Design space exploration for algorithm analysis and hardware architecture optimization

Saurav Bhattacharyya

2006

Saurav, B. (2006). Design space exploration for algorithm analysis and hardware architecture optimization. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/47486>

<https://doi.org/10.32657/10356/47486>

Nanyang Technological University

Downloaded on 26 Apr 2023 22:10:37 SGT

Design Space Exploration for Algorithm Analysis and Hardware Architecture Optimization

Saurav Bhattacharyya

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Master of Engineering

2006

TK
T885.7
S259
2006

Acknowledgements

In the capacity of a person who wrote this thesis and troubled innumerable others for support, the author would like to thank all those who contributed in their own ways towards the successful completion of this dissertation and the associated research.

With particular mention, the author thanks the members of the Centre for High Performance Embedded Systems (CHiPES) at the Nanyang Technological University (NTU) for their continuous and most attentive interest in what's new in system level design methodologies and what's not; where CHiPES can contribute greatly and where we do need to pay utmost contribution. Needless to say, all these discussions led to the formulation of a number of novel and inventive concepts for low power design and applications. Special thanks go out to Nah and Merilyn for facilitating various matters during the course of this research.

The author would like to take this opportunity to show his appreciation for Prof. Srikanthan for providing the opportunity in this area of research as well as enthusiastically supporting the endeavor in every manner. His support through mentoring the numerous changes in the directions and organic evolution of this master's research is most appreciated. Beyond the realms of this research and dissertation, the mentoring by Prof. Sri towards a patient approach and orchestrated evolution of thoughts shall stand the test of time and in many ways serves as a long-term investment for the author.

This research work would not have been possible if the teamwork with Kugan had not materialized part-way through the duration of this research. The numerous discussions and problem-hashing meetings were most instrumental towards shaping this research effort.

The author thanks Dr Clarke for taking time out from his whirlwind trips to CHiPES to hear me out on various low-power issues. He also expresses appreciation for the innumerable informal discussions with Mohit, which led to a better understanding of the close knit relationship with software design methodologies.

This research work would not have been possible without the continuous encouragement and support of Vishakha, Raj, Vikram, Manjeet and Mohit – all valuable friends for a lifetime. Tim is yet another individual who contributed in his own way over numerous lunches with valuable technical comments. Special mention goes out to Deepa, who was instrumental in the final push towards the successful completion of this dissertation with a horde of negative psychological techniques, and is definitely an individual who appreciates the finality of this effort.

Finally, the author thanks his sister and parents for their patient understanding towards the reasons for the various nuances in the timeline of this graduate work, and continuously encouraging me towards achieving this academic milestone.

The author also takes this opportunity to acknowledge the support received from Infineon towards this research on the rapid development of custom architectures for low power systems.

Table of Contents

<i>Acknowledgements</i>	<i>i</i>
<i>Table of Contents</i>	<i>ii</i>
<i>Index of Figures</i>	<i>vi</i>
<i>Index of Tables</i>	<i>x</i>
<i>Abstract</i>	<i>xi</i>
Chapter 1 Introduction	1
1.1 Motivation	1
1.2 Scope and Objectives.....	2
1.3 Major contributions of this dissertation	3
1.4 Thesis Organization.....	3
Chapter 2 Literature Survey	5
2.1 Role of Design Technology in Design Optimization	5
2.1.1 Design Process Challenges	7
2.1.1.1 Area-time-power optimization	8
2.1.1.1.1 Energy & Power – The new constraint for design optimization.....	10
2.1.1.2 System Level Design.....	11
2.1.1.3 Hardware-Software Partitioning.....	11
2.1.2 Power Consumption.....	12
2.1.2.1 Motivation for low power design	13
2.1.2.2 Power Dissipation	13
2.1.2.3 Reliability at high temperatures	14
2.2 Hierarchical approach to design optimization.....	15
2.3 Circuit Design	16
2.3.1 Power Consumption in conventional CMOS circuits	16
2.3.1.1 Dynamic Power	17
2.3.1.1.1 Device Capacitances.....	18
2.3.1.1.2 Switching Activity.....	19
2.3.1.1.3 Threshold Voltage Scaling	20
2.3.1.1.4 Voltage Scaling	21
2.3.1.1.5 Impact of voltage scaling on delay and power-delay product	21
2.3.1.1.6 Motivation for voltage scaling.....	23
2.3.1.2 Static Power	24
2.3.2 Different Circuit Styles.....	25
2.3.2.1 Dynamic versus Static Circuits	25
2.3.2.1.1 Spurious Transitions	25
2.3.2.1.2 Short-circuit Currents	26
2.3.2.1.3 Parasitic Capacitance	26
2.3.2.1.4 Power-down modes	26
2.3.2.2 Conventional CMOS Static gate versus Pass gate Logic:	27
2.3.3 Transistor Sizing.....	28
2.3.4 In Summary	28
2.4 Gate level	29
2.4.1 Logic Optimization Techniques.....	29
2.4.1.1 Retiming.....	29
2.4.1.2 Pre-computation Logic	30
2.4.1.3 State Assignment.....	30
2.4.1.4 Path Balancing	30
2.4.1.5 Clock gating and shutdown of functional blocks	31
2.4.2 In Summary	31

2.5	Architectural approaches.....	31
2.5.1	A data-path example of architecture driven optimization	32
2.5.2	Impact of micro-architecture.....	34
2.5.3	Motivation for Memory and Bus related energy reductions	35
2.5.4	Instruction set pipeline.....	36
2.5.5	In Summary	36
2.6	Algorithm Level	37
2.6.1	Design approaches at higher levels of design abstraction.....	37
2.6.2	Impact of number systems	38
2.6.3	LNS and RNS number representations for low power.....	39
2.7	System Design	41
2.7.1	Efficient bus architectures	41
2.7.2	Instruction set design	42
2.7.3	Multiple Supply Voltage Architectures	43
2.7.4	Partitioning the design based on latency requirements	44
2.7.5	Supply gating	44
2.7.6	Power-down Strategies	45
2.7.7	Modulating Clock frequency for low power.....	46
2.7.8	In Summary	46
2.8	Interconnect – The new digital divide.....	47
2.8.1	Technology limited constraints.....	47
2.8.2	Time of flight constraint	48
2.8.3	Physical level: Layout optimization techniques.....	49
2.8.3.1	Circuit Partitioning.....	49
2.8.3.2	Floor-planning.....	49
2.8.3.3	Placement	50
2.8.3.4	Routing.....	50
2.8.3.5	Gate Sizing	50
2.8.3.6	Wire Sizing.....	50
2.8.3.7	Clock Trees	51
2.8.3.8	Voltage swing on on-chip and off-chip interconnect	51
2.8.4	Problem with Interconnect analysis	51
2.8.5	Locality of Reference.....	51
2.8.6	Exploiting Regularity in Design for low power.....	52
2.8.7	In Summary	53
2.9	Summary	53
Chapter 3	<i>VHDL Based Techniques for Hardware Architecture Design Space Exploration</i>	56
3.1	K-means: The Algorithm	56
3.1.1	Algorithm description	57
3.2	Design Space Exploration using VHDL.....	58
3.2.1	Computing the <i>Euclidean</i> distance	59
3.2.2	Internal Memory	59
3.2.3	Architecture A: Massively Parallel Implementation.....	60
3.2.4	Architecture B: Folding by ‘m’.....	63
3.2.5	Architecture C: Folding by ‘m’ and ‘d’ for optimal tradeoff.....	64
3.2.6	Synthesis Results	65
3.3	Overview of the VHDL based design flow.....	66
3.4	Summary	67
Chapter 4	<i>SystemC Based Techniques for Hardware Architecture Design Space Exploration.....</i>	69
4.1	Constraint driven design space exploration.....	69
4.1.1	Energy/Power Constraints	69
4.1.1.1	Semantics of Power Consumption	69
4.1.2	Power Centric Approach.....	71

4.2	System Level Design Language: SystemC.....	72
4.3	SystemC based design flow	73
4.3.1	SystemC Simulation	73
4.3.2	Memory Instantiation for arrays	73
4.3.3	Compiling the SystemC code in dc_shell	74
4.3.4	Behavioral Analysis.....	74
4.3.5	Synthesis to Gate-Level Netlist	75
4.3.6	Power Estimation of architectures generated with SystemC	75
4.4	Effecting Architecture Transformations	77
4.4.1	Optimal word-length for data-path optimization	77
4.4.2	Pipelining based Optimizations	81
4.4.2.1	An initial Estimate.....	81
4.4.2.2	Pipelining lbl_F_LOOP7.....	83
4.4.2.3	Pipelining lbl_F_LOOP7 with a single cycle multiply.....	84
4.4.2.4	Pipelining loop lbl_F_LOOP8.....	84
4.4.2.5	Energy analysis	85
4.4.3	Parallel Computation & Memory Partitioning.....	87
4.5	Comparing VHDL and SLDL based design flow results	88
4.5.1	Results on system level partitioning using VHDL.....	88
4.5.2	Results on system level partitioning using SystemC	89
4.5.3	Summary of comparative results	90
4.5.3.1	Accuracy of estimation vs. simulation time	90
4.6	Summary	92
Chapter 5	System Level Analysis for Hardware Architecture Optimization.....	94
5.1	Cache Memory for Increased Performance	94
5.2	Techniques towards low power cache memory hierarchy.....	96
5.3	Prediction algorithms.....	98
5.3.1	Next-fetch-prediction table	98
5.3.2	NFPT Based Approach	98
5.3.3	Drawback of NFPT Scheme	99
5.3.4	Motivation for alternative prediction Scheme.....	99
5.3.5	Pattern history based predictor.....	99
5.4	Pattern Predictor	101
5.4.1	Experimental Setup.....	101
5.4.2	Pattern Predictor Performance Analysis	102
5.4.2.1	Prediction accuracy	102
5.4.2.2	Energy-delay product	104
5.4.2.2.1	Energy-delay product.....	104
5.4.2.2.2	Optimal FC size for energy-delay product reduction.....	104
5.4.3	SystemC based hardware architecture of the pattern predictor.....	105
5.4.4	Summary of experiments	106
5.5	Static Pattern Predictor	107
5.5.1	Static Pattern Predictor for Prediction of FC access	107
5.5.2	SystemC based hardware architecture of the static pattern predictor	109
5.5.3	Run-Time Optimization Options	110
5.5.4	Experimental Results.....	111
5.5.4.1	Prediction Accuracy.....	111
5.5.4.2	Normalized Cycle Time	111
5.5.4.3	Normalized Access Energy	112
5.5.4.4	Energy-Delay Product Reduction over NFPT for Static Pattern Predictor.....	112
5.5.5	Summary of Static Predictor Results	112
5.6	Micro-architecture (VHDL) Implementation for Pattern Predictor	113
5.6.1	Overview of the Filter Cache Hierarchy in a Processor.....	114
5.6.2	Pattern Predictor Architecture.....	114
5.6.3	Simulation Platform and Architecture Validation.....	116

5.6.4	Hardware Validation.....	116
5.6.5	Hardware Synthesis Results.....	117
5.6.6	Performance.....	117
5.6.7	Power Consumption.....	118
5.6.8	Energy-Delay Product.....	119
5.7	Overall Comparison of SLDL and HDL in Design Space Exploration	120
5.8	Framework for unified environment for system level evaluation of customized architectures.....	122
5.9	Conclusion.....	124
Chapter 6	Conclusion & Recommendations.....	125
6.1	Conclusion.....	125
6.2	Future Work	127
6.2.1	Design Entry and Specification capture.....	127
6.2.2	Extracting Inherent Parallelisms	127
6.2.3	Automation in SLDL to HDL Translation.....	127
6.2.4	Hardware Software Partitioning	128
6.2.5	Reducing Language Dependency	128
6.2.6	Physical Synthesis Approach to System Level Design.....	128
6.2.7	System Level Verification and Validation.....	128
6.2.8	System Level Estimation Tools	128
BIBLIOGRAPHY & REFERENCES.....		129
Conferences and Journals.....		129
Books & Technical Reports		133
APPENDIX A.....		134
APPENDIX B.....		137

Index of Figures

Figure 1.	Landscape of Design Technology [1]	6
Figure 2.	Required Evolution of Design System Architecture [1]	7
Figure 3.	Area-Time Constraints	9
Figure 4.	Area-Power-Time Tradeoff	9
Figure 5.	Design Exploration with Bounded Power Axis	9
Figure 6.	Design Exploration with Bounded Time Axis	9
Figure 7.	Design Exploration with Bounded Area Axis	9
Figure 8.	Battery Capacity Trends	13
Figure 9.	Sub-threshold Leakage as a function of die temperature	14
Figure 10.	Power consumption across processor generations	14
Figure 11.	Reliability of system parameters at High Temperatures	14
Figure 12.	The Cost of Removing Heat from a Microprocessor	15
Figure 13.	Hierarchies of Low Power Optimizations	16
Figure 14.	Charging up of capacitive load for a 1- 0 input transition	18
Figure 15.	Discharging of load transistor for a 0 to 1 input transition	18
Figure 16.	Parasitic capacitance model of a MOSFET transistor	19
Figure 17.	Effect of Capacitive load on short circuit current	19
Figure 18.	Compromise between dynamic and leakage power dissipation through V_t variation	21
Figure 19.	Power-Delay product exhibiting square-law dependence for two different circuits	22
Figure 20.	Graph demonstrating that delay characteristics follow a simple first-order theory	22
Figure 21.	Data showing improvement in power-delay product at the cost of speed for various circuit approaches	23
Figure 22.	Short-Circuit Dissipation	25
Figure 23.	Comparison of a conventional CMOS and CPL adder	27
Figure 24.	Tradeoffs in Transistor Sizing	28
Figure 25.	A simple data path with the corresponding layout [2]	32
Figure 26.	Parallel implementation of the simple data path [2]	33
Figure 27.	Pipelined implementation of the simple data-path [2]	34

Figure 28.	Power Density Evolution	35
Figure 29.	Reducing switching activity by optimizing number representation	38
Figure 30.	Activity reduction using sign-magnitude number representation.....	39
Figure 31.	Organization of an LNS adder/subtractor memory, composed of N small LUTs	41
Figure 32.	Logic with multiple supply voltages.....	43
Figure 33.	Sub-threshold Leakage as a function of Feature Size.....	44
Figure 34.	Dual Voltage Design Methodology	44
Figure 35.	Supply Gating or Sleep Transistor Technique	45
Figure 36.	Scaling of delays associated with gate and interconnect across generations.....	48
Figure 37.	Interconnect Capacitances	48
Figure 38.	Cascade filter layout – Non-local implementation from Hyper [32].....	52
Figure 39.	Cascade filter layout –Local implementation from Hyper-LP [32].....	52
Figure 40.	Preserving regularity leads to a simplified interconnect structure (a) regular assignment, (b) non-regular assignment	53
Figure 41.	Illustration of Clustering by K-means.....	58
Figure 42.	Iterative nature of the K-means algorithm	58
Figure 43.	Architecture A Description	62
Figure 44.	E-Distance Calculation	63
Figure 45.	Architecture-B.....	65
Figure 46.	(Architecture-C).....	65
Figure 47.	Area cost vs. nos. of folds, f.....	66
Figure 48.	%contrib. of various area vs. folds, f	66
Figure 49.	Architecture Design Space Exploration using HDL.....	67
Figure 50.	Illustration of (a) actual (b) peak and (c) average power consumption	70
Figure 51.	Area, Performance (time) & Power Tradeoff Trends in Server and Client Designs	71
Figure 52.	SystemC Based Design Flow.....	76
Figure 53.	SystemC fixed point designer reads in floating-point ANSI-C and writes out ANSI-C-code with all floating-point data types replaced by public open source SystemC fixed-point C++ data types	79
Figure 54.	Numerical analysis is allowed in floating-point, fixed-point and hybrid states ...	79
Figure 55.	Inserting in-line commands to analyze operator variances.....	80
Figure 56.	Constrained Data-objects	80

Figure 57.	A rich set of quantization and rounding modes area available in the SystemC fixed-point data types.....	81
Figure 58.	Schedule.....	83
Figure 59.	Lbl_F_LOOP7	84
Figure 60.	Illustration of Effective energy consumption	86
Figure 61.	Memory Architecture with (a) 8-bit word and (b) 32-bit word sizes	88
Figure 62.	Low Power flow using SystemC in CoCentric System Studio.....	89
Figure 63.	Power Estimation Methodology and Data Flow	89
Figure 64.	SystemC Compiler Input and Output Flow for RTL synthesis.....	89
Figure 65.	Depending on the abstraction of design implementation, simulation time for testing and debugging for SoC design can vary.....	91
Figure 66.	Architecture Design Space Exploration using SLDL	92
Figure 67.	Suggested Top-Down Feed Through of Power Budget Information to Block Level Optimization	92
Figure 68.	Prediction Algorithm	101
Figure 69.	Finite state machine of 2-bit saturating counter.....	101
Figure 70.	Simplescalar toolset overview	102
Figure 71.	Prediction accuracy for 256 and 512 byte FC.....	103
Figure 72.	Prediction accuracy with different PHT sizes.....	103
Figure 73.	Normalized energy-delay product for 256 and 512 byte FC	104
Figure 74.	Normalized energy-delay product.....	104
Figure 75.	Simplified architecture of prediction algorithm.....	106
Figure 76.	Static Pattern Prediction Algorithm	109
Figure 77.	Simplified Architecture of the prediction algorithm.....	110
Figure 78.	Prediction Accuracy for 128 and 256 byte FC.....	111
Figure 79.	Normalized Cycle Time for 128 and 256 byte FC.....	111
Figure 80.	Normalized Access Energy for 128 and 256 byte FC.....	112
Figure 81.	Access Paths for (a) conventional, (b) hierarchical FC and (c) FC prediction, Based Instruction Memory Hierarchy.....	114
Figure 82.	Single Cycle Pattern Prediction Architecture for FC Access Prediction.....	115
Figure 83.	Experimental Setup for Architecture Validation & Power Analysis	116
Figure 84.	Filter Cache and Predictor	117
Figure 85.	Typical HDL only design flow	121

Figure 86.	SystemC based unified flow for algorithm and architecture designer	121
Figure 87.	Unified Algorithm and Architecture Design Space Exploration	122
Figure 88.	Elaboration of Architecture Design Space Exploration for Algorithm Analysis	122
Figure 89.	Language and Tool Agnostic Framework of Unified Environment	123

Index of Tables

Table 1. Estimation of Power Dissipation in LNS subtraction-addition, in comparison to $n \times n$ bit fixed-point multiplication, for various fixed-point word lengths, n , and bases b 40

Table 2. Comparison of power consumption (mW) in the Hyper and Hyper-LP implementation of the cascade filter 52

Table 3. Time Complexity 62

Table 4. Design Complexity for the Various Architectures..... 63

Table 5. Word-length optimization based on accuracy..... 78

Table 6. Table V1..... 82

Table 7. Optimization Results at a glance (1M = 1 million clock cycles)..... 86

Table 8. Tabulation of results for various implementations with different word sized RAMS.. 87

Table 9. Power Requirement..... 88

Table 10. Impact on power consumption using SystemC for architecture partitioning 90

Table 11. Hardware area cost for predictor configurations without FC 106

Table 12. Proposed Static Prediction Table..... 108

Table 13. Hardware complexity comparison of NFPT and Static Pattern Predictor 110

Table 14. Percentage Energy-Delay Product Reduction with Static Pattern Predictor Compared to NFPT 112

Table 15. ASIC Synthesis Results 117

Table 16. Average Delay (in cycles) per access (Performance Metric)..... 118

Table 17. Power Consumption..... 119

Table 18. Energy Delay Product 120

Abstract

Designing embedded systems products is an increasingly challenging proposition due to multiple functional and non-functional constraints posed to design architects. Amongst non-functional constraints, the non-recurring engineering cost (NRE) and time to market (TTM) take a high precedence in decisions to be taken at the design and development stage. Further, due to the increased complexity of designs, and the yawning productivity gaps, designing products to meet specific design points in the area-time-power tradeoff analysis has become a difficult balancing act.

The comprehensive literature survey delved into the various techniques for design space exploration of hardware architectures at all levels of design hierarchy. Design optimization techniques at the gate and circuit level of hardware architecture design, have been examined in detail. The various time consuming techniques that are being incorporated by electronic design automation (EDA) tool vendors have also been investigated to evaluate their effectiveness towards realization of optimal hardware architectures. International technology roadmap reports for semiconductors (ITRS) were consulted to examine the industry trends in terms of anticipated growth and demands, and the challenges in meeting the design requirements thereof. System level analysis tools for optimal algorithm design and custom hardware design space exploration have been studied to evaluate the potential limitations particularly when constraint-aware hardware architectures must be designed despite increasing density of designs, TTM and NRE pressures.

In this research work, a thorough comparison between hardware description language (HDL) and system level design language (SLDL) based design space exploration has been established. K-means, an iterative clustering algorithm is used as an isolated architecture design vehicle to first benchmark the trade-off between HDL and SLDL based development using VHDL and SystemC respectively. Whilst, the SLDL domain analysis can provide for crude estimates of hardware architecture costs, it provides for detailed exploration of architecture variations and the relative costs before committing to detailed implementation in HDL. The distinct advantages of each design approach for custom architecture design has been demonstrated through implementations.

The clear benefits of extending the SLDL based design environment to facilitate the detailed analysis of the hardware architecture implementation at the system level have been made. A comprehensive design space exploration environment for a predictive instruction cache memory hierarchy was established to demonstrate the potential limitations of the isolated evaluation of hardware architectures using SLDL. System level profiling tools such as SimpleScalar and CACTI were selectively integrated with the SLDL based design flow to provide effective system level algorithm analysis and architecture exploration. The effectiveness of this approach towards algorithm innovation for system level optimization has been clearly validated through detailed simulations. The HDL based design implementation has also been integrated into this flow to allow for the eventual porting of the algorithm by taking full advantage of the low-level implementation of the hardware architecture. This environment ensures design closure and effective trade-off analysis without violating TTM and NRE concerns.

This research work has led to the abstraction of a framework for a unified design implementation environment to facilitate the algorithm analysis, detailed evaluation and implementation of hardware architectures by taking system level interactions into account. A SystemC environment has been established to integrate the profiling tools as well as the SLDL and HDL based representations. The proposed framework provides for a platform for a systematic approach to system level algorithm analysis and optimal hardware design of custom architectures.

Chapter 1 Introduction

Engineering a design has never been an exact science with a single solution for a given problem. Engineers juggling often-conflicting constraints and requirements to arrive at an optimum solution can be likened to artists with a palette of colors and canvas. Extending the analogy, an engineer also has to amalgamate his parameters and constraints to give birth to a new and more suitable solution. To be able to trade-off and balance the various requirements, given the constraints is an art. To be able to do so logically in a repeatable manner is science.

The equilibrium point of this process is specific to each system being designed. In the context of integrated circuit (IC) and embedded system hardware architecture design this means balancing costs such as silicon area, latency of computation, design time, and product size and format restrictions. In addition to functional constraints such as area and performance, due the high mobility of designs, power consumption is also becoming a constraint due to the disproportionate scaling (of power) with respect to area and latency, as the feature size is reduced. Thus, adding one more dimension to the cost graph and complicating the process of deducing the maximum performance for the cost function.

1.1 Motivation

The unprecedented growth of portable entertainment and communication devices has provided the impetus for the development of sophisticated embedded system designs. The performance demands of these devices, met by massive technology improvements in circuit design, come with the onus of higher design productivity. Further, deep sub-micron designs also imply a penalty in terms of higher power consumption. Comparatively stunted improvements in battery technology coupled with market demands for higher battery life has thus forced designers to address energy efficiency to make the portable devices viable. In an industry where small is fashionable, adequate cooling devices are inappropriate. This makes it important to address heat dissipation. It is thus no surprise that energy efficiency and power consumption are taking the center stage of research in the development of portable and embedded systems. Nevertheless, a single minded focus towards low power leads to a neglect towards area and performance measures which is also not permissible in the highly competitive environment of the embedded systems product industry. Thus, dictating a balanced approach to design space optimization when looking at improvements in any one dimension.

Adding another degree of complexity to the design space comes at the increased cost of design time and analysis. This adds to the time to market (TTM) and non recurring engineering (NRE)

costs for a design exercise. Higher productivity design tools focusing on these objectives are thus necessary to aid this process. However, at this point researchers worldwide are still unable to even estimate reliably the power consumption of a design at the initial stages, let alone an efficient methodology for optimization. This is further complicated by the applicability of various stages of design at which the power consumption can be addressed. This complicates matters by providing an abundance of disjoint methods that a single design engineer might be unable to address.

There is also the need to address functional constraints such as area, performance and power consumption at very early stages of design. This is especially so, given the implications of design re-engineering costs and the short time lines for development in the complicated dense designs that are required of products today. Having a relative scale of costs in terms of area, power consumption and performance at this stage to trade off different architecture variations would thus be invaluable to a designer as this will make the investment in time and resources for the complete design more focused. This is aimed towards a resulting embedded system which meets the design specifications and constraints posed at requirements analysis.

In this dissertation, the author has addressed various ways of design space exploration and design partitioning to address the trade-off in the various functional requirements. A system level design approach to addressing power consumption in tandem with micro-architectural mechanisms provide a complete suite of tools for achieving dedicated architectures and microprocessor systems with application specific requirements. The author has used a confluence of system level techniques for architecture cost estimation at a very early stage of design and ground-level detailed architecture definition to allow detailed analysis and impact of the hardware architecture on the overall system design. This hybrid approach to design space exploration and partitioning provides the designer with an optimization approach which further increases the probability of convergence to the design requirements.

1.2 Scope and Objectives

- To survey design space exploration techniques at all levels of design and implementation hierarchy with an emphasis on low power.
- Evaluate the strengths and weaknesses of hardware description languages (HDL) and system level design language (SLDL) based approaches to design space exploration and design partitioning.

-
- Devise a hybrid approach to design space exploration to partition algorithms and architectures towards achieving the specific design constraint requirements.
 - Propose a unified environment for system level evaluation of customized architectures.

1.3 Major contributions of this dissertation

The contributions of this research effort can be summarized as follows:

1. A comprehensive literature survey of design optimization techniques with an emphasis on power consumption at all levels of design abstraction
2. Trade-off analysis of architecture exploration and design partitioning of K-means algorithm using hardware description language and system level design language based approaches.
3. Novel pattern prediction architecture for dynamic and static prediction algorithms for filter cache based instruction memory hierarchy.
4. Basic framework for unified environment for the system level exploration of customized architectures using a hybrid design space of system level design languages and hardware description languages.

A number of international conference and journal publications resulted as a direct consequence of the innovative work in this research effort. The publications span industry user group conferences and academic publications of a high standard. This is representative of the relevance of the research presented in this dissertation. These publications are listed in Appendix A

1.4 Thesis Organization

Chapter 1 of this report introduces the motivation and objectives of this research. The publications related to this work by the author are listed as outputs to academia as a result of this research. Chapter 2 introduces the role of design technology in design optimization. It then continues to discuss techniques for design space optimization of embedded designs with a focus on the emerging constraint – power consumption. This chapter also sets the scene for the optimization approach taken in the rest of the chapters of this thesis.

Chapter 3 marks the beginning of the research contributions with hardware description language based approach for design partitioning of the K-means algorithm. It also highlights the advantages and disadvantages of using VHDL as a language to effect design optimizations and evaluations. Chapter 3 sets the scene to experiment with system level design languages and this

is done in Chapter 4. Chapter 4 illustrates how existing tools and SystemC (a system level design language - SLDL) can be used to evaluate different architectural translations and evaluation of their area, power and performance measures. Power consumption is taken as a constraint and area and performance measures are evaluated for an effective trade-off analysis in this design space exploration exercise. Finally, SLDL and HDL based approaches to architecture partitioning and evaluation are compared and contrasted to highlight the applicability in different scenarios of design.

Having evaluated HDL and SLDL based approaches to architecture evaluation in design space optimization efforts, Chapter 5 illustrates how system level tools such as SimpleScalar and CACTI are required to effectively morph algorithms to suite applications. As a result of application specific analysis, algorithms are innovated in this section which are then subjected to quick SLDL based cost-performance tradeoff analysis. This illustrates the necessity of SLDL based architecture analysis to allow for relative quantitative analysis of the resulting architectures of different algorithm variations to be used in a design. Chapter 5 then continues to provide a more in-depth analysis, using HDL based design of the algorithm variations to derive more accurate power, area and performance figures, which when augmented with the system level figures from SimpleScalar and CACTI lead to a concrete analysis of architecture cost in the system. Finally, Chapter 5 concludes with a framework for unified environment for system level evaluation of customized architectures. This framework empirically encapsulates a system level analysis and simulation environment that is specific to the application with SLDL based approach to perform selection resulting from design space exploration of the different possible architecture variants and HDL based approach for detailed analysis and implementation.

Chapter 6 then concludes the report and provides recommendations towards further research leading from this effort.

Chapter 2 Literature Survey

This chapter begins with some generic issues facing the design optimization for embedded designs today. Low power is also discussed as an increasingly important constraint for design optimization in section 2.1. Section 2.2 discusses the hierarchical approach to design optimization. The rest of the chapter takes a walk through the various abstractions layers at which design optimization can be applied. At all layers, the tradeoff of a single factor based design optimization (example: power consumption) approach is also discussed with its impact discussed on other design parameters (for example: associated silicon area and performance measures).

Section 2.3 introduces the basics of power consumption in transistors, and the various approaches that can be taken to provide a low power design. Tradeoffs of the various approaches at the semiconductor level are also discussed. Section 2.4 follows suit with gate-level approaches to power optimization. Section 2.5 then introduces architecture level approach to design-space exploration where a block level approach is taken to reduce power consumption. Section 2.6 then illustrates how design optimization can also originate at the other end of design hierarchy at algorithm development stage with a focus on low power. It is also noted that algorithm stage of optimization also has to deal with algorithm accuracy concerns especially in data driven applications. Section 2.7 then discusses system level approaches to design-space exploration. Having dealt with logic level issues, Section 2.8 discusses the impact of interconnect on power consumption. Having dealt with all levels of hierarchy in power optimization, Section 2.9 establishes a focus as a result of the literature survey and summarizes this chapter.

2.1 Role of Design Technology in Design Optimization

Design technology enables the conception, implementation and validation of microelectronics-based systems. Design technology includes: tools, libraries, manufacturing process characterization and methodologies. Design technology is the link that transforms and validates the ideas and objectives of a designer into a representation that can be manufactured and tested before committing to fabrication of the design. As an example of the economic impact of design technology improvements: If there had been no design technology improvements between 1993 and 2001, then the estimated design cost of a low-power System on Chip (SOC) PDA would be US\$342 million as compared to US\$15 million in factual terms [1].

As highlighted by ITRS [1], amongst the many design technology challenges facing the industry include, Design Process, System-level design and Logical, circuit and physical design. In Figure 1 and Figure 2, it is apparent that system-level design needs to encapsulate design partitioning and analysis on the key design parameters such as area, performance and power consumption at an early stage in design. Figure 1 further shows that there is a need to elevate design partitioning and design space exploration to the system level and use the RTL flow as a pure implementation layer based on specifications received as a result of the design partitioning and estimation exercise at the system level. Amongst the many challenges identified by ITRS, the subsections that follow elaborate on a few. These are challenges specifically addressed in this research

Figure 1. Landscape of Design Technology [1]

Figure 2. Required Evolution of Design System Architecture [1]

2.1.1 Design Process Challenges

To avoid exponentially increasing design cost, design productivity has to scale to the increasing requirements. The spectrum of design flows range from loosely coupled tools integrated via industry-standard sequential files to tightly integrated systems with closed/proprietary interfaces. In high-end design projects, no single design technology vendor, nor collection of all design technology vendors together, provide the full scope of required design technology. Hence, internally developed design technology must augment multi-vendor design flows. This is clearly addressed in this master's research.

To address system complexity, tool integrations must minimize unproductive data translation time and data redundancy, and support synthesis-analysis tool architectures. Time-to-market constraints of cost-driven SOC require common information models to describe the characteristics of reusable cores at levels of abstraction that permit efficient design optimization. New tools and integration will be required to experiment with reuse-centric design space exploration and design optimization. Electronic system design will always be to some extent an

art rather than a science. However, the precept of “measure, and then improve” must be applied if design technology is to advance systematically and rapidly. To develop methods for continuous design process and design productivity improvement, near-term technology needs begin with metrics of a design process with respect to quality and predictability. These in turn depend on metrics of design quality and design process predictability.

2.1.1.1 Area-time-power optimization

One of the most pervasive optimizations across the domain is that of silicon area, power and time. All micro-computer systems today consist of some form of digital and analog logic design on silicon substrate. Considering the large non-recurring costs of such process flows, the efficiency in the use of the silicon real-estate becomes a very important constraint. At the very basic level it involves the optimization of silicon real-estate, (i.e., area) whilst extracting the best performance (i.e., time) for the least amount of power consumption.

Before entering system level design, the basics of design exploration for integrated circuit designs at a high level of abstraction is visited. In most ASIC designs, the intent to partition is primarily to meet the area-time constraints as shown in the figures below. However, with increasing prominence a third dimension is also presenting itself as a constraint in the design partitioning exercise as shown in the figures below.

With the graphs in context, the designer can now partition the design based on maximal constraints in the area, power, or time axis. Design exploration with bounded area axis is now becoming close to obsolete due to the immense amount of density available in logic onboard custom silicon. However, one must not underestimate the extra cost involved in engineering complex designs with multi-million gate count. A combination of two axes can lead to a constrained space for design exploration. However, for a power optimal design, the upper bound on the power budget needs to be fixed before exploring the design space.

Figure 3. Area-Time Constraints

Figure 4. Area-Power-Time Tradeoff

Figure 5. Design Exploration with Bounded Power Axis

Figure 6. Design Exploration with Bounded Time Axis

Figure 7. Design Exploration with Bounded Area Axis

2.1.1.1.1 Energy & Power – The new constraint for design optimization

Energy is a quantitative measure of the amount of work. Power on the other hand is the rate of doing the work, or the rate of delivering the energy. Conversely, it also represents the rate of consumption of energy.

Consider a system which takes T_{ref} amount of time to process a particular task consuming E_{ref} at the rate of consumption P_{ref} . Thus, total energy required for the reference operation O_{ref} is:

$$E_{ref} = P_{ref} \times T_{ref} \quad (2.1)$$

To provide the required energy (E_{ref}) for the reference implementation of the task, a battery of weight W_{ref} is required. If the energy per unit weight of the battery is $E_{density}$, then,

$$E_{ref} = E_{density} \times W_{ref} \quad (2.2)$$

The technology driven elements in the above two equations are $E_{density}$, and E_{ref} , whereas, P_{ref} and T_{ref} are dictated by the design.

Analyzing equation 2.1 it is deduced that the modification of the operation to make it slower (increasing T_{ref}) will result only in reducing the power dissipation factor (P_{ref}), keeping the total energy for the task the same (E_{ref}). Thus, tweaking the architecture to reduce the power consumption does not benefit portable devices, as this only means that the time required for the task is now higher. The battery size has to be kept the same since the amount of energy required remains unchanged. However, assuming that there are no major changes in the algorithm and architecture, the responsibility of reducing the system weight is transferred to the battery technology. From technology roadmaps it is noted that batteries are not envisaged to keep pace with the energy requirements of embedded systems. This puts the onus back on the efficient design of digital logic devices. Thus, the focus on power optimization should be (implicitly) to reduce the amount of energy required for the task, while using the power dissipation as a metric to decide on trading off different architectures. Further, it is interesting to note that most applications demand that the performance factor – represented by T_{ref} in equation 2.1 is not to be sacrificed. This implies that the device should basically be consuming less energy whilst delivering the same performance.

This can thus only be achieved if the operations used in the particular computation are changed to reduced-intensity operations implying increased energy efficiency. In certain applications a careful evaluation of the algorithm can provide the necessary optimizations. As an example, a divide-by-2 operation can be replaced by series of arithmetic right-shift operations (or a barrel

shifter if the constraints allow). Simulations can be carried out to decide if the operation replacement can be applied in view of the accuracy tradeoffs.

2.1.1.2 System Level Design

The electronic system design process enables designers to allocate and exploit silicon resources in a top-down and structured fashion. At the system-level, silicon resources are defined in terms of abstract functions and blocks; design targets include software (embedded code in high level and assembly language), and hardware (cores, hardwired dedicated circuits, busses and reconfigurable cells). Behavior and architecture are independent degrees of design freedom, with hardware and software being two components of architecture. The aggregate of behaviors defines the system function, while the aggregate of architecture blocks defines a system platform. Platform mapping from system functionality onto system architecture is at the heart of system-level design and becomes more difficult with increased system complexity and heterogeneity.

The increasing range of implementation fabrics requires new tools for optimization of the function-architecture mapping. Criteria for such optimizations include power, area, throughput, etc. Estimation technology is a fundamental and yet-undeveloped adjunct of top-down design space exploration. Estimators allow accurate prediction of eventual values of the design criteria without going through all details of the design process. Such a capability must increasingly span multiple levels of abstraction. This is to be augmented by efficient behavioral synthesis. A continuing challenge for design verification here is how to specify the desired behavior of a design. A deeper understanding of what makes a specification clear or opaque, modifiable or intractable will be needed to guide the development of the languages needed to specify ever more complex designs. In addition, designers will have to be trained to use these languages and to become more disciplined about writing specifications.

These challenges of system complexity and design productivity create the need for:

- Higher level abstraction and specification
- Design space exploration and system-level estimation

2.1.1.3 Hardware-Software Partitioning

System level design also entails hardware–software partitioning, which decides the mapping of key processing tasks into either hardware or software, and which has major impact on system performance, energy consumption, on-chip communications bandwidth consumption, and other

system figures of merit. Multi-processor systems require “software” partitioning and co-design, i.e., assignment of software tasks to various processor options. While perhaps 80–95% of these decisions can be made *a priori*, particularly with platform-based or derivative SoCs, such co-design decisions usually made for a small number of functions that have critical impact. Thus, a system level environment for trade-off analysis is crucial towards an effective hardware-software partitioning exercise. Effective co-design is one of the enabling factors to design technology for partitioning, whether of hardware-software, analog-digital, fixed-reprogrammable, or die-package (-board), must be increasingly aware of cost and risk factors across the entire system development process. Trade-offs and partitioning across analog-digital boundaries require compatible specification and functional modeling abstractions, as well as the ability to simulate all parts of a system within a single execution and analysis environment. More generally, design optimization must increasingly consider a device in its system context rather than as a standalone entity.

2.1.2 Power Consumption

Non-ideal scaling of planar CMOS devices, together with the roadmap for interconnect materials and package technologies, presents a variety of challenges related to power management and current delivery. Increasing power densities worsen thermal impact on reliability and performance while decreasing supply voltages worsen switching currents and noise. These trends stress on-chip interconnect resources. Integration of distinct high performance, low operating power and low standby power devices demands power optimizations that simultaneously exploit many degrees of freedom [1].

With multi-million gate integrated circuits becoming commonplace, the designer cannot be faulted to ignore power consumption. Power consumption is an inevitable side-effect of a working design. Any kind of work done requires energy and this implies the power required by the task to complete the job in a given amount of time. Thus, the designer should try and reduce the wastage due to non-ideal physics to the minimum, and optimize power consumption for a given application.

Power dissipation is an inherently latent and non-tangible parameter in logic design and synthesis. In fact, until recently, most computer aided design (CAD) tools did not even list it as a parameter for module choice during compilation, let alone offer efficient compilation options for low power focussed designs. Only recently have power consumption reports become common place. Design Technology (DT) has to evolve to suit the needs for the next generation problems. This is highlighted in the ITRS-Design Roadmap – 2001 [1].

2.1.2.1 Motivation for low power design

Several factors in the current trends of product development have brought power consumption in integrated circuits into the spotlight. The myriad of portable and handheld devices, which have populated the consumer market, and the projection that this number is only going to grow further, has fuelled the need to focus on low power design methodologies.

Figure 8. Battery Capacity Trends

Amongst the most important objectives in portable device design is to maximize run-time and minimize size and weight associated with the batteries. However, battery technology has not scaled (Figure 8) in the same manner as power consumption by processors. Future projections also do not indicate the battery technology being able to close the gap. This has resulted in power budget having surpassed area and time as the primary constraint in the design of portable devices.

2.1.2.2 Power Dissipation

Design constraints for a portable product are not the only driving force for low power designs. Current trends in micro-processor designs dictate that processors will run at about 10-30 GHz in about 10 years from now, capable of about 1 trillion operations per second – about the same as the world's fastest supercomputer today. However, if all remains constant, then in proportion, the heat dissipated by such devices will be the same as nuclear reactors. This implies that the cost associated with cooling these processors can become prohibitive. The expenses include higher packaging costs, and run-time cooling costs. Thus, there is a clear advantage in reducing the power consumed by high performance systems.

Figure 9. Sub-threshold Leakage as a function of die temperature

Figure 10. Power consumption across processor generations

What is even worse is that power dissipation increases as running temperature of a device increases. This is due to the increased sub-threshold leakage, which is an inherent property of a transistor. This is illustrated in the Figure 9.

Figure 11. Reliability of system parameters at High Temperatures

2.1.2.3 Reliability at high temperatures

In addition to costs, reliability (Figure 11) is severely compromised at the high temperatures that these chips are subjected to at normal operational modes. This has to be addressed by either retarding the growth of device densities, implementing better cooling strategies or by reducing power dissipation and consumption. Retarding not being an option, and cooling systems being prohibitive in size and cost (Figure 12) implies that one needs to provide better power management at the integrated circuit and the embedded systems level.

Figure 12. The Cost of Removing Heat from a Microprocessor

2.2 Hierarchical approach to design optimization

Design parameters in an integrated circuit device is primarily accounted for by the transistors performing the logic functions, and interconnect which is used to stitch up the logic and the drive currents required by off-chip driver circuitry. One can argue that if the basic building blocks are optimized then addressing the issue at the higher levels of design is not required. However this is vetoed by the extensive thrust of hierarchical design strategies by the electronic design automation (EDA) companies and the increased reluctance of designers and backend engineers to touch legacy and “golden” design units. To bear testimony to the fact, higher levels of abstraction such as circuit, gate, architecture, algorithm, and system level design are used to reduce the complexity involved in handling million gate designs and making the design effort more manageable. Hence, decisions taken at the higher design levels eventually affect the building blocks, giving the basis to low energy optimizations to be carried out at higher levels of design hierarchy.

In the sections that follow, the various layers of design optimization in a hierarchical fashion as shown in Figure 13 is explored. Whilst the sections on Circuit and Gate level concentrate on power consumption issues, the higher level of abstractions focus on design abstraction and related issues.

System
Algorithm
Architecture
Gate
Circuit
Device / Technology

Figure 13. **Hierarchies of Low Power Optimizations**

2.3 Circuit Design

Circuit level is the most basic level of abstraction at which design optimization can be discussed. However, assuming a cell-based ASIC design flow, transistors of a particular feature size are characterized and cannot be tampered any further to meet the requirements. This does not imply that transistor design issues such as transistor sizing to reduce power dissipation need not be discussed. However, it does imply that the domain of substrates and physical properties of contacts in layout technology is out of scope for this dissertation.

Conventional complimentary metal oxide semiconductor (CMOS) gates form the building blocks of most digital logic designs today. Variant circuits of the same have been designed over the years due to various design requirements that have arisen at various stages of evolution and advancements in logic design. Thus the design space at the transistor level now presents quite few choices: conventional CMOS, Pass Transistor Logic (PTL), Dynamic logic and Domino gates. However, most designs are still constructed using static CMOS gates. Hence, the conventional CMOS circuit is discussed in detail in the sections that follow and compared to some of the other circuit design techniques.

2.3.1 Power Consumption in conventional CMOS circuits

There are four sources of power dissipation in digital CMOS circuits, which are summarized, in the following equations:

$$P_{avg} = P_{switching} + P_{short-circuit} + P_{leakage} + P_{static} \quad (1)$$

The power consumption of static CMOS gates can be basically divided into two parts: dynamic dissipation ($P_{switching}$), which occurs when the gate switches; and the static dissipation (P_{static}), which occurs when the gate output and inputs are held at a constant logic value. Further, there is short-circuit ($P_{short-circuit}$) and leakage ($P_{leakage}$) power consumption which is a function of the semiconductor feature size. These values are affected by the scaling in feature sizes of CMOS technology. This shall also be elaborated in the discussions that follow.

2.3.1.1 Dynamic Power

Equation 1 can be further elaborated as the follows:

$$P_{avg} = \alpha_{0 \rightarrow 1} C_L V_{dd}^2 f_{clk} + I_{sc} V_{dd} + I_{leakage} V_{dd} + I_{static} V_{dd} \quad (2)$$

$$P_{avg} = C_{eff} V_{dd}^2 f_{clk} + I_{sc} V_{dd} + I_{leakage} V_{dd} + I_{static} V_{dd} \quad (3)$$

The first term represents the switching component of power, where C_L represents loading capacitance, f_{clk} is the clock frequency and $\alpha_{0 \rightarrow 1}$ is the probability that a power consuming transition occurs (the activity factor). C_{eff} is the effective capacitance of the circuit given the probability of a 0 to 1 transition ($\alpha_{0 \rightarrow 1}$). In most circuits the voltage swing (V) to which the capacitance is charged is equal to the power supply rail (V_{dd}).

When a 1 to 0 transition occurs [Figure 14] on the input, the pull-up PMOS (P-channel metal oxide semiconductor) network of the CMOS gate is switched on and the output load capacitance (C_L) is charged up to the maximum swing voltage (typically V_{dd}). This results in energy expended to charge up external capacitances. Thus, making it imperative that load capacitance should be reduced for lower power dissipation.

Further, when a 0 to 1 input transition occurs (Figure 15), the charge in the load capacitance is drained back to the supply ground through the pull down N-channel metal oxide semiconductor (NMOS) network. Thus, the energy required for a single 1 to 0 to 1 cycle is $C_L \times V_{dd}^2$.

The dynamic component of power dissipation is thus directly proportional to the number of logic transitions in the implementation of a particular logic function. This can only be minimized, and never eliminated in static CMOS circuit. Implying that one of the most prevalent higher-level power optimization methods is to try and reduce circuit transition activity. To reduce the dynamic power dissipation in a circuit, is a problem of minimizing the switching function,

$$P_{switching} = \alpha_{0 \rightarrow 1} \times C_L \times V_{dd}^2 \times f_{clk}.$$

Figure 14. Charging up of capacitive load for a 1-0 input transition

Figure 15. Discharging of load transistor for a 0 to 1 input transition

Assuming only the switching power to be non-negligible component, power consumption can be reduced by focusing on one or more of the parameters below. In doing so, design space exploration is performed for circuit level design abstraction.

- Capacitive load (output, and internal nodes)
- Switching activity
- Supply voltage, and
- System Speed

2.3.1.1.1 Device Capacitances

Since, the dynamic performance of the device is directly proportional to the capacitances intrinsic to the device, it is essential to have a good understanding of the capacitive model. The capacitances of a metal oxide semiconductor field effect transistor (MOSFET) originate primarily from three sources:

- the basic MOSFET structure
- the channel charge, and
- the depletion regions of the reverse-biased PN-junctions of the drain and the source.

Figure 16 below illustrates the various components of transistor capacitances. C_{GS} is the Gate to Source Capacitance, C_{GD} is the Gate to Drain Capacitance, C_{SB} is the Source to Bulk Capacitance, C_{GB} is the Gate to Bulk Capacitance, C_{DB} is the Drain to Bulk Capacitance.

There is yet another motivation to reduce the load capacitance. Larger output capacitance implies that the short circuit current flows for a longer duration (Figure 17), thus dissipating

more power. In this research initiative, the device capacitance shall be assumed to be non-alterable for a given cell library technology and feature size. Since, the work done is primarily at the gate level and higher, this shall be one of the fixed parameters for the respective technology under consideration. This is even more so the case considering the fact that most of the development today is cell based.

Figure 16. Parasitic capacitance model of a MOSFET transistor

Figure 17. Effect of Capacitive load on short circuit current

2.3.1.1.2 Switching Activity

Switching activity represents the core functionality of a logic design. It is only by switching nodes that the logic circuits are put to use. Thus, there is no way that one can eliminate switching activity (to zero). However, most logic functions, when implemented in a particular manner, can reduce the switching activity to minimalist levels. This reduces overall power dissipation in the system.

There are two major components to switching activity: a static component and a dynamic component. The static component does not take into account the timing behavior and is strictly a function of the circuit topology and the signal statistics, whereas, the dynamic component takes into account the timing behavior of the circuit. The switching activity of a function is a complex function of several factors including the type of logic function, logic style, circuit topology, input data statistics and the sequencing of operations.

Whilst switching activity represents the core functionality of the circuit, there is usually a lot of power dissipation due to spurious switching in the circuit. This is usually due to different arrival times of data, unbalanced logic trees, etc. This is referred to as the toggle power.

2.3.1.1.3 Threshold Voltage Scaling

Since significant power improvement can be gained through the use of low-threshold MOS (metal oxide semiconductor) devices (Figure 18), the question of how low the thresholds can be reduced must be addressed. The limit of scaling is set by the requirement to retain adequate noise margins whilst trading the increase in sub-threshold currents.

Noise margins can be relaxed in low power designs because of the reduced currents being switched, however, the sub-threshold currents can result in significant static power dissipation. It has been reported that a 100% power penalty is implied in certain cases. Thus, it is not advisable to use true zero threshold devices, but instead to use thresholds of at least 0.2 V for static CMOS and a bit higher for dynamic logic. This provides for at least two orders of magnitude in reduction of sub-threshold current. This provides a good compromise between improvement of current drive at low supply voltage operation and keeping sub-threshold power dissipation to a negligible level. There has been widespread research in the area of reducing sub-threshold conduction so as to allow reduction in supply voltage of the design. However, this issue is very close to the process and technology of transistors and hence, cannot be addressed by most high-level designers.

Figure 18. Compromise between dynamic and leakage power dissipation through V_t variation

2.3.1.1.4 Voltage Scaling

In the expression for dynamic power, the voltage term denotes a quadratic dependency for power consumption. Thus, it seems to be the natural place to concentrate efforts on low power design. However, reducing the supply voltage of a system affects other performance parameters in the design. This is more accurately reflected in the power-delay product.

2.3.1.1.5 Impact of voltage scaling on delay and power-delay product

Energy per transition has a strict quadratic dependency on voltage. This is illustrated in Figure 19 below. Conversely a quadratic improvement in the power-delay product can be achieved by reducing only the supply voltage whilst maintaining the performance parameters.

However, it can be seen from the graph below (Figure 20) that varying circuits show the same trend for varying supply voltages - reducing supply voltage of a design comes at a penalty of increased delay. The three graphs illustrate that V_{dd} , delay, and power-delay product maintains a regular relationship for varying design complexities and logic styles.

Figure 19. Power-Delay product exhibiting square-law dependence for two different circuits

Figure 20. Graph demonstrating that delay characteristics follow a simple first-order theory

Figure 21. Data showing improvement in power-delay product at the cost of speed for various circuit approaches

2.3.1.1.6 Motivation for voltage scaling

Voltage scaling (as reported in [12] & [13]) is performed from a few perspectives. Some of these perspectives are discussed below:

Reliability driven Voltage Scaling

Some amount of voltage scaling comes bundled with technology advancements. Constant-voltage scaling of transistors comes plagued with reliability problems. Devices degrade with time – changes in threshold voltage, degradation of trans-conductance, and increase in sub-threshold currents. These effects can lead to the eventual breakdown of the device. These degradations occur because of the higher electric fields in the channel, which lead to the creation of hot carriers. Solutions to this process were developed along with the reduction of the supply voltage to 2.5V. This voltage supply level provides the minimum delay point in the supply voltage versus delay curve.

Technology driven voltage reduction

As feature sizes shrink to below 1.0 μm, the delay characteristics as a function of lowering the supply voltage deviate from usual expectations. Due to the carrier velocity saturation under high electric fields, the current is no longer a quadratic function of the voltage – but is linear. Hence, the current drive is significantly reduced. This meant that decreasing supply voltages below the

critical voltage does not penalize speed performance. At the same time increasing the voltage above this critical voltage also does not provide any advantage in performance. For the 0.3 μm technology, reducing the voltage to 3.3V provides a 60% reduction in power when compared to 5V (critical voltage was proposed as 2.43V).

To gain performance, lower threshold voltage is used. Lower threshold however, leads to increased leakage power (sub-threshold). Also, higher power densities increase device temperature. This (temperature increase) then increases junction leakage power. As technology scales, oxide thickness is also scaled which causes more leakage power (gate).

Constant voltage scaling and constant electric field scaling

Clearly constant electric field scaling (supply voltage scaling) gives the lower energy-delay product (ignoring the increase in leakage energy) and hence is preferable. However, it requires scaling the threshold voltage as well, which increases the sub-threshold voltage, which increases the sub-threshold leakage current, thus increasing the IC's power dissipation.

2.3.1.2 Static Power

Ideally, CMOS circuits dissipate no static power in steady state. This is because the PMOS pull-up network and the NMOS pull-down network are switched-on mutually exclusively. However, this ideal condition is almost never realizable in practice. Leakage currents, give rise to the static component of power dissipation. The three leakage sources are:

- Sub-threshold leakage
- Junction leakage
- Gate-substrate leakage

This energy consumed in the above parameters is a complete loss, as it performs no useful task in the circuit. The direct-path short-circuit current (I_{sc}), which arises when both the NMOS and CMOS transistors are simultaneously conducting (on), providing a conducting path from supply (V_{dd}) to ground (V_{SS}). This is illustrated in Figure 22

Figure 22. Short-Circuit Dissipation

The leakage current ($I_{leakage}$), which can arise from the substrate injection and sub-threshold effects, is dependent upon the fabrication technology. This leads to the leakage power term in equation 1. A detailed analysis of the above factors is presented in [I].

2.3.2 Different Circuit Styles

Having visited the basics of power consumption at the transistor level, other circuit level options are now discussed. In the sections that follow dynamic circuits, pass transistor logic and asynchronous vs. synchronous circuit design techniques are discussed. Due to the difference in circuit topologies, the circuits have different power dissipation characteristics.

2.3.2.1 Dynamic versus Static Circuits

Dynamic circuits have some inherent advantages such as elimination of short circuit current dissipation, reduced switching activity due to elimination of hazards, and reduced parasitic node capacitances. Dynamic circuits also require less silicon area due to the reduction in number of transistors required to implement the given function. On the other hand the disadvantages include requirement of a global clock distribution to synchronize the pre-charge and evaluation cycles. Static circuits also do not suffer from charge sharing.

Some considerations in deciding between dynamic and static circuits are dealt with in more detail below:

2.3.2.1.1 Spurious Transitions

Static designs can exhibit spurious transitions due to finite propagation delays from one logic block to the next (also called critical races and dynamic hazards). In other words, a logic block can have several unnecessary transitions in a single clock cycle before settling down to the correct logic level. These spurious transitions dissipate extra power that does not contribute

directly to the final result in the logic block. The number of these extra transitions is a function of input patterns, internal state assignment in the logic design, delay skew, and logic depth. As an example, an 8-bit ripple carry adder can dissipate as much as 30% of its total power in spurious transitions for a set of random input patterns. Numerous techniques aid the designer to reduce this energy loss inherent in conventional CMOS circuits by reducing these spurious transitions. However, in the case of dynamic logic, only valid power consuming transitions occur per clock cycle.

2.3.2.1.2 Short-circuit Currents

Short circuit currents in CMOS transistors are minimized by appropriate sizing of NMOS and PMOS transistors for equal rise and fall times. Careful design can reduce the short circuit component to 20% of total power dissipated. Again, dynamic circuits inherently do not have this component of power dissipation, other than in cases of severe clock skew, or when the pull-up devices are used to control charge sharing.

2.3.2.1.3 Parasitic Capacitance

Dynamic logic typically uses fewer transistors to implement a given logic function, which directly reduces the amount of capacitance being switched and this has a direct impact on the power-delay product. However, extra transistors may be required to ensure that charge sharing does not result in incorrect evaluation of logic.

2.3.2.1.4 Power-down modes

Power-down strategies such as clock gating cannot be applied to dynamic circuits. These techniques are more applicable to static circuits. If the logic state were to be preserved at shut down, dynamic circuits would require relatively small amounts of extra circuitry to preserve the state. However, this can result in higher parasitic capacitance and slower speed.

Power down modes should be applied with careful evaluation. In certain applications the latency involved in waking the circuit is not permissible. The circuit should be able to return to functionality as soon as possible. Further, the area and thus the capacitance introduced in the circuit should be a comparable tradeoff for the design. Strategies as to when the device should go into power down mode should also be thought through, as spurious power down signaling might only lead to an immediate wake-up signal thereafter.

This is one instance where a good coupling between the compiler, the instruction set and the architecture can lead to greater power savings and less occurrences of spurious power down of

circuits. This technique can be further extended for use by Real Time Operating Systems (RTOS) to schedule shutdown and wake-up signaling of circuits, keeping in mind the latency parameter.

2.3.2.2 Conventional CMOS Static gate versus Pass gate Logic:

Complementary Pass Transistor Logic (CPL) designs come out a clear winner when it comes to area required for implementation of logic functions. Directly associated with area, the total capacitance of the circuit is also lower than the more area consuming conventional complementary CMOS logic [2].

Figure 23. Comparison of a conventional CMOS and CPL adder

The full adder implementation schematic in CPL and static CMOS is shown in Figure 23. Amongst the various logic functions, XOR gate design is the most striking in CPL implementations where only 2 transistors are required. Thus, CPL implementation can lead to significant savings in arithmetic circuits where XOR is the key. However, a CPL implementation as shown in the Figure 23 has two basic problems. First the threshold drop across the single channel pass transistors results in reduced current drive and hence slower operation at reduced supply voltages; this is important for a low power design since it is desirable to operate the lowest possible voltage levels. Second, since the “HIGH” input voltage level at the regenerative inverter is not V_{dd} , the PMOS device in the inverter is not fully turned off and hence direct-path static power dissipation could be significant. To solve these problems, reduction of the threshold voltage has proven effective, although if taken too far will incur a cost in dissipation due to sub threshold leakage and reduced noise margins. The power dissipation for a pass-gate family adder with zero-threshold pass transistors at a supply voltage of 4V was

reported to be 30% lower than a conventional static design, with the difference being even more significant at lower supply voltages.

2.3.3 Transistor Sizing

Transistor sizing in a combinational gate circuit can have significant impact on circuit delay and power dissipation (Figure 24). If the transistors in a given gate are increased in size, then the delay of the gate decreases, however, power dissipated in the gate increases. Further, the delay of the fan-in gates increases because of increased load capacitance.

Given a delay constraint, finding an appropriate sizing of transistors that minimizes power dissipation is a computationally difficult problem. A typical approach to the problem is to compute the slack at each gate in the circuit, where the slack of a gate corresponds to how much the gate can be slowed down without affecting the critical delay of the circuit. Sub-circuits with slacks greater than zero are processed, and the sizes of the transistors reduced until the slack becomes zero, or the transistors are all minimum size. Variants of the above approach are presented in [3] and [5].

Figure 24. Tradeoffs in Transistor Sizing

2.3.4 In Summary

In this section the basics of power consumption in transistors and related circuit design paradigms is discussed. It is also observed that even though circuit design is firmly grounded in the technology and physics of transistors, they still provide flexibility to meet different design requirements. Needless to say, innovation at this basic level can lead to higher optimization gains. However, in the dense semiconductor designs of today, there is a large amount of legacy circuits which form the basis of semi-custom integrated circuit design (example: cell based ASIC flows). Such designs depend on cell based libraries which adhere to design rules for fabrication success. Thus the optimization task migrates to a higher level of design hierarchy.

2.4 Gate level

Having looked at the circuit level and the various circuit topologies possible, we now move up in the design hierarchy to elaborate on the effect of gate-level design on design space exploration. As the gate-level blocks are made of circuit level components, there exists a strong correlation between the parameters involved in optimization for power aware designs. For example, one of the main thrusts in this design hierarchy is to minimize functional switching, as well as to retime circuits to minimize spurious switching and glitches.

In the design of complex gates, for example, $f = \text{not}((a+b).c)$, choices regarding placement of individual transistors can be made, such that power consumption is minimized. For example, in the N part of the CMOS gate implementing the above function f , the parallel transistor pair $(a+b)$, can be connected to the gate output or the transistor driven by c can be connected to the gate output

Also, late arriving signals can be placed closer to the output node to minimize gate delay. However, the average power dissipated is dependent on the transition probabilities of the gate inputs and the internal node capacitances. (Internal node capacitance is due to parasitic drain, source capacitance and interconnects capacitances).

Ordering of gate inputs will affect both power and delay. In [3] and [4], methods to optimize the power and/or delay of logic-gates based on transistor re-ordering is explained. Moderate improvements can be achieved by judicious re-ordering of transistors in the transistor network.

2.4.1 Logic Optimization Techniques

Both, switching activity and capacitive loading can be optimized during logic synthesis. It therefore has more potential for reducing the power dissipation than physical design. On the other hand, less information is available during logic synthesis, and hence, factors such as slew rates, short circuit currents, etc. cannot be captured properly.

2.4.1.1 Retiming

Retiming [6] is the process of repositioning flip-flops (FFs), originally proposed for minimizing the clock periods of digital circuits. In [7] it is noted that the FF output may make at most one transition when the clock is asserted. Based on this observation, the authors then describe a circuit retiming technique targeting low power dissipation. This technique does not produce the optimal retiming solution as the retiming of a single node can dramatically change the switching activity in a circuit and it is very difficult to predict what this change will be.

The technique heuristically selects a set of nodes with the property that if FFs are placed at their outputs, switching activity in the network will be reduced. Nodes are selected based on the amount of glitch activity that is observed at their outputs and the probability that this glitch activity propagates through their transitive fan-outs.

2.4.1.2 Pre-computation Logic

A sequential logic optimization method is described in [2] that is based on selectively pre-computing the output logic values of the circuits one clock cycle before they are required and using the pre-computed values to reduce internal switching activity in the succeeding clock cycle. They present an automatic method of synthesizing pre-computation logic so as to achieve maximal reductions in power dissipation. Up to 62% reduction in average switching activity and power dissipation are reported with marginal increases in circuit area and delay.

2.4.1.3 State Assignment

In the past, many researchers have addressed the encoding problem for minimum area of two level or multi-level logic implementations. A state assignment procedure presented in [8] and [9] minimizes the switching activity on the present state input lines. This problem is equivalent to embedding a weighted graph on a hypercube of minimum (or given) dimensionality, which is an NP-hard, problem. Simulated annealing was used to solve this problem in [8]. Nevertheless state assignments [9] using minimum distance rules and Gray encoding do perform respectably in reasonably large state machines.

2.4.1.4 Path Balancing

Balancing path delays reduces hazards/glitches in the circuit, which in turn reduces the average power dissipation in the circuit. This can be achieved before technology mapping by selective collapsing and logic decomposition or after technology mapping by delay insertion and pin reordering.

The rationale behind selective collapsing is that by collapsing the fan-ins of a node into the node, the arrival time at the output of the node can be increased. This is however valid only if the node delay is determined by assuming an AND-OR implementation with the delay of AND and OR gates being a function of the number of inputs to the gate. Logic decomposition can be performed so as to minimize the level difference between the inputs of nodes that are driving high capacitive nodes. The key issue in delay insertion is to use the minimum number of delay elements to achieve the maximum reduction in spurious switching activity. This is a difficult

task as delay insertion at some nodes will not only change the spurious activity at the immediate output of the gate, but will also affect the spurious activity in the transitive fan-out of the node.

Path delays may sometimes be balanced by appropriate signal to pin assignment. This is possible as the delay characteristics of CMOS gates vary as a function of the input pin that is causing a transition at the output.

2.4.1.5 Clock gating and shutdown of functional blocks

Synchronous designs are plagued with the presence of unwanted signal transitions at all active clock edges. In many cases, the logic switches even though the computation is to remain the same as last cycle. Thus clock gating presents an avenue to reduce unwanted power dissipation.

However, with overaggressive clock gating, logic failures can arise, where a block should have been awake, but either did not turn on quickly enough, or is turned off too soon. On the other hand, conservative clock gating will not disrupt correct functionality, but will result in wasted power. Moreover, most gate level simulators do not report such errors.

Sharp current swings associated with gating causes di/dt noise. This noise is also referred to as the simultaneous switching noise and is characterized by the equation $dV = L di/dt$, where dV is the voltage fluctuation, L is the inductance of the wire supplying the current and dt is the rise/fall time of the gates.

This estimate of the switching power ($P_{switching} = \alpha_{0 \rightarrow 1} \times C_L \times V_{dd}^2 \times f_{clk}$) ignores the internal nodes of a CMOS gate, which have diffusion capacitance to the substrate. These capacitances could be dissipating power due to internal switching. This component is toggle power.

2.4.2 In Summary

Circuit level too has limited leverage in providing optimization opportunities as much of the basic libraries are all available as standard in a semi-custom cell based integrated circuit design flow. This is more so to manage the design complexity and density of large designs that is commonplace today. As illustrated in the sections that follow, the next few layers moving up in the design hierarchy will provide opportunities for design optimization.

2.5 Architectural approaches

Having looked at the basic blocks of digital logic design for low power approach to digital design, we now look at the next level of abstraction. The block level arrangement of digital blocks, that is, the architecture can be tweaked to provide a design that meets the design

specifications of power consumption without compromising on the performance requirements. However as is the nature of tradeoff, the penalty for this approach is in the silicon real-estate.

Using parallel hardware units that perform the same function can de-couple the dependency on clock frequency for raw processing power. Where possible, more hardware area can be used to significantly reduce the local clock frequency of the circuit and reduce power consumption. However, area increases due to the parallelism introduced should not increase beyond limits imposed by interconnect and the primary gain. Section 2.5.1 describes a design space exploration example which utilizes parallel and pipelining transformations exercised with different voltage and clock frequency options. This example is drawn from [2].

2.5.1 A data-path example of architecture driven optimization

Shown in the Figure 25 below is a simple function of adding A and B and comparing it with C. The data path is 8-bit wide and the design was implemented in 2.0 μm technology at 5V. The power consumption for this solution is the reference power = $P_{ref} = C_{ref} \times V_{ref}^2 \times f_{ref} \times C_{ref}$ is the total effective capacitance being switched per clock cycle.

Figure 25. A simple data path with the corresponding layout [2]

With the intent of reducing the power consumed, it was seen that the use of parallel adders (as shown in Figure 26) led to reduction of frequency to $f_{ref}/2$.

Figure 26. Parallel implementation of the simple data path [2]

This led to the now possible reduction of the supply voltage to 2.9V. As a result of the increased area and circuit, the capacitance now increases by 2.15 times. However, it is interesting to note that the power consumption actually reduced to 36% of reference power consumption.

$$P_{par} = C_{par} \times V_{par}^2 \times f_{par} = 2.15 \times C_{ref} (0.58 V_{ref})^2 f_{ref}/2 \sim 0.36 P_{ref}$$

This method of reducing power by using parallelism has the overhead of increased area, and would not be suitable for area-constrained designs.

Overheads of Parallel architecture – In general, the parallelism will have the overhead of extra routing (and hence extra power), and careful optimizations must be performed to minimize this overhead. This provides a niche area for development of partition techniques for minimal overhead. Interconnect capacitance also plays a very important role in deep-submicron implementations. This is because capacitance of interconnect ($C_{wiring} = C_{area} + C_{fringing} + C_{wiring}$) can become dominant part of total capacitance and cease to scale. i.e., equal to $C_{gate} + C_{junction} + C_{wiring}$.

This is indeed the problem of nanotechnology as interconnect does not scale in step with gate. Current research can also be aimed towards reducing this problem by proposing architectures that minimize on routing requirements, or subsidize the problem by better scaling of interconnect. Another high performance approach to quell the high power dissipation is to pipeline the architecture in Figure 26. This leads to a 15% increase in capacitance and allows us to reduce the voltage to 2.9V. Thus once again a high savings of 61% was reported.

$$P_{pipe} = C_{pipe} \times V_{pipe}^2 \times f_{pipe} = 1.15 \times C_{ref} \times (0.58 V_{ref})^2 \times f_{ref} \sim 0.39 P_{ref}$$

Figure 27. Pipelined implementation of the simple data-path [2]

Due to the introduction of latches, the logic depth was reduced, which led to reduced power wastage due to hazards and glitches. Furthermore, when a combination of parallel and pipelining techniques was used on the same example – a power savings of 80% is achieved.

$$P_{parpipe} = C_{parpipe} \times V_{parpipe}^2 \times f_{parpipe} = 2.5 C_{ref} (0.4 V_{ref})^2 f_{ref} / 2 \sim 0.2 P_{ref}$$

From the above discussion it is quite apparent that time-multiplexed approach to solving a problem is the most inefficient from the power perspective. This technique is generally called trading area for power.

The disadvantage of power scaling at a system level is the proliferation of different devices running at different supply voltages. This can lead to multiple power supplies in a system, thus complicating matters and probably wasting power (if efficient DC-DC converters are not used) in conversion between different voltage levels.

In contrast with common sense, high performance techniques have actually been used to reduce energy consumption of a given task maintaining the same performance throughput. The details of this low power design approach can be found in [2]. This example is used to test the efficacy of SLDL and HDL design flows in chapter Chapter 4.

2.5.2 Impact of micro-architecture

Micro-architectures are one of the chief constituents of the new microprocessor generation that include the likes of predictive micro-pipelines and microcode instruction sequences dedicated to increasing performance of the new generation microprocessors and reducing branch penalties associated with missed cache hits for data. However, this comes with a severe penalty of power. Since, the number of instructions to be processed per clock cycle now increases for a given task – this leads to increased power consumption. This implies the need for large current drivers. Having larger current drivers on the chip can lead to thermal and noise reliability problems. Thus, low power design catering to micro-architectures is also required.

The predicted growth of power dissipation in microprocessors is anticipated to reach the power density of a nuclear reactor. This is actually conceivable as can be seen from the hot-plate connotation of the Pentium Pro Processor in Figure 28.

Figure 28. Power Density Evolution

2.5.3 Motivation for Memory and Bus related energy reductions

On-chip memory constitutes one of the most important factors in ASSP or ASIC design. However, due to their VLSI structure, they tend to contribute largely to the power budget of the chip. With the widespread use of multimedia in most handheld devices, ASIC capable of providing such processing requirements are increasing in demand. Memory bandwidth is the typical requirement for such devices. Conventional off-chip memory based architecture imposes higher latency, external memory interface, and caching structures for faster accesses. This adds to the complexity of the design, and also increases the power consumption requirement of the design.

Thus, integration of memory on-chip on ASICs and ASSPs has seen a wider acceptance. This has led to memory access patterns, memory partitioning and memory design playing a major part in low-power design of these specialized ICs. Partitioning memories into two separate blocks as well as increasing the line size of a single access has shown significant power consumption reductions. Further, static power consumption of memories has been a concern which is addressed by better cell technologies.

Memory accesses contribute a significant portion to power dissipation. Especially in DSP applications where memory accesses occur quite rampantly for sample values and coefficients. Thus, studying memory structures or memory access techniques can help significantly in reducing consumed power.

Traditionally power optimization methods have been applied to data-path components. However, with the widespread design of multi-function ASIC's, global chip communications busses and accesses to memory by data intensive applications (video, speech processing) have begun hogging the power budget. This implies that with proper design, reduction in the memory related energy budget and the proper planning of on-chip communication busses shall lead to lower power consumption. Some techniques are listed in [14]& [15]

Memory access address sequencing ([16], [17] & [18]) can lead to lower power consumption figures. As an example, by noting the present value on the bus, if the next value can be manipulated to reduce the number of logic transitions, then energy can be saved. This is the central theme of most energy optimization transformation related to bus activity. Considerable amount of work has also been done to reduce the amount of energy consumed by the interconnect itself.

Bulk of power optimization work is done on data operators and data paths. Significant power savings can be obtained when address decoders and address buses are taken into the optimization steps. For example, sign magnitude number representation is certainly more suitable for use in bus transactions as the switching capacitance involved is much smaller for change in signs for the data values. The increase in switching capacitance due to the conversion circuit is much smaller than the switching capacitance that is eliminated in the process. Other schemes include Gray code and conditional inversion coding, etc.

2.5.4 Instruction set pipeline

Significant power optimization can be achieved by applying appropriate power optimal transformations to instruction set. For example, gray coded instruction set can result in higher savings. Such instruction sets can lead to a single bit change in instruction registers due to the coding used. The coding however can be difficult to implement keeping in mind temporal relationship between instructions. Thus, instruction-processing statistics can help in such calculations. Coupled with data-path modifications for most frequently used operations can result in high savings. Reducing capacitance switched in frequently used pipeline paths can also result in high savings.

2.5.5 In Summary

As is obvious, most of the techniques demonstrated in this section are quite applicable in the design flows that are currently in use today. However, the trade-offs in terms of area, power and performance of the tweaks in the digital logic design have to be analyzed as early in the design

phase as possible. This is sometimes not possible as design tools are not always that mature at the design entry phase to provide the architect with accurate analysis to make a decision at a system level. This aspect will be addressed as a part of this research study.

2.6 Algorithm Level

When formulating new algorithms for various application specific roles, researchers tend to only look at perfecting the performance from the viewpoint of accuracy. These algorithms assume ideal case situations, i.e., floating point representation of numbers, large bit-width operators, abundance of computational facilities, and no accuracy errors due to truncation etc. However, due to the physical limits applicable in implementation – floating-point format is usually avoided and arithmetic operators are usually applied to smaller bit-width operators. Such limitations make the original algorithm to deviate from its original accuracy. However, in most applications, accuracy is usually an accepted penalty for lower power and area requirements. In this section we shall look at some of the methods generally used to reduce power at the algorithmic level. Clearly, the designers have to keep a close watch on the accuracy of the design resulting from such optimizations. Thus, power and accuracy tradeoffs form a more important part in algorithmic optimizations. Area and performance estimation follows soon after the algorithm achieves a certain minimal level of accuracy as a performance benchmark.

2.6.1 Design approaches at higher levels of design abstraction

Compilers usually address the performance issue of an algorithm. However, with numerous techniques now becoming available for behavioral transformations aiding energy reductions – compilers are slowly but surely gaining ground in providing power minimization options. Compilers can now perform operator strength reduction, and other similar operations to intelligently reduce power consumption by making the task, energy efficient.

The problem of power consumption can be tackled at the algorithm level by using more efficient algorithms. Such algorithms can be alternatives to present techniques, or can be obtained by performing energy efficient transformations to existing algorithms. These transformations can be carried out by applying some or all of the following steps where relevant:

- Remove redundant operations
- Use weaker operators such as additions over multiplications and divisions where possible. Further a more efficient algorithm can also provide a leap in power savings.

- Use energy efficient number systems where possible. These include fixed point numbers over floating point numbers, LNS and RNS over 2's complement representation.
- Reducing the precision of the data representation also allows for power efficient algorithms

Simulations leading to the necessary performance measures required can prevent the designers from delivering a solution with too much of safety band. This means the architecture shall have slack time and shall still be incurring higher power consumption figures. Thus, delivering *just* the required amount of performance can keep the power budget under check.

2.6.2 Impact of number systems

2's complement number system is used in most designs because of the well characterized operations that can be implemented directly using the synthetic libraries available for synthesis. However, one of the most problematic issues with this number representation is the switching of all the most significant bits (MSB) of the register whenever there is a change in sign. This is further compounded by the fact that most of the dynamic range remains unused in most applications. This problem is somewhat attenuated by the use of sign-magnitude number representation. Figure 29 shows the transition probabilities of the various bit positions in a 2's complement and sign-magnitude number representation.

Figure 29. Reducing switching activity by optimizing number representation

$\sigma=0$, implies that the data is uncorrelated, and therefore the transition probability is $\frac{1}{2}$. That is, 50% chance that the output is going to change between positive and negative. A large positive correlation coefficient implies that the signal changes very slowly and therefore switches sign very infrequently. Similarly a negative correlation coefficient with a large value implies that the signal changes sign infrequently.

In sign-magnitude representation, the higher order bits have a low transition activity and bit 15 (sign bit) has the same transition probability as the 2's complement's bit 15. This is because bit

15 in both number systems serve as the sign bit. The higher order bits of the 2's complement number system also exhibit a relatively higher activity level as against that of sign-magnitude as the 2's complement number system suffers from the sign-extension phenomenon for negative numbers.

The reduction in the number of transitions for sign-magnitude over two's complement is a function of both the dynamic range of the signal and the signal correlation coefficient. Figure 30 shows the ratio of the number of transitions required in the two's complement representation to that required in the sign-magnitude representation as a function of the signal correlation and normalized dynamic range.

Figure 30. Activity reduction using sign-magnitude number representation

Going up the hierarchy, it is seen that high-level decisions made to reduce power dissipation have a higher impact than lower level optimizations. Thus, when it comes to algorithmic optimizations, it is imperative to choose the number system. It has been seen that inherent properties in various number systems cater for lower power dissipation in the silicon perspective. Also, in an algorithm or specific architecture one must apply the optimizations to the most frequently occurring computation.¹

2.6.3 LNS and RNS number representations for low power

Use of logarithmic number systems (LNS) in algorithms for ASIC or ASSP (application specific standard processor) architectures can lead to substantial power savings, as it reduces bit activity as well as reduces the complexity of certain arithmetic operations. Both of these factors directly contribute to lower power dissipation figures.

The LNS representation maps a real number, X , to a triplet as:

X to LNS Transformation-> (z,s,x = log_b|X|),

Where b is the base of the logarithm, z is the zero flag, and s is the sign of X . A zero flag is required because $\log_b|X|$ is not finite at $X=0$. Similarly, since the logarithm of a negative number is not a real number, the sign information of X is separately stored in the sign bit s . *Logarithm* $x = \log_b|X|$ is encoded as a binary number (2's complement or sign-magnitude), expressed as $x = I.F$, where I is the integral and F is the fractional part. LNS has been considered as an extreme case of floating point arithmetic [19].

Not only does a DSP application use multiply accumulate (MAC) operations, but increasingly, many emerging embedded devices used for portable applications are also processing audio and speech data as a routine chore. Thus, the MAC functional block was used as a comparison between linear and LNS implementations with a low power perspective.

n (bits)	LNS (mW/MHz)				Fixed-point Multiplication
	Base b	Addition	Subtraction	Average	
8	1.5	0.99	0.99	0.99	2.02
	2	0.90	0.89	0.89	
	2.5	0.98	0.98	0.98	
10	1.5	1.14	1.16	1.15	2.64
	2	1.18	1.18	1.18	
	2.5	1.09	1.11	1.10	
12	1.5	1.49	1.52	1.50	3.35
	2	1.58	1.58	1.58	
	2.5	1.46	1.49	1.48	
14	1.5	2.38	2.46	2.42	4.13
	2	2.62	2.61	2.62	
	2.5	2.26	2.36	2.31	

Table 1. Estimation of Power Dissipation in LNS subtraction-addition, in comparison to $n \times n$ bit fixed-point multiplication, for various fixed-point word lengths, n , and bases b

From the hardware architecture viewpoint, the LNS multiplier is both functionally and structurally identical to a fixed-point adder. Hence, the problem of comparing the amount of power dissipated by a combined multiplication-addition operation is transformed to comparing the power dissipation figures of LNS add/subtract operations and a corresponding fixed number multiplier.

¹ This analysis is similar to Amdahl's law governing the speedup of using parallel processors on a problem, space vs. using only one serial processor.

The power dissipation measures in Table 1 refer to 0.7 μm CMOS technology, operating at 5V **Error! Reference source not found..** The power dissipation figures are obtained by adding the components of an LNS addition/subtraction:

$$P_{total} = P_{comp} + P_{address} + P_{LUT} + P_{add}$$

Where, P_{comp} is the power dissipated at an initial comparison to determine the larger of the two operands, $P_{address}$ is the power dissipated for address generation, P_{LUT} is the power dissipated in the Look-Up Table (LUT) access, and P_{add} is the power dissipated for the final addition. Note that an LNS addition is not implemented using a single LUT, but by using a multitude of smaller LUTs as depicted below in Figure 31.

Figure 31. Organization of an LNS adder/subtractor memory, composed of N small LUTs

The breakdown of Look-Up Tables required to store the addition/subtraction function values is motivated by the shape of the addition and the subtraction function. For a more in-depth view of LNS LUT generation refer to **Error! Reference source not found..**

2.7 **System Design**

A hierarchical approach to design methodology is inevitable with the higher densities in logic design. As a result, system level optimizations have to be addressed to achieve a closure. System design precedes architecture design and probably goes hand-in-hand with algorithm design for low power.

2.7.1 **Efficient bus architectures**

As with most power consuming solutions, optimization can be applied either at the physical design level, or at a higher logic design level. Due to the disproportionate scaling of logic and interconnect in sub-micron designs, the proportion of power dissipation by interconnect has grown. Thus, a greater portion of power dissipation is now utilized by interconnect, rather than by the functional gates. This has led to the development of several techniques for low power bus

architectures at the physical design level. Thus facilitating low power design by providing cell-based developers with data-path, and interconnect libraries, which are power-optimal. Reduced voltage swing based receiver and driver circuits is one of the methods for low power interconnect designs. Charge recycling procedures are also used to reduce the wastage of energy in bus architectures. Use of double edge triggered flip flops seems a viable option of reducing power consumption as it implies halving the clock frequency.

2.7.2 Instruction set design

In programmable processors, the flow and sequencing of instructions dictates the power consumption. This is not unusual as the instructions decide the switching of the circuit in the programmable device. In a typical processor, an instruction may be fetched from the instruction cache, or from the instruction memory (off-chip or on-chip). The instruction is then decoded to further fetch more data from data cache (on-chip) or from the data-memory (on- or off-chip). The decoded instruction also provides information regarding the operation to be performed in the processor. The output is then loaded into the data-memory (on or off-chip).

In present programmable processor designs, the instruction set is designed flexibly with respect to the micro-code instructions that it drives. However, no other considerations are taken into account when the instruction set is designed. The only power-optimal consideration is Gray coding the instruction set codes so that minimal transitions take place in the execution process.

The instruction set should be designed with a number of power-optimal characteristics being taken into account. Further, an iterative development scheme with to-and-fro exchange of information between the design architects for the IC and the compiler designers is also proposed. By this process, the micro-architecture of the processor can be fine-tuned to accept the software programmer's inputs to perform power management operations.

Since power-shutdown is a major strategy in programmable processors, they should be fine-tuned to be dictated (also) by the instruction set. This is in stark contrast to logic methods used to detect non-activity in modules. By embedding "hints" for power-shutdown in the instruction set, less logic area is spent in acquiring a more flexible and optimal power-shut down monitor.

Additions to the instruction set can also be made so as to make the caching mechanism more intelligent and pre-informed about future instruction and data fetches. Further, the compiler can be more power aware by in-lining code where the calls to functions are not repetitive. Loops can also be unrolled where the iterations are not too many.

The above modifications imply a more intelligent compiler to make the required decisions and interleave the power optimal instructions in the code of the programmer. It is important to note that the semantics of the program remain unchanged due to the modifications. Since, the changes do not warrant a different outlook by the programmers, the schemes above are envisaged to be feasible.

2.7.3 Multiple Supply Voltage Architectures

Not all functional units in a design require the same throughput; hence, multiple supply voltages can be used to control power consumption. While this multiple supply voltage concept can be applied in a static manner, it is also pursued dynamically in most modern processors. In dynamic supply voltage scaling, the logic chip is designed to deliver maximum performance at the highest supply voltage. When the performance demand is low, the chip is operated at lower voltage, delivering lower performance but with substantial (quadratic) reduction in power. As an example, a laptop PC can be “speed-stepped” to run at lower power (voltage) when on battery and higher performance (voltage) when running on a docking station.

The logic chip could also detect the performance demand and adjust frequency and supply voltages accordingly [29]. However, it is important to note that this is not a trivial task and requires significant investment in area and design time.

Figure 32, below shows a static implementation of dual voltage supply. Techniques to arrive at such designs are shown in [23] and [24]. Higher supply voltage is used for performance critical logic, and consumes higher power.

Figure 32. Logic with multiple supply voltages

However, such implementations need careful evaluation. When a logic signal that emerges from a slow block is connected to a fast block, the signal levels are closer to the threshold voltages of the transistors, and could consume excessive leakage power and reduce noise margin. This scheme also requires additional power supply grid and associated support such as decoupling capacitors, to ensure error free operation.

As mentioned earlier, reducing supply voltages results in the exponential increase of subthreshold leakage current. Large leakage currents can severely degrade noise immunity of dynamic logic circuits, and compromise stability of SRAM cells, and increase leakage power consumption of the chip to an unacceptably large value.

Note in the graph below that almost $\frac{1}{2}$ of the chip's total power consumption is attributed to leakage power. Thus, dual voltage logic provides a mix of the advantages and disadvantages of the tradeoffs. High threshold transistors yield slower logic, but lower leakage, and low threshold voltage transistors yield faster logic, but higher ($\sim 10X$) leakage.

Figure 33. Sub-threshold Leakage as a function of Feature Size

Figure 34. Dual Voltage Design Methodology

Figure 33 and Figure 34 show the tradeoff between using high threshold voltage transistors everywhere as against a selective inclusion of low threshold transistors in the design.

2.7.4 Partitioning the design based on latency requirements

The fact that different parts of the circuit operate at different speeds can be used to partition the design into two portions running at two different power rails. In this manner, there is only one level-converter block required to interface to the different logic blocks. This is used in the clustered voltage scaling method of [22]. However, this method is unadaptive of the changes in requirements of the circuit. Hence, the variable supply voltage scheme [21] is more attractive. In this approach, a digital control to the supply voltage of the circuit is described which adjusts the supply voltage according to the critical circuit requirement.

2.7.5 Supply gating

Supply gating [21] as shown in Figure 35, is yet another technique to reduce active and standby subthreshold leakage power. This is similar to clock gating. However, in this case the power supply is cut off to the logic block. There are numerous issues such as performance degradation

due to the series transistor in the pull-up and pull-down networks, reduction in noise immunity, local power grid structure and careful design.

Figure 35. Supply Gating or Sleep Transistor Technique

2.7.6 Power-down Strategies

In synchronous designs, the logic between registers is continuously computing every clock cycle based on its new inputs. To reduce the power consumption in synchronous designs, it is important to minimize switching activity by powering down execution units when they are not performing “useful” operations. This is an important concern since logic modules can be switching and consuming power even when they are not being actively utilized.

However, the designs of synchronous circuits that can identify unused modules that need to be shut down require extra design effort. Sometimes incorporation of power down strategies at circuit level can lead to more power being dissipated. The need to balance the overhead of shut down detection and implementation circuits with the overhead introduced due to the extra logic circuits requires careful calculation. Thus, it is imperative to address the need to apply these techniques to functional modules.

Self-timed circuits provide relief from unnecessary transitions in synchronous circuits. However, they require additional circuitry for completion signal. This can add to the overhead and might in some cases outweigh the advantage of the method. There are several circuit approaches to produce the completion signal. One method is to use dual-rail coding, which is implicit in certain families such as DCVSL. However, these circuits consume twice the energy for a single output transition as compared to the conventional static CMOS. Thus, it is not advisable to use self-timed circuits in data-paths with high throughput.

The implementation of low power strategies in general purpose platforms is most popularly being addressed by power down strategies and limited amount of voltage scaling. However, this is heralded by the problem of isolating inactive periods in the functional units of the processor.

Further, in most real-time systems reaction time is largely affected by idle, low-power and shutdown modes. In applications where an operating system (OS) or a real-time operating system (RTOS) is available, these responsibilities are taken over by the application layer. However, this requires significant glue logic between the logic modules and the application layers.

However, data flow applications such as DSP cannot have shutdown as an option for power efficiency. Such applications can benefit mostly from better circuit strategies, parallel computation modules, memory partitioning, voltage-scaling, and pipelined modules.

2.7.7 Modulating Clock frequency for low power

Since clock signal in a circuit consumes most of the switching power, one of the methods available is to use lower slower clocks in domains where throughput required is lower.

A method whereby the clock in the circuit is variable [60] along with a variable voltage supply that is directly proportional to the clock in the circuit can be used. This way the processor speed and voltage supply are controlled according to the requirements of the task at a particular time. This can be used to create slacks and clear them when required (due to the variable clock). Further, the variable slack can be used to control the output of the dynamic supply voltage generator. This method also eliminates the need for additional logic in the data-path to detect the need to either change the clock frequency or alter the supply rails. Further, the instructions can be embedded in the code, which can be picked up by the on-chip clock and voltage scheduler.

In a timing driven mode, the scheduler can increase the clocks and supply rails to reduce latency of obtaining the outputs. Alternatively, when the processor is in a power driven mode, it can slow down the clocks and reduce the supply voltage to obtain the desired results.

This works well with the concept of an on-chip scheduler, which aids the higher application layer to schedule tasks. Extending this concept to a RTOS, the application layer can now request the hardware (through drivers) to alter the clock and supply voltage to increase or reduce the latency of a critical path, as well as the power consumption.

2.7.8 In Summary

Most of the techniques in this section are applicable to custom designs that involve novel intellectual properties in the intensely multi-media and multi-modal embedded systems products and applications which are futuristically pitched to the customer at any one given point in time. Thus, when researchers come up with novel algorithms to approach known problems, they have

to then perform an exercise involving the translation of the algorithm to a computing architecture to prove its efficacy as an embedded solution. Thus, design optimization techniques, especially from the context of low power has to be addressed at this stage, as these embedded cores become highly re-used components of more complex designs. Such concerns are addressed in this research study.

2.8 Interconnect – The new digital divide

Up until this section, we discussed design exploration primarily from a logic point of view. This is because of the contemporary integrated circuit architectures developed in the logic-centric era where interconnect and implications of interconnect was trivial. However, as more and more dense circuits are populating the work bench, designers are being forced to enter an era, where interconnect limits design boundaries.

The method in section 2.5.1 earlier has been successfully proven to provide significant gains in power consumption reduction at 0.35 micron and above technology. Power saving effect shall not be as powerful in the newer technologies such as 0.13 micron. This is because of the disproportionate scaling of interconnect with smaller feature sizes in the newer technologies.

Partitioning of combinational modules as well as the memory modules leads to increase in interconnect requirements due to the introduction of inter-module communication overheads. Interconnect represents a large portion of silicon real estate in present chips, as well as power consumption. The increase in the portion of the power consumption pie for interconnect is primarily due to the inability of the interconnect wires to scale with the rest of the circuit/ device parameters as the feature size is reduced.

2.8.1 Technology limited constraints

Wire vs. gate delays seems to be the most divergent factor as we are progressing towards smaller and smaller feature sized transistors. As shown in the Figure 36 [Source: NTRS 1997], Interconnect delays have increased as compared to gate delays. This can be primarily attributed to the high circuit density of IC's in design today. Due to the packaging of several functional modules on one chip, the designers are left with no choice but to route data and control signals throughout the chip. Moreover, the fact that most modules are image, speech or communications oriented, leads to 16, 32 or sometimes even 64-bit busses running the silicon area.

Figure 36. Scaling of delays associated with gate and interconnect across generations

Figure 37. Interconnect Capacitances

As the feature sizes of the transistor reduces, the wire width is also forced to become narrower. However, as the current carrying capacity of signal wires is directly proportional to the cross sectional area, the current carrying capacity reduces as we decrease the width of the wire (keeping the height the same). This cause current drive problems on Integrated Circuits. Hence, to circumvent the problem, the height of the wire has to be increased to suffice the current carrying needs of the wire. This leads to increased cross coupling and capacitance between adjacent wires (shown in Figure 37)

Synchronous communication being the most sought after communication mode, global clocks are made to run the length and breadth of the silicon area. This contributes to long wire lengths, and more capacitive load to be driven by the clock buffers. Thus, interconnect actually emerges as the main culprit for longer delays and higher power consumption in the sub-micron designs.

2.8.2 Time of flight constraint

As die sizes and megahertz rates for processors increase, the signals get even lesser time to travel between two parts of a chip. It is has been reported that a 750 square mm die cannot support a global clock frequency of more than 5.48 GHz (using Manhattan routing).

Thus, presenting us with one of the following options:

- Reduce global clock frequency and use local fast clocks
- Use synchronous methods for local and asynchronous methods for global communication

-
- Maintain functional dependency criteria for module layout. This will help reduce the wire lengths for signal communication between various sub-modules of a chip.

2.8.3 Physical level: Layout optimization techniques

Physical design fits between the netlist of gates specification and the geometric (mask) representation known as the layout. It provides the automatic layout of circuits minimizing some objective function subject to given constraints. Depending on the target design style (full-custom, standard-cell, gate arrays, FPGAs), the packaging technology (printed circuit boards, multi-chip modules, wafer scale integration) and the objective function (area, delay, power, reliability), various optimization techniques are used to partition, place, resize and route gates.

Under a zero-delay model, the switching activity of gates remains unchanged during layout optimization, and hence, the only way to reduce power dissipation is to decrease the load on high switching activity gates by netlist partitioning and gate placement, gate and wire sizing, transistor reordering, and routing. Simultaneously, if a real-delay model is used; various layout optimization operations influence the hazard activity in the circuit.

2.8.3.1 Circuit Partitioning

Netlist partitioning is the key in breaking a complex design into pieces that are subsequently optimized and implemented as separate blocks. In general, the off-block capacitances are much higher than the on-block capacitances (one to two orders of magnitude). It is therefore essential to develop partitioning schemes that keep the high switching activity nets within the same block as possible. Techniques based on local neighborhood search have been adapted to do this. In particular, it is adequate to assign net weights based on the switching activity values of the driver gates and then to find a minimum cost partitioning solution.

2.8.3.2 Floor-planning

Floor planning plays an important role during layout optimization as it determines the interface characteristics (shape, size, I/O locations) and positions of custom or semi-custom blocks in a hierarchical design environment. Reference [27], describes a floor-planner that considers power management. The idea is to generate a set of power-indexed shape functions and then use an implementation for each flexible module that satisfies the timing constraints while minimizing the dynamic power dissipation. In addition, this work considers constraints to mitigate power line noises and thermal reliability problems. Results show 18% reduction in power and more smoothly distributed power dissipation over the floor-plan area compared to conventional floor-planners with the same delay constraint. There is however an area penalty.

2.8.3.3 Placement

A performance driven placement algorithm for minimizing the power consumption is presented in [28]. The problem is formulated as a constrained programming problem and is solved in two phases: global optimization and slot assignment. The objective function used during either phase is the total weighted net length where net weights are calculated as the expected switching activities of gates driving the nets. Constraints on total path delays are also accounted for. On an average, this procedure reduces power consumption by about 10% at the expense of 2% increase in circuit delay compared to a placement program minimizing the total interconnect length.

2.8.3.4 Routing

Routing for low power can be performed by “net weighting” where again the net weights are derived from the switching activity values of the driver gates. The nets with higher weights are more critical and should be given priority during routing.

2.8.3.5 Gate Sizing

The treatment of gate sizing problem is closely related to finding a macro-model that captures the main features of a complex gate (area, delay, power consumption) through a small number of parameters (typically width or trans-conductance). The first major contribution to transistor sizing problem was done in [34]. This optimization technique is greedy in the sense that it picks a path that fails to meet the timing requirements, and resizes some transistor on the path so as to meet the constraint. The procedure is iterated until all timing constraints are satisfied or no further optimization is possible. A novel approach to gate sizing is described in [30]. This approach attempts to map linearly the path based timing constraints and uses a linear programming solver to find the global optimum solution. The drawbacks of this approach are the omission of slope factor (input ramp time) of input waveforms from the delay model and use of simple power dissipation model that ignores short-circuit currents.

2.8.3.6 Wire Sizing

Wire sizing and/or driver sizing are often needed to reduce the interconnect delay on time-critical nets. Wire sizing however tends to increase the load on the driver and hence increase the power dissipation. A combined wire sizing and driver sizing approach that reduces the interconnect delay with only a small increase in the power dissipation is explained in [26]. Experimental results show that for the same delay constraint, this approach reduces the power by about 10% when compared to the conventional method of driver sizing only. Alternatively, this

approach produces delay values that are up to 40% lower when compared to the conventional method (at the cost of increasing the power dissipation by 25%)

Due to the disproportionate scaling of interconnect parameters with decreasing feature sizes, there is an increasing gap in the compatibility factors for gate based capacitance, delay, and power and the same for Interconnects. Since, global busses are indispensable, techniques must be deduced to reduce the power consumption due to switching of logic levels on these global on-chip wires. One of the ways of reducing the net power dissipation is to reduce the activity factor on interconnect. Where communication cannot be avoided - introducing new encoding methods can reduce this and/or signaling mechanisms such that the net switched capacitance reduces. Note that an area penalty must be incurred in establishing such techniques in the design. Also, care must be taken to see that the energy required by the new encoding methodology does not even out the power advantage.

2.8.3.7 Clock Trees

The clock in a synchronous circuit has the highest switching capacitance, and hence must be carefully laid out in the floor plan. Various schemes for clock tree distribution have been discussed in various journals. They should be adopted carefully when low power is a concern.

2.8.3.8 Voltage swing on on-chip and off-chip interconnect

Reduced voltage swing are often used for saving power on interconnects. [31] provides some insight into the optimum swing for a high performance circuit. Their results indicate that on-chip interconnect swings between 100 and 400 mV are ideal for power savings to the order of 10x for .18 μm process.

2.8.4 Problem with Interconnect analysis

In the present and earlier CAD tools, interconnect was approximated with the wire-load model. This model was only approximate; however due to the small contribution of interconnect to the circuit power and delay; this was acceptable as small errors did not affect the overall approximation of the results. However, in DSM today, interconnect contributes significantly to the overall delay and power consumption figures. Hence, the primitive wire-load model does not suffice and timing closures of designs is more unreliable.

2.8.5 Locality of Reference

This feature can help reduce capacitance in interconnecting circuits [32]. Thus leading to reduced power consumption figures. This feature has to be exploited at the architectural and

layout level shown in Figure 38 and Figure 39 . Operations being performed for a particular function should be in the vicinity of the parent function. This prevents power hungry data transfers across global busses.

Figure 38. Cascade filter layout – Non-local implementation from Hyper [32]

Figure 39. Cascade filter layout –Local implementation from Hyper-LP [32]

Table 2 shows the comparison of power consumption for the layouts in the figures above.

Component	<i>Hyper</i>	<i>Hyper-LP</i>	Percentage reduction
Buses	2.0	0.4	80.0
Multiplexors	3.7	1.5	59.5
Buffers	1.0	0.9	10.0
Others	8.7	7.4	14.9
Total	15.4	10.2	33.77

Table 2. Comparison of power consumption (mW) in the Hyper and Hyper-LP implementation of the cascade filter

2.8.6 Exploiting Regularity in Design for low power

By exploiting the regularity and common computational patterns in the algorithm to reduce the fan-outs and the fan-ins of the interconnect structure (Figure 40) led to a reduction of almost 50% in power consumption [33]. This method is most applicable to ASICs in which exploitation of repeated occurrences of operations such as Multiply-And-Add can lead to a more regular interconnect pattern.

Figure 40. Preserving regularity leads to a simplified interconnect structure (a) regular assignment, (b) non-regular assignment

2.8.7 In Summary

It is observed in this section that interconnect is increasingly playing a more prominent role in design optimization towards area and power concerns. Whilst deep sub-micron technology has alleviated the concerns of interconnect delay (which was a concern up to 0.18 micron technology), it now poses new design challenges to optimization approaches.

2.9 Summary

In the very beginning of this literature survey it has been established that system level maturity is required to provide for effective design optimization and design space exploration. In the constraint driven design optimization and analysis approach it was also noted that power optimization is increasingly becoming the concern in the complex and dense designs posed to architects. Having looked into the need to have system level design and the recommendations by the ITRS, the rest of this chapter provided a broad based overview of the different design optimization options at various abstractions of the design hierarchy. The broad literature survey presented was necessary to provide an in-depth understanding of the integrated circuit design space exploration techniques.

Following the norms of economies of scale, the literature survey also makes it obvious that the maximum gains in design optimization efforts is obtained from addressing the most occurring design units. For example, when we take a cache access mechanism – most of the power is consumed in switching the high parasitic loads on the address and data interconnect. This power consumption figure is much higher than the power dissipation in the relatively simplistic state machine. Thus, concentrating on the hot spots provides the highest gains in power consumption reductions. Similarly, events that are infrequent are also less important to be addressed as compared to the more frequent events. For example, in a regular microprocessor, a floating-point operation occurs less frequently than an addition (or subtraction). Thus, concentrating on a more energy efficient adder is more rewarding than the less frequently used multiplier. This

strategy would not work for a digital signal processor (DSP) where multiplications are probably as frequent as an add/subtract operation.

Noting the high-level of design complexity in circuit designs today, it is also important to note that design alterations at the low level of design abstraction (circuit-level & gate-level) can be extremely costly in terms of design implementation effort for the various versions for design space exploration, and the simulation time required for the verification and validation of the various implementations. Thus, time to market implications (TTM) and non-recurring engineering (NRE) costs become increasingly important in dense designs. Multiple implementation and verification cycles can impact the TTM and NRE constraints.

Due to the wide range of design methodologies and applications, all approaches to design optimization do not work equally effectively in varied scenarios. Furthermore, in a bid to harmonize the design process, the ASIC tool vendors tend to constrain the circuit and gate level implementation details. As an example, in a cell-based ASIC design flow for an application specific integrated circuit (ASIC), the designer is given non-alterable constraints such as – fixed voltage cell-based library, data-path libraries and fixed circuit technology. In such situations the designer can only apply system, algorithmic, architectural, and gate level power optimization strategies.

It is also observed that various algorithm optimizations for low power can greatly affect the energy consumption. By reducing the level of complexity in the operators used in an algorithm, they can be fine tuned for lower computation requirement. Conversely in a full-custom ASIC flow, significant power reductions can occur if power optimal transformations are applied in a concerted manner involving top-down and bottom-up flows. In this approach, the designer can decide all parameters affecting the ASIC design to fine-tune power requirements in the circuit.

Various other parameters come into play in the design of programmable general-purpose processors or controllers. In such designs the architecture has to cater to various applications that can be ported to it. There is a lot of “generality” in the design of data-path, arithmetic logic units (ALU), memory access, on-chip cache and other functional units. Hence, the onus of low power is shifted to either the lower or higher hierarchies of design. In the design of a general purpose computing platform, circuit level optimizations in the full design and gate-level optimizations in the control logic can provide high gains. Further, careful instruction set design can lead to better energy efficiency. It is noted that the sequence of instruction processing can be decisive in the total energy consumed by the device. Thus compiler level optimizations are used to perform the optimal static ordering of instructions.

As one might expect – optimizations at different levels of abstraction provide different gains. Thus one must first apply coarse-grained high-level optimizations to achieve the highest gains. Then, fine-grained low-level optimizations can be applied. Thus one of the possible approaches is to develop a analysis, simulation and implementation in a hybrid manner – top down as well as bottom up. Each of these techniques could present a tradeoff in the amount of efficiency obtained in its approach towards low power devices. Some of the methods are often intractable and difficult to apply. Almost always they also add significantly to the complexity of the architecture, and end up adding significant silicon area.

The chapters that follow shall address design optimizations at the hardware description and system level design language levels to tradeoff their relative merits in algorithm analysis and architecture design space exploration. There shall be specific interest in the TTM implications of multiple design iterations at low level of design abstractions. Further, accuracy and performance analysis shall need to be addressed to identify suitable environment and design flows that tradeoff the TTM and NRE constraints of complex designs. One possibility shall be to evaluate a hybrid strategy.

In the current level of design complexity, it is imperative to look into the tradeoffs of exploration of architecture for algorithms at various levels of design abstraction. The efficacy of system level constraints to be imputed into the simulation environment will also need to be verified. Thus, system level language shall be experimented to fuse selected system level modeling tools for profiling and analysis towards facilitating the design of optimal hardware architecture. Case studies will allow us to come up with a framework for algorithm and hardware architecture design space exploration.

Chapter 3 VHDL Based Techniques for Hardware Architecture

Design Space Exploration

This chapter establishes the basics of design space exploration using a compute intensive algorithm as the design vehicle. The design space exploration follows an explicit design exploration approach to establish the intuitive approach towards design space exploration and architecture partitioning. This exploration process is accompanied by hardware implementation and synthesis using VHDL (VHSIC Hardware Description Language).

3.1 K-means: The Algorithm

The application for exemplifying the design partitioning and optimization towards different design constraints needs to possess a few inherent salient features. The design should have a strong data-path dependency, which can be used to apply the data centric power optimization steps. Control logic should also be strongly present in the logic characteristics to see the difference in impact of the techniques in data-path and control design.

The author has chosen the K-means algorithm which is used as a companion algorithm in speaker verification applications. In speaker verification [34], K-means provides the initial centers for the Estimation Maximization (EM) algorithm. Other applications [35] include Image segmentation, Artificial Intelligence, Fuzzy Systems and Vector Quantization.

K-means algorithm is an iterative descent technique that estimates a sequence of centres in an n -dimensional space. This clustering algorithm was proposed by Forgy [55] and McQueen [54]. The algorithm is primarily used to provide initial estimates of clusters in the sample space. The K-means algorithm is a computationally complex algorithm, which employs vector arithmetic operations across 20 dimensions. K-means architecture [56] is an unsupervised clustering algorithm used popularly in neural network and statistical techniques.

The algorithm above was chosen because of the inherent iterative nature that can be exploited for power consumption reduction as well as the diversity in the architecture. Since, this architecture is totally infeasible in the extremely parallel implementation; it has to be implemented with some amount of sequential control. Also, it requires interface with memory blocks and/or register files. All these properties of the design shall allow us to apply some basic transformations to the architecture and obtain differences in design parameter estimates, namely – area, power, and time. It is also a data-path centric design, which, allows us to apply pipeline techniques and test

for changes in the design parameters. Finally, the highly iterative nature of the design allows us to increase the energy gains affected by the modifications to the critical loop of the iteration.

The task can be likened to a digital signal processor, where real time data capture takes place along with simultaneous processing. Thus, there is not much avenue of shutting down units, which are not used in the chip. This restrains us from using obvious techniques of power management, which include power down modes for functional units not being used. The dynamic range also forces us to evaluate the tradeoff between accuracy and power by experimenting with different word lengths in the design.

3.1.1 Algorithm description

The K-means algorithm is used to perform unsupervised clustering of a given data set of D dimensional data vectors, $\vec{x}_i$, $1 \leq i \leq T$, where, T is the number of data vectors in the data set.

K-means is a classical clustering technique, which is usually performed to obtain an initial estimate of the parameters of the multi-variate distribution in the data set.

Given a stream of D dimensional data vectors $\vec{x}_i$, $1 \leq i \leq T$ The K-means clustering algorithm clusters the training vectors according to minimum Euclidean distance vector operation (for D -dimensions), $(\vec{x} - \vec{\mu})^T (\vec{x} - \vec{\mu})$ between the mean vector ($\vec{\mu}$) of the cluster and the training vector ($\vec{x}$). The following are the steps in an iteration of K-means algorithm.

- To initialize, M random vectors from the training set are selected as the means of M clusters.
- Each vector $\vec{x}_i$, $1 \leq i \leq T$ is assigned to cluster j , iff, $d(\vec{x}_i, \vec{\mu}_j) < d(\vec{x}_i, \vec{\mu}_k)$
 $\forall k \neq j, 1 \leq j, k \leq M$.
- The new mean of a cluster is the mean vector of all the vectors assigned to that cluster.
- The weights are calculated by determining the proportion of the vectors assigned to a cluster and the covariance matrix of the cluster is the covariance matrix of the respective vectors.

The data vectors, $\vec{x}_i$, $1 \leq i \leq T$, are D -dimensional values. Thus the mean (center) of the various clusters find themselves aligned in hyperspace as depicted in Figure 41. The smaller circles represent the M different clusters with the respective means. The data vectors belonging to the respective clusters (after convergence of the algorithm) are depicted inside the clusters.

Figure 41. Illustration of Clustering by K-means

Figure 42. Iterative nature of the K-means algorithm

The M clusters represent the number of bounded hyperspaces that the data should be clustered into. In the example used in this paper, we use $M = 16$ mixtures and $D = 8$ -dimensional data vectors.

The operation, $d(\bar{x}_i, \bar{\mu}_j) < d(\bar{x}_i, \bar{\mu}_k)$, involves the calculation of the Euclidean distance, which can be expressed as the vector operation $(\bar{x} - \bar{\mu})^T (\bar{x} - \bar{\mu})$. This can be expanded as:

$$d(\bar{x}, \bar{\mu}) = (\bar{x} - \bar{\mu})^T (\bar{x} - \bar{\mu}) = (\bar{x}_1 - \bar{\mu}_1)^2 + (\bar{x}_2 - \bar{\mu}_2)^2 + \dots + (\bar{x}_D - \bar{\mu}_D)^2$$

This involves $D=8$ subtractions, $D=8$ multiplication's and $D-1=7$ additions to compute the Euclidean distance between one data vector and one mean. This has to be done for $M=16$ mixtures/clusters. Steps II and III in the algorithm above are repeated till the means are stable. The iterative nature of the algorithm is illustrated in Figure 42. The models represent the state of the means and other system parameters at the end of each iteration.

3.2 Design Space Exploration using VHDL

Hardware architectures are considered for implementation of computationally intensive applications so that specific bit-widths and latencies can be catered for and exploited to give rise to fine-grained parallelism for an efficient design. Efficiency of the design can be gauged by the performance of the underlying algorithm implemented.

The K-means algorithm involves multi-dimensional input data vectors, x_i . Hence, it is intuitive and immediately apparent that there is inherent parallelism in the algorithm. This is an apt scenario where the appropriateness of hardware implementation shall be exhibited. Unlike a conventional microprocessor or a DSP, the ASIC can be geared towards the specific needs of the

application. In this case we shall employ multiple and parallel data-paths and ignore any complications in the design of the control unit that routes the data.

3.2.1 Computing the *Euclidean distance*

The Euclidean distance between $\mathbf{x}$ and $\boldsymbol{\mu}$ is given by

$$\text{E-distance } [\mathbf{x}, \boldsymbol{\mu}] = [\mathbf{x} - \boldsymbol{\mu}]^T [\mathbf{x} - \boldsymbol{\mu}] = [\mathbf{x}_1 - \boldsymbol{\mu}_1]^2 + [\mathbf{x}_2 - \boldsymbol{\mu}_2]^2 + \dots + [\mathbf{x}_d - \boldsymbol{\mu}_d]^2 \quad \dots (11.5)$$

Where, $\mathbf{x}$ is a data vector of d dimensions, and $\boldsymbol{\mu}$ is a mean vector of d dimensions,

Since this operation is highly parallel in nature, it implies that the architecture should have parallel data-paths to facilitate the calculation of the E-distance with the minimal latency. Such a highly parallel architecture (Architecture-A) can provide the host application with m E-distances in one clock cycle t_{clk} .

Assuming an application with d dimensional data vectors and m mixtures in a multivariate distribution; the architecture to exploit complete parallelism, shall need m E-distance calculation blocks as shown in the first stage of Figure 43. Once the m E-distances have been calculated, a tree of $\lceil \log_2 m \rceil$ stages of comparators is necessary to identify the nearest cluster k .

Each E-Distance computation unit can be detailed. A pipeline stage of m subtraction blocks is provided, and then a stage of m multiplier blocks. This shall be followed by $\lceil \log_2 d \rceil$ stages of $\lfloor d/2 \rfloor, \lfloor d/4 \rfloor \dots 1$ adder's at each level.

The re-use of arithmetic blocks and folding of the architecture in the E-distance computation decides the complexity of the control logic, the area of the design and speed. This shall be exhibited by introducing three variants – Architecture-A, Architecture-B and Architecture-C.

3.2.2 Internal Memory

The K-means architecture requires internal memory, which can be broadly subdivided into three blocks:

- accumulation memory
- “data vectors assigned to k ” memory
- mean vector memory

The accumulation memory is used to store the accumulation that results from the operation:

$$\text{AccumulatedSum}_k = \text{AccumulatedSum}_k + \mathbf{x}_i ;$$

Note: this is a vector operation of d -dimensions for $[1 \leq i \leq n]$.

The “data vectors assigned to k ” memory [k denotes the number of clusters that the data is getting segregating into] is used to keep track of the number of vectors assigned to a mixture. The mean vector memory is initialized externally for use by the algorithm in the first iteration. Thereafter, it is used to re-initialize the mean vectors to the values obtained in the new iteration.

After the comparators have the designated cluster k as the closest mixture to a given data vector $\mathbf{x}_i$, it has to be added up to the running, partially accumulated sum for the subsequent calculation of the mean. The comparator block identifies which mixture, k , the input data vector $\mathbf{x}_i$ should be allocated to. The value k is then used to reference to the locations where the partial summation values for each of the mixtures is stored. At the same time the value for the mixture k in the “data vectors assigned to k ” memory is incremented to represent that a new vector is assigned to the mixture k . After all the vectors in the data set are processed, the re-initialization of the mean memory bank is carried out.

Since, the input data vector $\mathbf{x}_i$ is required to perform the summation, a FIFO queue is introduced in the earlier pipe stages so as to remove the conflict of memory. This is possible since, both the pipe stages: the E -distance calculation block and the partial summation block shall require different data vectors at the same time.

The various scratchpad memory modules required can be maintained either on-chip or off-chip. However, keeping the memory off-chip shall slow down the access and hence the speed of the pipeline clock.

3.2.3 Architecture A: Massively Parallel Implementation

This proposed architecture (Architecture-A) in Figure 43 is capable of computing the E -distance between a single data vector $\mathbf{x}_i$ and each of the mixture *mean's* (μ) simultaneously in one clock cycle (ignoring the latency in filling up the pipeline stages).

The multiplier block dictates the maximum pipeline latency in the design. This implies that slower, *area-efficient* adder/subtraction blocks can be used in the other pipeline stages with the largest investment of area in the multiplier blocks.

Since the slowest pipeline stage has a latency of around 15ns, the clock period can be driven only as high as 66 MHz. The clock frequency can be further stepped up to about 100 MHz if we can further pipeline the multiplier block into two stages with a maximum latency of 10 ns. Beyond this the pipeline latches would incur too much of an overhead. The dividers are not included in the pipeline evaluation as they reside outside the pipeline, and are required only once per model re-initialization.

The complexity of the architecture A is shown in tables below. The illustrated measures do not include the net cost of the control unit that drives the data-path. However, it can be assumed that in this massively parallel architecture, the design of the control unit will not be as area intensive as the data-path.

This architecture however, imposes serious overheads in terms of memory blocks and data arrangement patterns. The control unit issues commands to fetch d data words (only one data vector's E-distance is being calculated) from the data memory and dm mean data words (as the E-distance from m mixtures is being calculated) simultaneously to feed them to the d -subtraction blocks in the first pipeline stage. Thereafter the multipliers in Figure 43 provide the square of the difference between each corresponding $\mathbf{x}_i$ and μ_i , $\forall i \in 1 \leq i \leq D$.

Figure 43. Architecture A Description

ARCHITECTURE	TIME COMPLEXITY		
Architecture A	$nT + mT_{div}$	$\sim [n + m]T$	$\sim 6\,032T$
Architecture B	$nTm + mT_{div}$	$\sim [nm + m]T$	$\sim 192\,032T$
Architecture C	$nTmf + mT_{div}f$	$\sim [nmf + mf]T$	$\sim 1\,536\,256T^+$

* T_{div} = time for one divide cycle; T = one clock cycle

+ f = number of folds in Architecture C

Table 3. Time Complexity

SUB BLOCKS	ARCHITECTURE A	ARCHITECTURE B	ARCHITECTURE C
Subtraction	Md	D	d/f
Multiplier	Md	D	d/f
Adder	$m(2d-1)$	$2d-1$	$2[d/f]-1$
'<' comparator	$m-1$	1	1
Divide	D	D	D/f

Table 4. Design Complexity for the Various Architectures

This requires d blocks of memory for the data vectors with the vector values striped across all the d -blocks. The requirement for the mean memory blocks is however higher, at dm memory blocks to supply to the dm subtractor blocks for simultaneous distance calculation of one data vector with respect to all the mixture means.

Figure 44. E-Distance Calculation

3.2.4 Architecture B: Folding by ‘m’

As the topology of *Architecture-A* is quite expansive in terms of area and quite unreasonable in terms of memory banks, a more area efficient architecture [*Architecture-B*] is proposed. In this architecture the E-Distance of each data vector is computed with only one mixture at a time. Effectively having folded the architecture by m , we have slowed it by a factor of m . However, as, can be seen in Table 3; the complexity on terms of area of the data-path is much reduced. This architecture uses only one of the E-Distance blocks (Figure 45). Thus the comparators results need to be stored in registers, as all the distances are not available at the same time. This shall be revisited in *Architecture-C*.

Due to the folding in the design, the computation of m E -distances slows by a factor of m . Thus, this topology shall sustain a rate of one clock cycle per E -distance calculation. This design however, requires d memory blocks each for the mean and data memories.

3.2.5 Architecture C: Folding by 'm' and 'd' for optimal tradeoff

To achieve a more adaptable and application generic design, a third design topology (Architecture C in Figure 46) is proposed, which further folds the design with respect to the number of dimensions being calculated at a time.

The disadvantage - since, we are breaking the concurrency and parallelism inherent in the algorithm, we now have to include a more complex control unit, which deals with the coordination of the summations. If, however, the number of dimensions and mixtures can be kept to a power of 2, then control unit becomes much more efficient and elegant as the entire dynamic range of the registers are used for addressing and the counters used to cycle through the various addresses do not need a comparator (to check for non- 2^n boundaries) as the addresses roll over at a 2^n boundary.

The implicit advantage of this more complicated control unit is in the fact that it now has better and more reconfigurable control over the arithmetic data-path. Thus, the architecture can now perform K-means clustering for any application with a general requirement of d dimensions and m mixtures. The provision of programmable-on-reset registers can make the configuration parameter a run-time variable. Else a command-driven interface can also be provided as a part of the K-means control unit. It can also be done at design stage for specific applications by using generics in the VHDL syntax.

Figure 45. Architecture-B

Figure 46. (Architecture-C)

3.2.6 Synthesis Results

The designs described in the section earlier were implemented in VHDL with parameterized design parameters. The designs were synthesized using the Synopsys Design Compiler and Passport Cell based libraries in 0.35 micron technology. Avant! Passport memory compiler was used to generate the memory blocks required in the design.

The experimental results in terms of ASIC gates (graphs in Figure 47 & Figure 48) suggest that we have 4 parallel arithmetic pipelines for the E-distance calculations. This also comes with the reasonable overhead of 4 banks of memory each for the mean and data values.

The FIFO buffer is used to store the data set vector x_i in rows of 4 each. This is devised so that the pipe stages above it are not stalled when the accumulation of the vector is done to a particular mixture k . The FIFO shall consist of d/f rows of the data.

Thus, we have now effectively slowed down the architecture by mf times, where f is the number of folds. Thus, one E -distance is calculated in f clock cycles now. The comparative quantitative measures are provided in the table above.

Figure 47. Area cost vs. nos. of folds, f

Figure 48. % contrib. of various area vs. folds, f

3.3 Overview of the VHDL based design flow

In the VHDL based design methodology used in this chapter, it is clear that there was a language translation involved in moving to the HDL. Further, tradeoffs had to be evaluated with completely revamped VHDL coding and was quite time consuming. This flow is illustrated in a descriptive manner in Figure 49. Though the analysis and simulation results are as accurate as they can get in the front-end ASIC design flow, it is also very time-consuming and result in fairly independent analysis of the hardware architecture costs. This is a limitation in implementing a thorough design space exploration exercise.

Figure 49. Architecture Design Space Exploration using HDL

3.4 Summary

In the analysis posed in this chapter, the latency of operations and area of silicon was traded off to provide a design with optimal speed and area. Being able to have a quick tradeoff analysis methodology at this point can be quite useful to a design architect. It is obvious that the highly parallel implementations will have a high power budget as compared to the implementations which are more sequential. However, these power aware decisions were not a part of this area-time design tradeoff analysis. Thus the design optimization exercise is incomplete at this point. Though, the power estimation can be done in this design flow, it is an intensely time consuming process for all the three architecture variants. It is however desirable that we can perform some initial analysis to decide which architecture framework to adopt with some initial estimates on area, performance and power. This will make the time consuming task of design parameter estimation more meaningful as it is then a more directed approach. Further, this makes it important to be capable of performing this tradeoff analysis at a system level design language level and derive estimates to provide the basis for design trade-off analysis. As an example, the number of folds in the architecture can be decided on the basis of area-time-power estimates at a very initial stage of design. It was clearly illustrated earlier that the Euclidean calculation module can be implemented as a pipelined data-path. Further, the exploration of data-path width for the various operands can also lead to an optimum balance between performance and power consumption. Dynamic range required by the data-path shall help us reduce the widths and, hence reduce power consumption in the circuit. Since, the module requires close interaction with

memory modules, memory-partitioning techniques can also be experimented to observe gains in power consumption. Splitting up the memory modules shall allow us to turn on and off the sense amplifiers and other internal circuitry of the memories selectively thereby only incurring leakage power. However, this will lead to a marginal increase in the silicon area occupied by the memory blocks. The motivation for controlling the data-path design is that different architectures have different requirements of clock period, pipelining, throughputs and resource requirements. Hence, dictating different exploration points in the design space.

Chapter 4 SystemC Based Techniques for Hardware Architecture Design Space Exploration

In the earlier chapter we discussed a brief design example where a complete architecture refinement process was *walked-through* by implementing all the individual architectures in hardware description language (VHDL). However, the highly time intensive nature of such multiple estimations can make complete analysis of such architectures infeasible. The need to realize closure in the three major design parameters: area, time and power or, energy in a time constrained manner is thus important. A framework for analyzing power at the system level would hence prove attractive when development cycles are subjected to time to market pressures (TTM). This chapter discusses the study of the use of high-level design language approach for architecture partitioning of integrated circuit designs. A system level approach is taken whereby the design is partitioned and budgeted for power and energy requirements (as the major constraint in this constraint driven approach) at the very abstract stage. In this chapter, a top-down flow of power budget information is implied to allow for better partitioning of design with energy as a constraint. The popular K-means unsupervised clustering algorithm is reused as the design example to demonstrate the potential for successful power closures using systems level considerations.

SystemC was chosen as the design language due to its high versatility and the promise of it being capable to nurture a design from concept to implementation. To effect the architectural changes, a series of design exploratory transformations were carried out using behavioral scheduling commands. Our investigations show that we can evaluate multiple architecture options by adopting standard techniques and commercially available tools at the system level. Exploratory analysis is also carried out on parallel computation blocks, which also results in decreased power consumption. It will be concluded that System Level design languages do promote for convenient architecture exploration leading to the identification of architectures with better throughput and energy efficiency.

4.1 *Constraint driven design space exploration*

4.1.1 Energy/Power Constraints

4.1.1.1 Semantics of Power Consumption

Power consumption is basically the rate of consumption of energy. Thus, average power consumed at 1 mW implies that 1mJ of energy is consumed per second. Power consumption

being a time-averaged term, it can be highly misleading in cases where the time of execution is much smaller than the time resolution of the reference (1 second). This is especially the case in today's design technology where clock periods are in the range on 1-10 ns or less.

Moreover, logic blocks in a design are seldom used for 100% of the time reference (1 second). This implies that the average power consumption is valid for only a fraction of the total time required for the task. Gated clock techniques and sleep modes can take advantage of this type of operation mode to reduce the average power consumption further. Thus, implying a scenario where we consume power at the peak power consumption rate for the time it takes to perform the task, following which we change the operation mode to sleep state so as to consume only the leakage power requirement.

Figure 50(a) illustrates the case where average power consumption (P_1) is consumed for the entire time that the processor/design is active. This average power consumption can however be reduced to P_2 in Figure 50(c) by operating the logic block at peak power consumption for the time that the operator is required and then switching it to low power mode for the time when it is not required. This shall lead to an average power consumption, $P_2 < P_1$, yet providing the same functionality and throughput.

Figure 50. Illustration of (a) actual (b) peak and (c) average power consumption

It is pragmatic to note that there is a high likelihood of area of the design increasing due to the extra low power decision logic and sleep logic structures required. However, in this design era, it is probably a small disadvantage due to the high logic density. This observation is important for portable devices where most of the ASIC's are designed for low power sleep modes, by the use of (almost mandatory) clock gating and sleep logic. Further, in battery-operated devices the total energy consumed between charges is the more important criteria. Hence, energy quantification is also necessary. It is however well appreciated that such design analysis is seldom possible in today's design flow. This is in keeping with the ever-increasing circuit sizes and density.

The author however, intends to show how such estimates can help at the system design level to compare different approaches to implementing architectures. A typical behavioral scheduling

report shows the clock cycles counts required by various parts of the design. However, it does not indicate the total number of clock cycles required to complete a task in iterative designs. The total cycle count (illustrated in sections that follow) can be used effectively to estimate the required energy consumption in a task cycle. This can be used to find out how much of energy is required by the design before it can be put into low power sleep modes.

4.1.2 Power Centric Approach

In most ASIC design challenges, throughput (latency of computation) is a static requirement. Unlike general-purpose microprocessors and microcontrollers, ASIC's have to cater to a certain application domain and hence have hard requirements in terms of throughput. Area and power on the other hand are parameters to be stretched in the design exploration space to achieve the optimum balance. The design exploration space of area versus power can differ in various applications. In applications where power supply is infinite (server applications), area can be optimized and the solution can be iterative in nature. However, in client applications (portable devices) where energy is derived from batteries, area can be considered a premium and energy conservation should get maximum priority, at the cost of increased area.

Figure 51. Area, Performance (time) & Power Tradeoff Trends in Server and Client Designs

The graph in Figure 51(a) above illustrates the design exploration space of typical server and client processor device. In client processor designs (Figure 51(b)), power being a limitation; the designer has to achieve the necessary performance requirements by exploring the two dimensional area and time axes. Meanwhile, Figure 51 (a) illustrates that a server application has the convenience of trading area and time for infinite source of energy. As in most design exploration scenarios, the designer will want to have a quantitative estimation of the tradeoff

derived from designing for higher area and lower power consumption. However, to do so the designer needs to rewrite the design to suit the new version of the architecture. This may involve rewriting lengthy VHDL or Verilog modules, which dampens the resolve to address power concerns. Even with tools like behavioral compiler for hardware description languages, the analysis is made difficult due to the widespread use of architecture specific coding methodologies.

In present design flows, the power consumption estimate can only be made after the HDL blocks are written and simulated. However, similar to timing closure issues, it can be envisaged that this too shall be too late a stage to address power consumption in the design cycles of the future. This gives rise to the need of having a system level design language, which is capable of allowing a quick power consumption estimate for a given architecture. The power centric partitioning done by the design architect shall then allow a top-down flow of power budget information to the designers fine-tuning and optimizing the individual blocks of the design.

In SystemC based partitioning that follows, the author provides analysis and methodology for helping the design architect to take decisions based on the bounded axes. As an example, constraining the energy budget for a device, the designer can now limit the maximal duplication of resources. In other words, increased performance will have to be provided by more efficient techniques of algorithm implementation and technology inputs.

4.2 System Level Design Language: SystemC

Even with the keen introduction of many different programming and modeling languages, ANSI C and C++ are still used as the staple choice for modeling algorithms of every kind. ANSI C and C++ are the first choice for expressing a new behavior, whether it is targeted at hardware or software implementation. It is hence no wonder that SystemC [36], a system level modeling language was based entirely upon the fundamentals of ANSI C and C++. SystemC allows conceptualization of an algorithm and its verification. It provides an executable specification, which, can be used for validation against results at any stage of design. This prevents intermediate involuntary errors from creeping into the design due to translation from a modeling language to a different implementation language. The basic nature of SystemC allows the Open SystemC Initiative (OSCI) to tap into the large number of programmers trained in C/C++. The SystemC Compiler provided by Synopsys integrates SystemC based design entry into their existing Front-End ASIC design flow. The compiler is capable of delivering a RTL or Gate level netlist. This can be taken through the traditional design flow using the Synopsys tools.

4.3 SystemC based design flow

The system design environment is important in supporting efforts for system level design partitioning. Even though the transformations might be possible without any specific design environment, no synthesis result can be analyzed without the presence of supporting results and compilers.

In our flow, we are using Avanti's Compass Passport 0.35um library and SystemC Language version 1.2.1b. The tools used are:

- a) Synopsys SystemC Compiler 2001.08
- b) Synopsys Behavioral Compiler 2001.08
- c) Synopsys Design Compiler 2001.08
- d) Synopsys Power Compiler 2001.08

The design flow discussed provides the overview of our flow for system level decisions to be quantified.

4.3.1 SystemC Simulation

The behavioral SystemC code is first simulated in Visual C++, and verified for correctness with the existing reference results. This process was simplified due to the common simulation platform for the SystemC code and original algorithm implementation. Further, the extensive file input/output handling available in C++ allows us to easily interface to the data stream files and compare and contrast the results. At this stage too the algorithm can be optimized using the usual strategies of optimizing C/C++ code. Following the validation of the simulation results and the above command, we move to the UNIX platform for synthesis and architecture exploration.

4.3.2 Memory Instantiation for arrays

At this point a few additions to the code had to be made so as to include memory libraries. This step allows for lower compilation and elaboration time.

Three data arrays were used in this architecture. The data array *data_vectors_in_clusters* was anticipated to not require more than 32 locations (typically 16), hence was mapped to a register files. The other two data arrays were mapped to memory blocks earlier generated using the 0.35 um memory compiler. Synopsys tools *Library Compiler* and *Memory Wrapper* were useful in importing the memory files into the SystemC compilation routines.

The code snippet below shows the instantiations of the memory blocks in the SystemC code file

```
sc_int<32> summation_mean[cM*cD];
/* synopsis resource RAM_B:
variables = "summation_mean",
map_to_module = "RAM256x8_wrap"; */

sc_int<32> data_vectors_in_cluster[cM];
/* synopsis resource RAM_A:
variables = "data_vectors_in_cluster",
map_to_registerfiles = "TRUE"; */

sc_int<32> mean[cM*cD];
/* synopsis resource RAM_C:
variables = "mean",
map_to_module = " RAM256x8_wrap "; */
```

It is important to note that different configurations of memory modules can be experimented with ease by changing the line *map_to_module*. The following commands are required in *dc_shell* to include the required synthetic and library files for the memories in the design.

```
dc_shell> define_design_lib ram_library -path $HOME/LowPower/Syn/RamLIB
dc_shell> analyze -f vhd1 ../Mem/ram_wrap.vhd -library ram_library
dc_shell> elaborate ram_wrap -architecture wrap -library ram_library
dc_shell> analyze -f vhd1 ../Mem/ramb_wrap.vhd -library ram_library
dc_shell> elaborate ramb_wrap -architecture wrap -library ram_library
dc_shell> analyze -f vhd1 ../Mem/RAM128x32_wrap.vhd -library ram_library
dc_shell> elaborate RAM128x32_wrap -architecture wrap -library ram_library
dc_shell> analyze -f vhd1 ../Mem/RAM256x32_wrap.vhd -library ram_library
dc_shell> elaborate RAM256x32_wrap -architecture wrap -library ram_library
```

The file is now ready to be compiled by the SystemC compiler. However, to enable collection of timing information for behavioral analysis, the command below is required to enable the SystemC compiler to gather timing information for behavioral analysis.

```
dc_shell> bc_enable_analysis_info = true
```

4.3.3 Compiling the SystemC code in *dc_shell*

The SystemC compiler (under *dc_shell*) was used for analysis for synthesis and syntax. It outputs a internal database file (.db) for timing analysis.

```
dc_shell > compile_systemc ../SystemC/kmeansFinal/kmeans.cpp
dc_shell > create_clock pClk -period 15
```

The design is now ready to be scheduled according to the various libraries available for operations. However, to obtain meaningful results, one must constrain blocks of code or operations as is desired.

4.3.4 Behavioral Analysis

This can be done in two ways. Compiler directives, like the one illustrated below can be used to constrain the hardware implementation. The command below unrolls *for loops* in the design. Thus resulting in parallel hardware blocks.

```
/* synopsys unroll */
```

Secondly, one can use `dc_shell` commands (below) to pipeline loops, which are a bottleneck in the design.

```
dc_shell> pipeline_loop loop_label -initiation_interval n -latency m
```

The two illustrations above are only to show the nature of commands that can be used to perform architecture exploration. The Cocentric SystemC Compiler Manuals [37] however, documents all such available options. In summary, such commands and directives together form a powerful combination in architecture exploration.

This step is followed by the following commands to allow the design to be scheduled

```
dc_shell > <architecture constraints are placed here>
dc_shell > bc_check_design -io_mode superstate_fixed | cycle_fixed
dc_shell > bc_time_design
dc_shell > schedule -io_mode superstate_fixed
```

At this step the design is ready for graphical analysis of the architecture with respect to reservation tables, loops, resource requirements and all other design parameters by using the behavioral view. The graphic user interface (GUI) can be invoked by:

```
dc_shell > bc_view
```

4.3.5 Synthesis to Gate-Level Netlist

This is then followed by compilation to gate level netlist by

```
dc_shell > compile -map_effort medium
dc_shell > write -hierarchy - output <file name.db>
dc_shell > write -hierarchy - output <output.vhd>
```

4.3.6 Power Estimation of architectures generated with SystemC

Synopsys provides two approaches to allow the designer to consider architecture tradeoffs using power consumption as the constraint for SystemC based design files.

Namely, RTL Power Estimation and Gate-Level Power Analysis [38]. RTL power estimator can estimate the power consumption of a RTL VHDL file by either using an activity file or by using the simulation activity file (.saif). The gate-level power analysis also computes the power consumption using the switching activity file generated via gate-level simulation. However, for either of these two methods one has to translate the SystemC testbench into an HDL testbench. This is rather undesirable, as extra effort has to be incurred in order to maintain the integrity between the SystemC testbench and the HDL testbench. A translator, which, is cycle accurate would be beneficial at this juncture. For analyzing power consumption of a SystemC based design, we have used the RTL power estimation methodology on the generated gate-level netlist. Gate Level estimates would provide higher accuracy, however would take longer and was not

implemented as yet due to time considerations. However, the trend is important and that is illustrated in Section 6.2.

The author views the following developments of to be useful for in allowing for ease of power analysis at the system level:

- Translator to obtain cycle accurate HDL testbench from SystemC testbench
- Generation of SAIF files from its SystemC simulation in the CoCentric System Studio.

Power analysis can also be carried out by using HDL wrappers and the co-simulation methodology illustrated in the CoCentric System Studio Manuals. However, these techniques have not yet been tested for power analysis using SystemC. Moreover for the analysis presented later, the relative trend is more important than the absolute and accurate measures.

The entire SystemC flow is capture in Figure 52.

Figure 52. SystemC Based Design Flow

4.4 Effecting Architecture Transformations

In this section we illustrate how the behavioral scheduling commands and behavioral compiler flags can be used to modify the implementation of the architecture. It is important to note that the SystemC code was not modified. It is also seen how the existing environment was used successfully to derive tradeoff analysis.

Architecture synthesis of a given SystemC architecture file can be controlled in three basic ways:

1. By writing RTL style code, and specifying exact intentions of the design in SystemC code. This method is useful when the architecture has been partitioned to a high level of detail.
2. By utilizing behavioral coding styles, the designer can explore the design by using one of the two scheduling strategies (*cycle_fixed* and *superstate_fixed*). Using compiler directives in SystemC comments, the designer can influence the compiler to synthesize the code in a particular fashion.
3. Further, *dc_shell* commands can be used to point at specific operations in the code and constrain or bind operations as is intended.

In this design exercise, behavioral coding style was employed to permit design exploration.

4.4.1 Optimal word-length for data-path optimization

As the literature survey suggests in chapter 2, word-length of data-paths and operations has an effect on the design parameters (area, power and performance). Thus, optimal word-length determination is important for system level architecture partitioning of the design.

In the case of the K-means algorithm the synopsys fixed point designer was used to arrive at the optimal word-length which provides a trade-off in terms of mean-squared error in the algorithm versus the max word-length required.

In Table 5 we note the following with respect to the mean-squared errors (MSE) in architectures with various word-lengths:

- 64 bit word-length with various integer and fractional permutations yield a minor variation in the MSE, however, the order of magnitude remains the same (10^{-11}).
- Moving from 64 bit to 56 bits keeping the number of fractional bits the same maintains the MSE at the level as that of a 64-bit word-length. This illustrates that the accuracy in the data-path is due to the fractional bits as a decrease in the integer bits did not yield a lower MSE.

- Increasing the integer word-length and decreasing the fractional number of bits whilst keeping the overall word-length the same (at 56 bits) yields a MSE which is 4 times higher.
- Further, when the number of fractional bits are lowered to 16 bits, the MSE further increases 15 times and slips 2 orders of magnitude (to 10^{-9})
- However, further reductions in the integer word-length from 40 to 32 bits, whilst maintaining the fractional bits at 16 has no significant effect on the MSE. Thus reducing complexity of the data-path without sacrificing MSE.
- However, when we drop the fractional bits to 14 and integer bits to 32, we note a drop in 7 orders of magnitude in the MSE.

Thus a word-length of 48 bits with 32 bit integer component and 16 bit fractional component offers a good tradeoff in MSE versus the architecture complexity that is obviously inferred from larger data-paths and operands for arithmetic and logic operations.

Wordlength	Integer	Fractional	MSE
64	40	24	1.565370E-11
64	44	20	1.757110E-11
56	36	20	1.757110E-11
56	38	18	8.299070E-11
56	40	16	1.316600E-09
54	38	16	1.316600E-09
52	36	16	1.316600E-09
50	34	16	1.316600E-09
48	32	16	1.316600E-09
46	32	14	3.964040E-02
46	30	16	1.641930E+02
48	28	20	7.411750E+02

Table 5. Word-length optimization based on accuracy

This analysis at the HDL level would be done using *generics* in VHDL and can result in large amounts of simulation time depending on the design size. However, with the use of SystemC based simulations compared against the golden C code algorithm results is much faster at evaluating the MSE trade-off for different word-lengths.

Further, using the fixed-point design tool of the Synopsys tool-set allows a quick SystemC fixed point library conversion which generates the first cut of the design with fixed point routines incorporated as shown in Figure 53.

Figure 53. SystemC fixed point designer reads in floating-point ANSI-C and writes out ANSI-C-code with all floating-point data types replaced by public open source SystemC fixed-point C++ data types

SystemC fixed-point designer allows simulation of hybrid code with floating-point and fixed-point representations, to allow full numerical analysis at any stage of modification of the ANSI-C-code. Thus, allowing a thorough analysis at various stages of modifications to arrive at the most accurate fixed-point representation.

Figure 54. Numerical analysis is allowed in floating-point, fixed-point and hybrid states

```

for(i=0;i<T;i++){
    sum=0;
    for(j=0 ; j < nmix ; j++){
        for(k=0;k<ndim;k++){
            diff[k]= FX_range( cepst[i][k]-mu[j][k] );
            q=ndim-1;
            ExpIndex=0;
            for(k=0;k<ndim;k++,q--){
                ExpIndex=ExpIndex+( FX_range(diff[k]) * FX_range(inv[j][k])
                                     * FX_range(diff[k]) );
                ExpIndex = ExpIndex * (-0.5);
                prob= FX_range(p[j]) * FX_range(mult[j]) *
                    FX_histF( exp(ExpIndex),0, 0.0000001, 50);
                sum = sum + FX_range(prob);
            }
            logsum=FX_range(logsum) + log( FX_range(sum) );
        }
    }
}

```

Figure 55. Inserting in-line commands to analyze operator variances

Figure 55 illustrates the insertion of two checkpoint directives to analyze the range for *inv[j][k]* and the *diff[k]*. Whereas, we analyze the histogram of *exp(ExpIndex)* to note the outlier data in a distribution of the data. The results of this analysis provides us with information on the first cut word-length information and fractional and integer partitioning. Using this information SystemC data classes for fixed data types (*sc_fixed* and *sc_ufixed*) can be used to perform simulations with constrained data objects as shown in Figure 56

```

for (i = 0; i < T; i++) {
    sum = 0;
    for (j = 0; j < nmix; j++) {
        for (k = 0; k < ndim; k++){

            diff[k] = sc_fix(cepst[i][k], 46, 8) - sc_fix(mu[j][k], 37, 8);
            q = sc_ufix(ndim - 1, 5, 5);

            ExpIndex = sc_ufix(diff[k] * inv[j][k] * diff[k], 68, 9);

            prob = sc_ufix(p[j], 27, -3) * sc_ufix(mult[j], 44, -44)

            sum = /* back-annotated */ sc_ufix(sum, 101, -48) +
                sc_ufix(prob, 101, -48);
        }
    }
    printf("score is %f\n", score);
}

```

Figure 56. Constrained Data-objects

Quantization Modes	Overflow Modes
Rounding to plus infinity	Saturation
Rounding to zero	Symmetrical saturation
Rounding to minus infinity	Saturation to zero
Rounding to infinity	Wrap-around
Convergent rounding	Sign magnitude wrap-around
Truncation	
Truncation to zero	

Figure 57. A rich set of quantization and rounding modes area available in the SystemC fixed-point data types

This unique feature of SystemC data types which allows fixed-point representations in the algorithm in standard ANSI-C++ code format with a rich set of quantization and rounding modes (Figure 57) allows a quick analysis on the appropriate fixed point format required by the architecture to have minimal accuracy vs. hardware complexity trade-off in a significantly shorter time-frame of evaluation. This is so, as the simulations are now at the C++ level which is much faster than RTL level simulations.

4.4.2 Pipelining based Optimizations

We shall now look at 4 basic transformations carried out on the code example in Appendix B. All the transformations were carried out without any change in the code. Compiler directives and *dc_shell* commands were used for design exploration.

4.4.2.1 An initial Estimate

To calculate energy required by the implementation for completion of a task, it is necessary to have an accurate measure of the time for which the design has to be active and running. The total run-time can be obtained by the product of clock cycles required for the complete operation and the clock cycle period. One of the ways by which the number of clock cycles can be measured is by using the *sc_time_stamp()* command in SystemC to obtain the total number of clocks required to run through the design. The simulation in Visual C++ gave an estimate of about 3.69 million clock cycles. This is however not the accurate figure for the implemented hardware as a calculation obtained from the hardware schedule summary (Figure 58) shown in Table 6. The same design after scheduling requires approximately 4.6 million clock cycles per iteration.

This discrepancy was traced to the conditional loops. During synthesis, the condition evaluation of a conditional loop is given the adequate number of clock cycles. This extra clock cycle(s) is not explicitly accounted for in SystemC simulation. From the extent of discrepancy it is already

obvious that the algorithm is quite iterative in nature. This shall be the focus of optimization in the design iterations that follow.

The script for compiling and scheduling this design is shown below:

```
dc_shell> define_design_lib ram_library -path /home/saurav/LowPower/Syn/RamLIB
dc_shell> analyze -f vhd1 ../Mem/RAM256x32_wrap.vhd -library ram_library
dc_shell> elaborate RAM256x32_wrap -architecture wrap -library ram_library
dc_shell> bc_enable_analysis_info = true
dc_shell> compile_systemc ../SystemC/kmeansSmallRam/kmeans.cpp
dc_shell> create_clock pClk -period 75
dc_shell> bc_check_design -io_mode superstate_fixed
dc_shell> bc_time_design
dc_shell> report_resource_estimate> ../SystemC/ report/V1_resource_estimates.rpt
dc_shell> schedule -io_mode superstate_fixed
dc_shell> report_schedule > ../SystemC /report/V1.sch
dc_shell> report_schedule -operations > ../SystemC /report/V1.opr
dc_shell> report_schedule -variables > ../SystemC/ report/V1.var
dc_shell> compile -map_effort medium
dc_shell> write -hierarchy -output ../SystemC/ DB/V1.db
dc_shell> write -hierarchy -output ../SystemC/vhd1/V1.vhd
```

The table below shows the clock cycle based requirement for ten iterations of the design.

Hierarchy	Loop Delay	Actual Delay [Pipeline]	Cascade Delay	Num. of Loops	Total Clk Cycles
F_LOOP1	3	3	0	128	384
W_LOOP2	7	7	2244704	10	22447110
F_LOOP2	2	2	8	16	160
F_LOOP3	1	1	0	8	8
F_LOOP4	6	6	368	6000	2244000
F_LOOP5	3	3	0	8	24
F_LOOP6	4	4	16	16	320
F_LOOP7	2	2	0	8	16
F_LOOP8	3	3	0	8	24
F_LOOP9	2	2	32	16	544
F_LOOP10	4	4	0	8	32
Total Clock Cycles for Design				22447494	clk cycles
Clock Cycle				7.50E-08	Seconds
Total Time for design				1.68E+00	Seconds
Area				64940	Gates

Table 6. Table V1

The hierarchy illustrated on the left side of the table is the same as the hierarchy labeled in the code of Appendix B. The loop delay is derived from the schedule summary report as shown in Figure 58. (Note – “lbl” has been dropped in the Tables in interest of column width).

Timing Summary		
Clock period 75.00		
Loop timing information:		
TOP	41 cycles	(cycles 0 - 63)
lbl F LOOP1	3 cycles	(cycles 1 - 4)
(continue) skip short branch 1		(cycle 4)
(exit) BB9 EXIT L81 365		(cycle 2)
lbl W LOOP2	34 cycles	(cycles 6 - 62)
(continue) skip short branch 1		(cycle 62)
(exit) BB13 EXIT L109 55		(cycle 7)
lbl F LOOP2	3 cycles	(cycles 7 - 10)
(continue) skip short branch 1		(cycle 10)
(exit) BB8 EXIT L117 327		(cycle 8)
lbl F LOOP3	1 cycle	(cycles 8 - 9)
(continue) skip short branch 1		(cycle 9)
(exit) BB7 EXIT L125 343		(cycle 9)
lbl F LOOP4	18 cycles	(cycles 12 - 34)
(continue) skip short branch 1		(cycle 34)
(exit) BB12 EXIT L149 151		(cycle 13)
lbl F LOOP5	3 cycles	(cycles 13 - 16)
(continue) skip short branch 1		(cycle 16)
(exit) BB9 EXIT L160 308		(cycle 14)
lbl F LOOP6	6 cycles	(cycles 18 - 27)
(continue) skip short branch 5		(cycle 27)
(exit) BB15 EXIT L198 232		(cycle 19)
lbl F LOOP7	2 cycles	(cycles 19 - 24)
(continue) skip short branch 1		(cycle 24)
(exit) BB10 EXIT L213 272		(cycle 20)
lbl F LOOP8	3 cycles	(cycles 29 - 33)
(continue) skip short branch 1		(cycle 33)
(exit) BB10 EXIT L283 189		(cycle 30)
lbl F LOOP9	6 cycles	(cycles 36 - 60)
(continue) skip short branch 1		(cycle 60)
(exit) BB8 EXIT L323 75		(cycle 37)
lbl F LOOP10	4 cycles	(cycles 37 - 59)
(continue) skip short branch 3		(cycle 59)
(exit) BB13 EXIT L331 109		(cycle 38)

Figure 58. Schedule

4.4.2.2 Pipelining lbl_F_LOOP7

From the table in Table 6 for design V1 above it is apparent that the FOR loop identified by lbl_F_LOOP7 (Figure 59) is a design bottleneck. It contributes most significantly to the 4.599 million clock cycles involved in a single iteration of K-means. Thus we try and pipeline the loop by the following *dc_shell* commands with the clock cycle period changed to 15ns:

```
dc_shell> loop_label1 = "TOP/lbl_W_LOOP2/lbl_F_LOOP4/lbl_F_LOOP6/lbl_F_LOOP7"
dc_shell> pipeline_loop loop_label1 -initiation_interval 1 -latency 5
```

This however fails because of the presence of a multi-cycle multiply in the *for loop*. This is indicated in the reservation table and in *bc_view*. Thus, the immediate remedy is to pipeline using the following constraints:

```
dc_shell> loop_label1 = "TOP/lbl_W_LOOP2/lbl_F_LOOP4/lbl_F_LOOP6/lbl_F_LOOP7"
dc_shell> pipeline_loop loop_label1 -initiation_interval 2 -latency 6
```

Note that, the *initiation_interval* has been modified to 2 due to 2-cycle multiplication and the *latency* to 6. The *latency* is modified to 6 as it has to be an integer multiple of the *initiation_interval*. We also change the clock period to 15ns. This is a representation of the fact

that the operations in the pipeline have a shorter critical path. Following the above modifications, we obtain results for design V2 as in the Table 7.

Figure 59. Lbl F LOOP7

Due to the reduction in critical path in the main loop of the design, the runtime of the algorithm reduces to about 20% of the original time, resulting in a throughput of 5X.

4.4.2.3 Pipelining lbl_F_LOOP7 with a single cycle multiply

By investing more area in the bottleneck operation of the design we arrive at a faster throughput of 1 clock cycle (with latency = 4) in the critical path of the design.

This was obtained by constraining the design as below to obtain Design V3:

```
dc_shell> loop_label1 = "TOP/lbl_W_LOOP2/lbl_F_LOOP4/lbl_F_LOOP6/lbl_F_LOOP7"
dc_shell> pipeline_loop loop_label1 -initiation_interval 1 -latency 4
dc_shell> bc_check_design -io_mode superstate_fixed
dc_shell> bc_time_design -fastest
```

In this case the throughput increases further to 7.5X

4.4.2.4 Pipelining loop lbl_F_LOOP8

Further, lbl_F_LOOP8 was pipelined using the following script:

```
dc_shell> define_design_lib ram_library -path /home/saurav/LowPower/Syn/RamLIB
dc_shell> analyze -f vhd1 ../Mem/RAM256x32_wrap.vhd -library ram_library
dc_shell> elaborate RAM256x32_wrap -architecture wrap -library ram_library
dc_shell> bc_enable_analysis_info = true
dc_shell> compile_systemc ../SystemC/kmeansSmallRam/kmeans.cpp
dc_shell> create_clock pClk -period 15
dc_shell> loop_label1 = "TOP/lbl_W_LOOP2/lbl_F_LOOP4/lbl_F_LOOP6/lbl_F_LOOP7"
dc_shell> pipeline_loop loop_label1 -initiation_interval 1 -latency 4
dc_shell> loop_label2 = "TOP/lbl_W_LOOP2/lbl_F_LOOP4/lbl_F_LOOP8"
dc_shell> pipeline_loop loop_label2 -initiation_interval 3 -latency 6
dc_shell> bc_check_design -io_mode superstate_fixed
dc_shell> bc_time_design -fastest
```

```
dc_shell> schedule -io_mode superstate_fixed
```

The results obtained show that throughput further increases to 7.85X. It can thus be concluded that maximum gains are obtained when optimizing a loops, which is in the critical path of the design.

4.4.2.5 Energy analysis

It can be seen that SystemC based architectures can be quite effortlessly transformed to give higher throughput architectures. The summary of the various architectural transformations resulting in architectures V1 to V4 is shown in Table 7.

It can be easily seen in Table 7 that the amount of time required to complete one iteration of K-means is in the order of $V4 < V3 < V2 < V1$. This represents the increasing throughput in the order $V4 > V3 > V2 > V1$. However, it would be interesting to obtain the power reports $\{P1, P2, P3, P4\}$ corresponding to the designs $\{V1, V2, V3, V4\}$. This would enable us to calculate the energy requirement as $\{E1, E2, E3, E4\}$. The impact of throughput and area increase is anticipated to result in degraded power consumption $\{P4 > P3 > P2 > P1\}$.

However, it can be hypothesized that:

if $P2 < (5 \times P1)$; then $E2 < E1$

if $P3 < (7.5 \times P1)$; then $E3 < E1$

if $P4 < (7.85 \times P1)$; then $E4 < E1$.

The above assumptions cannot be termed absolutely impossible, as the area increase is not explosive (10-20%). However, it can be judged that the pipelined stages would now consume more power, since they are clocked at 5X the original clock frequency of the design. Initial results using power estimator did look favorable to the above hypothesis, however, more concrete analysis has to be done based on simulation based power analysis to confirm the results.

Figure 60. Illustration of Effective energy consumption

It can then be used to illustrate the point that energy consumption of a device can be reduced while maintaining throughput. This is assuming that we apply the necessary clock gating techniques and sleep logic design methodologies to the output netlist. Without reducing power consumption to only the leakage component would not be beneficial, as useful energy would still be wasted as a function of peak power consumption. This is in keeping with the fact that the energy consumption reduction is due to the negligible leakage power consumption of a switched off device. This property would be highly beneficial to battery-operated embedded devices. Noting the fact that leakage power consumption does seem to be on the rise as we enter deep sub micron design methodologies, such design techniques will have to be tweaked to suit the challenges of emerging design environments.

	Area Gates	Time [1 iter.] Clk. Cycles	Clk. Period	Total Time	Throughput Reference
V1	64940	2.244M	75ns	1.68E+00	X
V2	56635	2.247M	15ns	3.37E-01	5X
V3	71318	1.477M	15ns	2.22E-01	7.5X
V5	72554	1.429M	15ns	2.14E-01	7.85X

Table 7. Optimization Results at a glance (1M = 1 million clock cycles)

It is also important to note that different application domains may show a different quantum of gain in energy conservation by applying the technique of higher peak power consumption for lower time. This method is most applicable to designs, which are iterative rather than a one-off computation. Further, the need for System level design languages to be used for architecture exploration is clearly demonstrated.

4.4.3 Parallel Computation & Memory Partitioning

It is also shown in past research papers [2], that increasing performance by utilizing multiple parallel elements can also aid in reducing energy consumption. Gains in energy efficiency using voltage scaling due to the reduced critical path were also illustrated. In this section we shall discuss the results obtained by replicating the data-path 4 times to increase throughput.

Parallel computation units for data-paths are the other conventional approach to increasing performance. This usually entails higher area. The penalty involved however is completely justifiable due to the high increase in throughput. However, parallel computational units demand higher memory bandwidth. Moreover simultaneous parallel memory access also becomes a necessity. This is illustrated in Table 8. Note that even though area has increased almost four-fold due to data-path and memory, the throughput has also gone up to 4X of original serial architecture (P1 versus P0). Further, the time averaged power consumption is also respectable considering the throughput.

This design example being no exception, most current ASIC's are known to have on-chip memories in high capacities. The motivations for having on-chip memory are many – lower access delay, possible area decrease from tighter integration and lower power consumption.

However, this also opens up the opportunity to perform memory architecture exploration if the data-path so demands. In most design cases, the memory might require to be split into several smaller blocks to provide for the parallelism in the data-path. However, this results in increased address decoding circuitry and overheads in the form of fixed area and power costs for implementing memories.

Table 8 below clearly illustrates that using a wider line size in the memory can result in immediate power consumption reduction. Further, complete parallel implementation of the data-path along with the memory also consumes much less averaged power and provides higher throughput in comparison to the serial implementation of the same.

	Architecture Par/Seq	Width of Data RAM [On-Chip]	ThroughPut	Area [Gates]	Power MW
P0	Sequential	8 bits	1x	2800	25.958
P1	4-way parallel w/ 4 RAM banks	8-bits	4x	10142	97.122
P2	4-way parallel w/ 1 RAM bank	32-bits	4x	8749	87.676

Note - RAM sizes: P1: 64x8; P2: 64x32 [depth x word size]

Table 8. Tabulation of results for various implementations with different word sized RAMS

Figure 61. Memory Architecture with (a) 8-bit word and (b) 32-bit word sizes

4.5 Comparing VHDL and SLDL based design flow results

Moving up the abstraction ladder in design languages has a tradeoff that the architecture expressions become more inefficient and ambiguous to support compiler based option evaluation for implementation. Thus, it is important to evaluate and appreciate the discrepancy between system level design languages and hardware description languages. The architecture example used in section 2.5.1 is implemented in this section in the two different language options and then synthesized using the Synopsys Design Compiler to the same cell based libraries to evaluate the impact of the two languages.

4.5.1 Results on system level partitioning using VHDL

Before probing whether partitioning using high-level system design languages such as SystemC can affect the power consumption figures, the author first worked the design-partitioning example in [2] using VHDL. The results obtained are shown in Table 9 below.

The designs offer the same throughput for lower and lower clock frequencies as we go down the rows of the table. Also, moving from left to right of the table gives us the power requirement due to operation at lower clock frequencies.

Architecture Variant	VHDL Coding Based RTL Level Analysis		
	@ 5ns	@ 10ns	@ 15ns
Serial	3.35	1.685	0.841
Parallel	6.745	3.403	1.693
SerPipe	4.273	2.163	1.081
ParPipe	8.617	4.329	2.168

Note - All figures are in mW

Table 9. Power Requirement

4.5.2 Results on system level partitioning using SystemC

The use of high-level languages like SystemC can minimize lengthy design cycle time. SystemC bundled in the CoCentric System Studio provides the designer with the implications on power requirements at the system level. This allows a high-level design architect to introduce power-efficient architectural optimizations without having to resort to detailed implementation stages of the design cycle. In this manner, the power budget requirements of complex designs could be better managed at the higher level.

Figure 62 and Figure 63 below show the overview of SystemC flow in the Synopsys tools environment and the specific flow for estimating power using Synopsys tools.

Figure 62. Low Power flow using SystemC in CoCentric System Studio

Figure 63. Power Estimation Methodology and Data Flow

Figure 64. SystemC Compiler Input and Output Flow for RTL synthesis

The flow in Figure 64 illustrates the SystemC to RTL flow using Synopsys environment. Note that the libraries provided by the vendors' plays an important part in correct power estimation in the front-end design flow. The fact that the results in the table above follow the trend as expected in [2], we can deduce that the libraries are reasonably complete in power characterization. Further, we should also note that the interconnect power cannot completely be characterized at RTL or even gate-level design. This is because routing is undecided at the front-end design stage. This can be estimated to reasonable levels of accuracy at 0.35-micron technology used for the simulation above. However, with the use of deep submicron technology (< 0.18 micron), this divergence increases and the need to have routing information arises for power analysis accuracy at the front-end design stage.

With the use of SystemC in the design flow we have an added compilation step. This comprises of translation of Open Source SystemC code to VHDL or Verilog code. The power analysis then continues as in the figure earlier. However, to evaluate the effectiveness of the translation and the ease of writing code in SystemC, the same design as earlier was ported to SystemC. The results were also encouraging as seen in Table 10 below.

Architecture Variant	SystemC Coding Based Gate-Level Analysis		
Clock	5ns	10ns	15ns
Serial	3.535	1.779	0.883
Parallel	11.851	5.92	2.900
SerPipe	5.899	2.947	1.481
ParPipe	11.851	5.941	2.974

Table 10. Impact on power consumption using SystemC for architecture partitioning

4.5.3 Summary of comparative results

System level design languages such as SystemC do obviously afford higher code density than hardware description languages. This is so due to the higher level of abstraction at which functional units are expressed in code. The resulting lower code complexity leads to better management of the design and much better communication amongst the design teams. However, it is the advantages in simulation time that allows for rapid design changes and trade-off analysis. This aspect is illustrated below.

4.5.3.1 Accuracy of estimation vs. simulation time

The differences in the power estimates can be attributed due to the gate-level analysis in this step. This suggests that gate-level analysis presents an optimistic view of power consumption.

However, this conclusion has to be substantiated with further experimentation. The tables above follow the trend as expected in [2]. The discrepancies are due to the different libraries used to characterize the power information. Further, the almost disciplined and uniform change in power consumption figures is due to the absence of interconnect power information. This cannot be helped at RTL level of power analysis. At Gate-level too most tools just use wire-load models of interconnect along with empirical values to project an estimate.

It is also noted that VHDL and other HDL based simulations do take up much more time than equivalent SLDL based techniques. The time complexity is several orders of magnitude higher as illustrated in the figure below.

Figure 65. Depending on the abstraction of design implementation, simulation time for testing and debugging for SoC design can vary

Thus we note that the two synthesis results mirror the same trends as is the classical analysis in section 2.6.1. This imbibes confidence in applying system level techniques to more complex designs.

Figure 66. Architecture Design Space Exploration using SLDL

Figure 67. Suggested Top-Down Feed Through of Power Budget Information to Block Level Optimization

4.6 Summary

Through the various examples and design iterations discussed in this chapter, it has been convincingly demonstrated that there is a strong urge to partition the design at system level where energy or average power consumption and throughput is of prime importance. The case for the use of a system level design language is also illustrated by showing that SystemC can be used to effectively explore the design space for energy consumption reduction. It was noted here that more versions of the algorithm could be effectively analyzed, thereby facilitating a more

effective design exploration. This was possible primarily due to the faster speed of simulations possible at the system level, coupled with the efficiency of coding in SLDL over HDL. However, it is evident that a system level design language alone cannot be useful enough. A complete system level design environment, such as SystemC, to tradeoff the performance (time), area and power/energy is absolutely essential.

The potential of the combination of a complete system level design environment along with the system level design language is clearly demonstrated to be very effective to facilitate power analysis. The usefulness in the ability to perform complete exploration along the three axes of design space cannot be ignored. It is noteworthy that the power estimation in a high-level design environment is often not a very good estimate in the “absolute” sense. However, it does provide the “relative” trend in power and energy efficiency for different architecture options. Yet another observation is in the fact that there are distinct advantages in approaching the problem of power-closure in a top-down (Figure 67) manner rather than bottom-up. The design architect should be able to readily compare and contrast different architectures of an implementation and decide on power budget for each block. This information along with the area and timing constraints can then be passed top-down in the design flow to the block level designers. The impact of such an approach would be a more seamless closure of a complex design.

With the emergence of deep sub-micron designs, one can note that the projection of the wire load capacitance and power consumption of interconnect may not be accurately represented. This will require greater maturity on the part of the power estimation provider to factor in the non-linear scaling challenges posed by the deep sub-micron trends for interconnect technologies. It was also noted that the underlying principles of the optimization steps are very similar to that of the performance extraction process of a design. The difference is in the perspective of the optimizations. This demonstrates the need to integrate more tools at the system design level to facilitate meaningful architecture explorations. For example, a power report could be made available as an option even in the SystemC compiler rather than having to go down the design flow to Power Compiler. This will allow the design architect to be prompted with appropriate decisions in optimization directions to be taken. Thus, performance, area and power consumption shall hence become the criteria for optimization even at the highest level of design. The conclusions from the two chapters thus indicate that a balance is required towards system level and HDL descriptions for a design exercise. The next chapter shall elaborate on algorithm innovation in a system level environment towards inherently efficient options for implementation.

Chapter 5 System Level Analysis for Hardware Architecture Optimization

Having delved in detail into SystemC and HDL based design flows in an independent manner; we realize that system level algorithm analysis cannot be done effectively in any one of the two abstractions alone. This is due to the complementary strengths and weaknesses of the two levels of abstraction in design languages and tool support. Thus a hybrid methodology is required. Further, algorithm analysis in an isolated manner without any posing system level constraints is also short-sighted and does not lead to an effective architecture design space exploration. Thus, there is a need to extend SLDL and HDL based design flows to work in tandem with the selective introduction of system level profiling and simulation tools that is specific to the system being designed. We shall experiment the selective introduction of system level simulation tools into the system level design environment based on SystemC with the aid of a design example which requires wide ranging accuracy and design cost analysis.

In this chapter we apply the broad framework to instruction set cache hierarchy design to prove the efficacy of the hybrid approach towards designing digital circuits for a particular set of constraints. In this instruction set cache hierarchy design we shall use various tools and methodologies to arrive at various design options that meet the varied design optimization constraints for a prediction algorithm which increases the efficiency of the instruction cache. System level modeling for instruction set streams, digital logic design parameter estimates using SystemC, and detailed HDL based implementation shall be used to arrive at various points in the design space to meet different requirements of prediction accuracy and cache performance. This chapter utilizes instruction level simulation of hits and misses and algorithmic issues that are derived from the simulations. The results from these system level simulation help innovate a prediction algorithm towards varying design requirements for a predictive filter cache based instruction set hierarchy. Having established the various algorithms for the different design options, and evaluated the hardware implications at the system level using SystemC, we shall then highlight the in-depth implementation costs using HDL based implementation. The experiments will prove that a hybrid approach is necessary to converge quickly on the design requirements.

5.1 Cache Memory for Increased Performance

Embedded systems are often designed with tight energy, cost and performance budget. Energy consumption is not only a concern for battery operated portable embedded systems, but also for

server devices. Higher energy consumption leads to increased heat dissipation. This necessitates more expensive packaging and cooling technology, which, increases cost, and decreases reliability of products in all segments of computing market from portable systems to high-end servers.

Various advancements in VLSI technologies have been catapulting processor speed and DRAM capacity dramatically. However, the disparate growth rates between processor and main memory has introduced a large and still growing performance gap. By improving both the clock speed as well as instruction level parallelism (ILP), the processor performance has been improving at a rate of 60% per year [39]. On the other hand, the access time for DRAMs has been improving at a rate of less than 10% per year.

The inability of memory systems to cope with faster processor speeds results in poor total system performance in spite of higher processor performance. Cache memories have long been playing an important role in bridging the performance gap between high-speed processors and low-speed off-chip main memory. There has been a lot of research done to improve cache performance and many high-performance cache architectures have been proposed [39]. However, the processor-memory performance gap is still increasing.

The growing mobile market requires not only high performance but also demands low energy consumption. One of the uncompromising requirements of portable computing is energy efficiency, particularly due to the finite nature of battery life. With deep submicron (DSM) process technology and high clock speeds, power consumption is becoming a major design issue even in cases where unlimited energy is available to power the device. In high-performance architectures, like server processors where infinite energy source is available, heat dissipation becomes a concern due to the high rate of energy consumption in the devices. Thus, there exists a need to realize energy-efficient high-performance memory architectures for all types of processors.

To alleviate the performance gap between memory systems and processors the trend is to invest the growing transistor density in increasing the cache capacity. Raising the cache capacity reduces the frequency of off-chip accesses due to improved cache hit rates. This approach seems to be useful in that the energy dissipated for driving external IO pins is reduced. Cache memories have been traditionally used in liberal quantities to boost the performance of processor architectures. However, this will increase the on-chip energy consumption due to frequent accesses to the larger cache memory. For instance, the on-chip cache for the Strong ARM SA110 consumes approximately 43% of the total chip power [40].

With Deep Sub Micron (DSM) process technology, the number of transistors devoted for cache memory is also increasing. This results in even higher on-chip power consumption. This provides a firm motivation for power and energy aware design of instruction and data caches for embedded processors.

Dynamic power reduction in cache hierarchy has been the focus of research for quite some time. Some of these dynamic power reduction techniques realize power optimization by predominantly accessing energy efficient auxiliary cache more often than the power inefficient L1 cache. Though future process technologies will allow bigger instruction caches to be fabricated on-chip, energy consumption and access time are likely to continue to be the limiting factors. This inability to achieve notable energy reduction using current techniques highlights the need for a memory hierarchy design that maximizes the instruction hit rate, while tackling the access time penalty and dynamic energy.

It is well understood that the dynamic or switching energy dissipation in instruction cache is typically higher than that for data cache. For instance in x86 architecture, simulation with SPECInt92 reveals that only 34% of the instructions access data cache [39]. Moreover, 70 to 90% of the power consumption in CMOS technology is attributed to switching activity [41]. In this section, we propose a predictor-based hierarchy to reduce the overall energy–delay product in the instruction cache.

5.2 Techniques towards low power cache memory hierarchy

Various small auxiliary cache hierarchies have been proposed to tackle the problem of energy dissipation in instruction cache hierarchy. A small cache closest to the CPU, in addition to the usual cache hierarchy is employed to decrease the overall access energy consumed by instruction caches. In such architectures the energy efficient auxiliary cache is accessed more often, hence resulting in overall energy savings. These auxiliary caches utilize the immediate spatial and temporal locality within the small loops of the application. Line Buffer, loop cache [42] and FC [43] are examples of such cache memories, which perform in slightly different ways. These mechanisms exploit the energy savings possible when a small cache is placed between execution core and main cache.

Loop cache [42] is a small auxiliary energy efficient cache that detects loops dynamically and enables caching. However, the loop cache can cache loops only if their size is smaller than the loop cache. In addition, there exists the limitation that the instructions in the loop should not contain any control transfer instruction. Hence, it requires compiler and architecture support.

Architectures with predicated instruction will be more suitable for the loop cache. Even then, the resulting energy-delay product is generally worse than what is possible with FC.

Filter Cache (FC) [43] hierarchy, an auxiliary cache based instruction memory hierarchy has been shown to achieve 58% power saving at the cost of 21% performance loss for MediaBench [44] benchmark suite. Though the energy-delay product achieved here is substantial, the performance degradation might not always be acceptable. Hence, the predictive FC based on Next Fetch Prediction Table (NFPT) was proposed in [45] and [46].

In such predictive FC architectures, a predictive parallel access path exists between the cache memories and execution core. Based on the prediction, the FC or the main cache is accessed. If the prediction is to access the FC and results in a miss (in the FC), then an extra clock cycle is wasted in accessing the main cache. Thus an incorrect hit prediction degrades performance. On the other hand, if the prediction is to access the main cache and the instruction is present in FC, then this will result in increased access energy, since possible energy reduction is not utilized. Thus an incorrect miss prediction results in loss of energy. Hence, the energy consumption reduction and the performance improvement depend largely on the accuracy of the prediction algorithm.

Incorporation of the predictor leads to four operating scenarios. The following considers the latency tradeoff for a prediction-based hierarchy.

(i) Predicted FC Actual FC: The prediction was to access FC, and the instruction is found in the FC. In this case the power savings expected with FC is obtained and at the same time, there is no performance penalty.

(ii) Predicted L1 Actual L1: The prediction was to access the L1 cache, and the instruction required is absent in the FC. In this case, since the predictor has correctly predicted L1, the performance will be equal to the case where FC is not present.

(iii) Predicted FC Actual L1: In the case of an incorrect hit-prediction in FC, performance is sacrificed as the data is now fetched from L1 cache or from a memory block that is further away in the hierarchy.

(iv) Predicted L1 Actual FC: In the case of an incorrect miss-prediction in the FC, where the data is actually in the FC but is predicted otherwise, possible savings in energy and increased performance are sacrificed.

Ideally the predictor should encounter zero number of cases (iii) and (iv) stipulated. A predictor which is capable of this is termed an ideal predictor in our discussion. For an ideal predictor the average number of memory access cycles is equal to the base case where FC is not present, and the access energy savings achieved is better than the case with (hierarchical) FC. In the Section that follows, we discuss the existing next-fetch-prediction-table-based predictor prior to introducing the new branch-prediction-based scheme for FC-based instruction cache hierarchy.

5.3 Prediction algorithms

5.3.1 Next-fetch-prediction table

The next-fetch-prediction table (NFPT) ([45], [46]) prediction introduced for predictive access to the FC was shown to achieve energy savings of 31.5% for the instruction cache with SPEC95 benchmarks for a wide-issue superscalar processor by limiting the performance degradation to 1.3%: The NFPT predictor predicts whether the instruction to be fetched will be present in the FC or not, in the previous cycle. Based on this prediction, the fetch unit will attempt to fetch the instruction either from the FC or L1 cache.

If the tag of the current fetch address is equal to the tag of the predicted next fetch address based on the previous control path (i.e. based on the next fetch entry in the NFPT for the corresponding cache line), then a hit in the FC is predicted. Otherwise the main cache is accessed.

If the control path falls within a small loop, the tags of the current fetch address and the predicted next fetch would be equal, and only if it is part of a small loop would it remain in the cache until the next iteration, or else it would have been flushed out. The drawbacks of the NFPT and the motivation for devising an alternative approach follow.

5.3.2 NFPT Based Approach

NFPT based predictor was introduced for predictive access to the FC. NFPT limits the performance degradation of a FC based hierarchy to 1.3%, at energy savings of 31.5% for the instruction cache (with SPEC95 benchmarks for wide issue superscalar processor). NFPT table based predictor predicts whether the instruction to be fetched would be present in the FC or not in the previous cycle itself. Hence, based on this prediction, the fetch unit will attempt to fetch the instruction either from the FC or L1 cache. NFPT based predictor captures the locality within small loops to predict whether the next instruction is expected to be in the FC. To predict, it utilizes the fact that the same control path might be taken repeatedly.

The algorithm for prediction is as follows: if the tag of the current fetch address is equal to the tag of the predicted next fetch address based on the previous control path (i.e. based on the next fetch entry in the NFPT for the corresponding cache line), then a hit in the FC is predicted. Otherwise, the main cache is accessed. Since the FC structure tends to be of direct mapped, the tags corresponding to both the current fetch address and predicted next fetch address would be equal for a case of a small loop. Also, only the tags of a small loop would remain in the FC until the next iteration due to the small size of FC.

5.3.3 Drawback of NFPT Scheme

Though NFPT is applicable in most cases, the motivation for an alternative scheme is based on the hypothesis that the NFPT algorithm will fail for loops with a function call to a different memory space. In such circumstances, even if the FC can hold all the instruction lines, the NFPT based predictor will predict a miss as the tags are different. This is the primary motivation to experiment with the two level binary predictor commonly used to predict the branch direction.

5.3.4 Motivation for alternative prediction Scheme

Though the NFPT has been shown to be a useful prediction technique, it is ineffective whenever the loops whose function calls disrupt the dependency on the tags to facilitate the prediction process. This is because, even if the FC can hold all the instruction lines, the NFPT predictor will predict a miss, as the tags will be different.

Moreover, the NFPT requires that tags must be maintained in a table, which can be costly in VLSI. Hence, there exists a need to identify a suitable predictor whose hardware complexity is not dictated by the size of FC. The need to minimize the energy consumption and improve prediction accuracy has been the primary motivation to experiment with the two-level binary predictor which is commonly used to predict the patterns of hit miss stream of the instruction line access. This is described as follows.

5.3.5 Pattern history based predictor

The two-level adaptive predictor has been researched extensively for branch prediction and numerous variants of this have been proposed [47]. Studies have shown which predictors and configurations best predict the branches in a given set of benchmarks. We focus on one such branch predictor known as two-level adaptive branch predictor for predicting a hit in the FC. These binary predictors have found widespread applications as explained in [48]. In our setup, we use the ‘GAg’ [47] scheme due to its higher prediction accuracy and simplicity. The two-level predictor consists of a shift register (SR) to record the hit “equals to” miss patterns of

instruction line accesses and the pattern history table (PHT) consisting of 2-bit saturating counters. These counter values are used to predict (hit/miss) the next instruction access. When the previous N outcomes of hit=miss are known then by indexing the PHT with the SR value we can predict whether the next one is going to be a hit or miss. The size of the PHT ($2N$ entries) is determined by the size of the SR (N bits) used. The finite state machine (FSM) shown in Fig. 1 depicts the prediction process.

Figure 68 illustrates the algorithm employed in the prediction process. Drawing from the insight provided by branch prediction research, we assume that the next instruction fetch address will be the current fetch address plus '4' (here each instruction is 32-bits wide and we assume that most branches are not taken so that next instruction address is current instruction address plus four bytes). We access the predictor only when cache line changes are encountered. Hence, we predict a hit in the FC if the cache line being accessed is unchanged. In the event of a change, we access the PHT corresponding to the SR value, and the prediction of a hit or a miss in the FC is made depending on whether the PHT value reads above the preset threshold or below. For a two-bit saturating counter, a threshold value of 2 is used. When the outcome of the prediction is known, the SR and PHT values are updated accordingly, as in the normal branch predictor scheme.

Figure 68. Prediction Algorithm

Figure 69. Finite state machine of 2-bit saturating counter

5.4 **Pattern Predictor**

5.4.1 **Experimental Setup**

Due to the necessity in providing the system level parameters for effective architecture design space exploration, SimpleScalar is chosen for this purpose. The SimpleScalar [49] tool set with

ARM instruction set architecture (ISA) was used for these experiments. To measure the per-access energy, CACTI version 3.0 [50] was used with parameters set to 0.18 μm technology. The processor model simulated was a single-issue processor typically found in general-purpose embedded processors where instructions are fetched from the cache hierarchy every clock cycle; if an instruction cache access results in a miss then it will result in a pipeline bubble. Also, the first level instruction access is in the critical instruction access path. A 32-kbyte L1 cache with 32-byte line size and four-way set-associative, organized as two sub-banks was used in all the experiments as the base case so that the setup is compatible to the instruction cache used in high-end embedded microprocessors. A set of benchmarks from MediaBench [44] and MiBench [51] were selected as they characterize embedded applications working sets for high-performance embedded processors. All the applications were compiled with gcc 2.93 and were run to completion. The conventional filter cache (CON), NFPT (NFPT predictor was simulated with 4-bit next fetch address as provided in the literature) and the pattern predictor cache configuration are implemented with the SimpleScalar tool set and simulated. All the FC configurations used in these experiments are with 16-byte line size and direct mapped configuration.

Figure 70. SimpleScalar toolset overview

5.4.2 Pattern Predictor Performance Analysis

5.4.2.1 Prediction accuracy

Prediction accuracy has the highest impact in the energy savings and the average memory access latency. Thus the prediction accuracy of the pattern prediction algorithm is compared with NFPT for 256 and 512-byte FC sizes. Then, investigations on the ability of the pattern predictor to cater

for varying requirements of the application, in an attempt to improve the prediction accuracy, are performed.

The pattern predictor with 32 entry PHT was employed at first for 256 bytes and 512 bytes of FC memory as suggested in ([45], [46]). Figure 71 compares the prediction accuracy of NFPT and pattern predictor for 256-byte and 512-byte FC sizes.

As seen from Fig. 62, for 256-byte FC, the pattern predictor always results in higher prediction accuracy than the NFPT predictor. For the 512-byte FC the pattern predictor results in higher prediction accuracy except for ADPCM decoder, ISPELL and SUSAN. For the benchmarks ADPCM decoder, ISPELL and SUSAN, the performance of the pattern predictor is comparable with NFPT. However, an anomaly is also visible in the form depreciated performance with increased FC size for a pattern predictor with some benchmarks (e.g. GS-SCRIPT DEC and JPEG-DEC). The relationship between FC size, PHT size and the prediction accuracy are further investigated in the following Sections

Figure 71. Prediction accuracy for 256 and 512 byte FC

Figure 72. Prediction accuracy with different PHT sizes

Having established the overall superiority of the pattern predictor over NFPT, we investigate the effect of varying PHT size on prediction accuracy. Barring a few exceptions, Figure 72 shows that, in general, the prediction accuracy increases as we increase the PHT size. Based on our findings, we can safely assume that the prediction accuracy with 8- and 32-entry PHT is generally comparable with and in most cases better than that obtained using the NFPT prediction scheme. We choose the PHT size to be 32 to generate all the results in the subsequent Sections.

5.4.2.2 Energy–delay product

The energy–delay product provides for unified quantification of the effectiveness of the architecture. This metric however, depends other metrics such as filter cache hit rate, filter cache performance index and normalized delay.

5.4.2.2.1 Energy–delay product

The impact on the incorrect predictions towards both the FC and L1 will affect the energy–delay measures. This is because incorrect predictions towards FC will incur additional access cycles while incorrect accesses to L1 will require additional access energy due to notable disparity between FC and LI sizes. Figure 73 compares the normalized energy–delay product [43] for 256-byte and 512-byte FC with both NFPT and pattern predictor. The configuration with 256-byte FC and pattern predictor results in better normalized energy–delay product for all the benchmarks. The same could be said for the combination of 512-byte FC and pattern predictor except for ISPELL and SUSAN. However, the difference in energy–delay product with ISPELL and SUSAN is negligible, as seen from Figure 73.

Figure 73. Normalized energy-delay product for 256 and 512 byte FC

Figure 74. Normalized energy-delay product

5.4.2.2.2 Optimal FC size for energy–delay product reduction

Figure 74 shows the normalized energy–delay products of pattern predictor and NFPT for various FC sizes. Here, the energy delay product of value 1 refers to lowest energy–delay product among the set. It is evident that the optimal energy–delay product for a given bench mark is governed by FC size. This implies that the size of FC can be tailored for a given application. Our investigations show that, in general, FC sizes range between 256 to 512 bytes for the benchmarks experimented. Detailed analysis of the comparison with further parameters can be found in [58].

5.4.3 SystemC based hardware architecture of the pattern predictor

Figure 75 shows the simplified implementation of the pattern predictor algorithm. It employs an AND gate to detect whether the current fetch address and the predicted next fetch address (PC+4) maps to the same cache line. Based on this, the next fetch prediction source is selected. The register 'last line' stores the last instruction line address accessed. While fetching a new instruction, the PC is compared with the 'last line' to detect line change in the instruction cache. In the event that an instruction line change is detected, the SR and PHT are updated. NFPT implementation is explained in [45]. In addition to the NFPT table, there exists an additional register, to hold the last line, and a 4-bit comparator. As an initial estimate of relative hardware cost, the two predictors were synthesized using the behavioral compiler of synopsys on 0.35 micron Avant! Libra Passport library. The architectures were described using SystemC [36] and compiled to gates using the CoCentric SystemC compiler [52]. Though the area measures produced by hardware synthesis from SystemC are not optimal, they serve to clearly illustrate the relative hardware cost of the two predictors. Table 11 shows the hardware resource requirements for implementing NFPT and pattern predictor for various cache sizes and configurations. It is clear from Table 11 that an application-centric selection criterion for determining the PHT size may provide for an area-efficient implementation of the pattern predictor. Unlike the NFPT, which is dependent on FC size, the pattern predictor is application-dependent. This distinct feature of the pattern predictor provides a mechanism to propose application centric configurations that could be tailored to satisfy area and energy–delay product trade-offs.

Figure 75. Simplified architecture of prediction algorithm

Predictor	Area (in gates)
NFPT-256* byte	1560.6
NFPT-512* byte	2798.3
NFPT-1024* byte	5218.6
Pattern-8 entry PHT	702.1
Pattern-32 entry PHT	1802.1
Pattern-128 entry PHT	6174.4

*NFPT predictor is FC size dependent, hence hardware complexity is shown with respect to FC size. Complexity (in gates), does not include hardware cost of FC

Table 11. Hardware area cost for predictor configurations without FC

5.4.4 Summary of experiments

The pattern predictor prediction scheme clearly demonstrated that the proposed prediction technique is more energy-efficient than NFPT prediction schemes. The experiments confirm that when compared with NFPT implementations the proposed predictor leads to an average energy–delay product reduction of 3.39% and 2.39% with 256-byte and 512-byte filter cache, respectively.

From the quick SystemC based modeling of the hardware architecture we also deduce that unlike the NFPT predictor, the pattern prediction approach can be configured independently to filter-

cache size to maximize the prediction accuracy. For example, for a typical filter cache of size 256 bytes, a relatively small PHT (8 to 32 entry table) is sufficient to contain the patterns exhibited by instructions lines. Moreover, the fact that the pattern predictor can be efficiently realized in VLSI further justifies its incorporation for use in filter-cache structures. Also, for a given configuration of the pattern predictor and application, an optimal size for the FC can be proposed by examining the normalized energy–delay product.

5.5 Static Pattern Predictor

Predictive filter cache based instruction cache hierarchy has been shown to effectively reduce the energy-delay product. In this section, a simplified pattern prediction algorithm is proposed for the filter cache hierarchy.

5.5.1 Static Pattern Predictor for Prediction of FC access

In the earlier sections it has been shown that the two-level adaptive predictor is effective in predicting hit/miss for the filter cache hierarchy. This dynamic predictor results in higher energy-delay product reduction compared to the NFPT scheme. However, in this dynamic scheme the PHT must be updated for each instruction line access, thereby leading to a relatively more complex architecture that must cater for PHT updates at run time. This will become an issue, particularly in the presence of a large PHT. This has motivated the need to eliminate the incorporation of saturation counters that require constant updating at run-time without compromising the reduction in the energy-delay product.

This static scheme is based on the two-level prediction scheme. The intuition for the static predictor comes from the small size of the FC. That is, FC being small in size will hold only small loops till the next iteration. If the loop is small and can be completely contained in the FC, it will have continuous hits while iterating in the loop, and hence continuous ones. If the loop is larger than the FC then the hit and miss patterns of instruction lines will be one's changing to zeros and then changing to one's depending on the loop size and filter cache size. Therefore, if the previous patterns are continuous misses then the probability of next instruction access from the cache being a miss (zero) is very high. The same is true for continuous misses. The hit/miss patterns of the instruction cache line accesses in the FC were analyzed for various benchmarks with different FC sizes. It became evident that when the previous N instruction line access's are continuous hits (or misses) then the next access will most probably also be a hit (or miss). For instance if the previous instruction line accesses are continuous misses the probability that the next access will miss is also extremely high. It is also possible to make useful static observations

of the probability of other hit-miss pattern sequences. Although this largely depends on the program, it can be generalized to implement a simplified approach hereafter known as the 'Static Pattern Prediction' scheme.

Pattern	Next Prediction	Pattern	Next Prediction
00000	0	01000	0
00001	1	01001	1
00010	0	01010	0
00011	1	01011	1
00100	0	01100	0
00101	1	01101	1
00110	0	01110	0
00111	1	01111	1
10000	0	11000	0
10001	0	11001	1
10010	0	11010	1
10011	1	11011	1
10100	0	11100	0
10101	1	11101	1
10110	1	11110	0
10111	1	11111	1

Table 12. Proposed Static Prediction Table

The heuristic intuition here is that a Static Pattern Table (SPT) can be used as effectively as a dynamic PHT based predictor. Profiling for patterns validated this and based on the patterns exhibited by the benchmarks, a constant static prediction table given in Table 12 was created. This table was used uniformly across all benchmarks.

Though, the static prediction table may not be optimal for all benchmarks, tailoring solutions leads to only marginal gains. Hence, the Static Pattern Table (SPT) was fixed for all the benchmarks in order to maximize the benefits of this approach for realizing efficient hardware implementations. Figure 76 shows the algorithm used in the static FC hit/miss prediction. It is assumed that the address for the next instruction will be the current fetch address plus "4" due to instruction being 32-bit wide. This would hold for most of the times based on the insights provided by branch prediction researches. The predictor is accessed only when cache line changes are encountered. Thus, we predict a hit in the FC if the cache line being accessed is unchanged.

In the event of a change, we access the SPT corresponding to the shift register value, and the prediction of a hit or a miss in the FC is made depending on the SPT value. In order to ensure that the prediction process does not interfere with the average cache access cycle, the prediction process must be hidden between accesses. The prediction for the next access is carried out in parallel with the current access.

5.5.2 SystemC based hardware architecture of the static pattern predictor

To get an insight into the hardware complexity of the SPP algorithm, SystemC was used once gain to model the architecture. The simplified hardware implementation of the algorithm is shown in Fig. 77.

Figure 76. Static Pattern Prediction Algorithm

Figure 77. Simplified Architecture of the prediction algorithm

Predictor	For 128 Byte FC	For 256 Byte FC
NFPT	A register to hold last Line address, 4X8 bit mem. array to hold NFPT Table, and 4-bit Comparator	A register to hold last Line address, 4X16 bit mem. array to hold NFPT Table, and 4-bit Comparator
Static Predictor	A register to hold last Line address, 32 Bit mem. for PT, and comparator for line change detection	A register to hold last Line address, 32 Bit mem. for PT, and comparator for line change detection

Table 13. Hardware complexity comparison of NFPT and Static Pattern Predictor

Table 13 shows the resources required for hardware implementation of the static predictor and NFPT. As can be seen from the table the static predictor is relatively simple to implement. Moreover, the area complexity of the static predictor architecture is independent of the FC size. This is not the case for NFPT.

5.5.3 Run-Time Optimization Options

The investigations show that the SPT entries can be computed off line in order to capture any peculiar characteristics of an application, thereby assigning a unique SPT for each application. This will pave the way to further minimize energy-delay product in an application centric manner.

5.5.4 Experimental Results

This section discusses the experimental comparison of static pattern predictor and NFPT based predictor with the FC size fixed at 128 and 256 bytes (similar to cache configuration used in [45]). The prediction accuracy and the energy-delay product are compared to demonstrate the effectiveness of the two predictors. The energy-delay product reduction is compared against the base case where FC is not a part of the hierarchy. Normalized cycle time is also illustrated to show the relatively negligible performance degradation.

5.5.4.1 Prediction Accuracy

Fig. 78 compares the prediction accuracy of Static Pattern Predictor and NFPT. It is clearly evident that the Static Pattern Predictor performs better than the NFPT for all the cases shown. However, this does not guarantee that the energy-delay product reduction for Static Pattern Predictor is less than that of NFPT. This is because the energy delay product measures heavily depend on whether the wrong predictions are directed to main cache or FC. If the wrong predictions are directed towards FC, it will result in performance degradation. If, on the other hand, the wrong predictions are directed towards main cache it will result in increased access energy. The average Cycle Time and average Access Energy are discussed next to provide further clarifications.

Figure 78. Prediction Accuracy for 128 and 256 byte FC

Figure 79. Normalized Cycle Time for 128 and 256 byte FC

5.5.4.2 Normalized Cycle Time

Fig. 79 compares the normalized access cycle time. Here the normalization is done with respect to the case where FC is not present. The table clearly demonstrates that the Static Pattern Predictor always outperforms NFPT. The average performance penalty with NFPT compared to the case where FC is not present is 5.34% whereas it is only 4.11% for Static Pattern Predictor. This performance penalty is computed for an ideal pipeline without stalls. Hence, maximum

performance penalties are unlikely in practice. Next, we will examine the access energy measures for both approaches.

5.5.4.3 Normalized Access Energy

Fig. 80 shows the normalized access energy with respect to the base case where filter cache is not present. As before, the Static Pattern Predictor consistently outperforms NFPT. For example, the use of Static Pattern Predictor results in an average access energy reduction of 42.6% while it is 41.6% for NFPT. Similarly, for the case with 256 byte FC, it is 50.8% and 49.6% respectively.

5.5.4.4 Energy-Delay Product Reduction over NFPT for Static Pattern Predictor

Table 14 shows the energy-delay product reduction achieved with Static Pattern Predictor as a percentage over NFPT. It is clearly evident that, for the benchmarks chosen, Static Pattern Predictor provides for better energy-delay product reduction of up to 6.79%.

Figure 80. Normalized Access Energy for 128 and 256 byte FC

Benchmark	For 128 Byte FC	For 256 Byte FC
ADPCM DECODER	0.01%	6.79 %
G721 DECODE	3.76 %	1.16 %
GS-SCRIPT DECODE	5.66 %	3.79 %
JPEG-ENC	3.29 %	2.36 %
GSM DECODE	1.28 %	0.37 %

Table 14. Percentage Energy-Delay Product Reduction with Static Pattern Predictor Compared to NFPT

5.5.5 Summary of Static Predictor Results

In this section, the Static Prediction Table based prediction scheme was proposed to replace NFPT. The proposed technique was shown to provide a reduction in the energy delay product of up to 6.79% than that incorporates NPFT. Moreover, the prediction technique occupies less

VLSI area than NFPT and Dynamic pattern prediction scheme due to the elimination of saturation counters. Unlike NFPT, the Static Pattern Predictor is independent of FC size and its superiority over NFPT is demonstrated when Filter Cache size was scaled up from 128 to 256 bytes. When compared to the base case without FC, the Static Pattern Predictor scheme provides for up to 66% access energy reduction. The ability to tailor the pattern table for a given application paves the way for further reduction in the energy delay product. Finally, the advantages of the Static Pattern Predictor in terms of area reduction and switching activity levels, due to the elimination of saturation counters, compensates for the marginal reduction in the energy delay product of the dynamic pattern prediction scheme.

5.6 Micro-architecture (VHDL) Implementation for Pattern Predictor

The pattern prediction algorithm has been shown to provide substantial improvements in area without sacrificing performance when used to predict access to the filter cache in the instruction memory hierarchy. Variations of the prediction algorithm were discussed and some early indications of silicon area were discussed on the basis of SystemC based modeling of the architecture and rudimentary synthesis using Synopsys design compiler.

Since, the pattern predictor (in its various forms) and the filter cache are an overhead to the existing cache architecture, it is imperative that they should together not lower the energy efficiency of the existing architecture. Thus, the extra energy consumed by the predictor should be minimal as compared to the overall energy reduction achieved by the introduction of the filter cache in the instruction memory hierarchy. In this chapter, we implement the detailed hardware architecture of low-power, single-cycle pattern predictor, which can be used in conjunction with the filter cache. The predictor is functionally validated against results from SimpleScalar simulations on the MediaBench benchmark suite.

The earlier section has shown that the pattern predictor algorithm is instrumental in providing accurate prediction as well as in reducing power consumption by using small FC sizes. However, the power consumption of the predictor architecture has not been quantified earlier. In this section, we propose and design the architecture for the pattern predictor, and subsequently analyzed the overall power consumption of the resulting cache memory architecture.

Figure 81. Access Paths for (a) conventional, (b) hierarchical FC and (c) FC prediction, Based Instruction Memory Hierarchy

5.6.1 Overview of the Filter Cache Hierarchy in a Processor

Depicted in Figure 81(b) & (c), is the filter cache (FC) based hierarchy, which incorporates the use of a small cache of the order of 256-512 bytes in addition to the L1 cache of the order of 8-32 Kbytes. The hierarchy in Figure 81 (b) provides power savings of up to 58% over the cache hierarchy shown in Figure 81 (a). The motivation here is that the FC should be able to provide the instructions while executing small repeated loops which accounts for most of the execution time in embedded applications, thus allowing the L1 cache to be switched off and thus conserving power. This is due to the fact that the capacitive and leakage power of the L1 cache is much higher than that of the relatively much smaller FC. However, due to the hierarchical path to the CPU, shown in Figure 81 (b), there is a massive performance penalty of up to 28% due to misses in the FC. Figure 81 (c) illustrates predictive FC, where the predictor predicts whether the next instruction is present in the FC or not. It has been shown that the predictor based hierarchy limits performance degradation to 1.3% whilst providing a power savings of up to 35% for a wide issue superscalar processor with spec95 benchmarks. However, the power of the prediction logic architecture is not accounted for in the reported papers.

Since the primary aim of introducing this novel hierarchy was to reduce the power consumption of the instruction memory hierarchy, it is important to assess the extra power consumption incurred by the incorporation of the cache memory and more importantly the predictor logic.

5.6.2 Pattern Predictor Architecture

Two level adaptive branch predictor has been researched quite extensively, and various kinds of two level pattern based predictors have been proposed. Numerous studies have shown which predictors and configurations best predict the branches in a given set of benchmarks. As shown

in the Figure 82, the two level predictor consists of a shift register (SR)(of N bits) to record the hit/miss patterns of instruction lines and the Pattern history Table (PHT) consisting of 2 bit (2N entry) saturating counters. When the previous N outcomes of hit/miss are known, then by indexing the PHT with this value we can predict whether next instruction access one is going to be a hit or miss. Drawing from the insight provided by branch prediction research, we assume that next instruction fetch address will be current fetch address plus “4” (each instruction has a 32-bit size and most branches are not taken so that even if the current instruction is a branch it will not be taken, so that next instruction address is current instruction address plus four). We access the predictor result (sPred) only when cache line changes are encountered. Hence, we predict a hit in the filter cache if the cache line being accessed is unchanged (sPredEn = 1). In the event of a change (sPredEn = 0), we access the PHT corresponding to the shift register value, and the prediction of a hit or a miss in the filter cache is made depending on whether the pattern history table value reads above the pre-set threshold (pCompareVal) or below.

Figure 82. Single Cycle Pattern Prediction Architecture for FC Access Prediction

Figure 83. Experimental Setup for Architecture Validation & Power Analysis

For a two-bit saturating counter pattern history table, a threshold value of 2 (102) is used. When the outcome of the prediction is known the shift register and pattern history table values are accordingly updated, as in the normal branch predictor scheme.

5.6.3 Simulation Platform and Architecture Validation

The overview of the pattern prediction architecture simulation and validation setup is shown in Figure 83. The pattern predictor algorithm was first tested for accuracy and performance using the SimpleScalar [49] tool set with the ARM ISA (instruction set architecture). The results are available in [57]. The address trace as well as the prediction result and hit/miss results were captured from the above simulations. MediaBench [44] benchmark suite was chosen for the purpose of testing the hardware architecture.

5.6.4 Hardware Validation

The design of the pattern predictor architecture was done in VHDL. The address trace captured in the software simulation was then applied to the VHDL testbench of the architecture depicted in Figure 83. The testbench does an automated verification to validate that the VHDL architecture presents the same prediction results as is in SimpleScalar simulations. The architecture is thus verified for functionality.

Figure 6: Filter Cache and Predictor

Figure 84. Filter Cache and Predictor

Figure 84 shows the hardware configuration for the pattern predictor and the cache memory in simulation. The Program Counter, HIT/MISS result and En/Disable Control are provided in the testbench from the software simulation.

5.6.5 Hardware Synthesis Results

The architecture was synthesized for configurations supporting PHT's with 8 and 32 entries to determine the maximum processor clock speed (Table 1) that it can support. The synthesis was done using the Synopsys Design Compiler with Passport Standard Cell Libraries for 0.35-micron process technology.

Pattern Entries	8	32
Critical Path [ns]	1.44	1.83
Max Freq [MHz]	694	500
Area [gates]		
Comb	545	1098
Non-Comb	605	827
Interconnect	428	706
Total	1578	2631

Table 15. ASIC Synthesis Results

5.6.6 Performance

Table 16 shows the average delay (in cycles) per access to the cache memory hierarchy for cases with success of cache hit in the instruction memory cache hierarchy. For the case with just L1 cache this value is 1, also the ideal case. However, with hierarchical L1+FC, this depreciates by 1.64% to 30.65%, with an average performance degradation of 30.65%.

Benchmark	L1	L1 + FC		L1+FC with Predictor Inclusive of Predictor Architecture	
	Cycles	Cycles	% Degradation	Cycles	% Degradation
ADPCM-DEC	1	1.31	30.65	1.07	6.87
ADPCM-ENC	1	1.30	30.06	1.05	5.05
EPIC	1	1.02	1.64	1.01	0.91
UNEPIC	1	1.05	4.96	1.02	1.70
G721.DEC	1	1.14	13.98	1.03	2.95
G721.ENC	1	1.15	15.26	1.04	4.18
GSM.TOAST	1	1.24	24.07	1.01	1.42
GSM.UNTOAST	1	1.23	22.85	1.02	1.63
MIPMAP	1	1.11	11.12	1.03	2.51
OSDEMO	1	1.18	18.09	1.03	3.36
TEXGEN	1	1.22	22.28	1.02	1.81
MPEG2.DEC	1	1.08	7.63	1.02	2.27
MPEG2.ENC	1	1.14	13.53	1.02	2.30
PEGWIT	1	1.05	5.29	1.01	0.92
PEGWIT.DEC	1	1.05	5.29	1.01	0.92
PEGWIT.ENC	1	1.07	7.04	1.01	0.88
JPEG.DEC	1	1.13	12.88	1.02	1.57
JPEG.ENC	1	1.08	7.96	1.02	1.74
Overall Statistics					
Min	1.00	1.02	1.64	1.01	0.88
Max	1.00	1.31	30.65	1.07	6.87
Average	1.00	1.14	14.14	1.02	2.39

Table 16. Average Delay (in cycles) per access (Performance Metric)

However, with the introduction of the prediction-based structure (L1 + Pred. FC), the degradation is limited to a maximum of 6.87% with the single cycle prediction architecture implemented.

5.6.7 Power Consumption

Power characterization was done based on the switching activity collected during the simulation of the test-bench (Figure 83). This gives the most accurate possible representation at this stage of design. The test-benches were run to completion and the switching activity was then back annotated to the gate level synthesized netlist to estimate the power requirement of the pattern predictor for the 32- entry PHT configuration. The 8-entry design was not characterized for power as it guaranteed to be lesser than the results obtained with 32-entry PHT. Further, CACTI [50] (Access and Cycle Time Model for On-Chip Caches) provides the power requirement estimate for optimized L1 and FC memory structures on 0.35 micron technology. Table 17 illustrates the power requirements for (a) conventional cache structure w/o FC; (b) FC w/L1, and (c) pattern predictor. The L1 and FC size used is 256 bytes and 32KB (4-way set associative w/ 32 byte line size) respectively Table 17, illustrates improvement in power consumption for the L1, L1+FC and L1+ Pred. FC hierarchies. It is clearly seen that even with the inclusion of the Pattern Predictor architecture power consumption, the predictive structure maintains a average

power consumption reduction of 67.96%. However, this is marginally lower than the 71.08% obtained from the hierarchical L1+FC structure. This is so because of the overhead of the FC Prediction architecture.

Benchmark	L1	L1 + FC		L1+FC with Predictor	
	mW	Conventional mW	% Improvement	Incl. of Predictor Architecture mW	% Improvement
ADPCM-DEC	34.34	22.56	34.30	21.17	38.36
ADPCM-ENC	44.23	28.73	35.04	25.65	42.00
EPIC	150.79	43.61	71.08	48.31	67.96
UNEPIC	24.87	8.24	66.87	8.92	64.12
G721.DEC	917.43	408.89	55.43	405.06	55.85
G721.ENC	961.46	444.11	53.81	451.31	53.06
GSM.TOAST	1347.16	772.69	42.64	652.74	51.55
GSM.UNTOAST	562.55	313.98	44.19	271.72	51.70
MIPMAP	5.40	2.21	59.07	2.21	59.05
OSDEMO	66.45	33.07	50.23	31.42	52.72
TEXGEN	94.10	51.84	44.91	45.42	51.74
MPEG2.DEC	466.25	170.22	63.49	181.61	61.05
MPEG2.ENC	2565.07	1128.41	56.01	1096.94	57.24
PEGWIT	48.50	16.27	66.46	16.80	65.37
PEGWIT.DEC	49.46	16.59	66.45	17.13	65.36
PEGWIT.ENC	101.69	36.37	64.23	36.23	64.38
JPEG.DEC	9.89	4.27	56.84	4.06	58.99
JPEG.ENC	36.75	13.57	63.07	13.90	62.18
Overall Statistics					
Min	5.40	2.21	34.30	2.21	38.36
Max	2565.07	1128.41	71.08	1096.94	67.96
Avg	415.91	195.31	55.23	185.03	56.81

Table 17. Power Consumption

5.6.8 Energy-Delay Product

The energy delay product (EDP) represents the overall efficiency of the L1 + Pred. FC instruction memory structure. The results illustrated in Table 18 shows that the predictive architecture maintains an average EDP improvement of 67.67% as against 70.6% of the L1+FC hierarchical structure.

Benchmark	L1	L1 + FC [conventional]		L1+FC with Predictor [Incl. of Pred. Arch.]	
	EDP	EDP	% Improvement	EDP	% Improvement
ADPCM-DEC	34.34	29.48	14.16	22.73	33.82
ADPCM-ENC	44.23	37.37	15.51	27.01	38.92
EPIC	150.79	44.32	70.60	48.76	67.67
UNEPIC	24.87	8.65	65.23	9.08	63.50
G721.DEC	917.43	466.07	49.20	417.36	54.51
G721.ENC	961.46	511.88	46.76	471.00	51.01
GSM.TOAST	1347.16	958.64	28.84	662.12	50.85
GSM.UNTOAST	562.55	385.72	31.43	276.22	50.90
MIPMAP	5.40	2.46	54.52	2.27	57.99
OSDEMO	66.45	39.05	41.23	32.51	51.07
TEXGEN	94.10	63.39	32.64	46.26	50.84
MPEG2.DEC	466.25	183.20	60.71	185.83	60.14
MPEG2.ENC	2565.07	1281.05	50.06	1122.83	56.23
PEGWIT	48.50	17.13	64.68	16.95	65.05
PEGWIT.DEC	49.46	17.47	64.68	17.29	65.04
PEGWIT.ENC	101.69	38.94	61.71	36.55	64.06
JPEG.DEC	9.89	4.82	51.28	4.12	58.34
JPEG.ENC	36.75	14.65	60.13	14.15	61.51
Overall Statistics					
<i>Min</i>	<i>5.40</i>	<i>2.46</i>	<i>14.16</i>	<i>2.27</i>	<i>33.82</i>
<i>Max</i>	<i>2565.07</i>	<i>1281.05</i>	<i>70.60</i>	<i>1122.83</i>	<i>67.67</i>
<i>Avg</i>	<i>415.91</i>	<i>228.02</i>	<i>47.96</i>	<i>189.61</i>	<i>55.64</i>

Table 18. Energy Delay Product

5.7 Overall Comparison of SLDL and HDL in Design Space Exploration

In a system level environment, we note that most of the system algorithms and control flow modeling is done first in a more popular language (ANSI-C/C++) or software design suite (Visual C++/Borland C++/Matlab). This generates the golden results which are then used for validation at various steps of the HDL flow as shown in Figure 85.

However, there is a gap between the translations from C/C++/Matlab code to HDL architecture implementation. This is primarily due to the human error in translation, and more importantly due to the inherent in-formalism in a specification document which is handed to the HDL designer and is subject to interpretation on certain occasions.

With the advent of SystemC as a language from concept to implementation, algorithm and design architects were promised a universal language for modeling (Figure 86). SystemC is ANSI-C++ compliant and SystemC is also supported by many electronic design automation tool vendors today for direct RTL synthesis. However, as noted in the earlier chapters of this thesis where we compare HDL and SLDL synthesis results, we realize that SLDL synthesis will always be inefficient as compared to HDL synthesis. This is so due to the high level of abstraction which languages such as SystemC address.

Figure 85. Typical HDL only design flow

Figure 86. SystemC based unified flow for algorithm and architecture designer

Nevertheless, SystemC plays the important part of providing a bridge between an algorithm architect and the architecture designer. Also it allows both the sets of teams to talk in a single language for trade-offs at the system and the architecture level.

5.8 Framework for unified environment for system level evaluation of customized architectures

We note in Figure 87 that enabling quick iterative analysis between the algorithm-level design space exploration and architecture-level design space exploration would be the key to have an efficient system level design space exploration environment. Hence, we propose the unified environment (Figure 88) which leverages on the respective advantages of SLDL and HDL, as well as conjoins the initial architecture exploration effort with the system level analysis and profiling tools. Since, the exact simulation tools involved at the system level will depend on the actual SoC or ASIC being designed; it has to be identified on a case by case basis.

Figure 87. Unified Algorithm and Architecture Design Space Exploration

Figure 88. Elaboration of Architecture Design Space Exploration for Algorithm Analysis

The illustration in Figure 89 further elaborates on the framework of the unified environment in a manner that is not specific to any tools and languages. This environment can be used for effective system level algorithm analysis and architecture design space exploration.

Figure 89. Language and Tool Agnostic Framework of Unified Environment

The framework for unified analysis at the system level leading to HDL implementation for accurate architecture analysis has several advantages. Though the advantages of this approach have been summarized in every chapter of this report, we present some of the salient points below for the readers:

- **Specification Convergence:** Closes the gap between the system level architect and the hardware architecture designer. They now talk in the same language and use executable specifications (SystemC executables) to communicate the desired specification.
- **System level architecture evaluation:** System level design partitioning can now be done by the design architect in a more informed manner, and in collaboration with the system architect. System level simulations can also include architecture description in SystemC so as to simulate the encompassing hardware architecture at a clock cycle level if desired. Thus evaluating the performance of the architecture with the rest of the system.
- **HDL Convergence:** The hardware architecture variants for a particular algorithm can now be evaluated with relative estimations of area, power and performance so as to invest the time and effort in the intensive HDL implementation stage in a focused manner. This also increases the confidence of convergence for the system designer.

Other advantages include the higher probability of meeting the TTM and NRE constraints, and the enablement of an environment where system level analysis tools can contribute to the design space exploration of architectures and system level algorithm analysis.

5.9 Conclusion

Although, powerful electronic design automation tools continue to emerge there will continue to exist the need for a tailored design space exploration process to realize optimal solutions. This is extremely important in the presence of constrained design requirements. Further, difficulty in the presence of modeling and electronic design automation tools necessitates the need to establish the design space exploration environment with a minimum set and maximal interoperability.

This chapter introduced the predictor based filter cache memory hierarchy for instruction cache. The prediction algorithm is first verified for accuracy in the system simulation context using SimpleScalar and CACTI tools. The prediction algorithm was then scrutinized at the system level using SystemC for an initial hardware complexity cost analysis from the area and performance perspective. Further, the simulations also lead to the innovative static pattern predictor which is also then evaluated to be much more architecture friendly. Having performed the design space exploration at the system level, the next section concentrated on evaluating the exact and more accurate overheads to determine the true extent of usefulness of the algorithms and the respective architectures. This was done in VHDL and the design was synthesized to gates and simulation based activity annotation was used to accurately predict the power consumption of the design in actual applications. The prediction architecture reduced the power savings by about 4% in average for the entire benchmark as compared to the power consumption of the hierarchical FC hierarchy. Nevertheless, the predictive architecture limits performance penalty to just about 6.87% for the L1+Pred. FC structure from 30% for the hierarchical L1+FC structure. The system level algorithm analysis and subsequent porting in VHDL has led to power savings of 67.96% with a mere 6.87% penalty in performance. Further, the synthesized design does not use up more than 2600 gates ($\sim 0.14 \text{ mm}^2$) on 0.35-micron technology and can operate in single cycle mode beyond speeds of 0.5 GHz. Further, this chapter proceeds to illustrate the comparative points between SLDL and HDL based design space exploration. Based on the algorithm and architecture design space exploration, a unified environment is then illustrated. Overall this chapter serves to provide a homogenous simulation and modeling environment for design space exploration with a minimum set of tools and design languages that permit a high degree of flexibility in architecture design space exploration and algorithm analysis.

Chapter 6 Conclusion & Recommendations

6.1 Conclusion

This thesis addressed the problem of system level algorithm analysis and constraint aware hardware architecture translation. The international technology roadmap for semiconductors reports that custom design environments are required for effective design space exploration at an early stage in design and this was further reinforced during the in-depth literature survey that was carried out in the early stages of this research work.

The tradeoffs between HDL and SLDL based design flows for architecture design, implementation and exploration was investigated in detail. Three variations of hardware architectures were implemented with different degrees of folding and pipelining. Although, these extremely time-consuming implementations had to be hand-coded each time, and functional verification had to be carried out at each step of the change, VHDL based design space exploration permitted the realization of accurate area-time trade-offs. For example, a single iteration of K-means took 9 μ s (at 100 MHz clock) with an area cost of 177K gates (0.35u), and in another extreme point in design space it took 2.3ms (at 66.67 MHz clock) with an area cost of 34K gates (0.35u).

The SystemC based flow in Chapter 4 illustrated the feasibility of high-level manipulations through compiler directives and behavioral compilation in comparison with that employing hand-coded VHDL. The speed of verification and effecting architectural transformations in the SLDL language environment facilitated the analysis of many variations of the architecture. For example, as many as twelve variations were analyzed for the K-means architecture to arrive at the optimal word-length for the data-path on the basis of accuracy (mean-squared error) trade-off with area and performance. Due to the high abstraction level, the architecture variations were analyzed with ease to provide an effective relative comparison of architecture cost (area, power, and time) estimates to identify the most appropriate architecture for eventual implementation. In contrast to the K-means implementation in VHDL, SystemC based design exploration permitted the fastest iteration of only 214ms (at 66.67MHz clock) with an area cost of 72K gates (0.35u), and the slowest iteration of 1.68s (at 13.33MHz clock) with an area cost of 65K gates (0.35u). The disparity in the architecture variations is thus notable between SystemC and VHDL estimates. Given the specific example of the K-means architecture, the author with detailed familiarity of the algorithm has found the fastest approach using SystemC to be several times slower and 2.45 times more expensive in area. However, the relative trend of architecture cost

estimates is consistent with SystemC and thus beneficial towards design space exploration at an early stage.

In algorithms where isolated design space exploration is infeasible, and the custom architecture is a part of a system, SLDL based algorithm analysis alone is insufficient. This led to the incorporation of system level analysis and profiling tools into the design space exploration environment. A comprehensive analysis of the design space exploration of custom hardware architecture in an integrated environment was presented using predictive filter cache based instruction cache hierarchy as a design vehicle. CACTI and SimpleScalar were two domain specific system level tools that were employed to provide the profiling environment to account for the performance characteristics of the microprocessor in which this cache hierarchy was to be integrated. The initial SLDL (SystemC) level algorithm analysis in conjunction with the system level analysis and profiling tools (CACTI & SimpleScalar) led to the demonstration of the proposed prediction technique as more energy-efficient than NFPT prediction schemes. This integrated exploration environment further confirmed that when compared with NFPT implementations, the proposed predictor leads to an average energy-delay product reduction of 3.39% and 2.39% with 256-byte and 512-byte filter cache, respectively. The system level analysis was rounded off with the SystemC architecture analysis providing initial estimates of 702 (8-entry PHT) to 6174 (128-entry PHT) ASIC gates (0.35u) for the various architecture variants. The completeness in the system level algorithm analysis and architecture design space exploration analysis at the early stage in design analysis led to the identification of the 32-entry PHT based pattern prediction architecture for subsequent implementation in VHDL to meet the desired architecture constraints. The tight coding in VHDL allowed for a single clock cycle design that only requires 2600 ASIC gates (32-entry PHT) in 0.35u technology and can operate beyond speeds of 0.5GHz. The system level algorithm analysis and subsequent porting in VHDL has led to power savings of 67.96% with a mere 6.87% penalty in performance. The hybrid design flow presented leverages on the SLDL based environment for rapid architecture exploration followed by the implementation of the optimal architecture in HDL.

A unified environment that is system specific, and utilizes a hybrid SLDL and HDL design flow to enable a tradeoff analysis for system level design space exploration of custom architectures has been proposed. This provides for a more comprehensive design analysis environment for rapid design convergence towards custom architectures.

Although, powerful electronic design automation tools continue to emerge there exists the need for a tailored design space exploration process to realize optimal solutions. Difficulty in the

presence of modeling and electronic design automation tools necessitates the need to establish the design space exploration environment with a minimum set and maximal interoperability. This dissertation serves to emphasize on the need for an integrated simulation and modeling environment towards realizing optimal hardware architectures through design space exploration with system level algorithm analysis.

6.2 Future Work

This dissertation has looked into the problem of system level algorithm analysis and architecture design space exploration of custom architectures. This falls under the purview of current challenges in architecture design of algorithms in the face of functional constraints and increasingly important, the non-functional constraints of hardware design. The research work done has provided a foundation for subsequent issues to be addressed. The sections that follow outline the major items of future work.

6.2.1 Design Entry and Specification capture

Research in design entry environment for high level design specification will also be required to accurately capture design information for accurate interpretation at the abstract level. This will involve semantics and syntax of design specification. Hierarchical specification of design units will probably be an area of research which will benefit specific functional and non-functional concerns as accurate specification will lead to unambiguous design implementation meeting the requirements.

6.2.2 Extracting Inherent Parallelisms

Tools that evaluate the inherent parallelism and deep pipelining issues keeping in mind data dependencies for algorithm to architecture translations would be of interest towards automation in design exploration. A good starting point would be to examine intermediate representations of the compilation phases in hardware synthesis and data flow analysis.

6.2.3 Automation in SLDL to HDL Translation

Seamless translation of SLDL specification to HDL expression will further aid the system level design space exploration to be instrumental in achieving faster TTM and lower NRE. Research in synthesis tools supporting system level design languages is also another research area which will aid design space exploration.

6.2.4 Hardware Software Partitioning

With the advent of technology offerings which include ASIC and FPGA fabric on a single chip, hardware-software partitioning is becoming a key issue in the design of complex embedded systems. Work towards integration of system level custom architecture design environments and hardware-software partitioning techniques would lend maximum flexibility in providing design options.

6.2.5 Reducing Language Dependency

The key to increasing productivity in custom architecture design would be to work towards language independence. It will be of interest to explore a uniform language structure which will lead to reducing the dependence on the expert levels of hardware designers.

6.2.6 Physical Synthesis Approach to System Level Design

As is currently existent in ASIC design flows, a front-to-back flow all the way from system level abstraction to netlist implementation at the transistor level will allow sensitivity to technology dependent issues such as semiconductor technology. Emerging trends in hardware design such as structured ASIC's could also be accommodated in this manner.

6.2.7 System Level Verification and Validation

Tools at the system level which can aid towards accurate verification and validation with the identified hardware architecture will be of value towards meeting TTM concerns. This will strengthen its appropriateness towards commercial deployment.

6.2.8 System Level Estimation Tools

The major reason behind the inaccuracy in analysis at the system level using SystemC was due to the estimation at the high level of abstraction not providing a high correlation when compared with the actual implementation results. System level design space exploration can be better facilitated by accurate estimation of area, power and performance characteristics at an abstract level of specification. This requires further research in the area of accurate estimation of such parameters in moving towards true abstract system level design space exploration.

BIBLIOGRAPHY & REFERENCES

Conferences and Journals

- [1] International Technology Roadmap for Semiconductors (Design), 2001 Edition.
<http://public.itrs.net/Files/2001ITRS/Design.pdf>
- [2] Chandrakasan, A.P.; Sheng, S.; Brodersen, R.W., **Low-power CMOS digital design**, Solid-State Circuits, IEEE Journal of, Volume: 27 Issue: 4, April 1992, Page(s): 473 –484.
- [3] Brodersen, R.W.; Chandrakasan, A.; Sheng, S., **Design techniques for portable systems**, Solid-State Circuits Conference, 1993. Digest of Technical Papers. 40th ISSCC., 1993 IEEE International, 1993 Page(s): 168 –169.
- [4] Hossain, R.; Wronski, L.D.; Albicki, A., **Low power design using double edge triggered flip-flops**, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, Volume: 2 Issue: 2, June 1994 Page(s): 261 –265.
- [5] Blair, G.M., **Designing low-power digital CMOS**, Electronics & Communication Engineering Journal , Volume: 6 Issue: 5 , Oct. 1994 Page(s): 229 –236.
- [6] Hyun-Gyu Kim; Hyeong-Cheol Oh. **Retiming for high speed and low power design**. VLSI and CAD, 1999. ICVC '99. 6th International Conference on, 1999 Page(s): 395 –398.
- [7] Sylvester, D.; Keutzer, K. **Getting to the bottom of deep submicron**. Computer-Aided Design, 1998. ICCAD 98. Digest of Technical Papers. 1998 IEEE/ACM International Conference on , 1998 Page(s): 203 –211.
- [8] Koegst, M.; Franke, G.; Rulke, S.; Feske, K. **Low power design of FSMs by state assignment and disabling self-loops**. EUROMICRO 97. New Frontiers of Information Technology., Proceedings of the 23rd EUROMICRO Conference , 1997 Page(s): 323 –330.
- [9] De-Sheng Chen; Sarrafzadeh, M.; Yeap, G.K.H. **State encoding of finite state machines for low power design**. Circuits and Systems, 1995. ISCAS '95., 1995 IEEE International Symposium on , Volume: 3 , 1995 Page(s): 2309 -2312 vol.3.
- [10] Koegst, M.; Franke, G.; Feske, K. **State assignment for FSM low power design**. Design Automation Conference, 1996, with EURO-VHDL '96 and Exhibition, Proceedings EURO-DAC '96, European , 1996 Page(s): 28 –33.
- [11] Kuroda, T.; Hamada, M. **Low-power CMOS digital design with dual embedded adaptive power supplies**. Solid-State Circuits, IEEE Journal of , Volume: 35 Issue: 4 , April 2000 Page(s): 652 –655.
- [12] Liu, D.; Svensson, C. **Trading speed for low power by choice of supply and threshold voltages**. Solid-State Circuits, IEEE Journal of , Volume: 28 Issue: 1 , Jan. 1993 Page(s): 10 –17.
- [13] Kuroda, T.; Suzuki, K.; Mita, S.; Fujita, T.; Yamane, F.; Sano, F.; Chiba, A.; Watanabe, Y.; Matsuda, K.; Maeda, T.; Sakurai, T.; Furuyama, T. **Variable supply-voltage scheme for low-power high-speed CMOS digital design**. Solid-State Circuits, IEEE Journal of , Volume: 33 Issue: 3 , March 1998 Page(s): 454 –462.

-
- [14] Panda, P.R.; Dutt, N.D. **Reducing address bus transition for low power memory mapping**. European Design and Test Conference, 1996. ED&TC; 96. Proceedings , 1996 Page(s): 63 –68.
 - [15] Arunachalam, P.; Abraham, J.; d'Abreu, M. **A hierarchical approach for power reduction in VLSI chips**. VLSI, 1996. Proceedings., Sixth Great Lakes Symposium on , 1996 Page(s): 182 –185.
 - [16] Bogliolo, A. **Encodings for high-performance energy-efficient signaling**. Low Power Electronics and Design, International Symposium on, 2001. , 2001 Page(s): 170 –175.
 - [17] Macchiarulo, L.; Macii, E.; Poncino, M. **Low-energy encoding for deep-submicron address buses**. Low Power Electronics and Design, International Symposium on, 2001. , 2001 Page(s): 176 –181.
 - [18] Rjoub, A.; Nikolaidis, S.; Koufopavlou, O.; Stouraitis, T. **An efficient low-power bus architecture**. Circuits and Systems, 1997. ISCAS '97., Proceedings of 1997 IEEE International Symposium on , Volume: 3 , 1997 Page(s): 1864 -1867 vol.3.
 - [19] Paliouras, V.; Stouraitis, T. **Low-power properties of the logarithmic number system**. Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on , 2001 Page(s): 229 –236.
 - [20] Hao-Yung Lo; Hsiu-Feng Lin; Yue-Yuan Ho. **Logarithmic conversion by four partitioned hybrid-ROMs**. Parallel Architectures, Algorithms, and Networks, 1996. Proceedings., Second International Symposium on , 1996. Page(s): 550 –552.
 - [21] Abram Dancy, Anantha Chandrakasan. **Techniques for Aggressive Supply Voltage Scaling and Efficient Regulation**. IEEE 1997, Custom Integrated Circuits Conference, pp. 579-586.
 - [22] Hamada, M.; Takahashi, M.; Arakida, H.; Chiba, A.; Terazawa, T.; Ishikawa, T.; Kanazawa, M.; Igarashi, M.; Usami, K.; Kuroda, T. **A top-down low power design technique using clustered voltage scaling with variable supply-voltage scheme**. Custom Integrated Circuits Conference, 1998. Proceedings of the IEEE 1998 , 1998 Page(s): 495 –498.
 - [23] Chunhong Chen; Srivastava, A.; Sarrafzadeh, M. **On gate level power optimization using dual-supply voltages**. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , Volume: 9 Issue: 5, Oct. 2001 Page(s): 616 –629.
 - [24] Mahnke T., Panenka S., Embacher M., Stechele W., Hoeld W. **Power Optimization Through Dual Supply Voltage Scaling Using Power Compiler**. SNUG Europe 2002.
 - [25] Dongwan Shin; Kiyoun Choi. **Low power high level synthesis by increasing data correlation**. Low Power Electronics and Design, 1997. Proceedings., 1997 International Symposium on , 1997 Page(s): 62 –67.
 - [26] Usami, K.; Ishikawa, T.; Kanazawa, M.; Kotani, H. **Low-power design technique for ASICs by partially reducing supply voltage**. ASIC Conference and Exhibit, 1996. Proceedings, Ninth Annual IEEE International, 1996 Page(s): 301 –304.
 - [27] Kai-Yuan Chao; Wong, D.F. **Floorplanning for low power designs**. Circuits and Systems, 1995. ISCAS '95., 1995 IEEE International Symposium on , Volume: 1 , 1995 Page(s): 45 -48 vol.1
-

-
- [28] Schimpfle, C.V.; Simon, S.; Nossek, J.A. **Optimal placement of registers in data paths for low power design**. Circuits and Systems, 1997. ISCAS '97., Proceedings of 1997 IEEE International Symposium on , Volume: 3 , 1997 Page(s): 2160 -2163 vol.3
 - [29] Z.X.Shen, C.C. Jong. **Exploring Module Space Selection Space for Architectural Synthesis of Low Power Designs**. 1997 IEEE International Symposium on Circuits and Systems, June 9-12, 1997, Hong Kong.
 - [30] Narayanan, U.; Stamoulis, G.I.; Roy, R.K. **Characterizing individual gate power sensitivity in low power design**. VLSI Design, 1999. Proceedings. Twelfth International Conference On , 1999 Page(s): 625 -628.
 - [31] Svensson, C. **Optimum voltage swing on on-chip and off-chip interconnect**. Solid-State Circuits, IEEE Journal of , Volume: 36 Issue: 7 , July 2001 Page(s): 1108 -1112.
 - [32] Mehra, R.; Guerra, L.; Rabney, J. **Exploiting locality for low-power design**. Custom Integrated Circuits Conference, 1996. Proceedings of the IEEE 1996. Page(s): 401 -404.
 - [33] Mehra, R.; Rabaey, J. **Exploiting regularity for low-power design**. Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on , 1996 Page(s): 166 -172.
 - [34] Naik, J. M., **"Speaker Verification: A Tutorial"**, *IEEE Communications Magazine*, Volume: 28 Issue: 1, Jan. 1990.
 - [35] G. McLachlan,; **Mixture Models**.; New York, NY: M. Dekker, 1 ed., 1988.
 - [36] www.systemc.org
 - [37] CoCentric SystemC Compiler Manuals: Quick Reference; Behavioral User Guide; Behavioral Modeling Guide; RTL User and Modeling Guide. www.synopsys.com
 - [38] Power Compiler Manuals: Quick reference; Reference Manual; User Guide. www.synopsys.com
 - [39] Patterson, D.A., Hennessy, J.L., **Computer Architecture: A Quantitative Approach**, 2 Edition, Morgan Kaufman, 1996.
 - [40] Montanaro, J., Witek, R.T., Anne, K., Black, A.J., Cooper, E.M., Dobberpuhl, D.W., Donahue, P.M., Eno, J., Hoeppner, W., Kruckemyer, D., Lee, T.H., Lin, P.C.M., Madden, L., Murray, D., Pearce, M.H., Santhanam, S., Snyder, K.J., Stehpany, R., and Thierauf, S.C.: **A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor**, IEEE J. Solid-State Circuits, 1996, 32, (11), pp. 1703-1714.
 - [41] 'Power products reference manual V3.5'. Synopsys, 1996.
 - [42] Anderson, T., and Agarwala, S. **Effective hardware-based two-way loop cache for high-performance low-power processors**. Proc. Int. Conf. on Computer Design, 2000, pp. 403-407.
 - [43] Kin, J., Gupta, M., and Mangione-Smith, W.H. **Filtering memory references to increase energy efficiency**, IEEE Trans. Comput., 2000, 49, (1), pp. 1-15.
 - [44] Chunho Lee, Miodrag Potkonjak and William H. Mangione-Smith; **MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems**. Proceedings of the 30th Annual International Symposium on Microarchitecture, pp. 330-335, Dec. 1997.
-

-
- [45] Tang, W., Gupta, R., and Nicolau, A.: 'Design of a predictive filter cache for energy savings in high-performance processor architectures'. Proc. Int. Conf. on Computer Design, 2001, pp. 68–73
 - [46] Tang, W., Gupta, R., and Nicolau, A.: 'Power savings in embedded processors through decode filter cache'. Proc. Int. Conf. on Design Automation & Test in Europe, March 2002, pp. 443–448
 - [47] Yeh, T.Y., and Patt, Y.N.: 'Alternative implementation of two-level adaptive branch prediction'. Proc. 19th Int. Symp. on Computer Architecture, 1992, pp. 124–124
 - [48] Yoaz, A., Erez, M., Ronen, R., and Jourdan, S.: 'Speculation techniques for improving load related instruction scheduling'. Proc. 26th Int. Symp. on Computer Architecture, May 1999, pp. 42–53
 - [49] www.simplescalar.com
 - [50] Shivakumar, P., and Jouppi, N.: 'An integrated cache timing, power and area model'. Tech. Report, Compaq Western Research Lab, Palo Alto, CA, 2001/2
 - [51] Ernst, D., Austin, T.M., Mudge, T., and Brown, R.B.: 'MiBench: a free commercially representative embedded benchmark suite'. Proc. 4th IEEE Int. Workshop on Workload characterization, Dec. 2001, pp. 3–14
 - [52] Synopsys CoCentric Compiler Behavioral User and Modeling Guide, v2002.05 <http://www.synopsys.com>
 - [53] E. Bellas, I. N. Hajj and C. D. Polychronopoulos, 'Using dynamic cache management techniques to reduce energy in general purpose processors', IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 8, No. 6, pp. 693 - 708 December 2000
 - [54] J. MacQueen, **Some methods for classification and analysis of multivariate observations**, in: L.M. LeCam, J. Neyman (Eds.), Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1—Statistics, UCLA Press, Berkeley, USA, 1967, pp. 281–297.
 - [55] Forgy, **Cluster analysis of multivariate data. Efficiency vs. Interpretability of classification**. WNAR Meetings, UCLS Riverside, CA, June 22–23, 1965. Abstract in Biometrics 21 (1965) 768.
 - [56] Bhattacharyya S., Srikanthan T. **K-means Clustering Algorithm - A Hardware Perspective**, *WSES International Conference on Speech Signal and Image Processing*, September 1-6, 2001. Malta. pp. 179-184. ISBN 960-8052-42-4
 - [57] K.Vivekanandarajah, T.Srikanthan, S. Bhattacharyya, P. Kannan. **Incorporating Pattern Prediction Technique for Energy Efficient Filter Cache Design**. The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications, pp 44-47, 2003.
 - [58] Kugan Vivekanandarajah, Thambipillai Srikanthan, Saurav Bhattacharyya. **Energy-delay efficient filter cache hierarchy using pattern prediction scheme**. *IEE Proceedings - Computers and Digital Techniques*. 2003
 - [59] Kugan Vivekanandarajah, Thambipillai Srikanthan, Christopher Turrel Clarke, Saurav Bhattacharyya. **A Novel Static Prediction Scheme for Filter Cache Structures**. Accepted for publication in *IEICE Transactions on Electronics, Special Issue on Low-Power System LSI, IP and Related Technologies*. April 2004.
-

-
- [60] Kevin J. Nowka, Gary D. Carpenter, Eric W. MacDonald, Hung C. Ngo, Bishop C. Brock, Koji I. Ishii, Tuyet Y. Nguyen, Jeffrey L. Burns. **A 32-bit PowerPC System-on-a-chip With Support for Dynamic Voltage Scaling and Dynamic Frequency Scaling.** IEEE Journal of Solid-State Circuits, Vol 37. No 11 November 2002.

Books & Technical Reports

- I. Chandrakasan A.P., **Low Power Digital CMOS Design**, Ph.D Dissertation, University of California at Berkeley, UMI Publications 1994.
- II. International Technology Roadmap for Semiconductors - Design, 2001 Edition
- III. Thomas D. Burd. **Low-power CMOS Library Design Methodology**, M.Sc. Thesis – U.C. Berkeley – 1994.
- IV. Hirsch, Ronny, Low-Power **Standard Cell Library for Synthesis**, M.Engg. Thesis – Carleton University, Ottawa, Canada – 1995.
- V. Cherkauer, Brian Scott, CMOS-based Architectural and Circuit Design Techniques for application to High Speed, Low Power Multiplication. Ph.D. Dissertation – University of Rochester, NY – 1995.
- VI. Raghunathan, A., Jha K. Niraj, Dey S. **High-Level Power Analysis and Optimization.** Kluwer Academic Publishers. ISBN – 0-7923-8073-8
- VII. Jan M. Rabaey. **Digital Integrated Circuits – A Design Perspective.** Prentice Hall Series. ISBN – 0-13-178609-1
- VIII. Chou, Tan-Li. Accurate Power Estimation of CMOS Digital Circuits and its application to Low-Power Digital Logic Synthesis. Ph.D. Thesis – Purdue University – 1996.
- IX. Benini, Luca. Automatic Synthesis of Sequential Circuits for Low Power Dissipation. Ph.D. Dissertation – Stanford University – 1997.
- X. Kaplan, John J. **Speed-Independent Asynchronous Sequential Circuit Design.** Ph.D. Dissertation – University of New Mexico – 1997.
- XI. Landman, Paul Eric. **Low-Power Architectural Design Methodologies.** Ph.D. Thesis – U.C. Berkeley – 1994.
- XII. Jan M. Rabaey, Massoud Pedram. **Low Power Design Methodologies.** Kluwer Academic Publications. ISBN 0-7923-9630-8.
- XIII. <http://www.acpi.info/index.html>

APPENDIX A

First Authored Publications by the Author

Industry Conferences Publications

1. Saurav Bhattacharyya, Quek Kai Hock, Dr. T Srikanthan, Ashish Panda, **Compute Intensive DSP Applications using SystemC: A Case Study Using Speaker Verification System**, *Synopsys Users Groups Conference* – Singapore, September 2003.
2. Saurav Bhattacharyya, Srikanthan T., Quek K.H., Lam S.K. **SystemC for Architecture Exploration and Energy Centric Power Management**. *SNUG [Synopsys Users Group]* Singapore Conference organized by Synopsys – 4 June 2002. Singapore. [Page numbers not applicable]. Online at http://www.snug-universal.org/asia/asia_sgpr.htm

Academic Conference Publications

3. Saurav Bhattacharyya, T. Srikanthan, K. Vivekanandarajah, “**Area and Power Efficient Pattern Prediction Architecture for Filter Cache Access Prediction in the Instruction Memory Hierarchy**”, VLSI International Symposium on Technology, System and Applications (VLSI-TSA), Hsinchu, Taiwan. April 27-29, 2005.
4. Bhattacharyya S., Srikanthan T., Krishnamurthy P. **Speaker Verification - A VLSI Perspective**. *Communication Systems, Networks and Digital Signal Processing* – 15th-17th July, 2002 (CSNDSP-2002). Staffordshire UK.
5. Bhattacharyya S., Srikanthan T. **Using High-Level Language for Rapid Prototyping of FPGA Based Solution**, *9th International Symposium on Integrated Circuits, Devices & Systems* (ISIC-2001). 3-5 September 2001, Singapore.
6. Bhattacharyya S., Srikanthan T. **K-means Clustering Algorithm - A Hardware Perspective**, *WSES International Conference on Speech Signal and Image Processing*, September 1-6, 2001. Malta. pp. 179-184. ISBN 960-8052-42-4

Industry Magazine Articles & Whitepapers

7. Dennis Nye, Bhattacharyya S., T. Srikanthan, **Rapid Prototyping-Losung: System Modellierung mit Handel-C**. *Design & Verification Magazine*, pages 55-58, June-July/2002 Issue. Adapted to German Language from original manuscript in English. Germany.

-
8. Bhattacharyya S., Srikanthan T. **New Approach to Rapid Prototyping of Area-Time Efficient FPGA Solutions**. Whitepaper submitted to Celoxica, UK.

Co-authored publications related to this research

As a part of a team at the Centre for High Performance Embedded Systems, the author also published related work which is listed as below:

Industry Conferences Publications

1. Quek Kai Hock, Saurav Bhattacharyya, Dr. T Srikanthan, **A Methodology for FPGA Based Implementations Using SystemC Based Designs**, *Synopsys Users Groups Conference* – Singapore, September 2003.

Academic Conference Publications

2. K. Vivekanandarajah, T. Srikanthan, S. Bhattacharyya, **“Dynamic Filter Cache for Low Power Instruction Memory Hierarchy”**, EUROMICRO Symposium on Digital System Design, IEEE CS Press, Rennes, France, pp. 607-610. August 31st – September 4th, 2004.
3. Quek Kai Hock, Saurav Bhattacharyya, Dr. T Srikanthan, **A Methodology for FPGA Based Implementations Using SystemC Based Designs**, *Synopsys Users Groups Conference* – Singapore, September 2003.
4. Kugan Vivekanandarajah, Thambipillai Srikanthan, Saurav Bhattacharyya. **Decode Filter Cache for Energy Efficient Instruction Cache Hierarchy in Super Scalar Architectures**. In proceedings IEEE/ACM Asia South Pacific Design Automation Conference (ASP-DAC) 2004. Pages 373-379. January 27-30, 2004. Yokohoma, Japan.
5. Kugan Vivekanandarajah, Dr. T Srikanthan, Dr. C.T. Clarke, Saurav Bhattacharyya, **Static Pattern Predictor Based Low Power Instruction Cache Design**, In *Proc. 3rd Embedded Systems and Application Conference 2003 (ESA'2003), MLPD 2003*, Las Vegas, Nevada, USA, pp. 210-215. June 2003.
6. Kugan Vivekanandarajah, Thambipillai Srikanthan, Saurav Bhattacharyya, Prasanna Venkatesh Kannan, **Incorporating Pattern Prediction Technique for Energy Efficient Filter Cache Design**, IWSOC 2003, Calgary, Canada. In *Proc. IEEE International Workshop on System-on-Chip for Real-Time Applications IWSOC 2003*, IEEE CS Press, Alberta, Canada, June/July 2003.

-
7. Kugan Vivekanandarajah, Thambipillai Srikanthan, Saurav Bhattacharyya, Prasanna Venkatesh Kannan. **Incorporating Pattern Prediction Technique for Energy Efficient Filter Cache Design.** Special issue of WSEAS Press

International Journal Publications

9. Kugan Vivekanandarajah, Thambipillai Srikanthan, Saurav Bhattacharyya. **Energy-delay efficient filter cache hierarchy using pattern prediction scheme.** *IEE Proceedings - Computers and Digital Techniques*. 2003
10. Kugan Vivekanandarajah, Thambipillai Srikanthan, Christopher Turrel Clarke, Saurav Bhattacharyya. **A Novel Static Prediction Scheme for Filter Cache Structures.** Accepted for publication in *IEICE Transactions on Electronics, Special Issue on Low-Power System LSI, IP and Related Technologies*. April 2004.
11. Kugan Vivekanandarajah, Thambipillai Srikanthan, Saurav Bhattacharyya. **Dynamic Management Scheme for Predictive Instruction Filter Cache Hierarchy.** Submitted to *IEEE Transactions on VLSI*

APPENDIX B

```

#include "systemc.h"
#define cM 16
#define cD 8
#define cT 6000
#define cMaxIter 10
SC_MODULE (kmeans) {
    //Module port declarations
    void TOP();
    //Module constructor
    SC_CTOR (kmeans) {
        SC_CTHREAD (TOP, pClk.pos());
        watching(pReset.delayed() == true);
    }
};

void kmeans::TOP(){
    sc_int<32> summation_mean[(cM*cD)+1];
    sc_int<32> data_vectors_in_cluster[cM+1];
    sc_int<32> mean[(cM*cD)+1];
    <INITIALIZATIONS IN RESET LOOP>...
    wait();
    lbl_F_LOOP1: for (int index=1; index<=cM*cD; index++) {
        <STATEMENTS>
    }
    lbl_W_LOOP2: while( iterations < cMaxIter) {
        <STATEMENTS>
        lbl_F_LOOP2: for(int m=1; m<=cM; m++){
            <STATEMENTS>
            lbl_F_LOOP3: for(int d = 1; d <= cD; d++){
                <STATEMENTS>
            }
            wait();
        }
        <STATEMENTS>
        lbl_F_LOOP4: for(int TrVectIndex = 1; TrVectIndex <= cT; TrVectIndex++){
            <STATEMENTS>
            lbl_F_LOOP5: for (int d=1; d<=cD; d++){
                <STATEMENTS>
            }
            <STATEMENTS>
            lbl_F_LOOP6: for(int m=1; m<=cM; m++){
                <STATEMENTS>
                lbl_F_LOOP7: for (int d=1; d<=cD; d++){
                    difference = (sDataVector[d] - mean[partial_distance_index]);
                    square = (difference * difference);
                    partial_distance = partial_distance + square;
                    partial_distance_index++;
                    wait();
                }
                wait();
                <STATEMENTS>
                lbl_F_LOOP8: for (int d=1; d<=cD; d++){
                    index_to_mean = summation_index + d;
                    summation_data = summation_mean[index_to_mean];
                    sum = summation_data + sDataVector[d];
                    summation_mean[index_to_mean] = sum;
                    wait();
                }
                wait();
            }
            <STATEMENTS>
            lbl_F_LOOP9: for(int m=1; m<=cM; m++){
                <STATEMENTS>
                lbl_F_LOOP10: for (int d=1; d<=cD; d++){
                    <STATEMENTS>
                }
                wait();
            }
            <STATEMENTS>
            wait();
        }
        wait();
    }
}

```