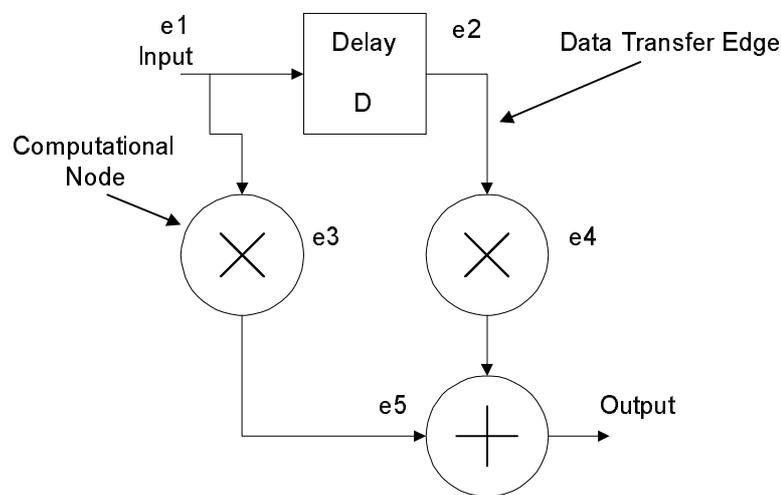


Figure 2.1 Example Data Flow Graph

Within the DFG, each node represents an operation on data by an element, e.g. addition, multiplication or delay. The directed edges represent data transfer between nodes, in a specific direction. In this example, each node has been numbered from e1 to e5.

A DFG inherently contains the precedence constraints between operations in a signal-processing algorithm. This precedence information indicates which operations have the potential to be executed concurrently. The precedence information is also important for determining which operations are critical to the

timing of the algorithm. A critical operation is an operation along the ‘Critical Path’ (CP) of the DFG. The CP limits the maximum operating speed of a DFG because each element along this path must have completed its function before the next processing iteration can occur. Therefore the maximum sampling frequency of the DFG is the inverse of CP period, as illustrated in Figure 2.2.

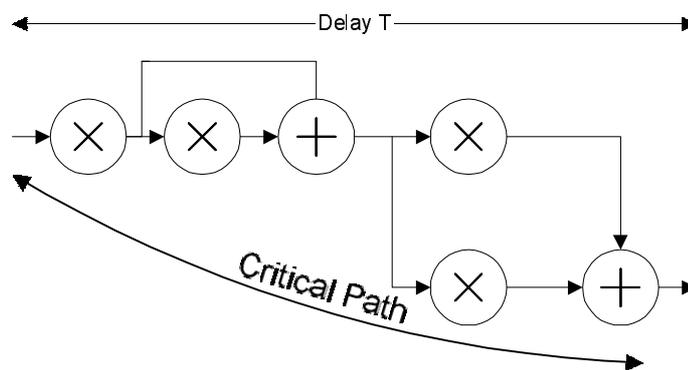


Figure 2.2 Example Illustrating Critical Path

Within a VLSI device, concurrent operations can be executed in parallel on separate processing units. For example, in Figure 2.2 there is no directed edge connecting the multiplication nodes. This implies that there is no precedence relation between these two operations and they can, therefore, be performed concurrently, independently of each other. This example illustrates how the DFG provides an intuitive view of the amount of parallelism available in a signal-processing algorithm. Concurrent execution of operations enables the algorithm to be processed at the maximum possible speed. In contrast, performing operations sequentially on the same unit may reduce speed but have the advantage of reducing area.

An important aspect of a DFG is that the precedence information is not stored implicitly, as it is with a serially executed program, listing a single operation on each line, with a clearly defined order. The DFG allows flexible ordering of the operations, within the bounds set by the precedence constraints. This enables area and speed decisions to be taken when the hardware is being designed, not when the algorithm is being designed.

This ability of a DFG to fully encapsulate a signal-processing algorithm without prescribing hardware implementation details has made the DFG a typical form of algorithm representation within high-level VLSI synthesis systems [Chan95, Gajski94, Guerts91, Trick87, Gela93].

### **2.2 Power Dissipation in CMOS Devices**

This section presents a brief summary of power dissipation in CMOS devices in order to explain the techniques used for power reduction. Figure 2.3 illustrates a typical CMOS digital circuit showing the current and voltage components.

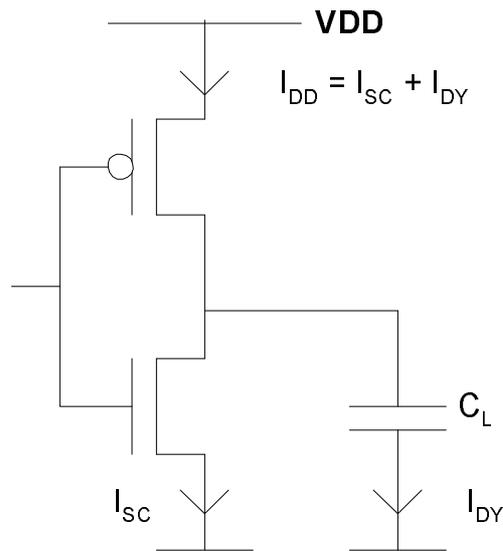


Figure 2.3 Typical CMOS Digital Circuit

$I_{SC}$  denotes the short-circuit current during the signal transition and  $I_{DY}$  is the dynamic current that is consumed while the transistor is switching. The leakage current is not illustrated here. The average power consumption of such a device is given by [Arslan95]:

$$P_{\text{average}} = P_{\text{switching}} + P_{\text{short-circuit}} + P_{\text{leakage}} \quad (2.1)$$

Equation (2.1) illustrates that the average power is a combination of the dynamic power (dissipated during switching, to perform the desired function) and the static power (associated with the short-circuit and leakage currents). The static power is a factor of the VLSI implementation and can be made negligible by applying appropriate design techniques [Arslan95, Blair94, Chan95]. Therefore, the average power is effectively the switching power of the CMOS device. The switching power is a factor of the supply voltage, capacitance and switching activity, expressed in (2.2) [Arslan95]:

$$P_{average} = V_{DD}^2 \times C \times f \times k \quad (2.2)$$

where  $V_{DD}$  is the supply voltage,  $C$  the capacitance,  $f$  the frequency and  $k$  the switching activity which is defined as the average number of times the design makes a power consuming (0 to 1) transition.  $C$  and  $k$  are usually combined to form the product  $C^*$ , known as the switched-capacitance or effective-capacitance of a design. Reduction in average power therefore concentrates on the minimisation of supply voltage and  $C^*$ , the activity-capacitance product.

Equation (2.2) identifies that power has a quadratic dependency on the supply voltage therefore reducing the supply voltage is targeted as a key method of reducing power. Unfortunately, the reduction of supply voltage by itself has an associated penalty as it leads to a reduction in speed [Chan95, Liu93, Chan92]. This is illustrated in Figure 2.4.

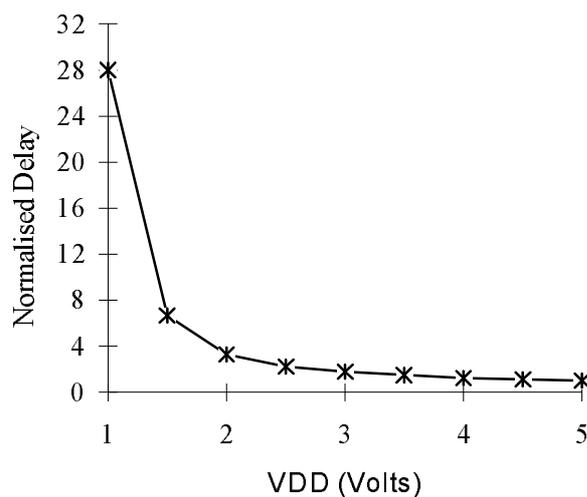


Figure 2.4 Relationship between  $V_{DD}$  and delay [Chan95a]. 2 $\mu$ m CMOS.

Figure 2.4 shows the effect of reducing the supply voltage of a device that has a nominal  $V_{DD}$  of 5V. As the voltage is decreased the delays in the device increase. For example, reducing the voltage to 2.9V will double the delay time of the device and hence halve the computational throughput. For many designs this reduction in speed is unacceptable. To compensate for the increase in delay, voltage-reduction strategies typically incorporate speed-up mechanisms to enable voltage to be reduced while preserving the original throughput. For a reduction to 2.9V the initial design requires a two-fold speed increase. Subsequent reduction of its voltage to 2.9V (from an initial 5V) will produce a low-voltage low-power device with the same throughput as the original.

Switched-capacitance reduction, while not apparently offering the large benefits of voltage reduction, is still an effective and practical technique for power reduction. Reduction of  $C^*$  is achieved through reduction of capacitance, reduction of switching activity or reduction of the activity-capacitance product. The capacitance is related to the physical implementation of the device; therefore, capacitance reductions are achieved through area reductions. Switching activity is both application and implementation-style dependent so it can be reduced through manipulation of the device and the algorithm. The reduction of the activity-capacitance product is typically achieved through assigning high-activity signals to low-capacitance connections.

### 2.3 Power Reduction

Power consumption can be influenced at all levels of the design process – technology, circuit, logic, architectural and the high-level behavioural description. This section presents an overview of techniques that can be used to reduce power

consumption during the design process. The work in this thesis targets the reduction in power of DSP devices (as discussed in Chapter 1). In application-specific data-intensive DSP applications the majority of power consumption is due to the datapath that processes the data [Rabaey95, Mehra94]. Although techniques for reduction of control and memory power consumption are available the discussion in this chapter concentrates on those techniques which affect the datapath power consumption.

At the technology level, reduction of supply voltage has been considered an effective means of power reduction [Blair94, Horo94]. As described in section 2.2, reduction of supply voltage incurs a speed penalty. However, if the threshold voltage and device dimensions are reduced by the same factor as the supply voltage the field strength in the device remains constant [Chan95a, Blair94, Horo94, Chan92]. Therefore, the reduction in supply voltage does not incur an associated speed decrease. This is known as ‘Technology Scaling’ [Horo94] or ‘constant electric field scaling’ [Blair94]. Unfortunately the side effect of threshold voltage reduction is an increase in the sub-threshold leakage currents [Blair94]. This can be significant enough to account for a large portion of the total power consumption. In addition, low threshold voltages reduce the noise margin in the device and the reduction of feature sizes requires modifications to the fabrication process, which requires a large capital investment. This voltage scaling approach has been typically used when feature sizes were decreased in search of higher speed and greater integration levels [Blair94] i.e. voltage reduction was a by-product of increased integration.

The choice of logic family can have an impact on power consumption. A promising alternative to CMOS is the use of Complementary Pass-gate Logic (CPL)

[Blair94, Chan92, Yeap96]. A CPL design only uses n-type transistors (CMOS uses both n-type and p-type semiconductor material) so a simple CPL circuit has half the transistors, and hence lower capacitance, of an equivalent function CMOS circuit. However, CPL requires a modification to the fabrication process and increased development in CAD tools that support this logic style. The use of CPL requires a large investment as it involves a fundamental change in the technology and design of VLSI devices.

Another technology level consideration is the use of adiabatic switching, where the load capacitor  $C$  is resonated with an inductor  $L$  which recovers some of the charge used to switch the capacitor [Lo98]. However, the delay of the LC circuit increases the delay of the device [Horo94].

At the circuit-level the main aim is to reduce the activity-capacitance product. Transistor sizing can have a significant effect on power as smaller transistors consume less power [Dev95, Horo94, Chan92, Yeap96]. However, smaller transistors are also slower. Therefore transistor sizes are selected to meet the specified speed requirements, effectively trading excess speed for power. In addition, cost functions which map high activity nodes onto small transistors help reduce  $C^*$  [Chan95a]. Similar techniques can be applied during the routing and placement stage, mapping high-activity nets onto shorter (lower capacitance) wires. This has produced power reductions of up to 18% [Singh95]. During the placement steps, consideration of locality, keeping blocks with a high degree of communication close together, also helps to reduce  $C^*$  achieving a 10% reduction in power [Singh95, Blair94]. During the floorplanning stage another objective is to give a more even spread to the power, to reduce ‘hot-spots’ in the chip and hence improve reliability.

The logic level has also been targeted as a design step in which power saving modifications can be made. To minimise  $C^*$ , logic blocks can be designed to minimise switching activity. For example, a ripple adder makes a lot of internal signal changes before settling on its final result. Replacing it with a larger carry-save adder may reduce  $C^*$  at the expense of an increase in area [Chan92, Chan95a], while automated design techniques can actually reduce  $C^*$  and area of adders [Kim98]. The strong dependence of power on switching activity means that glitch activity (spurious transitions), though not contributing to the overall computation, can account for 20-40% of the total power of a complete VLSI system [Singh95, Chan92]. One technique to reduce glitch activity is to place or move (retime) flip-flops within the circuit. Activity at the flip-flop's inputs is not propagated until the clock signal is enabled; the flip-flop effectively acts as a glitch filter [Mont92, Dev95]. The technique has been reported to achieve power reductions of approximately 8% [Singh95].

Careful designing of logic functions can also reduce overall switching activity as illustrated in Figure 2.5.

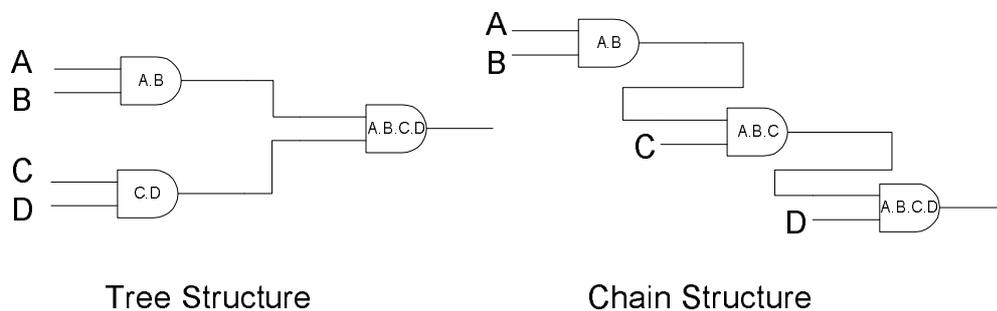


Figure 2.5 Example of Two Different Logic Designs with the Same Function

Both designs in Figure 2.5 perform the same function using the same number of gates. Assuming that all input signals are valid at approximately the same time then signal analysis determines that the chain structure will have the lowest switching activity if glitch activity is ignored [Chan95a]. However, the chain structure contains unbalanced paths (paths of different lengths). For example, input D is valid on the final NAND gate before the previous two NAND gates have finished processing signals A, B and C. Therefore the final NAND gate may change its output (a glitch) during the computation. The paths in the tree structure are balanced so it does not suffer from this problem. This example shows that consideration of glitch activity is important to determine the lowest activity and hence lowest switching power design. The paths can be balanced by inserting buffers in the shorter paths to equalise all path lengths or sizing down gates (to make them slower). These two methods reduce glitch activity but can also reduce the overall speed [Bella95b, Singh95, Najm94]. Multipliers have been designed which incorporate extra buffers for path balancing resulting in a decrease of  $C^*$  [Dev95].

Within a typical system certain logic blocks are not required all of the time. However, the inputs of these blocks are typically left connected to the data and clock lines so the logic block still processes the input data, resulting in unnecessary power-consuming transitions. Clock gating is a technique for shutting down unused sections of the design to prevent these spurious transitions. Logic gates on the clock-signal are used to prevent it from clocking the unnecessary logic blocks when they are not needed [Dev95, Balir94, Liu93]. An extension to clock gating is the use of precomputation logic to disable redundant sub-circuits. The precomputation logic is used to compute the output value of a given circuit one clock cycle before the

circuit is needed. If the precomputation analysis meets certain criteria then sections of the circuit can be disabled, as they are not required to process the input data [Dev95]. Techniques have been developed to automatically synthesise precomputation logic to maximise the power reduction [Kim96, Singh95]. The technique can produce savings of up to 62% in switching activity with a small increase in area and delay [Singh95].

The techniques previously described in this section are examples that target power reduction through voltage and capacitance reduction from the technology to the logic level. Other examples are; the choice of 2's complement, 1's complement [Dev95] or Gray code [Su94] number representation to reduce switching; the use of asynchronous designs to remove high capacitance clock-lines [Blair94]; increasing the correlation of consecutive signals to a logic block to reduce internal switching activity [Dev95, Singh95] and the use of parallel processing and pipelining to increase speed and hence reduce voltage [Blair94, Horo94].

Although the techniques described in this section have proven effective in reducing power consumption considerable advancements are still required if the targets of power reduction for the next generation of low-power devices are to be met [Fried94].

To achieve these targets high-level power reduction is expected to play an important role. High-level power reduction, where power is considered as early as possible in the design process, is expected to produce the greatest power reduction [Blair94, Horo94, Singh95, Chan95a]. Optimisation at lower-levels is constrained within the specifications of the high-level design. Power reduction at high-levels does not operate within such constraints and therefore provides greater freedom to explore the low-power design space [Rabaey96].

In addition, the advantages of high-level synthesis (as described in section 2.1) of flexibility and increased design speed also apply to high-level low-power synthesis. The earlier power consumption is considered in the design process the sooner designs can be verified and, if necessary, altered to conform to the power consumption specifications.

This work presents a high-level power reduction tool in order to exploit the advantages of high-level power reduction as described in this section. The next section presents a review of high-level power reduction techniques and applications including the technique used in this work.

### **2.4 High-Level Power Reduction**

High-level power reduction techniques can be conveniently grouped by the level of abstraction at which they operate. Architectural-level techniques optimise the power consumption during the synthesis of the behavioural description into a Register Transfer Level (RTL) design. RTL is a term used to describe the architectural level of abstraction where the design has been decomposed to a functional logic-block schematic, containing adders, registers, buffers, etc. Behavioural-level techniques optimise the high-level behavioural description or algorithm.

#### ***2.4.1 Architectural Level Power-Reduction***

During architectural synthesis each operation in the behavioural description has to be assigned an execution step (scheduling) and a functional hardware unit (allocation and binding) to produce the RTL level design. This is known as the architectural level of the synthesis process.

The scheduling and assignment of operations to particular functional units can be optimised to reduce the switching activity of the RTL design. This optimisation strategy uses the concept of correlation between two data samples. The correlation level is a measure of the number of different bits in the two sets of data; a higher level corresponds to fewer bit differences. If highly correlated samples can be successively assigned to the same functional unit it reduces the switching activity at the inputs of that unit, and hence the switching activity on its internal nodes. Figure 2.6 illustrates this process with an example in which signals  $e_1$  and  $e_2$  have a high-level of correlation. A non-power-conscious scheduling and allocation step may produce the schedule shown on the left, where the multiplication operations on  $e_1$  and  $e_2$  are assigned to different hardware units (M1 and M2 respectively, denoted by ovals in the figure). The schedule on the right is generated with a power-conscious methodology to exploit correlation between signals. The highly correlated signals are now successively applied to the same multiplier, therefore the activity at the inputs of this multiplier between time steps is relatively low, hence reducing its internal switching activity. The low-power schedule has reduced power consumption while using the same number of resources and the same schedule length.

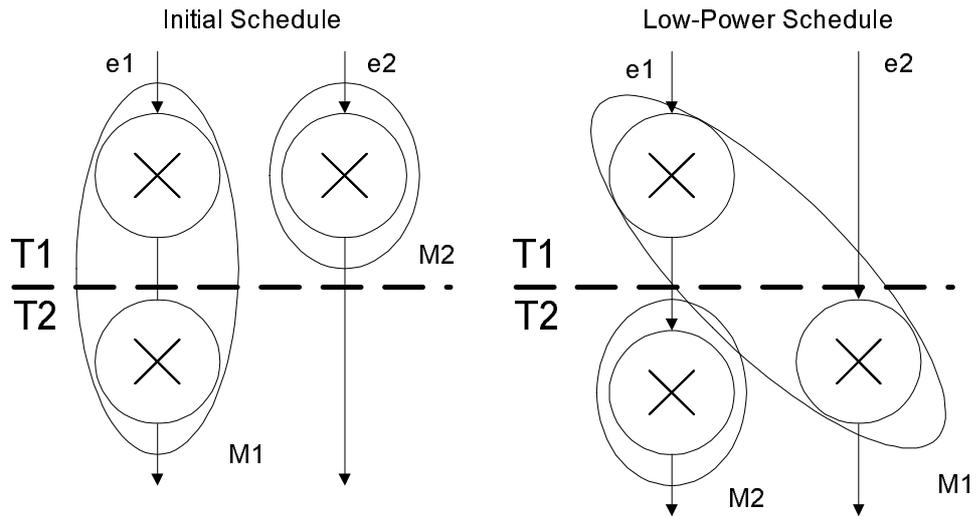


Figure 2.6 Scheduling and Allocation for Low-Power

However, such a scheduling strategy is a non-trivial task; therefore, specific algorithms have been developed to attempt to find the optimum solutions [Rag95, Rag95a, Rag94]. The algorithms incorporate linear programming techniques to solve the NP-hard problem of concurrent scheduling and allocation while considering competing objectives such as area and power. This illustrates the requirement for heuristic algorithms to be developed to tackle the complex problems of high-level low-power design.

In addition to reducing the switching activity of the functional units, similar techniques for increasing correlation can be used to reduce the switching activity of registers in the RTL design [Chang95, Kumar95]. High-level transformations such as loop-folding have been used to increase the amount of highly correlated sets of data in a single algorithm iteration, allowing for reduction of switching activity when used with a scheduling strategy which exploits such correlation [Kim97].

Spatial locality in an algorithm, where operations have a high-degree of communication, can also be exploited during the architectural synthesis process [Mehra96a]. The architecture is partitioned so that such operations are assigned to

the same hardware resource, thus reducing the overall activity-capacitance product as high activity data transfers are not performed on long (and hence high capacitance) data lines [Mehra96, Landman96]. Temporal locality (operations close in time) can also be exploited to reduce the number of registers files needed [Rabaey95].

During the high-level synthesis process each operation within the data-path may be implemented with a choice of functional units; e.g. multiplication with array or booth multipliers. Each unit has different area and speed characteristics that can be exploited for low-power operation. Power-Profiler [Martin95, Martin96] exploits the use of a library of functionally equivalent units to assign slower but less power-hungry functional units to non-critical-time sections of the algorithm; an example of this process is shown in Figure 2.7.

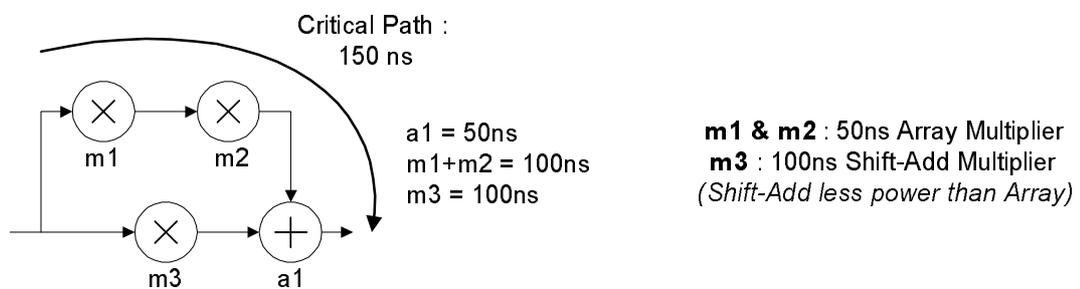


Figure 2.7 Use of Slower Low Power Functional Units on Non-Time-Critical Operations

In this example the period of the design is set to 150 nanoseconds. The adder requires 50ns to process the data therefore the m1 and m2 multipliers must have a combined delay of 100ns or less. However, m3 can have a delay of 100ns without violating the specified period constraint. Therefore m3 can be implemented

with a slower but less power-hungry multiplier, reducing overall power without affecting speed. The Power-Profiler tool optimally assigns operations to hardware units to reduce power while meeting area and speed constraints. A related optimisation strategy is the use of primitive operators, such as shift-add operations in the place of multiplications, to reduce the power consumption of the multiplications within a DSP algorithm. This strategy can have a significant effect as the multiply operations can consume a large part of the overall power due to their large capacitance and switching activity.

One of the first RTL power-optimisation techniques to gain commercial recognition is the use of clock-gating strategies. This technique uses extra logic to disable the clock signal to functional units that are not currently required for processing. If the clock-signal is fed to such units it results in unnecessary switching activity on their internal nodes. Therefore, clock gating can result in considerable reduction in switching activity [Mehra96]. Synopsys' Design Power tool now supports automatic synthesis of RTL designs with clock gating, the first commercial tool to support power-optimisation at this level of the design process [Synop98]. Clock gating has also been used in the new generation of Mobile Pentium Processors to reduce switching activity [Intel98b].

The discussed architectural level techniques are examples of power optimisation strategies that can be considered when synthesising a high-level algorithm into an RTL netlist consisting of logic blocks such as adders, registers, etc., control circuitry and data/control busses. Once the RTL netlist is created it is used to drive logic level tools in the next stage of the synthesis process. Logic level optimisation strategies, discussed in section 2.3, can then be used to further investigate a low-power implementation.

The architectural level techniques described in this section have been used to achieve significant power reductions in practical applications. However, as discussed previously, power consumption is best addressed as early as possible in the design stage to ensure that it is reduced at all levels of the design process. The next section describes the reduction of power at the behavioural level, tackling the power consumption inherent in the algorithm itself.

### ***2.4.2 Behavioural Level Optimisation***

One of the initial tasks in the design of a DSP system is the algorithm selection phase [Parhi92]. Typically, a number of algorithms exist for a desired application. For example, comparisons of 8 different Discrete Cosine Transform (DCT) algorithms in [Potko95] illustrated the different power characteristics of a suite of functionally equivalent algorithms. The work in [Potko95] also presented the development of a CAD tool for the automatic selection of an algorithm for a specific task based on its estimated power consumption. The presented results illustrate the benefits of correct algorithm selection at this phase of the design process.

After selection of the optimum algorithm, further power reductions can be obtained through optimisation of the algorithm itself. One of the most significant low-power design techniques presented in recent years is the use of behavioural transformations for high-level power optimisation [Chan95, Chan92a, Good94, Bright98b]. The technique is based on minimising the main factors contributing to power consumption of a CMOS device, the supply voltage and switched capacitance.

In systems such as HYPER-LP [Chan95, Chan92a] and that presented in [Sriv96], the use of behavioural transformations for power optimisation at the high-level has produced power reductions of an order of magnitude and greater. These tools are academic research projects; power reduction at the behavioural level is currently unsupported by commercial CAD tools. ASC Inc. are developing one of the first commercial CAD tools to incorporate behavioural transformations and scheduling for switching activity reduction. The tool, Power-Buster-D [ASC], is currently in the beta-development phase.

High-Level Transformations (HLTs) are used to modify the speed and capacitance characteristics of a behavioural description. One major degree of freedom available in optimising DSP systems is that once the required sampling rate has been achieved there is no benefit in increasing the speed of the algorithm. Therefore, the HLT-based approach increases the speed of an algorithm in order to trade excess speed for supply voltage reductions. Figure 2.8 presents an overview of the HLT-based technique for voltage reduction.

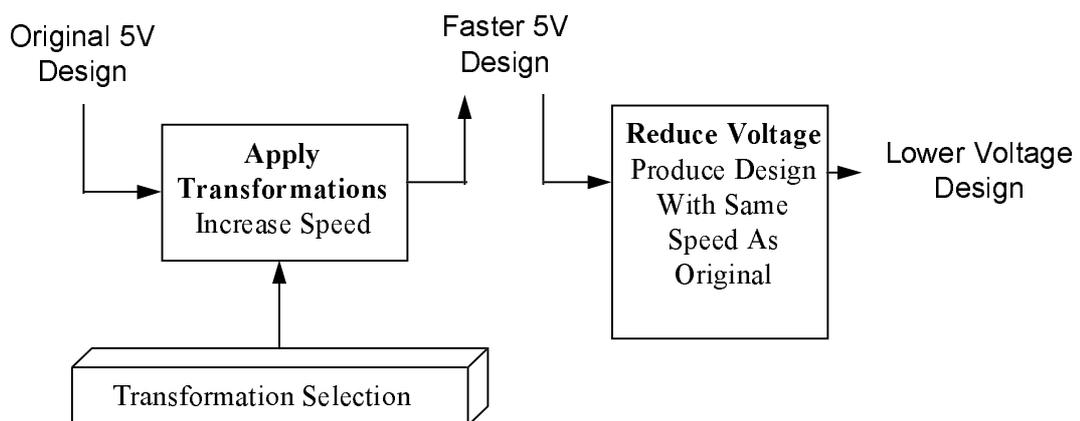


Figure 2.8 Use of Transformations to Reduce Supply Voltage

The transformational based approach can also be used to reduce the switching capacitance component of power by restructuring the high-level algorithm so it requires less capacitance and less switching activity.

The advantage of transformational-based power exploration is that it enables the algorithm to be modified without changing its function, thus allowing exploration of the design space while preserving the intended function of the design. Power reduction using high-level transformations is the core of the power reduction technique used in the tool presented in this thesis.

### **2.5 Power Reduction Using Transformations**

This section explains the methodology of using HLTs to reduce power. As previously described (in section 2.2) voltage reduction provides a significant power reduction at the cost of a decrease in system throughput. For example, reducing the supply voltage to approximately 3V doubles the delay of a device, halving its computation speed. This reduction in speed can be compensated for with the application of the HLTs. The HLTs are used to increase the speed of a design; reducing the supply voltage until its speed returns to that of the original then slows the design down. This results in a device with a lower operating voltage (and hence lower power) but the same computational throughput as the original design.

The transformation approach assumes that the initial behavioural description is designed to meet its required processing speed exactly. Therefore, any reduction in speed is not tolerable; so a reduction in voltage requires an increase in the speed of the device.

Section 2.2 also identifies switched capacitance as a component of power dissipation. HLTs can reduce capacitance through reducing the amount of hardware

resources required or reducing the switching activity. The distributivity transformation [Parhi95] is an example of a HLT that reduces the number of operations and hence the switched capacitance of a design. Careful ordering and cascading of operations in a design [Singh95] has also been shown to have an effect on the switching activity.

The disadvantage of HLTs is that they compound the already complex nature of the low-power synthesis problem. The use of only a single HLT known as retiming [Potko91] for design optimisation has been shown to be an NP-complete problem [Chan95, Chan92a, Lies83]. The complexity of the problem implies that it is very unlikely that an algorithm, guaranteed to find the solution in polynomial time, could be developed to synthesise the optimum solution [Chan95]. To solve the complex problem many low-power synthesis tools use a combination of heuristic and probabilistic techniques to apply the transformations, such as the use of simulated annealing in conjunction with VLSI design rules in [Chan95].

HLTs modify the high-level design descriptions represented as DFGs (described in section 2.1.1). The HLTs operate on the functional blocks of the DFG to alter its VLSI implementation characteristics (power, speed, area, etc.). HLTs, and their use for VLSI design and optimisation, are well documented in the VLSI design literature [Luck93, Parhi89, Huang94, Parhi95, Walker89, Koel, Wang95] as they have traditionally been used to optimise designs for speed and area. They have been successfully incorporated into automated design systems to improve the VLSI synthesis process [Huang96, Laksh98, Laksh98a]. There is, therefore, a range of transformations available, each with specific optimisation characteristics. The HLTs used within the system developed in this thesis are chosen for their qualities of

increasing design speed to allow for a reduction in supply voltage. The transformations are described in the following sections.

### 2.5.1 Retiming

Retiming [Parhi95, Potko91, Potko94, Lies83, Lock93] is the process of moving the delay elements from the input of a functional element to its outputs. This can reduce the critical path length which bounds the speed of a design. A shorter critical path produces a higher speed device. In addition, retiming can reduce the resource utilisation (and hence capacitance) as it can reduce the number of operations which are required to be performed at any one time. Figure 2.9 illustrates the application of retiming on a simple DFG.

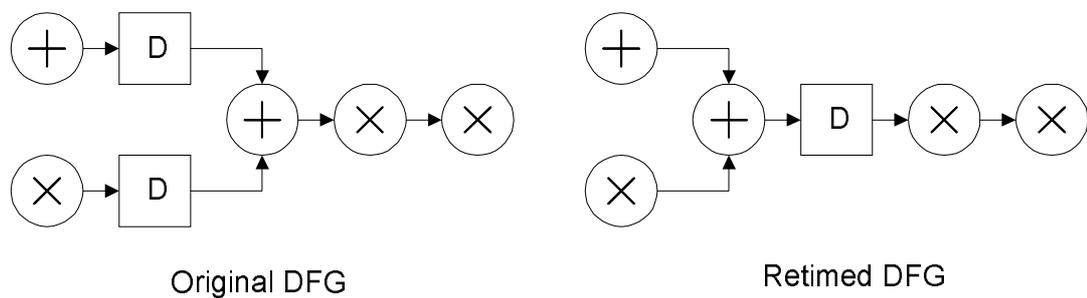


Figure 2.9 Example of Retiming Transformation

In the example of Figure 2.9 the two delays on the input of the adder have been retimed through the adder to become a single delay on its output. The retiming step has reduced the critical path from 3 to 2 nodes; the retimed DFG is 50% faster than the original. The figure illustrates that a retiming operation requires delays on all of the input nets of a selected node, otherwise the retime operation cannot be performed. This ensures that the functionality of the DFG is preserved.

## 2.5.2 Pipelining

Pipelining [Parhi89, Potko92a] is another transformation that can be used to reduce the critical path length, and hence increase speed. It operates by inserting delay elements into the DFG at specific points, known as *cutset points*. These *cutset points* are never found within the feedback loops of a DFG so the application of pipelining can be limited on designs with large feedback paths. The feedback or recursive loops of a DFG can effectively place a limit on the reduction of the critical path with transformations such as retiming and pipelining [Mess98, Fett93, Chun94].

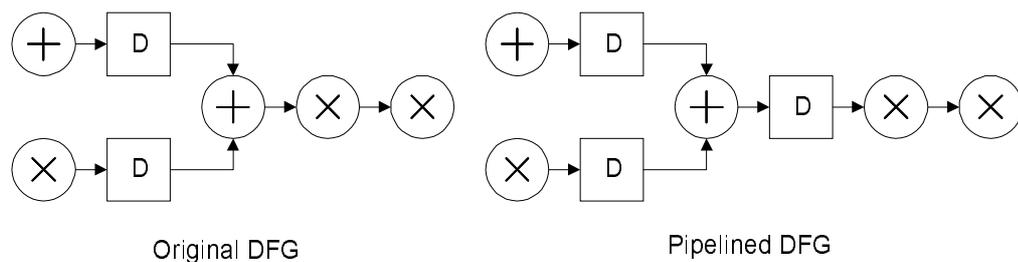


Figure 2.10 Example of Pipelining Transformation

In Figure 2.10 a pipeline delay has been inserted to reduce the critical path from 3 to 2 elements, a 50% speed increase. The effect of pipelining is to split the DFG into separate sections, pipeline stages, each of which can be processed concurrently. Therefore, insertion of a pipeline stage increases the amount of parallel processing that can be performed and hence increases the speed of the DFG (as all of the operations are performed in a shorter time length). However, another effect of pipelining is that it increases the latency of the DFG. Latency is defined as the time taken for a change at the inputs to have an effect on the outputs. Increased latency results in the DFG taking longer to produce its first output signal. An N-

stage pipelined DFG requires  $N$  processing iterations before the data reaches the output. Once the DFG has been iterated  $N$ -times the data is output on each subsequent iteration.

### 2.5.3 Automatic Pipelining

Automatic Pipelining is a special combination of retiming and pipelining [Luck93]. Delay elements are inserted on the primary inputs of the DFG and then retimed through the DFG.

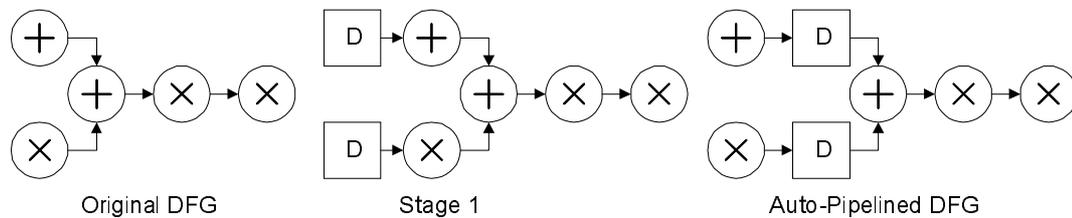


Figure 2.11 Example of Automatic Pipeline Transformation

Figure 2.11 illustrates that automatic pipelining is a two-stage transformation that combines properties of pipelining (insertion of delays on inputs in stage 1) and retiming (the second stage, retiming the delays into the DFG).

### 2.5.4 Loop Unfolding

Loop Unfolding [Parhi91] is a complex transformation that 'unfolds' the DFG to create a parallel implementation that processes  $N$  samples in parallel, where  $N$  is the 'unfolding factor'. Speedup is achieved in terms of samples processed as the DFG processes  $N$  samples per iteration; therefore, its effective sampling frequency is  $N$  times the initial design that processes one sample. The unfolded

design may also offer more efficient use of hardware resources as more operations may mean hardware resources left idle for shorter time periods.

The unfolding transformation is one of the most complex to apply as it affects every element within the DFG, producing a totally new DFG. The new DFG is in effect a parallel version of the original DFG, containing approximately  $N$  times the number of elements of the original, where  $N$  is the unfolding factor. The unfolding factor is a selectable parameter that determines how much parallelism is introduced into the DFG; for example, an unfolding factor of 3 creates a DFG that is effectively 3 of the original DFGs processing in parallel.

The implementation of the loop unfolding algorithm is based on that presented in [Parhi91]. The algorithm functions by initially creating a new DFG, (labelled the ‘child’ DFG) with  $N$  copies of each computational node in the original DFG (the ‘parent’ DFG). For example, a parent node  $U$  would create  $U_1, U_2 \dots U_N$  child nodes in the child DFG. The suffix attached to each child node is known as its ‘**unfolding integer**’.

There are a series of rules for the unfolding algorithm on how to connect the nodes (assign the precedence relations) in the child DFG, thus creating a functionally correct unfolded DFG. These rules are applied as a 3-step process. Figure 2.12 presents the pseudo-code for the three steps of the unfolding transformation, incorporating the rules to preserve the input-output functionality of the DFG.

Within the context of the unfolding transformation, a delay node is not considered as a computational node. For example, a net connecting an adder to a delay, followed by a net connecting the delay to another adder, is considered as a single net connecting the 2 adders, with a delay node on the net.

**Step 1:** For each node  $U$  in the original DFG, create  $N$  corresponding nodes in the new DFG. Label the nodes  $U_1, U_2, \dots, U_N$ .

**Step 2:** For each net from node  $U \rightarrow V$  in the original DFG, containing no delay nodes, connect a net from nodes  $U_k \rightarrow V_k$ , for  $K= 1$  to  $N$ .

**Step 3:** For each net  $U \rightarrow V$  in the original DFG containing delay nodes, perform Step 3A or Step 3B.

**Step 3A:** IF number of delay nodes ( $num\_delay\_nodes$ ) between computational nodes  $U \rightarrow V$  is LESS THAN  $N$ :  
 FOR  $q = (num\_delay\_nodes+1)$  TO  $N$ :  
 Connect nets from  $U_{(q-num\_delay\_nodes)} \rightarrow V_q$  with no delay nodes on the net.  
 FOR  $q = 1$  TO  $num\_delay\_nodes$ :  
 Connect nets from  $U_{(N-num\_delay\_nodes+q)} \rightarrow V_q$  with a single delay node on each net.

**Step 3B:** IF number of delay nodes between computational nodes  $U \rightarrow V$  is GREATER THAN OR EQUAL TO  $N$ :  
 FOR  $q = 1$  to  $N$ :  
 Connect nets from  $U_{(\lceil (num\_delay\_nodes-q+1)/N \rceil \times (N-num\_delay\_nodes+q))} \rightarrow V_q$ , with  $\lceil (num\_delay\_nodes-q+1)/N \rceil$  number of delay nodes on each net.

*Note:  $\lceil X \rceil$  denotes a 'ceiling function' i.e. the smallest integer greater than or equal to  $X$ .*

Figure 2.12 3 Steps of the Unfolding Process

Figure 2.13 illustrates this process for a small Infinite Impulse Response (IIR) filter. For this example an unfolding factor of 2 is selected ( $N=2$ ). The first step creates  $N$  copies of each node in the original DFG. Step 2 connects those nets that did not contain delay nodes in the original DFG, following the rules laid out in Figure 2.12. For the last step (step 3) the nets that did contain delays are connected. In this case the number of delays is less than the unfolding factor, therefore step 3A is executed. The results of the calculations for connecting nets and inserting delays are also shown in the example. The first calculation is to connect nets without delays, resulting in a connection from net  $A_1$  to  $B_2$ . The second calculation selects nets that will have a single delay, in this case  $A_2$  to  $B_1$ . These calculations result in the complete, 2-unfolded DFG presented in the figure.

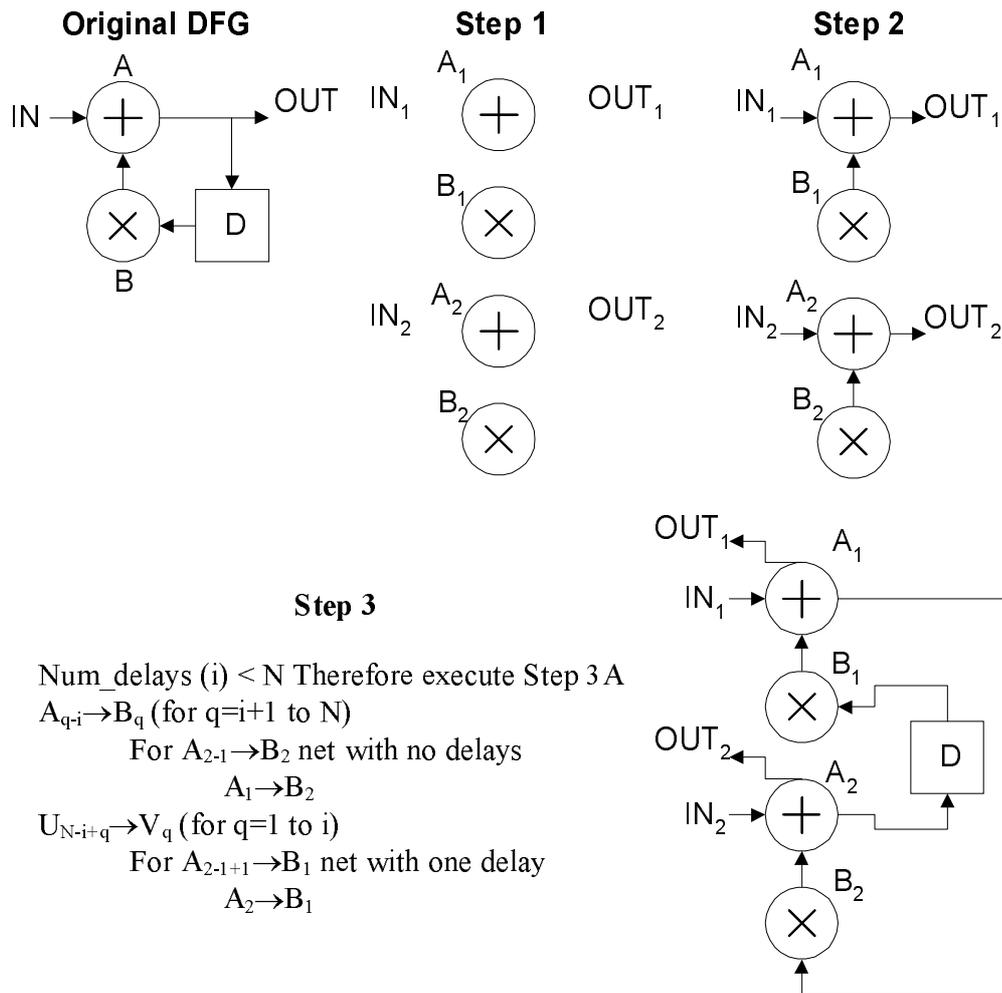


Figure 2.13 Example of the 3 Steps of the Unfolding Transformations

## 2.5 Summary

High-level synthesis offers significant advantages to the VLSI designer in today's competitive VLSI market where designs are increasing in complexity while requiring shorter times-to-market. The addition of new device criteria, such as power consumption, is further increasing the burden on the VLSI designer. This has led to the development of CAD tools to enable the designer to handle complex designs while exploring the solution space to meet competing criteria.

To support these CAD tools power optimisation techniques have been developed which tackle power at all levels of the design process. Power reduction at

the high-level is desirable, both to integrate with the advantages of high-level CAD tools and to obtain the greatest savings in power reduction. The high-level techniques presented in this chapter offer significant reductions in power consumption but require research into application methods to obtain optimum power reductions.

The transformation-based power reduction technique offers considerable benefits but increases the complexity of the VLSI task, both in its consideration of power as a design objective and the optimum application of the transformations. This complexity makes determination of the best low-power design an NP-hard task. Therefore, the work presented in this thesis incorporates the use of artificial intelligence techniques to explore the complex solution-space of transformation-based exploration for low-power designs.

## **Chapter 3 - Power Analysis**

During the design of low-power VLSI devices, the designer or CAD tool is faced with a large number of alternative designs that have different power consumption characteristics. Therefore, power analysis routines are necessary to enable comparisons between designs and parameter trade-offs. A VLSI power analysis tool is an essential component of any low-power design system.

A review of power analysis tools in [Pedram95] noted that prior to 1995 there was very little academic effort or interest in developing power analysis techniques and virtually no tools available from industry. However, the large increase in demand for power-conscious designs and low-power ICs has led to increased awareness of the need for advanced tools, to enable the increasing development of low-power design tools [Singh95, Coudert96, Tuck97]. The US Advanced Research Projects Agency has instigated a low-power electronics research program which includes the development of VLSI power-analysis tools [Lem94]. It is recognised that the development of power analysis tools is a prime requirement for the design and improvement of portable computing systems [Chwirka95]. Companies such as Motorola, Sente, Synopsys [Rabaey97], and Texas Instruments [Roy95] have identified the need for advanced tools that can operate at high-levels of abstraction.

This chapter gives an overview of the primary methods developed for power analysis and estimation. The analysis methods are conveniently grouped by the level of abstraction at which they operate: transistor, logic (or gate), architectural

(RTL) and behavioural or algorithmic. At each level some of the more popular power analysis tools are described to illustrate the use of power analysis in a practical synthesis environment. The power analysis tools that operate at these levels use various techniques to estimate the final power consumption of the VLSI device, therefore power analysis is also referred to as power estimation.

Traditional power analysis techniques have targeted the lower levels of abstraction, such as the transistor and more recently the logic level. The advantage of analysing lower levels of the design is that the analysis tool has a complete VLSI design to process, hence all of the required parameters (area, technology, transistor sizes, routing, etc.) are available. The power analysis is reduced to a simulation of the VLSI circuit, with tools such as SPICE [Rabaey]. The primary disadvantage of low-level tools is the time required to produce a result. PowerMill [Synop98c], a transistor level tool, is quoted as being able to simulate 1000 transistors per CPU second [Roy95]. An IC with 3 million transistors would require 11 months CPU time for a single run, clearly unacceptable for use as feedback in a low-power CAD tool.

Analysis tools that operate at a higher level of abstraction offer the benefit of enabling designs to be modified early in the design cycle, which is expected to yield the greatest benefits in terms of power reduction [Frenk97, Pedram95, Roy95]. In addition, high-level tools typically produce results more quickly. However, trying to estimate power dissipation at early stages in the design cycle is a non-trivial task [Singh95]. Designers need to trade off accuracy for the gain in speed. Therefore, high-level tools are typically used to compare the relative power consumption of alternative designs rather than provide absolute power values [Mehra96, Ketz94].

### 3.1 Power Dissipation in CMOS Devices

The main contributing factors of power dissipation were introduced in section 2.2. This section discusses the previously described factors in terms of power analysis of DSP devices. To summarise, the average energy per computation in a DSP device implemented in CMOS is given by [Arslan95]:

$$P_{average} = V_{DD}^2 \times C^* \quad (3.1)$$

where  $V_{DD}$  is the supply voltage, and  $C^*$  is the switched-capacitance or effective-capacitance of a design. Average power is used to calculate battery life and junction temperature [Frenk97] and is, therefore, typically used as the figure of merit [Yeap96, Bella95b]. Estimation of the parameters of equation (3.1) will yield an estimate of the average power consumption of the design.

Equation (3.1) illustrates that the average power is activity dependent (as  $C^*$  contains the switching activity estimation). This makes power estimation a complex process, as the switching activity is dependent upon the devices input signals and implementation style. Implementation style may be determined at lower levels of the design process (but not at higher levels), but input-signal data may not be available as it is strongly dependent upon the device application.

The optimisation tool presented in this thesis specifically targets the synthesis of low-power DSP algorithms; therefore the discussion of power analysis concentrates on DSP devices. The average power dissipation of DSP devices, implemented as ASICs, can be divided into algorithm and implementation dependent power. Algorithm inherent power dissipation comes from the execution

of the operations within the DSP (e.g. additions, multiplications). This power dissipation is relatively independent of the architectural implementation style used. The overhead power is a factor of the VLSI implementation such as the control logic, registers, buses, etc. required to implement the DSP algorithm. Therefore estimation of the power consumption of a DSP device requires analysis of both the algorithm and the intended VLSI platform.

### 3.2 Transistor Level Analysis

Transistor level analysis is one of the longest established areas of power analysis. The ‘de-facto’ power analysis tool is SPICE, a circuit-level simulator developed in the 1970’s at University of California at Berkeley [Rabaey]. SPICE simulates a circuit-level description of the design at the transistor level to produce current and voltage waveforms, which can be used to determine the power consumption of the design. The fact that SPICE operates on such a low-level description accounts for its high level of accuracy, it is effectively simulating the completed design, with accurate models of transistors and device level components.

One of the main disadvantages of SPICE is its speed; such low-level device simulations are very computationally intensive. Simulating relatively small devices (a few thousand components) can take more than 5 hours. Another disadvantage of SPICE is its dependence on the provided input patterns for simulation. Power consumption is highly dependent on switching activity, so the power estimations produced by SPICE are specific to the input patterns used to simulate the design; SPICE is a ‘strongly-pattern-dependent’ tool [Najm94].

While SPICE is a very slow tool its high level of accuracy makes it useful in later stages of the design process, to validate results produced by other power

analysis tools and to develop SPICE characterised modules for use at higher-levels. The commercial transistor-level power-analysis tool PowerMill [Synop98c] exploits this feature. PowerMill simulates the design using SPICE developed modules, effectively reducing the complexity of the simulation. For example, a digital multiplier with 1,500 devices requires 5.6 hours for simulation in SPICE whereas PowerMill requires 3.9 minutes, with only a  $\pm 5\%$  decrease in accuracy. This illustrates the speed and accuracy trade-off in the design of power-analysis tools, which is regarded as an acceptable sacrifice [Roy95, Singh95, Mehra96].

AMPS [Synop] is a power-optimisation tool which incorporates the analysis features of PowerMill within the optimisation process. AMPS automatically resizes the transistors to optimise for speed, area and power; PowerMill is used as feedback within the process to validate the effect of different optimisations.

While many techniques have been developed to attempt to increase the speed of transistor-level power-analysis [Liao96, Van93], the disadvantages of the tools have limited their use in high-level power optimisation strategies to validation and lower-level optimisation tools once the design has been synthesised through to the transistor level.

### 3.3 Logic Level Analysis

To overcome the disadvantages of transistor-level analysis several logic-level power estimation techniques have been proposed. The main aim of logic-level tools is to increase the speed of the process with an acceptable impact on accuracy. Logic- or gate-level power analysis is less accurate than transistor level analysis as the details of power dissipation of transistors are simplified in gate-level models.

As described in section 3.2, transistor-level tools suffer from a strong-pattern-dependence, one technique to reduce this dependence is to represent input and internal signals as probabilities; this is known as a probabilistic power-analysis technique. These signal probabilities, rather than the actual signals, are propagated through the logic circuit to produce signal probabilities for every node in the design. There are a number of ways of defining signal probabilities; two of the most common are [Najm94]:

- Signal Probability –  $P_s(X)$ , the probability that a signal at node X will be high
- Transition Probability –  $P_t(X)$ , the probability that a signal at node X will make a power consuming (0→1) transition.

By determining these probability factors for every node in the circuit the total power can be estimated, for example:

$$P_{average} = f \times V_{DD}^2 \times \sum_{i=1}^n C_i P_t(X_i) \quad (3.2)$$

where  $f$  is frequency,  $V_{DD}$  supply voltage,  $C_i$  the capacitance of node  $i$  and  $P_t(X_i)$  the transition probability of node  $i$ . Specific rules exist for the propagation of signal or transition probabilities depending on the type of logic gate the signals are passing through.

The use of probabilities rather than absolute signal values requires the consideration of signal correlations, which fall into two categories. Spatial

correlation refers to two signals in the circuit that are dependent upon each other e.g. both inputs to a particular AND gate may never be high. Temporal correlation refers to two signals with a dependency in time i.e. a 1 may never follow a previous 1 on a particular node. Temporal correlations are particularly important when analysing designs with feedback as the next set of signals is directly related to the current set [Najm95].

In addition to the consideration of correlation, logic-level analysis tools need to consider the effect of glitch transitions in the circuit which may not affect functionality but can consume up to 70% of the total power in a combinational logic block [Najm94] (20-40% of the total power in a VLSI system [Singh95]).

LTIMES [Cirit87] is an example of a power analysis tool that uses the propagation of signal probabilities together with a capacitance specification file to determine the total power consumption. It ignores all signal correlations, which enables it to produce results relatively quickly at the expense of accuracy. Consideration of correlation and glitches significantly increases the complexity of the analysis, so they are often left out or simplified during the power analysis [Najm94, Bella95b].

PowerGate [Synop98b] is an example of a tool that considers correlations and glitches. It can analyse an 8500 cell logic circuit 486 times faster than SPICE, with an associated 10% reduction in accuracy; it can analyse a 48000 cell logic circuit 60 times faster than PowerMill, with an associated 5% reduction in comparative accuracy. These qualitative examples show the benefit in speed of using logic-level analysis tools and the trade-off in accuracy made to achieve the speed increase.

An alternative to propagating signal probabilities, thus removing the problems of modelling correlations, is to simulate the logic circuit using actual input vectors. A Monte Carlo approach [Burch93] combines the advantages of simulation (no need to model correlation effects) with probability techniques (reduced pattern-dependence of analysis). Such an approach is known as a statistical analysis technique. The circuit is simulated with typical or randomly selected input vectors for a specified number of times. The activity and estimated current for each vector is used to determine the power consumption. The average power consumption is the average of the power estimations for each vector. Each additional simulation increases the confidence in the result; therefore the required accuracy can be specified up front while considering its effect on the speed of the analysis. The disadvantage of this technique is that it is still slower than probabilistic techniques, again illustrating the trade-off between accuracy and speed.

Design\_Power [Synop98d] allows the user to select between probabilistic and statistical analysis techniques, trading off speed for accuracy. Typically, probabilistic analysis is used initially to select design changes, its increased speed enabling design changes to be selected relatively quickly. Statistical methods are typically used later to validate sets of design changes, the reduction in speed compensated for by the increase in accuracy. Power Compiler [Synop98d] is a gate-level power optimisation tool that uses Design\_Power to perform power estimations and validate gate-level power optimisations more quickly than transistor-level tools would allow.

Logic-level analysis tools have become more popular in recent years, evidenced by the commercial and academic tools described here. Systems such as

Power\_Tool [Veritools] and XPower [Genashor] illustrate the increasing commercial support for logic-level analysis tools. Their main advantages over transistor-level tools are the reduction in analysis times at the expense of accuracy. The level of error is acceptable given the large speed increases, enabling the tools to be used within practical power optimisation systems.

### 3.4 Architectural Level Analysis

The architectural level refers to the RT level of abstraction, where the design is described in terms of modules such as adders, registers, multipliers, etc. RTL power analysis is the problem of estimating the power consumption of these modules and hence the total power consumption of the design. Many tools now use the RT-level as the point of entry for describing the design [Macii97]; therefore RTL analysis tools are becoming increasingly useful.

While section 3.3 outlined the use of logic-level power analysis tools within power optimisation systems, the tools still suffer from long run-times as the number of logic elements increases. This has reduced their application in high-level power optimisation systems.

Gate-level tools estimate the power consumed by logic (e.g. AND, OR) units whereas RTL tools estimate the power consumption of modules (e.g. adders, busses). RTL analysis typically deals with fewer discrete blocks as the large numbers of logic elements are grouped into the higher level modules. Therefore, the RTL power analysis should be faster than the logic-level, following the trend of higher levels of analysis producing results more quickly than lower levels.

The technique proposed in [Liu94] expresses the power consumption of typical components in an RTL system as a function of basic parameters such as

capacitance and size. The disadvantage of this technique is that assumptions made in building the power models (for extraction of the basic parameters) reduces the accuracy of the estimation.

The Power Factor Approximation (PFA) method [Powell90, Chau92, Powell91] is an RTL technique that develops power-characterised models for hardware units such as I/O, interconnection, execution units, etc. The technique targets the power analysis of DSP algorithms.

PFA is based on the assumption that it is the VLSI implementation technology and not the size of the RTL module that determines the average switching activity, load capacitance and current per logic element in a module. A PFA constant is derived to estimate the average power dissipated per gate, per clock-cycle for a specific type of module. Different modules have a different PFA constant depending on their function and implementation style (e.g. an array or booth multiplier). The average power of a module is the product of the PFA constant, the clock frequency and a measure of the complexity of the module. The complexity of the module can be related to factors such as the bit-width to enable their effect on power to be analysed. The average power of the whole design is the sum of the average power of all modules. The main advantage of the PFA technique is its speed. Once the PFA constant for each module type has been determined, it can be reused in any number of analyses.

The PFA constant for each module is determined by characterising each module for capacitance and switching activity. The disadvantage of the PFA technique is that the switching activity is derived using a Universal White Noise (UWN) model. However, typical DSP signals such as speech and music do not approximate UWN signals. The PFA technique ignores correlations in the signals

and as a result the accuracy suffers when the actual input signals do not have UWN properties. PFA’s primary advantage is its increased speed over logic-level analysis tools.

The Stochastic Power Analysis (SPA) technique [Landman93, Landman94, Landman96a] improves on the accuracy of the PFA technique by including the effect of typical signal activities on power consumption. The core of the SPA technique is the development of the Dual-Bit-Type (DBT) model for two’s-complement signal representation, illustrated in Figure 3.1.

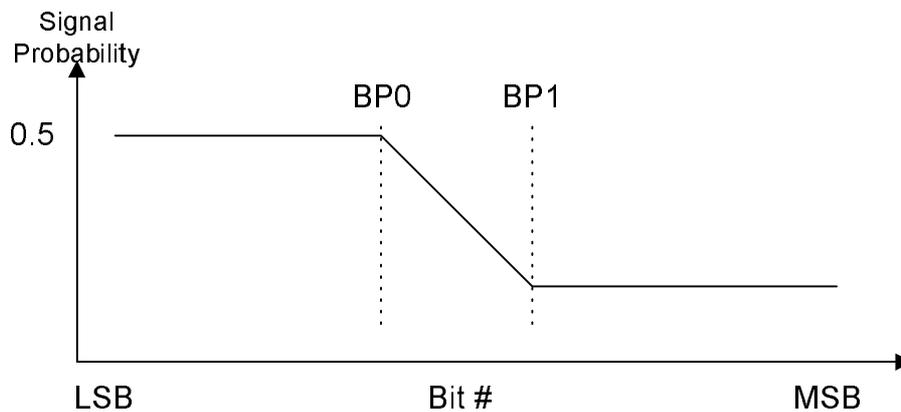


Figure 3.1 The Dual-Bit-Type Model

The DBT model was developed to model word-level statistics based on the bit-level statistics such as the signal probabilities described in section 3.3. The DBT figure illustrates the signal probability of each bit from the LSB to the MSB of a word. For the LSBs the probability is approx. 0.5, equivalent to the UWN model. For higher order bits the probability drops, indicating that the MSBs have a high-level of correlation. This explains the problems of the PFA technique as it assumes all bits of the signal follow a UWN model, the DBT model illustrates that this assumption is not correct.

The DBT model has been shown to be true for a range of typical signal types such as speech and music [Landman93]; the breakpoints and signal probability levels are derived for each signal type. Therefore, the power analysis is related to the typical application of the DSP algorithm without requiring detailed information of the actual input signals.

As with logic-level analysis tools the derived word-level statistics are propagated through the RTL design. Each type of module affects the word-level statistics in a pre-defined manner, enabling a complete set of word-level statistics to be obtained for all the nodes in the RTL design. The word-level statistics are combined with black-box capacitance models of RTL modules to produce a switching activity dependent estimate of power consumption.

Signal correlations such as temporal correlation (section 3.3) present a more difficult problem but heuristic techniques have been developed to model some correlations. Feedback in the RTL design is a primary problem; this is solved by approximating recursive sections with non-recursive sections with minimal effect on the accuracy of the result. As with the PFA technique, the SPA method is primarily targeted at DSP designs implemented as ASICs.

For a typical music signal application, compared with transistor level analysis tools the PFA technique has an error of 90% while the SPA technique has an error of 10% [Landman93]. This illustrates the importance of considering signal properties when analysing power. However, for differential image signals, which have similar properties to the UWN model, both techniques have an error of 5%. Therefore SPA is a better technique where the typical application of the DSP algorithm is known at the RTL level. Typically the SPA technique has an error of 10-15% when compared to transistor-level techniques [Landman96], with a large

increase in speed over transistor- and logic-level analysis. The inaccuracies are partially due to uncertainties about the final placement and routing of the modules, VLSI design aspects which are not finalised at the RTL level.

SPA has been incorporated within a low-power system design framework [Landman96] to illustrate its usefulness in evaluating high-level architectural design choices. In case studies, the power consumption of execution units was significantly overestimated; however, the overestimation was systematic i.e. the relative comparison between designs was still valid. This illustrates that high-level analysis tools are more useful for comparing designs than estimating absolute power consumption.

WattWatcher/Architect from Sente [Sente] is one of the few commercially available tools to tackle architectural level power analysis. The power is estimated directly from an RTL description of the design to take advantage of the speed benefits available at this level.

The previously described Design\_Power, a logic-level tool, can also be used for RTL analysis. The tool performs a fast synthesis on the RTL design to produce a logic-level layout from which the power can be estimated using logic-level tools. The disadvantage of this technique is the time overhead required to synthesise the RTL design and the large number of gates that could be produced from such a synthesis.

PowerPlay [Lidsk96] is a WWW based power analysis tool that was developed to address the problem of creating a library of RTL modules characterised for capacitance and power. The power analysis is a simple process that counts the number of units of each type and reports power consumption in relation to frequency and supply voltage. The main aim of the tool is not the

development of a novel power analysis method, but rather harnessing the power of the WWW to enable designers from all over the world to pool resources in creating a complete library of characterised RTL modules. These modules can be used as part of an RTL power analysis tool.

RTL power analysis tools are becoming increasingly popular for both commercial and academic design projects. However, it is recognised that power optimisation will yield the largest benefits if it is tackled from the highest levels of abstraction first. RTL tools require certain aspects of the design to be finalised, therefore greater attention has become focused on the development of behavioural level tools.

### 3.5 Behavioural Level Analysis

Behavioural level analysis attempts to estimate the power consumption of a design at the highest level of abstraction, the algorithmic description of the signal-processing task. The algorithm describes the task in terms of operations (e.g. additions, multiplications) and data transfers between those operations.

Power analysis at this level requires estimation of the power consumption of operations in the algorithm and the implementation overhead such as registers, interconnect, etc. The approach developed in [Chan95], [Mehra94] and [Rabaey95] identifies that power consumption comes from datapath, interconnect, control and memory units. It uses a combination of analytical and stochastic analysis techniques to produce a behavioural level analysis tool with an average error of 20% compared to RTL tools.

The accuracy of behavioural level power analysis tools is limited because there is very little VLSI implementation information available. For example,

operations in the algorithm are not yet mapped to specific units in the RTL design. Therefore the SPA technique cannot be used to propagate signal correlations around the design [Mehra96, Rabaey95, Mehra94].

The behavioural technique simplifies the estimation process by tying the DSP algorithm to a specific RTL implementation style, as illustrated in Figure 3.2.

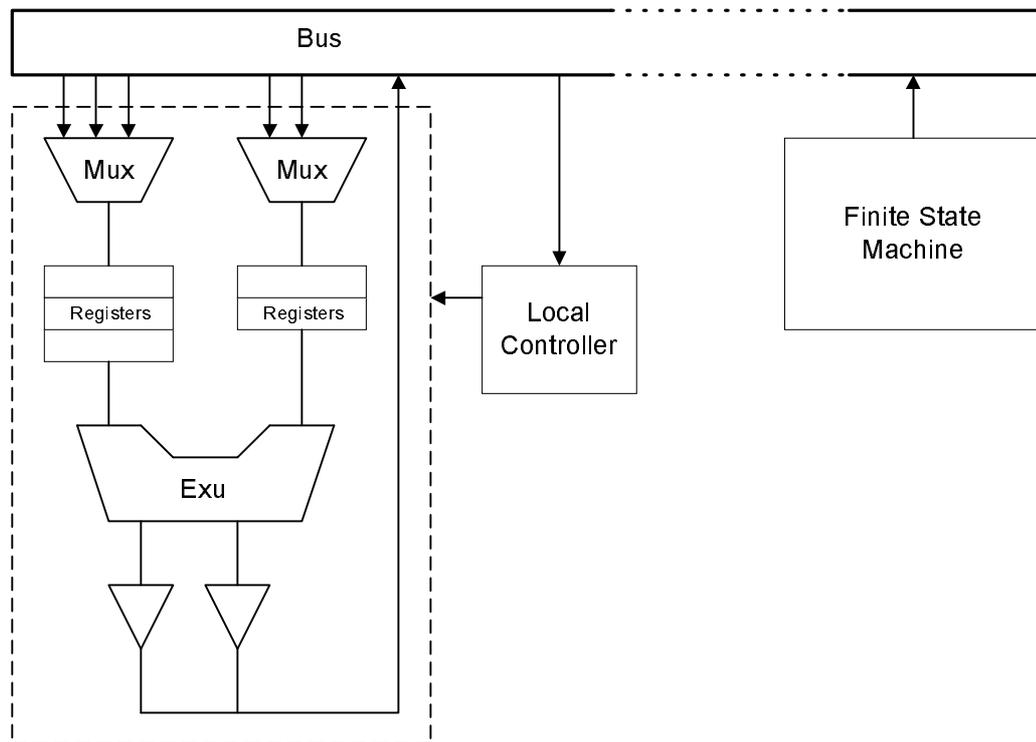


Figure 3.2 RTL Template for Behavioural Level Power Analysis

The figure depicts an RTL hardware unit consisting of registers, multiplexers and the main execution unit. The RTL design uses a Finite State Machine (FSM) as a global controller, with local controllers associated with each hardware unit. While this simplifies the estimation task it does limit the general applicability of the results.

The power analysis is split into estimating the consumption of datapath, interconnect, control and memory units separately. Some of the information is

estimated directly from the datapath while statistical models are developed to estimate the rest.

**Datapath** – The estimation of datapath power is related to the number of module accesses as opposed to the actual number of modules. The power consumption is relatively independent of the number of modules as 10 accesses to a single module dissipates approximately the same power as 1 access to 10 modules. The number of execution unit accesses is directly related to the number of operations of that type in the DFG. Pre-characterised execution unit modules are used to estimate the total switched capacitance and hence the total power consumption. Analysis of the algorithm yields upper and lower bounds on the required number of register accesses to execute the algorithm. The average of the two bounds is taken as the actual value; again, this is combined with characterised modules to determine the switched capacitance of the registers.

**Interconnect** – Though often ignored during high-level power analysis, interconnect can typically consume between 10% and 30% of the total power [Mehra94]. It is one of the most difficult parameters to estimate at the high-level as it is strongly linked to low-level layout parameters. For this reason a statistical model of interconnect capacitance is used which relates interconnect capacitance to the VLSI datapath area (containing the execution units, registers, busses, etc.), shown in Figure 3.3.

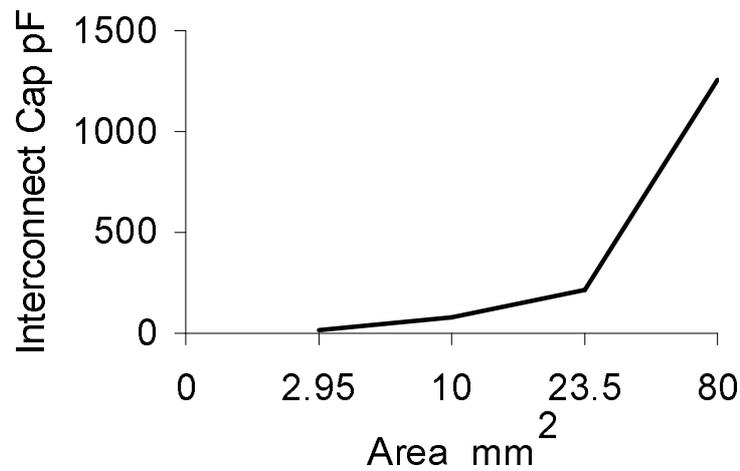


Figure 3.3 Interconnect Capacitance Model [Chan95]

The work presented in [Mehra94] and [Chan95] also presents techniques for estimation of control and memory units power consumption. However, it is noted that the datapath and interconnect component accounts for the majority of the power in the data-intensive algorithms targeted.

The clock distribution network or clock tree can account for a significant portion of the total power budget. The clock-tree is typically the longest net in the VLSI device and hence has a relatively large capacitance. The typically high frequencies and the large number of capacitive loads they drive increases the high power dissipation of clock lines. Clock power estimation is typically incorporated into the power estimation of register units, as these are the datapath units driven by the clock line [Chan95]. The register capacitance models are comprised of the registers switching capacitance and its contribution to the load capacitance of the clock line. Summation of the register capacitance will produce the total load on the clock line due to datapath units. The use of architectural and physical level design

strategies is used to reduce the intrinsic capacitance of the clock line, resulting in the load on the clock line being the most significant contributor to clock power dissipation. At the architectural level designs are developed with controllable and gated clocks; this enables reduction of clock activity and also reduces the length of any single segment of the clock line (and hence the activity-capacitance product) [Chan95, Pedram95, Ketz94]. At the physical level the use of a distributed buffer system, where buffer elements are placed throughout the clock tree, also reduces the length of any single clock elements [Rabaey96].

The results presented for behavioural level power analysis tools [Mehra94, Chan95] illustrate that accuracy is sacrificed both for increased speed and the ability to estimate power at such a high-level of abstraction. However, use of high-level analysis in tools such as Explore [Mehra94] and HYPER-LP [Chan95] illustrates the benefits of being able to compare designs for relative power consumption at such early stages in the design process. While behavioural level analysis tools are still in the early stages of development they are already providing invaluable feedback to high-level design tools.

As well as describing ‘state of the art’ [Mehra94] behavioural power analysis techniques, this section has illustrated the lack of behavioural level analysis tools. No commercial behavioural level power analysis tools are currently available [Coudert96] though prototypes are in development such as Power-Buster-D being developed by ASC and Princeton University [ASC].

## **Chapter 4 – Genetic Algorithms**

The power optimisation tool presented in this thesis uses a Genetic Algorithm (GA) as its core search and optimisation mechanism. This chapter introduces the main concepts of a genetic algorithm before describing their application in engineering design and optimisation problems. Finally, the last section of this chapter explains why it is advantageous to use a GA to solve the complex problems inherent in high-level power optimisation and synthesis.

### **4.1 Overview of Genetic Algorithms**

The GA is a search and optimisation technique inspired by the processes of natural selection in biological organisms. The lineage of GAs can be traced back to Darwin's "On the Origin of Species" [Darwin], in which it was recognised that species evolve according to their ability to survive and reproduce.

The evolutionary process can be viewed as an adaptive optimisation technique. The optimisation process is searching for a 'good' solution to the problem of surviving and reproducing in the changing natural world; each natural organism is considered to be an alternative solution to this problem. Those organisms better adapted to survive in the environment will have greater opportunity for reproduction. The process of reproduction results in the perpetuation of the genes of the parent organisms; therefore, the genes of the better organisms are more likely to be spread throughout the population as the evolutionary process continues. The solutions are taken from a range of billions of

possible solutions, the characteristics of which are dictated by the organisms' genetic sequences. The genes which create an organism that is better than its competitors will be reproduced and successively refined (during many generations of the reproduction process) to produce even better solutions.

The optimisation features of natural evolution have inspired evolutionary optimisation strategies since the 1960's [Schwef95]. In the 1970's John Holland invented the specific evolutionary optimisation technique which has become known as the GA; he proposed a rigorous definition and analysis of the technique in his seminal publication "Adaptation in Natural and Artificial Systems" [Holland92]. Holland's work developed relatively simple GAs that were shown to solve some extremely difficult problems. GAs have since been further analysed and refined in many basic texts [Davis91, Goldberg89, Mitch96, Beasley93, Teuk95a]. Since their initial inception, GAs have been applied to problems in fields as diverse as economics, biology, political science and engineering. In [Holland92] it is noted that GAs have become a recognised technique for the solution of practical 'real-world' problems. The increase and success in their application to practical engineering problems is evident in publications such as [Zalzala97] and the proceedings of the Genetic Algorithms in Engineering Systems conferences [Galesia95, Galesia97].

Within a GA, potential solutions to the problem are encoded as chromosomes; the chromosome consists of a set of genes, which represent the characteristics of that solution. The GA operates on a number of chromosomes, collectively termed a population. In order to assess each solution's potential for reproduction it is necessary to gauge the 'fitness' of each solution. This is a measure of the ability of that solution to satisfy the objective problem. Those solutions with

a higher fitness have a greater chance of selection for reproduction. The reproduction process creates new individuals with the combined characteristics of the ‘parent’ individuals. Hence, the properties of higher fitness individuals are propagated throughout the population.

Successive application of the reproduction process creates a whole new population of solutions, termed the ‘next generation’. This generation contains a higher proportion of the characteristics possessed by the fitter solutions of the previous generation. The mechanism of selection and reproduction facilitates the exploration of the solution space.

Successive generations of the GA explore a variety of solutions throughout the solution space. If the GA is successful it will converge to an optimal region of the solution space, producing an optimal solution to the objective problem.

The main advantages of the GA are its robustness and flexibility. It is able to outperform more traditional optimisation techniques due to its ability to escape local optima in the solution space. A typical problem may be characterised as a solution space with a large number of hills and valleys; however, one hill is higher than all the rest (the optimal solution). Traditional techniques may take the path of steepest ascent to find the peak; but the first peak found may not be the highest peak. The GA has the ability to evaluate many peaks at a time (due to the use of a population of solutions). In addition, the GA has the ability to suffer temporary decreases in solution quality (by moving downhill) in order to avoid becoming stuck on the first peak found. This ability is conferred upon the GA by basing the selection of individuals on probability rather than certain selection of the absolute fittest individual. Lower fitness individuals still have some chance of selection, enabling the GA to escape local optima.

The GA is a flexible optimisation technique, as it does not require any prior knowledge of the search space. Therefore, assumptions made about the problem do not limit the search process. The GA only requires a suitable coding structure, a method of modifying that structure during reproduction and a system for assessing the quality of individual solutions. If the goals of the process change only the fitness function is required to be modified instead of a complete modification of the optimisation algorithm.

### 4.2 Standard Genetic Algorithm Implementation

This section describes the implementation details of a basic GA. Figure 4.1 presents the pseudo-code of a basic GA.

#### **Algorithm Genetic\_Algorithm**

Initialise *Population\_of\_Solutions*

**While** stopping criteria not met

    Compute *fitness* of solutions

    Select solutions for reproduction

    Apply reproduction operators (genetic operators)

    Breed *next generation* of solutions

**End While**

Fittest solution found is *best solution*

**End Algorithm**

Figure 4.1 Pseudo-Code for a Basic GA

#### 4.2.1 Solution Representation

An initial requirement for implementation of a GA is to encode possible solutions to the problem into a chromosome structure suitable for manipulation by the GA.

Holland's original work [Holland92] suggested using binary-strings of 0's and 1's to represent the solutions. The individual elements within the chromosome, such as each 1 or 0, are known as genes. Subsequent research into GAs has seen the use of more complex representations such as alphabet-strings [Goldberg89] and decision trees [Schnecke95]. The most important point is that the representation encodes the important properties of the solution in a method which enables those properties to be fully explored by the GA.

The choice of chromosome representation is very important if the full power of a GA is to be exploited. An unsuitable choice of chromosome representation can place an unnecessary computational strain on the GA, requiring complex manipulation and decoding for quality evaluation. The choice of chromosome can also affect the performance of the GA in searching the solution space. It may even prevent the GA from determining an optimal solution as a poorly designed chromosome may not be able to represent all possible solutions or it may not allow certain operations.

### *4.2.2 Fitness Assessment*

For each application of a GA a means of determining the relative quality of each solution is required. Fitness assessment is the process of decoding the chromosome and calculating its relevant parameters. The quality or fitness is a direct measure of how well the parameters of the encoded solution satisfy the objective function. The objective function is goal or desired state of the optimisation process.

### 4.2.3 Reproduction Operators

The reproduction step of the GA selects individuals from the current generation to produce offspring that will comprise the next generation. The production of offspring involves the modification and combination of the genes of solutions in the current generation; this is the core of the search mechanism, as the next generation will consist of a different set of chromosomes to the previous generation. Modification of the chromosomes is achieved through the application of genetic operators, the two most common of which are known as mutation and crossover.

Mutation operates on a single chromosome to modify its characteristics through random manipulation of its genes. An example of the mutation process is illustrated in Figure 4.2 for a binary-string chromosome.



Figure 4.2 Example of a Mutation Operation

This example illustrates the random change of the value of a gene within the chromosome. The mutation has produced a new chromosome with different characteristics to the parent chromosome. Mutation is primarily used to introduce diversity into the population to encourage the GA to explore new areas of the solution space.

Crossover operates on two chromosomes, combining their genetic material to produce ‘child’ chromosomes. The application of ‘two-point’ crossover on a binary-string representation is illustrated in Figure 4.3

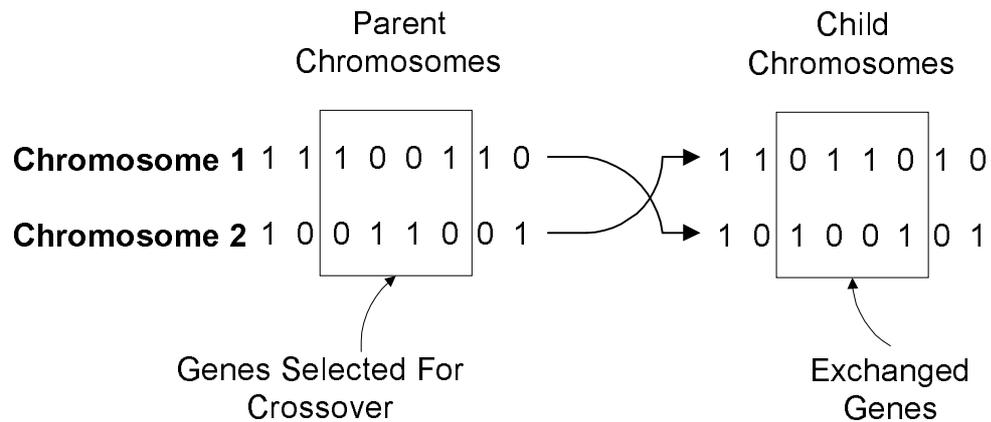


Figure 4.3 Example of a Crossover Operation

In this example a region of the same size has been selected in both chromosomes. Each region is swapped with that in the other chromosome to produce new children with some genes from both parents, they combine the genetic characteristics of two solutions.

The idea behind crossover is to use the information currently stored within the GA population to explore other regions of the solution space. If the selection of the parent chromosomes favours selection of the fitter individuals then crossover combines the characteristics of good parents to explore regions of the solution space that indicate the potential to yield good solutions.

The illustrated example of crossover, using ‘two-point’ crossover, is one example of a potential crossover operator. Many other crossover methods exist, such as ‘one-point’ crossover, ‘masked’ selection, order-based [Goldberg89, Mitch96, Teuk95b Beasley93a], etc. exist. Particular techniques have been developed to suit the chromosome implementation, such as the splicing of complete tree-subsets used in [Schnecke95]. The important aspect of crossover is the

combination of genetic material to produce new solutions, hopefully with improved fitness characteristics.

Crossover and mutation are typically both applied in a genetic algorithm, at application rates tailored to suit the specific problem. Crossover applied without an associated mutation operator may not be able to explore certain regions of the solution space as it can only combine information that is already present in the chromosomes. It is possible for a single gene to have the same value in all chromosomes. Without a mutation operation that gene would never be changed, consequentially blocking off a region of the solution space from the search. Crossover increases the likelihood of a good set of genes, which may have resulted from a mutation operation, being propagated throughout the population, increasing the efficiency of the search process.

### *4.2.4 Solution Selection*

The selection procedure of a genetic algorithm is a key component of the search process. The original concept of the selection mechanism [Holland92, Goldberg89] was to mimic the process of natural selection in nature. The evolutionary theory of natural selection is based on the idea of ‘survival of the fittest’, where individuals that are more successful within their environment have more chance of reproducing and hence propagating their characteristics into the next generation.

Within the context of a GA, this theory dictates that those individuals that are more successful in meeting the specified objective will have more chance of being chosen for reproduction into the next generation. Hence, the GA attempts to find better solutions by building on the current best solutions. This will guide the

search procedure to those locations within the search space that contain the best solutions.

Since the initial development of GAs, many techniques have been developed that aim to improve and build on this standard idea [Davis89, Beasley93, Whit89, Baker85] However, the basic idea behind all of these techniques is the probabilistic selection of individuals based on their quality. The most commonly used technique is known as the Fitness Proportionate Selection (FPS) method where individuals are selected with a probability based on their quality. Therefore, FPS is a probabilistic selection technique.

The FPS selection method was suggested by Holland based on his analysis of the famous ‘2-armed-bandit problem’ [Holland92, Goldberg89]. FPS allocates to each individual a fixed probability of being selected, based upon its fitness relative to the total fitness of all individuals within the generation.

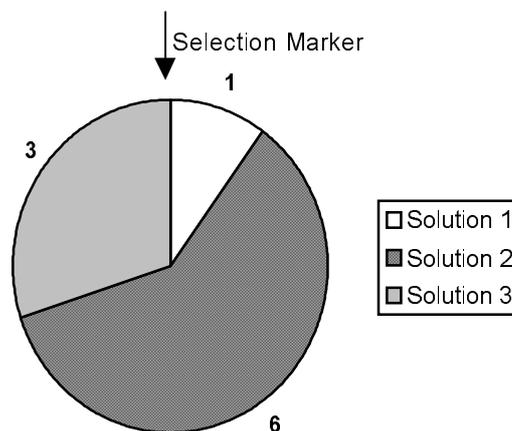


Figure 4.4 Roulette Wheel Creation

One implementation of FPS is to create a ‘roulette wheel’ where the size of segment allocated to each individual is directly related to its fitness. The larger the

fitness of an individual, the larger the segment allocated to it. Figure 4.4 illustrates a roulette wheel created from a population of 3 individuals.

The number next to each slice is that solutions fitness, so it can be seen that the solution with the largest fitness (solution 2) has the largest slice of the roulette wheel. A single spin of the wheel will result in one segment pointed to by the selection marker; this segment will be the selected solution. Repeated spins of the wheel are used to select enough individuals to create a whole generation. Those solutions with a larger slice of the wheel will have greater chance of being pointed to by the selection marker.

An important characteristic of the FPS roulette wheel method is that all solutions have some chance of being selected. A simple greedy algorithm, which always attempts to select the best solution, will repeatedly select solution 2; the search method will become trapped on solution 2. This could lead to sub-optimal solutions if solution 2 is only a local optimal solution. The FPS method gives lower fitness solutions some chance of selection, therefore helping the GA to escape local optima.

### 4.2.5 Stopping Criteria

The GA is typically required to determine a ‘satisfying’ solution to a problem where other search techniques have failed or are not expected to achieve much success. The required parameters for a satisfying solution are specified before the GA is executed. When a solution which achieves the objectives is produced the evolutionary process is terminated i.e. the stopping criteria has been satisfied.

In some cases the GA is used to determine the ‘best’ solution to the problem, the global-optimum point in the solution space. While a GA can not be

guaranteed to find the global-optimum, if implemented correctly it will typically find a ‘good’ or satisfying solution in a reasonable amount of time. In these cases the stopping criteria is usually specified in terms of the number of generations or solution evaluations since the last improvement in fitness was found. The actual number of generations is typically set very high to increase the likelihood that the GA has settled on the highest peak in the solution space.

Other techniques exist for determining whether a GA has terminated a search, such as assessing the convergence of the population. One method is to analyse every solution in the population. If 95% of the genes in each chromosome are identical in all solutions then the GA is considered to have converged and the search is terminated [Beasley93].

There are many variations to the basic GA operators and components described here, all of which are aimed at improving the efficiency and success of the GA [Beasley93a]. This chapter is not intended as an exhaustive review of all the features of a GA, it is an introduction to the fundamental concepts of the technique.

### 4.3 Non-Standard Genetic Algorithms

The previous section introduced the concept of a standard genetic algorithm, comprising of standard GA components that have undergone little or no change since their initial proposal by Holland and his successors. Several researchers using GAs to solve complex, real-world engineering problems have proposed that, for GAs to achieve their full potential in engineering design, it is necessary to specifically design the GA to suit the problem [Davis89, Davis87, Beasley93a]. This leads to the development of non-standard GAs, incorporating domain-specific

design and exploration techniques into a standard genetic framework. The GA based high-level design tool presented in this thesis is an example of such a problem-specific GA, it is effectively a hybrid of power design methodologies and GA search techniques.

Such hybrid systems offer the advantages of exploiting the efficient search technique of the GA while utilising expert knowledge about the problem; a simple GA does not incorporate any expert knowledge. Example hybrid or non-standard GA systems [Esch91, Sinclair98, Davis94, Schaff93, Arslan96a, Martin95, Lienin95] exploit the use of non-standard chromosome representation (e.g. a non-binary chromosome such as a tree representation of each component in a VLSI layout task), adaptive operator rates and non-standard genetic operators. Non-standard genetic operators incorporate problem-specific techniques to modify the chromosome according to standard design rules. Incorporation of these rules has been shown to improve the efficiency and results of the GA-based search technique for a range of problems. The primary advantage of hybrid or non-standard GA systems is their combination of the benefits of the GA and the standard design techniques. They enable design exploration to utilise existing techniques within a GA framework.

### 4.4 Applications of Genetic Algorithms

Since their inception GAs have been applied to a wide range of problems such as machine learning, scientific modelling and engineering optimisation or design applications [Mitch96]. The applications described in this section concentrate on those used during the VLSI design process as the application

described in this thesis is an example of a GA to optimise a parameter (power consumption) during VLSI design.

In the design and application of VLSI devices, GAs have been shown to produce superior solutions compared to other techniques for problems such as test-pattern generation [Odare94, Arslan96c]. In [Schnecke95, Schnecke95a] the GA was used to optimise the layout of a VLSI device, simultaneously minimising wire routing requirements (through optimal placement of cells) while solving the complex problem of obtaining a valid routing design. In the example described, the GA was used to both solve the complex problem and inherently optimise the produced solutions. The consideration of routing and cell placement in parallel enables a better solution to be determined in comparison with the traditional method of considering placement and routing as separate stages.

The use of parallel GAs [Chipp97] has also been demonstrated to provide an efficient VLSI routing tool in [Lienig97] while simultaneously minimising area and capacitance. The reduction of capacitance reduces the delays in connections, hence increasing the speed of the VLSI device. Again, the results obtained with the GA are superior to those found using other techniques.

GAs have also been used at the transistor-level to evolve VLSI designs for digital and analogue applications [Davis94], increasing the speed of the design process and optimising objective parameters.

In [Arslan96a, Arslan96b] a GA has been used to solve the complex problem of logic-level VLSI design. A library of component cells describing a variety of digital logic functions is provided for the GA. The GA synthesises a design, constructed from the cell library, which performs a specified function such as a binary adder/subtractor or a parity checker. The results demonstrate a primary

advantage of GAs, the ability to generate a range of solutions that satisfy the problem criteria but have different implementation characteristics. This is due to the multiple-solution nature of the GA search technique that processes and evaluates a large number of possible solutions. The generation of alternative designs is a useful feature for the VLSI designer as it gives them greater flexibility to integrate those designs into the whole system.

GAs have also been applied to higher-levels of the design process to optimise the scheduling and allocation steps in converting an algorithmic level description into a logic-module layout. Scheduling and allocation assign specific modules in the VLSI device to operations specified in the algorithm. This process has been tackled using a variety of heuristic solutions and techniques, none guaranteed to find the optimal solution due to the high degree of dependency between the tasks. The GA is used to find an optimal trade-off point between the conflicting parameters, to optimise for area [Grewal97] and reduction of the time taken to perform the scheduling and allocation process [Mandal96, Grewal97].

Power-Profiler [Martin95, Martin96] illustrates the use of a GA to tackle some aspects of the problem of low-power VLSI synthesis. The work presented in this thesis optimises a high-level design for low-power implementation. Power-Profiler operates at a lower level of the design process, optimising power during the scheduling, allocation and binding process of a specified high-level algorithm. The GA is used to assign VLSI cells such as adders and multipliers to operations in the algorithm. Each operation has a number of alternative cells such as the use of Booth, Array, etc. cells for a multiplication operation. Each multiplier cell-type has the same function but different area, speed and power characteristics. By assigning slower but less power-hungry cells to non-time-critical operations in the algorithm

the GA reduces the overall power consumption while simultaneously satisfying area and speed requirements.

Stages of the work presented in this thesis have been published in various periodicals and conferences [Bright96, Arslan96, Bright97, Bright98, Bright98a] illustrating the first use of a GA for manipulating a high-level design specification to reduce its VLSI power requirements.

This section has presented a number of successful applications of GAs to various problems of the VLSI design process. The systems described illustrate that the GA is useful at all stages of the design process both to solve the complex problems and optimise the VLSI implementation characteristics.

### 4.5 Application of GAs to the Problem of Low-Power Synthesis

The technique used for power reduction in this work was previously described in section 2.4. One of the main disadvantages of the technique is its complexity. It has been illustrated that the use of a single transformation, used for area minimisation, presents an NP-complete problem i.e. a problem that cannot be solved in polynomial time [Potko91].

Use of additional transformations further increases the complexity of the problem due to the interaction between transformations e.g. one transformation may subsequently enable or prevent the application of another. Previous use of transformations for high-level optimisation has required the development of specific heuristics [Potko91, Huang94] to perform a search of the solution space to determine the best solution. In [Chan95] a simulated annealing algorithm (another algorithm inspired by a natural process) was used to assist the transformation

application process. In that work, the application of the simulated annealing algorithm was limited to design refinements once the design had been transformed using application heuristics.

As discussed previously in this chapter, GAs have been applied to many VLSI design problems, producing results which typically outperform other techniques in many situations. In addition to the proven track record of GAs in VLSI design, the complexity of the high-level low-power design problem makes it suitable for the investigation of the application of GAs, to attempt to produce a tool that synthesises designs which satisfy the objective criteria.

## **Chapter 5 - The Genetic Algorithm**

The previous chapters have presented the techniques of power reduction, power estimation and GA-based search and optimisation. To investigate the use of GAs in a low-power design environment a prototype tool is developed to optimise high-level designs for low-power VLSI implementation. This chapter presents the development of the initial prototype **Genetic Algorithm for LOw Power Synthesis (GALOPS)**.

The prototype version of GALOPS processes an original specified signal-processing algorithm, to produce a new algorithm that has lower power consumption requirements than the original version. The target implementation platform of the algorithms is a CMOS VLSI device.

The main characteristic of the GALOPS tool is the use of high-level transformations, embedded within the genetic search and optimisation framework, to enable power exploration.

GALOPS is based on a typical GA structure as introduced in section 4.2. The structural flowchart of GALOPS is illustrated in Figure 5.1. The first step is to create the initial population of candidate solutions. As discussed previously in section 4.2 (GAs), the GA requires candidate solutions to be encoded in an appropriate format for GA based manipulation. The problem of encoding high-level signal processing designs in a suitable format is discussed in section 5.1.

Section 5.2 describes the population initialisation procedure, specifically addressing the conflicting requirements of a GA requiring a random starting point

and a VLSI synthesis system that needs to develop designs with a specific non-random function.

At the core of a GA is its selection procedure, which is used to select those designs that will be used to create the next population (the next generation) of solutions. This procedure is presented in section 5.3.

One of the most important features of a GA is its use of genetic modification procedures to actually perform the solution space exploration. The use of these operators within a high-level synthesis system requires significant modification to standard GA techniques. These genetic modification procedures are presented in sections 5.4 and 5.5.

The design evaluation procedure is used to assess the individual quality of each design. In the case of GALOPS this requires complex estimation of VLSI parameters such as the calculation of power consumption, area and supply voltage. The mechanisms for the evaluation of candidate solutions are presented in section 5.6.

Sections 5.7 to 5.9 describe the use of benchmark designs to calibrate the prototype GA synthesis tool and present initial results illustrating its use as a low-power synthesis tool.

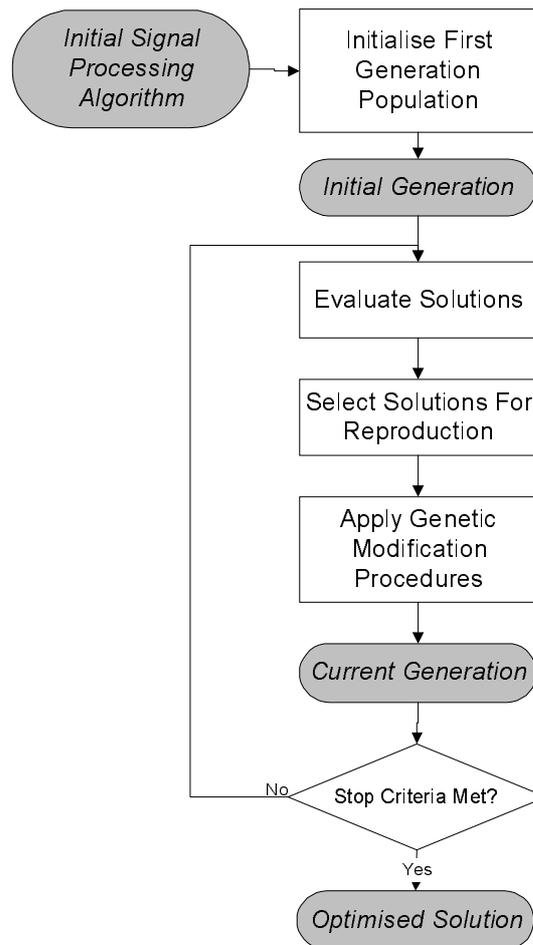


Figure 5.1 Overview of GALOPS – Low Power GA-Based Synthesis Tool

### 5.1 Problem Representation

GALOPS optimises high-level signal processing algorithms for low power CMOS VLSI implementation. To manipulate these algorithms GALOPS uses an internal DFG [Davis82] structure as discussed in section 2.1.1. This section describes the techniques used to incorporate this representation format into a GA framework.

### 5.1.1 Chromosome Representation

To use the GA as a high-level synthesis system the DFG algorithm representation format needs to be incorporated into a chromosome representation, to fully integrate with the various features of a GA.

The chromosome within GALOPS needs to store a large amount of information to fully represent a candidate solution DFG. Each computational node needs to be stored, together with the precedence information vital for preserving the knowledge of the correct algorithm function. A bit-string representation suggested by Holland [Holland92] and discussed in section 4.2, would require a very large chromosome to fully encode the function and precedence constraints of each node in the DFG. Even if the length of such a chromosome was not prohibitive, the amount of encoding and decoding required in interpreting the chromosome for evaluation and modification places an unnecessary burden on the already complex synthesis process. Several researchers using GAs to solve complex, real-world engineering problems have identified this problem of bit-string chromosome representation [Davis89]. Previous research has led to the recognition that, for GAs to achieve their full potential in engineering design, it is necessary to specifically design the GA to suit the problem [Davis89, Davis87, Beasley93a]. An important part of this process is the development of a problem-specific chromosome to fully utilise the abilities of a GA.

GALOPS uses a chromosome structure that is a complete representation of a DFG, storing both the required functional and precedence constraint information. This chromosome structure was chosen to minimise the amount of encoding and decoding that is required both for chromosome evaluation and modification during

the synthesis procedure. Figure 5.2 shows the chromosome template used to store a complete DFG.

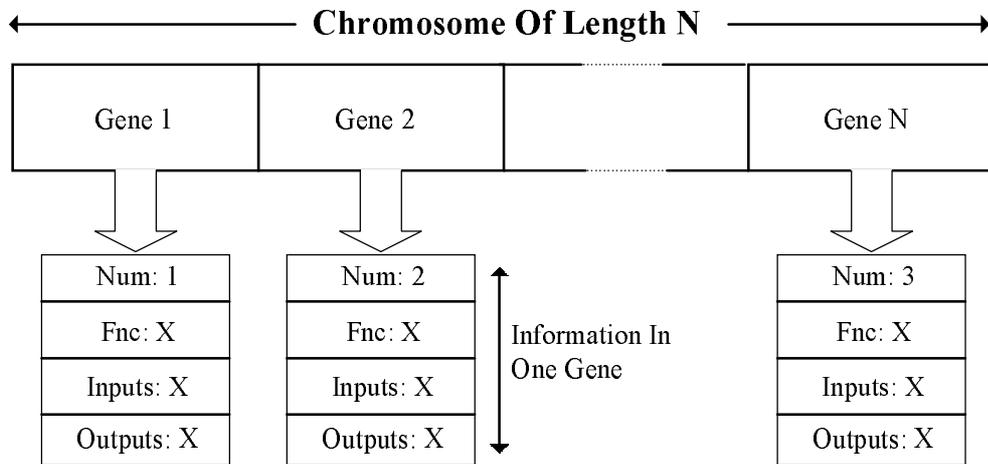


Figure 5.2 GALOPS Chromosome Template

As illustrated in Figure 5.2, each node of the DFG is stored in a separate gene in the chromosome. Within each gene is stored the relevant information for that node. This information consists of the following:

- **Num** – A unique Number used to identify each gene.
- **Fnc** – The computational Function of that node
- **Inputs** – A list of Input edges of that node.
- **Output** – The Output edge of that node.

To illustrate the storage of a DFG within this chromosome template Figure 5.3 depicts the FIR Filter DFG of Figure 2.1 (in section 2.1.1) stored as a chromosome. This example also illustrates the storage of the relevant information for each node e.g. node 2 (Num: 2) is a delay node (DEL) with 1 input net (e1) and an output net (e2).

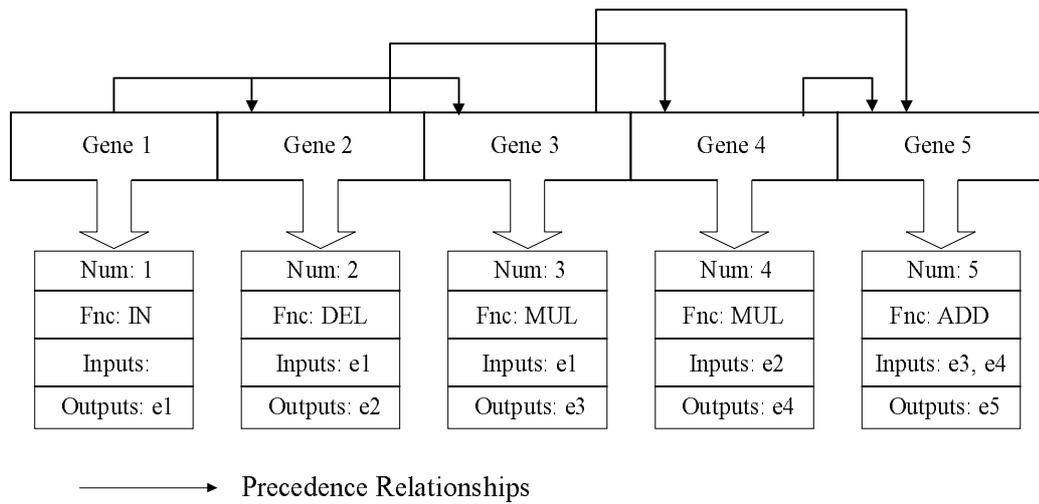


Figure 5.3 Example Chromosome Representation

The precedence lines in the example, though not actually present within the chromosome, illustrate how the **Inputs** information is used to extract the precedence information between each gene.

This chromosome structure has been specifically tailored to suit the low power high-level synthesis problem, providing a problem representation format with the following properties:

- Gene Based Information Storage – The input and output information in each gene ensures that the precedence constraints present in the DFG are preserved within the GA. GALOPS manipulates chromosomes without modifying the contents of the gene. This enables the GA to modify the overall chromosome without corrupting precedence information stored within genes.
- Problem-Specific VLSI Parameter Information – When evaluating a particular DFG a number of practical VLSI design parameters are used. The function (Fnc) of each node can be used to look up area, delay and capacitance characteristics specific to that node type.

- Storage of Primary Inputs – The primary inputs of the DFG are encoded as functional nodes. This provides convenient entry points in the chromosome from which it can be processed for evaluation or modification. The primary outputs are not required during the modification or evaluation processes so they are stored externally to the chromosome in a lookup table. This outputs table is used to decode the chromosome back into a high-level design (represented as a DFG) at the end of the genetic synthesis process.
- Variable Length Chromosome – Traditionally, GAs have used chromosomes of a fixed length to simplify their computational complexity. However, the modifications applied to the DFGs produce chromosomes of varying sizes throughout the evolutionary process. A fixed length chromosome, with “empty” genes left for future growth, is memory inefficient and may not be large enough to accommodate all possible modifications. To ensure complete flexibility the chromosome is of variable length so genes can be added and removed during the modification process.
- Complete DFG Representation - The chromosome representation format is capable of completely storing a DFG with varying numbers of nodes, types of node and precedence constraints.

Because GALOPS uses a specific chromosome format that has non-binary (and even non-alphabet [Davis89]) representation it precludes the use of a standard ‘off the shelf’ GA package for implementation [Filho94]. Packages such as GENESIS [Gref84] (one of the more popular standard GA packages) are typically used for binary-string and sometimes alphabet-string chromosome representation. In addition, such packages use a set of genetic operators suited to the particular

chromosome representation. The specific chromosome representation of GALOPS requires genetic operators designed to handle the structure. Therefore, all routines used in GALOPS are specifically developed as part of this research.

### 5.1.2 Input Data Format

The candidate designs, to be optimised for low-power operation, are presented to the synthesis tool in a netlist format used to describe combinational logic circuits in [Bright95]. The tool described in [Bright95] developed compilation routines that were used as a base for the chromosome compilation routines described in this work. Figure 5.4 gives an example netlist of the DFG presented in figure 2.1.1 (section 2.1.1).

The header lines at the beginning of the netlist file are used to describe the design and also list its input and output nets. This is then followed by the main body of the netlist. Each line in this netlist describes a functional node within the algorithm, followed by its data transfer inputs.

```
$ 2nd order FIR Filter
$ this is a comment line
$ primary inputs
e1
$ primary outputs
e2
$ circuits description
$ DFG level
$      output  type  inputs
      e2      del   e1
      e3      mul   e1
      e4      mul   e2
      e5      add   e3   e4
```

Figure 5.4 Example GALOPS Input File

The example shown contains 4 elements within the DFG; a delay, 2 multipliers and an adder. The output and input tags refer to the connections to each

operator e.g. the adder has two input nets (e3 and e4) and one output net (e5). The compiler routines within GALOPS convert this netlist into an internal representation of the DFG.

### 5.2 Population Initialisation

An important component of a GA based search mechanism is a series of random starting points [Beasley93]. This enables the GA to start searching the complex solution space in a range of locations, helping to prevent it from presenting local optima as the best solution. A traditional technique to generate such an initial population would be to repeatedly generate chromosomes with random genetic properties until the required population size has been created [Holland92, Goldberg89, Davis91]. However, the purpose of GALOPS is to optimise, for low power operation, signal-processing algorithms with a specific function. Throughout the optimisation procedure it is of paramount importance that the function of the specified design is not corrupted.

Random generation of chromosomes would not only produce designs with incorrect functionality, it would also generate a large number of unfeasible designs e.g. output nets connected to output nets. To solve this problem GALOPS uses the initial specified design as a seed for the random population.

The low power design techniques, introduced previously in section 2.5, are applied to the initial design, using a random selection to determine the type of technique applied. Each application produces a new design with different characteristics, effectively creating a new starting point from which to explore the low power solution space. Repeated application of the low power design techniques generates an initial population of random designs, from which the GA can begin its

search of the solution space. The initial population size is specified as a GA parameter.

### 5.3 Selection Procedure

The selection procedure, described in section 4.2.4, is used to select individuals for reproduction and hence create the next generation of solutions in the GA. The FPS scheme was outlined in section 4.2.4. Although a popular technique the FPS method does have some limitations. Figure 5.5 illustrates one of the problems of FPS. This figure depicts a number of solutions (denoted with X's) within an imaginary solution space, plotted across a fitness axis. The fitness values of each solution are also shown in Figure 5.5.

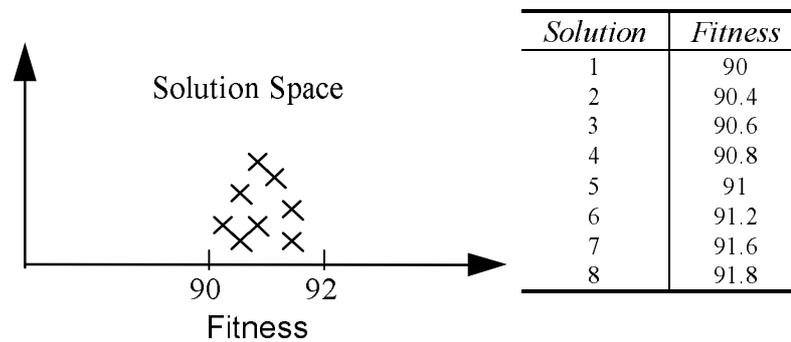


Figure 5.5 Plot of Solutions within a Search Space

The figure shows a number of solutions within a small fitness range. If the minimum fitness is separated from the maximum fitness by only a small increment then the FPS technique will create a roulette wheel where each solution will have an almost equal share of the wheel. Therefore, the selection pressure on each individual will be the same, rather than favouring the more fit individuals. This is a problem of 'fitness-scaling' [Beasley93].

A number of techniques have been developed to improve this aspect of the FPS reproduction process, such as Windowing [Beasley93, Hanc95] and Sigma Scaling [Beasley93, Hanc95]. The prototype version of GALOPS uses a technique known as ‘linearisation’ [Hanc95] to tackle the problem of ‘fitness-scaling’.

The linearisation process assigns a fitness value between 0 and 1 to every individual, based on their fitness relative to the total fitness. The formula to calculate an individual’s linearised fitness is:

$$linear\_fitness = \frac{fitness - \min(fitness)}{\max(fitness) - \min(fitness)} \quad (5.1)$$

Figure 5.6 illustrates how the process of linearisation effects the spread of fitness values shown in Figure 5.5.

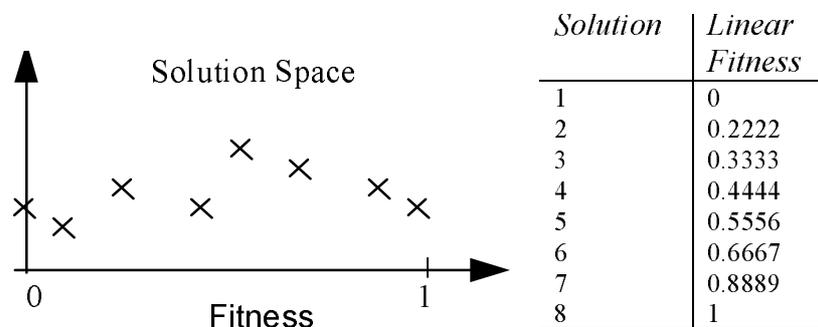


Figure 5.6 Effect of Linearisation on Fitness Values

As can be seen from the graph, the linearisation process spreads out the population between the specified minimum and maximum values, magnifying the differences between a range of solutions with similar fitness values. These linearised fitness values are used in the roulette wheel in place of the actual fitness values. The linearisation operation results in good solutions (linear fitness close to

1) taking a significantly larger segment of the roulette wheel in comparison to poor solutions (linear fitness close to 0). Hence, the better solutions will have a significantly greater chance of selection than the poorer solutions.

### 5.3.1 Implementation of Selection Procedure within GALOPS

The algorithm for selection of individuals, based on the previously discussed techniques, is summarised in Figure 5.7.

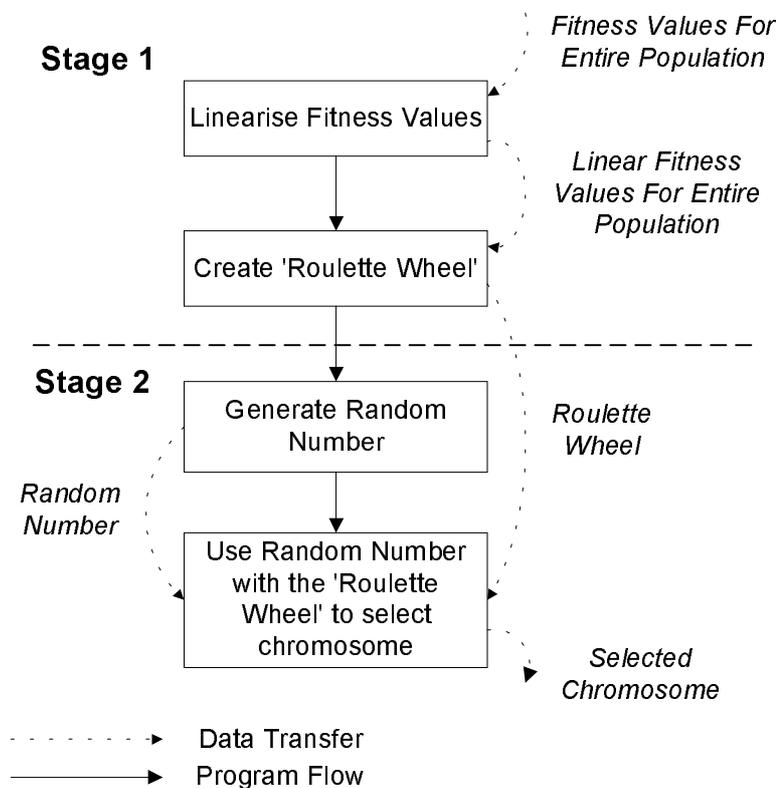


Figure 5.7 Chromosome Selection Procedure

The flowchart depicts both the program flow and the transfer of data. This algorithm is executed in two stages. The first stage, executed once per generation, creates the linearised values and the Roulette wheel. Stage 2 is then repeated until enough individuals have been selected to fill the next generation of the GA.

To efficiently select the desired chromosome from the roulette wheel the Bisection Method [Teuk95b] is used to rapidly search the wheel for the chromosome selected by the random number. A simple incremental search through the roulette wheel will require a maximum of  $M$  iterations, where  $M$  is the population size. The Bisection Method reduces the number of iterations to a maximum of  $\log_2 M$ , a considerable reduction in complexity.

The selection procedure is used to select the transformations for simple reproduction (copying to the next generation) and also for genetic modification, such as crossover or mutation.

### 5.4 Genetic Mutation Operators

The importance of a mutation operator within the genetic synthesis system was emphasised in section 4.2.3. To summarise, the mutation operator introduces diversity into the population by modifying the properties of chromosomes. This modification produces a new chromosome with different qualities. This section describes how the concept of mutation is incorporated into a genetic synthesis tool.

A Standard genetic mutation process randomly mutates the value of individual genes in a chromosome to produce genes of other values [Holland92, Goldberg89]. Within the context of the chromosome representing a DFG, such a mutation process would act on the nodes of the DFG. A typical mutation could be the replacement of one type of operator with another. Figure 5.8 illustrates the problem of incorporating this idea directly into the synthesis process.

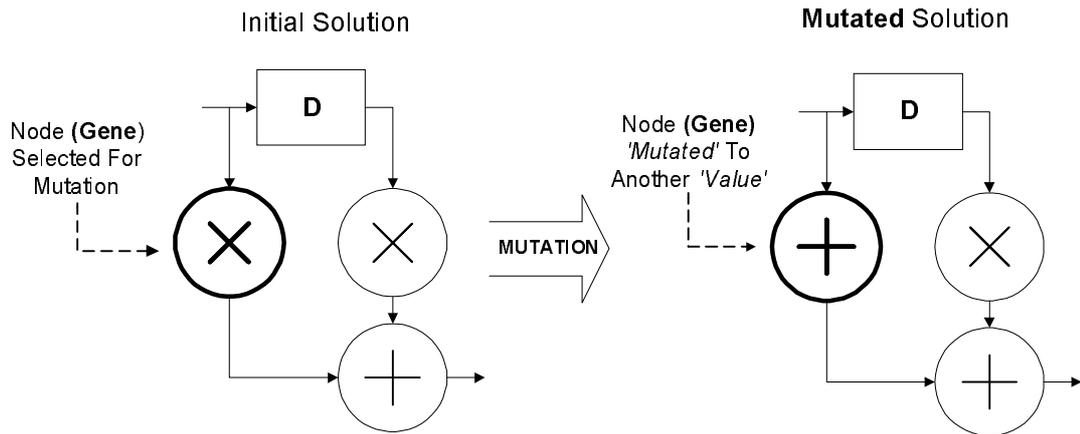


Figure 5.8 DFG Mutation Process

In this example, application of a random mutation (replacing a multiplier with an adder) has resulted in the mutated DFG having a different function to the initial DFG. If this mutation scheme were to be applied to the entire population of solutions, it would result in a large number of chromosomes with incorrect functional operation i.e. a function different to that of the specified DFG. These solutions would be unacceptable, as the low power solution must be one that has no functional difference to the original solution. Therefore, the mutation modification process needs to consider the importance of DFG functionality.

One method of modification used in genetic synthesis tools, where functionality of the chromosome is of paramount importance, is to use a 'repair operator' [Arslan96a, Arslan96b].

The repair technique analyses the function of mutated chromosomes. If a chromosome is corrupt (i.e. incorrect function) then the repair operator is invoked. The aim of the repair operation is to produce a chromosome with the desired function by modifying the corrupt chromosome. The repair technique requires detailed knowledge of both the required and actual function of the chromosome.

Such information for a DFG would be obtained through simulation, a computationally intensive task.

The computation expense is not the only disadvantage of using the repair operator within this VLSI synthesis system. Consider a single gene mutation of a complex DFG, replacing an adder with a multiplier, altering functionality. With the complex feedback paths and precedence constraints of a DFG, repairing the chromosome could require a considerable alteration of the DFG structure, which could negate the work of the mutation operation. In effect, the repair operator could return the mutated DFG to its initial state. Therefore, the use of a repair operator can affect the efficiency of the GA.

An alternative to using a repair operator is to modify the mutation process so that it does not corrupt DFG functionality; i.e. it does not produce incorrect chromosomes. This requires the development of 'problem-specific operators' that incorporate information specific to the problem [Beasley93a, Gref87]. In the case of high-level synthesis, problem-specific operators will only apply modifications that do not corrupt the functionality of the DFG.

Section 2.5 introduced HLT techniques that operate on DFGs to modify their implementation characteristics, such as speed and power. The HLTs are VLSI design techniques that dictate what modifications can be made to a DFG without corrupting the desired function. These HLTs are used as the mutation operators of the genetic synthesis system.

The HLTs can be considered as mutation operators as they operate on individual nodes within the DFG (genes within the chromosome) to produce a new DFG (chromosome) with different characteristics (fitness). An example of a

mutation operation is shown in Figure 5.9. This mutation operation uses the Retime transformation.

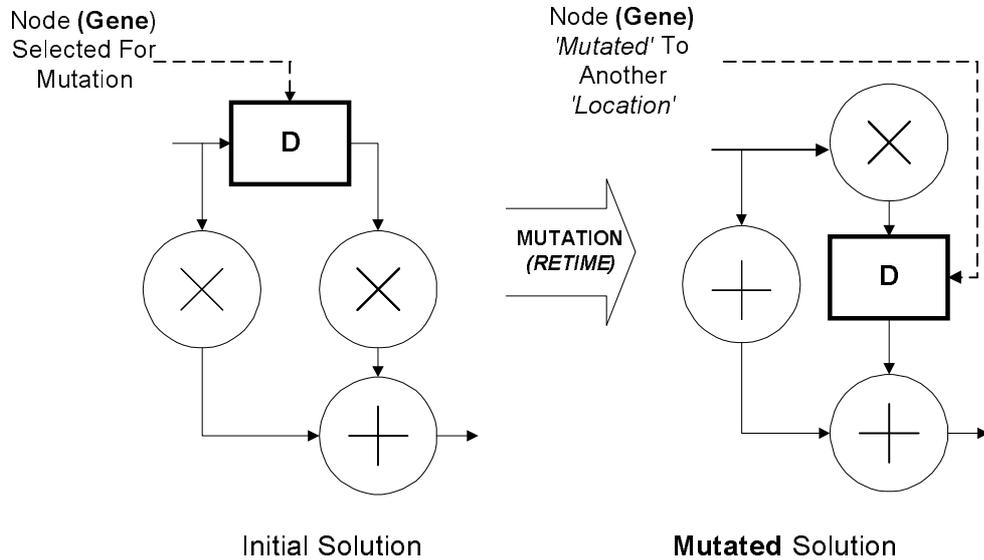


Figure 5.9 Example Mutation Operation Using A Retime Transformation

In this example, the mutation operation selects a delay element in the DFG. The retime transformation is used to move the delay through the adder to produce a new DFG with a different topology but the same function, a retimed DFG. This produces a DFG with different implementation characteristics to the original.

The prototype version of GALOPS has a number of transformations, introduced in section 2.5. The following sections describe how each of these transformations is implemented as a mutation operator within a genetic synthesis tool.

## 5.4.1 Retime Transformation as Mutation

The properties of the retime transformation, how it modifies the DFG, were discussed in section 2.5.1. This section describes how the properties of this transformation are incorporated into GALOPS as a genetic mutation operator.

The basic function of a retime operation is to move a group of delay nodes from the input net of a node to its output net. Therefore the retime-mutation in GALOPS must select a delay in the DFG and move the delay under the appropriate retime rules to ensure the DFG is not corrupted. Figure 5.10 presents the pseudo-code for the retiming algorithm.

**Algorithm Retime**

```

    Passed chromosome to be mutated

    Randomly select a delay_node in the chromosome

    Check retime constraints are met:
        IF the delay_node drives a primary output net
            DO NOT perform retime
        EXIT algorithm

        Randomly select an output_node driven by the delay_node
            IF the output_node is a delay
                DO NOT perform retime
            EXIT algorithm

        Check retime is possible on selected output_node
            IF NOT possible to retime
                DO NOT perform retime
            EXIT algorithm

    End check retime constraints

    Perform the retime operation:
        Remove the delay_node from the current location (remove gene)
        'Rewire' the nets around the removed delay
        Insert delay on output net of the selected output_node (insert gene)
        'Rewire' the nets around the inserted delay
    End perform retime operation
End Algorithm Retime

```

Figure 5.10 Pseudo-Code for the Retime Mutation Algorithm

The retime algorithm randomly selects a delay in the DFG that is to be retimed. The random selection of a delay exploits the ability of a GA to not require

detailed information about the problem to determine an optimum solution. Identifying a ‘good’ retime operation would be a complex process that would require detailed knowledge of both the problem domain and the DFG. This complexity would be compounded in the case of identifying a retime that has no obvious immediate benefits, but yields a DFG with potential to become an optimum solution. Rather than trying to identify a ‘good’ delay, the GA is used to apply repeated selection of a random delay. Throughout the evolutionary search, the GA will evaluate a wide range of possible retime operations, rather than determining a set of specific applications.

The **check retime constraints** sub-routine ensures that the non-corrupting properties of the retime transformation are incorporated within the retime algorithm. Within this sub-routine, some problem-specific knowledge is incorporated to increase the efficiency of the search process. If the selected delay node has been chosen to be retimed through another delay then the retime process is not performed. Retiming a delay through another effectively results in the two delays swapping locations, resulting in no change in the DFG; therefore, this is prevented from occurring.

Once the initial selections and checks have been performed and satisfied, the algorithm performs the **retime operation**, which consists of four distinct stages. The delay node is removed from the chromosome and a new delay node is inserted. The DFG is ‘rewired’ following the standard retime transformation rules, resulting in a DFG where the initial selected delay node has been retimed forward through a selected node. Figure 5.11 illustrates the four stages of this process.

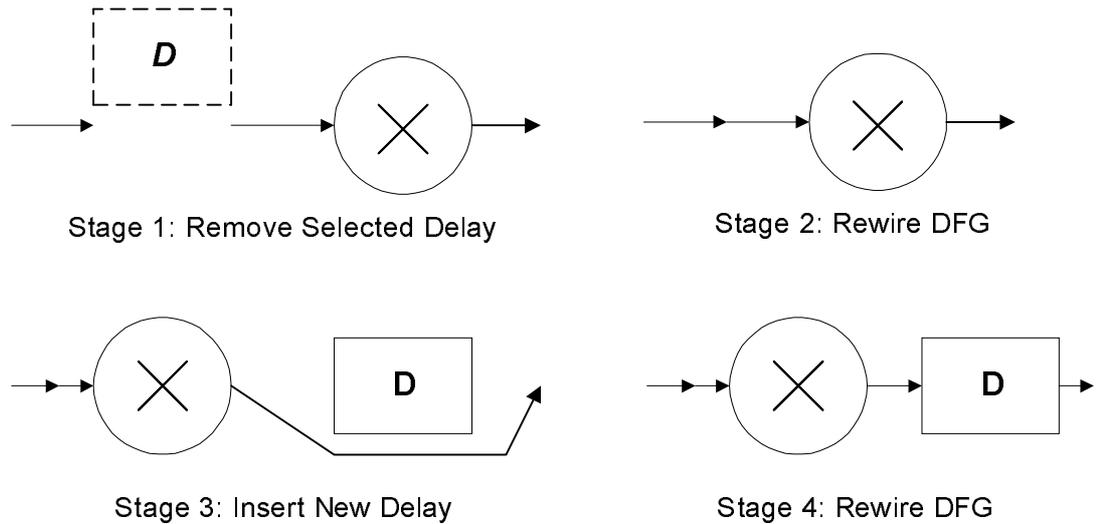


Figure 5.11 Four Stages of the Retime Mutation Algorithm

#### 5.4.2 Back Retime Transformation as Mutation

A particular feature of the retime operation is that it is bi-directional. The retime operation retimes delays from the inputs of a node to its output, but it can also perform a ‘back’ retime operation. This ‘back’ retime operation consists of moving a delay from the output of a node to its inputs, as illustrated in Figure 5.12.

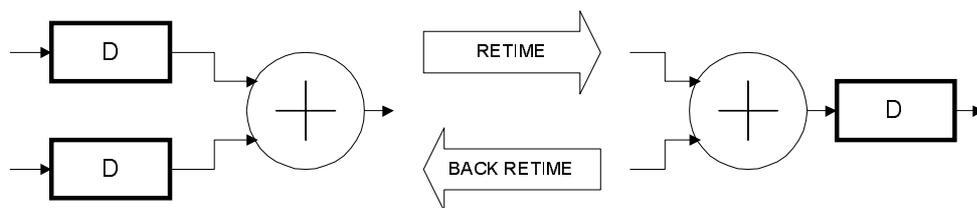


Figure 5.12 Example of the Back Retime Transformation

Although the back retime operation is an integral part of the retime transformation, it is implemented as a separate transformation. This enables GALOPS to apply back retiming as a separate mutation stage. The pseudo-code for the back-retime operation illustrated in Figure 5.12 is shown in Figure 5.13.

As with the retime transformation, the back retime operation must obey certain constraints to ensure that its application does not corrupt the functionality of the DFG. The primary constraint is that, for a delay node to be back retimed through a specified node, all of the fanout nodes of the specified node must be delay nodes. All of these delay nodes can then be removed and delay nodes placed on all input nets of the specified node. The execution of the back retime operation, replacing the nodes and rewiring the nets, is similar to the retime operation shown in Figure 5.11.

### Algorithm **Back Retime**

```
    Passed chromosome to be mutated

    Randomly select a delay_node in the chromosome
    Determine fanin_node of this delay_node

    Check back retime_constraints are met
        Determine all fanout_nodes of the fanin_node
        IF NOT all fanout_nodes are delay nodes
            DO NOT perform back retime
            EXIT algorithm
    End Check back retime constraints

    Perform back retime operation:
        Remove all fanout_nodes (of the fanin_node)
        'Rewire' the nets around the removed delay
        Insert delay node on all input nets of fanin_node
        'Rewire' the nets around the inserted delays
    End perform back retime operation
END Algorithm Back Retime
```

Figure 5.13 Pseudo-Code of the Back Retime Transformation

### 5.4.3 Pipeline Transformation as Mutation

This section describes how the pipeline transformation, which was described in section 2.5.2, is used as a genetic mutation operation within GALOPS.

The effect on the DFG of the pipeline transformation is to insert delay elements at specific points within the DFG. Initially a single pipeline delay is inserted, which then requires the insertion of other delays at 'cutset' points to

preserve the correct function of the DFG. The pseudo-code for the **pipeline algorithm** is shown in Figure 5.14.

Algorithm **Pipeline**

Passed *chromosome* to be mutated

Check **pipeline constraints** are met:

Determine *Critical Path* of the *chromosome*

IF the *Critical Path* is less than 2 nodes long

DO NOT perform pipeline

EXIT algorithm

Randomly select a node from the *Critical Path* (the *selected\_node*)

Select a *fanout\_node* of the *selected\_node*

Check **Pipeline possible** between these nodes:

IF feedback path exists between these nodes

DO NOT perform pipeline

EXIT algorithm

End check **pipeline constraints**

Insert *pipeline\_delay* between *selected\_node* and *fanout\_node*

Trace through the *pipelined\_path*

Use *pipelined\_path* data to insert 'cutset' delay elements

END Algorithm **Pipeline**

Figure 5.14 Pseudo-Code for the Pipeline Mutation Algorithm

The initial step in the pipeline algorithm is to determine the critical path (CP) of the DFG. The CP is the longest computational path in the DFG. The CP is used to invest the pipeline mutation operator with implicit optimisation capabilities. Implicit optimisation is achieved by targeting the application of mutations to those areas of the DFG that it is expected will result in an increase in fitness. In the case of pipelining, which is primarily used to increase the speed of the DFG (and hence reduce supply voltage), implicit optimisation is achieved by only inserting delays along the CP. This implicit optimisation process limits the number of pipeline stages inserted in a DFG (excessive pipelining is detrimental to both performance and area). Furthermore, this use of expert knowledge about the effects of the

pipeline transformation on power consumption aids in focusing the low power search on certain areas of the solution space, assisting in the efficiency of the search.

The pipeline delay is inserted between two nodes, selected from the CP. The check pipeline constraints sub-routine is used to ensure the correct application of the pipeline transformation. The first constraint checks that the CP contains at least 2 nodes. This check contains implicit optimisation characteristics, as pipelining a CP of less than 2 has no benefit in increasing the speed (a CP of 1 cannot be reduced any further by pipelining).

The second constraint is to ensure that the location for insertion of the pipeline delay element is not within a loop, as pipeline delays cannot be inserted within a loop. If a 'feedback path' exists between 2 elements then they are in a loop. This is illustrated by considering the DFG of Figure 5.15.

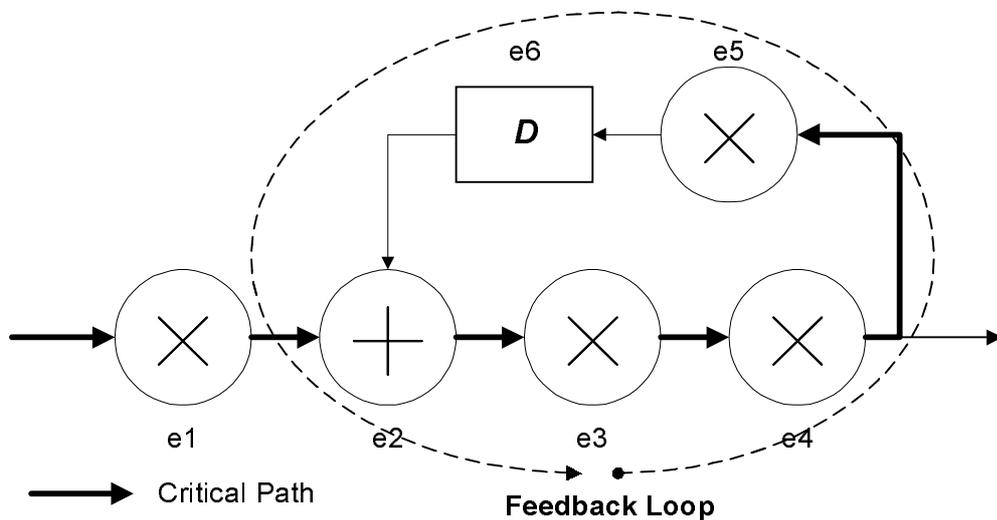


Figure 5.15 Example for Checking Pipeline Constraints

In this example, a large part of the DFG consists of a feedback loop. The CP is shown in bold, encompassing all of the computational nodes within the DFG.

Therefore, each node could be selected for a pipeline operation. However, all nodes except e1 violate the feedback loop constraint, therefore the only possible pipeline location is between nodes e1 and e2.

The loop constraint is checked using a recursive algorithm, which traces through all the fanin nodes of a selected node. For example, if node e2 were selected to have a pipeline delay inserted on its output net (between nodes e2 and e3) then the following process would be used to check the constraints:

```
Select node e2:
  Two fanin nodes, e1 and e6
Select node e1
  No fanin nodes, end search on this node
Select node e6
  One fanin node e5
Select node e5
  One fanin node e4
Select node e3
  One fanin node e2
  Fanin node is equal to initial search node THEREFORE
  Feedback loop exists, pipeline constraint violated
  End search process
```

In this example, the routine correctly detects that the constraint has been violated. Therefore, the pipeline mutation is not possible at that location in the chromosome. If the pipeline constraints are satisfied, the pipeline algorithm inserts the pipeline delay between the selected elements. The next stage of the pipeline process is to insert the 'cutset' delays in the DFG, to ensure that correct functionality is preserved. Figure 5.16 illustrates this process with an example DFG.

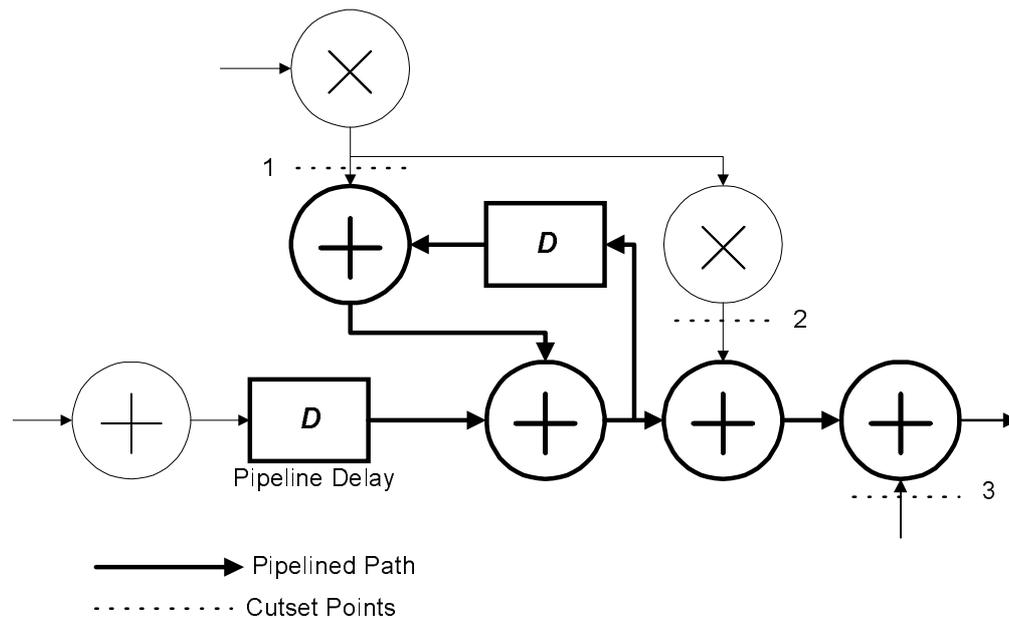


Figure 5.16 Identification of Cutset Points in Pipeline Mutation

The example in Figure 5.16 shows a pipeline delay inserted within a DFG. Also shown are the ‘pipelined path’ and the ‘cutset’ points. The pipelined path is characterised as all nodes within the DFG that are affected by the inserted delay. Identification of the pipelined path requires tracing forwards through the DFG, from the inserted pipeline delay, to identify any nodes along its fanout path. This is achieved using a recursive routine similar to that used to identify an element in a loop, which was described previously. In this example, 4 additions and 1 delay element are identified as belonging to the pipelined path.

After identification of the pipelined path it is used to identify the ‘cutset’ points. These points are any nets that, though not in the pipelined path themselves, are fanin nets of a node within the pipelined path. In this example, 3 ‘cutset’ points have been identified, shown as dotted lines in Figure 5.16. Inserting delay elements on the ‘cutset’ points completes the final stage of the pipeline transformation. This produces a pipelined DFG with no change in function, but an increase in latency. The consideration of latency was discussed in section 2.5.2. The “insert ‘*cutset*’

“delay elements” process, illustrated in the example of Figure 5.16, is summarised in the pseudo-code of Figure 5.17. An example of the application of the pipeline transformation was presented in Figure 2.10.

```
Stage 1 - Determine Pipelined Path:  
  Passed current_node (initially set to the pipelined delay)  
  Identify all fanout_nodes of current_node  
  
  IF no fanout_nodes  
    Exit Stage 1 (go onto next stage)  
  ELSE FOR each fanout_node  
    Store as pipelined_path_node  
    Set as current_node  
    Call Determine Pipelined Path (recursive call to Stage 1)  
End Stage 1  
  
Stage 2 - Determine Cutset Points:  
  FOR each pipelined_path_node  
    FOR each fanin_net of node  
      IF fanin_net not in pipelined_path  
        Store as cutset_net  
End Stage 2  
  
Stage 3 - Pipeline Cutset Points:  
  FOR each cutset_net  
    Insert delay element on cutset_net  
End Stage 3
```

Figure 5.17 Pseudo-Code for ‘Cutset’ Delay Insertion

#### 5.4.4 Automatic Pipeline Transformation as Mutation

The automatic pipeline transformation was described in section 2.5.3. This section presents the algorithms used within GALOPS to execute automatic pipelining as a genetic mutation operation.

The automatic pipeline transformation is a specialised form of retiming and pipelining. The process consists of two distinct stages. The first stage involves inserting delay nodes onto the inputs of the DFG (effectively pipelining the input stage). The second stage requires moving these delay nodes from the input nets into the body of the DFG (the retiming stage). Figure 5.18 presents the pseudo-code for this transformation.

```

Algorithm Auto-Pipeline
  Passed chromosome to be mutated

  Check auto-pipe constraints are met:
    Check retime stage possible:
      For each primary input_net of the DFG
        Check IF delay inserted on input_net
          That subsequent 'retime' step would be possible
        IF NOT possible
          DO NOT perform auto-pipe
          EXIT algorithm

      Check pipeline stage is necessary:
        For each primary input_net
          IF delay node already on input_net
            DO NOT perform auto-pipe
            EXIT algorithm
    End check auto-pipe constraints

  IF auto-pipe constraints satisfied
    FOR each primary input_net
      Insert delay element on input_net
      'Retime' delay element through node driven by input_net
End Algorithm Auto-Pipeline

```

Figure 5.18 Pseudo-Code for the Automatic Pipeline Mutation Algorithm

As with all of the transformations, automatic pipelining has a number of constraints that must be satisfied before application. The first constraint in Figure 5.18 checks that the transformation can be applied to the DFG. This check ensures that the first stage was successful before executing the second step. By checking this before inserting the delays, it prevents the unnecessary execution of the first stage in the case where automatic pipelining is not possible.

The second constraint is another example of the transformations implemented to contain implicit optimisation characteristics. This stage checks that there are no delay elements on the input nets of the DFG. The insertion of delay elements on the input nets of a DFG, which already contains delay nodes on its inputs, could result in redundant automatic pipeline stages. If left unchecked this could lead to a large number of delays inserted on the input of the DFG. As these delays would not affect functionality and have a marginal impact on power

consumption, they could go undetected by the synthesis process. Thus, the constraint prevents this from occurring.

If the constraints are satisfied, the automatic pipeline process is executed in two stages, pipelining and retiming, to produce a DFG that has been automatically pipelined (as illustrated in the example Figure 2.11).

### 5.4.5 Unfolding Transformation as Mutation

The properties of the unfolding transformation, how it modifies the DFG, were discussed in section 2.5.4. This section describes how the properties of this transformation are incorporated into GALOPS as a genetic mutation operator.

The 3 steps of the unfolding transformation, including the rules to preserve precedence information and functionality, were initially presented in section 2.5.4. To summarise, the original DFG is known as the 'parent' and the unfolded DFG is known as the 'child'. The unfolding process creates  $N$  child nodes for each parent node, where  $N$  is the unfolding factor. The connection of child nodes to form the completed unfolded DFG follows rules that preserve the functionality of the DFG. To ensure precedence information is correctly 'unfolded', for each child node  $U_x$  in the unfolded DFG, it is necessary to be able to identify its parent node  $U$  in the original DFG. This information is also necessary to determine the number of delay nodes on nets in the parent DFG, to correctly perform Steps 2 and 3. To facilitate this, during the unfolding process the GA creates both the child DFG and an '*unfolding list*' which stores the relevant information. Figure 5.19 illustrates how this unfolded list is created during the execution of Step 1 for a single node in the parent DFG.

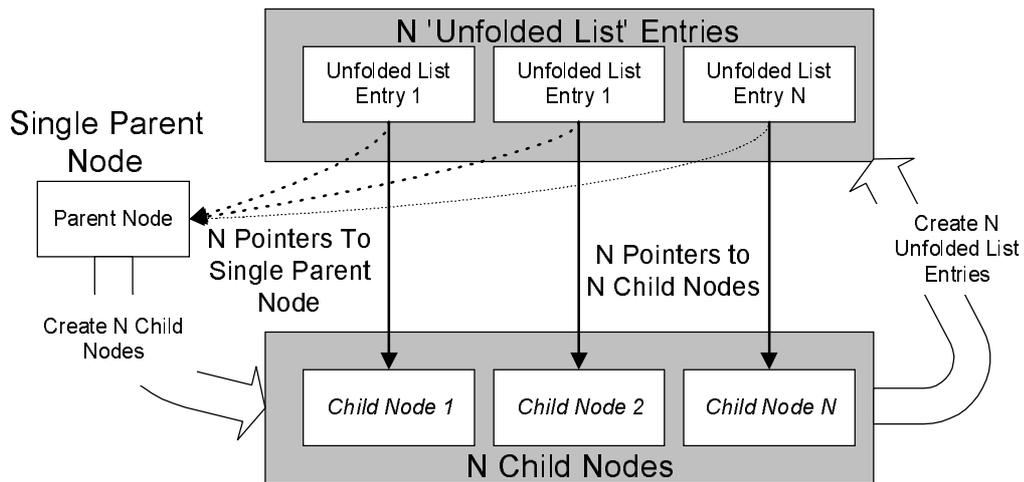


Figure 5.19 Step 1 Of Unfolding Transformation

The figure shows that for each child node created an associated node is created in the unfolded list, shown at the top of the figure. Each node in the unfolded list contains a pointer to both the child node and the single parent node from which it was created. Effectively, the unfolded list creates a link between a child and its associated parent node. Using the unfolded list, the parent node of any child node can be rapidly identified. Once the information stored in the unfolded list has been used to completely construct the unfolded 'child' DFG, the unfolded list is deleted.

Using this concept of the unfolded list, the pseudo-code for Step 2 of the unfolding process is shown in Figure 5.20.

**Algorithm Step 2 of Unfolding Transformation**

```

FOR each entry in the unfolded list determine child_node (node_V)
  Determine fanin_node of node_V's parent_node
  IF fanin_node IS NOT a delay node
    Determine child_node of this fanin_node (node_U)
    IF unfolded_integer of node_U EQUALS unfolded_integer of node_V
      Connect node_U to node_V
    
```

End Algorithm **Step 2**

Figure 5.20 Pseudo-Code for Step 2 of the Unfolding Transformation

The pseudo-code for the next step of the unfolding transformation, Step 3, is shown in Figure 5.21.

**Algorithm Step 3 of Unfolding Transformation**

```

FOR each entry in the unfolded list determine child_node (node_V)
  Determine fanin_node of node_V's parent_node
  IF fanin_node IS a delay node
    Count number of delay nodes on the entire net
    IF number of delay nodes IS LESS THAN unfolding factor (N)
      EXECUTE Step 3A
    ELSE
      EXECUTE Step 3B
  END FOR
Step 3A:
  IF unfolded_integer of node_V IS LESS THAN number of delays
    Determine fanin_node of the delays in parent DFG
    Determine child_node (node_U) with unfolded_integer EQUAL TO
      N-number_of_delays+unfolded_integer
    Connect node_U to node_V
  ELSE
    Determine fanin_node of the delays in parent DFG
    Determine child_node (node_U) with unfolded_integer EQUAL TO
      N-number_of_delays+unfolded_integer
    Connect node_U to node_V with ONE delay node on the net
  END Step 3A
Step 3B:
  IF number of delays IS GREATER THAN unfolding_integer of node_V
    Determine parent_node of node_V
       $q = \text{unfolded\_integer}$ 
    Compute ceiling function to find required child_node (node_U)
       $\lceil (\text{num\_delay\_nodes} - q + 1) / N \rceil \times (N - \text{num\_delay\_nodes} + q)$ 
    Insert 'ceiling function' num of delays on output net of node_U
       $\lceil (\text{num\_delay\_nodes} - q + 1) / N \rceil$ 
    Connect last inserted delay to fanin net of node_V
  END Step 3B
END Algorithm Step 3

```

Figure 5.21 Pseudo-Code for Step 3 of the Unfolding Transformation

Execution of all the steps of the unfolding transformation will produce a new DFG, unfolded by an amount specified by the unfolding factor (N). The three steps of the unfolding transformation were illustrated in Chapter 2, in Figure 2.13.

Due to its ability to create chromosomes that have excellent power implementation characteristics relative to the rest of the population, the unfolding transformation requires a special process of application to chromosomes. If left

unchecked the properties of the unfolding transformation could lead to the GA localising its search, very early in the search process, on the ‘unfolded design’ region of the solution space. This may prevent the GA from performing an efficient search of the larger solution space. While this would produce valid low power solutions, an unfolded design has the associated disadvantage of a large area overhead. To minimise this overhead the unfolding transformation is applied using the **Postponing Principle** [Huang94].

The idea behind the postponing principle is to incorporate some problem-specific VLSI knowledge into the transformation application process. This is achieved by attempting to order the application of transformations. However, as previously discussed, the optimum order of transformation application cannot be determined. Therefore, the postponing principle places a loose restriction on the order.

The principle achieves this restriction by reserving the application of certain transformations until a specified target has been reached. Such a target may be that a certain number of iterations have passed, using the current transformation set, during which no increase in solution quality has been produced. The postponed transformations are then applied to attempt to improve the solution. If successful, the initial transformation set is reapplied and the whole postponing process is executed again.

Within the context of the unfolding transformation, the principle postpones the application of the unfolding transformation, to first obtain an optimum solution with the other transformations. When a specified number of generations have passed with no increase in quality, unfolding is applied in an attempt to improve

this solution. If unfolding is successful, its application is again postponed while the other transformations attempt to improve the quality of the unfolded solutions.

The use of the postponing principle also has a beneficial effect on the speed of the GA. Unfolding creates large chromosomes, therefore unconstrained unfolding applied throughout the evolution would require considerable computation to process and evaluate these unfolded designs. By limiting the application of unfolding, the increased computation required to deal with unfolded designs is also limited. However, the postponing principle is applied in order to improve the synthesised designs, therefore this reduction in computation is not obtained by sacrificing the performance of the search process.

Another parameter used to control the application of the unfolding transformation is the 'Unfolding Factor'. As previously described, this specifies how much to 'unfold' the DFG. GALOPS uses a random weighted selection to choose from a range of unfolding factors, from 2 to 4. The weighted selection means that an unfolding factor of 2 is more likely than a factor of 3. These factors were chosen to limit the application of a large unfolding factor during a single generation. Gradual application of lower unfolding factors will create a DFG with large unfolding factors over a number of generations, rather than a single generation. This allows the GA to explore the possibilities of each unfolded DFG, instead of immediately generating huge DFGs with the associated area expense.

### ***5.4.6 Application of Mutation Operators***

The genetic mutation operators of GALOPS were introduced in sections 5.4.1 to 5.4.5. The operators constitute a **Transformation Library**, which can be accessed by the GA to optimise the candidate solutions. This section describes how

these operators are applied to the chromosomes to create new solutions, generation after generation.

At the start of a single generation the current population consists of a fixed number of solutions. The solutions are selected (using the Selection Procedure) for reproduction and modification. After modification, the chromosomes are stored in the new population. This process is repeated until the new population contains the required number of solutions. The older population is then deleted and the new population becomes the current population, starting a new generation of the synthesis process. In a 'generational' GA [Goldberg89, Whit89], of which GALOPS is a type, the size of the population remains constant from generation to generation. The evolutionary search consists of repeated iterations of the select-modify-store process, creating one generation after another. This process is summarised in Figure 5.22.

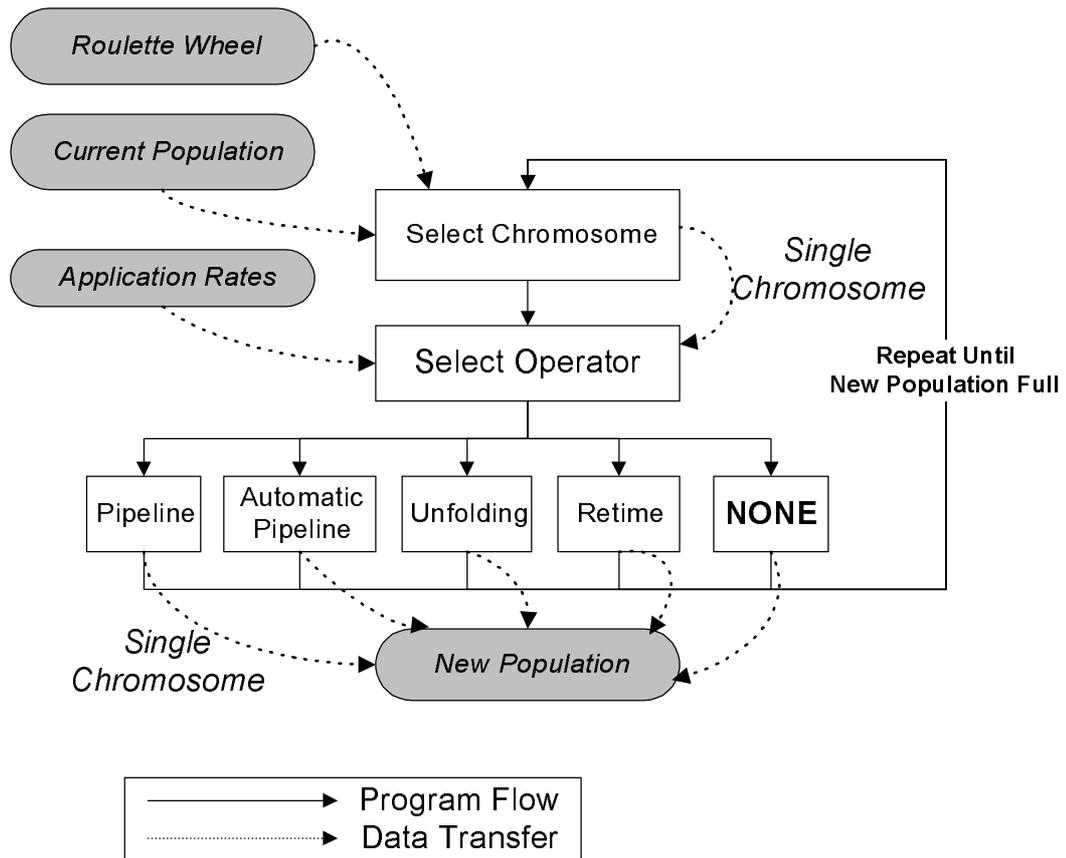


Figure 5.22 Application of Mutation Operators within GALOPS

An important process in this flowchart is the actual selection of the mutation operator. As the flowchart illustrates, the mutation operators are applied at predefined application rates. The ability to select application rates was implemented to provide a mechanism through which to analyse the effects of transformations. The required rate of application of each transformation may be dependent on each DFG, the interaction with other transformations, or other factors. The application rates can be used to analyse these factors.

The application rates are also useful for attempting to incorporate problem-specific knowledge into the synthesis process. For example, a known design heuristic is that pipelining creates large increases in quality, but has an associated latency penalty. However, retiming may require a large number of applications to

create a better solution, but it does not have the associated latency penalty. So, the application rates are set to take this into account by applying retiming at a much greater rate than pipelining; attempting to produce a low power design with minimum impact on latency.

The NONE box in the chart illustrates the fact that mutation is not applied to all solutions. In a standard GA, solutions are selected for mutation, crossover or simply reproduction (copied from one generation to the next).

### 5.5 Genetic Crossover Operators

The crossover operator, first discussed in Chapter 4, Section 4.2.3, is considered by some researchers to be a fundamental aspect of a GA [Holland92, Goldberg89, Beasley93]. Therefore, in addition to the development of mutation operators, problem-specific crossover operators are required for GALOPS.

The crossover operator attempts to find new, better solutions by combining the characteristics of current good solutions [Holland92]. Current good solutions are identified by the selection procedure before being passed to the mutation or crossover operators for genetic modification.

A simple implementation of a standard crossover operation would attempt to combine the characteristics of two parent chromosomes to produce two child chromosomes. The crossover operator selects a range of genes within each parent chromosome; these genes are then swapped to produce two new chromosomes with characteristics belonging to each parent. As with the mutation operation, the simple application of this process to DFGs creates problems. This is illustrated in Figure 5.23.

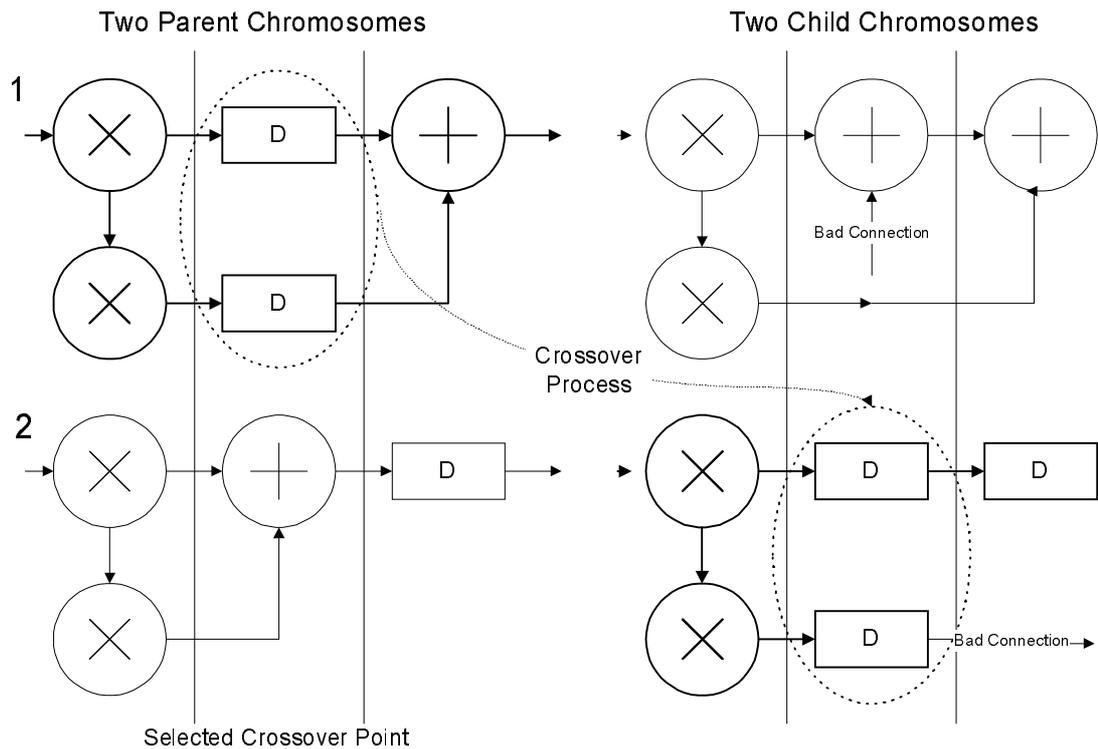


Figure 5.23 DFG Functionality Corruption due to the Crossover Process

This example demonstrates the crossover process for two parent chromosomes. The crossover point selects genes from each parent chromosome; the crossover process is highlighted in parent 1. In this parent, the two delay nodes are swapped with the single addition node of parent 2, to create child chromosome 2. As the example shows, this produces a chromosome with a number of problems. The output of one of the delay nodes is left open, resulting in a bad connection within the DFG. This contributes to the corrupt functionality, which is not only due to bad connections but also due to the incorrect signal processing function that the child chromosome now represents. Note that the original parent chromosomes have the same functionality as each other; parent 2 is a 'retimed' version of parent 1, two delay nodes have been moved from the inputs of the addition, to a single delay node on its output.

This example illustrates that crossover has the same problems of implementation as the genetic mutation operation, which required the development of problem specific genetic mutations. As with the mutation operator, the crossover operator is modified to produce new chromosomes with no change in DFG functionality. The modified crossover operation is a problem-specific genetic operator.

### *5.5.1 Problem-Specific Crossover Operator*

The fundamental aspect of crossover is that it attempts to combine the characteristics of solutions. Within GALOPS, both its desired function and the series of transformations that have been applied dictate the characteristics of a solution. Therefore, combining the characteristics, while maintaining the same functionality, is effectively combining the series of transformations that have been applied. The crossover system used within GALOPS is based on this concept.

To perform a crossover operation, Each chromosome is analysed to identify a transformation. The identification of transformations is made possible by assigning a specific label type to each node inserted by a specific transformation. Standard nodes, present in the original DFG, are typically labelled from  $e_1$  to  $e_L$ , where  $L$  is the number of nodes in the DFG. Each transformation uses a different prefix to label the node; for example, retimed delay nodes are labelled with 'r<sub>X</sub>', where  $X$  is the node number. The 'r' identifies that this delay node has been retimed to its current position in the DFG. The other transformations use similar methods of identification.

After identifying a transformation in a chromosome, the transformation is applied to the other chromosome, producing a chromosome with some of the

transformation characteristics of both chromosomes. An example of the first stage of this process, identifying the transformations in two parent chromosomes, is illustrated in Figure 5.24.

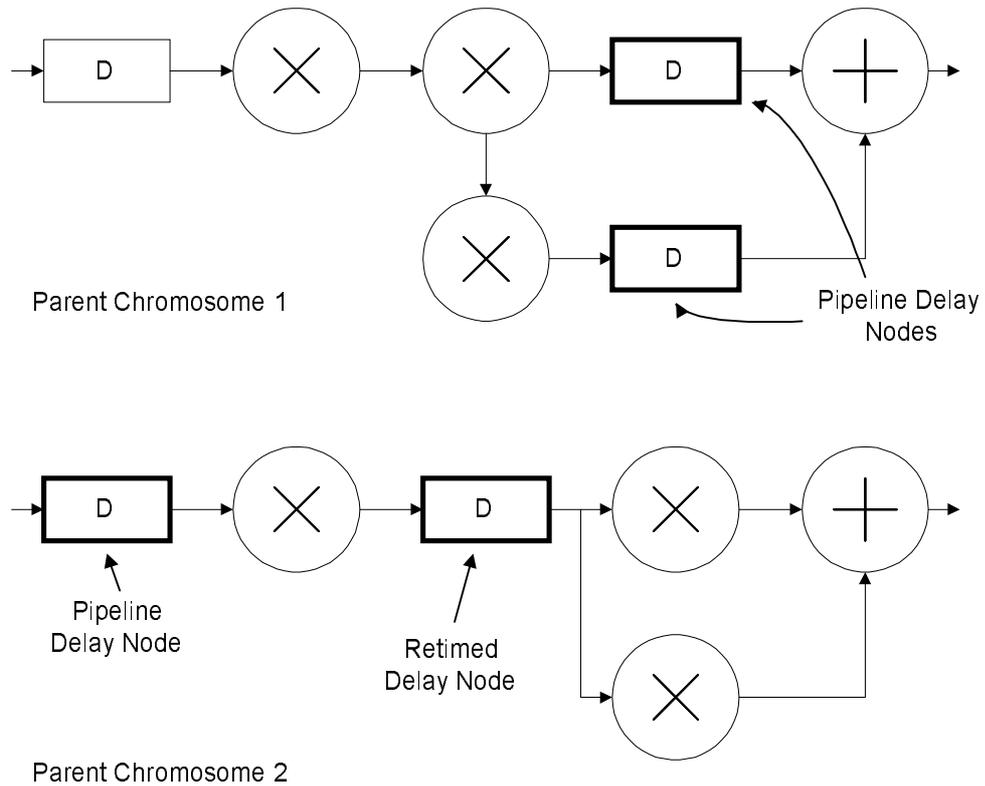


Figure 5.24 Identification of Transformations for Crossover Operation

In the example shown, a pipeline stage (with the relevant pipeline delay nodes) has been identified in parent 1. In parent 2, two transformations have been identified; a pipeline stage containing a single pipeline delay node and a retimed delay node. The crossover operation is executed on each parent in sequence.

The first operation is to apply the pipeline transformation of parent 1 to parent 2. Not only must the pipeline operation be applied correctly, it also has to be applied to the same location in parent 2. This is required to maintain consistency with the concept of a genetic crossover operation [Holland92]. This requires

identification of not only the type of transformation, but also its location in the original DFG (parent 1). When the location in parent 1 has been identified, the same location is searched for in parent 2. In some cases, this search may fail due to the application of transformations in parent 2 significantly altering the DFG structure. In this example, the location is easily identified and the pipeline transformation is applied at that location. Again, in some cases this operation may fail due to the pipeline constraints at that location in parent 2. The completion of the pipeline transformation crossover creates a new child chromosome, as shown in Figure 5.25.

Parent 2 has two crossover operations, therefore one is selected for application to parent 1. In this example, the retime transformation is applied, creating a new child chromosome, as shown in Figure 5.25.

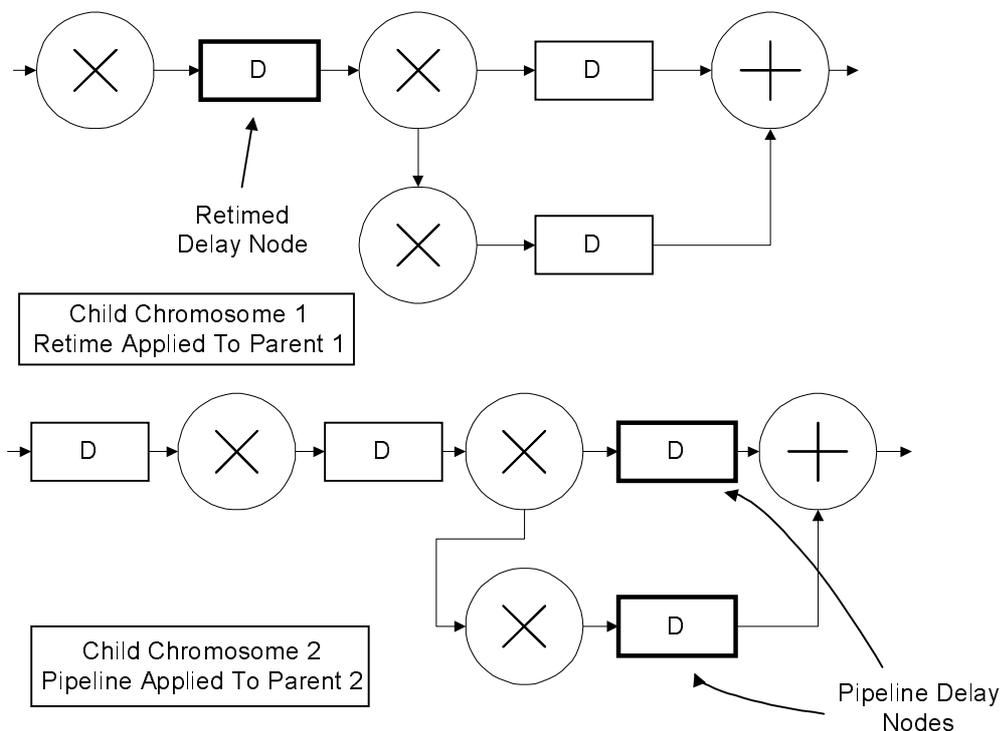


Figure 5.25 Effect of Crossover on Two Chromosomes

Parent 2 illustrates the fact that a chromosome may have had a number of transformations applied. If there are a collection of transformations to choose from, either a single transformation or a number of transformations can be selected for crossover. Selecting a single transformation for crossover is analogous to a standard two-point crossover mechanism with a localised selection of genes [Goldberg89, Beasley93]. GALOPS uses this two-point crossover method to identify and transfer a single transformation. Where a number of possible transformations exist, a single transformation is randomly selected for crossover.

For each of the possible transformations, a relevant crossover operation has to be implemented. The task of each crossover operator is to detect the transformation in one parent and attempt to apply it to another parent, ensuring that all application rules are obeyed. The next section describes the implementation of the crossover operator.

### *5.5.2 Implementation of Crossover Operator*

The algorithm for the crossover operator comprises both the selection of which operator to apply and the search for that operator in the chromosome. Once the desired crossover operator has been selected, the search performed and the crossover constraints satisfied, the transformational crossover is applied.

The mutation operator accesses its transformations from a core transformation library. The same library is used to perform the transformations within the crossover process. Therefore, the crossover operation consists of two distinct processes; identify the transformation then apply.

The pseudo-code for the crossover process is shown in Figure 5.26. This pseudo-code formalises the crossover procedures illustrated in Figure 5.24 and

Figure 5.25. The crossover algorithm processes two parent chromosomes, attempting to apply a transformation operator to parent 2 that was identified in parent 1. Therefore, this process will only create a single child chromosome, whereas crossover traditionally creates two children [Holland92, Goldberg89]. Applying a transformation to parent 1, which was identified in parent 2, creates the second child. This is achieved by re-applying the crossover transformation algorithm, but on this second iteration, the order in which the parent chromosomes are passed to it is switched. In effect, during the second iteration of the complete crossover algorithm *parent1* in the pseudo-code is actually parent 2, and vice-versa.

### Algorithm **Crossover**

```
Passed parent1 chromosome
Passed parent2 chromosome
Returns single child chromosome (parent 1 transform applied to parent 2)

IF parent1 IS THE SAME as parent2
    Crossover unsuccessful

REPEAT UNTIL crossover successful OR ALL transformations attempted
    Select crossover operator:
        Unfolding
        Retiming
        Pipelining
        Back Retiming
        Automatic Pipelining

        REPEAT UNTIL all selected crossover operators in parent1 attempted
            Attempt to detect selected crossover operator in parent1
            IF detected
                Find relevant location in parent2
                Check constraints on applying transformation
                IF constraints satisfied
                    Apply transformation to parent2
                    Child chromosome created
                    Crossover successful
                ELSE
                    Select new crossover operator in parent1
            END REPEAT
        END REPEAT
    END REPEAT

IF crossover NOT successful
    Copy parent1 chromosome to next generation (single child)
```

### End Algorithm **Crossover**

Figure 5.26 Pseudo-Code for Crossover Transformation Operation

On entrance to the crossover algorithm, the fitness values of the two parents are compared. Applying crossover to parents with the same fitness is prevented from occurring as the 1:1 allocation technique ensures that chromosomes with the same fitness have the same genetic structure.

The choice of which crossover operator to apply is based on specified application rates. For example, GALOPS may attempt to perform a retime crossover operation on the solutions selected for crossover. However, the search procedure may detect that a retime crossover is not possible. In this case, GALOPS selects another crossover transformation to be applied, again based on the specified application rates. This process is repeated until all of the transformations have been attempted. In certain cases, a transformational-based crossover may not be possible between two parents. In this instance GALOPS simply copies the parent to the next generation, producing a child chromosome with no modifications.

Each specific crossover operator may yield a number of possible locations in parent 1. For example, parent 1 may have a number of pipeline stages. In this case, each of the pipeline stages is selected for crossover until either one is successful or they have all been attempted.

The application of the unfolding crossover operator, as with the unfolding mutation operator, requires special consideration. The standard crossover transformation requires identification of transformations within a chromosome. In the case of the unfolding transformation, all of the nodes within a chromosome are affected, therefore no search is required to identify its application. It is simply identified by determining the current unfolding factor of that chromosome (non-unfolded chromosomes are considered to have an unfolding factor of 1). If unfolding is selected, the unfolding factors of both parents are determined. If the

unfolding factor of parent 1 is greater than parent 2, then unfolding is applied to parent 2, with an unfolding factor chosen to create a child with the same final unfolding factor as parent 1.

The entire crossover operator, as with each of the specific transformational based mutation operators, is applied to the population of chromosomes at a specific rate, labelled the crossover rate. This GA parameter, as with mutation application rates, is difficult to define for a wide range of solutions. The required crossover rate is strongly problem dependent, therefore it is most commonly defined after experimental analysis [Goldberg91, Hanc95]. Some attempt has been made to determine 'good' GA parameters such as the application rate of crossover, but the research has primarily been restricted to certain classes of problems with bit-string representation [DeJong].

The crossover operator and the mutation operator comprise the genetic mutation operators of the GALOPS system.

### **5.6 Fitness Evaluation**

As discussed in section 4.2.2, a GA requires a means of measuring the quality of a particular solution, in particular, the quality of a solution relevant to other solutions. Within the context of a synthesis system for low power operation the primary measure of quality is how well a particular design meets specified power consumption requirements.

As well as power consumption, other VLSI implementation parameters are of interest to help determine how the power reduction has impacted on items such as speed, area, complexity, etc. Therefore, the fitness evaluation section of a VLSI

synthesis system must be capable of calculating practical VLSI implementation parameters.

It is recognised that, at the high-level of abstraction (which GALOPS uses), accurate power estimation is an extremely complex process (as discussed in section 3.5). Error rates of up to 40% are considered acceptable, providing that the values provide a good indication of the relative quality of a range of solutions [Roy95, Chuang98]. However, the computation of relative power consumption, even with significant error tolerances, is still a complex process.

This ability to trade-off absolute for relative accuracy suits a standard property of a GA. A GA compares solutions for their relative merits when selecting chromosomes for reproduction; therefore, it is more useful to know the relative power consumption of two designs (e.g. which solution consumes less power) than it is to know the actual power consumption of each design. Therefore, a GA synthesis methodology has apparent benefits when combined with the problem of high-level power estimation.

Part of the problem of determining an accurate VLSI implementation parameter is the length of time such calculations take. Very accurate estimations of power dissipation are possible if very detailed information is available and time is not a limitation. However, a prime requirement for a GA fitness evaluation function is rapid execution speed, as a typical GA uses fitness evaluation many times during the evolutionary search process. The fitness evaluation function is very often the limiting factor in the execution time of a GA.

GALOPS therefore requires a rapid evaluation function, that estimates power consumption while tracking other VLSI implementation parameters, with emphasis on relative rather than absolute accuracy. While power estimation is a

complex and time consuming process, the fitness evaluation of a GA is required to return a value relatively quickly; this places the conflicting requirements of accuracy and speed on the fitness function. Hence, the fitness evaluation section of GALOPS comprises a high-level power estimation module that attempts to determine power quickly.

The main tasks in power estimation were described in chapter 3. To summarise, the equation for energy per computation of a signal processing algorithm implemented on a CMOS device is shown in equation 5.2.

$$\text{Energy per computation} = V_{DD}^2 * C^* \quad (5.2)$$

This equation shows that the main constituents of power are the supply voltage ( $V_{DD}$ ) and the switched capacitance ( $C^*$ ).

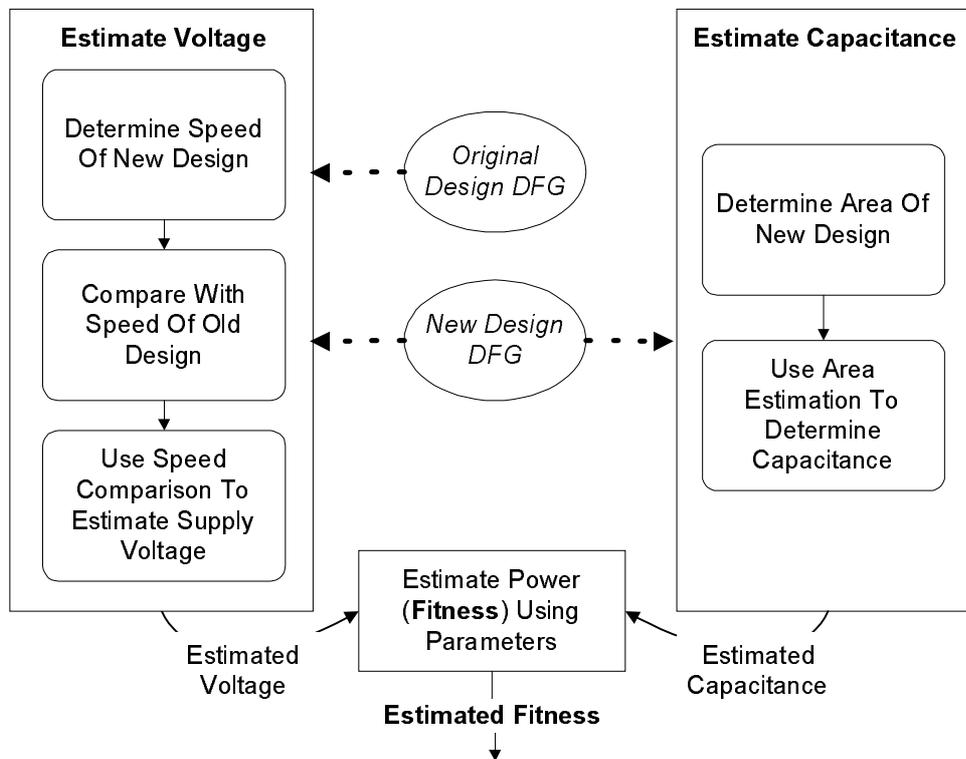


Figure 5.27 Fitness Evaluation of New Design

Figure 5.27 presents an overview of the fitness evaluation process. The figure shows that supply voltage and capacitance can be estimated separately, therefore the fitness evaluation process consists of two distinct sections. Also shown in this figure are the relevant steps required to estimate each parameter, as described in sections 5.6.1 and 5.6.2.

### *5.6.1 Supply Voltage Estimation*

As described in section 2.4, the supply voltage is used to reduce the power consumption by trading speed for supply voltage, as a reduction in supply voltage decreases the speed of a CMOS device. The original designs are assumed to have a supply voltage of 5V. Any increases in speed (due to the high-level transformations) can be traded for a reduction in the 5V supply voltage, and hence a reduction in power. The precise reduction in supply voltage for a specified speed can be calculated. Therefore, the required power supply of a particular chromosome can be calculated by comparing its operating speed to that of the original. The curve for speed against delay (Figure 2.4 in section 2.2) can be used to extrapolate the power supply of a specified DFG.

For example, if the throughput speed of the original design can be doubled it can tolerate a two-fold increase in the delay of the CMOS device. A reduction in voltage would return the throughput of the chromosome to its original. The increase in the delay of the device is achieved by reducing the supply voltage to 2.9V (determined from the delay-voltage graph). Therefore, a DFG with a maximum speed twice that of the original will operate at the same speed of the original if its supply voltage is 2.9V. This example shows that estimation of the speed of a DFG (in comparison to the speed of the original) will yield its required supply voltage.

The maximum throughput speed of a DFG is bounded by the length of its critical path (CP). Therefore, the calculation of speed requires estimation of the length of the CP of the DFG. The CP of a DFG is defined as the longest path between non-computational nodes i.e. delay nodes and input/output nodes. Therefore, the CP is determined by finding the longest path of computational nodes (e.g. adders, multipliers) within the DFG. The prototype version of GALOPS achieves this using an algorithm to determine the length of all computational paths, the longest found is then defined as the CP. A CP algorithm using a recursive structure, to explore all possible paths within the DFG, is illustrated in Figure 5.28.

```
Algorithm Search Path  
  Passed current_node  
  
  IF current_node is delay or input  
    EXIT algorithm (STOP search down this path)  
  
  For each input_net of the current_node  
    IF input_net in path_list  
      STOP search down this path (input_net is in a loop)  
    ELSE  
      Store input_net in path_list  
      Set current_node to input_net  
      Call Search Path (recursive call)  
End Algorithm Search Path
```

Figure 5.28 Pseudo-Code for Recursive Path Search Algorithm

The algorithm is executed for each node in the DFG, exploring all possible paths from that node. As each node in a path is traversed, it is added to a path list that contains all the nodes in that path. When that path has been fully explored, it is compared with the path currently stored as the CP. If it is a longer path, then it is stored as the CP. Although this method does correctly identify the CP in a DFG, it is a brute-force method that does not consider some properties of the DFG.

The definition of non-computational elements as ‘path-termination’ nodes is one factor considered within a refined CP algorithm. This is illustrated in Figure 5.29. The example shows that the start of a CP will always be the fanin element of a delay or output node (‘path-termination’ nodes). In this example, the CP comprises nodes 4, 2 and 1. Searching from node 2 would yield a path containing nodes 2 and 1, a subset of path 4,2,1. To prevent subset paths from being explored (unnecessary computation) the search for the critical path is only executed from start nodes, those which are the fanin nodes of termination nodes. This significantly reduces the amount of exploration required by the algorithm.

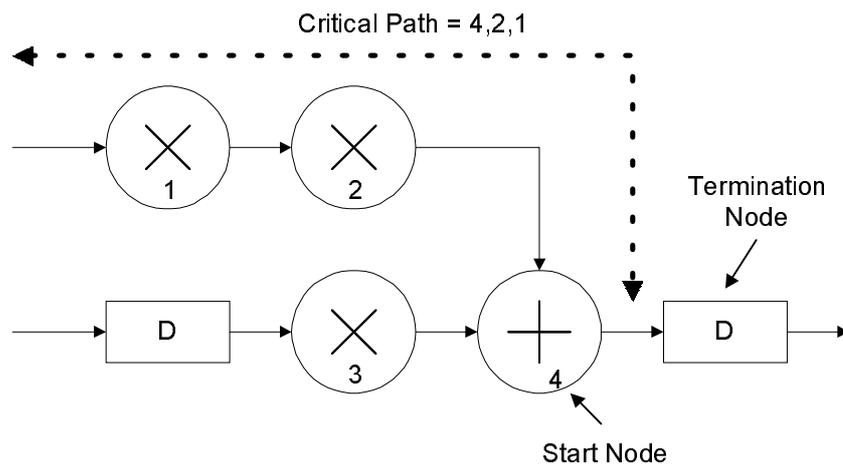


Figure 5.29 Start Point Of Critical Path

Another problem with the recursive algorithm is that it requires a large amount of recursion to explore paths of even moderate lengths e.g. a few hundred elements. Tests on this prototype path-exploration algorithm demonstrated that such heavily recursive algorithms are difficult to analyse and cause program failures for large paths. Therefore, the algorithm was replaced with a non-recursive version.

The non-recursive CP algorithm creates a path list for each possible path within the DFG, all paths are considered to start at ‘path-termination’ nodes. Figure

5.30 illustrates the operation of this algorithm with a simple example, exploring the path from a single start node.

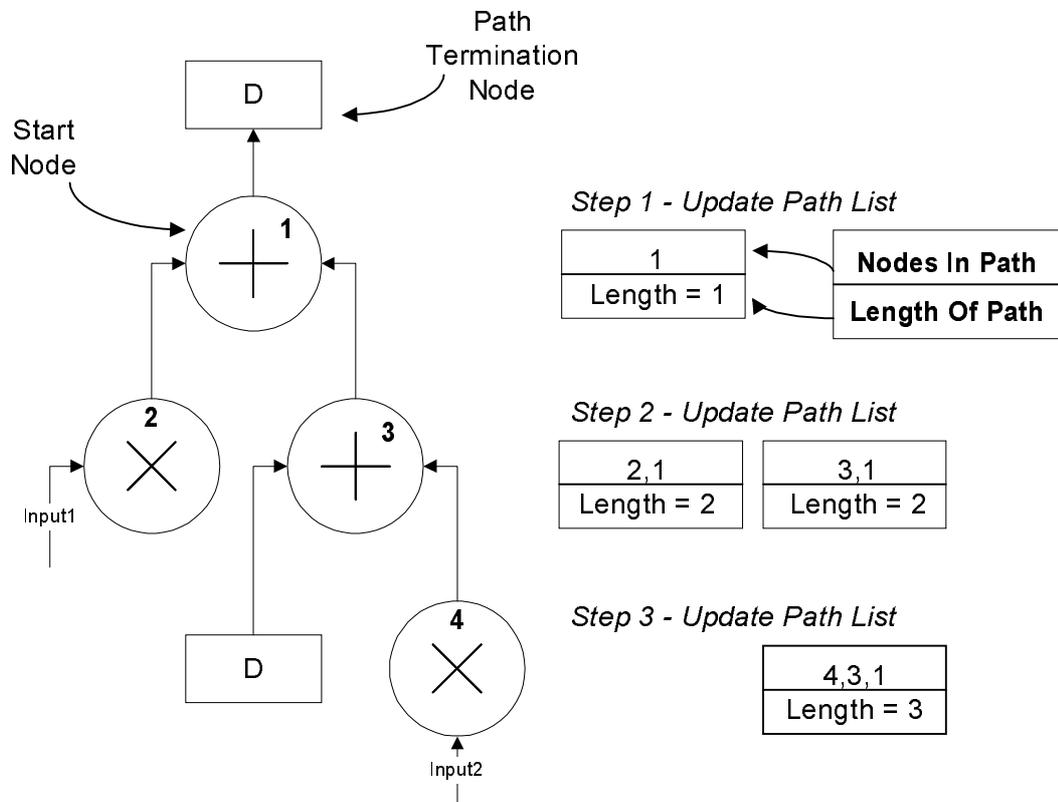


Figure 5.30 Example of Determination of Critical Path

The first step of the algorithm is the creation of the path list for the initial node. The path list, as shown in the diagram, contains a list of all the nodes in the path and an integer representing how many nodes are in the path. The algorithm proceeds by moving onto the fanin nodes of the current node. For each of these nodes, a copy of the current path list is made, before it is updated with the information for those nodes. There are now two path lists for the current path under exploration. This step is repeated again by moving onto the fanin nodes of the current nodes, nodes 2 and 3. In the case of node 2 (the multiplier) its fanin node is a primary input; therefore, the path is terminated. A complete path, nodes 1 and 2, is

now stored within a path list. The length of this path is compared with the length of the CP found so far. If the new path has a greater length then it is stored as the CP. Regardless of the result of this comparison, the terminated path list is deleted.

In the case of node 3, it has two fanin nodes. One is a delay node, therefore the search is terminated for that fanin node; the other fanin node is a multiplier, so the algorithm proceeds onto that node, creating an updated path list.

The fanin node of node 4 is a primary input, therefore the search is terminated at that point, and the path list is compared with the current longest and then deleted. As all path lists have been deleted, the search for all paths that start at node 1 has been completed. This example shows a small section of a DFG, with a single start node. The algorithm is repeated for all start nodes within the DFG. To determine the CP of the DFG.

Determination of the critical path does not produce an actual speed rating (for example in Hz) for the DFG. It implies the maximum operating speed, dependent on how fast data can be processed through the critical path. This is because the maximum speed is also governed by the speed of a single node. For example, a CP length of 5 nodes, each with a delay time of 1 millisecond, could be operated at a maximum speed of  $1/(5 \times 10^{-3})$  Hz (200 Hz). Therefore, determination of absolute speed requires specific knowledge of the speed and hence the implementation style of each node. Rather than limit the use of high-level tools to specific VLSI implementations, the delay time of each node is expressed in non-specific terms. GALOPS uses a unity-delay model [Micheli94], where each node is considered to have a delay of a single processing cycle. This widely used model [Iman96] has the benefits of reduced complexity, as each node can be considered to contribute the same delay to the overall CP length delay.

As using a unity-delay model removes the need for actual speed values, the speed of each DFG in GALOPS is expressed in relation to the speed of the original DFG. For example, if the original DFG has a CP of 2, and a new DFG has a CP of 1, the new DFG is considered to be twice as fast as the original. This ‘speed-ratio’ is used to calculate the required supply voltage for the new DFG, to force it to operate at the same speed as the original. The supply voltage is calculated using a piecewise linear model of the curve in figure 2.4 (section 2.2). The speed ratio is passed to this model and a required supply voltage is returned. Figure 5.31 presents the pseudo-code for the calculation of the supply voltage of a DFG.

```
Algorithm Calculate Supply Voltage  
    Passed critical_path_length of original DFG  
  
    Determine critical_path_length of new DFG  
    Speed Ratio equals original_CP/new_CP  
  
    Use piecewise linear model of VDD/Delay curve:  
        Speed Ratio gives displacement on y-axis  
        Model of curve returns displacement on x-axis  
        Returns supply_voltage  
    End use model  
  
End Algorithm Calculate Supply Voltage
```

Figure 5.31 Pseudo-Code for Supply Voltage Calculation

The use of an unfolding transformation, to gain power reductions through increased parallelism, increases the complexity of the supply voltage estimation process. As the unfolded DFG is a parallel implementation, that can handle N input samples in parallel, its effective critical path length is 1/N times its actual critical path length [Arslan95]. For example, consider an original DFG with a critical path length of 5 steps to complete one cycle. Unfolding with an unfolding parameter of 2 results in a DFG that process two samples in parallel, but has a critical path length of 10 steps. Therefore, 10 steps of either DFG will result in 2 samples being

processed; the original DFG will execute two cycles, the unfolded DFG one cycle. Therefore, it is apparent that the two DFGs have the same throughput. The unfolded DFG is effectively processing one sample per 5 steps of the critical path, therefore its effective critical path length (when compared with the original DFG) is 5 steps. This is equal to the actual critical path length divided by the unfolding parameter of the unfolding DFG.

This consideration of parallelism is incorporated within the CP algorithm. After determining the actual length of the CP, the unfolding parameter of each DFG is checked and if necessary an effective CP length is also calculated. The effective length is used to determine the comparative speed and hence the supply voltage of the unfolded DFG.

### 5.6.2 Capacitance Estimation

As described in chapter 3, switched capacitance estimation is a complex process that requires determination of specific VLSI implementation data for a given algorithm. Many of the parameters required to calculate the switched capacitance, such as area, floorplan layout, transistor sizes, etc., are not finalised at the high-level. To determine absolute values for these parameters, a full synthesis procedure, down to silicon layout level, would be required for each DFG and, hence, for each fitness evaluation. This would be extremely prohibitive in terms of computation time; a problem widely recognised in high-level power synthesis. A solution to this problem is to estimate this data from the identifiable properties of the DFG [Sheng92, Kumar95]. A range of techniques exist to estimate low level VLSI implementation parameters, though few operate at the high-level of

abstraction utilised by GALOPS (as described in chapter 3). This section outlines the capacitance estimation technique used in the prototype version of GALOPS.

The GALOPS capacitance estimation technique is based on estimating the VLSI implementation capacitance of the DFG's data path. The estimation and synthesis of memory and control units are not considered. Within the context of the data-intensive signal processing algorithms discussed in this work, the data path contributes the greatest portion of the total area and power dissipated [Chaud96].

The problem of estimating switching capacitance was discussed in greater detail in chapter 3. The switched capacitance of a VLSI device is a combination of both the physical capacitance and the switching activity. The physical capacitance is a characteristic of the actual VLSI implementation, whereas the switching activity is dependent on the desired signal processing function of the device (the input data). This dependence of switching activity on device function makes it a difficult parameter to estimate. Restricting the signal processing application to a particular function during the synthesis process can limit the flexibility of the synthesis tool.

The prototype version of GALOPS considers the physical capacitance rather than the switched capacitance. The physical capacitance can be considered as a worst case switched capacitance estimation, where each physical resource is switched all of the time. This limits the accuracy of the estimation, but it does simplify and hence increase the computation speed of the power estimation process; this is a similar approach to the PFA technique introduced in chapter 3.

The capacitance estimation technique in GALOPS uses a pre-characterised library of components, of the type required to implement signal-processing functions. Each component is characterised for area and capacitance. This is accomplished by creating a library of VLSI device components, created with the

SOLO1400 semi-custom VLSI synthesis tool. SOLO1400 was used to construct a range of components, a full layout synthesis of each component yielded physical capacitance and area information listed in Table 5.1 [Puck94, Wilk92].

<b>Component</b>	<b>Area (mm<sup>2</sup>)</b>	<b>Capacitance (pF)</b>
<b>8*8 Bit Array Multiplier</b>	1.8625	0.23168
<b>8*8 Bit Ripple Adder</b>	0.1554	0.02467
<b>8 Bit Register</b>	0.0462	0.00707

Table 5.1 Architectural Level Components

The choice of components to be incorporated in the component library was motivated by ease of design and demonstration of the capabilities of a GA based synthesis system that utilised high-level transformations. A wider choice of component types could be incorporated into the component library, increasing the flexibility of the synthesis tool. For example, the components were all designed for 8-bit word lengths; however, additional components with a range of word sizes could be easily characterised and incorporated into the component library.

With the parameters for each component of the data path determined, the task of physical capacitance estimation is one of determining which components are required to implement the data-path section of the DFG. The prototype version of GALOPS uses a 1:1 allocation technique where a single execution unit on the VLSI device executes each operation in the DFG. For example, a DFG with 3 additions and 2 multiplications would be implemented using 3 adders and 2 multipliers, together with the required registers and interconnections.

Application of the 1:1 allocation technique, together with data from the library of execution units, is used to determine the area of the data-path component of the VLSI device.

The switched capacitance of the functional units, in terms of the power consumption, is largely independent of the allocation technique used. For example, 10 multiplications on 1 multiplier, or 1 multiplication on each of 10 multipliers, will consume the same amount of power. Therefore, the 1:1 allocation technique reduces the capacitance estimation error introduced by only considering the physical capacitance. Using the 1:1 allocation technique, the capacitance of the data path can be extrapolated directly from the DFG.

As with the voltage estimation, the unfolded DFGs require special consideration when estimating their capacitance. An unfolded DFG creates a parallel and hence larger implementation of the original DFG, capable of processing a number of input samples in parallel. This capability of handling samples in parallel results in the DFG operating at a lower sampling rate (as it handles a number of samples) but still processing samples at the same throughput rate. Capacitance is increased by a factor of  $N$  but the speed of switching is reduced to  $1/N$ . Therefore, the switched capacitance remains the same. Therefore, the effective capacitance of the data path component of an  $N$  unfolded DFG is  $1/N$  times its estimated physical capacitance for a constant throughput rate [Chan95]. This consideration of parallelism is incorporated into the algorithms for estimation of capacitance.

As described previously, the execution unit library and the DFG can be used to determine the capacitance and area of the execution units. However, a significant portion of the total capacitance is due to interconnect capacitance. In large designs,

this capacitance begins to dominate the overall capacitance [Chan95]. This capacitance is determined using a statistically derived model shown in Figure 5.32 [Chan95].

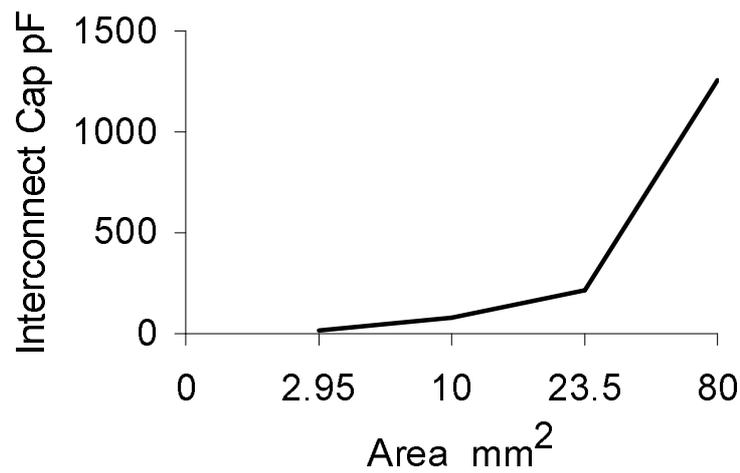


Figure 5.32 Interconnect Capacitance Model

This curve was characterised as a piecewise linear model in [Chan95] with four distinct points. The model returns a value for interconnect capacitance estimation from the estimated area of the data-path. The capacitance estimation procedure is summarised in Figure 5.33.

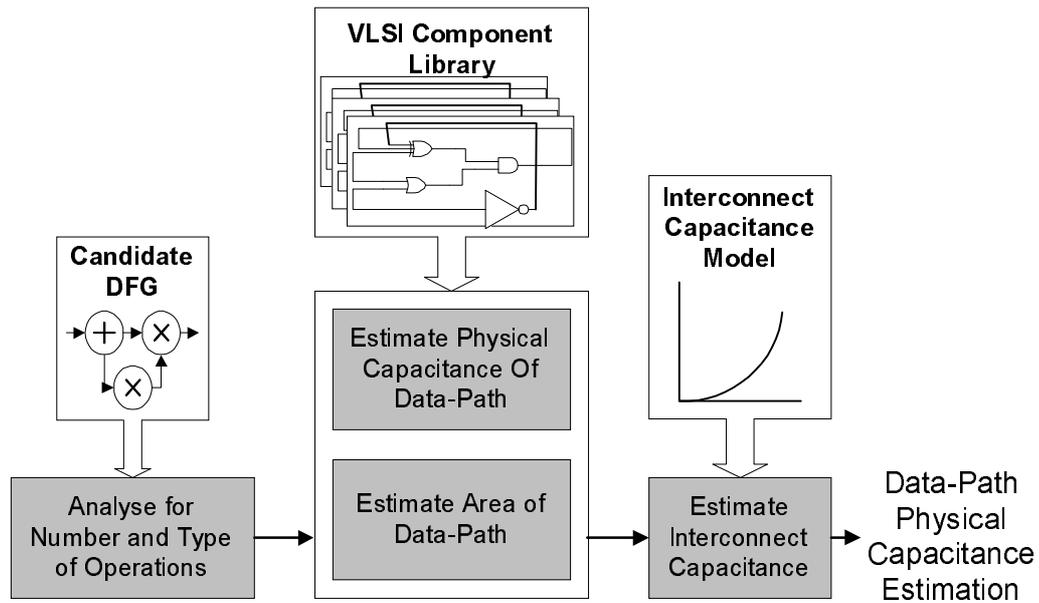


Figure 5.33 Capacitance Estimation Process

The process illustrated in Figure 5.33 will yield capacitance estimation for a candidate DFG. When combined with the voltage estimation, produced using the techniques of section 5.6.1, it will produce an estimate of power for the candidate solution.

### 5.6.3 Power Estimation Used as Fitness

The previous sections described the process of estimating the power consumption of a DFG. The fitness of each solution is obtained by scaling its estimated power consumption in relation to the estimated power of the original DFG, as illustrated in equation 5.3.

$$Fitness = \frac{\text{Power Of Original Design}}{\text{Power Of Current Design}} \quad (5.3)$$

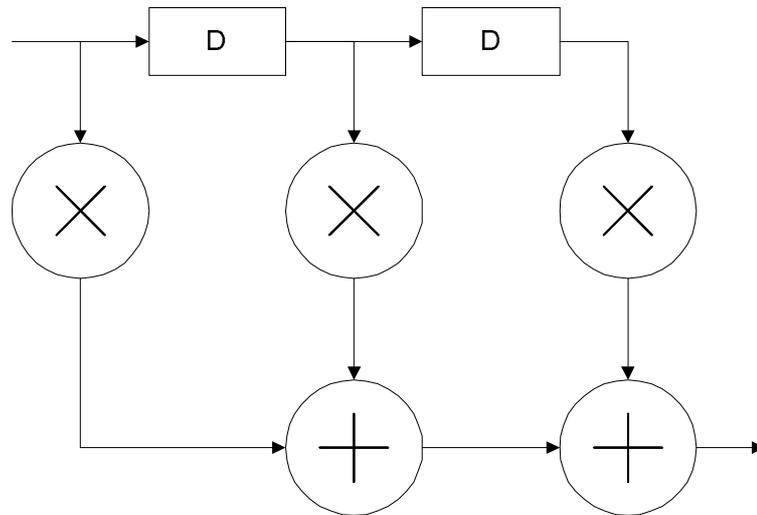
Using this equation, a design with power consumption lower than the original will have a fitness value higher than 1. In addition to fitness scaling, certain boundary conditions are placed on the estimated parameters. This is necessary because the genetic synthesis procedure may produce VLSI designs with a desirable power reduction but impractical implementation requirements.

A lower bound is placed on the required supply voltage, as designs with large speed increases may require supply voltages close to zero. Even ignoring the obvious problems of noise at such low levels, such voltages may be well below practical threshold voltages. The components used in the VLSI library are considered to operate with a threshold voltage of 1V, therefore any design with a supply voltage equal to or less than this is an impractical design. Such designs are assigned a very low fitness value to limit their chances of reproduction.

Another constraint placed on the designs is their implementation size. The interconnect capacitance model covers an area range from  $2.95\text{mm}^2$  to  $80\text{mm}^2$ . Any designs outside this range are also assigned a low fitness.

### 5.7 Benchmark Designs

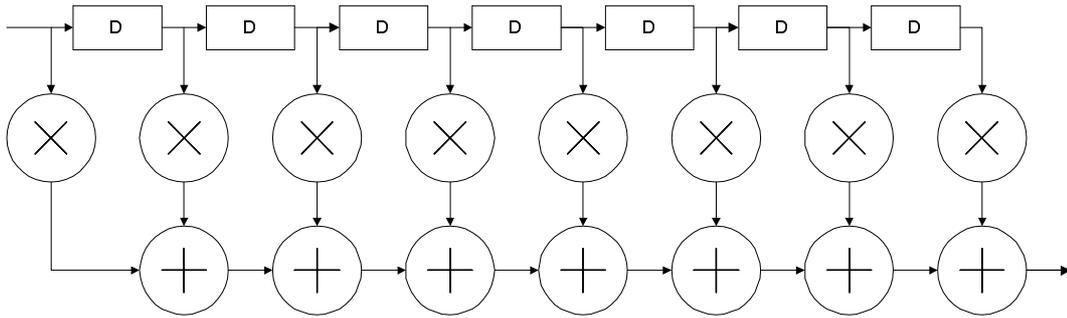
To illustrate the effectiveness of the GA and the algorithms presented, the performance of the prototype version of GALOPS is illustrated with a set of benchmark example designs.

5.7.1 3<sup>rd</sup> Order Finite Impulse Response Filter (FIR3)Figure 5.34 3<sup>rd</sup> Order FIR Filter

This filter is an example of a non-recursive signal-processing operation, as it has no feedback loops or recursive paths. The DFG for this filter is shown in Figure 5.34. This is a relatively simple filter with 3 multiplications and 2 additions. The actual DFG used is a direct form representation of the signal-processing algorithm.

5.7.2 8<sup>th</sup> Order Finite Impulse Response Filter (FIR8)

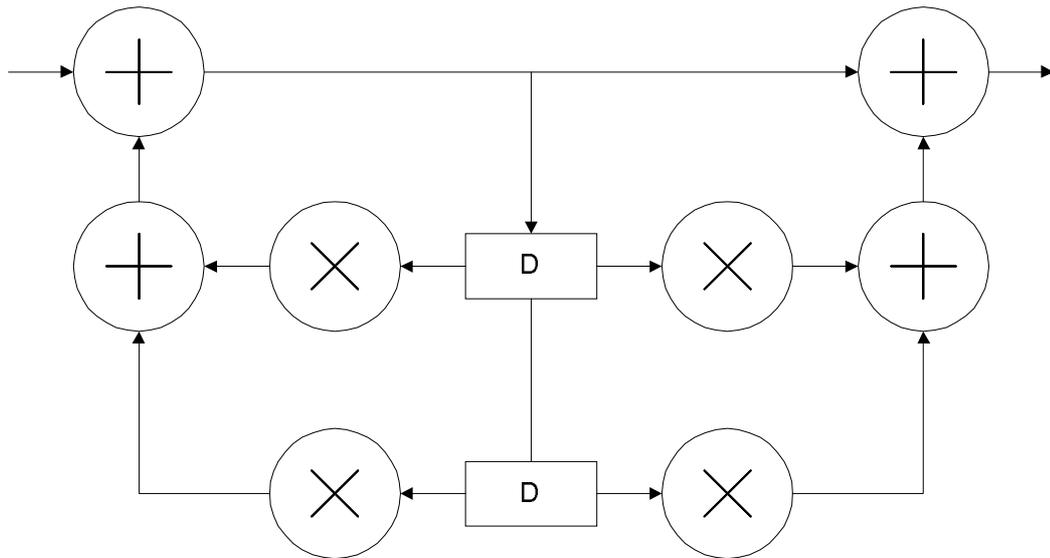
This is a more complex implementation of an FIR filter, with more operations and connecting nets resulting in a greater variety of possible options for optimisation. The DFG, shown in Figure 5.35, is in direct form.

Figure 5.35 8<sup>th</sup> Order FIR Filter

As the figure illustrates, this 8<sup>th</sup> order filter is a larger version of the 3<sup>rd</sup> order FIR filter comprising of 8 multiplications and 7 additions. This larger DFG can be used as simple illustration of the complexity of the optimisation process. For example, this filter has 14 possible pipeline locations if identified using the critical path technique. Each pipeline operation would create a new DFG with a different set of possible pipeline locations; added to this are all the possible retime, automatic retime and unfolding transformations.

### 5.7.3 2<sup>nd</sup> Order Lattice Filter (LAT2)

This is an example of a recursive filter operation as it contains a combination of feed-forward and feedback paths. As discussed previously, the feedback loops found in signal-processing applications affect the application of the transformations; recursive applications are more difficult to optimise as they place more restrictions on the optimisation steps that can be applied. The DFG for this algorithm is shown in Figure 5.36. This filter operation consists of 4 multiplications and 4 addition operations, again presented in direct form.

Figure 5.36 2<sup>nd</sup> Order Lattice Filter

#### 5.7.4 8<sup>th</sup> Order Avenhaus Filter

This large filter, shown in Figure 5.37, consists of 31 operations (16 multiplications and 15 additions). The large number of data-nets and nodes provides a wide range of possible transformation optimisations. For example, the delay nodes present 23 potential retiming mutations, each of which would create a new DFG with an associated number of potential retiming operations. Again, this illustrates the complexity of the synthesis process as each of these DFGs could in turn produce a number of DFGs. Over a number of transformation applications, the number of DFGs with distinct characteristics grows exponentially; hence the difficulty in identifying a set of transformations (and the correct order of application) that would yield an optimum design.

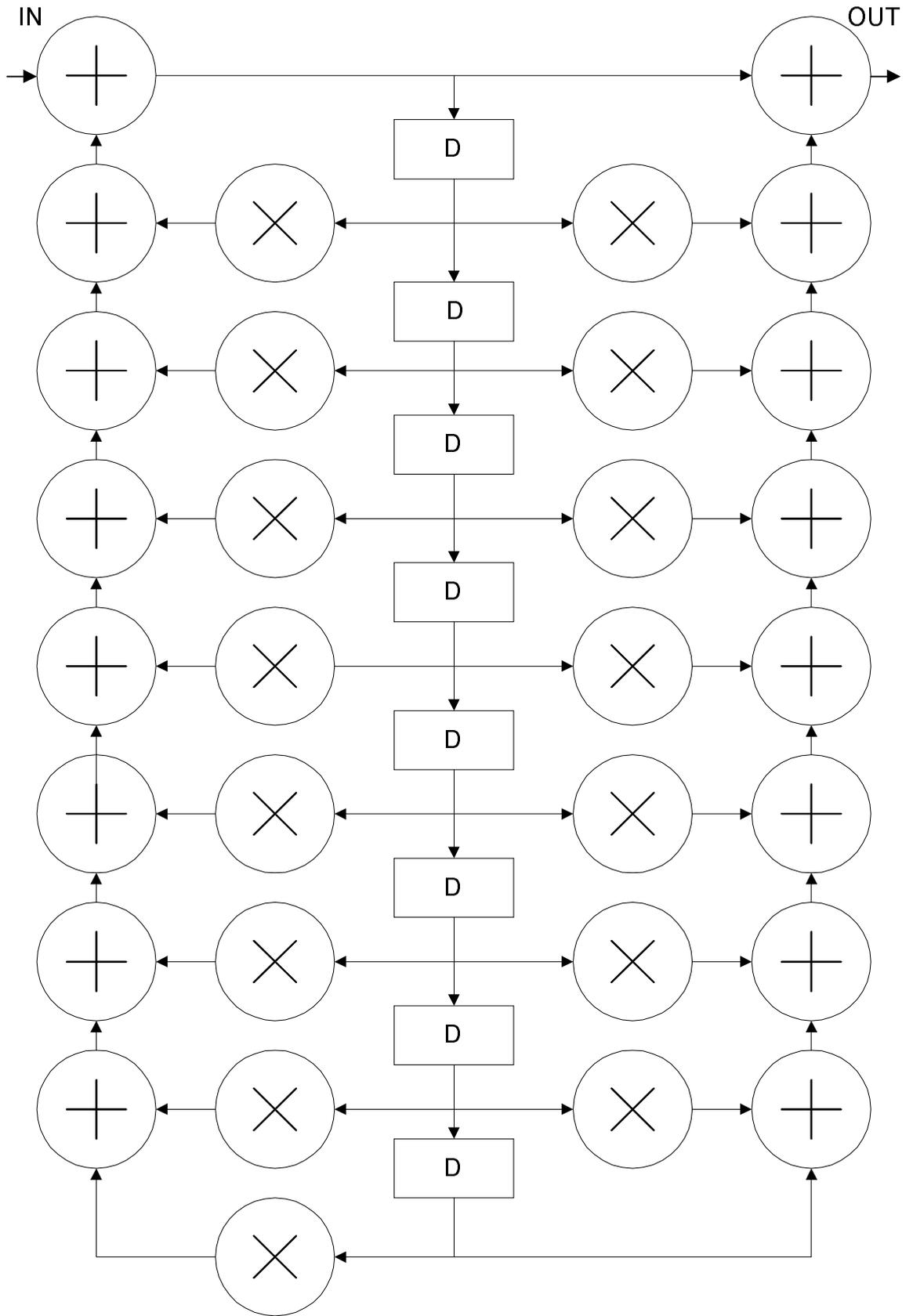


Figure 5.37 8<sup>th</sup> Order Avenhaus Filter – Direct Form Representation (AVEN8DI)

The filter of Figure 5.37 is one possible representation of an 8<sup>th</sup> order Avenhaus filter, a direct-form. Signal-processing algorithms can be represented in a range of formats, each producing the same function but with different operation characteristics. The 8<sup>th</sup> order Avenhaus filter of Figure 5.38 is presented in parallel-form, a distinct DFG with different implementation characteristics, such as word-length requirements and robustness. The direct and parallel forms of the Avenhaus filter are obtainable by applying a specified set of transformations to one, creating the other. The specific transformations required to switch between the two implementations have a significant effect on the numerical stability and word-length requirements of each design. The transformation set used in the GALOPS synthesis system has been chosen so as not to affect the word-length requirements.

This different representation of the signal-processing algorithm requires a distinct set of transformation mutations for optimisation, different to those of the direct-form representation. Therefore, this example also illustrates that the chosen signal-processing representation can have an affect on the optimisation process and consequently the power reduction obtained. This 'algorithm selection' phase is another level at which power can be optimised as discussed in chapter 2.

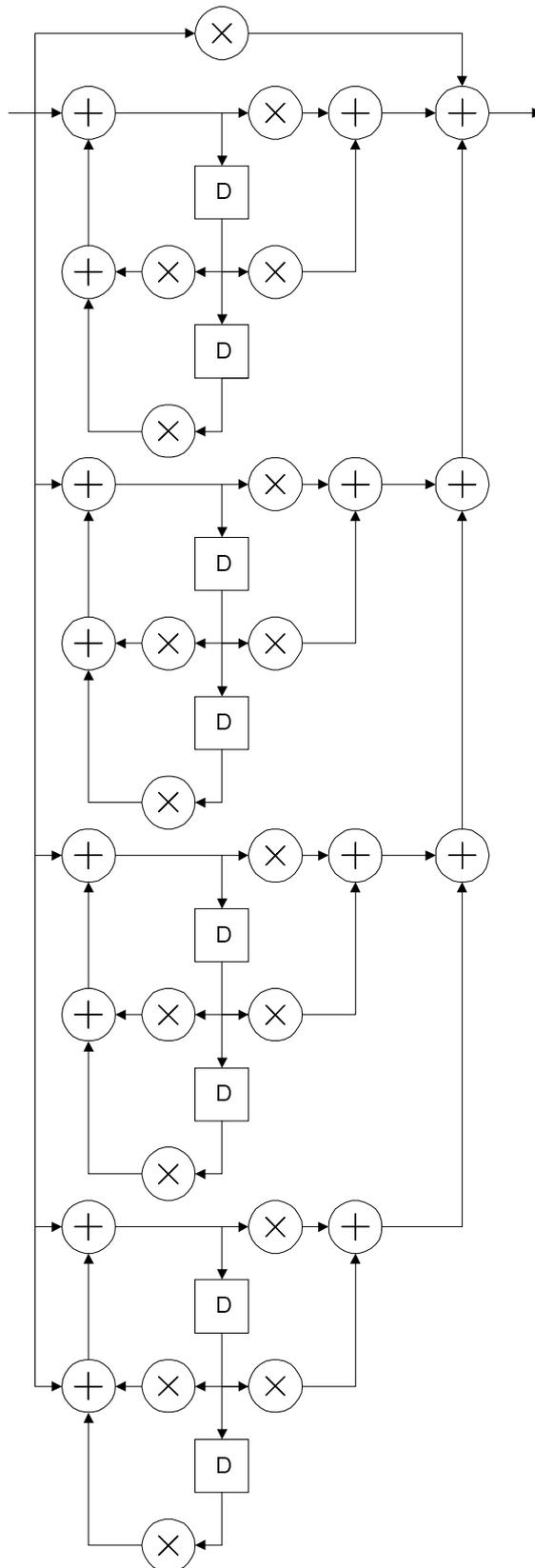


Figure 5.38 8<sup>th</sup> Order Avenhaus Filter – Parallel Form

### 5.8 Genetic Algorithm Operating Parameters

The GALOPS system has a set of genetic operators that are applied to the population at specified rates, as described in sections 5.4 and 5. The rate at which the genetic operations are applied is very much problem dependent and can have a significant impact on the performance of the algorithm [Davis91, Goldberg89]. If applied at too high or too low a rate it can lead to the generation of poor solutions and also result in very large numbers of generations required for optimisation. Therefore, selection of the application rates is an important process in designing the GA.

The prototype system has 5 mutation operators and a single crossover operation, each of which is assigned an individual application rate. The crossover rate is fixed at 20% to allow for a large amount of crossover to be applied while still allowing the 5 genetic mutations to be applied at significant rates (the total application rate can not exceed 100%). In contrast to a typical GA, the mutation operations of GALOPS perform a large amount of the exploration; therefore crossover is not applied at such a high rate as in other GAs [Davis91].

The unfold transformation is applied in conjunction with the postponing principle, so it is only active for a small number of generations (compared to the other transformations). It is intended as a transformation to further optimise the best solutions determined with the other transformations. Therefore, its application rate is not as critical because it only needs to be applied to a portion of the best solutions for a small amount of generations. The unfold application rate is fixed at 10% to ensure it is applied to a range of better solutions in the population, thereby optimising the current best solutions. The limit of 10% will provide a reasonable

amount of unfolded solutions for exploration without saturating the population with large, complex unfolded-designs.

The task is now reduced to the selection of application rates for the remaining 4 transformations (genetic mutations). An exhaustive experimental analysis, where each possible combination is analysed, would be a complex and time consuming task. For example, 4 operations with a limited range of 5 possible values would require  $5^4$  experiments.

The number of possible application rates and therefore possible combinations is reduced by not considering small fluctuations in application rates, as they are not expected to have a large impact on the GA. To further reduce the complexity of the task, the previously mentioned transformation heuristics (section 5.4) are used to select sets of potential application rates e.g. apply retiming at a higher rate than pipelining.

Table 5.2 shows seven sets of potential application rates, expressed as probability percentages. Each set is applied to each of the four example designs and the resulting GA performance analysed to gauge the performance of that set of parameters.

		<b>Application Rates</b>						
<b>Batch Number</b>		<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>
<b>Operator</b>								
<b>Retime</b>		1	5	10	5	10	20	20
<b>Back Retime</b>		1	5	10	5	10	20	20
<b>Pipeline</b>		1	5	10	1	1	2	5
<b>Auto-Pipe</b>		1	5	10	1	1	2	5

Table 5.2 Application Rates for Analysis

Batches 1-3 apply an approximate geometric progression to the application rates from 1 to 10. Batches 3-6 apply the retiming transformations at greater rates than the pipelining transformations in accordance with the transformation heuristics. This set of potential application rates is a small sample of the possible combinations. Therefore, the determined application rates may not be optimal. However, the chosen sets allow for analysis of the effect of the application rates without requiring a prohibitive number of combinations.

Each batch is applied to each of the five benchmark DFG designs 50 times, resulting in 1750 ( $50 \times 5 \times 7$ ) complete runs of the GA. Each design is synthesised 50 times to minimise the effects of the stochastic nature of the GA. The graph in Figure 5.39 demonstrates the effect of the application rates on determining the best solution.

Each run of the GA produces a design with the best-found power reduction (reported as a percentage of the original power consumption). The mode value from all 50 runs of a single design is taken as the best power reduction obtained for that design with the current set of application rates. For the purposes of this analysis, all results for each design are compared with the worst result found for that design across all seven batches. To allow the results of all designs to be compared on a single graph the y-axis is normalised with respect to the worst design. Therefore a higher value on the y-axis indicates a design superior to the worst, whereas a value of 1 on the y-axis indicates a design equal to the worst.

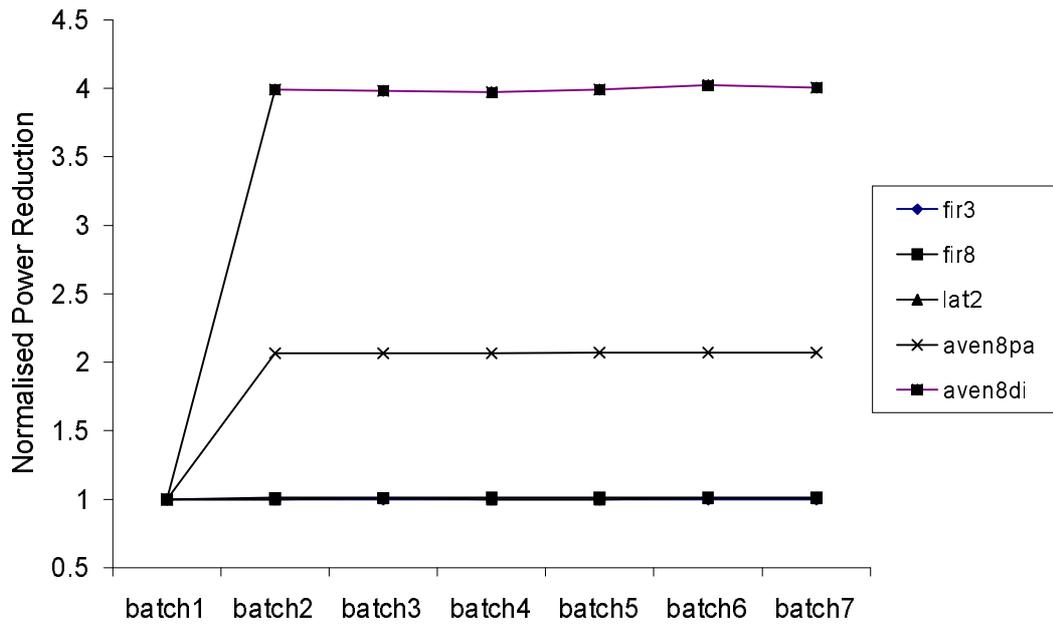


Figure 5.39 Effect of Varying Application Rates on Power Reduction of Benchmark Designs

The graph shows that batch 1 produced the worst results for all of the designs. The performance of batch 1 is so poor that the results of batches 2-7 appear indistinguishable i.e. it implies that the different application rates of batches 2-7 have no effect on the results. However, the graph of Figure 5.40 illustrates the results of batches 2-7 with the results of batch 1 omitted.

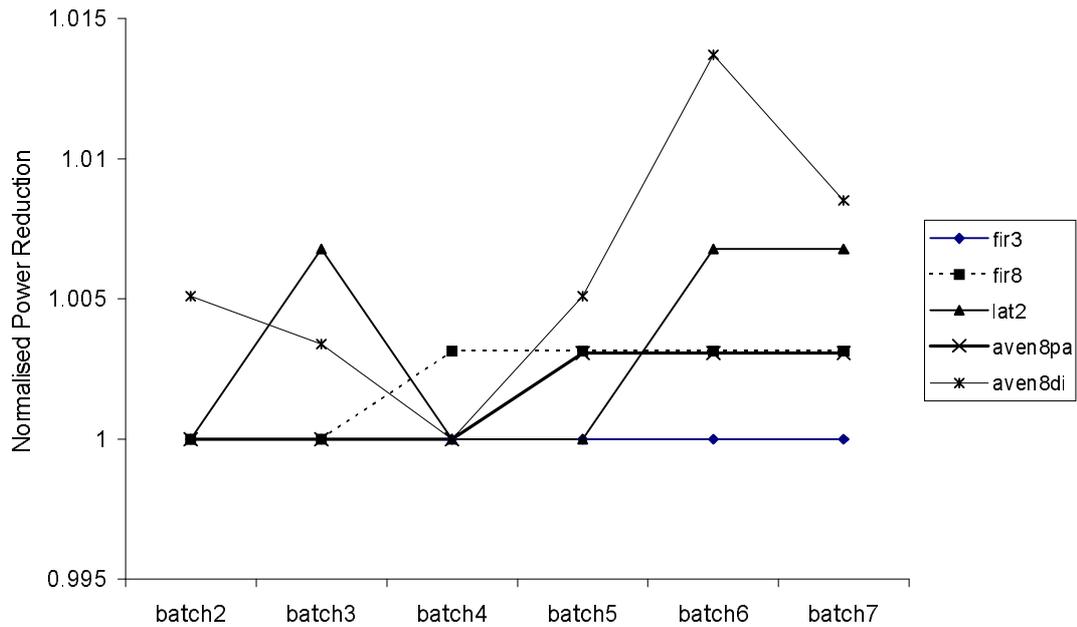


Figure 5.40 Effect of Varying Application Rates on Power Reduction of Benchmark Designs (Batches 2-7)

This graph illustrates that all designs, with the exception of the FIR3 design, are affected by the different application rates. The FIR3 is unaffected because it is a relatively simple design that requires a few simple optimisation steps.

The results do not follow a visible trend; however, it is clear that all of the designs have the best power reduction in batch 6. Note that although some of the designs also receive the best power reduction in other batches, batch 6 is the only batch where all of the designs receive the best power reduction. Therefore the application rates of batch 6 are chosen as the current set of application rates for GALOPS, in combination with the previously specified unfold and crossover rates.

The chosen set of rates applies the retiming transformations at a significantly greater rate than the pipelining transformations, supporting the assumptions made in analysing the properties of the transformations for their

optimisation capabilities. This illustrates the potential of the GA synthesis tool as a means of analysing the interaction between various optimisation techniques.

The graph in Figure 5.41 illustrates the effect of the application rates on the average number of generations required by the GA to produce the best results. Again, to enable comparison of all designs, the results are compared with the worst across all batches (the highest number of generations). Therefore, in this graph a value of 1 on the y-axis corresponds to the largest number of generations for that design, lower values denote less generations.

The selected application rates do not produce the minimum number of generations. However, the goal of the synthesis system is to produce the optimum result, with the required length of time to produce this result a secondary consideration.

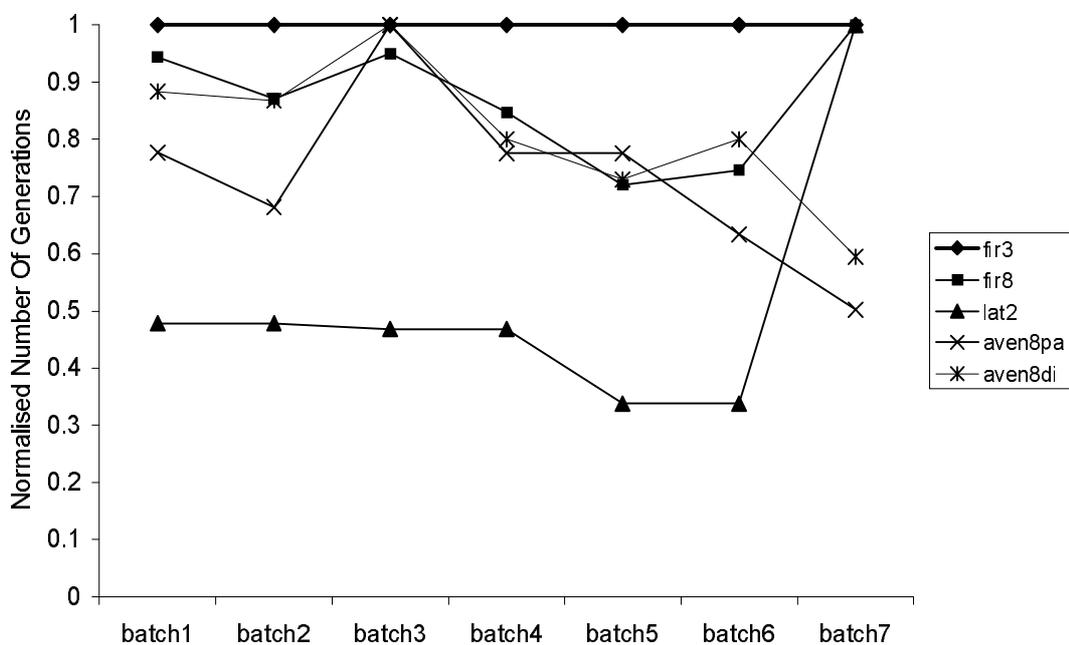


Figure 5.41 Effect of Varying Application Rates on Convergence Rate

### 5.9 Implementation and Results

GALOPS is implemented in the C programming language and is compiled and executed on a Pentium Pro 200MHz running the Microsoft Windows NT<sup>®</sup> operating system. The computer has 64 Megabytes of RAM. Different benchmark designs have different memory requirements. However, the designs presented here were all optimised within 16 Megabytes of RAM.

As discussed previously, the stochastic nature of a GA makes it possible that the GA will produce different results under the same operating conditions e.g. different required number of generations. Therefore, as with the determination of the operating parameters, each design was processed 50 times to produce a set of results for each design. The GA was executed with the operating parameters determined in section 5.8.

A distinction is made between the synthesis of designs with and without the application of the unfolding transformation. Unfolding is a powerful transformation that can produce designs with low-power characteristics, however its application can incur a large area overhead. It is not always desirable to minimise power at the expense of this increased VLSI device area that unfolding can produce. Therefore, the ability of the synthesis system to optimise designs without the application of the unfolding transformation is examined.

#### *5.9.1. Optimisation without Unfolding*

Table 5.3 presents the results obtained by GALOPS for the processed benchmark designs. The result reported is the mode value for the total set of 50 runs of each design. The mode, rather than the average, is used as it is an actual result

from the GA that refers to a synthesised design. An average of the results may refer to a design that was never actually produced (and may in fact be infeasible).

The Power Percentage column reports the estimated power consumption of the synthesised design as a percentage of the original power consumption. The Estimated Size compares the size of the optimised design to the original design; this provides the data for the area increase necessary to implement the low-power design. Supply voltage is the new  $V_{DD}$  for that design. The table also lists the average number of generations required to synthesise the best design, as a guide for the run-time of the GA. The results are all based on the assumption that the designs operate at an initial voltage of 5V at the required throughput rate.

<b>Design</b>	<b>Power Percentage (%)</b>	<b>Estimated Supply Voltage (Volts)</b>	<b>Estimated Size (New Size : Original Size)</b>	<b>Average Number Of Generations</b>
<b>FIR3</b>	19	2.1	1.04	1
<b>FIR8</b>	9	1.5	1.07	627
<b>LAT2</b>	59	3.9	1.00	6
<b>AVEN8PA</b>	21	2.3	1.02	298
<b>AVEN8DI</b>	16	2.0	1.02	506

Table 5.3 Results without Unfolding of Prototype GALOPS

The results demonstrate that GALOPS has reduced the power consumption of all of the benchmark designs to less than 60% of their original estimated power consumption. The reductions have been achieved with a significant reduction in the supply voltage of each design; it is clear from the table that the greater the supply voltage reduction the greater the corresponding power reduction.

The power consumption of the FIR3 filter has been reduced to approximately 20% of the original design. This was achieved by reducing the critical path of the filter from 3 control steps to 1, allowing a reduction in supply

voltage to approximately 2V. The reduction in critical path was achieved with the use of two pipeline stages, as illustrated in Figure 5.42. This figure was created with a graphical application specifically designed for use with GALOPS, to interpret and display the DFG designs [Rhis97].

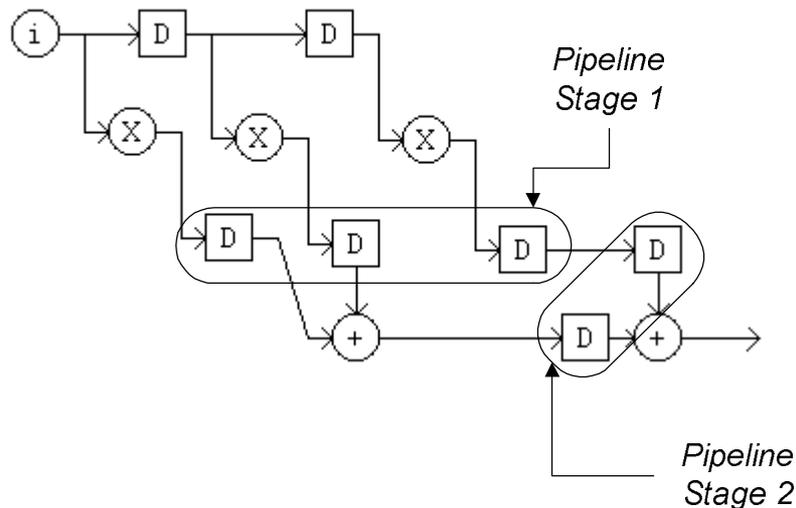


Figure 5.42 Illustration of Pipeline Stages in Optimised 3<sup>rd</sup> Order FIR Filter

The pipeline stages are marked out in the figure. The size increase of the design was approximately 4%, due to the increased number of registers that it is estimated will be required to implement the design. This apparently small increase in area is primarily due to the use of the 1:1 technique for the allocation of resources in the VLSI device. The 1:1 technique allocates a separate unit to each process node; the application of pipelining does not increase the number of process nodes so it does not significantly affect the area. In effect, the original design is area inefficient so the optimisations do not require considerable extra area.

However, GALOPS has determined a DFG for the signal-processing function, optimised for low-power operation using the 1:1 allocation technique. As

the allocation technique is a component of the fitness function other allocation techniques can be incorporated into the fitness function without requiring modification of the optimisation procedure. The FIR3 design was produced on the first generation, as it is simply the combination of the two pipeline stages.

The FIR8 design has a much greater power reduction than the smaller FIR design. This is because the critical path has been reduced from 8 elements to 1 element. This larger ratio of reduction allows the FIR8 filter to be operated at a much lower voltage while maintaining the same throughput as its original. The optimisation was produced with a combination of pipeline stages and the retiming of the inserted pipeline stages. Again, the small power increase is due to the extra registers required.

The LAT2 filter is the smallest recursive filter in the benchmark set. It reports the poorest power reduction but it is still reduced to 60% of the original. The application of retiming has reduced the critical path from 4 to 3 steps, allowing a 1.15V reduction in voltage. This small filter illustrates the constraint that the feedback loops in a DFG place on the use of transformations such as pipelining.

The power consumption of the Avenhaus filters has been reduced to less than 25% in both cases. However, the AVEN8DI has the greater power reduction. This is due to the fact that both designs have a reduced critical path of 3 steps, but the AVEN8DI had a greater initial critical path of 10 steps (compared to 8 steps in AVEN8PA). The critical path of AVEN8DI has been reduced by a greater amount so the supply voltage of AVEN8DI can be correspondingly reduced by a larger amount while still preserving the throughput rate.

It is important to note that the larger critical path of the non-optimised AVEN8DI, while giving greater scope for power reduction, requires a faster clock

speed to attain the same throughput as the non-optimised AVEN8PA, i.e. the AVEN8PA has the larger maximum speed and hence greater throughput rate. As both designs have maintained the same original throughput rate, the optimised AVEN8DI has lower power consumption but also lower throughput rate than the optimised AVEN8PA. This example illustrates the importance of the algorithm and representation format for power optimisation, as the same function produced by two different algorithm representations produces designs with different speed and power characteristics. Figure 5.43 illustrates the DFG of the optimised AVEN8DI DFG.

The DFG structure has been obtained through a combination of the retiming and pipeline transformations. The correct application of the transformations has resulted in each feedback loop of the DFG containing the same amount of delays as the corresponding loop in the original DFG. This is a requirement for the preservation of the original function of the DFG. This DFG could be further improved by optimising the allocation of delay nodes in the DFG. For example, the bank of 4 delay nodes located at the bottom of the DFG could be removed and the four delay nodes present in the vertical centre of the DFG used to supply the exact same data. However, such an optimisation could also be performed as part of the scheduling and allocation step.

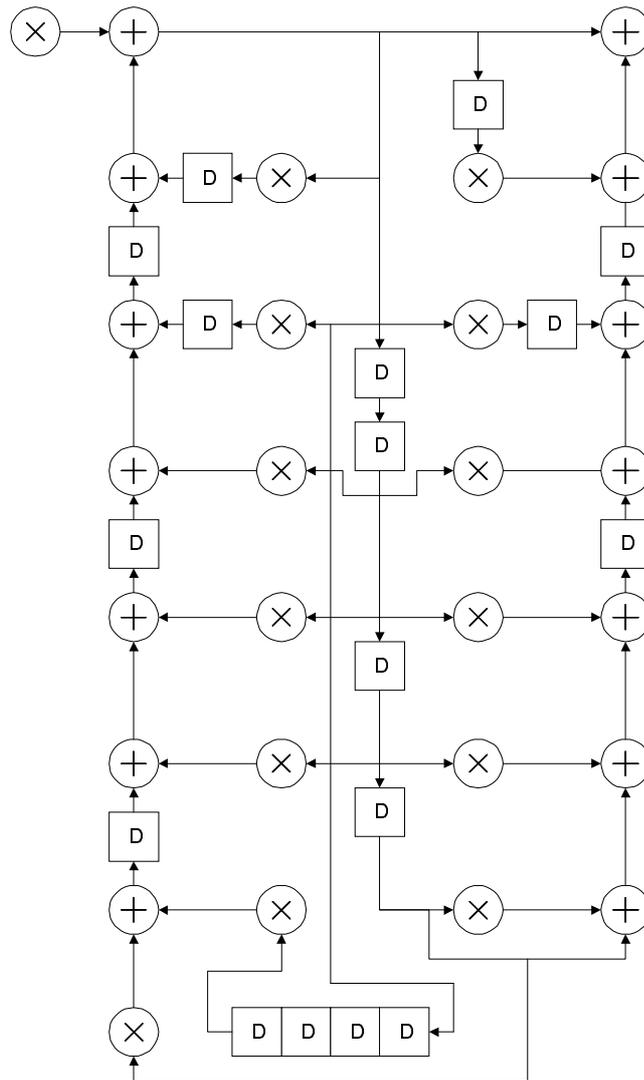


Figure 5.43 Processed DFG of 8<sup>th</sup> Order Avenhaus Filter – Direct Form

The performance of the DFG during the synthesis of this structure is illustrated in Figure 5.44 and Figure 5.45. Figure 5.44 depicts the fitness of the fittest solution in the population throughout evolution. This graph is an average over all runs that produced the DFG of Figure 5.43.

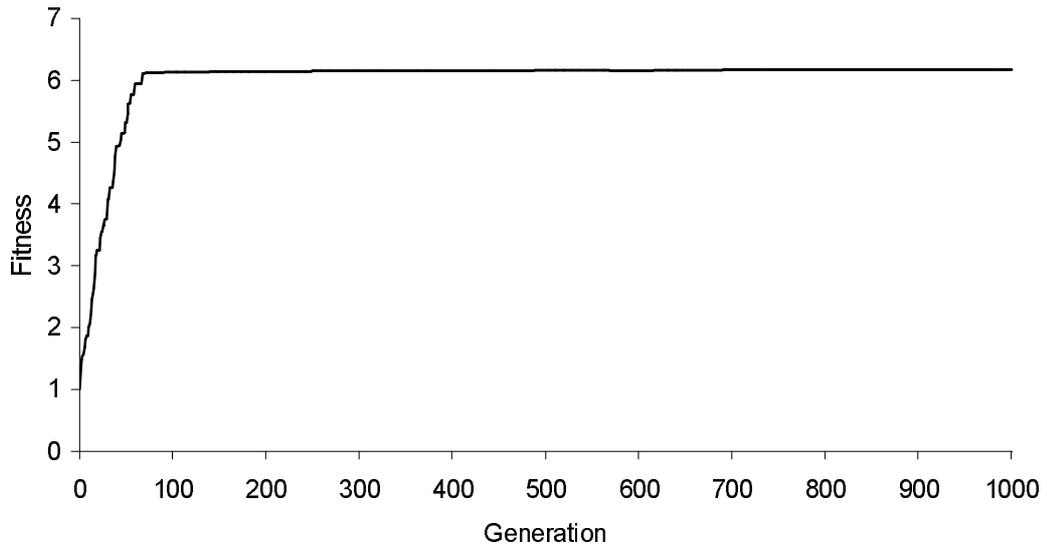


Figure 5.44 Plot of Best Fitness throughout Evolution from Average of Multiple Runs

The graph illustrates that the GA performs the majority of the fitness improvements in the first 100 generations. The best solution is found, on average, at between 400 and 500 generations.

Figure 5.45 presents the best performance of the synthesis of the AVEN8DI design; considered the best performance for this design as it produces the best design found in the minimum number of generations. This graph also shows the average fitness of the population, to illustrate how the population begins to converge around a fittest solution.

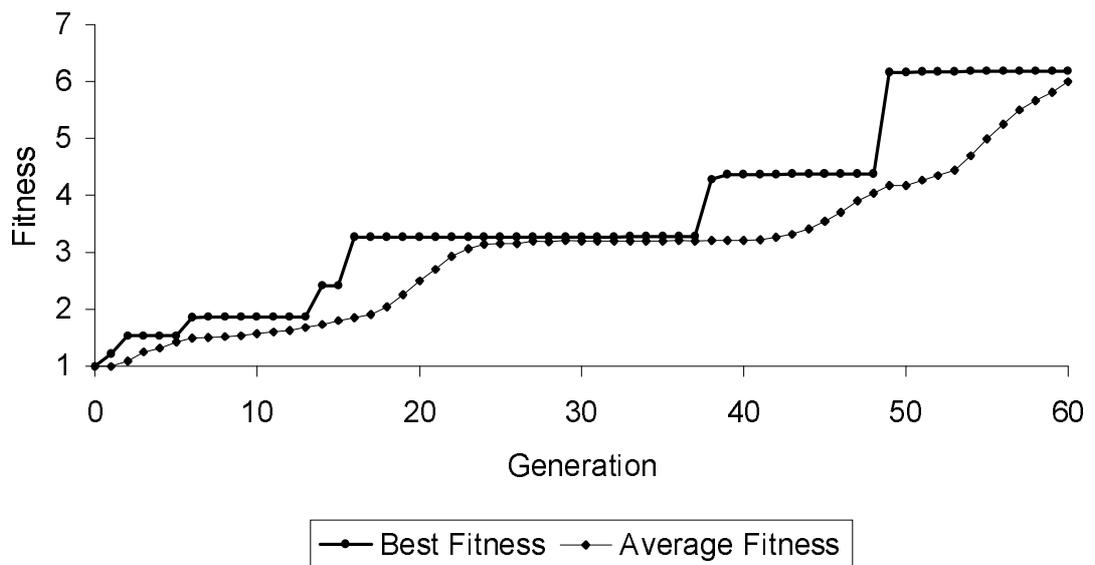


Figure 5.45 Best GA Performance for AVEN8DI Design

The graph illustrates the successive improvements to the design produced throughout evolution. Small increases in fitness are primarily due to decreases in the expected area of the design. The larger increases in fitness, such as at generation 50, are due to decreases in the voltage of the design; the voltage has the greatest impact on the power and hence the fitness of the design. The average fitness curve illustrates how the average fitness of the population slowly rises to the best fitness after each increase in best fitness. However, the average fitness never equals the best fitness, indicating that the population is never saturated with copies of the best design.

The results in Table 5.3 also illustrate how the complexity of the design affects the number of generations required for the optimisation process. The more complex designs, such as FIR8 and AVEN8DI require considerably longer generations than the FIR3 and LAT2 designs, which are relatively small. The AVEN8PA requires fewer generations than the AVEN8DI, implying that the

optimisation of the AVEN8PA representation is less complex for the purposes of low-power optimisation. Again, this illustrates the effect of algorithm representation and selection on the synthesis process.

*5.9.2 Optimisation with Unfolding*

<b>Design</b>	<b>Power Reduced To (%)</b>	<b>Estimated Supply Voltage (Volts)</b>	<b>Estimated Size Compared To Original</b>	<b>Average Number Of Generations</b>
<b>FIR3</b>	8	1.4	8.04	430
<b>FIR8</b>	7	1.3	4.11	1890
<b>LAT2</b>	59	3.9	1.00	6
<b>AVEN8PA</b>	21	2.3	1.02	298
<b>AVEN8DI</b>	16	2.0	1.01	506

Table 5.4 Results with the Application of Unfolding

As with the results for the non-application of unfolding, these results were produced from 50 complete runs of the GA for each of the designs (a total of 300 complete GA runs), with the GA operating parameters as selected from batch 6.

If these results are compared with those in Table 5.3, for the non-unfolded case, it is apparent that the recursive designs (the LAT2 and the two AVEN designs) have exactly the same results. In the case of the LAT2, the application of unfolding has failed to improve the fitness of the design produced with the rest of the transformation set. The simple recursive loop of LAT2, when unfolded creates a design with a significantly longer critical path. Even when considering the fact that the N-unfolded design is processing N samples in parallel, the unfolded LAT2 design has greater power consumption than the original due to the increased area required. Therefore the best LAT2 design determined with the GA system is a non-unfolded design.

The AVEN filters do not benefit from the application of unfolding. However, this is due to the increased size of the unfolded designs exceeding the 90mm<sup>2</sup>-area limit placed upon the system by the capacitance models. The unfolded designs are therefore assigned a very low fitness as they are assumed to be unfeasible low power designs.

The non-recursive designs have received the greatest power reductions from the application of the unfolding transformations. The fact that these designs do not include any feedback loops improves the success of the unfolding transformation in creating new designs with higher speeds through the use of parallelism. The higher speeds can be traded for reduced supply voltage and hence lower power consumption.

The increased use of parallelism accounts for the large increases in area of each design. The FIR3 design has an increased size over 8 times that of the original; the design has an unfold factor of 8, so it effectively consists of 8 times as many processing nodes, sampling 8 sets of input data in parallel. This illustrates one of the key exploitation areas for low power VLSI design. Increased integration levels have made it possible to place more transistors on a device; as this example shows, some extra transistors can be traded for low power operation. However, a size increase of 800% is only tolerable dependent on how critical the requirement for low power operation is. In effect, there is a trade off between low-power operation and area. The amount of significance to place on each parameter is very much application dependent.

The increased number of generations required to produce the unfolded FIR designs is due to the application of the postponing principle. This principle holds off the application of unfolding until the best fitness of designs within a given

population has not increased for a number of generations; therefore the postponing principle increases the number of generations required to produce the best design.

Figure 5.46 illustrates the effect of postponing with the fitness profiles obtained for the FIR8 filter during the best run (that which produced the best design in the minimum number of generations). The graph includes the best solution and the average fitness of the entire population in the current generation.

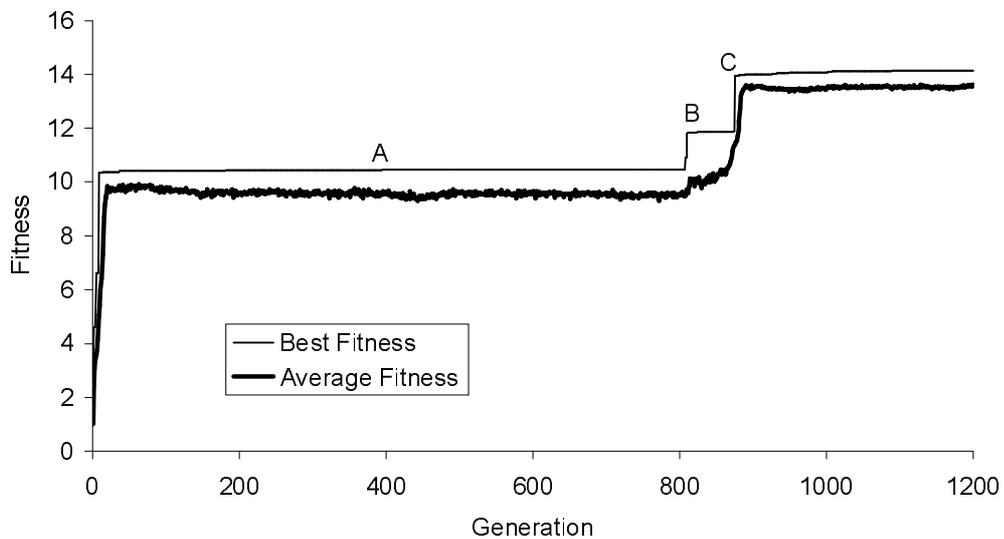


Figure 5.46 GA Performance and Effect of Unfolding on AVEN8DI Design

The graph has three main points of interest. Point A indicates the last increase in fitness due to the application of transformations without unfolding. At point B the best fitness has been static for a large number of generations so the unfolding principle is used to add the unfolding transformation to the set of transformations that can be applied. The application of unfolding at point B results in an increase in fitness. The unfolding transformation remains in the potential set of transformations for a number of generations; hence the subsequent increase in fitness at point C, again due to the application of the unfolding transformation. In

this case the unfolding transformation has increased the fitness of a previously unfolded design by increasing the order to which it is unfolded. The unfolding transformation has a cumulative effect; i.e. an N-unfolded design subsequently unfolded to a factor of P will have a total unfolding factor of  $N \times P$ .

After point C the unfolding transformation is unable to further increase the fitness of the design due to the limitations on the maximum size. The small increases in fitness after point C are due to the application of the other transformations, increasing the fitness of the unfolded design. This graph also illustrates that unfolded designs are produced over a larger number of generations than non-unfolded designs.

### ***5.9.3 Conclusions***

A Genetic Algorithm approach to high-level low-power synthesis has been introduced. The GA uses non-standard chromosome representation with problem specific genetic operators developed for the synthesis tool. The flexibility of the non-standard chromosome representation ensures that the GA is capable of synthesising signal processing designs of varying complexity and size. The chromosome representation incorporates knowledge specific to the problem domain such as the concepts of precedence of operations and the implementation details of VLSI functional units.

The problem specific operators modify the standard genetic crossover and mutation techniques in order to apply standard DSP design transformations within the GA framework, to ensure that design functionality is preserved. In addition to the preservation of functionality, the modifications enable the genetic operators to incorporate expert DSP VLSI design knowledge into the GA based synthesis tool.

The GA successfully searches the complex search space inherent in high-level low-power synthesis. The results presented for a variety of signal-processing architectures demonstrate that the GA is an effective tool for DSP synthesis.

The use of a transformation library to supply the GA with low-power optimisation techniques provides a framework for the addition of new techniques, to investigate their power reducing applications, or to determine the effect of interaction between transformations on providing the best solution. The examination of potential sets of transformation application rates produced results that supported hypotheses regarding the relationships between the transformations.

The transformations used, together with the application rates determined through experimental results and analyses, produce low-power designs without affecting design function or suffering loss in performance.

The prototype version of GALOPS incorporates standard features of a GA, such as roulette wheel selection and fitness scaling (linearisation). There are many possible modifications to a standard GA aimed at improving performance, such as the discussed use of linearised fitness values. Investigation of modifications to the GA are necessary, to examine their effectiveness and to possibly improve the performance of the GA.

The fitness function of GALOPS incorporates a power estimation module for the comparative analysis of designs for power consumption and area. More sophisticated techniques for the estimation of these parameters exist. These techniques need to be examined and incorporated into GALOPS to improve the performance of the GA and the reliability of the results.

## **Chapter 6 - Enhanced Fitness Estimation**

The previous chapter presented the initial version of GALOPS, describing the power optimisation and estimation techniques used to achieve the presented results. The initial version of GALOPS uses power estimation techniques that rely on the estimation of supply voltage and capacitance. The supply voltage estimation uses a statistical model, relating speed to voltage, to determine accurate voltage estimation.

The capacitance estimation technique analyses the target DFG design to estimate the required number of functional units to implement that design, together with the associated interconnect capacitance. The estimation of the number of functional units is based on a 1:1 allocation technique.

The allocation process is part of the high-level synthesis task. To summarise, high-level synthesis is the process of mapping a target DFG design onto a Register-Transfer-Level (RTL) design that implements the DFGs behaviour. The 1:1 allocation technique creates an RTL design with as many functional units as there are operations in the DFG. The 1:1 technique has the benefit of simplicity during the synthesis process; however, the 1:1 allocation ignores the fact that not all the operations within the DFG are required to be executed during every control step (or clock cycle). Operations of the same function, that are required to be executed at different times, can actually share the same hardware resource. Therefore, the 1:1 allocation technique creates an area-inefficient RTL design, with more functional units than necessary.

Practical high-level synthesis systems use a number of techniques to attempt to produce an RTL design with the minimum number of functional units, while obeying throughput constraints. Therefore, the fitness function of GALOPS needs to incorporate these techniques to produce more practical area estimations of the target DFG designs.

The three main tasks of high-level RTL synthesis are scheduling, allocation and binding. Scheduling and functional unit allocation typically occur in parallel, therefore it is the scheduling process which determines the number of functional units required for implementing the DFG.

This chapter describes improvements to the power estimation routines used in GALOPS, to improve their accuracy and make the power analysis more relevant to practical VLSI synthesis systems which use tasks such as scheduling and allocation to improve the efficiency of the final VLSI design. Section 6.1 introduces the concepts and techniques used in scheduling and allocation to synthesise high-level designs. Section 6.2 presents the requirements for a system which estimates the final number of hardware resources (to estimate area and capacitance) before sections 6.3 to 6.7 describe the hardware estimation techniques in detail. Section 6.8 integrates the new hardware estimation techniques into the capacitance estimation process. Section 6.9 presents an overview of the new power estimation routines incorporating the techniques described in this chapter.

### **6.1 Scheduling and Functional Unit Allocation**

The basic scheduling problem is defined as the problem of mapping a set of operations in the DFG onto a set of control steps, such that the precedence

relationships in the DFG are preserved. Figure 6.1 illustrates a simple DFG with one possible schedule.

The process of generating a feasible schedule, which obeys precedence constraints, is not considered a primary topic of the work presented in this thesis. A good overview of the main topics of the scheduling process is presented in [Walker95].

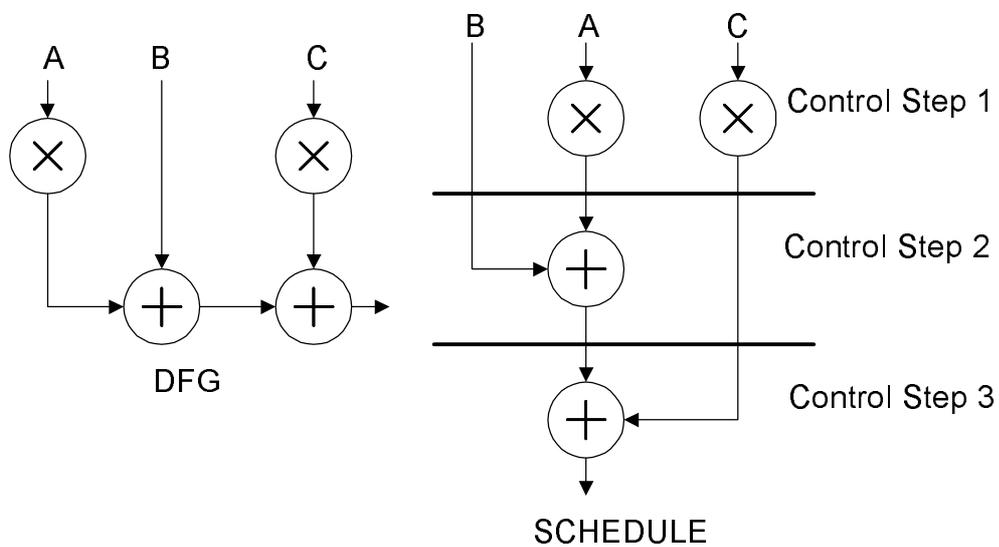


Figure 6.1 Example Scheduling Operation

The above schedule was generated using the As-Soon-As-Possible (ASAP) scheduling algorithm where every operation is placed into the earliest possible control step while obeying precedence relations [Hwang91, Micheli94]. The example DFG has been scheduled in 3 control steps, so a complete iteration of the DFG requires three control steps. The required number of functional units of each type is determined by counting the number of units in each control step. The maximum number over all control steps is the required number of units. In this

example, 2 multipliers are required because both multiplications are scheduled in the same control step. One adder is required to implement the DFG.

The example of Figure 6.1 illustrates one of the problems of the scheduling process, that of determining a schedule with the optimal (minimum) number of functional units. The multiplication of input C could be scheduled in control step 2, as its output is not required until control step 3. This would produce a schedule that requires 1 multiplier and 1 adder; 1 fewer multipliers than required with the initial schedule. Therefore, even with this simple DFG, different schedules exist with significantly different implementation requirements. Determining a schedule which meets implementation constraints on the number of available resources is known as the Resource Constrained Scheduling (RCS) [Walker95] problem, such as the limitation of 1 multiplier for the example in Figure 6.1.

Another common constraint in the scheduling process is the limitation of the number of control steps, known as Time Constrained Scheduling (TCS). The number of control steps is directly proportional to the speed of the implemented design; generally, fewer control steps results in a faster design [Walker95]. The combination of scheduling with limited resources and control steps is the problem of Time and Resource Constrained Scheduling (TRCS) [Walker95].

Determination of an optimal scheduling i.e. scheduling with minimum time and resources, is a non-trivial task as the scheduling problem is at least NP-complete [Rabaey91a, Gebot93]. Various heuristic algorithms have been developed to tackle the complex scheduling/allocation process. The next section introduces some of the more widely used heuristic scheduling algorithms, together with other techniques used to solve the scheduling problem.

An important point to note in the example of Figure 6.1 is that each operation is scheduled in a single control step. The control steps are of equal length in time; therefore, it is assumed that each operation is performed in the same length of time i.e. one clock cycle. However, in practical devices, multiplications can typically take longer than additions. The use of a unity-delay model in the scheduling simplifies the problem [Micheli94] at the cost of reducing the effectiveness of the scheduling algorithm. Consideration of non-unity operation lengths, such as allowing certain operations to span several control steps (multicycling [Walker95]) or scheduling a number of operations to be processed sequentially in a single control step (chaining [Walker95]) may enable shorter and more efficient schedules to be generated.

### 6.1.1 Scheduling Algorithms

ASAP and the similar As-Late-As-Possible (ALAP) scheduling are simple algorithms that topologically order the operations in the DFG [Rabaey91a]. Although these algorithms provide a rapid solution to the scheduling problem they are greedy algorithms which produce a single, often sub-optimal solution.

List-scheduling [Walker95] is an algorithm commonly used to solve the RCS problem. Whereas ASAP and ALAP scheduling process each operation of the DFG in turn, list scheduling processes each control step in turn; a maximum number of control steps is defined as part of the RCS problem. List-scheduling uses a ready-list to keep track of unscheduled operations that can be scheduled into the current control step without violating precedence constraints. The ready-list is sorted according to a priority function, usually based on the mobility of a particular operation. The mobility of an operation is defined as the number of control steps it

can be scheduled in. This is determined as the difference between its ASAP and ALAP control steps i.e. the earliest and latest possible control step that operation can be scheduled in. Operations with zero mobility have to be placed into the current control step, while operations with higher mobility can be placed lower in the ready-list, to be scheduled into a later control step. The use of the ready-list enables the list-scheduling algorithm to provide superior solutions to the RCS problem when compared with ASAP/ALAP scheduling. However, this increased performance comes at the cost of increased computational complexity while still not guaranteeing to provide a solution with the minimum required number of functional units.

Force-Directed Scheduling (FDS) [Paulin89] uses a stepwise refinement process to solve the TCS problem while attempting to reduce the required number of functional units. The number of functional units is minimised by uniformly distributing the operations of each type across the schedule (thus reducing the number of operations of the same type in any single control step). FDS is an iterative algorithm that successively builds up an entire schedule, keeping the schedule balanced as each operation is added. Therefore, FDS differs from list-scheduling in that it considers the whole schedule, not one control step at a time. The expected cost in functional units of assigning an operation to a particular control step is used to select which step the operation is scheduled into. As well as the direct cost of that particular operation, precedence relations with other operations create an indirect cost, which also has to be considered. As FDS concurrently considers all operations throughout the whole schedule, it produces solutions to the RCS problem while minimising functional unit cost in comparison to the list-scheduling algorithm. Unfortunately, the algorithm is less

computationally efficient than the list-scheduling and ASAP/ALAP algorithms, requiring longer execution times to create the schedule.

Another heuristic scheduling algorithm is the process of scheduling the critical path first (thus producing a design with the minimum number of control steps), followed by the scheduling of the remaining operations in control steps which minimise the functional unit requirements [Camp91].

The described heuristic algorithms suffer from the problem of determining sub-optimal solutions for many scheduling problems [Walker95]. Globally optimal solutions are desirable to maximise VLSI performance while minimising area, achieved by optimally solving the TRCS problem i.e. minimum area and maximum speed. Integer Linear Programming (ILP) [Gebot93] has been used to produce globally optimal solutions; however, the complexity of the scheduling problem means that the ILP solvers cannot determine an optimum solution in polynomial time. For relatively small problems the run-time of the ILP scheduling algorithm is large [Walker95]. Other attempts at producing globally optimal solutions, such as the use of simulated annealing [Dev89], also suffer from increased computational complexity.

The computational complexity, and the associated run-times of the discussed scheduling algorithms may not present a problem in a high-level synthesis system which is required to simply produce an RTL output for a specified design, as the scheduling algorithm is only required to be run once. However, within a design exploration system, where candidate designs are compared for desired characteristics, the long run-times can become prohibitive. This is especially the case within a GA design exploration system such as GALOPS, which typically evaluates many thousands of solutions throughout the design exploration process.

For these systems the complexity of the scheduling algorithms significantly increase the overall computational complexity.

The problem of determining the implementation parameters of the RTL design (such as speed and area), while minimising the length of time required for such an analysis has led to increased research into the estimation, rather than the calculation, of such parameters. Instead of determining the optimal RTL solution through the use of complex scheduling algorithms, less complex algorithms are used to obtain estimates on such parameters as the required number of functional units. The estimates can be used to guide the design exploration without the associated computational overhead of performing a complete RTL synthesis for optimal designs. The estimates are used to provide a fitness measure to the GA synthesis system (GALOPS). In addition, the estimates can also be used to provide starting points for the complete scheduling algorithms, reducing their required runtime [Rabaey90].

The variety of scheduling algorithms available in high-level synthesis present a range of possible RTL designs for a single power-optimised DFG. Therefore, the choice of scheduling algorithm can significantly affect the final power consumption of the RTL design. Indeed, scheduling algorithms have been developed which consider power as an objective parameter (section 2.4) during the scheduling and allocation process. To enable analysis of the power-reduction capabilities of the behavioural-level techniques presented in GALOPS, such architectural-level power reduction algorithms are not included within the current synthesis system. A complete low-power synthesis system would utilise the full range of techniques at all levels of the synthesis process.

## 6.2 Overview of Estimation of Hardware Resources

The fitness evaluation used in GALOPS estimates the power consumption of the data path component of the implemented design. Therefore, estimation of hardware resources involves the estimation of the functional units required to implement the data path as an RTL design. Precise determination of the minimum number of functional units is an NP-complete problem; therefore there is little hope of finding an estimation algorithm which could solve the problem in polynomial time [Chaud96]. However the determination of upper and lower bounds on the hardware resources presents a problem of reduced complexity which can be solved in polynomial time [Rabaey91]. Figure 6.2 illustrates the concept of upper and lower bounds on the number of functional units.

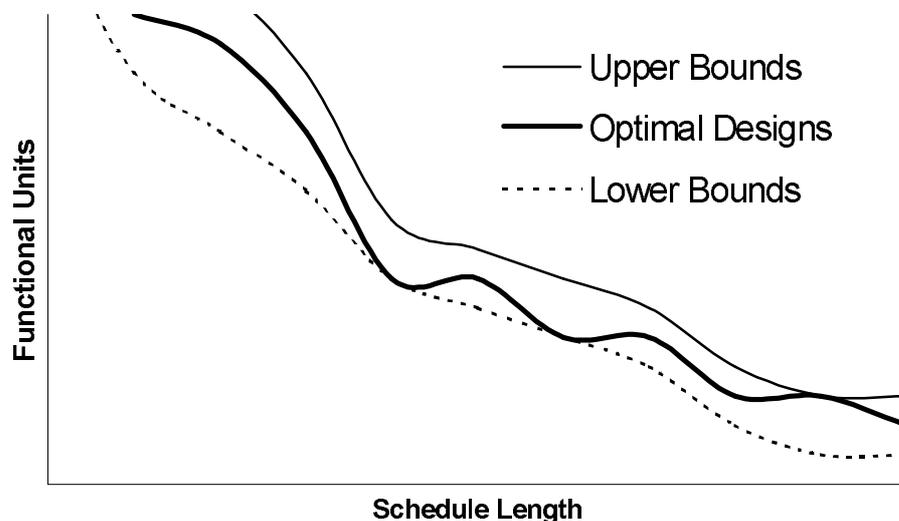


Figure 6.2 Illustration of Design Space with Upper and Lower Bounds

The graph illustrates the optimal number of functional units, for a specific design, for different schedule lengths. The optimal design is the feasible design

created with the least number of functional units. The feasible design space is represented by the area above the optimal designs curve i.e. all feasible designs require at least as many functional units as the optimum design. The upper bounds curve is within the feasible design space; i.e. all upper bounds are feasible designs. The lower bounds curve is the theoretical minimum number of functional units required to implement the design. In some cases the theoretical lower bounds may not be able to produce feasible designs; however, the lower bound provides valuable information for estimating the minimum area required to implement the DFG, thus providing a means of exploring the solution space.

The algorithms presented in [Potko89] provide estimations of the upper and lower bounds for a given DFG with a specified schedule length. Within the GALOPS synthesis system the bounds derived with these algorithms are used to compare the area and hence the capacitance of rival designs. This area estimation is based on the minimum estimated area required to implement the DFG; therefore the lower bounds on functional units are used to provide this area estimation. The lower bounds are typically used as a measure of the quality of the solution, in relation to the expected area, as they are typically the closest to the practical solution [Rabaey91a]. The determination of upper bounds on functional units is not presented in this thesis as the data is not required as part of the GA fitness evaluation.

### 6.3 Initial Estimates of Lower Bounds on Functional Units

The estimation process for determining lower bounds starts by obtaining a crude estimate on the initial lower bounds of functional estimates [Rabaey91a,

Rabaey90]. Additional steps in the estimation process are then used to refine these crude lower bounds, providing a more accurate estimate.

The crude estimate is the absolute lower bound on the number of functional units; it is obtained by assuming that the operations in the DFG can be equally spread out between all control steps. That is, given  $N$  operations of type  $i$  ( $N_i$ ), to be executed in  $X$  control steps, an absolute minimum of  $N/X$  number of resources of type  $i$  ( $R_i$ ) will be required (assuming that all operations are performed within a single control step).

For the systems under consideration in GALOPS, which are critically timed (i.e. operating at maximum throughput), the number of available control steps is the same as the critical path length ( $C_p$ ). That is, the entire DFG is processed in the same length of time that it takes to process the critical path of the DFG.

Therefore, the absolute minimum number of resources is given by:

$$\text{abs\_min } R_i = \left\lceil \frac{N_i}{C_p} \right\rceil \quad (6.1)$$

However, this is an optimistic estimate as it assumes that the operations can be evenly spread throughout the schedule. In practice, precedence relations prevent this from occurring. Therefore, the actual number of operations is typically higher than this lower bound, as illustrated by the example of Figure 6.3.

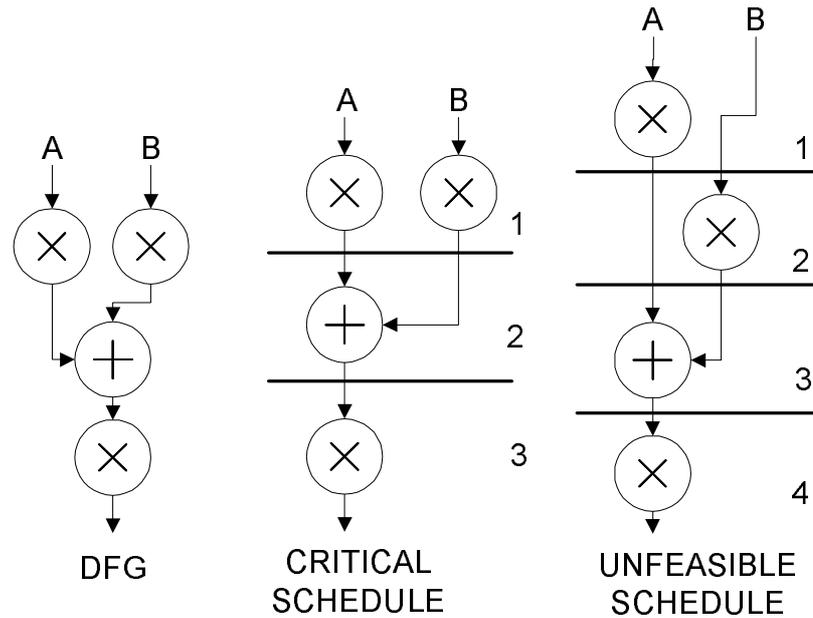


Figure 6.3 Example of Unfeasible Schedule Generated by Absolute Lower Bounds on Functional Units

The example DFG in Figure 6.3 would be estimated to require 1 multiplier if using the minimum bounds provided by equation 6.1 ( $\lceil 3/3 \rceil$ ). However, the optimum critical schedule shows that the precedence constraints on the first two multiplication operations require them to be performed in the first control step as their outputs are required for the addition in the second control step. Therefore, the actual number of multipliers required is 2. If the lower bound was used as a limitation on the number of resources the unfeasible schedule in Figure 6.3 may be generated. The schedule is unfeasible as it has more control steps than the length of the critical path. Therefore, strictly adhering to the lower bound estimation can create an implementation with a lower throughput rate than required.

Analysis of the crude lower bound [Rabaey94] for 50 DFG examples, covering a wide range of benchmark applications, resulted in a maximum observed

error between the lower bound and the actual cost of 386%. The average observed error was 72%.

An improved estimation of the lower bound is used to produce a value that is closer to the actual number of functional units that will produce a feasible schedule. This improved value increases the accuracy of the area estimation based on the minimum bound technique.

### 6.4 Improved Estimate of Lower Bounds on Functional Units

The improved estimation process begins with a topological ordering and levelling of the input DFG with respect to its input and output nodes [Rabaey91a, Rabaey90]. This is achieved by scheduling the graph using the ASAP and ALAP scheduling algorithms [Walker95, Micheli94]. The information provided by this scheduling step, together with the crude lower bounds, is then used within a relaxed estimation technique [Rabaey94] to determine more accurate lower bounds on the required number of functional units.

#### 6.4.1 ASAP/ALAP Scheduling

ASAP and ALAP are the simplest scheduling algorithms, processing each node in turn and placing it in the earliest or latest possible control step. The pseudo-code for the ASAP algorithm is shown in Figure 6.4.

```
Function ASAP_Schedule  
  Passed DFG  
  Note: C_step = Control_Step  
  
  WHILE NOT every operation scheduled  
    FOR each operation in the DFG  
      IF no predecessor operations  
        C_step of current_operation = 1  
      ELSE IF all predecessor operations have been scheduled  
        C_step of current_operation =  $\max[C\_step(\text{predecessors})]+1$   
      Next operation in DFG  
    End WHILE  
END Algorithm ASAP_Schedule
```

Figure 6.4 Pseudo-code for ASAP Scheduling Algorithm

The ASAP algorithm processes every operation in the DFG in turn until they have all been scheduled. If an operation has no predecessor nodes on its inputs it is scheduled in the first control step i.e. it is executed as-soon-as-possible. Operations defined as having no predecessor nodes are those where all inputs are driven by primary DFG inputs or delay nodes. Delay nodes in a DFG indicate inter-iteration loops i.e. a delay node indicates where the current iteration of the DFG begins and ends.

Operations with predecessor nodes can only be scheduled if those predecessor nodes themselves have been scheduled. If this constraint is satisfied the control step for the current operation is determined by finding the largest control step of all predecessors and scheduling the current operation in the next control step. This ensures that the operation does not try to process its inputs before they have been determined by its predecessor nodes i.e. precedence constraints are not violated. Performing the ASAP algorithm will produce a *schedule\_list*. The list stores every functional node in the DFG along with its ASAP control step.

The ALAP schedule puts every functional operation into the latest possible control step; the pseudo-code for this algorithm is shown in Figure 6.5. The

maximum number of control steps has to be defined before the ALAP schedule can be performed. For critically timed DFGs the maximum number of control steps is the same as the length of the critical path of the DFG. The *schedule\_list* is also used to store the ALAP control step of each functional operation.

```

Function ALAP_Schedule
  Passed DFG
  Note: C_step = Control_Step

  WHILE NOT every operation scheduled
    FOR each operation in the DFG
      IF no descendent operations
        C_step of current_operation = critical_path_length
      ELSE IF all descendent operations have been scheduled
        C_step of current_operation =  $\min[C\_step(\text{descendants})]-1$ 
      Next operation in DFG
    End WHILE
  END Algorithm ALAP_Schedule
  
```

Figure 6.5 Pseudo-Code for ALAP Scheduling Algorithm

An example of the result of a scheduling operation, for the 3 Tap FIR filter, is shown in Figure 6.6.

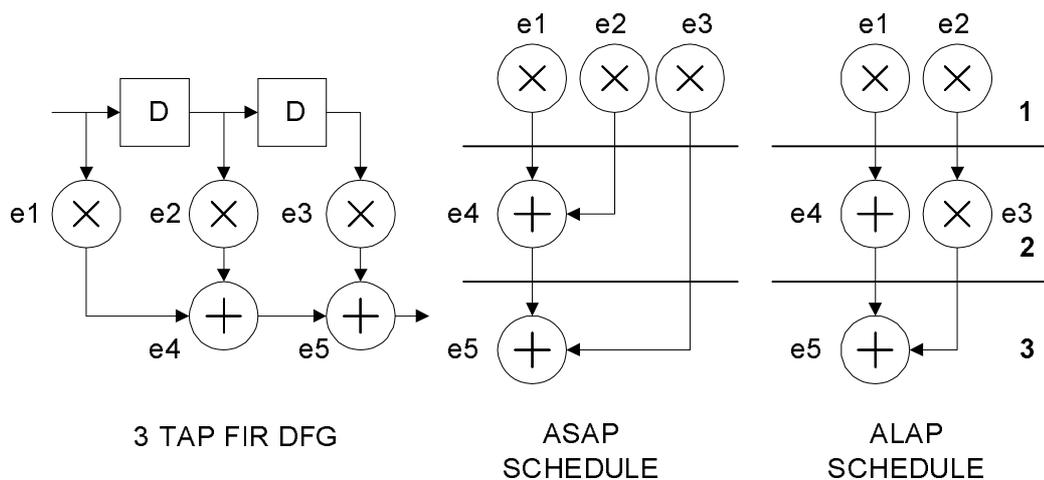


Figure 6.6 ASAP and ALAP Scheduling of 3 Tap FIR Filter DFG

An important point to note about the ASAP schedule is that it produces schedules of the same length as the critical path of the source DFG. Therefore, as both critical path length determination and functional unit estimation are required during the fitness estimation procedure, the current critical path estimation procedure can be replaced with the ASAP scheduling step.

Once the *schedule\_list* has been created with both the required ASAP and ALAP information it can be used for the next step of the estimation process.

### 6.4.2 The Relaxed Estimation Technique

The relaxed estimation technique is an iterative improvement algorithm used to refine the crude lower bounds obtained from the initial analysis of the DFG, thus reducing the error between the estimated number of functional units and the actual number of functional units.

The iterative refinement process is outlined as:

- Set maximum number of available functional units of type  $i$  ( $max\_Ri$ ) equal to crude lower bound on functional units of type  $i$  ( $abs\_min Ri$ ) determined from equation (6.1).
- Determine the minimum number of control steps ( $min\_C\_step$ ) required for implementing the DFG algorithm with the above constraint ( $max\_Ri$ ). This is effectively performing RCS scheduling i.e. scheduling with a constraint on the number of available resources to determine a minimum schedule.
- If the number of control steps is greater than available ( $CP\_length$ ) or no feasible schedule can be generated with the resource constraint

( $max\_Ri$ ), increment  $max\_Ri$  and repeat the scheduling step. Otherwise,  $max\_Ri$  is the relaxed estimation of the required number of resources of type  $i$ .

The process determines whether the current estimation of the number of functional units (see equation (6.1)) will produce a feasible schedule that does not require more control steps than those available. If the resource constraint prevents such a schedule from being generated the constraint is *relaxed* (incremented) and another schedule generated. The process is repeated until a feasible schedule of the required length is generated i.e. a schedule that satisfies the timing constraints.

The process can be illustrated by considering the 3 Tap FIR example. Initial lower bound estimates for this DFG creates a limit of 1 on the number of add and multiply functional units. The schedule generated with this constraint is shown in Figure 6.7. As only one operation of each type can be scheduled in each step, the scheduling process generates a schedule of length 4. The critical path length of the 3 Tap FIR filter is 3, therefore this schedule violates the timing constraints. The limitation on the maximum number of resources needs to be relaxed to produce a schedule that satisfies the timing constraints. In this example, increasing the number of available multipliers to 2 will produce a schedule that satisfies the timing constraints. Therefore the lower bound estimation on multipliers has been refined to a more accurate estimate.

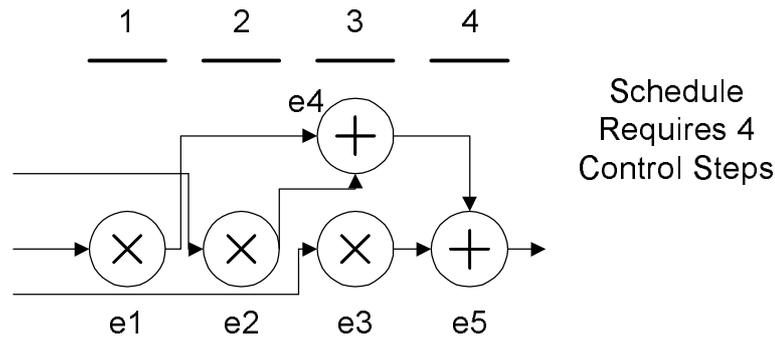


Figure 6.7 Illustration of Schedule Generated Using Crude Lower Bounds as Limits on Functional Units

A problem arises when considering which resource bound to increment when considering more than 1 type of resource i.e. should the constraint on adders or multipliers be relaxed? Analysis of the problem in Figure 6.7 reveals that it is the multiplier constraint that needs to be relaxed, but it is difficult to generalise the identification process for all algorithms. The relaxed estimation process reduces the complexity of this problem by considering only one functional unit type at a time. The relaxed estimation process is as follows:

1. Determine crude lower bounds on adders and multipliers
2. Use iterative refinement process to estimate number of adders using lower bounds on adders as an initial constraint but no constraint on the number of multipliers.
3. Use iterative refinement process to estimate number of multipliers using lower bounds on multipliers as an initial constraint but no constraint on the number of adders.

While this ‘relaxation’ of constraints considerably reduces the complexity of the problem, it may produce estimates that do not actually generate feasible results [Rabaey94]. More complex algorithms have been developed to consider the parallel estimation of 2 or more types of functional units [Chaud95, Chaud96]. However, they rarely outperform algorithms that only consider one type of resource [Chaud96, Potko97] while requiring a more complex and hence computationally intensive estimation process.

The actual scheduling process is performed with the list-scheduling algorithm. This algorithm is commonly used to solve the RCS problem as it produces relatively efficient schedules without requiring large run-times [Walker95]. The pseudo-code for this algorithm is shown in Figure 6.8.

**Algorithm List\_Schedule**

Passed *ASAP* and *ALAP* scheduling information  
Passed limits on numbers of functional units

Compute *slack\_times* of each operation

*Current\_Cstep* set to 0

While NOT all operations scheduled

    Increment *Current\_Cstep*

    Identify *data-ready* operations

    Place *data-ready* ops into *ready-list*

    Sort *ready-list* in order of *slack\_times*

    For each op in the *ready-list*

        Schedule into *Current\_Cstep* IF functional units limit not broken

    Next element in *ready-list*

End While

Return length of schedule (*Current\_Cstep*)

End Algorithm **List\_Schedule**

Figure 6.8 Pseudo-code for List-Scheduling Algorithm

The list-scheduling algorithm uses a ready-list to keep track of data-ready operations. A data-ready operation is defined as an operation that can be scheduled

into the current control step without violating precedence constraints. To determine which operations are selected from the ready-list for scheduling, the ready-list is sorted to prioritise selection. The sorting criteria is based on the slack [Micheli94] of each operation, which is computed from the ASAP and ALAP times of each operation; hence the requirement for the ASAP and ALAP information.

The slack of an operation is defined as its mobility, the difference between its ASAP and ALAP control step. Operations with a smaller slack-time rate a higher priority in the scheduling process as there are comparatively fewer control steps into which those operations can be scheduled. These operations with small slack-times have less freedom for movement around the schedule when compared with operations with larger slack-times. Therefore, the core of the list-scheduling algorithm is to postpone the scheduling of the operations with more freedom until last, then schedule them into control steps that satisfy the resource constraints.

Figure 6.9 illustrates the slack-times for the 3 Tap FIR Filter example.

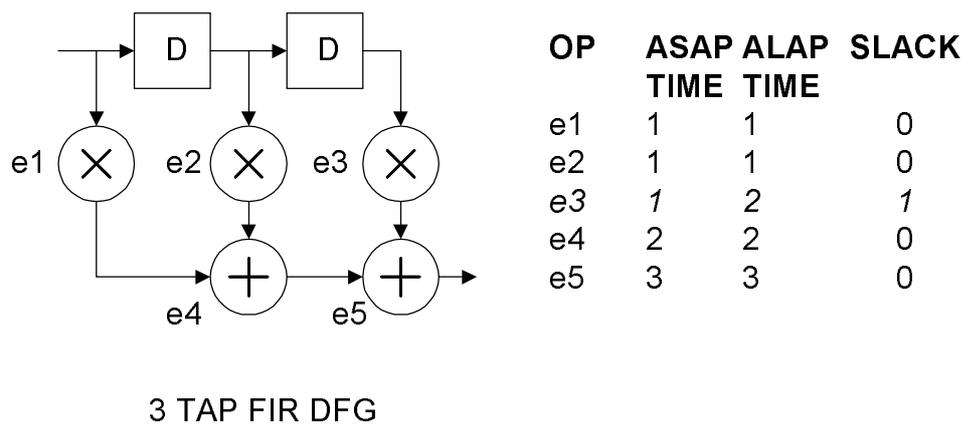


Figure 6.9 Slack Times for 3 Tap FIR Filter

Operation e3 has a slack time of 1 as it can be scheduled in control step 1 or 2. The other operations have a slack time of zero; they have no mobility so they have to be scheduled into the specified control step. Note that operations on the critical path of the DFG always have a slack time of zero.

Application of the list-scheduling algorithm produces a schedule with a certain amount of control steps, which satisfies the resource constraints specified by the lower bounds on the number of functional units. If the length of this schedule is greater than that specified by the timing constraint (the length of the critical path) then the schedule is considered unfeasible. The bounds on resources are then revised and the list-scheduling process repeated to attempt to generate a feasible solution.

The pseudo-code for the complete lower-bounds estimation process is shown in Figure 6.10.

### Algorithm **Lower\_Bounds\_Estimation**

Passed *ASAP* scheduling information (performed during crude estimation)  
Passed *DFG*

Determine crude lower-bounds on Add and Multiply units  
(*add-min*)  
(*mult-min*)

*ALAP* schedule the *DFG*

WHILE *schedule-length* greater than *critical-path-length*  
List-schedule using *add-min* as resource limit on adders  
(no limit on multipliers)

IF *schedule-length* greater than *critical-path-length*  
Increment *add-min* (increase resource limit)

List-schedule using *mult-min* as resource limit on multipliers  
(no limit on adders)

IF *schedule-length* greater than *critical-path-length*  
Increment *mult-min* (increase resource limit)

End WHILE

Return *add-min* and *mult-min* bounds as estimates of minimum number of functional units required

End Algorithm **Lower-Bounds-Estimation**

Figure 6.10 Pseudo-Code for Lower Bounds Estimation of Functional Units

The relaxed estimation technique considerably reduces the error between estimated and actual costs of functional units. Results produced on the same benchmark set used to analyse the crude lower bounds show the maximum error has been reduced from 383% to 67% with this technique, while the average and median errors are reduced from 72% to 14% and from 86% to 7% respectively.

### 6.5 Examples of Lower Bound Estimation

To illustrate the performance of the new functional unit estimation routines, results are presented for the benchmark *DFG* examples described in section 5.7. Table 6.1 lists the estimated lower bounds on the number of add and multiply units together with the crude lower estimates and the actual number of operations in the *DFG*.

<b>DFG</b>	<b>Adders</b>			<b>Multipliers</b>		
	<i>Crude Lower Bound</i>	<i>Refined Lower Bound</i>	<i>Number of Operations in DFG</i>	<i>Crude Lower Bound</i>	<i>Refined Lower Bound</i>	<i>Number of Operations in DFG</i>
<b>FIR3</b>	1	1	2	1	2	3
<b>FIR8</b>	1	1	7	1	2	8
<b>LAT2</b>	1	2	4	1	2	4
<b>AVEN8PA</b>	2	4	15	3	4	18
<b>AVEN8DI</b>	2	2	16	2	2	16

Table 6.1 Estimates on Minimum Number of Functional Units for Benchmark DFGs

The table illustrates that, for the designs in Table 6.1, the required number of functional units is always less than the number of operations in the DFG. This highlights the need to replace the 1:1 allocation technique for the area estimation of the functional units with a more robust estimation technique that is more applicable to practical VLSI devices.

The 8<sup>th</sup> order Avenhaus DFG, highlighted in the table, also illustrates the importance of the relaxed estimation technique to obtain a more accurate estimation of the required area. With the 1:1 estimation allocation technique the number of multipliers is 18, equal to the number of multiplication operations in the DFG. A crude estimate on the lower bound produces a requirement of 3 multipliers, a considerable saving, especially when the cost of multipliers in terms of area is considered. The relaxed estimation technique refines the initial estimate to 4 multipliers, requiring 4 iterations to refine both the multiplier and adder estimates. The expense of the extra iterations required to refine the initial crude estimates is compensated for by the improved accuracy.

Another important benefit of the improved area estimation routines is evident from analysing the estimates for example DFGs throughout the power exploration process. Figure 6.11 and Figure 6.12 illustrate the estimates for a wide range of possible DFGs. Both graphs depict estimations for 3501 DFGs generated throughout the exploration process. Figure 6.11 depicts the estimated lower bound on the number of multipliers, together with the actual number of multiply operations in the DFG. Figure 6.12 depicts the estimated lower bound on the number of adders, together with the actual number of add operations in the DFG.

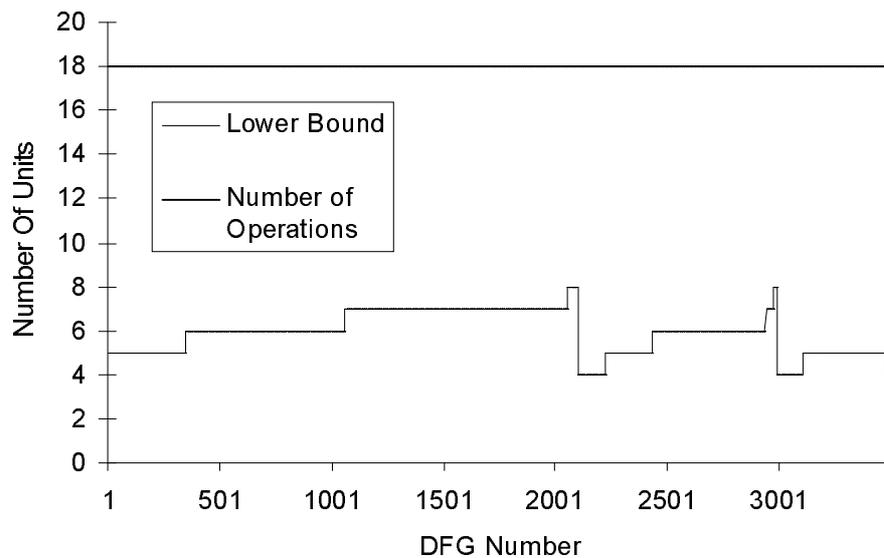


Figure 6.11 Estimates of Number of Multipliers for Range of DFGs Generated for Low-Power Exploration of 8<sup>th</sup> Order Avenhaus Filter

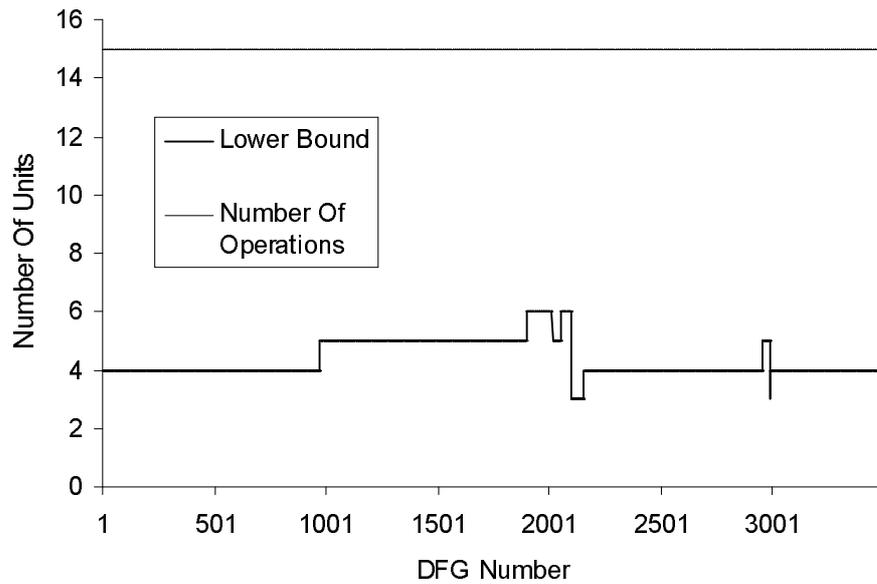


Figure 6.12 Estimates of Number of Adders for Range of DFGs Generated for Low-Power Exploration of 8<sup>th</sup> Order Avenhaus Filter

The graphs illustrate that the estimated number of resources required for implementation varies throughout evolution, while the actual number of operations in the DFG never changes. This is due to different DFG topologies of the same function having different scheduling characteristics, resulting in different implementation characteristics. The 1:1 allocation technique shows no change in the required number of functional units throughout evolution but the more accurate estimation technique is more sensitive to changes to the DFG i.e. the effects of the transformations are more accurately analysed with the improved area estimation technique.

### 6.6 Estimation of Lower Bounds on Registers

In addition to improving the estimates of the number of functional units (to execute operations in the DFG) the improved fitness estimation routines include

more accurate estimates of the number of registers required for the RTL implementation of the DFG.

Register estimation is the process of determining how many registers are required to implement the RTL so a sample is processed in the required time i.e. within a maximum number of control steps. The minimum number of registers of type  $i$  (those connected to an operation of type  $i$ ) is computed from the following formula [Potko89]:

$$Ri \geq \left\lceil \sum_{n=0}^k t_{life\ min}^n / Total_{control\_steps} \right\rceil \quad (6.2)$$

where  $k$  is the total number of signals feeding into operation  $i$ ;  $Total_{control\_steps}$  is the number of control steps available for the schedule (equal to the length of the critical path).

The variable  $t_{life\ min}^n$  is the minimum lifetime of the  $n^{th}$  signal feeding into operation  $i$ . The minimum lifetime is defined as the minimum amount of time (control steps) that the variable is required to be ‘alive’ i.e. the amount of time it is required to be stored in a register. By determining the minimum lifetime requirements of all variables the minimum number of registers to store those variables can be determined. The example DFG in Figure 6.13 is used to illustrate the process of determining the minimum lifetime of a variable [Rabaey94].

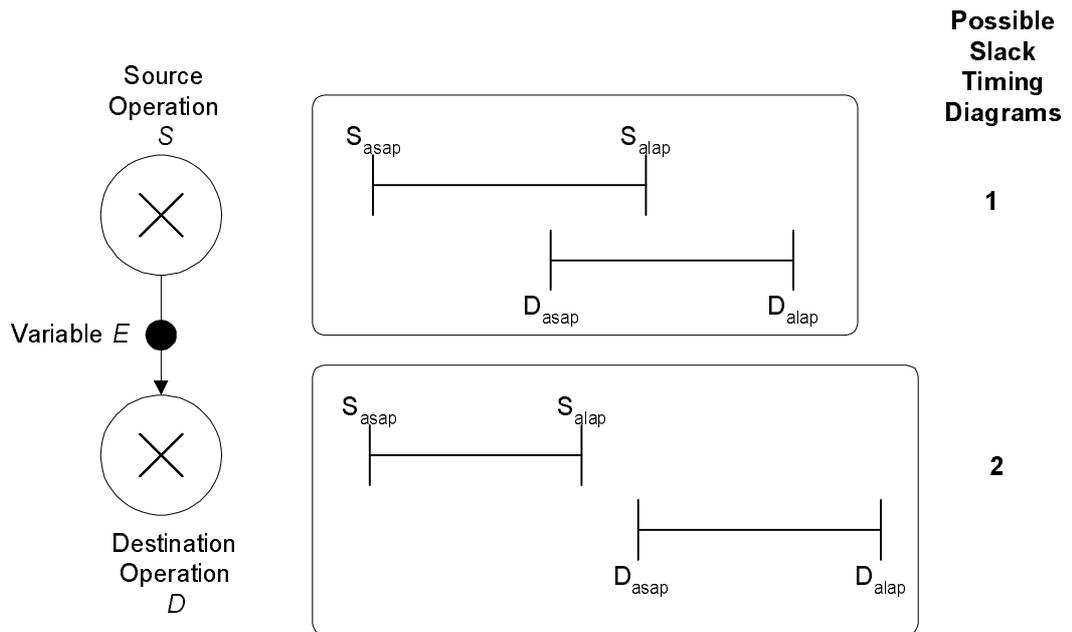


Figure 6.13 Determination of Minimum Lifetime of a Variable E

The DFG in Figure 6.13 illustrates a simple operation between two nodes (a source and destination), passing a variable between the two nodes. The minimum lifetime of the variable is determined by the scheduled execution time of the source and destination nodes. Each node has an associated ASAP and ALAP scheduling time, determined by application of the ASAP and ALAP scheduling algorithms during execution of the functional unit’s estimation process. The ASAP and ALAP times are used to produce the slack time of each operation.

As illustrated in Figure 6.13 there are two cases of interest:

1. The slack times of the source and destination overlap. In this case the source ALAP time is later than the destination ASAP time ( $S_{alap} > D_{asap}$ ). The minimum possible lifetime is achieved if the operations are scheduled to allow operation D to consume variable E immediately after

generation by operation S. The minimum lifetime is therefore the time taken for the destination D to process the variable E, equal to one control step. Therefore,  $t_{lifemin} = t_{control\_step}$ .

2. The slack times of the source and destination node do not overlap ( $S_{alap} < D_{asap}$ ). In this case variable E has to be minimally alive for a longer period of time, as operation D is not available to process the variable immediately after operation S has produced it. The variable has to be alive from  $S_{alap}$  to  $D_{asap}$  plus the processing time of operation D. Therefore,  $t_{lifemin} = D_{asap} - S_{alap} + t_{control\_step}$ .

The equations presented for the two cases also extend to the third possible case, where  $S_{alap} = D_{asap}$ . The pseudo-code for the register estimation process is illustrated in Figure 6.14.

**Algorithm Register Estimation**

Passed *ASAP* and *ALAP* scheduling information

Passed *DFG*

$Schedule\_length = critical\_path\_length$

**Determine minimum lifetimes of registers:**

For each *operation* in the *DFG*

For each *input\_net* of the *current\_operation*

Determine *source* and *destination* of *input\_net*

IF  $source\_ALAP \geq destination\_ASAP$

$Min\_lifetime$  of *input\_net* = 1

ELSE

$Min\_lifetime$  of *input\_net* =  
 $destination\_ASAP - source\_ALAP + 1$

Next *input\_net* of *current\_operation*

Next *operation* in the *DFG*

End **Determine minimum lifetimes**

$Registers\_estimate = \lceil (\sum min\_lifetimes) / schedule\_length \rceil$

End Algorithm **Registers Estimate**

Figure 6.14 Pseudo-Code for Estimation of Minimum Number of Registers

This algorithm produces a lower bound on the number of registers required to implement the DFG as an RTL design. The area of a single register is stored in the library of functional unit models so the lower bound can be used to compute the lower bound on the total area required for the registers in the data-path segment of the DSP design.

### **6.7 Hardware Unit Models**

In addition to producing more accurate estimations of the number of functional resources, the improved fitness estimation technique incorporates more accurate models of the actual hardware units in terms of their switched capacitance. The original hardware models presented in section 5.6.2 used a VLSI synthesis tool to return implementation characteristics of designed functional units. These units were characterised by area and capacitance for use in the power estimation process. However, the capacitance estimation of these models neglected the effect of internal switching activity on their capacitance. More accurate models presented in [Chan95] include the modelling of internal switching activity when determining the capacitance of the hardware units. While the current fitness estimation technique does not consider switching activity over the whole system, the new hardware models will ensure that internal switching activity of hardware modules is considered. This improves the accuracy of the power estimation process [Chan95].

The new capacitance estimation models also incorporate clock-tree capacitance data, as the registers are the datapath units driven by the clock line [Chan95]. The register capacitance models are comprised of the registers switching capacitance and its contribution to the load capacitance of the clock line. Summation of the register capacitance will produce the total load on the clock line

due to datapath units. As described in chapter 3, the intrinsic capacitance in the clock line is reduced using architectural and physical level design strategies. The new capacitance characteristics of the hardware models are shown in Table 6.2.

<b>Hardware Unit</b>	<b>Switched Capacitance (Pico Farads)</b>
8×8 Bit Array Multiplier	16.2
8×8 Bit Ripple Adder	1.162
8 Bit Register	0.482

Table 6.2 Switched Capacitance Estimations of Hardware Units

However, the results presented in [Chan95] do not include the area estimation of the hardware units. Therefore, these improved capacitance estimations are used with the area estimations presented in section 5.6.2.

### 6.8 Capacitance Estimation

The improved area estimation routines process a candidate DFG to determine how many hardware units of each type are required to implement the DFG. While this has an effect on the area of the VLSI implementation it does not have as significant an effect on the capacitance contribution to power.

The capacitance of a DSP algorithm is related to the number of operations it performs, rather than the number of hardware units those operations are performed on. 10 multiplication operations will require approximately the same amount of switched capacitance whether 10 parallel or 1 shared multiplier is used. The original method of 1:1 allocation for each operation ensured that a dedicated functional unit performed each operation; therefore, the capacitance and area estimations were able

to use the same estimation data of the number of functional units. However, as the new area estimation routines remove the direct relationship between the number of functional units and their total switched capacitance, new capacitance estimation routines are required. The total switched capacitance for the data-path segment of the RTL design is determined from:

$$C_{\text{total}} = C_{\text{functional-units}} + C_{\text{registers}} + C_{\text{interconnect}} \quad (6.3)$$

The new capacitance estimation routines estimate  $C_{\text{functional-units}}$  and  $C_{\text{registers}}$ ; the estimation of  $C_{\text{interconnect}}$  is unaffected by the new area estimation routines as it still follows the same relationship with area.

#### *6.8.1 Switched Capacitance of Functional Units*

The amount of capacitance switched when performing the operations (add, multiply, etc.) of the DSP operation is determined through analysis of the DFG. The quantity of each operation is used in conjunction with the new switched capacitance values for each operation type (Table 6.2) to determine the overall switched capacitance for each operation type. So the total capacitance of functional units is equal to:

$$C_{\text{functional\_units}} = \sum_{n=1}^m C_i \quad (6.4)$$

where  $m$  is the total number of operations of type  $i$  in the DFG.  $C_i$ , the capacitance of module type  $i$ , is determined from the functional unit hardware models.

### 6.8.2 Switched Capacitance of Registers

The total switched capacitance of the registers in the RTL design is directly proportional to the number of accesses made to registers. Within a scheduled DFG all data transferred from one operation to another across a cycle boundary must be stored in a register [Micheli94]. As the GALOPS systems uses a unity-delay model, where each operation is scheduled in a single control step, the inputs and outputs of all operations cross cycle boundaries. Therefore, each data-net in the DFG implies access to a register, as illustrated in Figure 6.15.

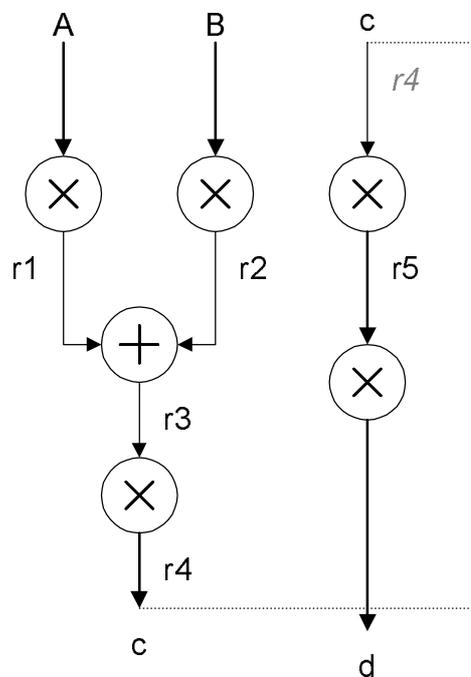


Figure 6.15 Register Accesses in an Example DFG

This register access methodology does not store primary input and output values in registers [Micheli94.]. Therefore, the example in figure 14 requires 5 register accesses. Input ‘c’ is a feedback signal from the output of the DFG; in this case a register is required to transfer the variable back to the inputs.

Once the number of register accesses has been determined the total register capacitance, as a contribution to power, is determined from:

$$C_{registers} = \sum_1^n C_{single\_register} \quad (6.5)$$

where  $n$  is the number of connection nets in the DFG which require a variable to be stored and  $C_{single\_register}$  is the switched capacitance of a single register.

6.9 Summary of Enhanced Fitness/Power Estimation Module

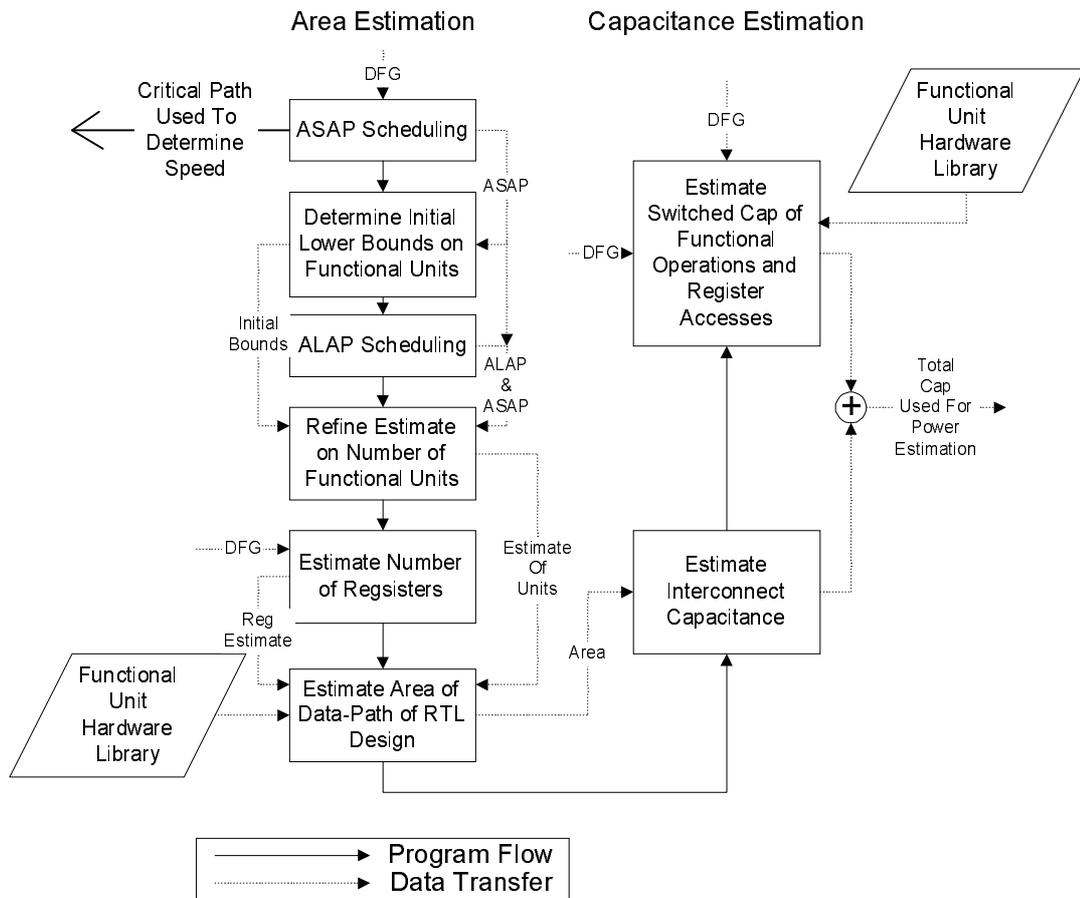


Figure 6.16 Overview of Enhanced Area and Capacitance Estimation Procedures

The new fitness estimation routines incorporate more accurate area estimation techniques built on validated techniques used for exploration and design of systems at the high-level [Potko89, Chan95, Rabaey91a, Rabaey94]. The improved area estimation techniques removed the direct relationship between physical area and switched capacitance; therefore, new switched capacitance estimation routines were implemented. In addition, the capacitance hardware library was improved to incorporate capacitance values that include the consideration of switching activity within each operation type.

In addition to improving the estimate of the interconnect capacitance, the area estimation is invaluable in analysing the effects of the power optimisation process. Figure 6.16 summarises the new area and switched capacitance estimation procedure. The improved estimation routines not only increase the accuracy of the power estimation procedure but also make the analysis more relevant to practical VLSI systems.

## **Chapter 7 – Modifications to the Prototype GALOPS**

Chapter 5 presented the initial implementation of GALOPS, which incorporated a number of standard GA techniques to facilitate an exploration of the low-power design space. GAs have been the subject of intensive research since their introduction, resulting in the development of a number of techniques for improving the efficiency of the search mechanism. This chapter describes the implementation of some of the techniques into the GALOPS tool with the aim of improving both the search efficiency and the results obtained.

### **7.1 Enhanced Mutation Operator (Remove-Pipeline)**

The HLTs used in the prototype version of GALOPS are used to explore the solution space for low-power designs. The retiming and back-retiming HLTs enable complete exploration of the retime search space, as any search move can be undone through application of its reverse transformation.

The pipeline HLT has no such reverse operation; as the application of pipelining in a standard design task rarely involves the removal of inserted pipeline stages. Therefore, a pipeline operation on a chromosome confines all descendants of that chromosome to be pipelined designs. This could result in the search prematurely settling in the pipelined area of the solution space.

To enable complete exploration of the solution space, GALOPS uses a *reverse-pipeline* operator for the removal of pipelines from pipelined designs. The reverse pipeline operator enables the guided walk through the solution space to

‘backtrack’ into non-pipelined areas. The reverse-pipeline operator is required as without any backtracking capability the GA may not efficiently explore non-pipelined sections of the solution space. The retime and pipeline transformations are now both bi-directional; a quality of basic genetic algorithms as illustrated in Figure 7.1 [Davis91, Holland92, Goldberg89].

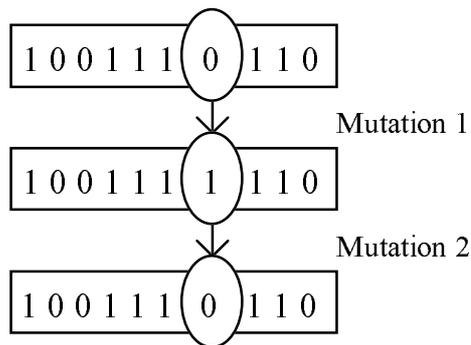


Figure 7.1 Reversible Exploration

The removal of pipelines involves the identification of pipeline stages along with all modifications made to the DFG to insert that pipeline, such as the insertion of cutset delays. The relevant genes are then removed, restructuring the chromosome to remove the pipeline stage. The pseudo-code for the remove-pipeline transformation is shown in Figure 7.2.

```
Algorithm Remove Pipeline
  Passed chromosome DFG to be mutated

  Create a list of all pipelined delays in the DFG
  IF no pipeline delays then remove_pipeline NOT possible
    Exit algorithm

  Randomly select a pipeline_delay in the DFG
  Determine the cutset_points associated with the pipeline_delay
  IF all cutset_elements are NOT delays
    Pipeline cannot be removed
    Exit algorithm
  ELSE
    Remove pipeline_delay from the DFG
    Remove cutset_elements from the DFG

  End IF

End Algorithm Remove Pipeline
```

Figure 7.2 Pseudo-Code for Remove-Pipeline Mutation

An example of the application of the remove-pipeline mutation is shown in Figure 7.3. This example shows the 3 TAP FIR filter with a single pipeline stage. Pipeline delays are denoted with a ‘p’ whereas cutset delays are denoted with a ‘c’. The first constraint of the algorithm, identify pipeline delays, is satisfied so the pipeline delay is selected. The cutset delays are then determined with the same routine as that used to insert the pipeline stage. Subsequent mutation and crossover operations to a pipelined DFG (such as retiming mutations) may have produced a DFG where the cutset elements are no longer delays but actual functional operations. Removal of such elements would corrupt the DFG function; therefore, the remove pipeline operation is only applied if it does not corrupt the DFG.

The pipeline delays in this example satisfy the constraints (they are not in loops and can be removed) so the pipeline delay is removed, producing the middle DFG. Removal of the remaining cutset delays then produces the right-hand DFG, a non-pipelined 3 TAP FIR design.

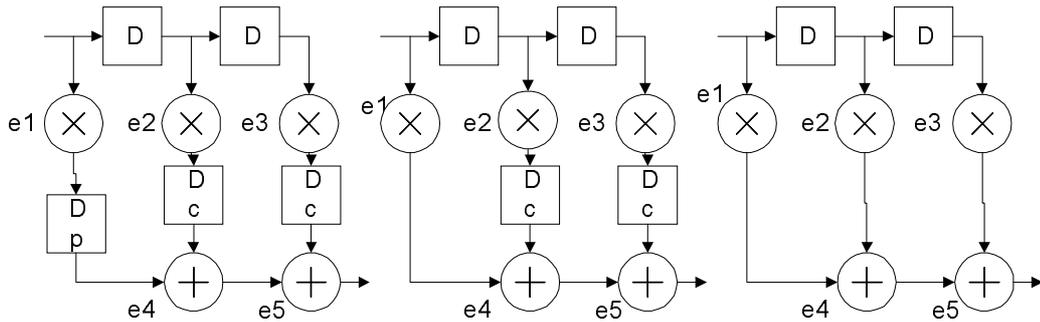


Figure 7.3 Application of Remove-Pipeline Mutation

## 7.2 The Elitism Mechanism for Chromosome Selection

As discussed in section 5.3 the traditional chromosome selection techniques can be modified to improve the results obtained with the GA, such as the use of fitness linearisation. This section introduces another selection scheme known as elitism.

The elitism mechanism [Hanc95] (sometimes known as Top-n selection) is not a replacement for the traditional FPS scheme; it is used in addition to the standard GA selection method. The probabilistic selection method could result in the current best solution not being selected for reproduction, hence the loss of the chromosomes valuable genetic data. In addition, as the reproduction scheme is used to select chromosomes for genetic modification the best chromosome could also be lost through mutation or crossover. Hence, the GA loses the chance to exploit the current best information.

The elitist strategy ensures that a specified portion of the next generation is created from reproductions (copies) of the current best solution. The basic application of the elitist operator makes a single copy of the best solution. However, the number of copies of the best solution can be specified, with different values having different effects on the synthesis process. The more copies of the current

best solution available, the greater the exploitation of the properties of that solution. However, too many copies and the exploration process stagnates around the current best solution. Therefore, an elitism application rate has to be determined to trade-off between the benefits of exploration and exploitation.

### 7.2.1 Experimental Analysis of Effect of Elitism

Experimental analysis is used to determine the actual elitism application rate to be used in the GALOPS system. The values analysed are: 1, 1%, 2%, 5%, 10%, and 20%. This covers a wide range of cases from the simplest of only applying elitism to a single chromosome, to applying elitism to 20% of the population (i.e. 20% of the next generation are copies of the current best solution). Typically, elitism is applied at a much lower rate than 20%; therefore, analysis of the results with the selected application rates will yield values for a wide range of cases. In addition, the case of no elitism (elitism set to 0) is also analysed for comparison.

The elitism application rates were tested on the 5 benchmark DFG designs described in section 5.7, using the mutation and crossover application rates determined in section 5.8. Each elitism rate is applied to each of the 5 benchmark designs 20 times to minimise the effects of the stochastic nature of the GA on the elitism analysis. The analysis requires 600 complete runs of the GA ( $20 \times 6 \times 5$ ).

Each run of the GA produces a design with the best found power consumption, reported as a percentage of the original designs power consumption. The mode value from all 20 runs of each design is taken as the optimum power consumption determined for that design. Table 7.1 lists the mode power consumption for each design with varying elitism application, quoted to 4 significant figures.

Batch	Elitism	Benchmark Design Power Consumption				
		FIR3	FIR8	LAT2	AVEN8PA	AVEN8DI
0	0	23.54	18.29	59.44	23.79	20.13
1	1	23.54	15.76	59.44	23.82	20.04
2	1%	23.54	15.76	59.44	23.82	20.01
3	2%	23.54	15.76	59.44	23.79	20.01
4	5%	23.54	15.86	59.44	23.82	20.04
5	10%	23.54	16.04	59.44	23.79	20.04
6	20%	23.54	18.25	59.44	23.89	23.82

Table 7.1 Mode Power-Consumption for Benchmark Designs with Varying Elitism Parameter

The table illustrates that FIR3 and LAT2 are unaffected by the variation in the elitism parameter. These are relatively less complex designs, therefore optimum synthesis of these designs is essentially independent of fluctuations in GA parameters. However, the more complex designs (FIR8, AVEN8PA and AVEN8DI) are affected by the elitism parameter, as illustrated in Figure 7.4.

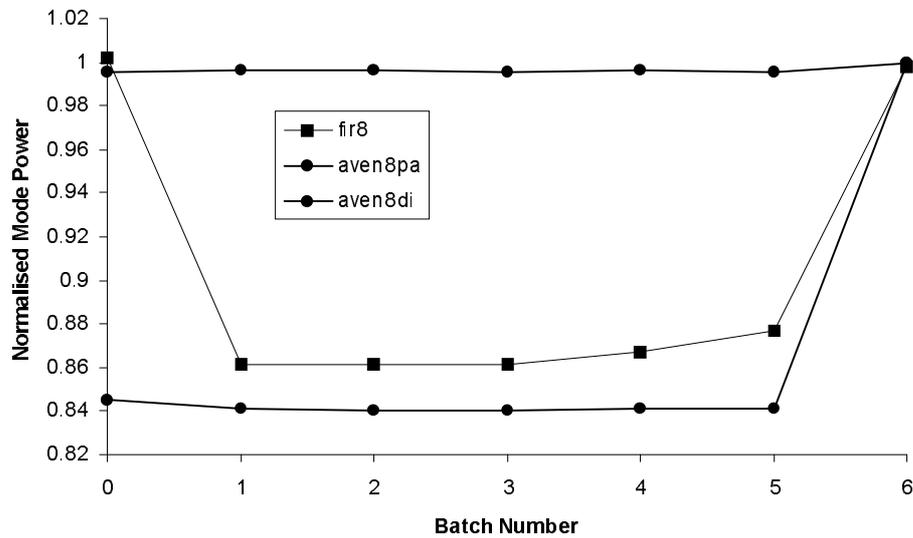


Figure 7.4 Effect of Increasing Elitism Application on Power Consumption of Benchmark Designs

To enable comparison of the different benchmark designs, the power consumption for each design is normalised with respect to the worst (highest) found for that design across all elitism parameter analyses in Table 7.1. Therefore, a lower y-axis value in this graph indicates a superior solution. The graph illustrates that the application of elitism results in an increase in the quality of the solutions i.e. produces designs with lower power consumption. However, increases beyond a certain level of elitism results in the GA producing poorer solutions with higher power consumption i.e. the GA is more likely to become stuck in local optima.

The minimum point for all curves corresponds to an elitism parameter of 2% i.e. batch 3. Although some of the curves are also at the minimum value in other batches, batch 3 is the only batch where all curves are at a minimum. Therefore, batch 3 produces the best overall results for the benchmark designs.

Figure 7.5 illustrates the effect of elitism on the average number of generations required to determine the best solution.

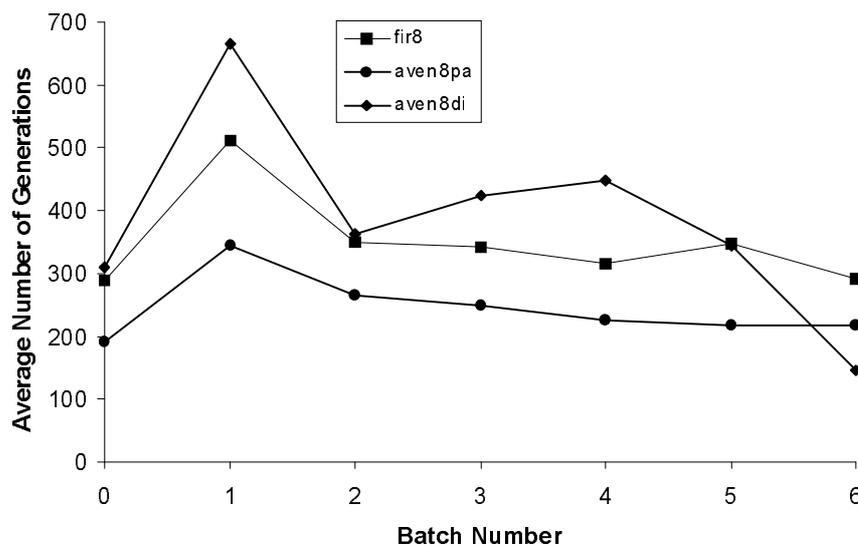


Figure 7.5 Effect on GA Iteration Length of Increasing Elitism Application

Initial application of elitism results in an increase in the required number of generations to determine the best solution. This indicates that the introduction of elitism slows down the convergence rate i.e. the GA takes longer to settle on an optimum solution. However, in the case of the *fir8* and *aven8di* designs, the increase in the number of generations also corresponds with an increase in the quality of the best solution. The GA is exploiting the current best information to determine better solutions.

The graph indicates a general trend of increased application of elitism reducing the number of generations required in determining the best solution. This implies that the higher the rate of elitism, the faster the population converges on the best-found solution. As the results in Table 7.1 show, this fast convergence results in the determination of sub-optimal solutions when compared with the results for lower rates of elitism. The increased number of copies of the best solution in each generation results in the population becoming saturated with solutions very similar to the best solution.

The selected application rate that produced the best results, batch 3, lies between the two extremes of maximum and minimum numbers of generations. This implies that the selected application rate is successful because it makes a good trade-off between exploiting the information in the current search area and exploring the solution space for new search areas.

### 7.3 Selection of Operator Application Rates

In section 5.8 it was noted that GA operators are applied at specified rates, the actual rates can have an effect on the success of the GA. The prototype version of GALOPS uses GA application rates determined using a relatively small set of

experiments. The results obtained by a GA can be sensitive to the applications rate selected. Therefore, it is necessary to perform a comprehensive examination of the application rates. This chapter reviews techniques used to determine an optimum set of GA operation parameters before describing the use of a particular method, the Taguchi method, to optimise the GA parameters in order to improve the success of the GA.

For each implementation of a GA a set of parameter values is defined which specifies the operator probabilities i.e. the probability of a particular mutation, crossover or reproduction (the genetic operators) being applied to a chromosome. The correct setting of operator probabilities is very important to produce an effective algorithm that efficiently utilises the available computing resources to determine a good solution. Unfortunately, the determination of an optimum set of parameters is far from a trivial task. This is due, in part, to the fact that Genetic Algorithms are used in a wide range of problem solving applications; therefore, it is highly unlikely that an ideal set of parameters will suit all GA applications.

One of the earliest investigations into determining a set of ideal operator probabilities was addressed in [DeJong]. However, in this case the problem is simplified by only considering bit-string chromosome representations and a limited number and type of genetic operators. In recognition of the difficulty of determining a set of parameters suited to a wide range of GA applications, the problem was further simplified by analysing the GA on a restricted test-suite of applications. The simplifications used limit the application of the resulting operator probabilities to similar types of GA with binary-string representation [Davis89].

Many real-world optimisation problems (such as GALOPS) have a set of complex genetic operators and do not use binary-string representation [Davis89]. For such GAs it is highly unlikely that a generic set of operator probabilities exist which encompass all possible implementations of real-world problems. Therefore, the optimum set of parameters must be tuned for each GA through analysis and investigation of its properties.

The use of ‘meta-level’ GAs is one technique that has been suggested for optimum parameter tuning of GAs [Gref86]. A meta-level GA attempts to determine an optimum set of parameters for a target-GA by using another GA to perform the optimisation process. Potential sets of operator probabilities are encoded as chromosomes for the meta-level GA to optimise. Fitness evaluation of each chromosome involves executing the target-GA with the set of proposed probabilities, assigning a score to the chromosome based on the observed performance of the GA.

Such a ‘meta-level’ GA will typically require many thousands of complete target-GA optimisations [Turton94]. Each target-GA optimisation will typically require many thousands of evaluations of the target objective function. Therefore, meta-level GAs are typically prohibitive in terms of computation time.

An alternative to the use of meta-level GAs or pre-computed parameter values is the use of the Taguchi Method [Turton94]. The Taguchi method is a quality engineering methodology developed by Dr. Geneci Taguchi [Ranjit90]. It was initially developed for use in manufacturing industry to improve the quality of the production process. One of the components of the Taguchi method is the determination of operating parameters to produce the best quality solution. The Taguchi method uses a defined set of orthogonal arrays to plan a series of

experiments for the determination of the best parameter settings. An example of an orthogonal array is given in Table 7.2.

Experiment	Parameter		
	A	B	C
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Table 7.2 Example Orthogonal Taguchi Array

Each row in the array corresponds to a specific experiment. Each column defines a particular parameter to be optimised; therefore, each row in the array defines a set of values for all of the parameters of an experiment. The settings are described as ‘levels’; each level corresponds to a specified real world value for that parameter, determined by the nature of the problem.

The advantage of the Taguchi method is its reduction in the number of experiments required to efficiently optimise the set of parameters. In the example of Table 7.2, there are 3 variables, each with 2 possible levels. A standard experimental procedure to determine the optimum values would require the testing of all possible combinations (factorial experiment design). Therefore, 8 ( $2^3$ ) experiments would be required instead of the 4 needed for the Taguchi method. For larger numbers of parameters with even more levels, the reduction in the required number of experiments is greater. The statistical validity of the technique is not analysed here; the reader is referred to [Ranjit90, Turton94] for a more thorough examination of the Taguchi method.

After completion of the experiments, the best level for each parameter is determined through analysis of the results. The example array and a set of results are presented in Table 7.3 to illustrate this process.

<b>Experiment</b>	<b>Parameter</b>			<b>Result</b>		
	A	B	C	<i>No. 1</i>	<i>No. 2</i>	<i>No. 3</i>
<b>1</b>	1	1	1	4	5	4
<b>2</b>	1	2	2	3	6	5
<b>3</b>	2	1	2	7	7	6
<b>4</b>	2	2	1	4	5	5

Table 7.3 Set of Results Produced from Taguchi Experiments

As the table shows, each experiment is performed a number of times under the same conditions. This is of particular importance in the case of a GA as the probabilistic nature of the GA makes it possible that different results will be produced under the exact same set of operating conditions. Therefore, a GA is typically run a number of times to analyse its performance.

For each experiment, the average of the results can be used to evaluate the result. However, Taguchi presented a further refinement to the Taguchi method through the introduction of the concept of signal-to-noise ratio. The signal-to-noise analysis is more robust than the average as it determines which set of parameters produce not only the best result, but also the smallest variation around that result. The formula for determining the signal-to-noise ratio is shown in (7.1),

$$\text{Signal To Noise} = -10 \log_{10} \left( \frac{\sum_{i=1}^n y_i^2}{n} \right) \quad (7.1)$$

where  $y_i$  is the result of the  $i^{th}$  trial and  $n$  is the number of trials per experiment. This formula is used when the aim is to minimise the result  $y_i$ . Table 7.4 shows the revised Taguchi array, incorporating the signal-to-noise values for each experiment.

Experiment	Parameter			Result			Signal-To-Noise
	A	B	C	No. 1	No. 2	No. 3	
1	1	1	1	4	5	4	-12.79
2	1	2	2	3	6	5	-13.68
3	2	1	2	7	7	6	-16.50
4	2	2	1	4	5	5	-13.42

Table 7.4 Signal-To-Noise Ratio Analysis for Experiment Results

Each parameter is analysed in turn to determine the best level. An average of the S/N ratios for a particular level of the parameter is calculated from the table. The following example shows this calculation step for parameter A.

**Parameter A – Level 1**

Exp1: -12.79, Exp2: -13.68  
Average = -13.235

**Parameter A – Level 2**

Exp3: -16.50, Exp4: -13.42  
Average = -14.96

This information can then be plotted on a graph as shown in Figure 7.6.

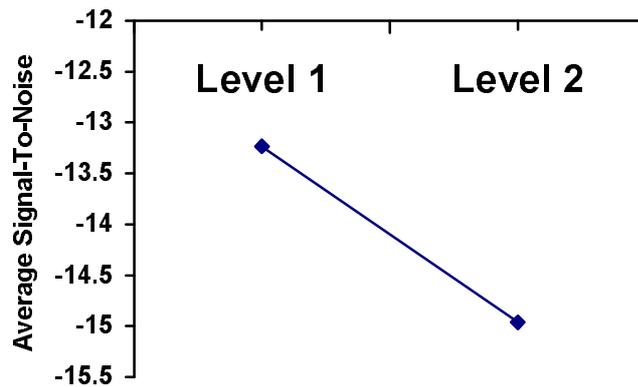


Figure 7.6 Example Graph Showing the Effect of Parameter A

The graph shows that level 1 produces the highest average S/N value, it is therefore chosen as the best setting for parameter A (assuming lowest result is best in this problem). The actual value of the parameter is determined by matching the level with the predefined set of real world values. This process is repeated for each parameter to produce the set of values that it is expected will result in the best system performance. The resulting set of values may not actually have been analysed in an experiment (there may not have been such a combination in the Taguchi array) Therefore, a validation step is often performed to verify the performance of the values.

### 7.3.1 Taguchi Optimisation of GA Application Rates

The current version of GALOPS has 7 possible reproduction choices (6 transformations and crossover), each of which has to be assigned a specific application probability. Table 7.5 lists all of the choices (parameters) with a set of possible application rates (values). The values are presented as percentages, the percentage probability of that operator being selected.

<b>Parameter</b>	<b>Values (%)</b>
Retime	1, 2, 5, 10, 20
Back-Retime	1, 2, 5, 10, 20
Auto-Pipe	1, 2, 5, 10, 20
Pipeline	1, 2, 5, 10, 20
Remove-Pipeline	1, 2, 5, 10, 20
Unfold	1, 2, 5, 10, 20
Crossover	1, 2, 5, 10, 20

Table 7.5 Possible Application Rates of Each Operator

Each variable has four possible values, from 1% to a maximum of 20%. If each operator were applied at the maximum value of 20%, it would result in a total application rate of 140%. Therefore, some combinations (such as all rates at 20%) are not investigated where it would lead to unfeasible application rates.

The table of possible values is simplified by the removal of the unfold parameter, as described in section 5.8. To summarise, the unfolding principle limits the application of the unfolding transformation so its application rate is not critical to GA performance.

The selection of crossover rate is simplified by reducing the number of values to 10% and 20%. This ensures that crossover is applied at a high rate without limiting the potential applications of the mutation operators. GALOPS is not a conventional GA as the mutation operators perform a lot of the exploration, therefore the crossover rate is not as high as in a typical GA application.

The optimisation process consists of 6 variables, 5 of which have 5 possible levels; crossover has two possible levels. When using a Taguchi array for such a process, depending on the array used there may be a requirement for orthogonality between the variables. For this reason, the  $L_{50}$  array is selected as it allows a large degree of interdependence between the parameters. This optimisation process

requires the use of a modified form of the  $L_{50}$  array, which has 50 experiments. The annotated array for the Taguchi optimisation of the GALOPS operating parameters is shown in Table 7.6. The application rates are listed as percentages i.e. the percentage probability of each operator being selected. The total percentage is also illustrated for each experiment to ensure that no sum of application rate percentages is greater than 100%. If the sum were greater than 100% the GA would attempt to apply the operators to more than 100% of the current generation, an unfeasible operation. Therefore, the elitism application rate is also included in this table for the purposes of calculating the total application rates.

The GALOPS tool is a flexible DSP synthesis tool which can process a wide range of signal processing applications. In this case, Taguchi optimisation of the GA parameters for one particular DFG topology may result in the GA being optimised for that particular design. The Taguchi optimisation is therefore applied to two DFG designs, the direct and parallel forms of 8<sup>th</sup> order Avenhaus filter (AVEN8DI and AVEN8PA respectively). The direct and parallel implementations are significantly different DFGs, requiring a different set and order of transformations for optimisation. In addition, both designs contain recursive and non-recursive sections that require different methods of optimisation.

The optimisation process is performed 5 times for each experiment in the  $L_{50}$  array. The results for the AVEN8DI and AVEN8PA design are shown in Table 7.7 and Table 7.8 respectively, along with the calculated S/N ratio for each experiment. The %Power columns refer to the power of the optimised design expressed as a percentage of the power of the original non-optimised design.

Experiment	Crossover	Retime	Back Retime	Auto Pipe	Pipeline	Remove Pipe	Elitism	Total
1	10	1	1	1	1	1	2	17
2	10	1	2	2	2	2	2	21
3	10	1	5	5	5	5	2	33
4	10	1	10	10	10	10	2	53
5	10	1	20	20	20	20	2	93
6	10	2	1	2	5	10	2	32
7	10	2	2	5	10	20	2	51
8	10	2	5	10	20	1	2	40
9	10	2	10	20	1	2	2	47
10	10	2	20	1	2	5	2	42
11	10	5	1	5	20	2	2	45
12	10	5	2	10	1	5	2	35
13	10	5	5	20	2	10	2	54
14	10	5	10	1	5	1	2	34
15	10	5	20	2	10	20	2	69
16	10	10	1	10	2	20	2	55
17	10	10	2	20	5	1	2	50
18	10	10	5	1	10	2	2	40
19	10	10	10	2	20	5	2	59
20	10	10	20	5	1	10	2	58
21	10	20	1	20	10	5	2	68
22	10	20	2	1	20	10	2	65
23	10	20	5	2	1	20	2	60
24	10	20	10	5	2	1	2	50
25	10	20	20	10	5	2	2	69
26	20	1	1	1	10	20	2	55
27	20	1	2	2	20	1	2	48
28	20	1	5	5	1	2	2	36
29	20	1	10	10	2	5	2	50
30	20	1	20	20	5	10	2	78
31	20	2	1	2	1	5	2	33
32	20	2	2	5	2	10	2	43
33	20	2	5	10	5	20	2	64
34	20	2	10	20	10	1	2	65
35	20	2	20	1	20	2	2	67
36	20	5	1	5	5	1	2	39
37	20	5	2	10	10	2	2	51
38	20	5	5	20	20	5	2	77
39	20	5	10	1	1	10	2	49
40	20	5	20	2	2	20	2	71
41	20	10	1	10	20	10	2	73
42	20	10	2	20	10	20	2	84
43	20	10	5	1	5	1	2	44
44	20	10	10	2	2	2	2	48
45	20	10	20	5	1	5	2	63
46	20	20	1	20	2	2	2	67
47	20	20	2	1	5	5	2	55
48	20	20	5	2	10	10	2	69
49	20	20	10	5	20	20	2	97
50	20	20	20	10	1	1	2	74

Table 7.6 Taguchi Array for Optimisation of Genetic Operator Application Rates

## Chapter 7 – Modifications to the Prototype GALOPS

Experiment	%Power 1	%Power 2	%Power 3	%Power 4	%Power 5	S/N Ratio
1	20.10	25.62	25.55	20.13	25.44	27.19
2	25.59	20.20	20.13	25.52	20.10	26.80
3	20.17	20.17	20.17	25.43	20.04	26.42
4	20.20	20.20	20.20	20.13	20.20	26.10
5	20.04	20.08	20.17	20.10	20.10	26.06
6	20.13	20.20	20.10	34.34	20.10	26.69
7	20.33	20.22	20.17	20.26	20.13	26.12
8	20.04	20.04	20.29	20.33	20.20	26.10
9	20.15	20.24	25.46	20.15	20.10	26.43
10	20.04	20.06	20.04	20.17	20.13	26.06
11	20.26	20.22	20.17	20.10	20.17	26.10
12	20.10	20.08	20.20	20.13	20.17	26.08
13	20.33	20.06	20.01	20.33	20.10	26.09
14	20.04	20.15	20.04	20.24	20.10	26.07
15	20.06	20.04	20.08	20.04	20.13	26.05
16	20.10	20.17	20.04	20.20	20.13	26.08
17	20.13	20.15	20.06	20.06	20.10	26.06
18	20.17	20.01	20.17	20.20	20.22	26.09
19	20.06	20.04	20.15	20.04	20.15	26.06
20	20.15	20.17	20.08	20.24	20.15	26.09
21	20.11	20.15	20.17	20.08	20.08	26.07
22	20.20	20.06	20.06	20.06	20.08	26.06
23	20.01	20.04	20.08	20.08	20.01	26.04
24	20.04	20.08	20.04	20.06	20.01	26.04
25	20.04	20.01	20.01	20.15	20.13	26.05
26	25.52	20.38	20.15	25.68	20.31	26.84
27	20.24	20.31	20.22	20.06	20.26	26.12
28	20.22	25.55	25.52	20.22	20.13	26.81
29	20.17	20.06	20.20	20.13	20.13	26.08
30	20.08	20.13	20.24	20.04	20.01	26.06
31	20.13	20.20	20.17	20.22	20.10	26.09
32	20.15	20.20	20.15	25.47	20.08	26.42
33	20.26	25.52	20.15	20.10	20.24	26.44
34	20.24	20.06	20.13	20.04	20.06	26.07
35	20.06	20.10	20.17	20.29	20.15	26.09
36	20.04	20.17	20.31	20.17	20.24	26.10
37	20.24	20.17	20.38	20.06	20.36	26.12
38	20.15	20.13	20.06	20.10	20.17	26.07
39	20.20	20.24	20.04	20.08	20.20	26.09
40	20.06	20.22	20.29	20.15	20.06	26.09
41	20.04	20.06	20.22	20.26	20.29	26.10
42	20.06	20.10	20.13	20.08	20.10	26.06
43	20.40	20.10	20.10	20.04	20.13	26.09
44	20.08	20.06	20.17	20.17	20.17	26.08
45	20.13	20.06	20.15	20.15	20.08	26.07
46	20.17	20.22	20.06	20.17	20.08	26.08
47	20.15	20.08	20.17	20.06	20.13	26.07
48	20.04	20.17	20.06	20.04	20.04	26.05
49	20.01	20.06	20.13	20.06	20.10	26.05
50	20.04	20.13	20.13	20.13	20.04	26.06

Table 7.7 Taguchi Results for AVEN&DI DFG

## Chapter 7 – Modifications to the Prototype GALOPS

Experiment	%Power 1	%Power 2	%Power 3	%Power 4	%Power 5	S/N Ratio
1	23.84	23.89	23.84	23.79	23.82	27.54
2	23.92	23.92	23.82	23.85	23.87	27.56
3	23.92	23.87	23.84	23.87	23.84	27.56
4	23.92	23.89	23.89	23.87	23.87	27.56
5	23.89	23.87	23.82	23.92	23.89	27.56
6	23.87	23.82	23.82	23.79	23.89	27.54
7	23.89	23.79	23.82	23.82	23.82	27.54
8	23.82	23.84	23.87	23.84	23.87	27.55
9	23.84	23.84	23.79	23.89	23.84	27.55
10	23.79	23.89	23.89	23.89	23.89	27.56
11	23.84	23.87	23.89	23.79	23.82	27.55
12	23.74	23.79	23.84	23.82	23.82	27.53
13	23.87	23.74	23.79	23.84	23.79	27.53
14	23.82	23.82	23.89	23.89	23.77	27.54
15	23.89	23.87	23.89	23.89	23.89	27.56
16	23.82	23.77	23.82	23.82	23.79	27.53
17	23.82	23.84	23.79	23.82	23.79	27.54
18	23.82	23.82	23.84	23.82	23.82	27.54
19	23.79	23.74	23.79	23.82	23.84	27.53
20	23.84	23.79	23.89	23.82	23.79	27.54
21	23.84	23.87	23.89	23.77	23.84	27.55
22	23.82	23.79	23.82	23.84	23.84	27.54
23	23.79	23.79	23.74	23.82	23.89	27.53
24	23.79	23.77	23.77	23.82	23.79	27.53
25	23.77	23.82	23.79	23.77	23.72	27.52
26	23.84	23.92	23.79	23.77	23.87	27.54
27	23.87	23.87	23.99	23.94	23.84	27.57
28	23.84	23.87	23.84	23.77	23.79	27.54
29	23.82	23.87	23.89	23.89	23.79	27.55
30	23.84	23.84	23.87	23.89	23.92	27.56
31	23.89	23.84	23.79	23.82	23.84	27.54
32	23.82	23.87	23.84	23.79	23.87	27.54
33	23.89	23.79	23.89	23.89	23.92	27.56
34	23.87	23.84	23.89	23.79	23.89	27.55
35	23.87	23.92	23.94	23.94	23.87	27.57
36	23.87	23.82	23.87	23.87	23.82	27.55
37	23.82	23.79	23.77	23.82	23.82	27.53
38	23.84	23.77	23.82	23.89	23.84	27.54
39	23.84	23.79	23.74	23.77	23.77	27.52
40	23.79	23.79	23.79	23.82	23.84	27.53
41	23.87	23.82	23.87	23.87	23.79	27.55
42	23.79	23.84	23.79	23.79	23.82	27.53
43	23.82	23.74	23.82	23.82	23.82	27.53
44	23.79	23.82	23.77	23.79	23.74	27.52
45	23.77	23.82	23.77	23.74	23.84	27.53
46	23.77	23.77	23.84	23.82	23.82	27.53
47	23.82	23.82	23.89	23.82	23.79	27.54
48	23.79	23.82	23.84	23.79	23.82	27.54
49	23.79	23.82	23.82	23.82	23.82	27.54
50	23.82	23.79	23.82	23.74	23.82	27.53

Table 7.8 Taguchi Results for AVEN8PA

The data in these tables is used to determine the average S/N ratios for each level of each parameter i.e. the application rate for each transformation. This data, for both DFGs, is presented in Table 7.9 and Table 7.10.

Transformation	Application Rate				
	1%	2%	5%	10%	20%
Crossover				-65.66	-65.50
Retime	-26.56	-26.37	-26.09	-26.08	-26.06
Back Retime	-26.46	-26.23	-26.27	-26.12	-26.07
Automatic Pipeline	-26.31	-26.31	-26.28	-26.14	-26.12
Pipeline	-26.33	-26.22	-26.32	-26.18	-26.08
Remove Pipeline	-26.21	-26.33	-26.12	-26.27	-26.22

Table 7.9 S/N Ratio Analysis for AVEN8DI DFG

Transformation	Application Rate				
	1%	2%	5%	10%	20%
Crossover				-68.86	-68.86
Retime	-27.55	-27.55	-27.54	-27.53	-27.53
Back Retime	-27.54	-27.54	-27.54	-27.54	-27.55
A-Pipe	-27.54	-27.54	-27.54	-27.54	-27.54
Pipeline	-27.54	-27.54	-27.54	-27.55	-27.55
Rem-Pipe	-27.54	-27.54	-27.54	-27.54	-27.54

Table 7.10 S/N Ratio Analysis for AVEN8PA DFG

The tables illustrate that the higher rate of crossover (20%) produces the largest S/N noise ratio. For the other transformations the graphs of S/N versus Application Rate are plotted in Figure 7.7 and Figure 7.8 for the AVEN8DI and AVEN8PA respectively.

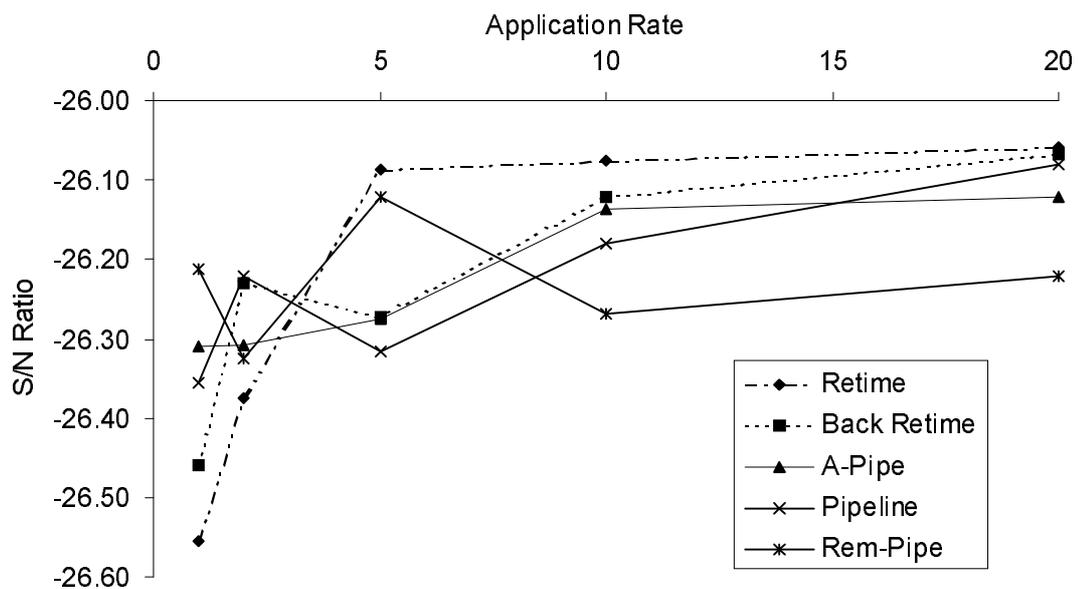


Figure 7.7 Plot of S/N Ratios for AVEN8DI DFG

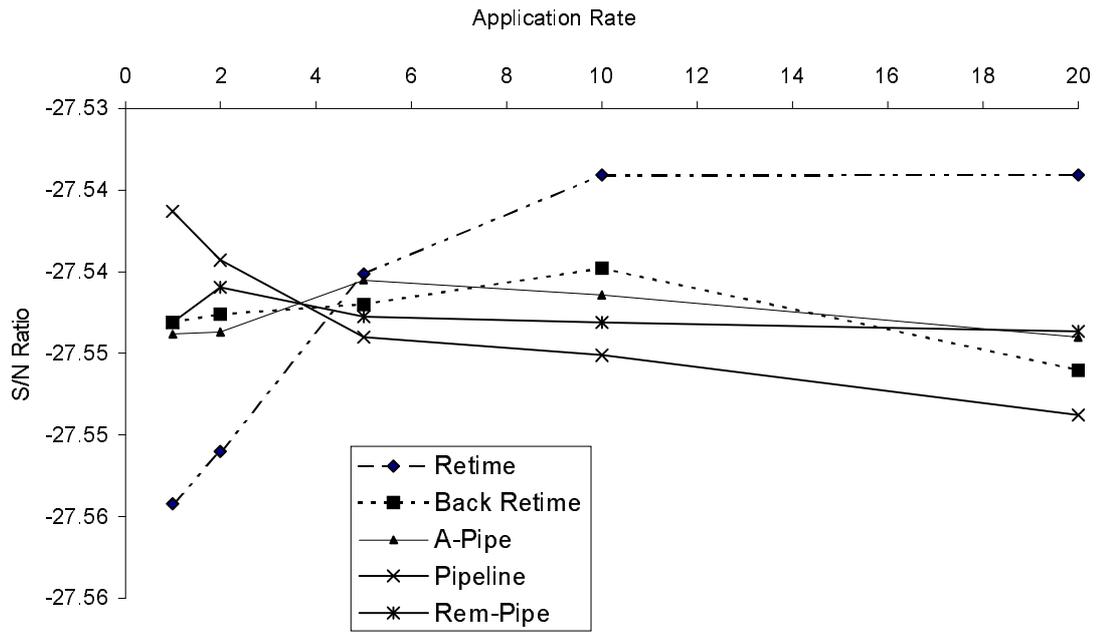


Figure 7.8 Plot of S/N Ratios for AVEN8PA DFG

For each transformation, the application rate that gives the largest S/N ratio is the best application rate for that transformation; the selected rates are listed Table 7.11.

Transformation	Application Rate	
	AVEN8DI Result	AVEN8PA Result
Retime	20%	20%
Back Retime	20%	10%
Automatic Pipeline	20%	5%
Pipeline	20%	1%
Remove Pipeline	5%	2%

Table 7.11 Selected Application Rates from Taguchi Analysis of Two DFG Designs

The table highlights the difficulty in determining an optimum set of parameters for a GA intended to optimise a wide range of DFGs with different characteristics. For the pipeline transformation, a high rate of 20% is selected for

the AVEN8DI, but a lower rate of 1% is selected for the AVEN8PA due to the different effect that pipelining has on both DFGs. In addition, the determined application rates for the AVEN8DI design, together with a crossover rate of 20%, would produce a total application rate greater than 100%, clearly an unfeasible set of application rates. There is no ideal solution for selecting the best set of application rates from those presented by the Taguchi analysis. The selected application rates for the GALOPS system are chosen by averaging the values returned by the Taguchi analysis, illustrated in Table 7.12.

<b>Transformation</b>	<b>Application Rate</b>
<b>Retime</b>	20%
<b>Back Retime</b>	15%
<b>Automatic Pipeline</b>	12%
<b>Pipeline</b>	10%
<b>Remove Pipeline</b>	3%

Table 7.12 Selected Application Rates for GALOP System

#### 7.4 Ranking Selection Scheme

The prototype GA presented in chapter 5 uses the Fitness Proportionate Selection (FPS) scheme to select individuals for genetic modification and hence reproduction into the next generation of individuals. The goal of a selection scheme in an optimisation algorithm is to balance the two objectives of exploration of the solution space and exploitation of the current information about that solution space [Goldberg91].

Exploration is achieved by processing a large selection of individuals throughout the evolutionary process, thereby sampling (exploring) a range of points in the solution space. Exploitation aims to determine which regions of the solution

space are worth exploring by combining and modifying the characteristics of known good solutions, exploiting the information stored within the current population.

A totally random search is a good example of pure exploration, where the next set of individuals is created totally at random. The opposite of pure exploration is a hill-climbing algorithm where the next sample point is entirely dependent upon the current-best sample point. Pure exploration will result in an inefficient sampling of the solution space that may require a large amount of time to determine the optimum solution, if it ever does. Pure exploitation will result in premature convergence, where the population of individuals converges around local optimum in the solution space. Premature convergence results in the production of a sub-optimal solution, as the search space has not been sufficiently explored. The ideal combination of exploration and exploitation will result in an optimum solution determined by an efficient exploration of the solution space.

The FPS method attempts to combine the required characteristics of exploration and exploitation. The selection of the better individuals exploits the information stored in the current generation. The probabilistic selection of those individuals, where weaker individuals have some chance of selection, provides the mechanism for reducing the chance of premature convergence.

However, the problem of rapid convergence, where the entire population prematurely converges on a single chromosome or sub-optimal region of the solution space [Baker85, Hanc95], is still a common problem in GAs that use FPS. This leads to the loss of valuable genetic information stored in the population throughout the exploration of the solution space. This produces a stagnation of the search process; the exploration becomes fixed on a particular solution. Such

premature convergence usually results in the GA producing a solution that is sub-optimal.

Premature convergence arises when using FPS due to its method of assigning a probability of selection. The probability is directly proportional to the solutions absolute fitness relative to that of the other solutions in the population. As illustrated in Figure 7.9, this method can suffer from the problem of ‘super-fit’ individuals. A super-fit individual is a chromosome with a fitness significantly above the average fitness of the entire population.

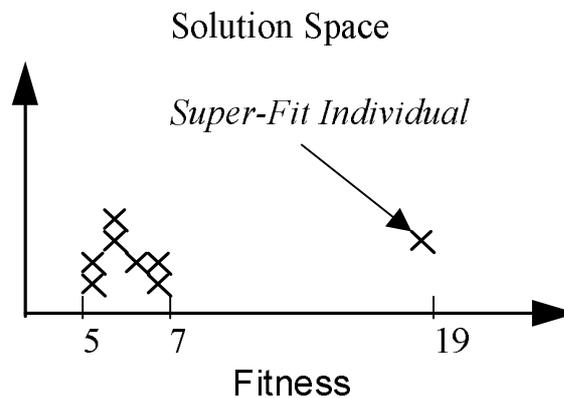


Figure 7.9 Domination of Fitness by Super-Fit Individual Chromosome

In the example of Figure 7.9, the super-fit individual would be allocated the majority of selections, as it would be assigned a large selection probability. The super-fit individual would dominate the population. The population would rapidly converge upon a single chromosome.

Baker [Baker85] first suggested ranking as a possible solution to the problem of premature convergence. The basic idea of ranking is to sort the individuals from best to worst and assign each individual a rank in the ordered

population i.e. the best solution has the highest rank. The selection probability is then determined from a solutions rank as opposed to its absolute fitness value. This dependence of selection on rank, rather than fitness, is an effective means of controlling the dominance of super-fit individuals [Whit89]. The effect of ranking the example population of Figure 7.9 is shown in Figure 7.10.

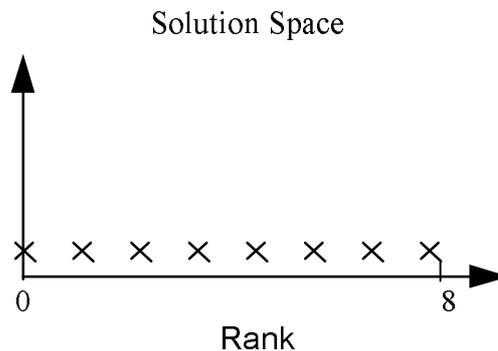


Figure 7.10 Effect of Ranking on Simple Population Representation

The effect of ranking is to spread out the solutions across the rank range, removing the dominance of any particular individual. In effect, the solutions have been mapped from the fitness dimension to the ranked fitness dimension. While the linearisation method can also increase the range of fitness values, it is unsuitable for dealing with the problem of dominant individuals. The application of ranking preserves the relative order of quality of solutions.

The use of ranking also assists in reducing the dependence of selection on the quality of the GAs evaluation function. In the case of GALOPS, the evaluation function is a power estimation technique that is subject to inaccuracies due to the recognised problems of high-level power estimation. Inaccurate power estimations, though useful as a relative measure of quality, effectively introduce noise into a

selection process based on absolute fitness values. One of the features of a GA is that it is a robust system able to handle noisy data [Beasley93, Goldberg89]. However, the ranking procedure acts as an additional noise filter, removing the inaccuracies in the absolute data while preserving the ability to compare the solutions relatively.

The comparative analysis of selection schemes in the literature [Goldberg91, Hanc95], though useful in understanding the implications of each technique, do not categorically specify which particular selection technique will give the best performance. This is partly due to the range of parameters that affect GA performance, such as population size, mutation rates, number of generations, etc. In addition, the specific problem also affects GA performance and the selection of optimum parameters; hence, generalisations for the best selection technique are extremely difficult [Goldberg91]. Therefore, the use of ranking is evaluated for the particular problem in this thesis in comparison with the standard FPS technique used in the prototype version of GALOPS.

The application of ranking involves two distinct processes. First, the chromosomes are sorted in order of fitness; next, the rank of each individual (based on its sorted order) is used in a proportionate selection mechanism. Therefore, ranking has the additional complexity of requiring a sorting step. As noted in [Goldberg91], the complexity of the sorting step can dominate the overall time complexity of the complete selection procedure. Therefore, it is important that a reasonably fast sorting algorithm is used. GALOPS uses a heapsort algorithm [Teuk95a], which has a time complexity of  $O(n \log n)$ .

The assigned rank of each solution is not merely its order in the sorted population, as this could result in solutions with the same fitness having a different

rank. For example, a number of solutions each with the same fitness would have a successively increasing rank. This would unfairly bias the selection procedure to those solutions placed at the top of the pile (of same fitness solutions) during the sorting process. The algorithm used to calculate the rank assigns the same rank value to each of these ‘tied’ fitness values. The assigned rank value is equal to the mean of all the rank values that would have been successively calculated for each ‘tied’ value, as illustrated in Figure 7.11. In this example, all three fitness values of 8 are assigned the same mean rank  $((3+4+5)/3)$ .

<i>Sorted Fitness</i>	2	5	8	8	8	10	17	33
<b>Simple Rank</b>	1	2	3	4	5	6	7	8
<b>Assigned Rank</b>	1	2	4	4	4	6	7	8

Figure 7.11 Assignment of Rank to a Population of Individuals

The algorithm for the implementation of ranking within GALOPS is shown in Figure 7.12.

### Algorithm Rank Population

```
Passed population of chromosomes at start of each generation
Sort the population in ascending order of fitness

Reset initial_rank
FOR all chromosomes in population
  IF fitness of next_chromosome NOT EQUAL to current
    SET rank to initial_rank
    Increment initial_rank
  ELSE
    Reset cumulative_rank
    UNTIL end of range of same fitness values
      Add initial_rank to cumulative_rank
      Increment initial_rank
    END UNTIL
    Calculate mean_rank
    Assign mean_rank to range of same fitness values
  END IF
END FOR
```

End Algorithm Rank Population

Figure 7.12 Pseudo-Code for the Ranking Algorithm

The pseudo-code contains the instructions used to determine a mean rank for a range of chromosomes with the same fitness value. Once the rank values have been generated they are used in place of the fitness values in the FPS scheme i.e. the roulette wheel is now comprised of the rank values rather than the actual fitness values.

### 7.4.1 Experimental Analysis of Effect of Ranking

The benchmark designs (AVEN8PA, AVEN8DI and 8TAPFIR) described in Chapter 5 are used to investigate the effect of the ranking selection scheme. The LAT2 and FIR3 designs are not used because they are relatively simple designs whose synthesis is unaffected by the selection scheme used. Each design was synthesised 20 times to reduce the effect of stochastic errors on the results. Table 7.13 lists the mode power consumption (expressed as a percentage of the original

power consumption) for each of the designs, both for the use of the FPS and Ranked selection schemes.

<b>Design</b>	<b>Mode Power Consumption Percentage</b>	
	<b><i>FPS</i></b>	<b><i>Ranking</i></b>
FIR8	15.95	15.86
AVEN8DI	20.06	20.01
AVEN8PA	23.82	23.82

Table 7.13 Mode Power Consumption Obtained with FPS and Ranking Selection Schemes

The table illustrates that for two of the three designs analysed the power optimisation process is improved with the use of a ranked selection scheme. The AVEN8PA is not improved but the use of ranking does not have a detrimental effect either. Therefore ranking is incorporated as a standard component of the GALOPS tool.

### 7.5 Summary

This chapter has introduced extensions to the genetic framework in an attempt to improve the efficiency and success of the search and optimisation process. Taguchi optimisation, a technique for determining a good set of operating parameters, was applied to the system to determine application rates for the genetic operators. The results illustrated the difficulty in determining a generic set of parameters in a system that processes a wide range of problems with different optimisation requirements.

Ranking selection was investigated and shown to improve the results obtained with GALOPS compared with the FPS scheme. The rank-based selection scheme is therefore incorporated as a component in the GALOPS system.

## **Chapter 8 – Results**

The algorithms used within GALOPS have been described in the previous chapters. It is implemented in the C language and the current version comprises approximately 7000 lines of source code. The system was developed on a Microsoft Windows95 platform using Microsoft Developer Studio. However, the use of ANSI standard C and the exclusion of any machine specific directives (such as the use of windows) was chosen to produce portable code that can be transferred to any system with a C compiler.

This chapter presents the results obtained with the GALOPS tool. The results are illustrated with a set of ten benchmark designs chosen to cover a range of signal-processing applications; these designs are introduced in section 8.1. Section 8.2 presents the operating parameters of GALOPS used to produce the results in this chapter. Section 8.3 presents the results for all ten designs in terms of the power reduction obtained with the GALOPS tool.

### **8.1 Benchmark Circuits**

The benchmark designs presented in section 5.7 were used both to illustrate the effectiveness of the synthesis tool and to investigate improvements and additions to the original tool. In addition to those designs, this chapter presents another 7 benchmark designs for analysing the GALOPS tool. The designs were selected to cover a wide range of signal processing applications of varying

complexity. The set of 10 benchmark designs used to analyse the final version of GALOPS are:

- 8TAPFIR – The 8<sup>th</sup> order FIR Filter. This is an example of a purely non-recursive DSP operation, the DFG consists entirely of feed-forward data transfers.
- AVEN8DI – The direct form of the 8<sup>th</sup> Order Avenhaus Filter
- AVEN8PA – The parallel form of the 8<sup>th</sup> order Avenhaus filter. Both versions of the Avenhaus filter contain a combination of recursive and non-recursive sections in their DFGs.
- DCST – A specific implementation of a discrete cosine and sine transformation algorithm for high-speed VLSI implementation [Chiu94]. This is an example of a non-filtering signal processing application, extensively used in multimedia and data compression.
- BIQUAD3 – A 3<sup>rd</sup> order Biquad filter [Rabaey91a]. This filter is comprised of two distinct stages, both of which contain recursive loops.
- GMLAT4 – A 4 stage Gray-Markel filter [Gela93], a complex filter structure with a large recursive loop on its output stage.
- ELLIP5 – A 5<sup>th</sup> order elliptic wavelet filter. This complex filter contains 40 nodes in its DFG.
- LMS5 – A 5<sup>th</sup> order least mean square algorithm [Gela93].
- VOLTERRA –A Volterra filter. This design presents a complex problem to the synthesis system as it consists of a single large recursive loop.
- ORTH2LAT – A 2<sup>nd</sup> order orthogonal lattice filter [Gela93].

The DFG of each of the benchmark designs is included in Appendix A.

## 8.2 GALOPS Operating Parameters

GALOPS is executed with the operating parameters determined in previous chapters of the thesis –

- **Ranked Based Selection of Chromosomes**
- | <b>Transformation</b> | <b>Application Rate</b> |
|-----------------------|-------------------------|
| Retime                | 20%                     |
| Back Retime           | 15%                     |
| Automatic Pipeline    | 12%                     |
| Pipeline              | 10%                     |
| Remove Pipeline       | 3%                      |
| Crossover             | 20%                     |
| Unfolding (Postponed) | 5%                      |
| Elitism               | 2%                      |

Each design is synthesised 30 times to reduce the effects of the stochastic nature of the GA on the presented results for each design. The stopping criteria for the GA is based on a timeout parameter; if no improvement in the fittest solution is obtained for 500 generations then the GA is considered to have found its ‘best’ solution and the synthesis process is stopped. The best solution found over all generations is stored as the best solution determined by the GA for that particular design.

## 8.3 Results for Ten Benchmark Designs

As discussed previously in chapter 5, the unfolding transformation is applied later in the evolutionary process to attempt to optimise designs produced with the other transformations. Application of unfolding produces large designs that effectively are parallel versions of the original. Therefore the results are presented

for both the unfolding and non-unfolding case, to illustrate the power reduction that can be obtained without creating a parallel processing form of the algorithm.

Table 8.1 lists the results obtained for the ten benchmark designs using the operating parameters listed in section 8.2.

Benchmark Design	Without Unfolding			With Unfolding		
	Power (% Of Original)	Area (% Of Original)	Supply Voltage (Volts)	Power (% Of Original)	Area (% Of Original)	Supply Voltage (Volts)
8TAPFIR	15.76	441.2	1.5	13.37	795.8	1.5
AVEN8DI	20.06	304.2	2.0	19.64	524.8	2.0
AVEN8PA	23.82	155.8	2.3			
DCST	30.68	102.9	2.8	28.68	150.8	2.8
BIQUAD3	30.89	104.3	2.8			
GMLAT4	32.71	152.7	2.8	32.39	382.4	2.8
ELLIP5	69.86	58.98	4.3			
LMS5	88.90	62.73	5.0	82.55	117.0	5.0
VOLTERRA	95.56	69.06	5.0			
ORTH2LAT	100.0	100.0	5.0	97.55	162.1	5.0

Table 8.1 Overall Power Reduction and Area Increase for Benchmark Designs

The table is split into two halves, the left half shows the results obtained without unfolding and the right half shows the results obtained with unfolding. The shaded entries in the unfolding section of the table denote benchmark designs that did not benefit from the application of unfolding i.e. the unfolded designs did not consume less power than the non-unfolded designs.

The Power column refers to the estimated power consumption of the best design found throughout the evolutionary search process. It is expressed as a percentage of the original designs’ power consumption. The percentage illustrates the improvement in power consumption obtained by the GALOPS synthesis tool, rather than presenting absolute values estimated using the high-level power estimation module. As discussed previously, the techniques used in the high-level

power analysis module are not intended to provide absolute power consumption statistics; rather, the power analysis enables the comparison of power consumption between two designs. In this case that comparison, between the optimised and the original design, is expressed as a percentage.

The Area column refers to the area of the best-found design with the power consumption reported in the power column. This is also expressed as a percentage to illustrate the increase or decrease in size in relation to the original design. The Supply Voltage column presents the estimated supply voltage that the best-found design requires in order to meet the power consumption reported in the power column, expressed in Volts.

The following sections discuss the results obtained with each of the benchmark designs.

### 8.3.1 8TAPFIR

The 8TAPFIR design reports the largest power reduction, the best-found design consumes 15.763% of the original design for the non-unfolded case. This was obtained through the use of pipelining and retiming to reduce the critical path from an initial 8 elements to 1 element; this results in an 8-fold increase in speed which can be traded for a reduced supply voltage of 1.469V. The implications of operating at low voltages are discussed in section 8.5. of this chapter. The estimated area of the datapath component is approximately 4.4 (440%) times the original design. The initial design was estimated to require 2 multipliers and 1 adder, whereas the low-power design requires 8 multipliers and 7 adders, 6 extra multipliers and 6 extra adders. The reduction in power consumption was obtained through exploiting the available concurrency in the graph, increasing the parallelism

through pipelining. This increases the speed of the design (hence allowing a reduced voltage) but also increases the amount of operations that must be performed in parallel and hence increases the required amount of hardware.

The unfolded 8TAPFIR design consumes 13.37% of the power of the original, a further reduction of approximately 2% compared to the non-unfolded design. The area is approximately 800% that of the original design, primarily due to the requirement for 16 multipliers and 5 adders. The results illustrate that the unfolded design is larger and requires a higher supply voltage than the non-unfolded design but consumes less power. This is possible as the power analysis process considers the average power consumed per sample. The unfolded version is processing 4 samples in parallel, therefore its effective switched capacitance is  $\frac{1}{4}$  of its actual switched capacitance (as explained previously in section 5.6). While the unfolded version processes 4 samples in parallel, its estimated area is not 4 times that of the non-unfolded version (it is approximately twice the non-unfolded version). Therefore the interconnect component of switched capacitance (related to area) has also not increased 4-fold, resulting in an overall reduction in effective switched capacitance.

While unfolding does produce further power reduction this example illustrates that there exists a trade-off between area and power. The unfolded version only produces a further 2% reduction in power yet it is twice the size of the non-unfolded version.

Figure 8.1 illustrates the performance of the GA for a typical synthesis run; the figure illustrates the best fitness in the population and the average fitness of the whole population.

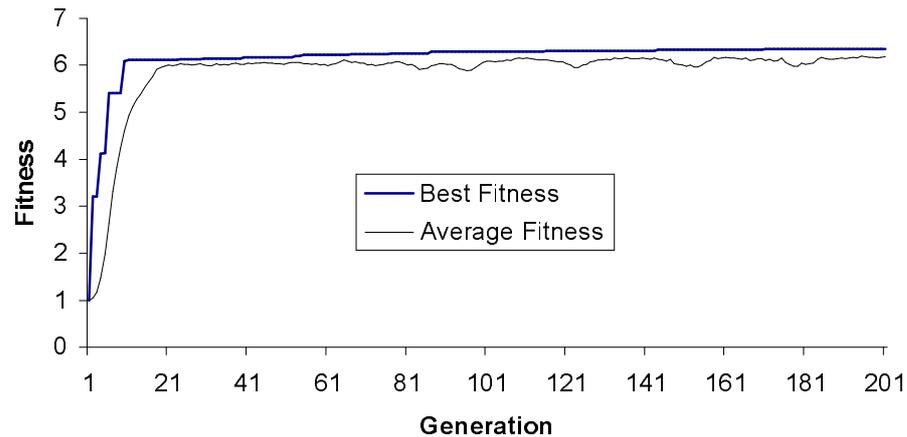


Figure 8.1 GA Performance for 8TAPFIR Non-Unfolded

For this example the design was produced within 180 generations; on average the 8TAPFIR required 420 generations to produce the best-found design. The unfolded designs are produced, on average, within 640 generations. The longer time is primarily due to the use of the unfolding principle, as described previously.

### 8.3.2 AVEN8DI

This filter has a power consumption of 20.058% of its original high-level design with a corresponding area of 304% of the original design. The power reduction was primarily obtained through a reduction in supply voltage to 1.995V. This was achieved by reducing the critical path from 9 to 3 elements, a 3-fold speed increase. As with the 8TAPFIR design, this was achieved by increasing the parallelism in the design through pipelining and retiming; the initial design required 2 adders and 2 multipliers, the optimised design requires 6 adders and 6 multipliers, hence the 3-fold increase in area.

The application of unfolding further reduces the power consumption of the AVEN8DI design, but it is unable to reduce the effective critical path below 3

elements. The reduction in power consumption is due to the parallel processing of samples; the unfolded design processes two samples in parallel while the area has increased less than two-fold. The unfolded design is approximately 1.7 times the size of the non-unfolded design, yet this large increase in area only yields a further 0.42% reduction in power.

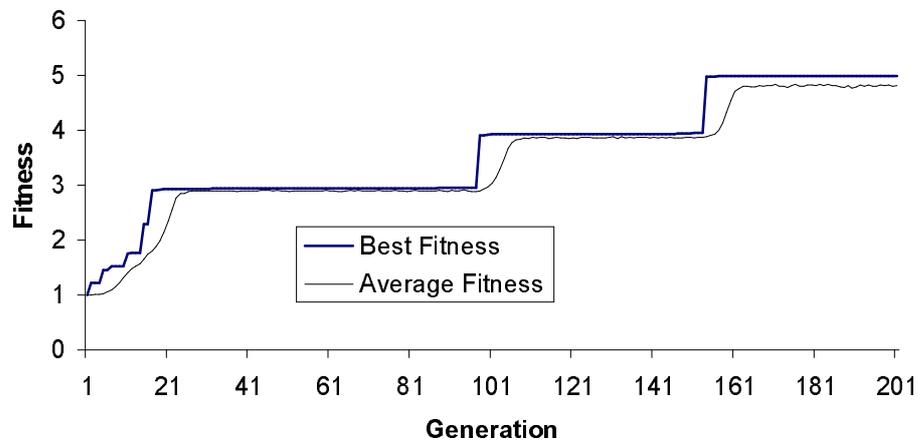


Figure 8.2 GA Performance for AVEN8DI Non-Unfolded

The GA performance is illustrated in Figure 8.2. The best-found design is found within an average of 210 generations (155 generations for this example). The unfolded version is found within an average of 862 generations.

### 8.3.3 AVEN8PA

Reducing the critical path from 8 to 3 elements has reduced the power to 23.816%. The area of the optimised design is 150% of the original design, a 50% increase in area with greater than 75% reduction in power. The application of unfolding did not improve the obtained power reduction as unfolding could not reduce the effective length of the critical path and could not reduce the effective

switched capacitance. The unfolded AVEN8PA designs were greater than N times the size of their non-unfolded originals, where N is the unfolding factor (and the number of samples the unfolded version processes in parallel).

The GA performance is illustrated in Figure 8.3. In this example the best-found design was found on generation 86, the average is 100 generations.

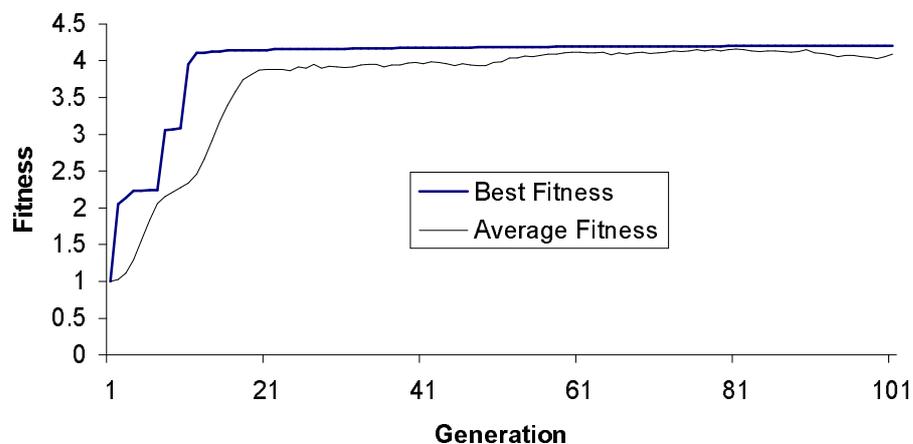


Figure 8.3 GA Performance for AVEN8PA

#### 8.3.4 DCST

This non-filter signal processing application has a reduced power consumption of 30% due to a reduction of its critical path from 6 to 3 elements. The large power reduction has been obtained with relatively minimal impact on area.

Unfolding produces a further decrease, again due to the processing multiple samples in parallel reducing the effective capacitance switched per sample. The GA performance is illustrated in Figure 8.4.

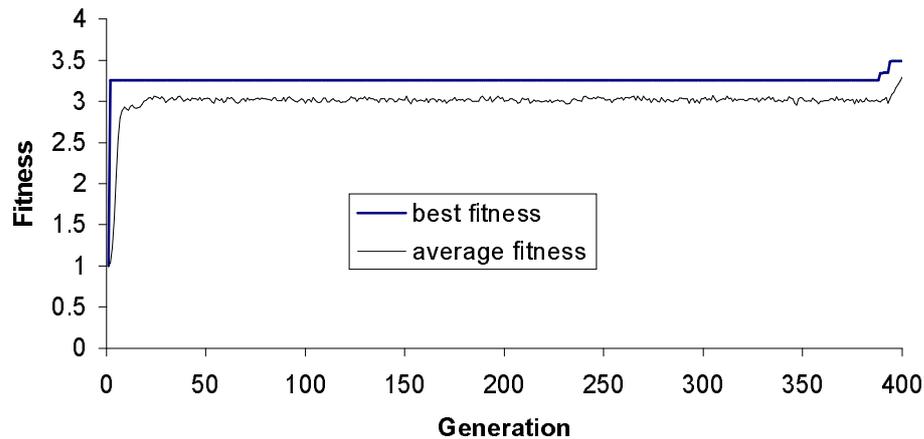


Figure 8.4 GA Performance for DCST Unfolded

This example illustrates the effect of unfolding on the GA performance. The non-unfolded best design is found within an average of 5 generations. The postponing principle reserves application of the unfolding transformation for a few hundred generations to ensure that the best non-unfolded design has been found. In this example the application of unfolding at generation 389 begins the search of the unfolded design solution space; the best-found unfolded design is found at generation 395 (on average it is found within 393 generations).

### 8.3.5 BIQUAD3

This design has a 2-fold speed increase due to the critical path being reduced from 6 to 3 elements. The transformations increased the available parallelism in the design enabling more operations to be performed in parallel. However, rather than requiring more resources the transformations enabled more efficient use of the initial resources (2 adders and 2 multipliers). The small area increase is due to the extra number of registers required to implement the faster design (achieved with a combination of pipelining and retiming). The initial design is estimated to require 5

registers while the optimised design is estimated to require 8 registers. Unfolding failed to improve the design.

The GA performance is illustrated in Figure 8.5. The optimum BIQUAD3 design is found, on average, within 8 generations. This relatively fast optimisation is due to the pipelining transformations specifically targeting the minimisation of the critical path. The inherent optimisation characteristics of the implemented pipeline transformation, to target critical path reduction, assist the GA search mechanism in quickly finding the best solution for this design.

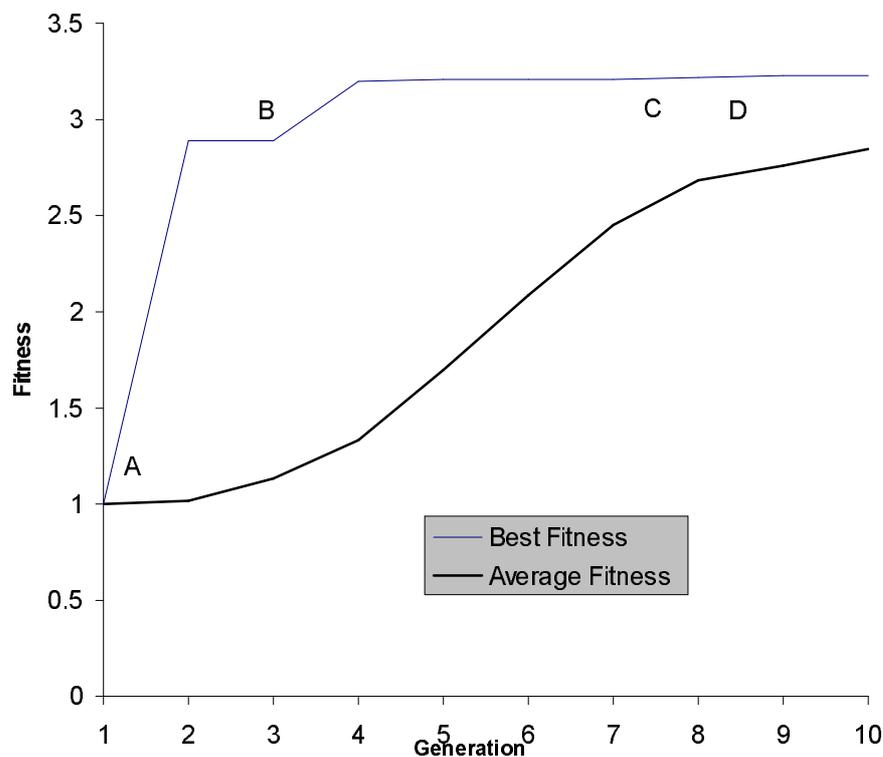


Figure 8.5 GA Performance for BIQUAD3

As this is a relatively simple optimisation process the effect of each optimisation step can be illustrated by points A, B, C and D. Points A and B

correspond to large increases in fitness where the critical path is reduced to its final length of 3 elements. Points C and D correspond to further optimisation steps, which reduce the area of this faster design. As voltage has the greatest impact on power it is the speed reducing transformations that account for the majority of the fitness increase.

This curve is typical of the synthesis process for all of the benchmark designs. Initial fitness improvements are primarily due to the reduction of critical path, resulting in a reduction in supply voltage. Once the shortest critical path has been determined the GA searches for the best (smallest) area implementation of that fastest design. These reductions in area correspond to smaller increases in fitness; in effect the fastest design is subsequently fine-tuned for small area implementation.

### 8.3.6 GMLAT4

This DFG has a lower supply voltage by reducing the critical path from 8 to 5 elements. The area has increased by 50% due to the requirements for an extra adder and an extra multiplier in the optimised version. The optimised version also requires 13 registers (compared to the original's 7) due to the use of pipelining increasing parallelism and hence reducing the critical path. Unfolding produces a design with an unfolding factor of 3 (processing 3 samples in parallel) with only a 0.316% further reduction in power compared to the non-unfolded version, despite requiring approximately 3 times the size of the non-unfolded version. This illustrates the importance of analysing both the unfolded and non-unfolded results as the very small reduction in power for the unfolded version comes at a large expense in area compared to the non-unfolded version.

The GA performance is illustrated in Figure 8.6. The optimum design is found on generation 12 (found on average within 12 generations). The unfolded design is found, on average, within 390 generations.

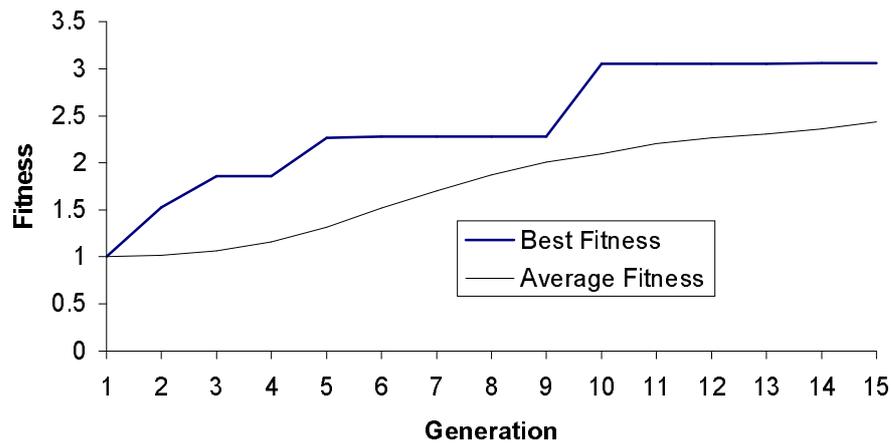


Figure 8.6 GA Performance for GMLAT4 Non-Unfolded

### 8.3.7 ELLIP5

This large filter has an initial critical path length of 14 elements, the GALOPS tool has reduced this to 12 elements enabling the voltage to be reduced to 4.328V. In addition, the non-optimised version is estimated to require 3 adders, 2 multipliers and 10 registers. The optimised version is estimated to require 3 adders, 1 multiplier and 10 registers, a reduction of 1 multiplier. As the multiplier is the most complex and largest component in the datapath, this reduction produces a significant saving in area of approximately 40% [Ma90]. This example illustrates that the GALOPS tool optimises for supply voltage and area. The reduction of supply voltage produces the largest power savings; the impact of area on switched capacitance (and hence power) results in the tool indirectly optimising area while reducing power consumption.

The GA performance is illustrated in Figure 8.7. The best design is typically found within 19 generations.

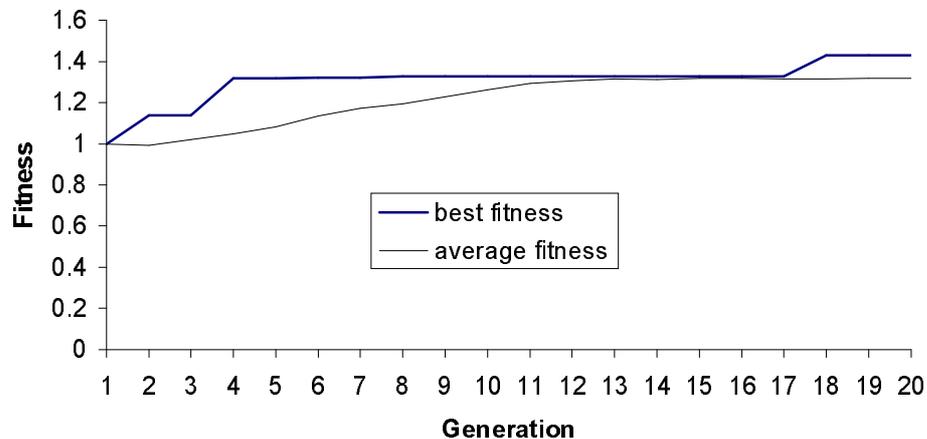


Figure 8.7 GA Performance for ELLIP5

### 8.3.8 LMS5

This DFG has a reduced power consumption of approximately 90% without any reduction in supply voltage. The GALOPS tool was not able to reduce the critical path length but was able to reduce resource utilisation from 4 adders and 5 multipliers to 3 adders and 3 multipliers, a reduction in area of approximately 40%. Unfolding further reduced power by reducing the effective switched capacitance but with an associated 17% increase in area compared to the initial design (88% larger than the optimised non-unfolded design).

The GA performance is illustrated in Figure 8.8. The initial few generations show a drop in the average fitness. The transformations are exploring the solution space, producing inferior designs with longer critical paths and larger areas than the initial design. Eventually the search moves towards a better region of the solution space that contains designs with smaller areas, hence the average fitness of the

entire population begins to increase. The best solution is found, on average, within 32 generations and 430 generations for the non-unfolded and unfolded cases respectively.

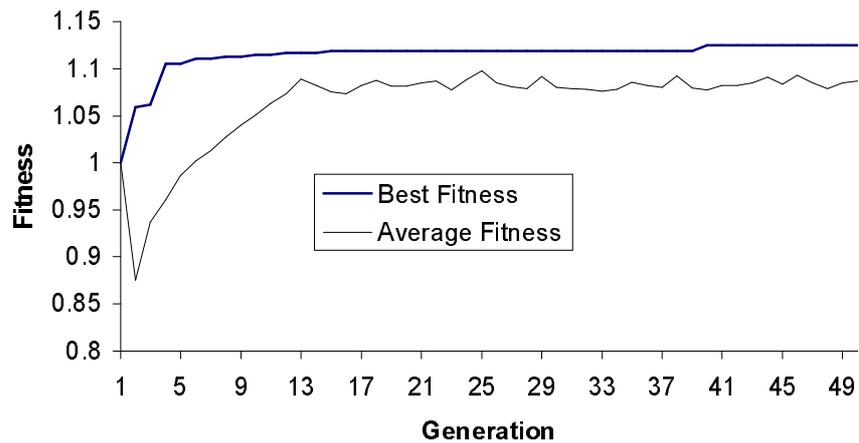


Figure 8.8 GA Performance for LMS5 Non-Unfolded

### 8.3.9 VOLTERRA

As with the LMS5 design, the GALOPS tool was unable to reduce the critical path length of the VOLTERRA design. The critical path of the VOLTERRA design is a large feedback loop that limits the application of pipelining transformations. The power reduction of 5% is achieved by optimising resource utilisation; the optimised VOLTERRA is estimated to require 1 less multiplier than the initial design.

The GA performance is illustrated in Figure 8.9. The fittest solution is found on the first generation, the critical path is not reduced but the area is. As with the LMS5 design the average fitness initially drops before increasing.

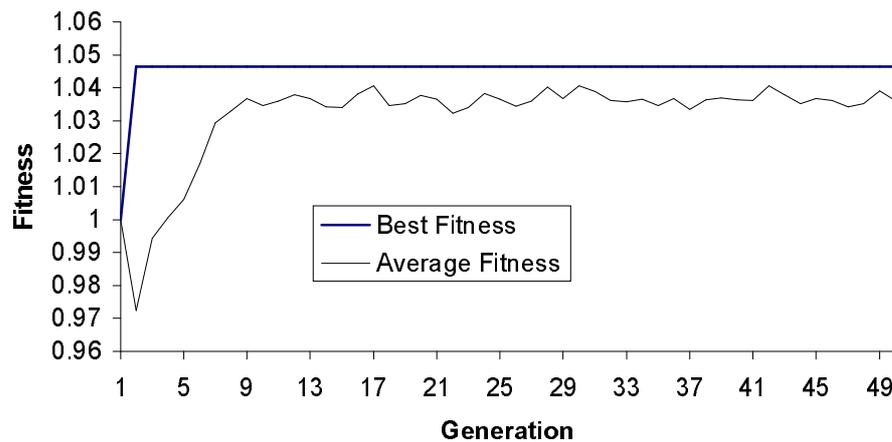


Figure 8.9 GA Performance for VOLTERRA

### 8.3.10 ORTH2LAT

The results illustrate that the GALOPS tool failed to reduce either the voltage or area of this design, the power consumption was not reduced. As with the LMS5 and VOLTERRA the critical path is a recursive loop which limits the application of the transformations for path reduction. In addition, the transformations were unable to reduce the resource utilisation.

The unfolding transformation produced a design with a power consumption of 3% less than the original design with an associated 62% increase in area. The effective capacitance of the 2-unfolded design is less than that of the original design, thus the 3% reduction in power.

The GA performance is illustrated in Figure 8.10.

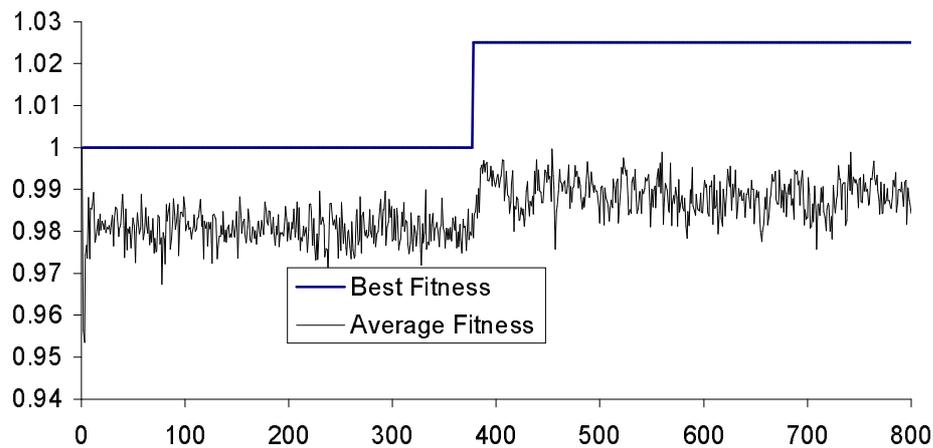


Figure 8.10 GA Performance for ORTH2LAT Unfolded Design

In this example the application of unfolding at generation 376 produces the best-found design. The average fitness varies throughout evolution as the transformations are producing designs which are sub-optimal compared to the current fittest. This decreases the average fitness of the population. The process of natural selection then favours the selection of the fitter members of the population, increasing the average fitness. This process is continually repeated throughout the evolution hence the variation in the average fitness.

#### 8.4 Discussion

Figure 8.11 illustrates the power reduction and area increase obtained with all of the benchmark designs, the 'X' denotes a design produced with the unfolding transformation.

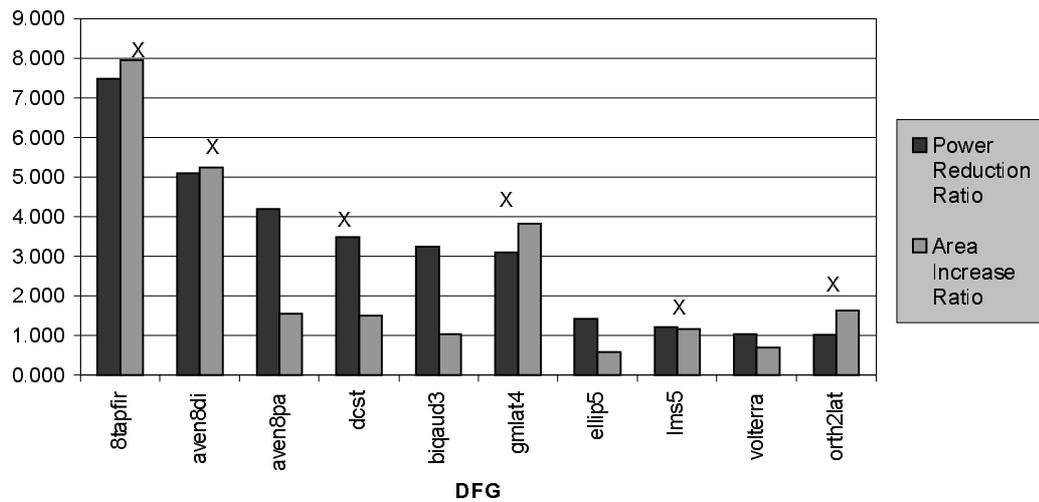


Figure 8.11 Power Reduction and Area Increase for Ten Benchmark Designs

The 8TAPFIR design has the largest power reduction and also the largest area increase. This is due to the large initial critical path in this design that is reduced to its minimum possible length of 1. Therefore the speedup ratio is large, allowing for a large decrease in voltage and hence a large reduction in power. The non-recursive nature of the 8TAPFIR design is also beneficial for power reduction; as it does not contain any feedback loops the application of the transformations is not limited and unfolding can produce designs with considerable power savings by reducing the effective critical path length (hence the large area increase).

The AVEN8DI to ELLIP5 designs display decreasing amounts of power reduction. This is partly due to the increased amount of recursion in these designs limiting the application of the transformations; in particular, when the critical path is within a recursive loop the transformations are unable to significantly reduce its length. For these designs, excepting the AVEN8PA design, unfolding does not produce a reduction in the length of the critical path but it does reduce the effective switched capacitance by processing more samples in parallel.

The last three designs all contain significant amounts of recursion; of particular importance is the fact that the critical path is contained within a recursive loop. Therefore the transformations are unable to speed-up the design so the supply voltage cannot be reduced. In these designs the power reduction is obtained through reducing the effective capacitance.

The results for each design are presented in comparison with their original design instead of absolute values. This accounts for some of the difference in power consumption between the benchmark designs. Where a design has a relatively large critical path, which can be significantly reduced, it can operate from a significantly lower supply voltage and hence report a large percentage decrease in power. If a design has a small initial critical path then reducing it may not produce the same speed-up, and hence the same voltage reduction, as the design with the larger critical path. For example, consider the AVEN8DI and DCST designs. The AVEN8DI has an initial path of 9 elements, the DCST has an initial path of 6 elements. Both are reduced to 3 elements, a 3-fold and 2-fold increase in speed respectively. Therefore, the AVEN8DI has a lower supply voltage (due to its larger speed increase) and hence a larger percentage power reduction is reported. The example illustrates that the reported power reduction is related to the complexity of the initial design; hence the larger design in this case reports a larger power reduction.

The ELLIP5 and VOLTERRA designs are optimised for power while reducing the estimated area of the design, illustrating the dual consideration of speed and area during the low-power design process. However, speed will always be the main optimisation factor due to the quadratic influence of supply voltage on power. The results clearly illustrate that large power savings are possible through

reduction of supply voltage, operating the device at a supply voltage that produces the required throughput.

## 8.5 Conclusions

This thesis has presented a low-power synthesis tool that integrates various GA search and optimisation techniques within a power-exploration framework. The power exploration includes a high-level power estimation tool that considers the effect of the high-level synthesis process on the candidate designs, estimating the power consumption due to low-level parameters such as capacitance and area.

Results were presented for ten benchmark designs that cover a range of applications of varying complexity and implementation requirements. The results illustrate that the tool is capable of processing and analysing a range of designs, producing significant power reductions in most cases.

The suggested supply voltages for the low-voltage designs are significantly different to the industry standards of 5V and 3.3V. Different supply voltages for ICs, where the whole core of the chip operates at a non-standard supply voltage, are possible through the implementation of small-area, efficient, voltage-level converters on the IC [Strat94, Strat94a]. The IC is fabricated to require the specified external supply voltage (such as 3.3V), but the on-chip voltage converter supplies the required low-power voltage to the device core. Such systems have been used to implement practical, fabricated signal-processing devices, for example in a low-power portable multimedia terminal [Chan94, Sheng92] and low-voltage DSP devices [Lee98].

The latest generation of mobile Pentium-processor chips is an example of the application of low-voltage to produce low-power devices. They have been

specifically designed to meet the challenge of providing full functionality and longer battery life for modern portable computing systems. The devices operate from a 3.3V power supply but they have a core voltage of 2.45V to reduce power consumption [Intel98b]. The required low-voltage for the core is provided with the addition of an extra supply voltage input, an alternative solution to using an on-chip voltage converter. The description of the processor in [Intel98b] also notes that future developments of the processor may utilise an even lower core supply voltage in order to reduce power further.

Recent techniques for power reduction have seen the use of dynamically varying voltages, where the voltage of the processor can be determined at run-time. Certain tasks can be processed over long run-times where the result is not required immediately. Rather than process the task as fast as possible the supply voltage of the processor is selected to reduce system speed so that the task is processed within the required time [Hong98]. In addition separate systems can be run at different voltage levels, to maximise the potential voltage reduction for the overall system [Shiue98].

These practical examples illustrate the feasibility of operating ICs at different core voltage levels. While non-standard supply voltages have an impact on cost and fabrication complexity, they are essential for producing ICs where low power consumption is a critical requirement, as illustrated by the results produced with the GALOPS tool.

The area increases for those designs with large power reductions follow the traditional trend of trading off area for speed (and hence lower supply voltage). The closer the design is expected to operate at its minimum possible critical path length, the more parallel processing will be required and hence a larger number of

resources is required. The application of unfolding, to utilise parallel processing of samples for higher speed, increases the area-speed trade-off. Traditionally, increases in VLSI integration, allowing for devices with more transistors, were traded off for speed increases. The presented results illustrate that in future, increases in device integration and device size may be traded off for power reduction where the desired speed is already obtained.

## Chapter 9 - Multi-Objective Search Space Exploration

The optimisation process presented in GALOPS attempts to optimise a single parameter, the estimated power consumption of a signal-processing algorithm, for VLSI implementation. However, practical VLSI design, as with many engineering design problems, involves the simultaneous optimisation of a number of objective parameters such as power, area, speed, cost, etc.

Within the context of a GA, the multiple parameters need to be integrated into the optimisation process, to ensure that all parameters are simultaneously optimised i.e. no single parameter is concentrated on at the expense of others. The simultaneous optimisation of all parameters will produce a single solution, a global optimum solution for all objectives. One technique for implementing multiple-parameter optimisation is the use of a weighted fitness function [Arslan96a, Arslan96b] as illustrated in (9.1).

$$Total\_Fitness = \alpha.Parameter_1 + \beta.Parameter_2 + \dots + \zeta.Parameter_N \quad (9.1)$$

The use of a weighted fitness function combines the discrete fitness values for each parameter into a scalar fitness value. The scalar fitness value is used to assess the overall quality of the solution. Equation (9.1) demonstrates that a typical implementation of a weighted fitness function is a sum of all the individual fitness scores for each of the parameters, from 1 to N different parameters. The variables  $\alpha, \beta$ , etc. are used to weight the individual contribution of each parameter to the overall fitness.

One of the main problems with such a technique is the definition of the weighting variables, as these will significantly affect the performance of the system in determining the globally optimal result [Gwee96]. The weights effectively place a priority on the optimisation of a particular parameter in relation to the optimisation of other parameters. Therefore, assigning the correct priority to the multiple parameters is required to enable the optimisation process to produce the global optimum solution.

In a variety of practical engineering problems, prioritisation of the multiple objectives is a complex and difficult process. Such problems are often characterised by a number of competing objectives, where improvements in one parameter come at the cost of degrading the quality of other parameters. The trade-offs between such competing parameters are often non-linear. For example, consider the dual-objective problem of synthesising a minimum area and minimum power VLSI device. The two parameters are in competition with each other as many low-power design techniques use extra area to decrease power. In a practical design process the VLSI device may be targeted for a certain die size. An increase in die size past this upper limit may result in a large increase in cost (and decrease in fitness), whereas increases below this limit will have a small effect on cost and overall fitness. Thus, the effect of area on the overall fitness is dependent upon the actual fitness value. A single weight to assign a specific priority to the area parameter would not accurately reflect the total fitness for all solutions.

If the non-linear nature of the objectives can be overcome, there is still the problem of assigning a relative priority to each parameter for all cases e.g. is area more important than power, and if so, how much more important?

The example illustrates the difficulty of combining distinct parameters into a single fitness evaluation to generate a single globally optimal solution. An alternative to a weighted fitness function is to use the GA to explore the solution space and present a range of alternative non-dominated solutions (NDS) that are each optimal for a single parameter [Esben96, Gwee96]. This removes the need to prioritise parameters during the optimisation process. The alternative solutions can then be analysed by the expert designer to select the solution that best satisfies the specified requirements.

During the optimisation process the GA already evaluates many alternative solutions. Rather than discard this useful information when presenting a single solution, the information can be used to illustrate the trade-offs between different parameters in the optimisation problem. The trade-off between competing parameters is typically presented as a Pareto-surface [Goldberg89].

The use of Pareto-surface information, collected throughout the GA optimisation process, has been previously illustrated to be beneficial to the low-power synthesis process in the Power-Profiler tool [Martin95, Martin96]. Power-Profiler used the points explored during the GA search to present architectural trade-offs as pareto-surfaces, illustrating the trade-off between number of functional units and power for example.

### 9.1 Pareto-Surface

A Pareto-surface is made up from a set of non-dominated solutions (NDS); that is, for the parameters under analysis, each point on the pareto-surface has no better values for those parameters. Figure 9.1 shows an example of a pareto-surface for a 2-dimensional optimisation problem.

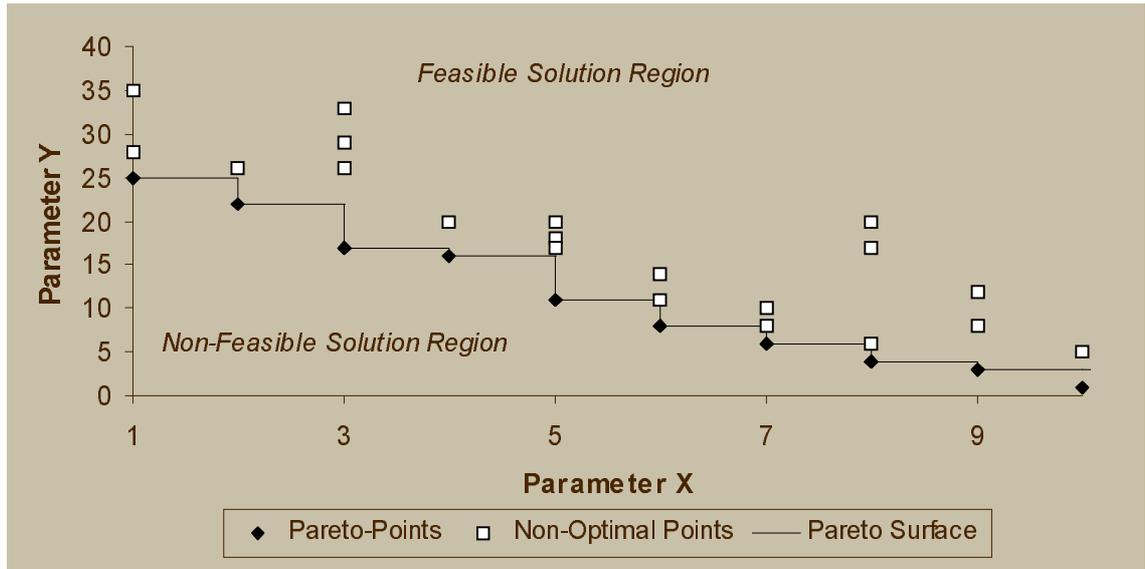


Figure 9.1 Example Pareto-Surface for 2-Dimensional Optimisation

The chart illustrates a typical engineering design example where 2 competing parameters (X and Y) are required to be minimised. Each solution is illustrated on the chart as either a diamond or a square. The diamonds denote the set of NDS i.e. those solutions for which no point has a lower value for both the X and Y parameter. These points are known as Pareto-optimal or Pareto-points. The Pareto-surface, a curve joining the set of Pareto-points, marks the boundary between the range of feasible and non-feasible solutions, illustrating the trade-off between each parameter.

The set of NDS does not present an obvious single optimum solution. The task of selecting the best solution is left to the design engineer, dependent upon the priority placed on each parameter. One of the main advantages of the Pareto-chart is that it shows the effect of varying a parameter, as opposed to presenting the designer with a single point solution. For example, the chart may show that a small variation in X, previously set as a constraint, may allow such a large reduction in Y

as to make the overall design feasible. The Pareto-chart allows the designer to use their expert knowledge of the problem to select an optimum solution.

The range of alternative solutions may be more useful than a single point solution for the next stage of the design and implementation process. A single solution may be selected from the NDS based on some additional criteria not present in the optimisation strategy, such as ease of understanding or preferred implementation styles. Such criteria are difficult to define in qualitative terms and hence very difficult to incorporate into a design objective function.

Within the context of a CAD synthesis tool, the presentation of design alternatives is regarded as essential by most design engineers; CAD users do not like point solutions [Gajski94]. The presentation of a set of optimal solutions enables the design engineer to gain a greater understanding of the low power solution space and the power characteristics of the problem to be optimised.

The results presented for the GALOPS tool in chapter 8 are the lowest power solution of each design. The results illustrate that the specified power reduction is typically achieved through an increase in area. While the identification of the lowest power solution is important, it is also useful to the CAD designer to present solutions that span a range of areas, enabling the designer to select the lowest power solution for a range of area constraints.

The pareto-surface is used to illustrate the trade-off between area and power of the designs. For each point on the area axis the lowest power solution determined by GALOPS is identified. These points make up the set of NDS. The generation of pareto-surface information is used to examine the effect of the power optimisation process on the area of the design. Rather than generating a single low-power solution, the pareto-surface information can be used by the design engineer to select

the best low-power solution for a specified area constraint. Therefore, the pareto-surface enables the smallest implementation of a particular low-power design or, it enables the identification of the lowest-power design that can be synthesised for a specific area. Rather than trying to optimise a single solution for low-power and small-area, GALOPS concentrates on optimising for low-power, consequently presenting a range of low-power solutions across the area axis.

### 9.2 Pareto Surface Generation in GALOPS

A pareto-point is defined as that which has no lower value in both the X and Y axes, the area and power axis in this case. Identification of Pareto-points can be split into a two-stage process, where all designs with the lowest Y value for every generated X value are first identified. This set of designs is sorted into ascending X-values. The next stage steps through the set of sorted designs, removing any with a larger Y value than the previous lowest found. This process is illustrated in figure X.

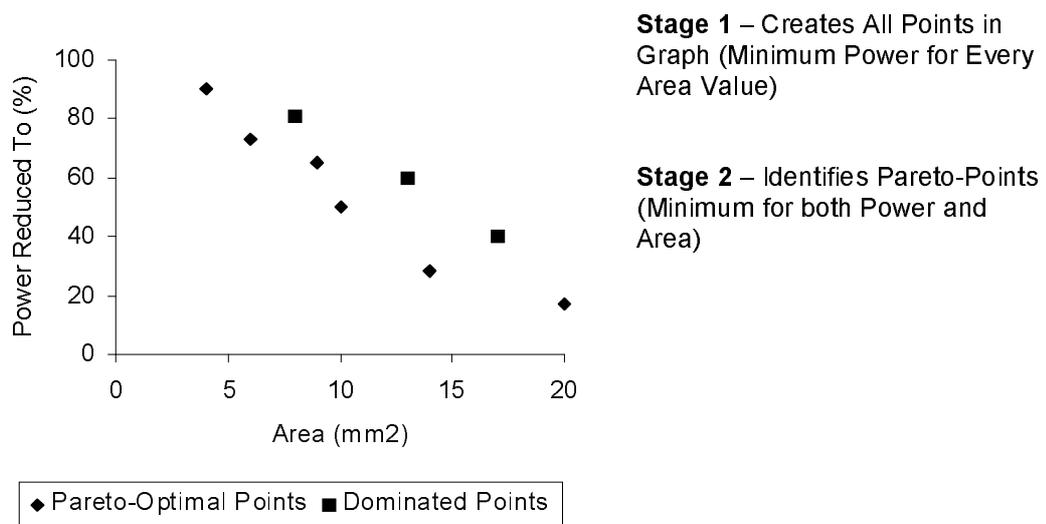


Figure 9.2 Identification of Pareto-Points in GALOPS

Executing both stages produces a set of NDS, the area-power pareto-optimal points. After each generation is created, the power and area of each design is analysed to create a list of points following stage 1 i.e. the minimum power for each area explored during the synthesis process. After the GA has determined the lowest power solution, stage 2 is executed to determine which of these points are pareto-optimal.

### **9.3 Area and Power Trade-off for Benchmark Designs**

This section illustrates the area and power trade-offs, through the use of pareto-surface analysis, for the benchmark designs presented in section 8.1. The pareto-charts were generated using the techniques described in section 9.2. The GA parameters were the same as those used to generate the results presented in section 8.3. Each pareto-chart is presented in a separate section for that design. In each chart a solid line denotes the pareto-surface. The pareto-surface is presented as a straight line rather than a curve joining the pareto-points. This is due to the nature of the VLSI synthesis problem. Solutions exist at discrete points in the solution space; a curve may imply that there is a range of solutions between two points when no feasible solutions actually exist between those points.

Each chart also shows the unique points in the solution space examined while searching for an optimum solution for that design. Each 'x' denotes a unique power-area point for a design. It should be noted that each area-power point does not necessarily correspond to a single design. A number of designs can have the same area-power characteristics, as illustrated in Figure 9.2. Therefore, the number of 'x' points in the graph may not be representative of the total number of designs analysed throughout the search process

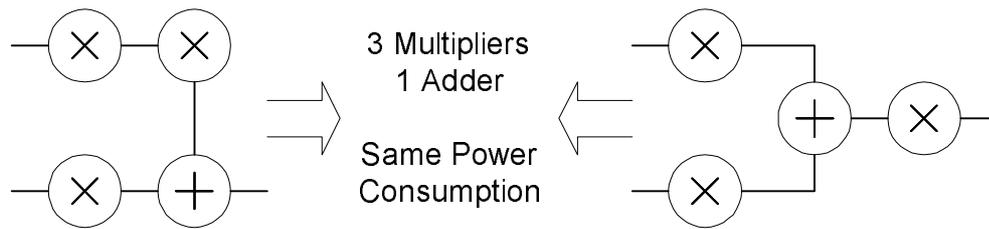


Figure 9.3 Example of different DFGs with the Same Power and Area Characteristics

### 9.3.1 8TAPFIR

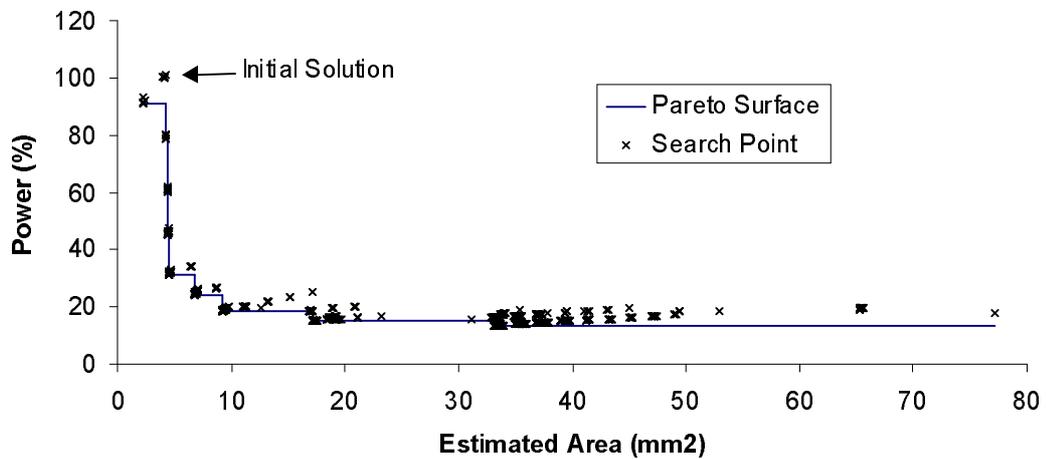


Figure 9.4 Pareto-Surface for 8TAPFIR

The pareto-chart in Figure 9.4 illustrates the initial solution, with a power consumption of 100% at 5V. The chart clearly illustrates the trade-off between area and power; the greater the area increase, the larger the power reduction which can be obtained. However, once minimum power consumption has been obtained the graph also illustrates that designs larger than approx.  $33\text{mm}^2$  offer no further reduction in power.

The advantage of the graph is illustrated by comparing it with the point-solution data presented in section 8.3. The point-solution specified a design with a power consumption of 13.381% with an area of 33mm<sup>2</sup> (an increase of approx. 600% compared to the initial solution). The graph illustrates that a power consumption of 14.9% is achievable with an area of 18mm<sup>2</sup> (an area increase of approx. 350%). Using the pareto-chart the VLSI designer can decide whether the further increase in area is worth the extra, small reduction in power. A similar analysis on trade-offs, especially when compared to the single-point solutions, can be performed for all of the Pareto-surfaces in this section.

The graph also illustrates that the power can be reduced to 92% while the area is also reduced. Viewed as a whole, the graph presents a family of solutions to the VLSI designer, the required solution can be selected based upon the particular implementation criteria.

### 9.3.2 AVEN8DI

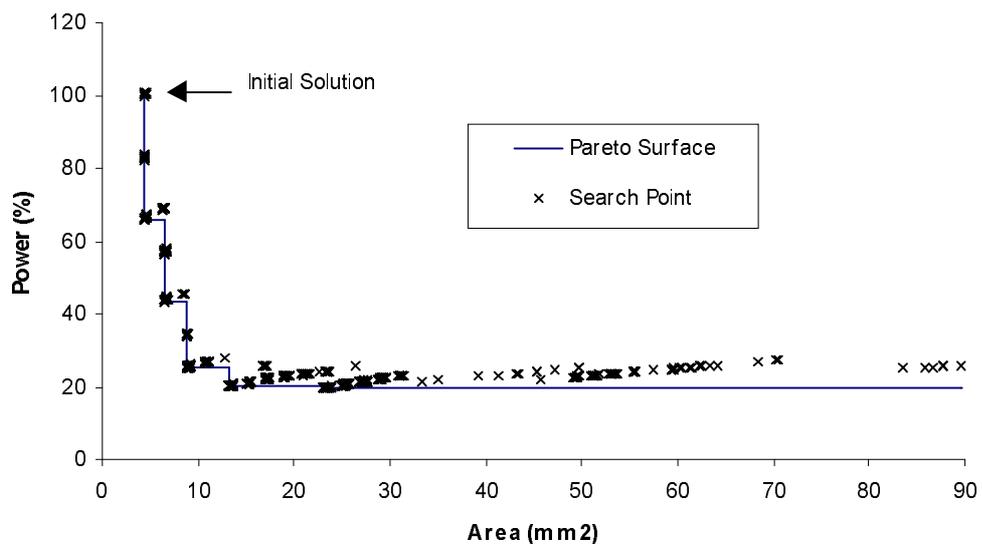


Figure 9.5 Pareto-Surface for AVEN8DI

This pareto-surface also illustrates the small decreases in power consumption that are obtained with successively larger designs. The large amount of search points for larger areas illustrates the search of the unfolded-design search space, producing larger designs but with larger power consumption than the 23mm<sup>2</sup> design.

### 9.3.3 AVEN8PA

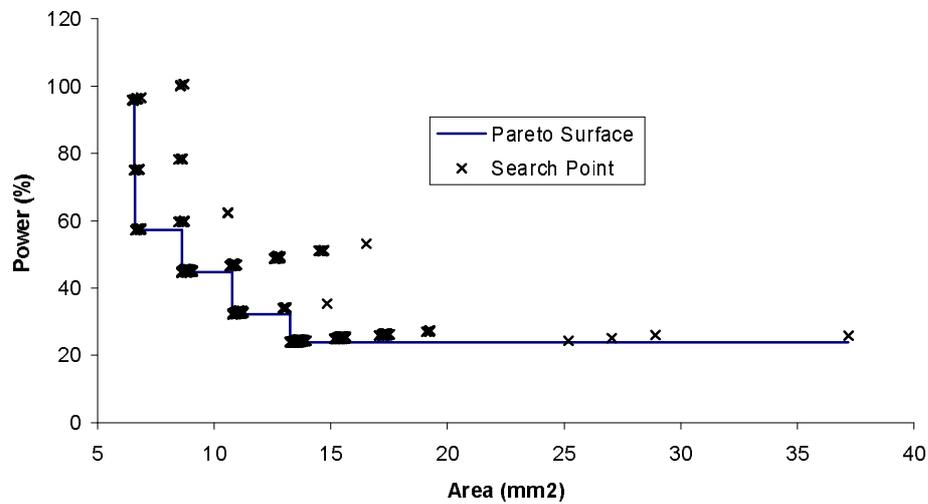


Figure 9.6 Pareto-Surface for AVEN8PA

The chart illustrates that power can be reduced to less than 60% while also reducing area from approx. 8.5mm<sup>2</sup> to 6.5mm<sup>2</sup>, which cannot be deduced from a single solution showing the lowest power obtained. The AVEN8PA design did not benefit from the unfolding transformation, the solutions larger than 15mm<sup>2</sup> represent the unfolded designs with larger power consumption than the non-unfolded design.

9.3.4 DCST

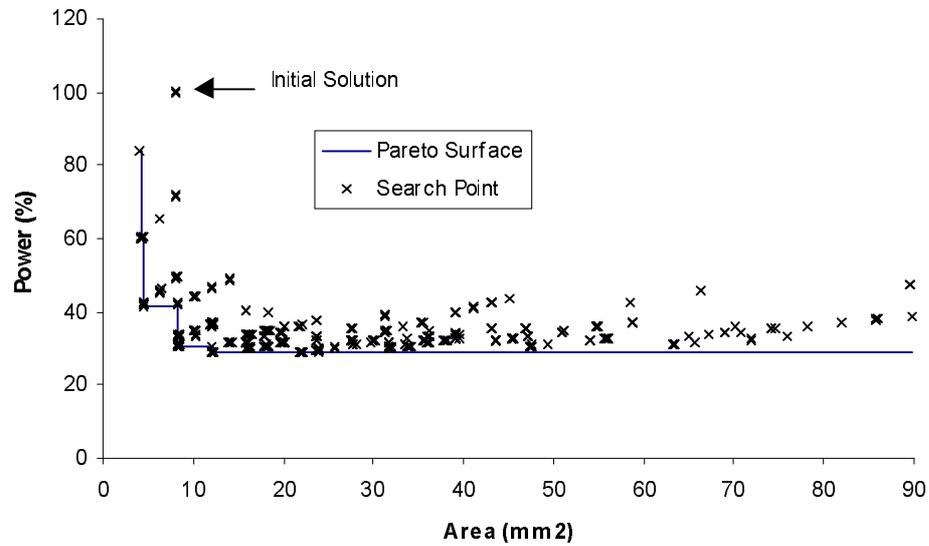


Figure 9.7 Pareto-Surface for DCST

This chart illustrates that power was reduced to approx. 40% with a decrease in area; the lowest-power design has a power consumption of 30% with a 2% increase in area (an area of approx. 12mm<sup>2</sup>). The chart shows a large number of search points explored above this area, illustrating the exploration of the pipelined and unfolded solution space.

9.3.5 BIQUAD3

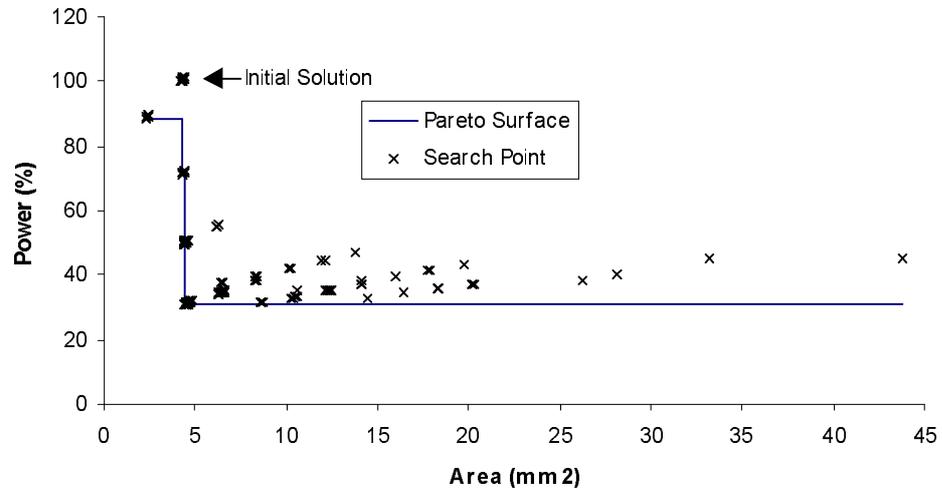


Figure 9.8 Pareto-Surface for BIQUAD3

This chart also illustrates that power reduction is possible while reducing area.

9.3.6 GMLAT4

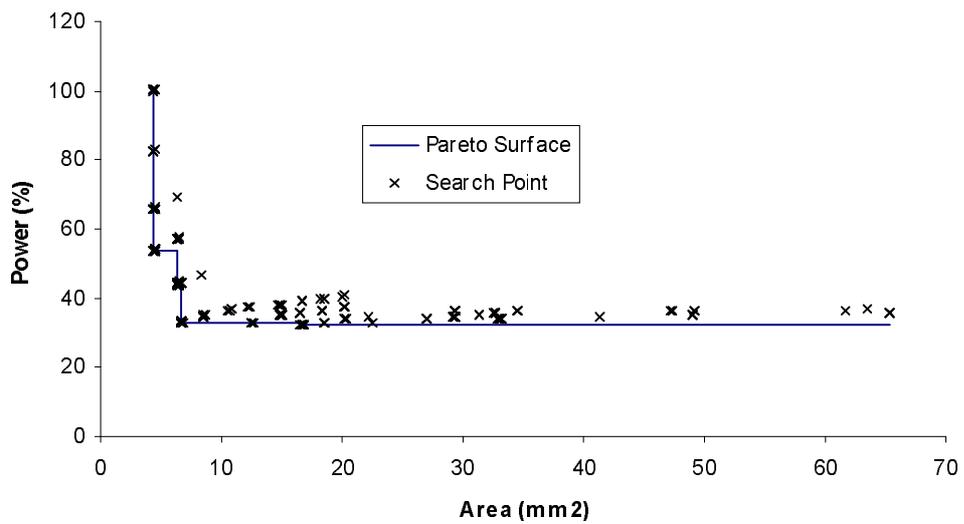


Figure 9.9 Pareto-Surface for GMLAT4

The power consumption of this design can be reduced to approx. 50% with only a 0.1mm<sup>2</sup> increase in area. Reduction to the lowest-found power of approx. 30% requires an area increase of approx. 12mm<sup>2</sup>.

### 9.3.7 ELLIP5

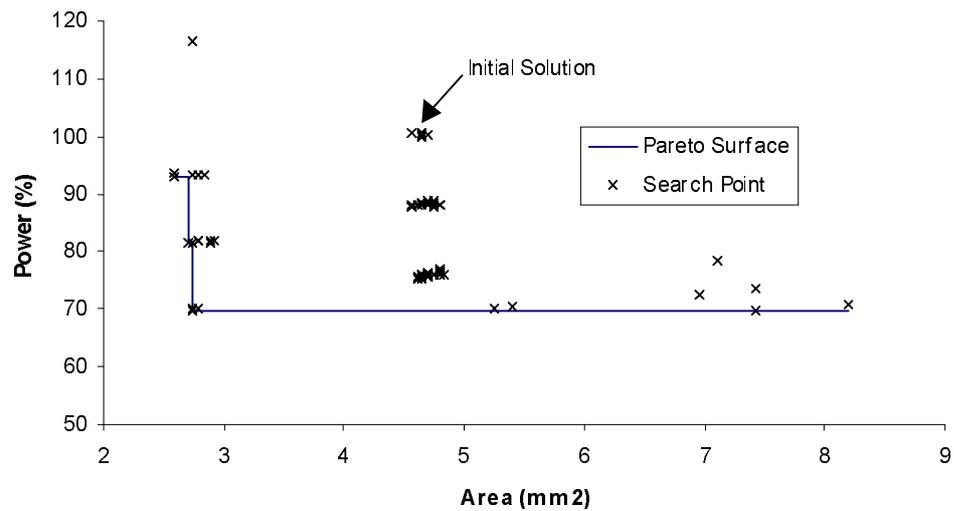


Figure 9.10 Pareto-Surface for ELLIP5

This design clearly illustrates that a large reduction in area (40%) is possible while reducing the power consumption to 70%. The chart illustrates the inefficiency of directly implementing the initial design without implementing power- or area-reducing transformations to produce a more power- and area-efficient design.

9.3.8 LMS5

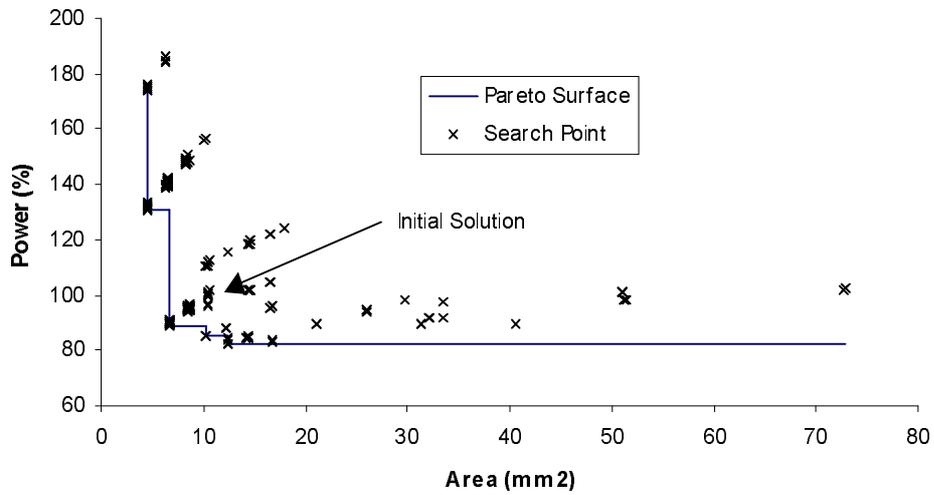


Figure 9.11 Pareto-Surface for LMS5

This chart illustrates that application of the transformations produced a lot of designs that were smaller than the initial design but with considerably higher power consumption. However, reduction of the power to 90% is possible with a 4mm<sup>2</sup> decrease in area. The lowest-power design (83%) requires an area increase of 2mm<sup>2</sup>.

9.3.9 VOLTERRA

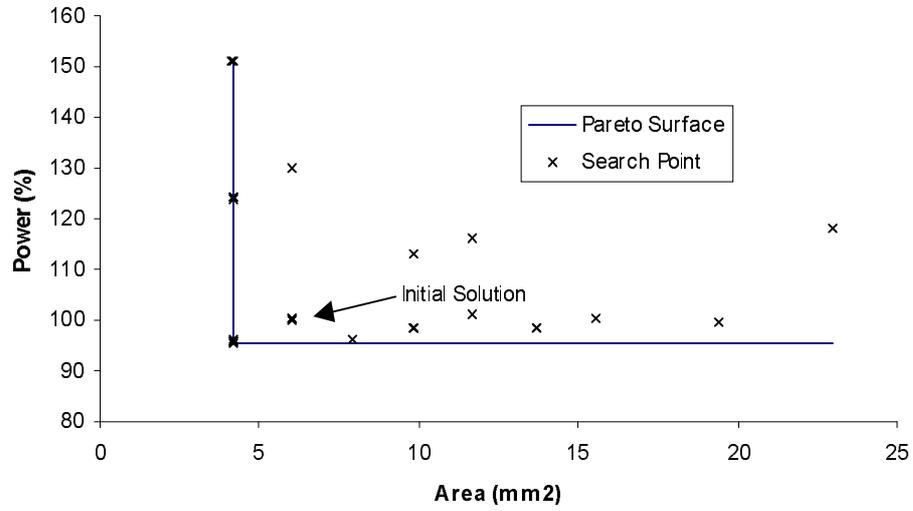


Figure 9.12 Pareto-Surface for VOLTERRA

This chart illustrates the difficulty that GALOPS had reducing the power consumption of the VOLTERRA design. The recursive nature of the design restricted the application of the transformations hence the small number of search points (26).

9.3.10 ORTH2LAT

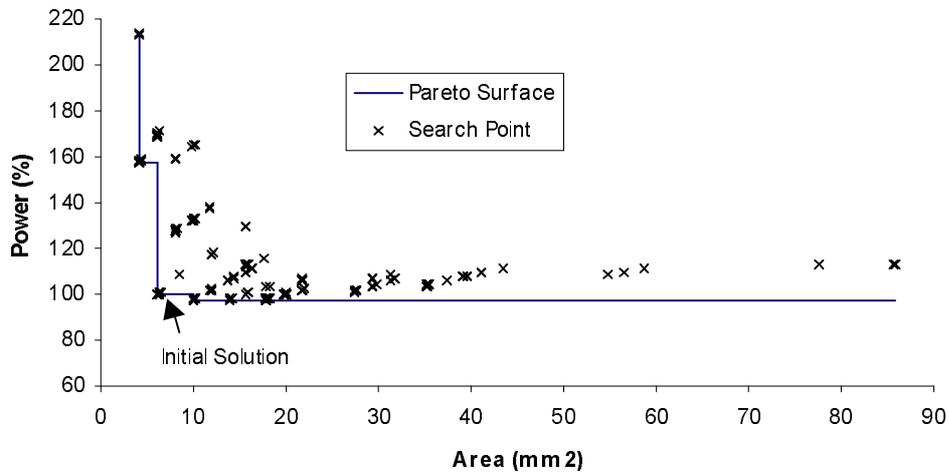


Figure 9.13 Pareto-Surface for ORTH2LAT

This pareto-surface illustrates that unfolding is required to reduce the power consumption by 3%, with a 60% increase in area. Even though designs are explored up to approx. 90mm<sup>2</sup> (the maximum in GALOPS) no further power reduction is obtained.

9.4 Discussion

All of the pareto-surface charts follow the same trend, illustrating a relationship between power and area. Decreasing power consumption is typically achieved through greater area. Where designs were optimised with a decrease in area this was due to inefficiencies in the initial DFG specification. In these cases, once the area had been reduced to achieve a power reduction, further power reduction typically required more area. This is primarily due to the impact of speed on power as described before. Operating at a lower speed (and hence a lower

voltage) requires more parallel processing to retain throughput; therefore, more resources are needed in the datapath.

Figure 9.14 illustrates the typical features of the pareto-surface charts.

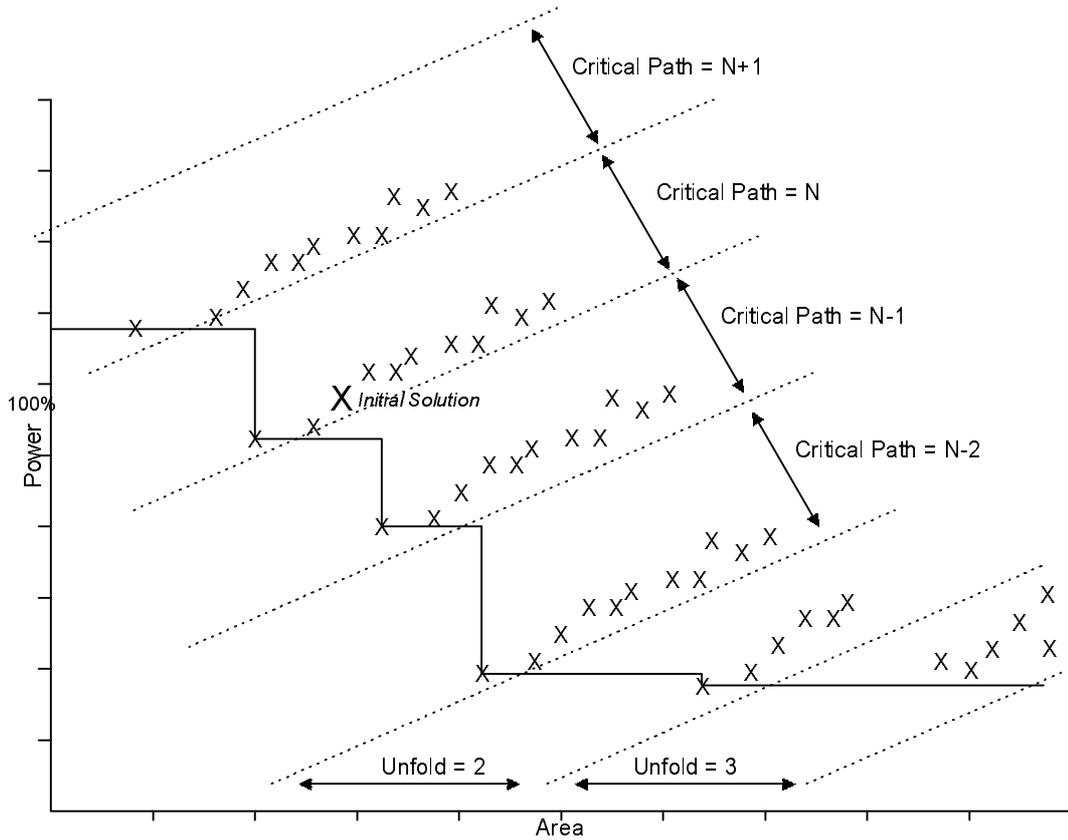


Figure 9.14 Typical Pareto-Surface Chart

In this figure, the pareto-surface is denoted by a solid-line. The pareto-surface follows the typical trend of increased area allowing reduced power. The search-points, denoted by X's, can be grouped by the length of their critical path. The initial critical path length (N) contains the initial solution. For each critical path a range of solutions exist with different area requirements. However, the power consumption of these designs is relatively similar. This is because the length of the

critical path specifies the design's supply voltage, which has the greatest impact on power. The area has a lesser impact on power.

The graph also shows grouping of solutions by unfolding factor; as unfolding is always performed at integer intervals the increase in area approximates to the product of the unfolding factor and the area of the non-unfolded design.

### **9.5 Conclusion**

This chapter has presented the use of pareto-surface charts for the determination of the best design to meet the required parameters. These parameters may be VLSI implementation details, such as power and area, or higher-level design decisions such as the cost of a particular device size or the preferred supply voltage (related to the critical path length).

The multi-solution nature of the GA, which inherently performs an exploration of the solution space, is exploited to produce the pareto-surface charts. Rather than discard the information built up during the search process the pareto-surface enables use of this information for design selection and analysis of the search procedure. They provide information about the nature of the solution space to provide an intuitive view of low-power design trade-offs, as discussed in section 9.4.

For a number of solutions the pareto-surface charts highlight low-power solutions that require smaller areas with only minimal reduction in the overall power reduction, compared to the single 'best-power' solutions presented in chapter 8. This information is invaluable to the design engineer as he can select the design that best suits his overall requirements rather than the single criteria of reducing power.

## **Chapter 10 – Development of Alternative Search Techniques**

As described in Chapter 4, the GA has many advantageous properties that prompted its selection as the core of a high-level power-conscious design tool. Its parallel nature, sampling multiple points of the solution space, provides a robust and efficient search mechanism, capable of determining ‘good’ solutions in complex, discontinuous search spaces. In addition, GAs have been successfully used in many VLSI design and analysis applications, outperforming more traditional algorithms. Therefore, the GALOPS system uses a GA to search the complex solution space inherent in the low-power synthesis problem. However, while the GA has been demonstrated as being able to produce solutions to a range of benchmark designs, it is not the only method of searching the solution space to determine a ‘good’ solution that meets the objective requirements.

This chapter presents the development of alternative search algorithms to explore the high-level power design space. The selected algorithms represent a small subset of search techniques, incorporating simple heuristics (Gradient search) and naturally inspired optimisation processes (Simulated Annealing). These techniques are traditionally used to solve a range of problems, including VLSI design problems, and as such the choice of a GA for the GALOPS system needs to be analysed in terms of its ability to outperform other techniques.

The chapter outlines the development of the algorithms that require the core transformation and exploration routines that were developed for the implementation of the GA-based search tool. The performance of the developed algorithms is

compared with the GA using the results obtained for the set of benchmark designs listed in Appendix A.

### **10.1 Gradient Search**

One of the simplest search algorithms is the gradient or hill-climbing algorithm. The algorithm attempts to find the optimum solution through successive modifications of the initial solution. If the modification produces an improvement, then the modified solution is chosen as the starting point for the next modification. The algorithm is known as a hill-climbing algorithm because it only accepts improvements to the solution i.e. only ‘uphill’ movements in the solution space are accepted.

A modification to the simple hill-climbing algorithm is known as steepest-gradient-search [Rich93], in which a large number of modifications are applied to the current solution to create a set of modified solutions. The modified solution with the largest increase in fitness (the steepest gradient) is selected as the next solution. The disadvantage of such a technique is the complexity of generating all possible solutions as this may require the creation of a prohibitive number of solutions. There is a trade-off between generating a suitable set of solutions and limiting the amount of time required in generating and testing those solutions.

Another disadvantage of the gradient search algorithm is due to its ‘greedy nature’ of only accepting positive moves. This can often result in it settling on the first peak it locates in the solution space; it becomes stuck in a local maximum. In a multi-modal solution space with lots of peaks this can often be a sub-optimal peak. One of the main advantages of a GA is its ability to escape local maximum points in the search space, to enable it to find a global optimum. For this reason the

performance of a gradient search is compared with the results determined by GALOPS, to illustrate that the ability to escape local maximums is a required component of a high-level low-power synthesis tool.

*10.1.1 Implementation and Results of Gradient Search*

The core of the GALOPS system is used to implement the gradient search algorithm. The high-level transformations are used to create modified solutions from a single initial solution (the specified design). The best modified-solution (the lowest power) is selected and used to generate the next set of modified solutions. The algorithm is terminated when there is no improvement in the solution (no reduction in power consumption) for 1000 iterations. The current solution will be the best solution found with the gradient search.

The results are compared with those presented in chapter 8 for the ten benchmark circuits used to illustrate the power reductions obtained with GALOPS. Figure 10.1 lists the results obtained with the gradient search technique alongside those found with the GA.

	Gradient Search			GALOPS			Gradient Power – GALOPS Power
	Power (% of original)	Area (% of original)	Supply Voltage (Volts)	Power (% of original)	Area (% of original)	Supply Voltage (Volts)	
<b>8TAPFIR</b>	31.21	109.3	2.8	15.76	441.2	1.5	<b>-15.45</b>
<b>AVEN8DI</b>	43.75	152.1	3.2	20.06	304.2	2.0	<b>-23.69</b>
<b>AVEN8PA</b>	44.74	102.0	3.3	23.82	155.8	2.3	<b>-20.92</b>
<b>DCST</b>	30.73	103.4	2.8	30.68	102.9	2.8	<b>-0.059</b>
<b>BIQUAD3</b>	31.08	105.4	2.8	30.89	104.3	2.8	<b>-0.188</b>
<b>GMLAT4</b>	43.92	147.0	3.2	32.71	152.7	2.8	<b>-11.21</b>
<b>ELLIP5</b>	75.41	100.0	4.3	69.86	58.99	4.3	<b>-5.550</b>
<b>LMS5</b>	89.36	63.17	5.0	88.90	62.73	5.0	<b>-0.459</b>
<b>VOLTERRA</b>	95.56	69.06	5.0	95.56	69.06	5.0	<b>0.000</b>
<b>ORTH2LAT</b>	100.0	100.0	5.0	97.56	162.1	5.0	<b>-2.447</b>

Table 10.1 Comparison of Results Obtained with Gradient Search and GALOPS

The table is split into 3 sections. The first section lists the results obtained with the Gradient search, including the power (as a percentage of the original), the area (as a percentage of the original) and the supply voltage. The second section lists the same three parameters as determined with GALOPS. The last section lists the difference between the power results obtained with the gradient search and GALOPS.

The table illustrates that in no case does the gradient search outperform the GA-based search performed by GALOPS. Those designs with a difference of greater than 10% (8TAPFIR, AVEN8DI, AVEN8PA and GMLAT4) all obtained a lower supply voltage with GALOPS than with the gradient search technique. GALOPS was able to reduce the critical path of these designs by a greater degree than the gradient search. For example, the 8TAPFIR critical path has been reduced to 1 element with GALOPS but only reduced to 4 elements with the gradient search. The gradient search has settled on a local optima, which prevents further optimisation of the design. GALOPS is able to escape this local optima to find the design with the shortest critical path.

Those designs with a difference in power reduction of less than 10% but greater than 0% (DCST, BIQUAD3, ELLIP5, LMS5 and ORTH2LAT) have the same supply voltage, and hence the same critical path length, as those designs synthesised with GALOPS. The difference in power consumption is due to the reduced area of the designs produced with GALOPS. These designs have critical paths that can be effectively reduced by the application of the transformations that have been implemented to target critical path length reduction. However, the reduction of area while reducing power is a more complex process that requires the GA-based search mechanism. For example, the ELLIP5 design produced by the

gradient search requires 3 adders and 2 multipliers. The design produced by GALOPS requires 1 less multiplier while still obtaining the same voltage reduction. This is a considerable saving in size and complexity when the size of a multiplier is considered (the GA produced design is 30% smaller than the gradient produced design). In the case of ORTH2LATS the GA was able to reduce power consumption (through the application of unfolding) where the gradient search failed. The best-found unfolded design required a design with higher power consumption than the original as a starting point in the unfolding process. The greedy nature of the gradient search algorithm prevented such designs from surviving in the population; hence unfolding did not improve the performance of the gradient search for this design.

The designs with a small difference in power reduction (less than 1%) have a small area difference due to the GA-synthesised designs requiring less registers than the gradient search designs. The functional block requirements (adders, multipliers, etc.) of the RTL design are the same but restructuring the DFG has created more efficient use of the variables and hence requires less physical registers i.e. there are less variables alive at the same time.

Both gradient search and GALOPS obtained the same reduction in power for the VOLTERRA design through a reduction in area. As discussed in chapter 8, the VOLTERRA design is difficult to optimise with the transformation set due to the large feedback loop that contains the critical path. A simple retiming operation on this path results in reduced hardware; this simple operation was discovered by both search techniques.

## 10.2 Simulated Annealing

Simulated Annealing (SA) [Davis87a, Reeves95], like the Genetic Algorithm, is an optimisation technique inspired by a natural process. It was first presented as an optimisation technique in the early 1980s [Kirk83] and has been used for applications such as VLSI routing [Vecchi83] and VLSI synthesis [Neil]. The algorithm simulates the cooling of a material in a heat bath – a process known as annealing. The final structural properties of the material depend upon the cooling rate. If the material is cooled too quickly the molecules of the material settle into high-energy states, resulting in a polycrystalline or brittle material. If the material is cooled more slowly a perfect crystal arrangement can be achieved, as the molecules are more likely to settle into lower energy states – cooling too slowly wastes time in determining the best solution. In terms of optimisation, determination of a low energy state is analogous to determining a global minimum (or optimum) for a problem.

The SA algorithm can be considered a modification of traditional search and optimisation techniques, such as the previously discussed gradient search. A traditional gradient search determines the optimum solution by always moving in the direction of improvement. In each step of the search, the current solution is modified to produce a set of neighbour solutions. The best solution from the set of neighbour solutions is selected as the next solution.

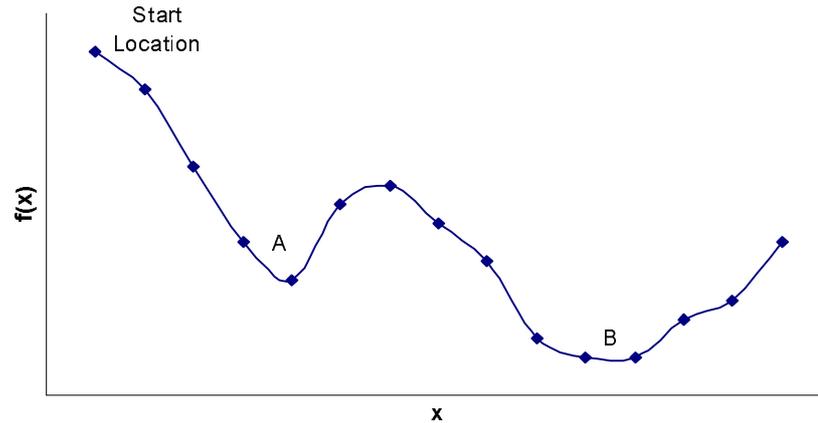


Figure 10.1 Illustration of an Optimisation Process

The main problem of this technique is its ability to become stuck in local optima, as discussed previously in section 10.1. The steepest-gradient technique is unable to select moves that temporarily result in degradation of solution quality. In problems with a range of peaks in the solution space this may result in the determination of a sub-optimal solution as illustrated in Figure 10.1. Traditional steepest-descent techniques will become stuck in the local minima at point A, unable to escape this to determine the global minima at point B.

It is clear from the example in Figure 10.1 that locating the global minima requires some uphill moves. However, as the final objective is the determination of a minimum result the uphill moves must be used sparingly and applied in a controlled manner. SA uses a control method based on a law of thermodynamics that states that at a temperature  $T$ , the probability of an increase in energy of magnitude  $\delta E$  is given by:

$$p(\delta E) = e^{-\delta E/kT} \quad (10.1)$$

where  $k$  is Boltzmann's constant. This equation is used to determine the probability of accepting an uphill move (in a minimisation problem).  $\delta E$  is the change in energy of two solutions ( $E_2 - E_1$ ). In an optimisation process the energy of a solution is its fitness in relation to the objective function.  $E_1$  is the energy of the current solution and  $E_2$  is the energy of the neighbouring solution. If  $E_2 < E_1$  then the probability will be greater than unity i.e.  $E_2$  is always selected as it is a better solution (assuming lowest fitness is best, as it is in a minimisation problem). If  $E_2 > E_1$ , then  $E_2$  is selected according to the probability  $p(\delta E)$ . Boltzmann's constant can be removed from the equation as the specification of  $T$  can take  $k$  into account at the start of the optimisation process.

The implementation of the algorithm is similar to gradient search, where an initial solution is modified to generate possible alternative solutions, selected or discarded according to  $p(\delta E)$ . The process is repeated for a certain number of iterations before  $T$  is reduced. As  $T$  is reduced, the probability of accepting a higher energy state is also reduced;  $T$  is the control parameter in the optimisation process.

The rate at which  $T$  is reduced, and the size of reduction, is determined by the cooling schedule. If  $T$  is reduced too quickly the system will become stuck in a local optima; conversely, if  $T$  is reduced too slowly the algorithm will spend an unnecessary amount of time exploring the solution space. The determination of a suitable cooling rate is one of the main problems with SA as it can have a significant effect on the results of the algorithm [Teuk95].

The algorithm is terminated when the frozen state is achieved i.e. the solution with the desired parameters is found. The SA process is effectively a gradient-descent algorithm with the addition of the capability to escape local optima

(providing the specified cooling schedule enables it). This is the primary advantage of simulated annealing over more traditional optimisation techniques.

### 10.2.1 Implementation and Results of Simulated Annealing

As with the gradient search algorithm the SA algorithm uses the core of routines developed for the GALOPS system. The pseudo-code for the SA algorithm is presented in Figure 10.2.

#### Algorithm **Simulated Annealing**

```
Passed Initial DFG Design (set to Current_Design)
Annealing Temperature Initialised To TEMP
WHILE Frozen_State not achieved

    FOR 100 design modifications
        Randomly Select a Transformation
        Apply Transformation to Current_Design to Produce New_Design
        Determine Fitness of New_Design

        SA SELECTION PROCESS
        IF New_Fitness > Old_Fitness
            Select New_Design as Current_Design
        ELSE
            Calc. Probability Of Selecting New_Design (0%-100%)
             $SA\_Prob = \exp(-((1/New\_fitness)-(1/Old\_Fitness))/TEMP)$ 

            Generate Random Probability R_Prob(0%-100%)
            IF SA_Prob > R_Prob
                Select New_Design as Current_Design
            END IF
        END SA SELECTION PROCESS
    END FOR
    Reduce TEMP by 10%
End WHILE
End Algorithm Simulated Annealing
```

Figure 10.2 Pseudo-Code for Simulated Annealing Algorithm

The SA algorithm begins by modifying the initial design using the high-level transformation techniques. The particular transformation is randomly selected from the library of possible transformations. The application of the transformations produces a new design that is either discarded or kept to replace the initial design,

depending on its relative fitness and the results of the SA probabilistic selection process. The annealing process is repeated until the frozen\_state is achieved i.e. the solution with the required specifications is generated.

Specification of the initial value of TEMP is suggested to produce a probability value of between 50% [Teuk95] and 80% [Khalifa97] for the initial iterations of the SA process. The value of TEMP needs to be scaled in relation to the typical  $\delta E$  comparisons executed throughout the process i.e. TEMP is problem specific. The initial value of  $T$  for the power minimisation problem is determined through experimental analysis; the algorithm is run for a number of generations and the average  $\delta E$  determined. This is used to calculate an initial value of  $p$  which will produce an average  $p(\delta E)$  of between 0.5 and 0.8. This analysis produced a value of 0.05, which produces an average  $p(\delta E)$  of approximately 0.6 in the initial iterations of the SA process.

TEMP is reduced by 10% every 100 generations, as suggested by the standard annealing parameters in [Teuk95]. For comparison with the GA, the frozen\_state is the production of the design with the lowest power consumption. As this cannot be specified beforehand the frozen\_state is replaced with a timeout parameter. If the best solution found so far does not improve for 5000 design modifications the search is terminated. Initial experimental results illustrated that the SA algorithm determines its optimum solution well within this timeout margin.

The performance of the SA algorithm is compared with that of GALOPS in determining the lowest power solution for the ten benchmark circuits presented in chapter 8. For each design, both GALOPS and the SA algorithm were executed 20

times to reduce the effect of both algorithm’s probabilistic nature on the results.

Table 10.2 presents the results for the benchmark designs.

	Simulated Annealing			GALOPS			Gradient Power – GALOPS Power
	Power (% of original)	Area (% of original)	Supply Voltage (Volts)	Power (% of original)	Area (% of original)	Supply Voltage (Volts)	
<b>8TAPFIR</b>	16.27	453.4	1.5	15.76	441.2	1.5	<b>-0.51</b>
<b>AVEN8DI</b>	20.36	304.2	2.0	20.06	304.2	2.0	<b>-0.30</b>
<b>AVEN8PA</b>	24.17	156.8	2.3	23.82	155.8	2.3	<b>-0.35</b>
<b>DCST</b>	30.73	103.5	2.8	30.68	102.9	2.8	<b>-0.06</b>
<b>BIQUAD3</b>	31.08	105.4	2.8	30.89	104.3	2.8	<b>-0.19</b>
<b>GMLAT4</b>	32.76	152.7	2.8	32.71	152.7	2.8	<b>-0.05</b>
<b>ELLIP5</b>	69.86	58.98	4.3	69.86	58.98	4.3	<b>0.00</b>
<b>LMS5</b>	89.68	63.17	5.0	88.90	62.73	5.0	<b>-0.78</b>
<b>VOLTERRA</b>	95.56	69.06	5.0	95.56	69.06	5.0	<b>0.00</b>
<b>ORTH2LAT</b>	100.0	100.0	5.0	97.55	162.1	5.0	<b>-2.45</b>

Table 10.2 Comparison of Results Obtained with SA and GA

The table of results is in the same format as Table 10.1. As Table 10.2 illustrates, the SA approach did not obtain a better result than the GA for any of the ten benchmark designs. Differences in power reduction between SA and GA produced designs are less than 3%. For both the SA and GA produced designs the supply voltage is the same because both search techniques have been able to reduce the critical path by the same amount. As supply voltage has the greatest impact on power consumption the power reduction is similar for designs with the same supply voltage.

The reduction of supply voltages to the same level could be due, in part, to the set of transformations used for critical path reduction. As discussed in chapter 2 the transformations were selected for their particular properties of critical path reduction. In addition, the transformations were implemented to contain inherent critical path reduction qualities, such as the pipeline transformation targeting the

critical path of the DFG for the insertion of pipeline stages. The transformation set, together with a heuristic search technique, is capable of significantly reducing the critical path length of the benchmark DFGs.

The power differences are primarily due to area differences in the designs. The area differences point to the fact that the GA produced designs that require less registers (hence less area and capacitance) than the SA produced designs.

The typical Pareto-surface chart (in chapter 9) is useful for explaining the differences in power consumption. Both techniques (SA and GA) reduce the critical path to the same length. However, the GA is also able to optimise the area of the designs, effectively moving further left across the area axis of designs with same critical path length, as illustrated in Figure 10.4.

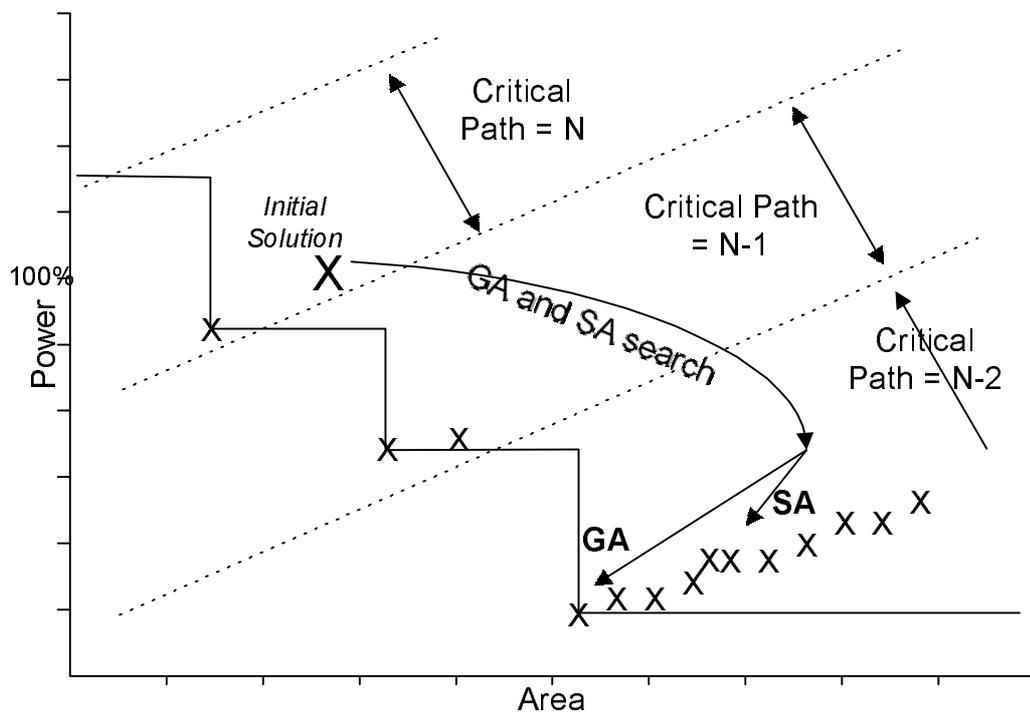


Figure 10.3 Comparison of GA and SA Search

Figure 10.4 is a representation of a typical GA and SA search process. Both techniques are able to reach the same region of the solution space that contains the solutions with the shortest critical path lengths. The selected set of transformations enables both techniques to determine the low-voltage region of the solution space. However, the GA is able improve on these results by locating the small area designs within that region of the solution space; hence producing lower power designs. These are typically designs that require fewer physical registers than the SA designs. As well as offering lower power consumption and smaller area, designs with fewer registers reduce the complexity of the layout and routing tasks of the VLSI synthesis process, as there are fewer elements to process.

As described in chapter 9, an advantage of the GA is its ability to provide information about the solution space, particularly the generation of Pareto-surface data to illustrate trade-offs between competing parameters such as area and power. Figure 10.4 shows two Pareto-surface charts for the non-unfolded synthesis of the 8TAPFIR design. One chart was generated with the GA (GALOPS) and the other generated with the SA search mechanism. As an example point on the chart, the GA produces a design with 18.2% power consumption at  $9.1\text{mm}^2$  while the SA reports that designs of that area consume 24.3% of the original power. In addition, the GA pareto-surface is characterised with 10 pareto points while the SA pareto-surface contains 6 points, the GA presents a more comprehensive set of trade-off points to the VLSI designer. The ability of the GA to determine smaller designs than the SA search mechanism is exploited throughout the search of the solution space, thus the GA produces better pareto-surface charts with better area-power trade-offs.

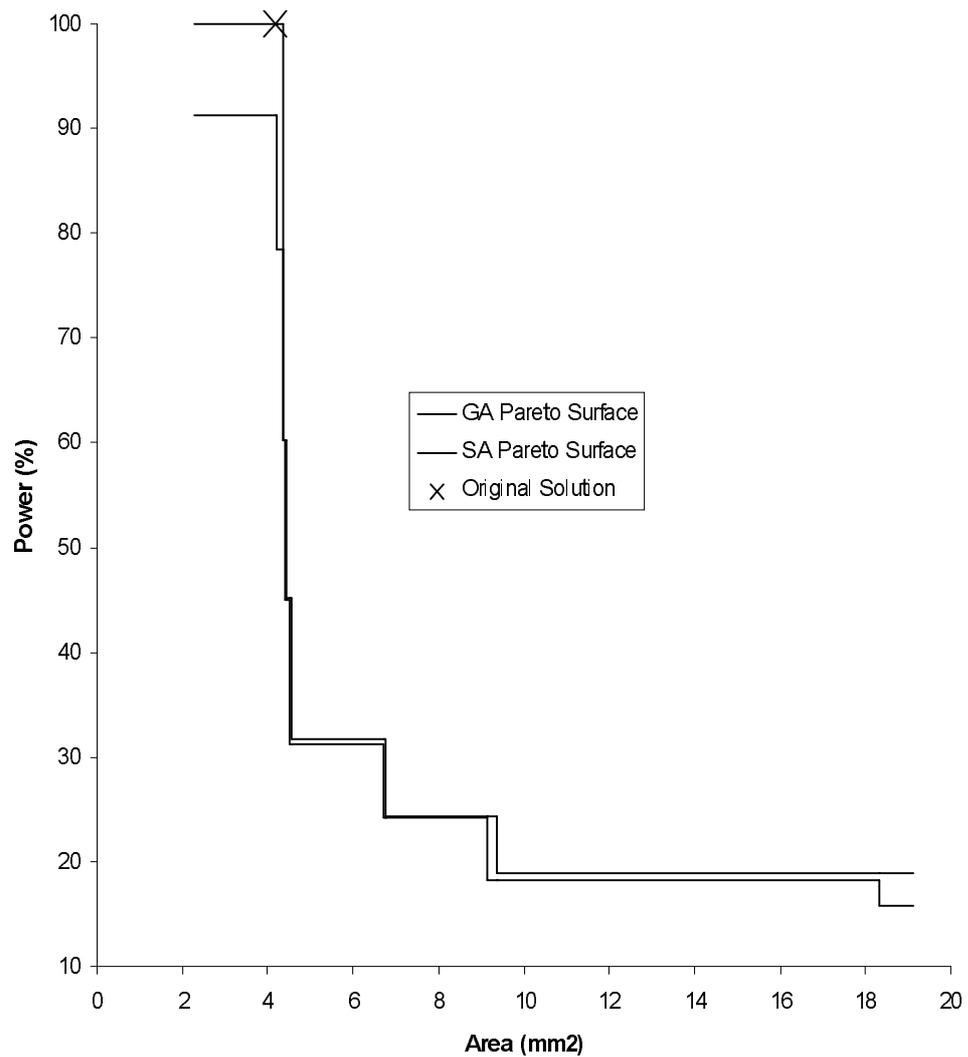


Figure 10.4 Pareto-Surface Information Generated with SA and GA

### 10.3 Conclusion

The GA-based search technique used in GALOPS has been compared with two alternative search techniques. The greedy nature of the gradient search algorithm, though operating with a population of multiple solutions, failed to determine a set of solutions with the same power reductions as the GA. The nature of the low-power solution space requires a system that can suffer degradation in quality of solutions to ultimately lead to an improvement. Therefore, gradient

search techniques that always move in the direction of greatest improvement over the search space will become stuck in local optima. Where the supply voltage obtained with both techniques was the same the results highlight the ability of the GA to determine low-area implementations for that supply voltage in comparison with the gradient search technique.

The SA and GA techniques produced designs with the same speeds and hence the same supply-voltages, resulting in power reductions of the same order of magnitude. The production of designs with the same supply voltage illustrates the effectiveness of the transformations for critical path reduction, due in part to the selection of transformations for that specific purpose and the implementation of the transformations to target critical path reduction. As with the gradient search, where designs with the same supply voltage were determined, the GA still produced lower power consumption in most cases due to reduction of the area. The properties of the transformations ensure that supply voltage reduction through speed-up transformations is achieved. The heuristic properties of the SA and GA methods take advantage of the transformation properties to produce fast designs and hence low supply voltages. However, the GA provides low voltage operation while additionally optimising the area of the design, hence the difference in power consumption. While both techniques can achieve significant power reductions, the specific properties of the GA are required to gain the extra power reduction through area reduction. The GA is able to determine the ‘hard’ solutions that the other algorithms cannot.

In addition, the analysis of Pareto-surface information illustrated the superiority of the GA in providing invaluable information about the solution space. Therefore, the GA not only provides better point solutions but it is also a better tool

for determining important trade-offs between area and power so that the VLSI designer can select the solution which best meets the specified requirements.

The implementation of the gradient search and SA search mechanisms used many of the core algorithms developed for the GALOPS system, including the chromosome DFG representation and the implementation of the transformations for manipulation of the chromosomes. The gradient and SA search techniques presented in this chapter also required the transformation-mutation operations that were developed as part of this work. The implementation of the gradient and SA algorithms would not have been possible without the development of the transformation and search mechanisms which form the core of the GA-based search mechanism.

## **Chapter 11 – Conclusions and Future Work**

This Chapter details the main conclusions of the research presented in this thesis. Section 11.1 highlights the primary contributions of this work to the fields of low-power design and VLSI synthesis using GAs. Section 11.2 gives an overview of the developed system, GALOPS, in terms of the implemented source code. Section 11.3 discusses the results obtained with the GALOPS system in the context of low-power design implications. Section 11.4 presents a summary of the conclusions derived from this work. Finally, section 11.5 discusses future directions and developments for the work initiated in this thesis.

### **11.1 Primary Features of This Work**

The primary, original contribution of this thesis is the development of a novel low-power design tool based around a core GA search and optimisation technique. The tool, dubbed GALOPS (Genetic Algorithm for Low Power Synthesis), targets low-power implementation of DSP algorithms.

The summary of power optimisation techniques at all levels of the design process, in Chapter 2, was used to illustrate that targeting power as a high-level objective parameter, alongside the traditional area and speed criteria, offers the greatest benefits in terms of power reduction. Thus, the developed tool targets power reduction at the behavioural level, modifying high-level algorithms using behavioural transformations.

Behavioural level optimisation required the development of a non-standard chromosome representation that was specifically developed for the task of high-level power conscious design. The chromosome does not use standard binary or alphabet chromosome representation but incorporates the features of a standard signal processing structure, the Data Flow Graph. The flexible nature of the chromosome ensures it can represent a wide range of DFGs of varying size and complexity, both initial designs and those generated throughout the search and exploration process.

Chapter 5 illustrated that application of standard genetic operators, during the search process, would result in corrupt designs with incorrect functionality. Rather than implement a repair operation, which increases computation time and can reduce the efficiency of the search technique, problem-specific VLSI design techniques are incorporated into the GA. This work has developed problem specific genetic operators to embed high-level transformations in a genetic exploration framework. This enables the advantageous properties of a GA, as described in Chapter 4, to be applied to the problems of high-level low power design. The developed operators enable the behavioural level transformations to be used to explore the low power solution space. The operators apply the transformations so as to preserve algorithm functionality and specifically target speed increase for low voltage operation.

Chapter 10 presented the development of alternatives to the GA-based search technique. These algorithms were implemented using the core of the GA-based technique, incorporating the transformation application and solution manipulation routines developed as part of the GA tool. A comparative analysis of the search algorithms was performed to illustrate the superiority of the GA in

determining the best-found low power solution. The ability of the GA to provide trade-off information to the design engineer was also presented, again illustrating the advantages of using a GA for high-level design and exploration.

### 11.2 System Implementation

The GALOPS system was implemented in the C programming language. The tool was designed and tested using the Microsoft Developer Studio platform running under the Windows 95 operating system. The PC platform was a 100MHz Pentium for development with 32 Megabytes of RAM. For testing and operation the results presented in this thesis were produced with a Pentium-II 333MHz with 64 Megabytes of RAM running under the Windows NT operating system.

The GALOPS tool comprises approximately 7500 lines of code, broken down into the following routines:

- The main core of the program, comprising program flow control, the user interface and system administration routines such as disk access, memory management, etc. The core also contains the standard GA routines such as parent selection, roulette wheel construction, etc.– Approx. 1750 lines.
- Routines to interpret and compile the DFG input files into the chromosome representation – Approx. 200 lines
- Routines for the fitness calculation procedures – Approx. 900 lines.
- Routines to determine the length and components of the critical path of a DFG – Approx. 200 lines.
- Routines for the calculation of the capacitance and voltage of the designs – Approx. 830 lines.

- The retime and back\_retime transformations – Approx. 270 lines.
- The pipeline and remove\_pipeline transformations – Approx. 460 lines.
- The automatic pipeline transformation – Approx. 130 lines.
- The unfold transformation – Approx. 390 lines.
- The implementation of the problem specific crossover operator – Approx. 970 lines.
- Various program control, test, IO and characterisation routines – Approx. 1340 lines.

### 11.3 Discussion

The comparative analysis of original and optimised designs was performed with the use of high-level power estimation strategies. The difficulties and properties of behavioural level power estimation were discussed in Chapter 3. Behavioural level power estimation is a complex, and typically inaccurate process, due to the large amount of undecided VLSI implementation variables. The prototype version of GALOPS uses a technique similar to PFA where pre-characterised functional modules and operation counts are used to estimate capacitance. Voltage is estimated from a model of the effects of voltage reduction on device speed. This enabled the GA to perform comparative evaluations between different solutions during the search process. Chapter 5 illustrated the prototype version of GALOPS on a set of varying complexity signal processing benchmarks. The results illustrate that GALOPS was able to reduce the power consumption of all designs.

The prototype version of GALOPS utilised a relatively simple technique for power estimation. Chapter 6 details the integration of more complex high-level area and power estimation routines into the system. The routines incorporate high-level design tasks such as scheduling and allocation to increase the accuracy of the power estimation module. In addition, the results obtained with the improved module are more relevant to standard signal processing design systems that incorporate such high-level design tasks.

GA research has yielded a body of techniques that can be used to improve the efficiency of a search and the results obtained. Unfortunately, no guiding metric is available to select techniques for any particular application. Therefore Chapter 7 investigates techniques with the aim of improving GA performance. Elitism is a selection scheme intended to exploit information currently in the GA population pool by ensuring a set of the best solutions is always present in the pool. Elitism was investigated for different application rates and the results indicate that a certain level of elitism improved the results obtained, compared to not applying elitism or applying it at too great a rate. A similar analysis for the Ranked Selection Technique illustrated its superiority over fitness proportionate selection for this problem. Chapter 7 also examined the Taguchi method for optimal GA parameter settings. The results illustrated the difficulty of determining a global set of parameters for a GA that is designed to process a wide range of problems with different properties.

The modified GALOPS tool was analysed with a set of ten benchmark circuits that cover a range of signal processing algorithms of varying complexity. The results illustrated the success of the tool in obtaining low-power implementations. The amount of power reduction obtained is very dependent on the

properties of each individual algorithm. The amount of recursion in an algorithm can limit the application of the transformation set and place a bound on the minimum critical path length hence restricting voltage reduction techniques.

The generated low-power solutions typically require supply voltages below the industry standards of 5V and more commonly 3.3V. As discussed in Chapter 8, the required voltages could be achieved with small, on-chip level converters. These converters enable the chip to source from a standard 3.3V power supply (to integrate with other devices in a system) while the core of the chip operates at the required low-power voltage. As discussed in Chapter 8, such systems have been used in the development of practical DSPs and microprocessors.

The low-power solutions presented in Chapter 7 typically require an area increase compared to the standard implementation of their non-optimised original designs, which has obvious implications for the fabrication of low-power devices. With decreased feature sizes and increased die sizes, the number of transistors that can be placed on a single chip is increasing. This extra capacity was traditionally used to increase the performance of the device. The results illustrate that where the required performance constraint has been achieved the increased number of available transistors can be traded for low power through reduced supply voltage. In addition, the development of Multi-Chip Modules (MCMs) promises to provide devices with giga-scale integration levels, which increases the potential to trade-off area for low-power and low-voltage operation.

The results obtained in Chapter 8 are not compared with other high-level power conscious design tools. The majority of 'high-level' systems, such as Power Profiler [Martin95, Martin96], Design Power [Synop98] and others [Rag95, Rag95a, Rag95b] operate during the translation of behavioural level descriptions

into RT-level architectures. GALOPS operates on the behavioural description itself, enabling subsequent application of the techniques applied in these tools. GALOPS is not intended as a competitor or replacement for these tools, rather as a step in the low-power design process that will incorporate techniques and design tools that operate at all levels, to maximise total power reduction.

As with GALOPS, the HYPER-LP system [Chan95, Chan92a] considers power at the behavioural level by optimising the high-level algorithms. However, the results presented for HYPER-LP use power analysis techniques significantly different to those used in GALOPS. In addition, the HYPER-LP system applies a different set of high-level algorithmic transformations; therefore, differences in results could be due to the set of transformations contained within the system rather than the method of applying that set. In addition to differences due to the optimisation technique used, the power analysis method could also account for differences in the results obtained with different systems. The problem is that no standard technique exists for benchmarking the performance of high-level optimisation tools. The presentation of a range of benchmark designs in this thesis and the publications arising from this thesis, is an attempt to address this problem of comparison between different high-level design techniques. By presenting a range of designs it is hoped that researchers can use them to compare the performance of different high-level optimisation and analysis techniques.

One of the core optimisation techniques used in HYPER-LP is Simulated Annealing (SA) for the application of certain transformations, hence the comparison between GALOPS and a SA-based search. SA is a search process inspired by a natural physical phenomenon that occurs during the cooling of materials. It has been used to solve complex problems of VLSI design such as circuit-level routing.

Unlike gradient search it incorporates a mechanism for escaping local maxima i.e. it can suffer temporary degradation in solution quality during the search. In Chapter 10 the performance of the GALOPS system was compared with an SA algorithm that utilised the same set of transformations and power analysis methods. The results for SA compared to GALOPS show that both SA and GALOPS can reach the minimum voltage area of the solution space. However, GALOPS improves on the power reduction achieved through minimum-voltage design by refining the solution to require fewer resources and less switched capacitance. A comparison of Pareto-surfaces generated with GALOPS and the SA algorithm also demonstrate that GALOPS provides more comprehensive trade-off information, characterised by more points and lower power values at comparative areas. Therefore GALOPS not only produces lower power single-point solutions but also presents more accurate trade-off charts when compared to SA.

The results for the benchmark designs illustrate that low-power implementation typically requires a considerable increase in area. This is due to the low power design technique used where increase in speed is traded for a reduction in supply voltage. This speed increase is achieved through increasing and exploiting the concurrency in the algorithm. Processing more operations at the same time often requires more hardware units hence the increase in area. This typically results in the lowest power solution also being a large area solution. Rather than present a single lowest power (and hence large area) solution, Chapter 9 exploits the multi-solution nature of the GA search mechanism to provide trade-off information to the VLSI designer. The information, presented in the form of Pareto-surfaces, illustrates the lowest power solutions across the area range. This enables the designer to select the solution that best meets the implementation requirements, rather than concentrating

on the minimisation of a single parameter. In addition, the Pareto-surfaces are useful in examining the nature of the solution space, illustrating the large effect that speed increases can have on power reduction as well as the trend of decreased power requiring larger area. The GALOPS tool is the first behavioural-level power-optimisation tool to present such trade-off information as part of the optimisation process, exploiting the inherent characteristics of the chosen GA search technique.

Chapter 10 compares the performance of the GA with two other search techniques for optimising complex problems – steepest gradient and simulated annealing (SA). Gradient search is a greedy algorithm that searches the solution space only selecting solutions that improve on the current one. The results from this search illustrate that the low power solution space is a complex, multi-modal problem with non-optimum peaks. That is, determination of the highest peak in the solution space requires a search mechanism that can suffer degradation in solutions so as not to become stuck on a local maximum.

### 11.4 Conclusions

- This dissertation has presented the first use of a Genetic Algorithm for the design and optimisation of behavioural level signal-processing algorithms intended for low-power operation.
- The problem-specific chromosome representation enables application of high-level VLSI design rules to explore the low power solution space without corrupting algorithm functionality. The chromosome's flexibility enables the tool to process a wide range of signal processing algorithms.
- The development of problem-specific genetic operators, to incorporate the

high-level algorithmic transformations, enables traditional high-level design techniques to be incorporated into a genetic search and optimisation framework; hence enabling high-level design to utilise the advantages of GAs.

- The presented tool, dubbed GALOPS, successfully reduces the power consumption of a wide range of signal processing benchmark designs of varying complexity.
- The presented results illustrate that low-voltage operation, when initial maximum throughput is a constrained parameter, requires full exploitation of the available concurrency in a signal-processing algorithm. In Application-Specific signal processors this requires utilisation of maximum resources to minimise the critical path length of the design.
- The use of a transformation library to supply the GA provides a framework for the addition of new transformations, to investigate their power reducing applications or to determine the effect of interaction between transformations on providing the best solution.
- The multi-solution nature of the GA enables it to provide trade-off information to the VLSI designer to meet objectives in addition to the reduction of power (such as area constraints).
- Incorporation of GA techniques such as Elitism and Ranking improve the results when compared to a basic GA implementation.
- Development of alternative heuristic search algorithms utilising the core transformation application and solution manipulation routines developed for the GA-based search. A comparison of the GA with Gradient Search and Simulated Annealing illustrates the superiority of the GA-based search, both in presenting

single low-power solutions and a range of area-power trade-offs.

### 11.5 Future Work

On the circuits side the problems of accurate and fast high-level power estimation have been discussed. Further refinements to the power analysis module, such as the consideration of switching capacitance and increased numbers of characterised functional units, could improve the accuracy of the analysis. However, the improved accuracy must not incur too great a speed penalty as to render the optimisation process prohibitive in terms of time.

The current system processes high-level algorithms described as Data Flow Graphs, which are compiled into the internal GA chromosome representation. External compilers could be developed to process VHDL and Verilog high-level descriptions as these languages are frequently used in high-level design systems. This would improve the applicability of the tool in practical design processes.

The extension of the standard GA, to incorporate other genetic features such as Ranking and Elitism, was shown to improve the performance of the GA. The wealth of GA research contains a large body of techniques for enhancement that could not all be investigated in this work due to time and space constraints. Selection techniques such as Tournament Selection [Hanc95] and Stochastic Remainder [Hanc96] could further improve the efficiency and results.

A related area is the extension of the crossover operator, which is currently a localised operator that considers the crossover of single transformations. Further crossover operators could transfer multiple transformations and also contain the ability to recognise transformation histories i.e. the specific set of transformations that produce the current solution.

As discussed in Chapters 1 and 2, successful power reduction should target power reduction at all levels of the design process. As such, GALOPS should be integrated into a power-conscious design framework incorporating power analysis and optimisation techniques at all levels. Architectural-, logic- and circuit-level techniques, described in Chapter 2, could be used subsequent to GALOPS to provide a complete automated low-power design process.

One of the most interesting developments in evolutionary design of recent years has been the development of evolvable hardware. The evolutionary-inspired processes typically used to design fixed hardware can be incorporated onto a programmable device. The optimisation process can be performed in real-time to produce a design that meets the current specifications with optimal objectives. The evolvable device reconfigures itself to provide the optimum solution to the current application. The recent research into dynamically variable voltages can be incorporated into a GALOPS system on a re-configurable signal-processor chip. The desired algorithm function can be optimised to meet the current throughput requirements, with voltage levels set to provide the minimum power implementation. This would result in a general-purpose signal-processing device that achieves optimal low power operation dependent on the current throughput and functional requirements.

The current GALOPS system is an implementation of a generational GA where the entire population is replaced once per cycle (a generation). Steady-state GAs keep a constant population, removing and adding a specific amount of new solutions once per cycle in an attempt to improve the exploitation of the knowledge currently stored in the population. Some research into steady-state GAs has produced encouraging results that indicate the superiority of this approach for some

problems [Whit89]. The GALOPS system could be modified to examine the advantages of applying steady-state GA methods.

Parallel GAs (PGAs) are an evolution of the standard GA which effectively employs parallel implementations of a GA or a single GA maintaining parallel populations. One of the main advantages of PGAs is the exploitation of parallel processing computers and environments. A parallel GA can operate on a number of processing systems to increase the speed of the search and optimisation process. However, PGAs do not only offer increased speed. Different implementation styles use the parallel framework to improve the success of the search. Migration of individuals is used to place members of one population into another. The different populations can be running under different objective criteria or different operating parameters to enable the GA to reduce problems such as premature convergence and multi-objective optimisation criteria. Modifying a standard GA to become a parallel GA is a non-trivial task, especially where the GA is expected to run on a parallel processing system. Investigation of PGAs may improve the results obtained with GALOPS.

The Pareto-surface generation uses the data produced during the GA search and optimisation process. This could be further refined by incorporating Multi-Objective GA (MOGA) techniques such as niching and fitness sharing. Such techniques can improve the ability of the GA to fully explore the range of available trade-offs.

The set of transformations currently incorporated in GALOPS are a subset of transformations, specifically selected to increase system speed. The implementation of GALOPS, to access a library of transformations, enables relatively simple application of other transformations such as distributivity,

associativity, etc. which need to be investigated for their power reducing characteristics.

The GA has been compared with a number of techniques that target optimisation of combinatorial problems. There are many other techniques such as Tabu Search, Newton-Raphson, Integer Linear Programming, Lagrangian Relaxation [Reeves95], etc. The development of SA and Gradient search techniques illustrate that the GA-based framework provides a core set of routines for the implementation of alternative search and optimisation techniques. The GA framework could be used to investigate the efficiency of these other techniques for high-level design.

In terms of evaluation of the tool, the fabrication of the devices will enable practical examination of the power reductions and the associated implications for area, speed and functionality.

The user interface of the GALOPS tool is a relatively simple textual interface that prompts the user for information at the required moments. Although a simple script-driven system has been developed, to enable a set of results (complete synthesis runs) to be obtained from a single execution of the program, a practical version of the tool will require a considerably more advanced user interface. This was not tackled as part of this research project as the user interface is not traditionally seen as a ‘novel’ or ‘interesting’ research avenue. The DFG designs presented in Appendix A were obtained with a specifically developed tool. An extension to this tool would be to provide a graphical front-end to the GALOPS system, incorporating graphical capture of the designs as well as a windows-based operating environment.

## References

- [Ajluni95] C. Ajluni, "Flat-panel displays strive to cut power," *Electronic Design*, January 9 1995, pp. 88-90
- [Arslan95] T. Arslan, D.H. Horrocks and A.T. Erdogan, "Overview and design directions for low-power circuits and architectures for digital signal processing," *IEE Colloquium (Digest)*, No. 122, 1995, pp. 6/1-6/5
- [Arslan96] T. Arslan, E. Ozdemir, M.S. Bright, D.H. Horrocks, "Genetic synthesis techniques for low-power digital signal processing circuits," *Proc. IEE Colloquium on Digital Synthesis*, London, UK, 15th Dec 1996, pp. 7/1-7/5
- [Arslan96a] T. Arslan, D.H. Horrocks, E. Ozdemir, "Structural cell-based VLSI circuit design using a genetic algorithm," *Proc. IEEE Int. Symposium On Circuits and Systems*, Atlanta, USA, 1996, vol. 4., pp.308-311
- [Arslan96b] T. Arslan, D.H. Horrocks, E. Ozdemir, "Structural synthesis of cell-based VLSI circuits using a multi-objective genetic algorithm," *IEE Electronic Letters*, Vol. 32, No. 7, 28th March 1996, pp. 651-652
- [Arslan96c] T. Arslan, M.J. O'Dare, "Transitional gate delay detection for combinational circuits using a genetic algorithm," *IEE Electronics Letters*, vol. 32, no. 19, Sep 12th 1996, pp. 1748-1749
- [ASC] Alternative Systems Concepts, Inc., "Low power synthesis and optimization," Available *HTTP*  
[http://www.ascinc.com/products/low\\_power/](http://www.ascinc.com/products/low_power/)
- [Ash95] P.J. Ashenden, *Designers guide to VHDL*, Los Angeles, USA: Morgan Kaufman Publishers, 1995
- [Baker85] J.E. Baker, "Adaptive selection methods for genetic algorithms," *Proc. 1st Int. Conf. On Genetic Algorithms and Their Applications*, PA, USA, July24-26, 1985, pp.101-111
- [Beasley93] D. Beasley, D.R. Bull, R.R. Martin, "An overview of genetic algorithms : part 1, fundamentals," *University Computing*, 1993, 15(2) pp. 58-69

- [Beasley93a] D. Beasley, D.R. Bull, R.R. Martin, "An overview of genetic algorithms: part 2, research topics," *University Computing*, 1993, 15(4), pp. 170-181
- [Bella95] A. Bellaouar and M.I. Elmasry, *Low-Power Digital VLSI Design: Circuits and Systems*, Mass., USA: Kluwer Academic Publishers, 1995
- [Bella95a] A. Bellaouar and M.I. Elmasry, "Low-power VLSI design: an overview," in *Low-Power Digital VLSI Design: Circuits and Systems*, Mass., USA: Kluwer Academic Publishers, 1995, Chapter 1, pp. 1-12
- [Bella95b] A. Bellaouar and M.I. Elmasry, "Low-power VLSI design methodology," in *Low-Power Digital VLSI Design: Circuits and Systems*, Mass., USA: Kluwer Academic Publishers, 1995, Chapter 8, pp. 490-526
- [Bentz] O. Bentz., "Hyper: synthesis for datapath intensive architectures," Available *HTTP*  
<http://infopad.EECS.Berkeley.EDU/~hyper/Background/paperBackground.html>
- [Bentz97] O. Bentz, J.M. Rabaey and D. Lidsky, "A dynamic design estimation and exploration environment," *Proc IEEE Design Automation Conference, DAC 97*, Anaheim Calif. USA, 1997, pp. 190-195
- [Bhask90] J. Bhasker and H-C. Lee, "An optimizer for hardware synthesis," *IEEE Design and Test of Computers*, Oct 1990, pp. 20-36
- [Blair94] G.M. Blair, "Designing low power digital CMOS," *Electronics & Communication Engineering Journal*, Oct 1994, pp.229-236
- [Bright95] M.S. Bright, *Fault Analysis of VLSI Circuits*, B.Eng. Dissertation, School of Engineering, Cardiff University, 1995
- [Bright96] M.S. Bright, T. Arslan, "A genetic framework for the high-level optimization of low power VLSI DSP systems," *IEE Electronics Letters*, Vol. 32, No. 13, 20th June 1996, pp. 1150-1151
- [Bright97] M. S. Bright and T. Arslan, "A genetic algorithm for the high-level synthesis of DSP systems for low power," *Proc. IEE/IEEE Conf. on Genetic Algorithms in Engineering Systems, Innovations and Applications (GALESIA '97)*, Glasgow, UK, 2-4 Sept. 1997, pp. 174-179

- [Bright98] M. S. Bright and T. Arslan, "Transformational-based synthesis of VLSI based DSP systems for low power using a genetic algorithm," *IEEE Int. Symposium on Circuits and Systems, ISCAS 98*, Monterey CA, 31 May - 3 June 1998
- [Bright98a] M.S. Bright and T. Arslan, "Supply voltage reduction through high-level design techniques," *Proc. IEE UK Low Power Forum*, Sheffield, UK, Sept. 1998, pp. 10.1-10.5
- [Bright98b] M.S. Bright and T. Arslan, "Low-power high-level DSP system methodologies and techniques: impact on CAD," *Proc. IEE UK Low Power Forum*, Sheffield, UK, Sept. 1998, pp. 7.1-7.5
- [Burch93] R. Burch, F. N. Najm, P. Yang, T. N. Trick, "A Monte-Carlo approach for power estimation," *IEEE Trans. on VLSI Systems*, vol. 1, no. 1, March 1993, pp. 63-71
- [Burd96] T.D. Burd and R.W. Broderson, "Processor design for portable systems," *Journal of VLSI Signal Processing*, Kluwer Academic Publishers; Volume 13, Numbers 2/3, August Sept. 1996, pp. 203-222
- [Bursky95] D. Bursky, "Power-reduction schemes promise "cool" digital ICs," *Electronic Design*, January 9 1995, pp. 51-64
- [Camp90] R. Camposano, "From behavior to structure: high level synthesis," *IEEE Design and Test of Computers*, Oct 1990, pp. 8-19
- [Camp91] R. Camposano, "Path-based scheduling for synthesis," *IEEE Trans. On Computer Aided Design*, vol. 10, no. 1, Jan. 1991, pp. 85-93
- [Casa80] A. E. Casavant, D. D. Gajski and D. J. Kuck, "Automatic design with dependence graphs," *Proc. 17th ACM/IEEE Design Automation Conference*, Minneapolis, June 1980, pp. 506-515
- [Chan92] A.P. Chandrakasan, S. Sheng and R.W. Broderson, "Low power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, April 1992, pp. 473-483
- [Chan92a] A.P. Chandrakasan, M. Potkonjak, J. Rabaey, R.W. Broderson, "HYPER-LP: A system for power minimization using architectural transformations," *Proc. IEEE Int. Conference on CAD*, 1992, pp. 300-303
- [Chan94] A.P. Chandrakasan, A. Burstein, R.W. Broderson, "A low-power chipset for a portable multimedia I/O terminal," *IEEE Journal Of Solid-State Circuits*, vol. 29, no. 12, Dec 1994, pp. 1415-1428

- [Chan94a] A.P. Chandrakasan, R. Allmon, A. Stratakos and R.W. Broderson, "Design of portable systems," *Proc. IEEE Custom Integrated Circuits Conf.*, USA, 16th May 1994, pp. 259-266
- [Chan95] A.P. Chandrakasan, M. Potkonjak, R. Mehra, J.M. Rabaey and R.W. Broderson, "Optimizing power using transformations," *IEEE Transactions On Computer Aided Design Of Integrated Circuits and Systems*, vol. 14, no. 1, Jan 1995, pp. 12-31
- [Chan95a] A.P. Chandrakasan, R.W. Broderson, "Minimizing power consumption in digital CMOS circuits," *Proc. of the IEEE*, vol. 83, no.4, April 1995, pp.498-523
- [Chan98] A.P. Chandrakasan, R. Amirthajarah, J. Goodmand, W. Rabiner, "Trends in low power digital signal processing," in *Proc. IEEE International Symposium on Circuits and Systems , ISCAS '98*, Monterey, CA, 1998
- [Chang95] J-M. Chang, M. Pedram, "Register allocation and binding for low power," *Proc. IEEE/ACM 32nd Design Automation Conference*, San Francisco, CA, June 1995, pp. 29-35
- [Chau92] P.M. Chau, S.R. Powell, "Power dissipation of VLSI array processing systems," *Journal of VLSI Signal Processing*, vol. 4, 1992, pp. 199-212
- [Chaud95] S. Chaudhuri, S.A. Blythe, R.A. Walker, "An exact methodology for scheduling in a 3D design space," *Proc. 8th IEEE Int. Symp. on System Synthesis*, Cannes, France, Sept 1995, pp. 78-83
- [Chaud96] S. Chaudhuri, R.A. Walker, "Computing lower bounds on functional units before scheduling," *IEEE Trans. on VLSI Systems*, vol. 4., no. 2, June 1996, pp. 273-279
- [Chipp97] A. Chipperfield, "Introduction to Genetic Algorithms - Parallel GAs," in *Genetic Algorithms in Engineering Systems*, London UK: IEE, 1997, Chapter 1, pp. 20-30
- [Chiu94] C.T. Chiu, K.H. Tsui, "VLSI implementation of a generic discrete transform processor for real-time applications," *Proc. of IEEE Asia-Pacific Conf. on Circuits and Systems*, 1994, pp. 79-84.
- [Chuang98] C-T. Chuang, S-L. Lu, K. Soumyanath, H. Partovi, T. Sakurai, V. De, K. Roy, "Challenges for low-power and high-performance chips," *IEEE Design & Test of Computers*, July-Sept. 1998, pp. 119-124
- [Chun94] J.G. Chung, K.K. Parhi, "Pipelining of lattice IIR digital filters," *IEEE Trans. on Signal Processing*, vol. 42, no. 4, April 1994. pp. 751-761

- [Chwirka95] S. Chwirka, "Power analysis tools evolve for portables," *Electronic Design*, January 9 1995, pp. 113-116
- [Cirit87] M. A. Cirit, "Estimating dynamic power consumption of CMOS circuits," *IEEE Int. Conf. Computer-Aided Design*, Nov. 9-12, 1987, pp. 534-537
- [Coudert96] O. Coudert, R. Haddad, K. Keutzer, "What is the state of the art in commercial EDA tools for low power?," *Proc. of Int. Symposium on Low Power Electronics and Design*, Monterey, CA, USA, 1996, pp. 181-187
- [Darwin] C. Darwin, *On the origin of species*, Available *FTP* <ftp://sunsite.unc.edu> Directory: /pub/docs/books/gutenberg/etext98/ File: otoos10.txt
- [Davis82] A.L. Davis, R.M. Keller, "Data flow program graphs," *IEEE Computing Magazine*, 1982, Feb 15, pp. 26-41
- [Davis87] L. Davis (Ed.), *Genetic algorithms and Simulated Annealing*, Morgan Kaufman, LA, USA, 1987
- [Davis87a] L. Davis and M. Steenstrup, "Genetic algorithms and simulated annealing: an overview," in *Genetic Algorithms and Simulated Annealing*, Morgan Kaufman, LA, USA, 1987, pp.1-11
- [Davis89] L. Davis, "Adapting operator probabilities in genetic algorithms," *Proc. 3rd Int. Conf On Genetic Algorithms*, CA, June 1989, pp. 61-69
- [Davis91] L. Davis (Ed.), *Handbook Of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991
- [Davis94] M. Davis, L. Liu, J.G. Elias, "VLSI circuit synthesis using a parallel genetic algorithm," *Proc. IEEE Conf. On Evolutionary Computation*, vol. 1, 1994, pp. 104-109
- [DeJong] K. DeJong, *Analysis of a Class of Genetic Adaptive Systems*, Ph.D. Thesis, Dept. of Computer and Communications Science, Univ. of Michigan, 1975
- [Dev89] S. Devadas, R. Newton, "Algorithms for hardware allocation in data path synthesis," *IEEE Trans. On Computer-Aided Design*, vol. 8, no. 7, July 1989, pp.768-781
- [Dev95] S. Devadas, S. Malik, "A survey of power optimization techniques targeting low power VLSI circuits," *Proc. IEEE Design Automation Conference*, San Francisco, CA, 1995, pp. 242-247

- [Esb96] H. Esbensen and E.S. Kuh, "Design space exploration using the genetic algorithm," *Proc. IEEE Int. Symposium on Circuits and Systems, ISCAS 96*, Atlanta, USA, May 1996, pp. 500-503
- [Esh91] L.J. Eshelman, "The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination", in *Foundations of Genetic Algorithms*, G.J.E. Rawlins (Ed.), San Mateo, CA: Morgan Kaufmann, 1991, pp. 265-283.
- [Fett93] G.P. Fettweis, L. Thiele, "Algebraic recurrence transformations for massive parallelism," *IEEE Trans. On Circuits and Systems I: Fundamental Theory and Applications*, vol. 40, no. 12, Dec 1993, pp. 949-952
- [Filho94] J.R. Filho, C. Alippi and P. Treleaven, "Genetic algorithm programming environments," *IEEE Computer*, vol. 27, number 6, June 1994, pp. 28-43
- [Frenk97] J. Frenkil, "Tools and methodologies for low power design," *Proc. IEEE Design Automation Conference, DAC'97*, California, USA, 1997, pp. 76-81
- [Fried94] E.G. Friedman, "From 100 milliwatts/mips to 10 microwatts/mips," *Proc. IEEE Int. Symposium on Circuits and Systems*, London, UK, May 1994, pp. 1-6
- [Gajski94] D.D. Gajski, L. Ramachandran, "Introduction to high-level synthesis," *IEEE Design and Test of Computers*, Winter 1994, pp. 45-54
- [Galesia95] *Genetic Algorithms in Engineering Systems: Innovations and Applications*, Conf. Proc., GALESIA '95, IEE/IEEE, Univ. Sheffield, UK, 12-14 Sept. 1995
- [Galesia97] *Genetic Algorithms in Engineering Systems: Innovations and Applications*, Conf. Proc., GALESIA '97, IEE/IEEE, Univ. of Strathclyde, UK, 2-4 Sept. 1997
- [Gary94] S. Gary, P. Ippolito, G. Gerosa, D. Dietz, J. Eno, H. Sanchez, "Power PC603, a microprocessor for portable computers," *IEEE Design and Test of Computers*, Winter 1994, pp. 14-23
- [Gary96] S. Gary, "Low-power microprocessor design," in *Low Power Design Methodologies*, Mass. USA: Kluwer Academic Publishers, 1996, Chapter 9, pp. 255-288
- [Gebot93] C.H. Gebotys, "Throughput optimized architectural synthesis," *IEEE Trans. On VLSI Systems*, vol. 1, no. 3, Sept 1993, pp. 254-261

- [Gela93] P.R. Gelabert and T.P. Barnwell III, "Optimal automatic periodic multiprocessor scheduler for fully specified flow graphs," *IEEE Trans. on Signal Processing*, vol. 41, no. 2, Feb. 1993, pp. 858-888
- [Genashor] Genashor Inc., "Xpower," Available *HTTP* <http://pluto.njcc.com/~genashor/xpower.html>
- [Glover98] F.W. Glover, *Tabu Search*, Kluwer Academic Publishers, 1988
- [Goldberg89] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, Mass: Addison-Wesley Publishing Co. Inc., 1989
- [Goldberg91] D.E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations Of Genetic Algorithms*, Gregory J.E. Rawlins (Ed)., Morgan Kaufman, CA USA, 1991, pp. 69-93
- [Good94] L. Goodby, A. Orailoglu and P.M. Chau, "A high-level synthesis methodology for low-power VLSI design," *Proc. IEEE Symposium on Low Power Electronics*, 1994, pp. 48-49
- [Gowan98] M.K. Gowan, L.L. Biro, D.B. Jackson, "Power considerations in the design of the Alpha 21264 microprocessor," *Proc. 35th Design Automation Conference, DAC 98, San Francisco, CA, 1998*, pp. 726-731
- [Gref84] J. J. Grefenstette, "GENESIS: A system for using genetic search procedures," *Proc. 1984 Conf. on Intelligent Systems and Machines*, 1984, pp. 161-165
- [Gref86] J.J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans. on SMC*, vol. 16, no. 1, 1986, pp. 122-128
- [Gref87] J.J. Grefenstette, "Incorporating problem specific knowledge into genetic algorithms," in *Genetic Algorithms and Simulated Annealing*, Morgan Kaufman, LA, USA, 1987, pp.43-60
- [Grewal97] G.W. Grewal, T.C. Wilson, "An enhanced genetic solution for scheduling, module allocation, and binding in VLSI design," *Proc. IEEE 10th Int. Conf. on VLSI design*, Jan 1997, pp. 51-56
- [Guerra98] L. Guerra, M. Potkonjak, J.M. Rabaey, "A methodology for guided behavioral-level optimization," *Proc. 35th Design Automation Conference, DAC 98, San Francisco, CA, 1998*, pp. 309-314

- [Guerts91] W. Guerts, F. Catthoor, H. De Man, "Cathedral III : architecture-driven high-level synthesis for high throughput DSP applications," *Proc. 28th IEEE Design Automation Conference, DAC 91*, USA, June 1991, pp. 597-602
- [Gwee96] B. H. Gwee and M. H. Lim, "Polyominoes tiling by a genetic algorithm," *Computational Optimization and Applications 6*, Mass. USA: Kluwer Academic Publishers, 1996, pp. 273-291
- [Hanc95] P.J.B. Hancock, "Selection methods for evolutionary algorithms," *in Practical Handbook of GAs : New Frontiers Vol. II*, Lance Chambers (Ed)., CRC Press Inc, 1995, pp.67-92
- [Holland92] J. H. Holland, *Adaptation in Natural and Artificial Systems*, 2nd. ed., Cambridge, Mass: MIT Press, 1992
- [Hong98] I. Hong, D. Kirovski, G. Qu, M. Potkonjak and M.B. Srivastava, "Power optimization of variable voltage core-based systems," *Proc. 35th Design Automation Conference, DAC 98*, San Francisco, CA, 1998, pp. 176-181
- [Horo94] M. Horowitz, T. Indermaur and R. Gonzalez, "Low-power digital design," *Proc. IEEE Symposium on Low Power Electronics*, 1994, pp. 8-11
- [Huang94] S-H. Huang, J.M. Rabaey, "Maximizing the throughput of high performance DSP applications using behavioral transformations," *Proc. EDAC-ETC-EUROASIC'94*, Paris, France, March 1994, pp. 25-30
- [Huang96] S-H. Huang, J. M. Rabaey, "An integrated framework for optimizing transformations," *Proc. IEEE VLSI Signal Processing*, San Francisco, CA, Oct. 1996
- [Hwang91] C-T. Hwang, J-H. Lee, Y-C Hsu, "A formal approach to the scheduling problem in high-level synthesis," *IEEE Trans. On Computer Aided Design*, vol. 10, no. 4, April 1991, pp. 464-475
- [Ikeda95] T. Ikeda, "Think-pad low-power evolution," *Proc. IEEE Symposium on Low Power Electronics, Digest of Technical Papers*, San Jose, CA, Oct. 1995, pp. 6-7
- [Iman96] S. Iman and M. Pedram, "Pose: power optimization and synthesis environment," *Proc. 33rd IEEE Design Automation Conference, DAC 33*, 1996, Las Vegas, USA
- [Intel98] Intel Inc., "Mobile PC power," Available *HTTP* <http://www.intel.com/mobile/techforum/power.htm>

- [Intel98b] Intel Inc., "Mobile Pentium processor with MMX technology," *Datasheet No. 243292-004*
- [Intel98c] Intel Inc., "Pentium II processor at 233 MHz, 266MHz, 300 MHz and 333MHz," *Datasheet No. 243335-003*
- [Ketzer94] K. Ketzer, "The impact of CAD on the design of low power digital circuits," *Proc. IEEE Symposium on Low Power Electronics*, 1994, pp. 42-45
- [Khalifa97] Y. M. A. Khalifa, *Evolutionary Methods for the Design of Electronic Circuits and Systems*, Ph.D. Thesis, School of Engineering, Cardiff University, 1997
- [Kim96] H. Kim and S.Y. Hwang, "Heuristic algorithm for low power design of combinational circuits," *IEE Electronics Letters*, 6th June 1996, vol. 32, no. 12, pp. 1066-1067
- [Kim97] D. Kim and K. Choi, "Power-conscious high level synthesis using loop folding," *Proc IEEE Design Automation Conference, DAC 97*, Anaheim Calif. USA, 1997, pp. 441-445
- [Kim98] T. Kim, W. Jao, S. Tjiang, "Arithmetic operation using carry-save-adders," *Proc. 35th Design Automation Conference, DAC 98, San Francisco, CA, 1998*, pp. 433-438
- [Kirk83] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, "Optimization by simulated annealing," *Science*, 220(4598), May 1983, pp. 671-680
- [Koel] A.M. Koelmans, F.P. Burns and D.J. Kinniment, "Use of a theorem prover for transformational synthesis," *Technical Report 426*, Department of Computing Science, University of Newcastle upon Tyne, 1993
- [Kumar95] N. Kumar, S. Katkoori, L. Rader, R. Vemuri, "Profile-driven behavioral synthesis for low-power VLSI systems," *IEEE Design and Test of Computers*, Fall 1995, pp. 70-84
- [Kunii95] S. Kunii, "Means of realizing long battery life in portable PCs," *Proc. IEEE Symposium on Low Power Electronics, Digest of Technical Papers*, San Jose, CA, Oct. 1995, pp. 20-23
- [Laksh98] G. Lakshminarayana and N.K. Jha, "FACT: a framework for the application of throughput and power optimizing transformations to control-flow intensive behavioral descriptions," *Proc. 35th Design Automation Conference, DAC 98, San Francisco, CA, 1998*, pp. 102-107

- [Laksh98a] G. Lakshminarayana and N.K. Jha, "Synthesis of power-optimized and area-optimized circuits from hierarchical behavioral descriptions," *Proc. 35th Design Automation Conference, DAC 98*, San Francisco, CA, 1998, pp. 439-444
- [Landman93] P.E. Landman, J.M. Rabaey, "Power estimation for high level synthesis," *Proc. EDAC-EUROASIC '93*, Paris, France, Feb 1993, pp. 361-366
- [Landman94] P.E. Landman and J.M. Rabaey, "Black-box capacitance models for architectural power analysis," *IEEE Int. Workshop on Low-Power Design*, Napa, CA, April 1994, pp.165-170
- [Landman94a] P.E. Landman, *Low-power architectural design methodologies, Ph.D. Thesis*, Univ. Berkeley, CA, USA, 1994
- [Landman96] P.E. Landman, R. Mehra, J.M. Rabaey, "An integrated CAD environment for low-power design," *IEEE Design and Test of Computers*, vol. 13, no. 2, Summer, 1996
- [Landman96a] P.E. Landman and J.M. Rabaey, "Activity sensitive architectural power analysis," *IEEE Trans. on Computer Aided Design*, vol. 15, no. 6, June 1996, pp. 571-587
- [Larson98] E. Larson, "Low-power radio frequency circuit architectures for portable wireless communications," in *Proc. IEEE International Symposium on Circuits and Systems , ISCAS '98*, Monterey, CA, 1998
- [Lee98] W. Lee, P.E. Landman, B. Barton and G. Frantz, "A 1V programmable DSP for Wireless Applications," in *Proc. IEEE International Symposium on Circuits and Systems , ISCAS '98*, Monterey, CA, 1998
- [Lem94] Z.J. Lemnios, K.J. Gabriel, "Low-power electronics," *IEEE Design and Test of Computers*, Winter 1994, pp. 8-13
- [Liao96] H. Liao, W.W-M. Dai, "A new CMOS driver model for transient analysis and power dissipation analysis," in *Low Power VLSI Design and Technology*, G.K. Yeap, F.N. Najm (Eds.), London, UK: World Scientific, 1996, pp. 47-63
- [Lidsky96] D. Lidsky, J.M. Rabaey, "Early power exploration - a world wide web application," *Proc. ACM/IEEE 33rd Design Automation Conference, DAC 96*, Las Vegas, USA, 1996,
- [Lienig97] J. Lienig, "A parallel genetic algorithm for performance-driven VLSI routing," *IEEE Trans. Evolutionary Computation*, vol. 1, no. 1, April 1997, pp. 29-39

- [Lies83] C. E. Lieserson, F. M. Rose, J. B. Saxe, "Optimizing synchronous circuitry by retiming," *Proc. 3rd Caltech Conf on VLSI*, Pasadena CA, March 1983, pp. 87-116
- [Liu93] D. Liu and C. Svensson, "Trading speed for low power by choice of supply and threshold voltages," *IEEE Journal of Solid State Circuits*, vol. 28, no. 1, Jan 1993, pp. 10-17
- [Liu94] D. Liu and C. Svensson, "Power consumption estimation in CMOS VLSI chips," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 6, June 1994, pp. 663-670
- [Lo98] C-K. Lo and P.C.H. Chan, "Design of low power differential logic using adiabatic switching technique," in *Proc. IEEE International Symposium on Circuits and Systems , ISCAS '98*, Monterey, CA, 1998
- [Lock93] B. Lockyear, C. Ebeling, "The practical application of retiming to the design of high-performance systems," *Proc. Int. Conf. CAD*, June 1993, pp. 288-295
- [Luck93] L.E. Lucke, K.K. Parhi, "Data flow transformations for critical path time reduction in high level DSP synthesis," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 12, no. 7, July 1993, pp.1064-1066
- [Lyon93] R.F. Lyon, "Cost, power and parallelism in speech signal processing," *IEEE Custom Integrated Circuits Conference*, 1993, pp. 15.1.1-15.1.9
- [Ma90] G-K. Ma, F.J. Taylor, "Multiplier policies for digital signal processing," *IEEE ASSP Magazine*, Jan, 1990, pp. 6-19
- [Macii97] E. Macii, M. Pedram and F. Somenzi, "High-level power modeling, estimation and optimization," *Proc IEEE Design Automation Conference, DAC 97*, Anaheim Calif. USA, 1997, pp. 190-195
- [Madis96] V.K. Madisetti, "Rapid digital system prototyping: current practice, future challenges," *IEEE Design and Test of Computers*, Fall 1996, pp. 12-22
- [Madis96a] V.K. Madisetti, M.A. Richards, "Advances in rapid prototyping of digital systems," *IEEE Design and Test of Computers*, Fall 1996, pp. 9-11
- [Mandal96] C A Mandal, P.P. Chakrabarti, S. Ghose, "Allocation and binding in data path synthesis using a genetic algorithm approach," *Proc. IEEE 9th Int. Conf. On VLSI Design*, Jan 1996, pp. 122-125

- [Martin95] R.S. Martin, J.P. Knight, "Power Profiler : optimizing ASICs power consumption at the behavioral level," *Proc. IEEE Design Automation Conference*, San Francisco, CA, 1995, pp. 42-47
- [Martin96] R.S. Martin and J.P. Knight, "Optimizing power in ASIC behavioral synthesis," *IEEE Design and Test of Computers*, Summer 1996, pp. 58-70
- [McFar83] M.C. McFarland and A.C. Parker, "An abstract model of behavior for hardware systems," *IEEE Trans. on Computers*, vol. C-32, no. 7, July 1983, pp. 621-637
- [McFar90] M.C. McFarland, A.C. Parker, R. Camposano, "The high-level synthesis of digital systems," *Proc. of the IEEE*, vol. 78, no. 2, Feb 1990, pp. 301-318
- [Mcgrath95] S. McGrath and E. Scully, "Low power ASIC design for wireless communications," *IEE Colloquium on 'Low Power Analogue and Digital VLSI: Techniques and Applications'*, 2 June 1995, London, UK, pp. 3/1-3/6
- [Mehra94] R. Mehra, J.M. Rabaey, "Behavioral level power estimation and exploration," *Proc. 1994 Int. Workshop On Low Power Design*, Calif., USA, 1994
- [Mehra96] R. Mehra, D.B. Lidsky, A. Abnous, P.E. Landman and J.M. Rabaey, "Algorithm and architectural level methodologies for low power," in *Low Power Design Methodologies*, Mass. USA: Kluwer Academic Publishers, 1996, chapter 11, pp. 335-362
- [Mehra96a] R. Mehra, L. Guerra and J.M. Rabaey, "Low-power architectural synthesis and the impact of exploiting locality," *Journal of VLSI Signal Processing*, Kluwer Academic Publishers; 1996, Available *HTTP*: <http://infopad.eecs.berkeley.edu/~renu/papers/vlisp96.ps>
- [Meng98] T.H. Meng, A.C. Hung, E.K. Tsern and B.M. Gordon, "Low-Power signal processing system design for wireless applications," *IEEE Personal Communications Magazine*, June 1998, pp. 21-31
- [Mess98] D.G. Messerschmitt, "Breaking the recursive bottleneck," in *Performance Limits in Communication Theory and Practice*, J.K. Skwirzynski (Ed.), Kluwer Academic Publishers, 1998, pp. 3-19
- [Micheli90] G. De Micheli, "High-level synthesis of digital circuits," *IEEE Design and Test of Computers*, Oct. 1990, pp. 6-7
- [Micheli94] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, New York, NY: McGraw-Hill, Inc., 1994

- [Mitch96] M. Mitchell, *An Introduction to Genetic Algorithms*, Cambridge, Mass: MIT Press, 1996
- [Mont92] J. Monteiro, S. Devadas, A. Ghosh, "Retiming sequential circuits for low power," *Proc. Int. Conf. CAD*, June 1993
- [Najm94] F.N. Najm, "A survey of power estimation techniques in VLSI circuits," *IEEE Transactions on VLSI Systems*, vol. 2, no. 4, Dec. 1994, pp. 446-455
- [Najm95] F.N. Najm, "Feedback, correlation, and delay concerns in the power estimation of VLSI circuits," *Proc. IEEE/ACM 32nd Design Automation Conference*, San Francisco, CA, June 1995, pp. 612-617
- [Neil] J.P. Neil and P.B. Denyer, "Simulated annealing based synthesis of fast discrete cosine transform blocks", in *Algorithmic and Knowledge-Based CAD for VLSI*, G.E. Taylor, G. Russell (Eds.), London, UK: IEE Circuits and Systems Series, No 4, 1992
- [Odare94] M.J. O'Dare, T. Arslan, "Generating test patterns for VLSI circuits using a genetic algorithm," *IEE Electronics Letters*, vol. 30, no. 10, May 1994, pp. 778-779
- [Parhi89] K.K. Parhi, "Algorithm transformation techniques for concurrent processors," *Proc. of the IEEE*, vol.77, no.12, Dec. 1989, pp. 1879-1895
- [Parhi91] K.K. Parhi, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. On Computers*, vol. 40, no. 2, Feb. 1991, pp. 178-195
- [Parhi92] K.K. Parhi, "Impact of architecture choices on DSP circuits," *IEEE Region 10 Conf., Tenccon 92*, Nov 1992, pp. 784-788
- [Parhi95] K.K. Parhi, "High-level algorithm and architecture transformations for DSP synthesis," *Journal of VLSI Signal Processing*, vol. 9, 1995, pp. 121-143
- [Park98] N. Park and A.C. Walker, "Sehwa: a software package for synthesis of pipelines from behavioral descriptions," *IEEE Trans on Computer-Aided Design*, vol. 7, no. 3, March 1998, pp. 356-370
- [Paulin89] P.G. Paulin, J.P. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC's," *IEEE Trans. On Computer Aided Design*, vol. 8, no. 6, June 1989, pp. 661-679

- [Pedram95] M. Pedram, "CAD for low power: status and promising directions," *Proc. Int. Symposium on VLSI Technology*, 1995, Available: [HTTP http://atrk.usc.edu/~massoud/signdownload.cgi?vlsi-95-power-survey.ps](http://atrk.usc.edu/~massoud/signdownload.cgi?vlsi-95-power-survey.ps)
- [Potko89] M. Potkonjak, J.M. Rabaey, "A scheduling and resource allocation algorithm for hierarchical signal flow graphs," *Proc. IEEE 26th Design Automation Conference*, Las Vegas, USA, June 7-12th, 1989, pp. 7-12
- [Potko91] M. Potkonjak, J.M. Rabaey, "Retiming for scheduling," *VLSI Signal Processing IV*, H.S. Moscovitz, K. Yao and R. Jain (Eds.), IEEE Press, New Jersey, 1991, pp. 23-32
- [Potko92a] M. Potkonjak, J.M. Rabaey, "Pipelining: just another transformation," *Proc. 1992 Application Specific Array Processors Conference*, USA, pp. 163-175
- [Potko94] M. Potkonjak, J.M. Rabaey, "Optimizing resource utilization using transformations," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 13, no. 3, March 1994, pp. 277-292
- [Potko95] M. Potkonjak, J.M. Rabaey, "Power minimization in DSP application specific systems using algorithm selection," *Proc. IEEE Int. Conf Acoustics, Speech and Signal Processing*, 1995, vol. 4, pp. 2639-2642
- [Potko97] M. Potkonjak, *Personal Communication*, E-mail, Wed 26 Nov. 1997
- [Powell90] S.R. Powell, P.M. Chau, "Estimating power dissipation of VLSI signal processing chips: the PFA technique," *VLSI Signal Processing IV*, 1990, pp. 251-259
- [Powell91] S.R. Powell and P.M. Chau, "A model for estimating power dissipation in a class of DSP VLSI chips," *IEEE Trans. on Circuits and Systems*, vol. 38, no 6, June 1991, pp. 646-650
- [Puck94] D.A. Pucknell, K. Eshragian, *Basic VLSI Design*, Silicon Systems Engineering Series, Prentice Hall, 1994
- [Rabaey] J. M. Rabaey, "The Spice Page," Available [HTTP http://infopad.eecs.berkeley.edu/~icdesign/SPICE/](http://infopad.eecs.berkeley.edu/~icdesign/SPICE/)
- [Rabaey90] J.M. Rabaey and M. Potkonjak, "Resource driven synthesis in the HYPER system," *Proc. Symposium on Circuits and Systems*, vol. 4, IEEE Press, New York, 1990, pp. 2592-2595
- [Rabaey91] J.M. Rabaey, M. Potkonjak, "Complexity estimation for real time circuits," *Proc. ESSCIRC*, Milan, Italy, Sep. 1991, pp. 201-291

- [Rabaey91a] J.M. Rabaey, C. Chu, P. Hoang, M. Potkonjak, "Fast prototyping of datapath-intensive architectures," *IEEE Design and Test Of Computers*, June 1991, pp. 40-51
- [Rabaey94] J.M. Rabaey, M. Potkonjak, "Estimating implementation bounds for real time DSP application specific circuits," *IEEE. Trans. on CAD of Integrated Circuits and Systems*, vol. 13, no. 6, June 1994, pp. 669-683
- [Rabaey95] J.M. Rabaey, L. Geurra, R. Mehra, "Design guidance in the power dimension," *Proc. IEEE Int. Conference On Acoustics, Speech and Signal Processing*, 1995, vol. 5, pp. 2387-2840
- [Rabaey96] J.M. Rabaey and M. Pedram (Eds.), *Low Power Design Methodologies*, Mass., USA: Kluwer Academic Publishers, 1996
- [Rabaey97] J.M. Rabaey, "Low-power design tools - where is the impact?," *Proc IEEE Design Automation Conference, DAC 97*, Anaheim Calif. USA, 1997, pp. 82
- [Rag94] A. Raghunathan, N.K. Jha, "Behavioral synthesis for low power," *Proc. ICCD '94*, 1994, pp. 318-322
- [Rag95] A. Raghunathan, N.K. Jha, "An iterative improvement algorithm for low power data path synthesis," *Proc. IEEE/ACM ICCAD '95*, Digest of Technical papers, 1995, pp. 597-602
- [Rag95a] A. Raghunathan, N.K. Jha, "An ILP formulation for low power based on minimizing switched capacitance during data path allocation," *Proc. IEEE Int. Symposium on Circuits and Systems*, vol. 2, 1995, pp. 1069-1073
- [Ranjit90] R. K. Ranjit, *A Primer on the Taguchi Method*, New York, US: Van Nostrand Reinhold. 1990.
- [Reese94] B. Reese, "Using hyper to teach datapath design techniques in an ASIC design course," *Proc. of the Annual IEEE International ASIC Conference*, NY, USA. Sep 1994, pp. 200-203
- [Reeves95] Colin R. Reeves (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*, London, UK: McGraw-Hill Int., 1995
- [Rhis97] G. Rhisiart, *The Development of Graphics Routines for an Intelligent DSP Synthesis System in C++*, B.Eng. Dissertation, School of Engineering, Cardiff University, 1997
- [Rich93] E. Rich and K. Knight, "Heuristic search techniques," in *Artificial Intelligence*, McGraw-Hill, 2nd edition, USA, 1993, chapter 3, pp. 63-73

- [Roy95] K. Roy, "A design and test roundtable: low power design," *IEEE Design and Test of Computers*, Winter 1995, pp. 84-90
- [Sato94] T. Sato, M. Nagamatsu and H. Tago, "Power and performance simulator: ESP and its application for 100MIPS/W class RISC design," *Proc. IEEE Symposium on Low Power Electronics*, 1994, pp. 46-47
- [Schaff93] J.D. Schaffer, L.J. Eshelman, "Designing multiplierless digital filters using genetic algorithms," *Proc. Fifth Int. Conf. on Genetic Algorithms*, Illinois, USA, July 17-21 1993, pp. 439-444
- [Schnecke95] V. Schnecke, "Genetic design of VLSI layouts," in *Genetic algorithms In Engineering Systems*, London, UK: IEE Press, 1995, pp. 229-253
- [Schnecke95a] V. Schnecke, O. Vornberger, "Genetic design of VLSI layouts," *Proc. 1st IEE/IEEE Conf. On GAs In Eng. Systems: Innovations and Apps, GALEZIA'95*, Sep 1995, UK, pp.430-435
- [Schwef95] H-P. Schwefel, *Evolution and Optimum Seeking*, New York, NY: Wiley-Interscience, 1995
- [Sente] Sente, "WattWatcher Product Information," Available *HTTP* <http://www.powereda.com/wwinfo.htm>
- [Sheng92] S. Sheng, A. Chandrakasan, R.W. Broderson, "A portable multimedia terminal," *IEEE Communications Magazine*, Dec 1992, pp. 64-75
- [Shiue98] W-T. Shiue and C. Chakrabarti, "Low power scheduling with resources operating at multiple voltages," in *Proc. IEEE International Symposium on Circuits and Systems , ISCAS '98*, Monterey, CA, 1998
- [Sinclair98] M.C. Sinclair, "Operator-probability Adaptation in a Genetic-algorithm/Heuristic Hybrid for Optical Network Wavelength Allocation," *Proc. IEEE Intl. Conf. on Evolutionary Computation (CEC'98)*, Anchorage, Alaska, USA, May 1998, pp. 840-845
- [Singh95] D. Singh, J.M. Rabaey, M. Pedram, F. Catthoor, S. Rajgopal, N. Sehgal, T.J. Mozdzen, "Power conscious CAD tools and methodologies: a perspective," *Proc. of the IEEE*, vol. 83, no. 4, April 1995, pp. 570-594
- [Snyder94] J.H. Snyder, J.B. McKie and B.N. Locanthi, "Low-power software for low-power people," *Proc. IEEE Symposium on Low Power Electronics*, 1994, pp. 32-35

- [Sriv95] M.B. Srivastava, "Optimum and heuristic transformation techniques for simultaneous optimization of latency and throughput," *IEEE Trans. on VLSI Systems*, vol. 3, no. 1, March 1995, pp. 2-19
- [Sriv96] M. Srivastava, M. Potkonjak, "Power optimization in programmable processors and ASIC implementations of Linear Systems: Transformation based approach," *Proc. 33rd IEEE Design Automation Conference, DAC 33*, 1996, Las Vegas, USA
- [Strat94] A.J. Stratakos, R.W. Broderson and S.R. Sanders, "High-efficiency low-voltage DC-DC conversion for portable applications," *Proc. IEEE Int. Workshop on Low-Power Design*, CA, USA, April 1994
- [Strat94a] A.J. Stratakos, S.R. Sanders and R.W. Broderson, "A low-voltage CMOS DC-DC converter for a portable battery-operated system," *IEEE Power Electronics Specialists Conference*, June 1994
- [Su94] C-L. Su, C-Y. Tsui, A.M. Despaigne, "Low power architecture design and compilation techniques for high-performance processors," *IEEE Compcon*, Feb 1994, pp. 489-498
- [Synop] Synopsys, "AMPS Datasheet," Available *HTTP* [http://www.synopsys.com/products/etg/amps\\_ds.html](http://www.synopsys.com/products/etg/amps_ds.html)
- [Synop98] Synopsys Inc., USA, "Designing for low power," Available *HTTP* [http://www.synopsys.com/products/power/power\\_br.html](http://www.synopsys.com/products/power/power_br.html)
- [Synop98b] Synopsys Inc., USA, "Powergate," Available *HTTP* [http://www.synopsys.com/products/etg/powergate\\_ds.html](http://www.synopsys.com/products/etg/powergate_ds.html)
- [Synop98c] Synopsys Inc., USA, "PowerMill Datasheet," Available *HTTP* [http://www.synopsys.com/products/etg/powermill\\_ds.html](http://www.synopsys.com/products/etg/powermill_ds.html)
- [Synop98d] Synopsys Inc., USA, "Power Family Datasheet," Available *HTTP* [http://www.snyopsys.com/products/power/power\\_ds.html](http://www.snyopsys.com/products/power/power_ds.html)
- [Szek98] V. Szekeley, M. Renz, B. Courtois, "Tracing the thermal behavior of ICs," *IEEE Design & Test of Computers*, April-June 1998, pp. 15-21
- [Teuk95] S.A. Teukolsky, W. T. Vetterling, W. H. Press and B. P. Flannery (Eds.), "Simulated annealing methods," in *Numerical Recipes in C*, 2nd Ed., Cambridge, UK: Cambridge Univ. Press, 1995, pp. 444-445
- [Teuk95a] S.A. Teukolsky, W. T. Vetterling, W. H. Press and B. P. Flannery (Eds.), "Sorting," in *Numerical Recipes in C*, 2nd Ed., Cambridge, UK: Cambridge Univ. Press, 1995, pp. 329-346

- [Teuk95b] S.A. Teukolsky, W. T. Vetterling, W. H. Press and B. P. Flannery (Eds.), *Numerical Recipes in C*, 2nd Ed., Cambridge, UK: Cambridge Univ. Press, 1995
- [Tiwari98] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, F. Baez, "Reducing power in high-performance microprocessors," *Proc. 35th Design Automation Conference, DAC 98*, San Francisco, CA, 1998, pp. 732-737
- [Trick87] H. Trickey, "Flamel: a high-level hardware compiler," *IEEE Trans. on CAD*, vol. CAD-6, no. 2, March 1987, pp. 259-269
- [Tuck97] B. Tuck, "Editorial: prepare for deep submicron with low-power strategy," *Computer Design*, August 1997, Available *HTTP* <http://www.computer-design.com/Editorial/1997/08/asic/897assub.html>
- [Turton94] B.C.H Turton, "Optimization of genetic algorithms using the Taguchi method," *Journal of Systems Engineering*, vol. 4, 1994, pp. 121-130
- [Van93] P. Vanoostende, P. Six, J. Vandewalle, H.J. De Man, "Estimation of typical power of synchronous CMOS circuits using a hierarchy of simulators," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 1, Jan 1993, pp. 26-39
- [Vecchi83] M.P. Vecchi, S. Kirkpatrick, "Global wiring by simulated annealing," *IEEE Trans. on CAD of Integrated Circuits*, vol. CAD-2, no. 4, Oct 1983, pp. 215-222
- [Veritools] Veritools, "Power\_tool a PLI based power calculation too for Verilog," Available *HTTP*: [http://www.veritools-web.com/power\\_t.htm](http://www.veritools-web.com/power_t.htm)
- [Walker89] R.A. Walker, D.E. Thomas, "Behavioral transformation for algorithmic level IC design," *IEEE Trans on Computer Aided Design*, vol. 8, no. 10, Oct 1989, pp. 1115-1127
- [Walker94] R.A. Walker, "The status of high-level synthesis," *IEEE Design and Test of Computers*, Winter 1994, pp. 42-43
- [Walker95] R.A. Walker, S. Chaudhuri, "Introduction to the scheduling problem," *IEEE Design & Test Of Computers*, Summer 1995, pp. 60-69
- [Wang95] C-Y. Wang, K.K. Parhi, "High-level DSP synthesis using concurrent transformations, scheduling and allocation," *IEEE Trans. on CAD of ICs*, vol. 14, no. 3, March 1995, pp. 274-295

- [Whit89] D. Whitley, "The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best," *Proc. 3rd Int. Conf On GAs*, CA, June 1989, pp. 116-121
- [Wilk92] B. Wilkinson, R. Makki, *Digital System Design*, London, UK: Prentice Hall International, 1992
- [Yeap96] G.K. Yeap, F.N. Najm (Eds.), *Low Power VLSI Design and Technology*, London, UK: World Scientific, 1996
- [Zalzala97] A. M. S. Zalzala and P. J. Fleming (Eds.), *Genetic Algorithms in Engineering Systems*, London, UK: IEE, 1997

# Appendix A – Benchmark Designs

## CONTENTS

A1. 8TAPFIR NETLIST .....	2
<i>A1.1 8TAPFIR DFG</i> .....	3
A2. AVEN8DI NETLIST .....	4
<i>A2.1 AVEN8DI DFG</i> .....	5
A3. AVEN8PA NETLIST .....	6
<i>A3.1 AVEN8PA DFG</i> .....	7
A4. DCST NETLIST .....	8
<i>A4.1 DCST DFG</i> .....	9
A5. BIQUAD3 NETLIST .....	10
<i>A5.1 BIQUAD3 DFG</i> .....	11
A6. GMLAT4 NETLIST .....	12
<i>A6.1 GMLAT4 DFG</i> .....	13
A7. ELLIP5 NETLIST .....	14
<i>A7.1 ELLIP5 DFG</i> .....	15
A8. LMS5 NETLIST .....	16
<i>A8.1 LMS5 DFG</i> .....	17
A9. VOLTERRA NETLIST .....	18
<i>A9.1 VOLTERRA DFG</i> .....	19
A10. ORTH2LAT NETLIST .....	20
<i>A10.1 ORTH2LAT DFG</i> .....	21

## A1. 8TAPFIR Netlist

\$8tapFIRfilter

\$primary inputs

e1

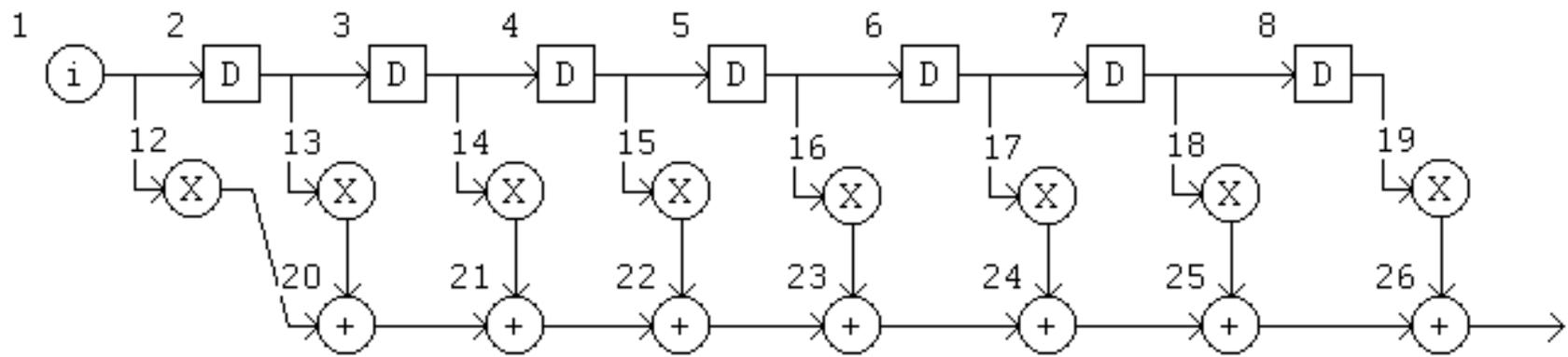
\$primary outputs

e26

\$circuit description

\$	output	type	inputs
	e2	del	e1
	e3	del	e2
	e4	del	e3
	e5	del	e4
	e6	del	e5
	e7	del	e6
	e8	del	e7
	e12	mul	e1
	e13	mul	e2
	e14	mul	e3
	e15	mul	e4
	e16	mul	e5
	e17	mul	e6
	e18	mul	e7
	e19	mul	e8
	e20	add	e12 e13
	e21	add	e20 e14
	e22	add	e21 e15
	e23	add	e22 e16
	e24	add	e23 e17
	e25	add	e24 e18
	e26	add	e25 e19

A1.1 8TAPFIR DFG



## A2. AVEN8DI Netlist

\$ 8th Direct From Avenhaus

\$Primary Inputs

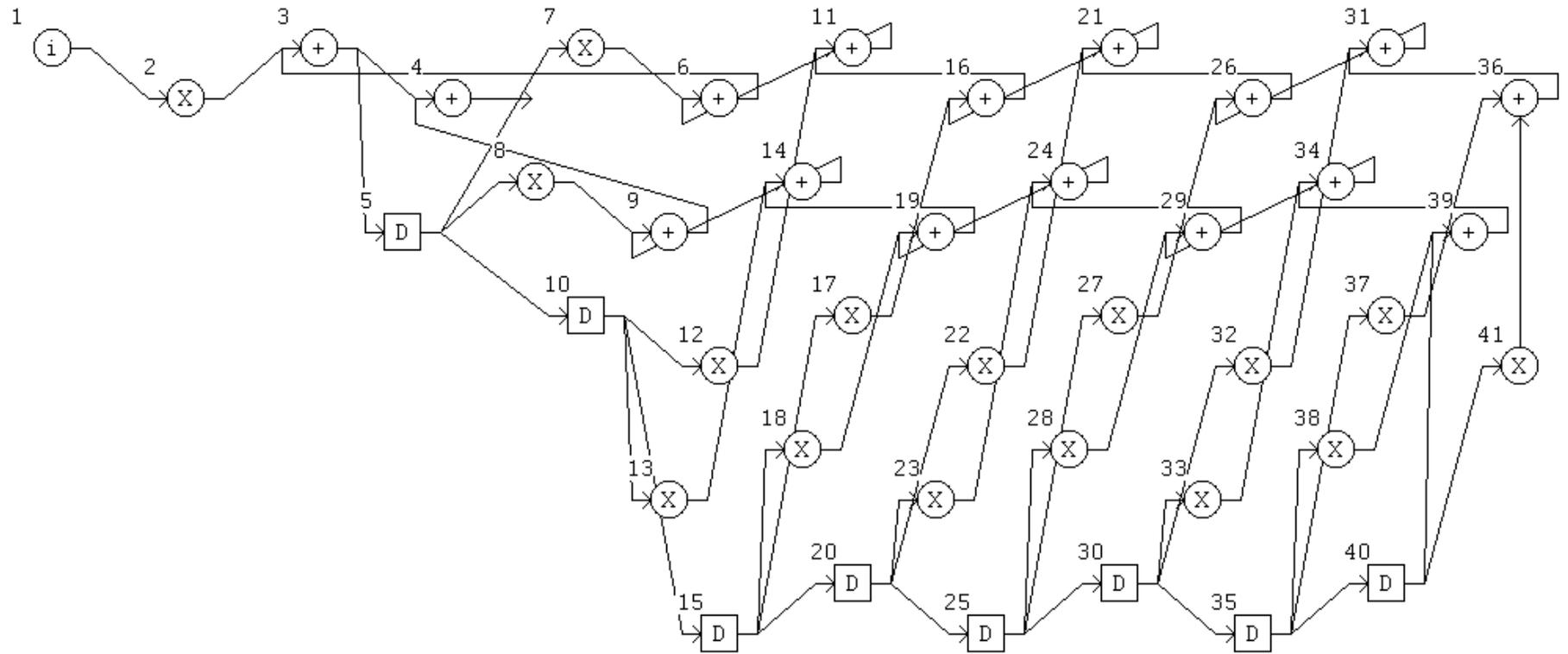
e1

\$ Primary Outputs

e4

\$	output	type	inputs
	e2	mul	e1
	e3	add	e2 e6
	e4	add	e3 e9
	e5	del	e3
	e6	add	e7 e11
	e7	mul	e5
	e8	mul	e5
	e9	add	e8 e14
	e10	del	e5
	e11	add	e12 e16
	e12	mul	e10
	e13	mul	e10
	e14	add	e13 e19
	e15	del	e10
	e16	add	e17 e21
	e17	mul	e15
	e18	mul	e15
	e19	add	e18 e24
	e20	del	e15
	e21	add	e22 e26
	e22	mul	e20
	e23	mul	e20
	e24	add	e23 e29
	e25	del	e20
	e26	add	e27 e31
	e27	mul	e25
	e28	mul	e25
	e29	add	e28 e34
	e30	del	e25
	e31	add	e32 e36
	e32	mul	e30
	e33	mul	e30
	e34	add	e33 e39
	e35	del	e30
	e36	add	e37 e41
	e37	mul	e35
	e38	mul	e35
	e39	add	e40 e38
	e40	del	e35
	e41	mul	e40

A2.1 AVEN8DI DFG



### A3. AVEN8PA Netlist

\$8th order Parallel Avenhaus

\$Primary Inputs

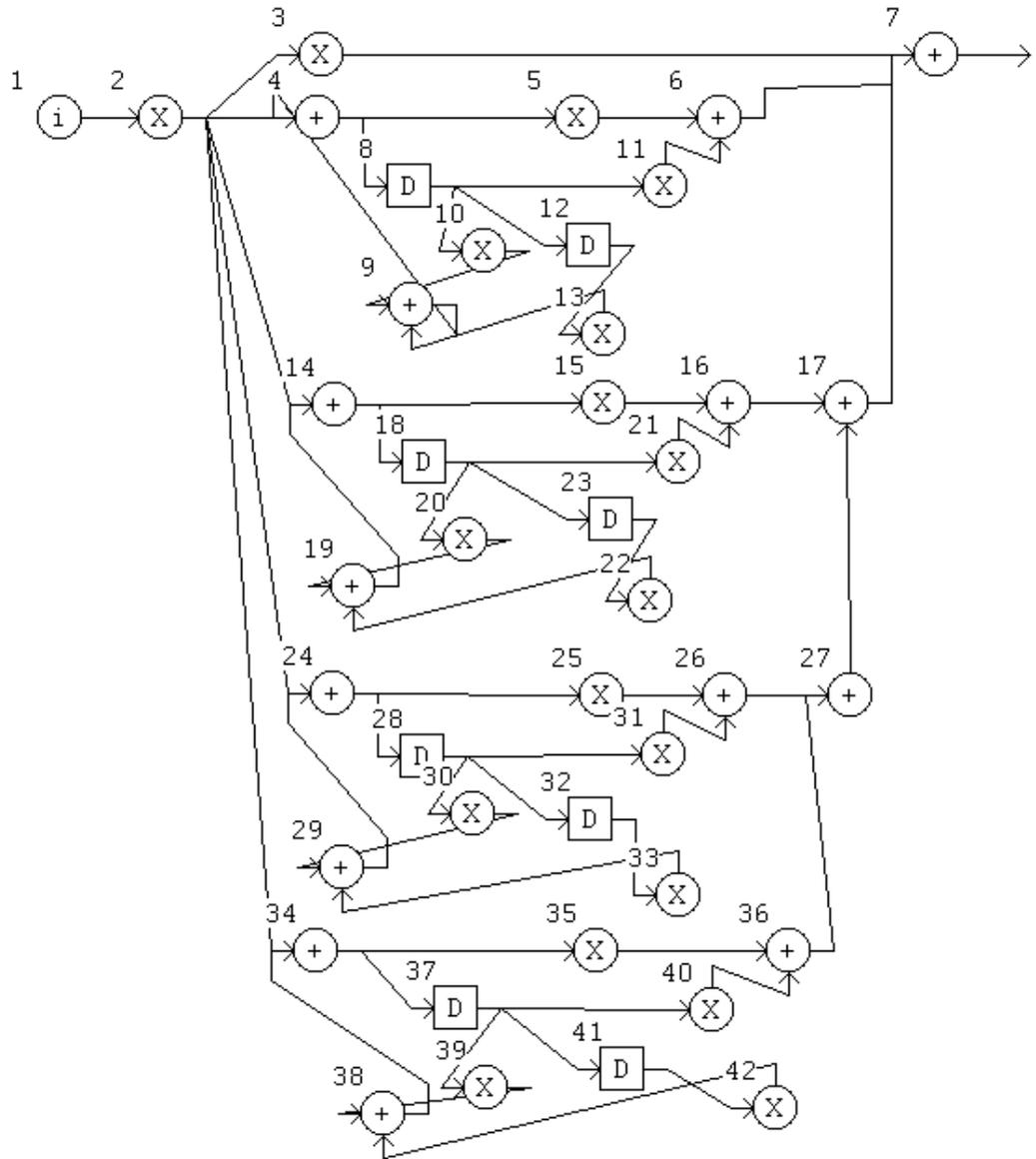
e1

\$Primary Outputs

e7

\$	output	type	inputs
	e2	mul	e1
	e3	mul	e2
	e4	add	e2 e9
	e5	mul	e4
	e6	add	e5 e11
	e7	add	e6 e17 e3
	e8	del	e4
	e9	add	e10 e13
	e10	mul	e8
	e11	mul	e8
	e12	del	e8
	e13	mul	e12
	e14	add	e2 e19
	e15	mul	e14
	e16	add	e15 e21
	e17	add	e16 e27
	e18	del	e14
	e19	add	e20 e22
	e20	mul	e18
	e21	mul	e18
	e22	mul	e23
	e23	del	e18
	e24	add	e2 e29
	e25	mul	e24
	e26	add	e25 e31
	e27	add	e26 e36
	e28	del	e24
	e29	add	e30 e33
	e30	mul	e28
	e31	mul	e28
	e32	del	e28
	e33	mul	e32
	e34	add	e2 e38
	e35	mul	e34
	e36	add	e35 e40
	e37	del	e34
	e38	add	e39 e42
	e39	mul	e37
	e40	mul	e37
	e41	del	e37
	e42	mul	e41

A3.1 AVEN8PA DFG



#### A4. DCST Netlist

\$ DCT/DST algorithm

\$ primary inputs

e1

e2

\$ primary outputs

e14

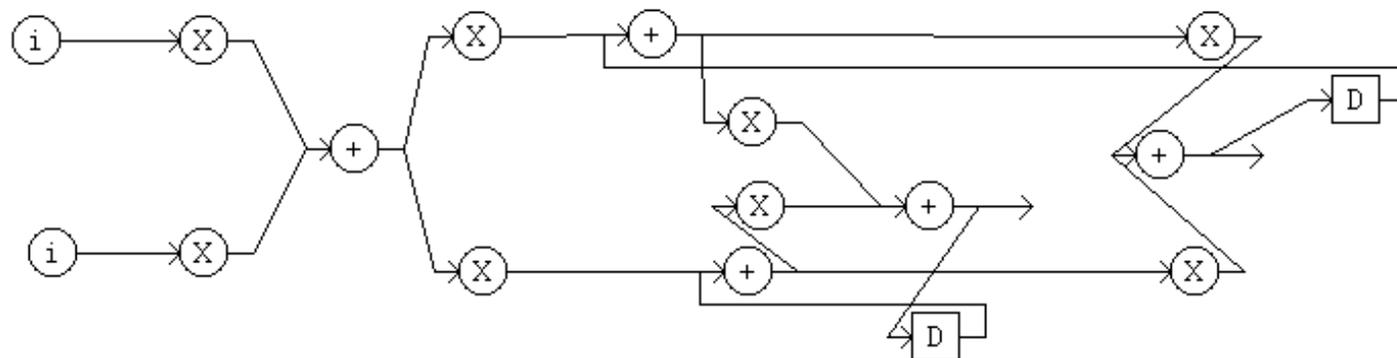
e15

\$ circuit description

\$ DFG level

\$	output	type	inputs
	e3	mul	e1
	e4	mul	e2
	e5	add	e3 e4
	e6	add	e7 e16
	e7	mul	e5
	e8	mul	e5
	e9	add	e8 e17
	e10	mul	e6
	e11	mul	e9
	e12	mul	e9
	e13	mul	e6
	e14	add	e10 e12
	e15	add	e13 e11
	e16	del	e14
	e17	del	e15

A4.1 DCST DFG



## A5. BIQUAD3 Netlist

\$ 3rd order biquad filter

\$

e1

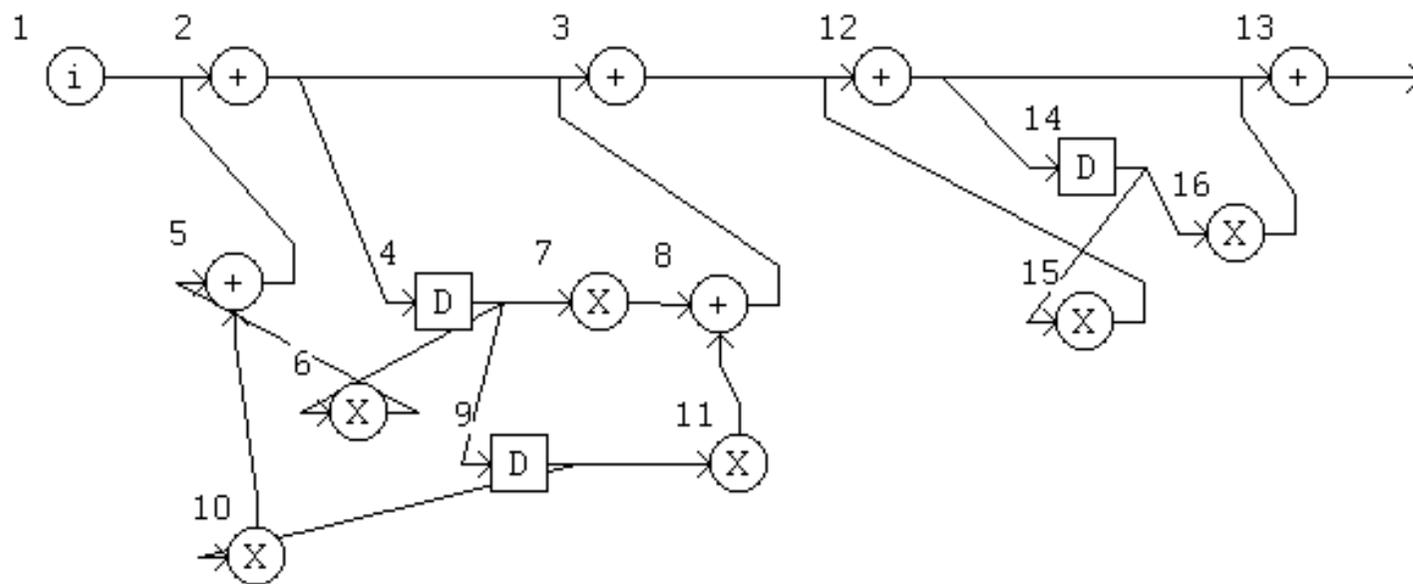
\$

e13

\$

outputs	type	inputs	
e2	add	e1	e5
e3	add	e2	e8
e4	del	e2	
e5	add	e6	e10
e6	mul	e4	
e7	mul	e4	
e8	add	e7	e11
e9	del	e4	
e10	mul	e9	
e11	mul	e9	
e12	add	e3	e15
e13	add	e12	e16
e14	del	e12	
e15	mul	e14	
e16	mul	e14	

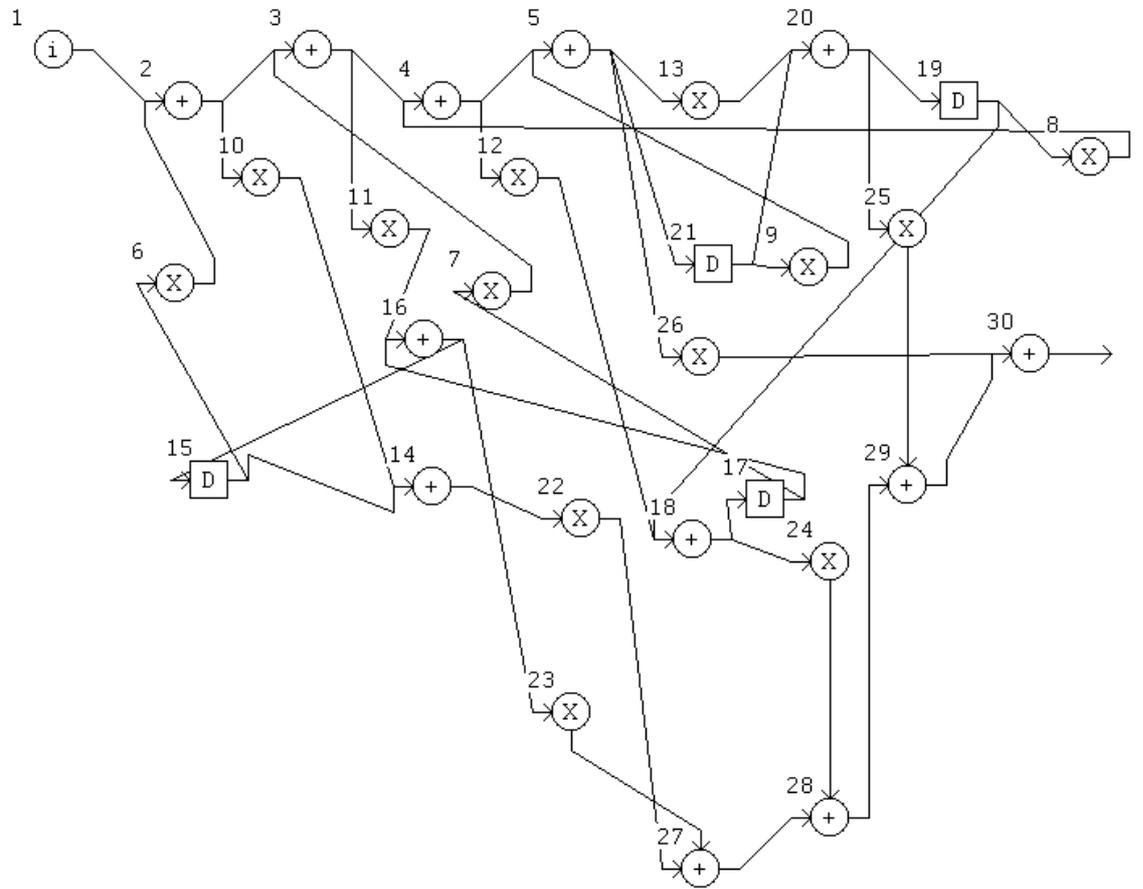
A5.1 BIQUAD3 DFG



## A6. GMLAT4 Netlist

```
$ 4stage gray-market filter
$ primary inputs
e1
$ primary outputs
e30
$ circuit description
$ DFG level
$      output  type  inputs
      e2      add   e1    e6
      e3      add   e2    e7
      e4      add   e3    e8
      e5      add   e4    e9
      e6      mul   e15
      e7      mul   e17
      e8      mul   e19
      e9      mul   e21
      e10     mul   e2
      e11     mul   e3
      e12     mul   e4
      e13     mul   e5
      e14     add   e10   e15
      e15     del   e16
      e16     add   e11   e17
      e17     del   e18
      e18     add   e12   e19
      e19     del   e20
      e20     add   e13   e21
      e21     del   e5
      e22     mul   e14
      e23     mul   e16
      e24     mul   e18
      e25     mul   e20
      e26     mul   e5
      e27     add   e22   e23
      e28     add   e24   e27
      e29     add   e25   e28
      e30     add   e26   e29
```

A6.1 GMLAT4 DFG



## A7. ELLIP5 Netlist

\$ 5th order elliptic wave filter

\$ primary inputs

e1

\$ primary outputs

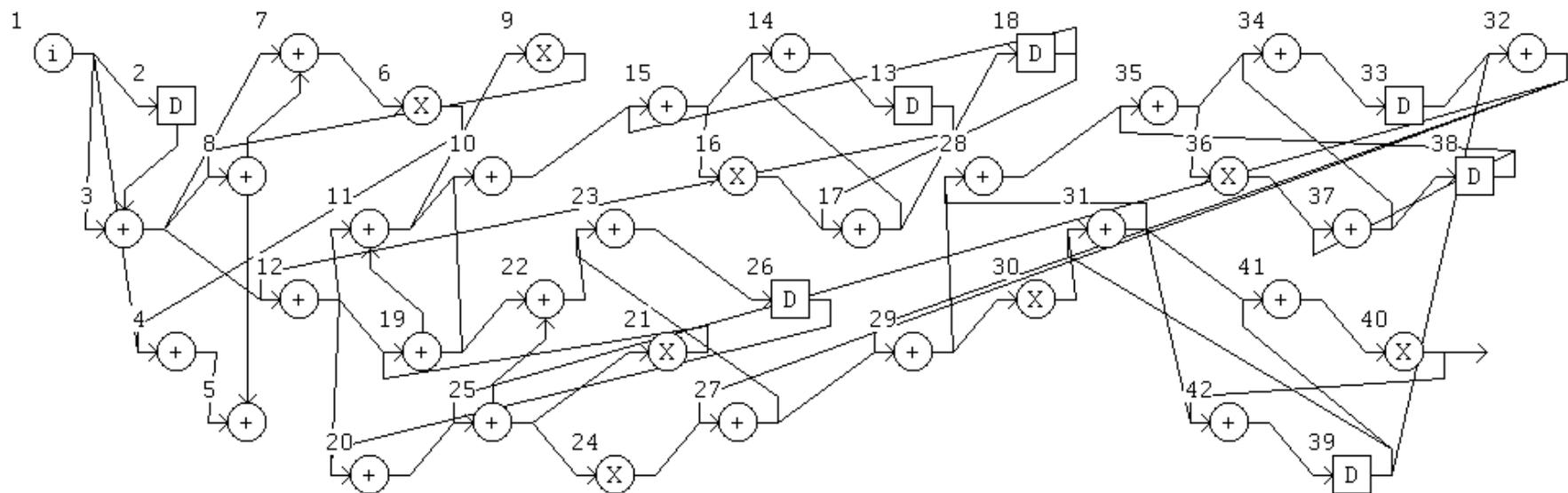
e40

\$ circuit description

\$ DFG level

\$	output	type	inputs
	e2	del	e1
	e3	add	e2 e1
	e4	add	e1 e6
	e5	add	e4 e8
	e6	mul	e7
	e7	add	e3 e8
	e8	add	e3 e9
	e9	mul	e11
	e10	add	e11 e19
	e11	add	e12 e19
	e12	add	e3 e13
	e13	del	e14
	e14	add	e15 e17
	e15	add	e10 e18
	e16	mul	e15
	e17	add	e16 e18
	e18	del	e17
	e19	add	e12 e21
	e20	add	e12 e26
	e21	mul	e25
	e22	add	e19 e25
	e23	add	e22 e27
	e24	mul	e25
	e25	add	e20 e32
	e26	del	e23
	e27	add	e32 e24
	e28	add	e29 e31
	e29	add	e32 e27
	e30	mul	e29
	e31	add	e30 e39
	e32	add	e33 e39
	e33	del	e34
	e34	add	e35 e37
	e35	add	e28 e38
	e36	mul	e35
	e37	add	e38 e36
	e38	del	e37
	e39	del	e42
	e40	mul	e41
	e41	add	e39 e31
	e42	add	e31 e40

A7.1 ELLIP5 DFG



## A8. LMS5 Netlist

\$ 5 stage LMS algorithm

\$ primary inputs

e1

e2

\$ primary outputs

e10

\$ circuit description

\$ DFG level

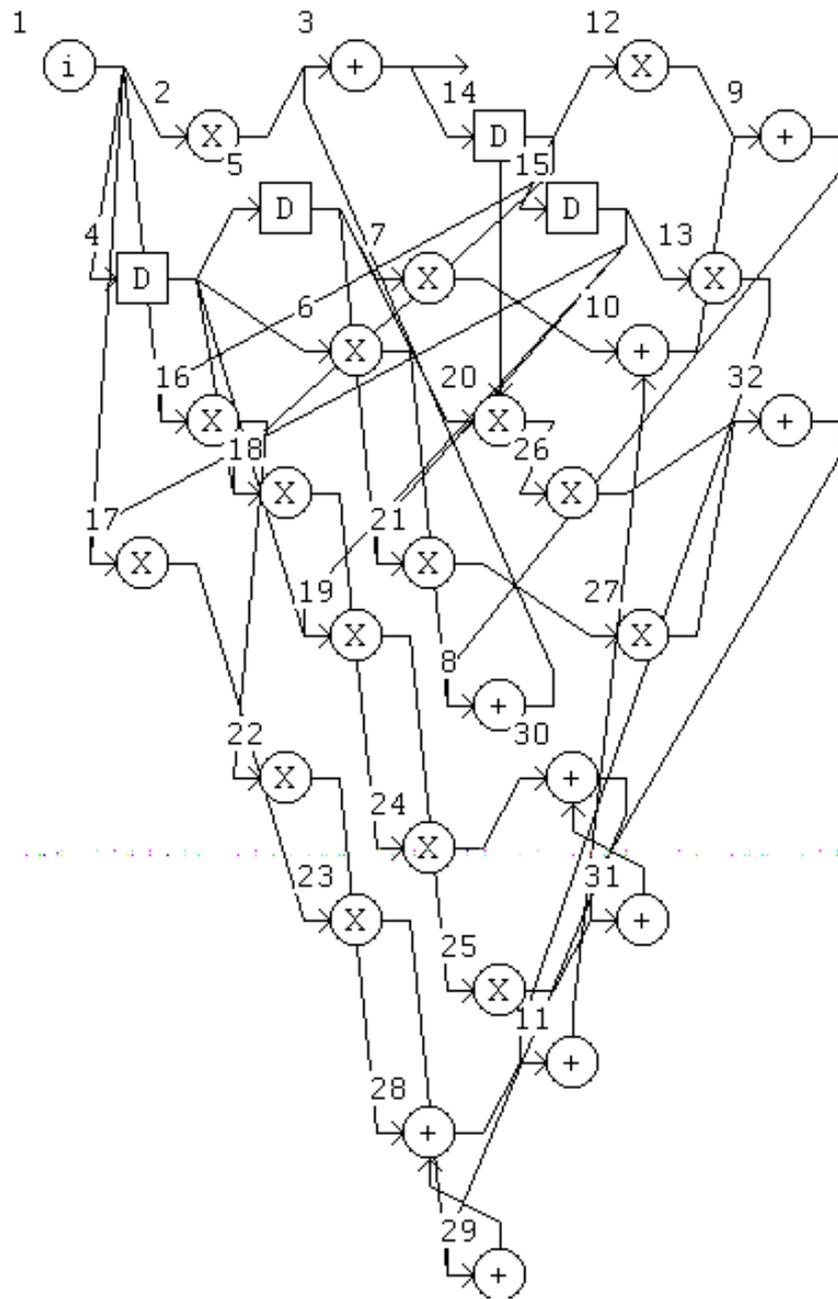
\$	output	type	inputs
	e3	mul	e1
	e4	del	e3
	e5	mul	e3 e10
	e6	add	e5 e7
	e7	del	e6
	e8	mul	e3 e7
	e9	add	e2 e8
	e10	add	e9 e17
	e11	del	e4
	e12	mul	e4 e10
	e13	add	e12 e14
	e14	del	e13
	e15	mul	e4 e14
	e16	add	e15 e22
	e17	add	e16 e28
	e18	del	e11
	e19	mul	e11 e10
	e20	add	e19 e21
	e21	del	e20
	e22	mul	e11 e21
	e23	del	e18
	e24	mul	e18 e10
	e25	add	e24 e26
	e26	del	e25
	e27	mul	e18 e26
	e28	add	e27 e32
	e29	mul	e23 e10
	e30	add	e29 e31
	e31	del	e30
	e32	mul	e23 e31



## A9. VOLTERRA Netlist

```
$ volterra filter
$ primary inputs
e1
$ primary outputs
e3
$ circuit description
$ DFG level
$      output  type  inputs
      e2      mul   e1
      e3      add   e2   e8
      e4      del   e1
      e5      del   e4
      e6      mul   e4
      e7      mul   e5
      e8      add   e6   e9
      e9      add   e10  e12
      e10     add   e7   e11
      e11     add   e28  e13
      e12     mul   e14
      e13     mul   e15
      e14     del   e3
      e15     del   e14
      e16     mul   e14   e1
      e17     mul   e1   e15
      e18     mul   e4   e14
      e19     mul   e4   e15
      e20     mul   e5   e14
      e21     mul   e5   e15
      e22     mul   e16
      e23     mul   e17
      e24     mul   e18
      e25     mul   e19
      e26     mul   e20
      e27     mul   e21
      e28     add   e22  e29
      e29     add   e23  e30
      e30     add   e24  e31
      e31     add   e25  e32
      e32     add   e26  e27
```

A9.1 VOLTERRA DFG



## A10. ORTH2LAT Netlist

```
$ 2nd order orthogonal lattice filter
$ 30/3/98
$ primary inputs
e1
$ primary outputs
e13
$ circuit description
$ DFG level
$      output type  inputs
      e2   mul   e1
      e3   add   e2   e11
      e4   add   e3   e15
      e5   del   e4
      e6   mul   e5
      e7   mul   e6
      e8   add   e7   e17
      e9   del   e8
      e10  mul   e1
      e11  mul   e12
      e12  mul   e5
      e13  add   e10  e14
      e14  mul   e12
      e15  mul   e18
      e16  mul   e6
      e17  mul   e9
      e18  add   e16  e19
      e19  mul   e9
```

A10.1 ORTH2LAT DFG

