High level synthesis of VLSI systems for low power

Xing, Xianwu

2008

Xing, X. (2008). High level synthesis of VLSI systems for low power. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/18772>

<https://doi.org/10.32657/10356/18772>

HIGH LEVEL SYNTHESIS OF VLSI SYSTEMS FOR LOW POWER

XING XIANWU

School of Electrical and Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

2008

Acknowledgements

I would like to thank my supervisor, Associate Professor Jong Ching Chuen, for his efficient supervision, continuous instructions, support and easy-going communication throughout the entire research program. Particularly, I wish to thank him for helping me understanding the philosophies of research and for teaching all the intangible skills in the research work. These skills are not only very helpful in the research program, but also invaluable for my future career.

I would also like to thank Dr. Gu Jiangming for the thorough discussions about all the technical problems. I would like to thank Dr. Xu Fei, Dr. Luo Jianwen and Mr. Xia Fei for their helps and discussions.

To the staff and other students in the Center for Integrated Circuits and Systems and the Integrated Systems Research Laboratory, I wish to express my appreciation to all of them for their kind and friendly assistance.

I would like to convey my gratitude to my parents and my wife Cui Jie for their consistent support and encouragement during all periods of my study and my life.

Table of Contents

ACKNOWLEDGEMENTS..... I

TABLE OF CONTENTS..... II

SUMMARY..... V

CHAPTER 1 INTRODUCTION.....1

1.1 MOTIVATION1

1.2 OBJECTIVES2

1.3 MAJOR CONTRIBUTIONS.....3

1.4 ORGANIZATION6

CHAPTER 2 LITERATURE REVIEW.....8

2.1 INTRODUCTION TO EDA AND HIGH LEVEL SYNTHESIS8

2.2 POWER CONSUMPTION MODEL AND POWER OPTIMIZATION TECHNIQUES11

2.3 SWITCHING ACTIVITY REDUCTION TECHNIQUES.....14

2.3.1 Switching Activity Estimation14

2.3.2 Switching Activity Reduction Techniques16

2.4 VOLTAGE SCALING AND FREQUENCY SCALING.....19

2.4.1 Multi-Voltage High Level Synthesis.....20

2.4.1.1 Layout Issues20

2.4.1.2 Supply Voltage Generator.....21

2.4.1.3 Level Converters.....21

2.4.2 Multi-Voltage Synthesis Techniques22

2.4.3 Frequency Scaling Techniques25

2.5 INTEGRATING LOW POWER HIGH LEVEL SYNTHESIS WITH PHYSICAL SYNTHESIS27

2.5.1 Introduction to Floorplanning.....28

2.5.2 Floorplan-Driven Low Power High Level Synthesis29

2.6 HIGH LEVEL TRANSFORMATION TECHNIQUES IN HIGH LEVEL SYNTHESIS32

2.6.1 Low Power High Level Transformation Techniques.....33

2.6.2 Introduction to Symbolic Computer Algebra35

2.6.3 Using Symbolic Computer Algebra in High Level Transformation Techniques.....37

2.7 SUMMARY38

CHAPTER 3 LOOK-AHEAD SYNTHESIS FOR SWITCHING ACTIVITY REDUCTION41

3.1 INTRODUCTION41

3.2 LOW POWER SCHEDULING AND BINDING VIA SWITCHING ACTIVITY REDUCTION.....42

3.2.1 Switching Activity Profiling.....43

3.2.2 SA Reduction in Resource Binding46

3.2.3 SA Reduction in Operation Scheduling.....48

3.3 ALGORITHM FORMULATION OF LAS49

3.3.1 Resource-Constrained Look-Ahead Synthesis (RC-LAS).....50

3.3.2 Latency-Constrained Look-Ahead Synthesis (LC-LAS)56

3.3.3 Register Binding59

3.3.4 Operand Matching Optimization62

3.4 AN ILLUSTRATIONAL EXAMPLE62

3.5 EXPERIMENTAL RESULTS65

3.5.1 Data Flow Graph Construction.....65

3.5.2 Experiments on RC-LAS67

3.5.3 Influence of Look-Ahead and Backtracking.....70

Table of Contents

3.5.4 Experiments on LC- LAS.....	71
3.5.5 Influence of Register Binding.....	72
3.5.6 Algorithmic Complexity	74
3.5.7 Comparison.....	75
3.6 SUMMARY	77
CHAPTER 4 VOLTAGE SCALING AND FREQUENCY SCALING FOR ENERGY OPTIMIZATION	78
4.1 INTRODUCTION	78
4.2 MUVOF SYNTHESIS FOR FUNCTIONALLY PIPELINED DATAPATH	79
4.2.1 Multi-Frequency Functionally Pipelined Datapath Model.....	80
4.2.2 Multi-Voltage and Multi-Frequency Synthesis	82
4.2.3 Influence of State Partitioning and Scheduling	88
4.3 ALGORITHM FORMULATION OF MUVOF	92
4.3.1 Core process of the Algorithm	92
4.3.1.1 Stage Partition.....	93
4.3.1.2 Multi-Voltage Assignment.....	97
4.3.2 The Iterative Process of the Algorithm	101
4.3.3 Clock Period Extension.....	104
4.4 EXPERIMENTAL RESULTS	106
4.4.1 Experiments under Tight Constraints	106
4.4.2 Experiments under Loose Constraints	108
4.4.3 Influence of Resource Constraints and Throughput Constraints.....	111
4.4.3.1 Influence of Resource Constraints	111
4.4.3.2 Influence of Throughput Constraints	113
4.4.4 Comparison.....	116
4.4.4.1 Average Improvement Comparison	116
4.4.4.2 Benchmark Comparison.....	117
4.5 SUMMARY	117
CHAPTER 5 FLOORPLAN-DRIVEN LOW ENERGY HIGH LEVEL SYNTHESIS FOR MULTI-VOLTAGE DATAPATH.....	119
5.1 INTRODUCTION	119
5.2 INTERCONNECT POWER OPTIMIZATION AND MULTI-VOLTAGE DESIGN.....	120
5.2.1 Impact of Floorplanning on Power.....	120
5.2.2 Multi-voltage Design	123
5.2.3 Floorplan Optimization for Multi-voltage Design	126
5.3 ALGORITHM FORMULATION OF FLOM.....	127
5.3.1 Algorithm Flow.....	127
5.3.2 Outer-Level Iterative Algorithm.....	129
5.3.3 Local Moves in the Outer-Level Algorithm.....	130
5.3.4 Inner-level Floorplanning Algorithm for Multi-Voltage Design	134
5.4 EXPERIMENTAL RESULTS	139
5.4.1 Experimental Results on Synthesis.....	140
5.4.2 Experimental Results on Synthesis and Floorplanning.....	142
5.4.3 Floorplan-unaware VS Floorplan-aware Low Power Synthesis	144
5.4.4 Algorithmic Complexity	146
5.4.5 Comparison.....	147
5.5 SUMMARY	149
CHAPTER 6 APPLICATION OF SYMBOLIC COMPUTER ALGEBRA IN HIGH LEVEL SYNTHESIS	151
6.1 INTRODUCTION	151
6.2 HIGH LEVEL TRANSFORMATION TECHNIQUES BASED ON SCA	153
6.2.1 Symbolic Computer Algebra	153
6.2.2 Factorization of Subexpressions	154

Table of Contents

6.2.3 Mapping Complex Components	156
6.3 PROPOSED SUBEXPRESSION FACTORIZATION	158
6.3.1 Problem Formulation	158
6.3.2 Factorization Algorithms	163
6.3.2.1 Subexpression Factorization	163
6.3.2.2 Polynomial Factorization	165
6.4 PROPOSED COMPLEX COMPONENTS MAPPING	167
6.4.1 Problem Formulation	167
6.4.2 Mapping Algorithm	169
6.5 RELEVANT TRANSFORMATION TECHNIQUES	172
6.5.1 Term Merging and Coefficient Expansion	172
6.5.2 Tree Height Reduction	172
6.5.3 Term Factorization	174
6.5.4 MAC Mapping	175
6.6 EXPERIMENTAL RESULTS	177
6.6.1 Experiments on Subexpression Factorization	177
6.6.2 Experiments on Complex Components Mapping	181
6.6.3 Comparison	184
6.7 SUMMARY	185
CHAPTER 7 CONCLUSIONS AND FUTURE WORKS	187
7.1 CONCLUSIONS	187
7.2 SUGGESTIONS FOR FUTURE WORK	190
7.2.1 Adopt More Accurate Power/Energy Models	190
7.2.2 Close the Gap between High Level Synthesis and Physical Synthesis	192
7.2.3 Low Power High Level Synthesis for Control Intensive Applications	193
7.2.4 An Integrated Low Power/Energy High Level Synthesis Framework	194
REFERENCE	196

Summary

In this thesis, circuit parameters that are related to low power/energy high level synthesis for VLSI systems are studied. The identified parameters include switching activity, supply voltage, frequency and capacitance. Several high level synthesis algorithms are developed to optimize one or more of these parameters for low power/energy during high level synthesis.

In order to reduce the switching activity, a novel technique named Look-Ahead Synthesis (LAS) is proposed. This technique performs the switching activity reduction through an integrated process of scheduling and binding, employing the concept of look-ahead, backtracking and weighted bipartite matching. Special attention is given to register optimization in LAS, which adopts a register management technique to eliminate spurious switching and uses the bipartite matching to perform the register binding to reduce the switching activity on registers. The experimental results show that LAS is able to reduce the switching activity by 54.8% on average under the resource constrained synthesis, although its effectiveness varies from design to design.

In order to optimize the supply voltage and frequency, a technique named Multi-voltage Multi-frequency synthesis (MuVoF) is proposed. This technique targets energy optimization of functionally pipelined datapath with multiple supply voltages and multiple frequencies under resource and throughput constraints. MuVoF consists of two steps. The first step performs pipeline stage partitioning and multi-voltage assignment to obtain an optimal initial result on an initial schedule, and the second step iteratively refines the result to relieve the impact of non-optimal initial schedule. When tested with a set of benchmarks, MuVoF achieves 50% energy reduction on average, which shows that the multi-voltage multi-frequency technique can be effective but the complexity in circuit design will be increased due to the multi-voltage multi-frequency datapath.

In order to optimize the energy on interconnects, a technique named Floorplan-driven Multi-voltage synthesis (FloM) is proposed, which performs floorplan-driven high level synthesis targeting multi-voltage datapath for low energy. The technique adopts a two-level iterative approach, aiming to improve the optimization results by more accurately estimating the energy consumption on wires. In the inner level, the floorplanning for multi-voltage datapath is performed by iteratively refining the floorplan for energy optimization on wires. In the outer level, the multi-voltage datapath is iteratively optimized in terms of energy consumption on both the datapath and the connecting wires based on a simulated annealing algorithm. The two nested iterative algorithms work together to form FloM, achieving an optimized datapath and an optimized floorplan such that the total power consumption is minimized. Experimental results show that 42.4% energy reduction in datapath and 42.6% reduction in wires are achieved. Note that this technique has the drawback of increased complexity in circuit design due to the multi-voltage datapath.

High level transformation techniques are usually applied at high level, at which level accurate estimation of the cost in the real circuits are difficult to compute. However, high level transformation techniques can be used to reduce the total number of operations (and thus the total switching activity) as well as to optimize the capacitance of resources. Two novel low power high level transformation techniques, Subexpression Factorization and Complex Components Mapping, are proposed in the thesis. By evaluating different subexpressions of an input polynomial, Subexpression Factorization finds the energy-optimized factorized subexpressions from a set of possible subexpressions. On the other hand, Complex Components Mapping finds the energy-optimized mapping of an input polynomial to pre-optimized complex modules, such as square, sine and cosine functional modules. It also incorporates other transformation techniques, such as Term Merging, Coefficient Expansion, Tree Height Reduction (THR), Term Factorization and Multiplier/Accumulator (MAC) mapping. The Subexpression Factorization is able to reduce the energy by 63.7% averagely when tested with a set of benchmarks while the Complex Components Mapping can achieve 29.5% reduction on average. For both

techniques, the critical path delay worsens by 8.3% for the Subexpression Factorization and 2.6% for the Complex Components Mapping.

The concepts, formulations, developments and testing of these proposed algorithms are documented in this thesis, which also includes the detailed experimental results and the comparisons with the results published in the literature.

Chapter 1 Introduction

1.1 Motivation

Ever since its introduction in the 1960's, the Integrated Circuit (IC) technology has been proven much smaller and power efficient than the conventional discrete components and quickly dominated the research area of electronics. With the wide spread of IC applications, the integration density of ICs has been growing tremendously. The well-known Moore's Law predicts that the number of transistors in a single IC chip will double every 18 months. To date, the increase in transistor count doubles every year. In the so-called Very Large Scale Integration (VLSI) technology, the transistor count of a chip can easily reach hundreds of thousands. The Ultra Large Scale Integration (USLI) technology also emerges, in which a single chip can easily integrate 1 million to 5 million transistors. It is estimated that by the end of 2010's, a System-on-Chip (SoC) will accommodate up to 4 billion transistors by using 50nm transistors operating below one volt [BEN02].

To handle ICs with such a large transistor count, today's IC designers largely depend on the help of Electronic Design Automation (EDA) tools to perform both designs and verifications. Modern EDA tools are very powerful and versatile. The available tools cover high level synthesis, logic synthesis, physical synthesis, functional simulation, functional verification, etc. For the front-end design, VLSI designers usually describe the systems in the Register Transfer Level (RTL), and let EDA tools perform the RTL synthesis, logic synthesis and physical synthesis. However, as the complexity of IC applications continues to grow, it is preferred to describe the designs at the behavioral level and to have EDA tools to perform the remaining work, whenever it is applicable. The advantages of this design flow are two folds. First, time-to-market is reduced, because the behavioral level descriptions are much easier to design and verify. Second, the designs can be optimized at a higher level of abstraction so that better results can be obtained. As a result, High Level Synthesis, which translates behavioral level

descriptions to RTL structures, has been a very active research area ever since its introduction in the 1980's [DEM94].

On the other hand, due to the ever-increasing demands in portable and wireless devices, such as cellular phones, Personal Digital Assistants (PDA) and laptop computers, power and energy consumption has become the major design constraints in many modern IC applications. The reasons can be generally divided into two: portability and reliability. The portability refers to lengthen the battery life, because many portable systems require continuously power supply from batteries. Reliability is another major reason for having low power/energy designs. This is mainly because high power dissipation systems generate large amount of heat. This problem is especially critical because of the high complexity and high frequency of the modern digital applications. Besides, as described in [DAS98], submicrometer scaling increases current density in modern technology. This makes the electromigration (EM), which is a major cause of failure in ICs, much more likely to happen.

In a nutshell, high level synthesis has gained much popularity since it was first introduced more than two decades ago. In the meantime, power dissipation has replaced area and performance as the most important and daunting design constraints in many applications. To reduce power dissipation more efficiently, power optimization must be performed at very high level of abstraction. This stimulates the advent of low power/energy high level synthesis, which has been widely explored in recent years and is the focus of this work.

1.2 Objectives

The objectives of this work can be summarized as follows:

- (1) To study the existing power dissipation models and to identify the parameters to be optimized in order to reduce power consumption.

According to the power models of CMOS circuit, there are several parameters that are related to the power dissipation. They include switching activity, load capacitance, supply

voltage as well as operating frequency. In order to minimize power, one or more of these parameters must be optimized in one way or another.

(2) To propose and develop high level synthesis algorithms for power/energy optimization.

Embedded with power dissipation models, high level synthesis algorithms can translate behavioral descriptions into more power efficient RTL structures. In this work, several high level synthesis techniques are proposed to perform power optimization via switching activity reduction, supply voltage scaling, frequency scaling, floorplan-driven synthesis, as well as high level transformations. The targeted datapath types include multi-cycle datapath, multi-voltage datapath and multi-frequency datapath as well as functionally pipelined datapath. Experiments are to be carried out to investigate the effectiveness of the proposed techniques. The results obtained are to be analyzed to study the efficiency of these techniques.

1.3 Major Contributions

The major contributions of the work consist of the development of several high level synthesis techniques for achieving low power/energy by means of: (1) switching activity reduction, (2) supply voltage scaling and frequency scaling, (3) integration of high level synthesis and physical synthesis, and (4) high level transformations, as elaborated below. All contributions specifically focus on dynamic, capacitive power/energy dissipation and not on leakage or static power/energy dissipation.

In the work on the switching activity reduction, a novel technique named Look-Ahead Synthesis (LAS) is proposed. This technique performs switching activity reduction through an integrated process of scheduling and binding. It employs a concept of look-ahead, backtracking and weighted bipartite matching. During the list-scheduling-based scheduling process, several control steps are looked ahead and the bipartite-matching-based binding is performed to obtain the optimal binding for these control steps to achieve the minimal switching activity. Then backtracking is performed to limit the local

minima. This process is repeated until a complete synthesis solution is obtained. Special attention is given to the register optimization in LAS, which adopts a register management technique to eliminate spurious switching and uses bipartite matching to perform the register binding to reduce the switching activity on registers.

In the work on the supply voltage scaling and frequency scaling, a technique named MuVoF is proposed. This technique targets the power optimization of functionally pipelined datapath with multiple supply voltages and multiple frequencies under resource and throughput constraints. MuVoF consists of two steps. The first step performs pipeline stage partitioning and multi-voltage assignment to obtain an optimal initial scheduling. The second step iteratively refines the result. Other techniques such as Clock Period Extension are also incorporated in MuVoF to achieve better results. As far as the author is aware, this is the first proposed work on low power high level synthesis targeting functionally pipelined datapath with multi-voltage and multi-frequency.

In the work on the integration of high level synthesis and physical synthesis, it is demonstrated that it is extremely important to incorporate physical level information in high level synthesis. A technique named FloM is proposed to perform floorplan-driven high level synthesis targeting multi-voltage datapath for low power. The technique adopts a two-level iterative process. In the inner level, the floorplanning for the multi-voltage datapath is performed by iteratively refining the floorplan to optimize the power on the wires. In the outer level, the multi-voltage datapath is iteratively optimized in terms of power consumption on both the functional units and the connecting wires. The two nested iterative algorithms work together in FloM, to achieve an optimized datapath and an optimized floorplan where the total power consumption is minimized.

Lastly, in the work on the high level transformations, two novel low power high level transformation techniques, namely the Subexpression Factorization and the Complex Components Mapping, are proposed. Performed before high level synthesis, the high level transformations aim to modify the DFG to give high level synthesis a good starting point. The foundation of the two techniques, Symbolic Computer Algebra is introduced,

by the example of the famous Maple tool. In the two techniques, the input behavioral level descriptions are formulated as polynomials. By finding the best subexpressions of an input polynomial, the Subexpression Factorization finds its best factorization form in terms of power. On the other hand, the Complex Components Mapping finds the best subexpressions of an input polynomial in order to map it to pre-optimized complex modules, such as squarer, sine and cosine functional modules. By integrating them with other transformation techniques, such as Term Merging, Coefficient Expansion, THR, Term Factorization and MAC mapping, the two techniques are able to achieve considerable improvements on mapping of the input polynomial to DFG.

Figure 1.1 illustrates where the proposed techniques fit into a typical VLSI synthesis flow. It can be seen that the Subexpression Factorization and the Complex Components Mapping belongs to high level transformation techniques, which are performed right before high level synthesis. LAS and MuVoF are typical high level synthesis techniques. FloM is a high level synthesis technique with back-annotated physical information. It performs floorplanning without detailed logic mapping (logic synthesis).

Figure 1.1 A Typical VLSI Synthesis Flow

1.4 Organization

The thesis consists of seven chapters.

Chapter 1 introduces the motivation and objectives of the research. Major contributions of the research work are also summarized.

Chapter 2 gives a survey of the literature relating to the research. EDA and high level synthesis are briefly introduced. Focus is given to low power high level synthesis. The importance of power estimation models is explained and existing works on power estimations are reviewed. This is followed by a review of various low power high level synthesis techniques, including switching-activity-based, supply-voltage-scaling-based, frequency-scaling-based, floorplan-driven-based as well as transformation-based techniques. Other relevant topics, such as switching activity estimation, multi-voltage issues, floorplanning and Symbolic Computer Algebra are also discussed.

Chapter 3 describes the first proposed technique LAS, which optimizes power/energy via switching activity reduction. The concept of Switching Activity Table is introduced and the method to obtain it is described. This is followed by a detailed description of the algorithms, including look-ahead, backtracking, bipartite matching and spurious switching elimination. An illustration example is included. Several experiments were carried out and the effects of several parameters of LAS are studied. The results obtained from the experiments and the comparisons with published works are presented.

The novel technique MuVoF is proposed in Chapter 4. The functionally datapath model with multiple supply voltages and multiple frequencies is first described. The influence of the stage partitioning and scheduling on power consumption is investigated. This is followed by the algorithm formulation of MuVoF, including the stage partition process, the voltage assignment process, as well as the iterative refinement process. Experiments were performed to show the effectiveness of MuVoF and the effect of resource

constraints and throughput constraints. The experimental results are compared with several existing works.

Chapter 5 focuses on the proposed FloM technique, which integrates floorplanning into high level synthesis for the optimization of wire power dissipation and total power dissipation. The impact of floorplan on power is studied. The problem raised by multi-voltage datapath is also studied. This is followed by the description of the algorithm of FloM. The local moves to perform rescheduling, rebinding and voltage reassignment are explained in detail, which is followed by the floorplanning algorithm. Finally, experimental results and comparisons with existing works are presented.

Chapter 6 presents the two transformation techniques, the Subexpression Factorization and the Complex Components Mapping. Symbolic Computer Algebra, which is the foundation of the two proposed techniques, is introduced. The originality of the two techniques is also discussed, followed by the detailed formulations of the algorithms, including some transformation techniques that are integrated with the two techniques, e.g, Term Merging, MAC mapping, etc. The effectiveness of the two techniques is analyzed and demonstrated through experiments on the test benchmarks. The experimental results and comparisons are presented.

Lastly, Chapter 7 summarizes the various low power high level synthesis techniques proposed in the thesis, followed by some recommendations for future research work.

Chapter 2 Literature Review

The focus of this thesis is on techniques for low power/energy high level synthesis. In this chapter, a literature review on various related research works that have been published is presented. This chapter is organized as follow. In Section 2.1, a brief introduction of modern Electronic Design Automation (EDA) and High Level Synthesis is given. Section 2.2 introduces power consumption model of CMOS circuits and gives an overview of power optimization techniques on different levels of the design abstraction. Section 2.3 to Section 2.6 describe techniques to reduce power/energy based on switching activity reduction, multi-voltage multi-frequency, physical-level optimization and high level transformations respectively. A summary is given in Section 2.7.

2.1 Introduction to EDA and High Level Synthesis

The past two decades have witnessed a tremendous growth in the area of Electronic Design Automation (EDA) in the field of electronic circuits and system. [MAC00] gives a good survey of EDA in the industry, including physical design, simulation/verification, synthesis and test. For design synthesis, today's tools for Register Transfer Level (RTL) synthesis, logic synthesis and physical synthesis are becoming more and more mature and being widely used in the industry. However, as the complexity of digital circuits and applications increases, more aggressive automation methodologies on the behavioral level and system level are demanded. This has become an active research area of EDA in recent years.

High level synthesis (behavioral synthesis) operates in a high-abstraction level in the EDA flow. The input is a behavioral (algorithmic) description, usually written in C or Hardware Description Language (HDL). High level synthesis translates the description into a structural RTL description, usually consisting of Functionality Units (FU), storage components and interconnections.

High level synthesis can be divided into several subtasks [ELE98, GER99, DEM94], which are interdependent. However, early researchers often treated them separately, because in this way it is much easier to define optimization problems and to develop solving techniques. To formulate the synthesis problems, an internal representation of the input specification is needed. The most popular representation is the Control/Data Flow Graph (CDFG). Control Flow Graph is for control synthesis, which is mostly treated as a subset of logic synthesis. In this project, DFG is used, which is especially useful in data-intensive applications, such as multimedia processing and digital signal processing.

Three main subtasks of high level synthesis [DEM94] are:

- (1) Operation Scheduling: Given a DFG $G(V, E)$, where V is a set of operation nodes, and E is a set of edges, a schedule of G is a function $\phi: V \rightarrow Z^+$, where Z^+ means the space of positive integer numbers, and $\phi(v_i) = t_i$ denotes the start time of the operation $v_i \in V$ such that $t_i \geq t_j + d_j, \forall i, j: (v_j, v_i) \in E$, in which d_j is the delay of operation v_j .

Scheduling problems can be divided into two categories: time-constrained and resource-constrained. Detailed information of various scheduling techniques can be found from [LEE89, PAU89, DEM94, GER99].

- (2) Resource Allocation: This task allocates a set of resources R , on which all operations will execute. It determines which and how many resources should be used. Description of this task can be found in [GER99].
- (3) Resource Binding (Assignment): A resource binding is a mapping $\beta: V \rightarrow R \times Z^+$, where $\beta(v_i) = (t, r)$ denotes that the operation v_i with type $T(v_i) = t$, is implemented by the r th instance of resource type $t \in R$ for each $i = 1, 2, \dots, n_{ops}$. Usually, the resource binding includes the binding of FUs, storage elements (registers and memories) and interconnections (multiplexers and buses). More binding techniques can be found in [TSE86, SPR94, KUC90, JON95, DEM94, GER99].

Allocation and binding are intrinsically related, because a technique needs the allocated resources to perform binding. As a result, many published work treated allocation as part of binding to avoid statement redundancy. Throughout the thesis, the author adopts the same implication: when there is no resource constraint, allocation is internally performed for binding. This means that when there is no resource constraint, integrated allocation and binding is implicated when binding is referred.

As stated earlier, these three tasks are interdependent and can be performed in different sequences. Most high level synthesis systems perform scheduling before binding, because it leverages the complexity in the two tasks. However, to improve the quality of synthesis results and deal with other design space dimension, e.g. power consumption, techniques to integrate scheduling and binding are being explored. Early researchers formulated concurrent scheduling and binding as an Integer Linear Programming (ILP) problem [GEB92]. In recent years, there are many research works exploring the interactions between scheduling and binding in high-level low-power synthesis [LYU03, DAS98, KHO99, MUR03]. This is also a major objective and contribution of this work. Other active research areas include the integration of physical synthesis and high level synthesis [DAV05, FAN94], synthesis of control-intensive applications [KHO99, LAK99b, LAK99c], special structures [HAM01, KIT03, MAS03, WEI01b], multiple wordlength systems [CON01, KUM01], etc.

Due to the fast growth of FPGA-based applications, high level synthesis for FPGA architectures also gains significant attention. More information can be found from [CAR03, CHE03, DIE00, HUA00, MAK03, SRI01].

2.2 Power Consumption Model and Power Optimization Techniques

Traditionally, performance and area are the two major optimization targets for digital applications. As chip density and clock frequency of ICs are rapidly increasing, power consumption has become a very important optimization dimension in modern digital circuits and systems. This is especially true for portable and wireless devices, in which power consumption must be minimized to lengthen battery life. As a result, power optimization has been widely explored by designers and researchers in recent years.

Power optimization can be performed at different levels of the design cycle. [BEN00] gives a good tutorial on system-level power/energy optimization techniques on computation, communication and storage units. In [HEN02], a system-level power estimation and optimization platform is presented for embedded system design. In [HON99, JHA01] power optimization techniques, such as Dynamic Voltage Scaling (DVS) and Dynamic Power Management (DPM) for system-level SoC design are proposed and surveyed. [NAR98] tackles power minimization at logic level. [CHA95a] optimizes power during floorplanning. In this work, the focus is on high level synthesis for low power. Some literatures related to high level synthesis for low power are discussed below.

Accurate power modeling and estimation not only potentially achieve more power efficient results in power optimization techniques, but also give the designers an idea which part of the design is power intensive and needs more attention. Therefore much research work has been focusing on the area of power modeling and estimation on variable levels of abstraction. [MAC98] provides a thorough survey of power estimation and modeling techniques in the literature. In [LIU94], accurate power consumption estimation on logic gates interconnects, clock distribution, on chip memories, and off chip devices in CMOS VLSI is presented. [CHE00] develops a novel estimation technique based on Markov chains to more accurately estimate power sensitivities to primary inputs in CMOS sequential circuits. In [GUP00], the authors propose a power

macromodel, which consists of a single 4-D table, to captures the dependence of the power dissipation of combinational logic circuits on its input/output signal switching statistics. In [KRU01], an approach to calculate lower and upper bounds on the power consumption of a scheduled DFG is proposed for fast pre-estimation. [KUR02] presents an effective metric to evaluate the power dissipation of a scheduled DFG before resource binding is performed for early evaluation. In [NEM99], the authors first develop an empirical area model from a functional description of the logic circuit. Then based on the area model, the capacitance and thus power consumption is estimated. The authors in [RAG03] present efficient techniques for switching activity and power consumption estimation at Register Transfer Level (RTL), using a combination of macro-modeling for datapath blocks and control logic analysis techniques based on partial delay information.

In terms of high level synthesis, the most popular power model is derived as follow.

It is well-known that the power consumption of a CMOS circuit can be modeled as (2.1) [BEL95, CHA92].

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

Dynamic power, which is the power consumed when a signal transition occurs on the output, contributes a large part of the average power. For a gate that is synchronized by a clock, the dynamic power is modeled as (2.2).

$$P_{dynamic} = 0.5 \cdot V_{dd}^2 \cdot f \cdot C_{out} \cdot E_{transition} \quad (2.2)$$

where V_{dd} is the supply voltage, f is the frequency of the clock, C_{out} is the switched capacitance at the gate output and $E_{transition}$ is the average number of transitions occur on the gate output per clock cycle. The total dynamic power consumed by a digital circuit can be obtained by summing up all the dynamic power consumed by all the gates as given in (2.3).

$$P_{dynamic} = \sum_i 0.5 \cdot V_i^2 \cdot C_i \cdot E_i / T \quad (2.3)$$

where V_i is the supply voltage, C_i is the switched capacitance and E_i is the transitions happened during time T . For a synchronous circuit, the total power can be deduced with (2.4). And therefore, energy consumption can be formulated as (2.5).

$$P_{total} \approx P_{dynamic} = \sum_i 0.5 \cdot V_i^2 \cdot C_i \cdot E_i / T = \sum_i \alpha_i \cdot V_i^2 \cdot f \cdot SA_i \quad (2.4)$$

$$E_{total} = \sum_i 0.5 \cdot V_i^2 \cdot C_i \cdot E_i = \sum_i \alpha_i \cdot V_i^2 \cdot SA_i \quad (2.5)$$

where f is the clock frequency, α is a constant that relates to the type and implementation of a resource and SA is the average Switching Activity occurred on the resource per clock cycle.

From (2.4) and (2.5), an obvious observation is that minimization of α , V and SA directly results in the minimization of both power and energy, while minimization of frequency reduces power but not energy. However, in Chapter 4, a technique shows that optimization of frequency can also reduce energy in an indirect way.

The optimization of α in high level synthesis is generally a module selection problem. For low power designs, usually the module that consumes less power has priority to be chosen. However, as low power modules are usually slower, timing constraints may be violated. Therefore design space of module selection must be explored so that power is optimized while timing constraints are still met. In [SHE97], a low-power module selection technique is proposed without consideration of resource constraints. Low power scheduling, allocation and module selection problem is addressed in [GAI98], but it is assumed that the scheduling and allocation tasks are performed before the module selection task. An efficient module selection technique is proposed in [CHA00]. The authors introduce a powerful model called acceptability function, which models design

objectives that are based on tradeoffs among variable design constraints as well as the user's willingness. The key of the technique is to use inclusion scheduling introduced in [CHA98] to estimate design performance and a module utility metric to evaluate how good the modules are.

2.3 Switching Activity Reduction Techniques

From (2.4) and (2.5), it can be seen that power and energy of a module have a linear dependency on the total Switching Activity (SA) of the module during its operation period. Suppose the clock frequency and supply voltage are the same for all the modules, (2.4) and (2.5) can be simplified as (2.6) and (2.7) respectively.

$$P_{total} = V^2 \cdot f \cdot \sum_i \alpha_i \cdot SA_i \quad (2.6)$$

$$E_{total} = V^2 \cdot \sum_i \alpha_i \cdot SA_i \quad (2.7)$$

The task of optimization of power and energy is simplified as the task of optimization of $\sum_i \alpha_i \cdot SA_i$, which is the cost function of many SA-based low power/energy high level synthesis systems.

2.3.1 Switching Activity Estimation

The first problem to be solved in SA-based techniques is how to obtain the total SA of an implementation. Technically, precise SA can be obtained by circuit level simulation using tools like SPICE. However, as circuit level simulation is extremely slow, it cannot be used in high level synthesis, and higher level estimation of SA must be adopted. In [LUN01], the authors propose a hierarchical method for SA estimation of arithmetic modules in DSP applications based on the SA of basic building blocks such as a full adder, a half adder and a one-bit delay. This technique may not be generic enough for high level synthesis because it mainly targets arithmetic modules. In [RAG03], an RTL level SA estimation technique is proposed, using a combination of macro-modeling for datapath blocks and control logic analysis techniques based on partial delay information.

The technique also considers glitch power consumption and gives accurate estimation results. There also exist many analytical techniques based on information-theoretic approaches, which estimate SA for logic blocks based on the entropy of their input and output signals [MAR95, NEM96].

The above SA estimation techniques can be embedded in high level synthesis as power profiler. However, as SA estimation is typically performed frequently in SA-based low power high level synthesis, a fast SA estimation method is desirable.

Many researchers optimize SA based on a two-dimensional matrix, each cell of which represents the average SA between two signals [CHA96, LYU03]. For example, $SA(x, y)$ represents the average SA between data x and data y . Consider the DFG in Figure 2.1.

Figure 2.1 An Illustrational DFG

$SA(e_3, e_5)$ represents the average SA of a signal if the value of the signal changes from e_3 in one clock to e_5 in another clock. Therefore, if a_3 and a_4 are implemented in the same FU, an input of the FU will change from e_3 to e_5 in consecutive clock cycles.

Similarly, suppose a module has only one input, and the values (edges) of the input has a sequence $e_1, e_2, e_3, \dots, e_n$, then the average SA of the module is $SA(e_1, e_2) + SA(e_2, e_3) + \dots + SA(e_{n-1}, e_n)$. Summarily, given the SA matrix, what operations are bound to a resource (binding), and their operation sequence (scheduling), the total power consumption can be quickly calculated by (2.8).

$$P = \sum_i \alpha_i \cdot \sum_j SA(e_{i,j}, e_{i,j+1}) \quad (2.8)$$

How to obtain the SA matrix? In [LYU03, RAG94], DFGs are simulated by arbitrary stimulus vectors to calculate the average SA of each pair of edges. This technique is very flexible, but the estimation accuracy will suffer if the input stimulus vectors cannot simulate real world scenario with sufficient precision. On the other hand, [CHA95b, CHA96] uses world-level joint probability density functions of inputs for calculation. This technique is typically more accurate than the above method, but needs the statistical density functions of the input signals.

2.3.2 Switching Activity Reduction Techniques

In the literature, there exist many techniques for SA-based power optimization for FUs, registers, as well as interconnects. Early researches incorporate SA as cost function in the binding process of a scheduled DFG. In [CHA95b], power optimization in register binding is formulated as a minimum cost clique covering problem of an appropriately defined compatibility graph, which is solved optimally using a max-cost flow algorithm in polynomial time. The authors further handle power optimization in module assignment for functionally pipelined datapath by formulating it as a max-cost multi-commodity flow problem and solve it optimally, as shown in [CHA96]. In [RAG94], a heuristic algorithm is proposed to achieve low power module and register binding by iteratively merging two

nodes of a weighted graph. The two edges representing the merged nodes will be assigned to the same register. A method of controller optimization is also proposed. In [CHO02], the authors present a low power FU binding technique that first determines a feasible binding solution by partially utilizing the computation steps for finding a maximum flow of minimum cost in a network and then refines it iteratively. In [LUO04], the authors propose an algorithm for register binding so that no spurious SA, which is the SA in FUs that happens when the FUs are idle (not used) in some clock cycles but their input and output values change due to register sharing, will occur in FUs.

The above low power binding techniques are not sufficient for SA reduction because scheduling is performed in a power-unconscious way. Due to the high interaction of scheduling and binding, power-unconscious scheduling makes some low power synthesis solution never be found in the binding process. Consider the DFG in Figure 2.1, as $m1$ and $m6$ have the same inputs, binding them in the same resource is probably beneficial to SA reduction. However, a power-unconscious scheduling may schedule them in the same clock cycle, preventing them to be bound to the same resource. Because of this, many works have been focusing on the integration of scheduling and binding for SA reduction.

A complete low power high level synthesis system called SCALP is proposed in [RAG97]. The system keeps a complete synthesis solution (RTL architecture) and iteratively refines it. There are two types of local moves in the iterative refinement process. Move of class A is called Module Selection with Rescheduling. This move transforms the datapath by replacing a FU $fu1$ by another FU $fu2$, in which $fu1$ and $fu2$ are different library template but perform the same functionality (e.g. carry look-ahead adder vs. ripple carry adder). In this move, if $fu2$ is slower than $fu1$, rescheduling may be required. Move of class B is called Hardware Sharing/Splitting with Rescheduling. It replaces two FUs $fu1$ and $fu2$ with a single FU fu . Rescheduling is needed when an operation performed by $fu1$ has an overlapping lifetime with an operation performed by $fu2$. With these two moves, rescheduling and rebinding are performed iteratively to take their interaction into account. As implemented as iterative, this algorithm is easy to

implement, and the results show its effectiveness. On the other hand, power management, such as spurious SA reduction, is not concerned.

In [MUR03], a novel methodology is described based on game theory for power minimization during scheduling and binding. The problems are formulated as auction-based non-cooperative finite games for which solutions are proposed based on the Nash equilibrium. In the scheduling algorithm, a first-price sealed-bid auction approach is used, while in the binding algorithm, each FU in the datapath is modeled as a player bidding for executing an operation with the estimated power consumption as the bid. Furthermore, the integration problem of scheduling and binding is formulated as a single non-cooperative auction game with the FUs in the datapath modeled as players bidding for executing the operation in a particular clock cycle. This technique does not consider power management techniques either. Besides, although the technique is novel and well formulated, experimental results show that improvements are limited.

In [LYU03], a network-flow-based algorithm is evaluated for simultaneous low power scheduling and binding. The technique is a two-step procedure. First a max-flow computation step is performed, which finds a valid flow solution (maximum flow) that conserves the previous solution as much as possible. Then a min-cost computation step is launched to incrementally refine the flow solution obtained in the first step using the concept of finding a negative cost cycle in the residual graph of the flow. As only the negative cost cycle part is refined and resolved, computation time is significantly saved compared to the time needed to do a complete refining on the flow solution. The technique shows significant reduction in CPU time. However, only interconnect power consumption is considered, and the influence of FUs on scheduling and binding is not considered.

There are many other published techniques on integrated scheduling and binding for SA-based power optimization. The authors in [DAS98] use a simulated-annealing-based algorithm to perform simultaneous scheduling and binding to minimize bus transitions. In [SHI00b], first an ILP-based model is performed for latency constrained scheduling that

minimizes the peak power, and then a LP-based model is used for resource binding that minimizes the amount of switching at the input of the FUs. An efficient heuristic algorithm based on the concept of Traveling Salesman's Problem [PAP82] is proposed in [MAS03] for the low power synthesis specifically targeting sum-of-product computation structures.

The SA-based power optimization has gained much popularity in the area of low power high level synthesis. Early works focuses on power efficient binding to reduce SA. To extend the search space, power-conscious scheduling has been incorporated in one way or another, e.g. scheduling and binding are performed in a combined manner to reduce SA and thus power consumption. In Chapter 3, a novel technique called Look-Ahead Synthesis (LAS) is proposed to address the SA reduction problem during simultaneous scheduling and binding.

2.4 Voltage Scaling and Frequency Scaling

An important observation from (2.4) and (2.5) is that power and energy consumption have a quadratic dependency on supply voltage. Therefore reducing supply voltage is the most effective way to reduce power and energy. Meanwhile, reducing frequency can reduce power, but not energy. As a result, recent years have witnessed many research works in the area of power/energy optimization based on voltage and frequency scaling.

However, according to (2.9), which models the delay of a CMOS circuit, reducing V_{dd} increases the delay of the circuit, hence, increasing the probabilities that the timing requirement is violated. Similarly, decreasing f will extend the clock period, which directly imposes a threat to break the performance constraint. Conclusively, power-performance tradeoff and performance constraints must be accounted when supply voltage and frequency optimization techniques are used to reduce power and energy.

$$t_d = k \cdot \frac{V_{dd}}{(V_{dd} - V_T)^\gamma} \quad (2.9)$$

A tool called Power Profiler in [MAR95a] clearly shows that reducing the supply voltage of a library component substantially reduces the power but increases delay. The tool can also perform power optimization using either heuristic algorithms or genetic algorithms. In [WIL00], the tradeoff of power and performance due to scaling of supply voltage is explored in their high level synthesis system.

2.4.1 Multi-Voltage High Level Synthesis

The observation that reducing supply voltage will increase circuit delay and risk violations in performance constraints stimulates the advent of multi-voltage supply techniques. In these techniques, resources can operate at different supply voltages. The rule-of-thumb is that, operations in the critical path operate on resources with higher voltage to maintain performance, and operations not in the critical path operate on resources with lower voltage to reduce power and energy.

In a multi-voltage high level synthesis system, the datapath consists of multiple supply voltage levels. This multi-voltage datapath imposes several problems when practical designs are considered.

2.4.1.1 Layout Issues

The multi-voltage scheme imposes a big challenge during layout. It is supposed to be necessary that multi-voltage datapath may be partitioned into several operating regions, each of which operates at a different voltage. Isolation between different voltage regions will be needed, probably by increased use of substrate contacts and slightly increased spacing between wells. The grouping of modules that operate at the same voltage into a common region can be incorporated into the multi-voltage high level synthesis algorithm, which is covered in Chapter 5. Detailed discussion can be found in [JOH97].

A real-world dual-supply-voltage (VDDH and VDDL) media processor has been devised in [USA98], in which two layout schemes, area-by-area architecture and row-by-row architecture are proposed and compared. In the area-by-area architecture, VDDL cells are grouped and placed in one area, while VDDH cells are placed in the other area. This

architecture is similar to what is assumed in the proposed technique in Chapter 5, with the advantage of allowing a designer to generate layout with existing P&R tools. On the other hand, in the row-by-row architecture, VDDH cells and VDDL cells are placed in different rows, with no mixture of VDDH cells and VDDL cells in the same row. It is shown that the row-by-row architecture is preferred when minimal connecting wire length is desired. More details can be found in [USA98].

2.4.1.2 Supply Voltage Generator

In a multi-voltage datapath, there are more than one voltage supplies. If all the supplies are generated off-chip, additional power and ground pins are indispensable. As the number of pins is usually limited for integrated circuits, on-chip power supplies can be used. In this case, several voltage supplies are generated by an on-chip supply voltage generator from the primary voltage supply pin. This supply voltage generator occupies area and consumes power, and thus might be a concern in practical designs.

2.4.1.3 Level Converters

Level converter plays a very important role in multi-voltage designs. It is the interface between two resources with different supply voltages that are connected together. It is reported [JOH97, USA98] that when a resource of higher voltage drives a resource of lower voltage, it is not necessary to convert the voltage level. However, when a resource of lower voltage drives a resource of higher voltage, a level converter must be placed between the two resources to convert the lower level signal into a higher level signal. Otherwise, the PMOS device of the driven resource will be in a weak ON condition, conducting static current from the supply to ground. In [CHA94], a Differential Cascade Voltage Switch (DCVS) is proposed as a level converter, as illustrated in Figure 2.2 (reported in [JOH97]).

Figure 2.2 DCVS Level Converter

Note that level converters also occupy area and consume considerable power. In some situations, the level converters consume more power than what is saved by voltage scaling. Under these circumstances, a partially voltage-scaled datapath would be preferable, in which the supply voltage of some resources will not be scaled down, even if they can be, to avoid unnecessary level converters.

2.4.2 Multi-Voltage Synthesis Techniques

As reducing supply voltage can achieve high power reduction ratio, many papers published addressed supply voltage reduction and performance compensation using multi-voltage techniques. In multi-voltage synthesis problems, it is usually assumed that the average SA is the same (0.5) for all fanouts in the circuit. Therefore the total power of a circuit can be estimated by (2.10).

$$P_{total} = 0.5 \cdot f \cdot \sum_{i=1}^N \alpha_i \cdot V_i^2 \quad (2.10)$$

Where N is the number of operations (instead of number of resources), α is a technology dependent parameter (capacitance) and V_i is the supply voltage of the resource on which the operation i is performed. As a result, binding does not have a direct impact on the power consumption in multi-voltage synthesis under the assumption that all SA is the same, although it is indispensable to maintain the resource constraint. Therefore multi-voltage synthesis is frequently referred as multi-voltage scheduling problem.

In [JOH97], a system called MOVER is proposed to minimize datapath energy dissipation through the use of multiple supply voltages. The assumed datapath model is shown in Figure 2.3. All operator outputs are registered and each operator output feeds only one register, which operates at the same voltage as the operator supplying its input. Level converters are inserted at the operator inputs when needed. The core of MOVER is an ILP program that repeatedly evaluates possible supply voltages, partitions operations between different supply voltages, and generates a schedule that minimizes resource usage. The ILP variables and equalities are well formulated, and both resource constraints and latency constraints are considered. However, ILP algorithms suffer from exponential complexity problems.

Figure 2.3 A Multi-Voltage Datapath Model

In [CHA97], it is demonstrated that multi-voltage scheduling problem was NP-hard and a dynamic programming approach was proposed. The algorithm has pseudo-polynomial

complexity and produces optimal results for trees, but is suboptimal for general directed acyclic graphs. Further, the dynamic programming technique is extended to handle functionally pipelined designs with the help of a novel approach called revolving schedule. Generally, this technique has low complexity and generates results with good quality, but resource constraints are not considered.

In [LIN97], an ILP-based algorithm and a heuristic algorithm are proposed for variable voltage scheduling. ILP formulations for resource-constrained, time-constrained and resource-and-time-constrained scheduling are described. In order to reduce computational complexity, the list-scheduling based heuristic algorithm uses a priority function to perform variable voltage scheduling, resulting in much faster run-time and comparable results with regard to those obtained by ILP when 3.3V and 5.0V resources are considered. However, power consumed by level converters is not accounted.

In [SHI00a], resource-constrained and latency-constrained algorithms are proposed. The resource-constrained algorithm tries to achieve a balance between reducing the latency and maximally utilizing the resources operating at reduced voltages. The algorithm is based on list scheduling and has polynomial complexity. The latency-constrained algorithm (also has polynomial complexity) tries to reduce resource voltages as much as possible without violating timing constraint, under the assumption that there is no restriction on the number of available resources.

Resource and latency constrained algorithms are proposed in [MAN02] for power/energy minimization with resources operating at 5V, 3.3V, 2.4V and 1.5V. The algorithms consist of two steps. The first step is a minimum-time resource-constrained algorithm, which finds an initial schedule with minimum execution time. In the second step, available time slacks between nodes are distributed (so that assigned voltage of some nodes can be reduced) using the Lagrange Multiplier method in an iterative fashion. A low complexity $O(n^2)$ algorithm and a high complexity $O(n^2 \log(L))$ algorithm are derived, where n is the number of nodes and L is the latency.

In [MOH03, MOH04a], mobility-based, calculated by As Soon As Possible (ASAP) and As Late As Possible (ALAP) scheduling algorithms, heuristic algorithms are proposed for the scheduling problem of multi-voltage datapath for simultaneous energy and transient power reduction. In the work, a new metric called Cycle Power Function (CPF) is defined to capture the transient power characteristics as an equally weighted sum of the normalized mean cycle power and the normalized mean cycle differential power. The authors further develop an ILP-based scheduler in [MOH04b, MOH05] to address similar problems.

2.4.3 Frequency Scaling Techniques

Power can also be reduced by frequency scaling as power consumption has a linear relation on the frequency. However, simply scaling down frequency for power optimization directly reduces the performance of the circuit. In the literature, there exists some works on frequency scaling for low power under performance constraints. However, it has not been widely explored.

In [RAG97], clock selection is incorporated in the synthesis process for power optimization. The influence of slack (the interval between the time at which a FU finishes execution and the clock edge at which its output is actually used) is discussed. It is argued that slacks can cause two undesirable effects in the context of power dissipation minimization. First, the presence of slacks acts to inhibit greater supply voltage scaling. Second, slacks can result in a datapath with higher switched capacitance. It is also pointed out in the paper that it is a misleading argument that having a small clock period would minimize the slack and hence is advantageous.

An interesting multiple clocking scheme for RTL datapath synthesis is proposed in [PAP99]. The technique partitions the datapath into multiple disjoint modules, and assigns frequency f/n to each module, where f is the global clock frequency for the entire circuit and n is the number of partitions. In this way, inactive partitions are literally turned off during their off duty cycle to reduce power dissipation. Efficient as it is, this technique is constrained by the structure of the datapath, because the number of multiple

disjoint modules into which the datapath can be partitioned is determined by the structure of the datapath.

[MOH03, MOH04a, MOH04b, MOH05] use a technique called Dynamic Frequency Clocking (DFC) to achieve optimization on variable power and energy parameters. In DFC, all FUs are clocked by a single clock line which switches at run time, so that the clock frequency is varied on-the-fly based on the active FUs in that cycle. DFC is demonstrated in Figure 2.4, as extracted from [MOH05]. The datapath of DFC has a Dynamic Clocking Unit (DCU), which generates the required clock frequency usually using clock dividers. It is noted that the purpose of DFC is not to directly reduce power, but to compensate the extended delay caused by reducing voltage supply of some resources. Therefore, DFC indirectly reduces energy as well as power dissipation.

Voltage scaling and frequency scaling have been studied for low power/energy synthesis. The multi-voltage synthesis, in which the datapath consists of resources operating at different levels of supply voltages, has drawn considerable attention. Some circuit level issues (layout, level converters, etc.) must be considered in a multi-voltage datapath. Frequency scaling techniques have also been proposed to directly or indirectly reduce power and/or energy. In Chapter 4, a Multi-Voltage Multi-Frequency Synthesis (MuVoF) technique is proposed for functionally pipelined datapath for power/energy optimization.

(a) Conventional single frequency clocking scheme.

(b) DFC scheme in which clock width can be different in every cycle.

(c) Scheme for dynamic frequency generation using DCU.

Figure 2.4 Dynamic Frequency Clocking

2.5 Integrating Low Power High Level Synthesis with Physical Synthesis

It is well-known that to obtain an accurate estimation of cost functions (area, delay, power etc.) in high level synthesis is extremely difficult, because physical level information is not available at the high level. On the contrary, in modern process technology, interconnects, whose information can only be obtained at the physical level design, have a major impact on various circuit parameters, especially performance and power dissipation due to the ever-decreasing device size. To close the gap, predicting and extracting physical level information during high level synthesis has attracted a lot of attention [MCF90, KNA92, XU97, DOU00].

For power optimization, it is reported that interconnect (wires, buffers, etc.) power consumption averagely can occupy over 20% [ZHO05] of the total power. As detailed information of interconnects (such as the total length of connecting wires) is only available at the physical level, power optimization in high level without physical

information is not sufficient for power minimization. Many papers have been published on the integration of physical information into high level synthesis for power optimization, in which floorplan-driven techniques are more popular.

2.5.1 Introduction to Floorplanning

Floorplanning is the first stage of physical synthesis, followed by placement, routing and layout. It is also a major research field in the EDA area [GUO01, TAN06]. In floorplanning, the relative geometrical positions of all resources (often shaped as rectangles) are determined.

Floorplan can be classified into two categories, slicing and non-slicing. Consider the two floorplans in Figure 2.5. The two floorplans both consist of five rectangles representing five resources. The slicing structure is a rectangle dissection that can be obtained by recursively cutting a rectangle vertically or horizontally, while the non-slicing structure cannot be. It is reported that slicing structures have several advantages over general rectangle dissections in floorplan design [OTT82] and are computationally easier to handle [STO83]. More information on floorplanning can be found in [GER99].

(a) A non-slicing floorplan

(b) A slicing floorplan

Figure 2.5 Two Floorplan Styles

2.5.2 Floorplan-Driven Low Power High Level Synthesis

As floorplanning is the first stage of physical synthesis, it is relatively easier and less computationally intensive to incorporate it into high level synthesis. The major information of a floorplan that can be utilized in high level synthesis is the wire information. Although the exact positions (placement) of resources of their connections (routing) are not available in the floorplan, wire length can be estimated much more accurately than at the behavioral level. A simple model to calculate the wire length is given in Figure 2.6.

Figure 2.6 A Simple Wire Length Model in Floorplanning

In this model, the length of the connecting wires of two resources $r1$ and $r2$ is estimated by the distance of their central points $p1$ and $p2$. Suppose there are N connections between the two resources, and every connection is implemented with M wires, then the total wire length between the two resources is:

$$L(r1, r2) = N \bullet M \bullet \text{distance}(p1, p2) \quad (2.11)$$

The total wire length of a circuit can be derived as:

$$\sum L_i = \sum_{i=1}^n \sum_{j=1}^n N_{i,j} \bullet M_{i,j} \bullet \text{distance}(p_i, p_j) \quad (2.12)$$

When power is concerned, like a normal resource, power consumed by a wire can also be modeled as:

$$P = \alpha \cdot V^2 \cdot f \cdot C \quad (2.13)$$

In which α is the average SA occurred on the wire, V the voltage of the signal that flows through the wire, f is the operating frequency, and C is the total capacitance of the wire. Typically C is modeled as $C = c \cdot L$, in which c is the average capacitance per length, and L is total wire length. Summarily, in this model, the power consumed by a wire is proportional to the length of the wire. Suppose all the connecting wires have the same α , V , f and c , then the total power consumed by all the connecting wires would be:

$$P = \alpha \cdot c \cdot V^2 \cdot f \cdot L \quad (2.14)$$

Under the assumption that SA, frequency, voltage and capacitance are kept constant for all wires, the total wire power consumption of a circuit can be modeled as:

$$P_{\text{wire}} = \alpha \cdot c \cdot f \cdot V^2 \cdot \sum L_i \quad (2.15)$$

From (2.15), it is concluded that in a single-voltage design, the optimization of wire power consumption is the same task as the optimization of total wire length.

To minimize the power consumed by connecting wires, many high level synthesis techniques are proposed, that incorporate the floorplan information into the synthesis tasks.

In [PRA99], a technique is proposed to do simultaneous scheduling, binding and floorplanning based on the simulated annealing algorithm. The combined scheduling and binding problem is formulated as a placement problem in a two-dimensional table. When a local move is performed, fast evaluation is carried out by a constructive power driven

floorplanning algorithm that considers both SA on the wires and the wire length. This algorithm is effective in reducing interconnect power. However, as power consumed by datapath is not accounted, the tradeoff between interconnect power and datapath power is not explored.

In [STA03], the authors perform floorplan-driven low power high level synthesis using an algorithm that consists of two levels of simulated annealing algorithm. The inner loop uses a famous floorplanning algorithm proposed in [WON86] to find a power optimized floorplan regarding a cost function that involves wire power consumption. The outer loop iteratively refines the binding. In the process, the binding is refined in each outer loop, and a corresponding power efficient floorplan is found so that wire power consumption is optimized. This technique is similar to that proposed in Chapter 5, except that it does not integrate multiple voltage techniques.

An innovative technique is developed in [DAV05]. This technique performs simultaneous power-driven binding floorplanning based on a probabilistic approach [DAV03] to improve the estimation accuracy of the net parameters, such as delay and power. In a probabilistic approach, the distribution of a net-length rather than the fixed estimate is modeled so that a range of values for the net-length together with their associated probabilities are considered. Using such probabilistic models, the optimization targets to maximize the likelihood of meeting the design constraints while minimizing the overall power. Moreover, the probabilistic models achieve more accurate estimation of power during optimization. The drawback is that the influence of scheduling is not considered.

In Chapter 5, a Floorplan-driven Multi-Voltage Synthesis (FloM) technique is proposed. The approach uses the simple wire length model illustrated in Figure 2.6 to achieve fast estimation. The technique addresses the power optimization of wires for multi-voltage datapath, which is a new problem in the literature.

2.6 High Level Transformation Techniques in High Level Synthesis

Although scheduling and binding can be performed directly on the input behavioral description, high level transformations are usually performed beforehand to potentially reduce the delay, area and the power consumption of the resulting implementation. Actually, the motive of high level transformation is to change the DFG and thus change the structural implementation of the input behavioral description. As a result, high level transformations are usually applied before normal high level synthesis tasks such as scheduling and assignment.

In [DEM94], many traditional software transformation techniques that have been applied to high level synthesis are introduced, including Constant Propagation, Variable Propagation, Common Subexpression Elimination (CSE), Operation Strength Reduction, etc. Among them, Tree Height Reduction (THR) gains much attention. It reduces the height of the arithmetic expression tree. Consider Figure 2.7, in which THR is performed on the expression $R = a + bc + d$. Traditional THR performs commutativity, associativity as well as distributivity. It can achieve a speed-up of $O(n/\log n)$ for an expression with n operations in the best case. THR has proven useful in high level synthesis to reduce the latency [NIC91, KOL94, WAN96]. Typically THR will reduce the Critical Path Delay (CPD) at the expense of more resources.

Some specific high level transformations aim to increase the throughput (input rate) of a sequential circuit with delay nodes (registers). Note that when throughput is concerned, the CPD is defined as the maximum delay between any pair of IOs or delay nodes. These techniques are especially important for DSP and multimedia applications, in which data are fed iteration by iteration. Classic techniques are retiming and pipelining [GER99, BRI01]. Retiming is the process of moving delay nodes between the input of a function element and its output. Pipelining inserts delay nodes in certain points, known as cutset points, so that the input can fire (start operation) before the output for its predecessor has been computed. These techniques increase throughput, but do not affect the overall delay.

(a) Before THR (b) After THR

Figure 2.7 An Example of THR

In modern computational intensive applications, such as DSP and multimedia applications, loop transformation techniques, which are designed specifically for the optimization of loop structures, play a very important role. These techniques do some manipulations on the loop structures and may significantly influence the area, delay and throughput of the resulting implementation [WEI99, WEI01a, WEI01b, KIT03]. These techniques include Loop Rolling/Unrolling, Loop Merging, Loop Tiling and Loop Distribution, etc.

2.6.1 Low Power High Level Transformation Techniques

In fact, most high level transformation technique can be reconsidered to optimize power consumption. In [LUN01], it is showed that even a simple commutativity transformation can influence the power consumption of an adder and a multiplier. In [MAS03], the best datapath of a sum-of-product computation is found based on the Traveling Salesman's Problem. The idea actually is to find the best transformation that rearranges the sequence of the partial products in the sum-of-product expression.

According to [CHA95c], there are two distinct approaches to optimize power using transformations. The first approach enables the reduction of supply voltage using speedup transformations, such as retiming, pipelining, algebraic manipulations, loop transformations, etc. The main idea is to reduce the number of control steps, so that slower control clock cycles can be used for a fixed throughput, allowing for a reduction in supply voltage. The second approach is to reduce the switched capacitance on the resources. For example, CSE techniques reduce number of operations by finding subexpressions that appear in many expressions. As the number of operations become diminished, the average number of operations executed on a resource is decreased, therefore SA and hence power dissipation falls. Several such techniques are proposed in [CHA95c], including Operation Reduction (reduce the number of operations), Operation Substitution (use power-efficient operations, e.g, convert multiplications with constants into shift-add operations), Resource Utilization (potentially reduce the number of resources by transforming the DFG so that operations are more uniformly distributed) and Wordlength Reduction (minimize the number of bits), etc.

However, it is pointed out in [CHA95c, MAC98] that transformations may have adverse effects. Consider Figure 2.8. This distributivity reduces the number of operations, but increases the delay in the critical path. Therefore, when performing high level transformation, attention must be paid to whether constraints are violated.

Figure 2.8 An Adverse Transformation

In [PEY01a, PEY01b, PEY03a, PEY03b], a novel high level transformation technique is proposed. The technique uses the concept of modern Symbolic Computer Algebra (SCA)

for optimization of CPD and area. In Chapter 6, several techniques are proposed to extend the work and target not only on delay and area, but also on power consumption.

2.6.2 Introduction to Symbolic Computer Algebra

Different from the traditional Numeric Computer Algebra, which manipulates fixed-length integers and fixed-precision floating-points numbers, modern Symbolic Computer Algebra (SCA) deals with undefined symbols, which have arbitrary length and precision. It means that the only information about the input variables is that they are variables. Neither their word length nor their precision is known, i.e. they may have infinite precision.

The major purpose of a Computer Algebra System (CAS) is to manipulate a formula symbolically using computer. For example, expanding, factorizing, root finding, or simplifying an algebraic polynomial are some of the common uses of CAS. Nowadays there are many CASs in the market, in which Maple [MAP] is one of the most successful tools.

Like most SCA Systems, in Maple, the users input a manipulation, Maple calculates it and outputs the result. For example:

```
> f:=a+b+c; g:=d+e;
> result:=f*g;
result := (a + b + c) (d + e)
```

There are two crucial Maple functions: “factor” and “simplify”. Here are some examples on how they are used.

```
> h:=expand(result);
h := a d + a e + b d + b e + c d + c e
> r:=factor(h);
r := (a + b + c) (d + e)
```



```
> simplify(h,{a+b+c=x0});
(d + e) x0
```

The function “factor” tries to factorize the input polynomial. If it fails, it outputs a functionally equivalent polynomial. The function “simplify” will decompose the input polynomial using equations called *side relations* in Maple.

The Maple function “factor” refers to the problem of factoring a polynomial in a single or several variables and over rational numbers, finite fields or complex numbers, which has witnessed its success over the past 40 years.

Earlier versions of polynomial factorization can only deal with univariate polynomial. The first algorithm appears in the early 1960s and is found quite inefficient. After that, other mathematicians applied many new ideas like randomization, primality testing, and the powerful Hensel lifting lemma is introduced to computer algebra. Algorithm with polynomial-time complexity for rational coefficients is established in the early 1980s, well known as lattice basis reduction algorithm.

The multivariate polynomial factorization first appeared in the DeMillo and Lipton/Schwartz/Zippel lemma [ZIP81]. Today there are many different techniques on the factorization of multivariate polynomials. In [ROB99], X. F. Roblot factors polynomials modulo a prime ideal over number fields. In [GAO00], Gao uses partial differential equations to factor multivariate polynomials. A more detailed history of polynomial factorization can be found in [KAL03].

The Maple function “simplify” refers to polynomial decomposition, also called polynomial simplification, which means to simplify a polynomial using a set of equations [PEY03a, DAV88].

Generally, when decomposing a polynomial p with respect to a set of polynomials Q (called *side relations* in Maple), p is not reduced with Q directly, but using the Gröbner

basis of $\langle Q \rangle$ instead ($\langle Q \rangle$ is an ideal generated by Q). Gröbner basis is a reduced basis and a canonical representation for a multivariate polynomial ideal. It is the foundation of polynomial decomposition. Detailed algorithms of reduction and the generation of Gröbner basis can be found in [PEY03a, DAV88].

2.6.3 Using Symbolic Computer Algebra in High Level Transformation Techniques

When using SCA in high level synthesis, first the input DFG must be transformed into a polynomial representation. Although there may be both word-level and bit-level implementations in a datapath and they may be much complicated than polynomial, the polynomial representation formalism is an effective method.

As proved in [SMI98, SMI99, SMI01], and summarized in [PEY03a], polynomial representation is effective to describe the datapath of any basic block, which is usually extracted from the DFG of an algorithmic level DSP model. The polynomial is formulated as follow according to the implementation of the basic block:

- (1) The basic block implements a polynomial function: the polynomial representation is extracted directly.
- (2) The basic block implements a series of bit manipulations or Boolean functions: the equivalent polynomial representation can be calculated by interpolation-based algorithms in [SMI98].
- (3) The basic block implements a transcendental function: an approximate polynomial representation can be obtained by any series expansion, such as Taylor, Chebyshev series expansion. In this case, the polynomial should be verified by simulation so as to satisfy the precision requirements.
- (4) The basic block implements a synchronous acyclic circuit: the equivalent polynomial is constructed by determining the combinational equivalents.

In [PEY01a, PEY01b, PEY03a, PEY03b], a SCA-based technique called Minimal Component Decomposition is proposed. This technique automates the process of using complex library components, such as sine and cosine, to implement a DFG that is

formulated as a polynomial. Embedded in the algorithm there are many supporting transformation techniques, such as Factor, Expand, Horner Form, Substitution and Elimination, etc. Experimental results show that both CPD and area can be significantly reduced.

In Chapter 6, two techniques, Subexpression Factorization and Complex Components Mapping, are proposed. Based on SCA, the techniques use polynomial factorization and polynomial decomposition, coupling with many other supporting transformation techniques, achieve considerate optimization in area, CPD as well as power consumption.

2.7 Summary

Modern digital IC design largely depends on EDA tools at different abstraction levels. When physical synthesis and logic synthesis tools are indispensable in today's IC design, high level synthesis is still mainly in the research field, yet expecting to play an important role in the next generation of IC design.

Traditional high level synthesis mainly targets the optimization of area, delay and throughput. Due to ever-growing demands in portable devices and mobile applications, power and energy optimization becomes more and more important. Power optimization in high level synthesis also attracts wide attention.

In this chapter, several areas of research on power modeling are introduced. A typical power/energy model in high level synthesis, formulated as (2.4) and (2.5), is explained. From this model, it is observed that power/energy optimization can be achieved in several aspects, specifically, switched capacitance, supply voltage and frequency.

Reduction of switched capacitance can be achieved by using power-efficient resources or by reducing total switching activity occurred on the circuit. Power-efficient resources are typically larger and slower than normal resources; therefore care must be taken so that area and performance constraints are not violated. In the literature, it is shown that SA reduction techniques have been widely explored. Early works focus on taking SA

reduction into account during the resource binding process. More recently, simultaneous scheduling and binding are considered to explore larger design space. SA optimization can be performed on FUs, buses as well as registers. In the literature, techniques based on network flow, ILP, iterative algorithm and heuristics have been proposed to address various SA reduction problems.

Power can be efficiently decreased by using lower supply voltage, because it has quadratic dependency on the supply voltage. However, as lower supply voltage results in longer delay, performance tends to degrade. To address this problem, multi-voltage datapath has been proposed. In this type of datapath, resources in the critical path are assigned with higher voltage to meet performance constraints, while those not in the critical path is assigned with lower voltage to reduce power. This is often referred as multi-voltage scheduling problem. Techniques based on ILP, dynamic programming, and heuristics have been proposed to solve this problem. Multi-voltage-based low power techniques have gained much popularity recently because of their effectiveness. However, researchers are aware that physical implementation problems on multi-voltage datapath must be addressed. Real-world multi-voltage ICs [USA98] have been reported to prove that multi-voltage is physically implementable and provides very promising results.

Power reduction can be achieved by frequency scaling. However, scaling down frequency directly reduces the performance of the circuit. It is shown in [PAP99] that a circuit can be partitioned into several parts and the frequency of each part can be scaled down by a factor of n , which is the number of partitions. This technique is a direct way of frequency-scaling-based power optimization. There also exist other techniques which indirectly use frequency scaling for power optimization. In these techniques, frequency is scaled up rather than scaled down. The purpose is to compensate performance degradation caused by other power optimization techniques, such as voltage-scaling-based techniques. An example is the dynamic frequency scaling used in [MOH05].

Floorplan-driven power optimization is also introduced here. In this generation of semiconductor technology, it is very important to integrate physical level information

into high level synthesis, because interconnects, such as wires, become the dominating factor in many design parameters, such as area, delay and power consumption. Compared to other physical synthesis tasks, such as routing and layout, floorplan is much less computational intensive and it also provides physical level information in decent accuracy. Therefore the integration of floorplanning into high level synthesis is the mainstream research in the field of integration of physical synthesis and high level synthesis. When power is concerned, in the literature, there are extensive works on both wire power estimation and wire power optimization, among which iterative algorithms are widely used.

High level transformation techniques have existed in the advent of high level synthesis. They are also being studied for power optimization. Many techniques aim to decrease the number of operations, which in turn can potentially reduce the switched capacitance and thus the power dissipation. Other techniques try to reduce the number of control steps to provide more space for other techniques to reduce the power.

Summarily, many topics related to low power/energy high level synthesis are introduced in this chapter. Extensive power modeling, estimation and optimization techniques are discussed. It is shown that power optimization can be achieved by the optimization of variable parameters. Special attention is given to SA reduction techniques, multi-voltage techniques, frequency scaling techniques, floorplan-driven techniques as well as high level transformation techniques. In the following chapters, several low power/energy optimization techniques targeting various problems are proposed. The proposed techniques achieve comparable good results to those reported in the literature.

Chapter 3 Look-Ahead Synthesis for Switching Activity Reduction

3.1 Introduction

As discussed in Section 2.3, research work done has shown that power consumption in digital integrated circuits can be effectively reduced by reducing the switching activity (SA) occurred on the functional modules, registers and interconnects. High level synthesis of digital integrated circuits for low power often optimizes the SA during the two main synthesis processes, operation scheduling and module binding, which are usually performed one control step at a time as two separated stages. As the two processes are highly interdependent, optimization of SA in a step by step manner and in two separated stages usually leads to sub-optimal solutions.

In this chapter, a technique named Look-Ahead Synthesis (LAS) is proposed to address the operation scheduling, module binding and register binding for power optimization under resource and latency constraints. With LAS, operation scheduling and module binding are performed simultaneously in an integrated manner. The proposed scheduling algorithm is based on a list scheduling approach, so the circuit latency overhead is negligible. Unlike a typical list scheduling algorithm, which schedules operations based on their criticalness, the proposed LAS schedules operations such that the SA within a certain number of control steps is minimized. In other words, scheduling is driven by power-conscious binding. To minimize local effects in each control step, different scheduling and binding patterns with different switching activities are found at the same time using a weighted bipartite matching technique. Each of these patterns is in turn used to schedule and bind more operations for the next one or more control steps. The best scheduling and binding pattern in terms of SA is then used to backtrack the synthesis. In this way, LAS performs concurrent scheduling and binding in a branch-and-bound approach with look-ahead evaluation of SA in each control step and backtracking of one or more steps. As a result, LAS is able to search a large design space for optimal solution

and at the same time, effectively reducing the possibility of falling into local minima. After scheduling and module binding are completed, power-conscious register binding is performed. The register binding process not only optimizes SA on registers, but also eliminates spurious SA on modules using a modified method similar to [LAK99a]. Experimental results show that LAS can reduce SA on modules and registers significantly, with only slight area overhead due to extra registers and negligible latency overhead.

This rest of the chapter is organized as follows. Section 3.2 introduces low-power scheduling and binding, and demonstrates the influence of scheduling and binding on SA. Section 3.3 describes the proposed LAS algorithms which perform simultaneous scheduling and binding. An illustrating example is given in Section 3.4, followed by the experimental results and comparisons in Section 3.5. A summary is given in Section 3.6.

3.2 Low Power Scheduling and Binding via Switching Activity Reduction

It is shown in Chapter 2 that the dynamic power dissipated by a resource (modules, registers, or interconnects) can be modeled as:

$$P_{total} = \sum_i \alpha_i \cdot V_i^2 \cdot f \cdot SA_i \quad (3.1)$$

In high level synthesis, a popular approach to reduce the power is to reduce the SA, because V and f are usually constrained by circuit performance specifications. When resource type, word length and technology are determined, SA depends not only on the circuit structure, but also on the input signals. A table representing the SA (x, y) between a variable x and a variable y is called an SA Table. There are two ways to construct the SA table: (1) when the statistical information of the input variables are available, statistical signal processing can be used to obtain the joint probability density function of each variable pair, from which the average signal transition between each variable pair can be

calculated. (2) typical input patterns or random inputs can be used to simulate the average signal transition.

3.2.1 Switching Activity Profiling

Since SA Table is used to represent the total SA, it can represent the dynamic power consumption during the proposed LAS technique assuming V and f are fixed. In order to profile the SA of a given DFG, a high-level simulation program named the SA Profiler has been developed for the LAS. In the SA Profiler, all edges of a DFG are modeled as variables.

The SA Profiler consists of two levels of loops (N iterations of outer loop and M iterations of inner loop, totally $N*M$ iterations). In every iteration of the outer loop, all inputs are fed by input vectors randomly generated. This is to model the reset effect in a circuit.

The inner loop models the normal operation of a circuit, and each iteration represents a set of input vectors. Two scenarios are considered to generate the input vectors. In the first scenario, certain inputs are driven by delayed signals of certain outputs. Then the delayed values of the outputs are used as the input vectors of the specific inputs to maintain the circuit functionality. This information can be extracted from the DFG, as demonstrated in Section 3.5.1. In the second scenario, if certain inputs are from outside, it is assumed that the values of an input has Gaussian distribution. This means that the input value in the current iteration has higher probability to be close to the value of the last iteration.

It is assumed that all the variables have a Q-15 format [BEL00], as used in typical fixed-point digital signal processing applications. In each iteration of simulation, the input vectors are propagated through the DFG to obtain the value of every edge. The Hamming distance between every pair of variables is accounted for the SA table construction. For example, suppose the value of a variable x is “0000111010000110” and the value of

another variable y is “0001111000110000”, then the Hamming distance between x and y is 6.

For a pair of variable x and y , the average Hamming distance for M inner loop iterations is given by:

$$SA_{inner}(x, y) = \frac{1}{M} \sum_{i=1}^M \text{hamming_distance}(x_i, y_i) \quad (3.2)$$

The average SA is the average over N outer loop iterations, therefore:

$$SA(x, y) = \frac{1}{MN} \sum_{i=1}^N \sum_{j=1}^{M-1} \text{hamming_distance}(x_{i,j}, y_{i,j}) \quad (3.3)$$

Figure 3.1 DFG of a Differential Equation Solver

Consider the DFG of a Differential Equation Solver, as illustrated in Figure 3.1. In the SA profiling, the DFG is reset N times. After a reset, totally M input vectors are propagated into the DFG. Therefore for every variable v , values $v_{0,0}$ to $v_{N,M}$ are recorded. Using Equation (3.3), Table 3.1 is constructed. Note that the signal value represented by an edge is not a single bit signal but rather a vector of 16 bits. Therefore the switching activities could be larger than 1.

Although the precision of the DFG simulation is limited due to lack of scheduling and binding information, many researchers [LYU03, RAG94] adopted this approach for its efficiency and flexibility. There exist a lot of work on how to improve SA estimation quality at high level [LUN01, MAR95, NEM96, RAG03], however, they are typically targeting specific applications and of high complexity, thus not suitable for our application.

Table 3.1 Switching Activity Table of the Differential Equation Solver

	i1	i2	i3	i4	c1	c2	e1	e2	e3	e4	e5	e6	e7	o3
i1	0.0	8.02	7.97	8.0	7.93	8.0	7.31	7.98	8.13	8.37	7.0	7.47	7.31	7.98
i2	8.02	0.0	7.74	8.0	7.81	7.81	8.02	7.7	7.96	7.95	7.99	7.93	8.02	7.16
i3	7.97	7.74	0.0	7.3	7.46	8.53	7.85	7.13	7.84	7.29	8.07	7.39	7.85	2.89
i4	8.0	8.0	7.3	0.0	7.71	8.22	7.89	7.77	7.9	6.76	8.08	6.43	7.89	7.31
c1	7.93	7.81	7.46	7.71	0.0	8.0	7.85	4.87	7.83	7.63	8.04	7.84	7.85	7.45
c2	8.0	7.81	8.53	8.22	8.0	0.0	8.13	8.17	8.16	8.25	7.95	8.14	8.13	8.53
e1	7.31	8.02	7.85	7.89	7.85	8.13	0.0	7.83	7.63	7.93	7.81	7.92	0.0	7.82
e2	7.98	7.7	7.13	7.77	4.87	8.17	7.83	0.0	7.83	7.76	8.06	7.86	7.83	7.23
e3	8.13	7.96	7.84	7.9	7.83	8.16	7.63	7.83	0.0	7.9	7.66	8.06	7.63	7.83
e4	8.37	7.95	7.29	6.76	7.63	8.25	7.93	7.76	7.9	0.0	8.31	7.05	7.93	7.3
e5	7.0	7.99	8.07	8.08	8.04	7.95	7.81	8.06	7.66	8.31	0.0	8.05	7.81	8.07
e6	7.47	7.93	7.39	6.43	7.84	8.14	7.92	7.86	8.06	7.05	8.05	0.0	7.92	7.35
e7	7.31	8.02	7.85	7.89	7.85	8.13	0.0	7.83	7.63	7.93	7.81	7.92	0.0	7.82
o3	7.98	7.16	2.89	7.31	7.45	8.53	7.82	7.23	7.83	7.3	8.07	7.35	7.82	0.0

3.2.2 SA Reduction in Resource Binding

Power optimized binding for modules, registers and interconnects has been widely explored in the literature as discussed in Section 2.3. A scheduled DFG of the Differential Equation Solver, as shown in Figure 3.2 and its SA Table in Table 3.1, is used here to illustrate the impact of resource binding on SA. Table 3.1 is constructed using the switching activity profiler introduced in Section 3.2.1.

Figure 3.2 A Scheduled DFG of the Differential Equation Solver

Inter-iteration SA, which is the SA of two values belonging to different sets of stimuli, is not considered here. In a resource-constrained synthesis, suppose the resource constraint is two multipliers ($M1, M2$) and one ALU ($ALU1$) with the ALU capable of performing addition, subtraction and comparison. Two of the possible bindings, namely $B1$ and $B2$, for the schedule with 5 control steps (csteps) shown in Figure 3.2, can be obtained as shown in Table 3.2 and Table 3.3. If the power factor (which is defined as a variable

relating to the load capacitance of a resource) of the multiplier is 21.2 while that of the ALU is normalized to 1 (obtained from Synopsys Design Compiler). *B1* and *B2* will have the total power cost in terms of SA given by:

$$\begin{aligned} cost(B1) &= 21.2 * (SA(m1, m4) + SA(m4, m6) + SA(m2, m3) + SA(m3, m5)) + \\ &\quad 1 * (SA(a1, a2) + SA(a2, a3) + SA(a3, a4) + SA(a4, a5)) \\ cost(B2) &= 21.2 * (SA(m1, m3) + SA(m3, m6)) + 21.2 * (SA(m2, m4) + SA(m4, m5)) \\ &\quad + 1 * (SA(a1, a2) + SA(a2, a3) + SA(a3, a4) + SA(a4, a5)) \end{aligned}$$

where *SA* (*m1*, *m4*) is the SA that occurs at *M1* caused by switching from *m1* to *m4* when moving from cstep 1 to 2 and is equal to *SA* (*i1*, *i4*) + *SA* (*i2*, *c1*) = 8 + 7.81 = 15.81 according to Table 3.1, with *SA* (*i1*, *i4*) being the SA caused by switching the input *i1* of *M1* at cstep 1 to *i4* at cstep 2. And *SA* (*i2*, *c1*) is the SA at the other input of *M1*. Therefore *cost* (*B1*) = 1327 (When optimal operand matching for every module is assumed, explained in Section 3.3.4) and similarly, *cost* (*B2*) = 1156. It is clear that Binding *B1* yields a 14.8% higher SA than *B2*. *B2* is preferred and could be found by optimizing the SA between consecutive operations that are executed in a module. The technique is also applicable to bindings of registers, memories, multiplexers and buses.

Table 3.2 Resource Binding B1

Resources	CStep 1	CStep 2	CStep 3	CStep 4	CStep 5
M1	m1	m4	m6		
M2	m2	m3	m5		
ALU1	a1	a2	a3	a4	a5

Table 3.3 Resource Binding B2

	CStep 1	CStep 2	CStep 3	CStep 4	CStep 5
M1	m1	m3	m6		
M2	m2	m4	m5		
ALU1	a1	a2	a3	a4	a5

3.2.3 SA Reduction in Operation Scheduling

Although different resource bindings result in different SA, hence different power consumption, the process of resource binding itself depends on the results of operation scheduling, which can totally or partially alter the execution sequence of operations. Therefore, it is also important to consider SA during scheduling as different schedules lead to different bindings. Consider another schedule of DiffEq shown in Figure 3.3 which takes SA into account, a different binding $B3$ can be obtained as shown in Table 3.4. The SA cost of $B3$ is now:

$$\begin{aligned} \text{cost}(B3) = & 21.2 * (\text{SA}(m6, m1) + \text{SA}(m1, m5)) + 21.2 * (\text{SA}(m2, m4) + \text{SA}(m4, m3)) \\ & + 1 * (\text{SA}(a1, a2) + \text{SA}(a2, a5) + \text{SA}(a5, a3) + \text{SA}(a3, a4)) \end{aligned}$$

Again if optimal operand matching is assumed, then $\text{cost}(B3) = 659$, which is 101.4% and 75.4% lower than $\text{cost}(B1)$ and $\text{cost}(B2)$ respectively. The reduction is partly due to the different schedule of $B3$ where $m1$ and $m6$, which have the same inputs, are scheduled into two consecutive csteps so that they can share a module to reduce the SA. Clearly, it is desirable to obtain $B3$ rather than $B1$ or $B2$. In traditional list scheduling, urgency of the nodes decides the priority of the scheduling sequence of the operational nodes and as a result, $m1$, $m2$, $m3$ and $m4$ will be scheduled first before $m6$, depriving the chance for $m6$ to share the same module with $m1$ in two consecutive csteps. With this example, it is clear that scheduling has vital impact on binding when SA is considered. When binding determines the operation flow that is executed on a resource and thus has a direct influence on the power consumption, scheduling brings in different bindings and potentially different SA and thus has an indirect but significant influence on the power

consumption. To fully explore the minimization of power consumption, both scheduling and binding should be considered, not separately but in an integrated way. The proposed LAS algorithms presented in the next section perform scheduling and binding simultaneously aiming to optimize the SA.

Figure 3.3 A Power-Conscious Scheduling of the DiffEq

Table 3.4 Resource Binding B3

	CStep 1	CStep 2	CStep 3	CStep 4	CStep 5
M1	m6	m1	m5		
M2	m2	m4	m3		
ALU1	a1	a2	a5	a3	a4

3.3 Algorithm Formulation of LAS

In the proposed LAS technique, operation scheduling, module binding and register binding are accomplished in an integrated manner. During scheduling and module

binding, all operations that are ready for scheduling and binding at a control step are evaluated for the current control step as well as a number of control steps ahead. Several schedules and bindings are generated and their switching activities are calculated. An optimal binding in terms of SA cost is then chosen as the schedule and binding for the current step and all the others are discarded before the process is moved to the next cstep, which is treated as unscheduled. In this way, local minima due to scheduling of operations on a single csteps are minimized. After all the operations are scheduled and bound, register binding is performed not only to reduce SA on registers, but also to eliminate the spurious SA occurred at modules. LAS is applied to both resource-constrained synthesis (RC-LAS) and latency-constrained synthesis (LC-LAS). In RC-LAS, resource usage is constrained in advance and it aims to optimize SA with minimum latency overhead. In LC-LAS, the resources usage optimization and SA reduction are handled simultaneously while satisfying the latency constraint. RC-LAS and LC-LAS are described below.

3.3.1 Resource-Constrained Look-Ahead Synthesis (RC-LAS)

RC-LAS is developed based on a modified list scheduling approach. In each cstep, instead of selecting the urgent operations on the critical path for scheduling, operations are selected in order to reduce SA between the current cstep and the previous scheduled cstep as well as between the current cstep and a number of csteps ahead. A series of local schedule and binding pairs (called *sb* pairs) on the current cstep and the T_A number of csteps ahead are established. The establishment of the *sb* pairs is achieved by first generating several *sb* pairs for the current cstep. Each pair is in turn used to derive more *sb* pairs for the next cstep and the process repeated until the look-ahead T_A csteps are reached. The SA costs of these *sb* pairs are evaluated and the best pair is used for backtracking of T_B number of csteps, where the best *sb* pair is chosen as the final schedule and binding before the process moves to the next cstep. RC-LAS is outlined in Algorithm 3.1, where M is the resource constraints. The function *look_ahead_sb* () in Line 5 is to find a set of *sb* pairs for T_A csteps ahead and is described in Algorithm 3.2. The function *best_of* () in Line 6 evaluates the SA cost of each *sb* pair and chooses the

best one, which is used for backtracking to update the current schedule and binding. The function *backtrack* () in Line 7 performs backtracking on some csteps to reduce local minima. The function *local_reg_binding* () in Line 9 performs register binding with power optimization as described in Section 3.3.3.

// Algorithm 3.1: RC-LAS

```

1: RC-LAS (DFG, M) {
2:   Simulate SA of the DFG and construct SA Table;
3:   (scurrent, bcurrent) = scheduling and binding with no operation scheduled or bound;
4:   while (scurrent has operations not scheduled) {
5:     (S, B) = look_ahead_sb (scurrent, bcurrent, TA, M);
6:     (s, b) = best_of (S, B);
7:     (scurrent, bcurrent) = backtrack (s, b, TB);
8:   }
9:   local_reg_binding (scurrent, bcurrent);
10: }
```

The functions used in Algorithm 3.1 are described below.

- (1) *look_ahead_sb* (*s*, *b*, *T_A*, *M*) finds a set of scheduling and binding pairs for the next *T_A* csteps for the module set *M*, based on the current scheduling *s* and binding *b*. It is described in Algorithm 3.2.
- (2) *best_of* (*S*, *B*) selects the best scheduling and binding in terms of SA from a set of scheduling and binding pairs *S* and *B*.
- (3) *backtrack* (*s*, *b*, *T_B*) backtracks and updates the synthesis information for scheduling *s* and binding *b* for *T_B* steps. Suppose the synthesis information of *s* and *b* reaches the *N*-th cstep, then after the backtracking, synthesis information is kept up to the (*N*-*T_B*)-th cstep.
- (4) *local_reg_binding* (*s*, *b*) performs power-conscious register binding for the current scheduling *s* and binding *b*. This function is described in Algorithm 3.4 in Section

3.3.4.

```

*****
// Algorithm 3.2: Look-Ahead Scheduling and Binding
1: look_ahead_sb ( $s_{current}, b_{current}, T_A, M$ ) {
2:    $(S, B) = (s_{current}, b_{current}) \cup \Phi$ ; //  $(S, B)$  is a set of  $(s, b)$  pairs
3:   Level1: for ( $i = 0; i < T_A; i++$ ) {
4:      $(S', B') = \Phi$ ;
5:     Level2: for (each  $(s, b) \in (S, B)$ ) {
6:       if (all operations in  $s$  have been scheduled and bound) continue Level2;
7:       // set of operations ready for scheduling
8:        $ready\_nodes = get\_schedule\_operations(s)$ ;
9:        $(S', B') = (S', B') \cup generate\_sb\_pairs(s, b, ready\_nodes, M)$ ;
10:    }
11:   }
12:   return  $(S, B)$ ;
13: }
*****

```

The functions used in Algorithm 3.2 are described below.

- (1) *get_schedule_operations* (s) finds the operation nodes that are ready for scheduling in the current control step. All predecessors of these operation nodes must have already been scheduled in scheduling s to avoid any precedence violation.
- (2) *generate_sb_pairs* ($s, b, ready_nodes, M$) derives a set of scheduling and binding pairs from the current scheduling s and binding b . For each new scheduling and binding pair, certain operation nodes are selected from the *ready_nodes* for scheduling and binding as described in further details below.

The look-ahead technique is effective in avoiding local minima. However, if the number of nodes or the look-ahead number of steps (T_A) is large, the number of *sb* pairs generated

could be extremely big, resulting in inefficient computation. For example, if at a certain cstep the number of available operation candidates c_1 for operation type 1, c_2 for operation type 2, c_n for operation type n , and the number of available modules are m_1 for operation type 1, m_2 for operation type 2, m_n for operation type n , the exhaustive searching of sb pairs will result in a total of $C_{c_1}^{m_1} C_{c_2}^{m_2} \dots C_{c_n}^{m_n}$ possible combinations, which is large, time-consuming to generate and unacceptable for cost evaluation. In the proposed algorithm, the generation of the sb pairs is limited to those that are more likely to maximize the reduction of SA between the current and the previous cstep, as well as the SA between the current and the next cstep. First the function *get_schedule_operations ()* in Line 7 finds a set of nodes such that they are ready for scheduling subject to precedence. From the set of ready operations, the function *generate_sb_pairs ()* in Line 8 applies a branch and bound approach to find a set of schedule and binding pairs. During the branch and bound process, the most optimal (lowest SA cost) sb pair is first found by the bipartite matching, which schedules and binds the nodes at the same time as described later. Then each node of the optimal sb pair is removed from the set of ready operation nodes and the next optimal sb pair is found. The process repeats until the set of ready operation nodes is empty or an sb pair with high enough SA cost is found. From practical experience, it is found that an SA cost of 10% higher than the most optimal sb pair is high enough to preclude further search. Therefore, the number of sb pairs generated is restricted, which also limits the generation of more sb pairs during the subsequent look-ahead csteps. The DiffEq example of Figure 3.1 is used here as an illustration. In the first step, the ready operation nodes are $\{m1, m2, m4, m6, a1\}$. If the module resources are $\{M1, M2, ALU1\}$, a matching between $\{M1, M2, ALU1\}$ and $\{m1, m2, m4, m6, a1\}$ can be found using the bipartite matching algorithm, and say the binding $\{m1, m6, a1\}$ is the match. However, if the SA between $m1$ and $m6$ is very small, the opportunity of mapping them on the same module is lost. With this consideration, another matching between $\{M1, M2\}$ and $\{m1 \text{ (or } m6), m2, m4\}$ is searched.

With a set of schedule and binding pairs found within T_A number of csteps, *backtrack ()* keeps the (s, b) with the lower SA cost and discards all the other schedules and bindings. Backtracking can be any number of csteps between 0 to $T_A - 1$, i.e. $0 \leq T_B \leq T_A - 1$.

Backtracking with 0 cstep means no backtracking and the schedule and binding including all the csteps up to T_A is accepted. Backtracking with $TA - 1$ csteps means only 1 cstep of the best (s, b) is accepted and the schedules and bindings on all the other csteps are discarded.

Each (s, b) pair in the function *generate_sb_pairs* () is found by the weighted bipartite matching algorithm, which performs the scheduling and binding simultaneously. A standard weighted bipartite matching algorithm applied to the scheduling and binding problem is described here.

Weighted Bipartite Matching is an algorithm devised decades ago to solve the optimal matching problem in graph theory. Relevant definitions are given below.

Definition 3.1: A Bipartite Graph B is a graph $B = (V, U, E)$, where

- (1) The vertices are partitioned into two subsets V and U such that $V \cup U = \Phi$.
- (2) Each edge $e_i = (v_i, u_j) \in E$ has one end point $v_i \in V$ and the other end point $u_j \in U$.

Definition 3.2: A Weighted Bipartite Graph is a bipartite graph $WB = (V, U, E, W)$, such that

- (1) $W = \{w_{i,j} | i = 1, 2, \dots, |V|, j = 1, 2, \dots, |U|\}$ is a set of weight numbers.
- (2) For each edge $e_i = (v_i, u_j) \in E$, $w_{i,j} \geq 0$.

Definition 3.3: The matching problem is to find a matching MA for a graph $G = (V, E)$ such that

- (1) $MA \subset E$.
- (2) No two edges of MA share the same node.
- (3) $|MA|$ is the maximum and $|MA| \leq |V| / 2$.

Definition 3.4: A Weighted Bipartite Matching Problem is to find a matching MB for a weighted bipartite graph $WB = (V, U, E, W)$ such that

$$(1) \sum_{j=1}^{|U|} x_{i,j} = 1, \text{ for } i = 1, 2, \dots, |V|.$$

$$(2) \sum_{i=1}^{|V|} x_{i,j} = 1, \text{ for } j = 1, 2, \dots, |U|.$$

$$(3) x_{i,j} \geq 0.$$

$$(4) \sum_{i,j} w_{i,j} x_{i,j} \text{ is the minimum.}$$

Where

$$(1) N = |V| = |U|.$$

$$(2) w_{i,j} \in W.$$

$$(3) X = \{x_{i,j} \mid i, j = 1, 2, \dots, N\} \text{ is a set of variables and } x_{i,j} \in \{0, 1\}, \text{ and}$$

$$x_{i,j} = \begin{cases} 1 & \text{if } (v_i, u_j) \in MB \\ 0 & \text{if } (v_i, u_j) \notin MB \end{cases}$$

Figure 3.4 (a) gives an example of the weighted bipartite graph, in which $\{v1, v2, v3\}$ and $\{u1, u2, u3\}$ are two sets of nodes. A weight is associated with every edge that connects a node in one set to a node in the other set. The weighted bipartite matching problem is to find the optimal matching between two sets of nodes in terms of total weight. A solution is given in Figure 3.4 (b), where the total weight is $2 + 3 + 4 = 9$.

(a) A Weighted Bipartite Graph

(b) The Optimal Matching

Figure 3.4 An Example of Weighted Bipartite Matching and Its Solution

The weighted bipartite matching problem can be solved by the Hungarian method, in $O(N^3)$ arithmetic operations. Detailed description of the weighted bipartite matching problem and the Hungarian method can be found in [PAP82].

When applying $WB = (V, U, E, W)$ to the scheduling and binding problem here, V is the set of available modules, U is the set of operations and w_{ij} is the SA when $u_j \in U$ is assigned to $v_i \in V$. Note that some dummies nodes, whose weights are all zero, are created to make the cardinality of U and V equal. The complexity is $O(N^3)$, in which N is the number of available modules. The bipartite matching algorithm matches operations with modules available at a cstep and assigns the operations to the matched modules and the csteps, performing scheduling and binding simultaneously.

3.3.2 Latency-Constrained Look-Ahead Synthesis (LC-LAS)

A low-complexity version of the proposed LAS algorithm is developed for latency-constrained synthesis problem. Given the latency constraint, LC-LAS tries to optimize both resource usage and SA. Resources can also be pre-constrained as a starting point. The number of pre-constrained resources is either derived from the result of any traditional latency-constrained synthesis or given arbitrarily. During the synthesis process, more resources are allocated if necessary.

LC-LAS employs the same concept of assuming future synthesis information to guide the current synthesis process. Unlike RC-LAS, LC-LAS always looks ahead only one step, so that it has the full control of the latency while maintaining the dependency between operations, which also makes LC-LAS less complex compared to RC-LAS. Algorithm 3.3 lists the pseudo codes of LC-LAS, where L is the maximum latency allowed and M is the initial resource constraints.

In LC-LAS, scheduling and binding are also performed simultaneously in a stepwise manner. In each iteration, the mobility information of all the unscheduled operations is first updated. Two temporary bindings, fixed-first binding (b_f) and mobile-first binding

(b_m) , are then found. The two bindings are obtained as described below. Let O_a be the set of operations already scheduled, O_c the set of operations of which the mobility contains the current cstep and O_n the set of operations of which the mobility contains the next cstep. b_f and b_m are both derived from first binding some selected operations (denoted as O_{cs}) from O_c and then binding some selected operations (denoted as O_{ns}) from O_n . The difference is that when deriving b_f , some operations from O_c are selected and bound so that the SA between O_a and O_{cs} is first optimized, hence the name fixed-first binding, and then the SA between O_{cs} and O_{ns} . When deriving b_m , some operations (O_{cs}) from O_c and some operations (O_{ns}) from O_n are selected and bound first so that the SA between O_{cs} and O_{ns} is optimized before the SA between O_{cs} and O_a is optimized, hence the name mobile-first binding. In either case, the node selection is done by using the same bipartite matching algorithm described in Section 3.3.2 and critical operations are guaranteed to be scheduled to satisfy the latency constraint. If the number of some types of critical operations exceeds the number of their resources, more resources are added. After b_f and b_m are obtained, the one with a lower SA cost is used to update the original binding. Both b_f and b_m are two steps longer than the original binding. However, only the current cstep of the original binding is updated. In another word, LC-LAS performs the synthesis of not only just the current cstep but also one look-ahead cstep.

// Algorithm 3.3: LC-LAS

```

1: LC-LAS (DFG, L, M) {
2:   simulate SA of the DFG and construct SA Table;
3:   (s, b) = scheduling and binding with no operation scheduled or bound;
4:   cstep = 1;
5:   while (s has unscheduled operations){
6:     profile_mobility (s, cstep, L);
7:     (sf, bf) = fixed_first_synthesis (s, b, cstep);
8:     (sm, bm) = mobile_first_synthesis (s, b, cstep);
9:     if (cost (bf) < cost (bm)) update (s, b, sf, bf);

```

```

10:      else update (s, b, sm, bm);
11:      cstep = b.cstep + 1;
12:  }
13:  local_reg_binding (s, b);
14: }

```

The functions used in Algorithm 3.3 are described below.

- (1) *profile_mobility* (*s*, *cstep*, *L*) profiles the mobility of every unscheduled operation nodes according the current scheduling *s*, subject to the current control step *cstep* and the latency constraint *L*.
- (2) *fixed_first_synthesis* (*s*, *b*, *cstep*) derives the synthesis information from scheduling *s* and binding *b* for the *cstep* and (*cstep* + 1) in a fixed-first manner, in which SA between (*cstep* - 1) and *cstep* takes optimization priority.
- (3) *mobile_first_synthesis* (*s*, *b*, *cstep*) derives the synthesis information from scheduling *s* and binding *b* for the *cstep* and (*cstep* + 1) in a mobile-first manner, in which SA between *cstep* and (*cstep* + 1) takes optimization priority.
- (4) *cost* (*b*) calculates the SA cost for binding *b*.
- (5) *update* (*s*, *b*, *new_s*, *new_b*) updates the synthesis information of scheduling *s* and binding *b* from the newly derived scheduling *new_s* and binding *new_b*.
- (6) *local_reg_binding* (*s*, *b*) performs power-conscious register binding for the current scheduling *s* and binding *b*. This function is given in Algorithm 3.4 in Section 3.3.4.

The DiffEq example is used again here to illustrate the LC-LAS process. Suppose *m2* is scheduled in *cstep* 1 and the current step is *cstep* 2. The maximum latency *L* is set to 6 csteps. $O_c = \{a1, m1, m4, m6\}$ and $O_n = \{a1, a2, a5, m1, m3, m4, m5, m6\}$. In the fixed-first binding, operations in O_c are selected and bound to resources so that minimal SA occurs. Suppose these operations are $O_{cs} = \{a1, m4\}$. After *a1* and *m4* are bound, operations in O_n are processed in the same manner. Suppose the selected operations in O_n are $O_{ns} = \{a5, m6\}$. Hence, the temporary binding b_f would be the initial binding *b* plus $\{a1, m4\}$ in *cstep* 2 plus $\{a5, m6\}$ in *cstep* 3. Note that this binding actually violates the

operation dependency, because $a5$ depends on $m6$ and they cannot be scheduled in the same cstep. This situation is allowed in LC-LAS because the binding for cstep 3 is only temporary and will be discarded after updating, i.e. only the current cstep binding (cstep 2 in this case) is adopted.

In the mobile-first binding, the optimal matching between $O_c = \{a1, m1, m4, m6\}$ and $O_n = \{a1, a2, a5, m1, m3, m4, m5, m6\}$ is first found. Then certain pairs of O_{cs} and O_{ns} are selected using bipartite matching, e.g., $O_{cs} = \{a1, m6\}$ and $O_{ns} = \{a2, m5\}$, so that the SA between current cstep and next cstep is minimal. In this case, $\{a1, m6\}$ will be scheduled in cstep 2 and $\{a2, m5\}$ will be scheduled in cstep 3.

The costs of b_f and b_m are then compared to determine which one should be used to update the current binding. If b_f yields a lower cost, $\{a1, m4\}$ will be adopted as the binding in cstep 2 of the current binding, otherwise, $\{a1, m6\}$ are adopted for cstep 2. In either case, the binding for cstep 3 is discarded and the process moves to cstep 3.

In either the fixed-first binding or the mobile-first binding, momentary dependency violation is possible and allowed in LC-LAS in the last cstep (current cstep + 1, which is the look-ahead cstep). On the other hand, critical operations are guaranteed to be scheduled and bound in both the current cstep and the look-ahead cstep. If there are more critical operations than the resources available, additional resources are allocated in order to satisfy the latency constraints.

3.3.3 Register Binding

In the LAS, register binding is performed after scheduling and binding. The aims of the register binding are two folds. First, it optimizes the SA occurred on the registers. Second, it eliminates the spurious SA on the modules due to unnecessary data loading into the registers. In the proposed register binding technique, the optimization of the SA on the registers is performed in a stepwise manner. In each cstep, the edges to be bound are first found, and then suitable registers are selected. Finally the optimal binding for this step is found using the bipartite matching.

When the value of a register changes in a cstep, SA occurs on all the modules driven by the register. If some modules are not used in this cstep, the SA is spurious. One way to avoid the spurious SA is to use operand isolation, which means to freeze the inputs of a module when it is idle. This can be achieved by putting transparent latches on the module inputs. However, this method not only introduces area overhead and complicates the control logic, but also increases the critical path delay, which is highly undesirable in high performance systems. In the proposed technique, a different approach is used. In this approach, spurious SA elimination is achieved by avoiding using the registers which will introduce spurious SA. In every register binding step, the registers connected to idle modules are reserved and cannot be used to bind edges in the current step. If this results in insufficient registers, new registers are allocated. Although extra registers may be introduced (not always), spurious SA is completely eliminated. The register binding process is described in Algorithm 3.4.

// Algorithm 3.4: Local Register Binding

```

1: local_reg_binding (s, b) {
2:   profile_life_time (s, b); // find the life time of each edge in s
3:   for (step = 0; step < s.cstep; step++) {
4:     reserved_regs =  $\Phi$ ;
5:     for each module  $\in$  modules not used in this step {
6:       // find reserved registers, which can lead to spurious SA
7:       regs = last_used_regs (module);
8:       reserved_regs = reserved_regs  $\cup$  regs;
9:     }
10:    available_regs = b.regs  $\setminus$  reserved_regs; // exclude reserved registers
11:    edges = edges whose life time contains this step;
12:    for each edge  $\in$  edges {
13:      if (edge is already bound) {
14:        edges = edges  $\setminus$  edge; // remove edge

```

```

14:         available_regs = available_regs \ get_reg (edge); // remove register
15:     }
16: }
17: if (|available_regs| < |edges| ) allocate_new_regs (b, available_regs);
18: matching = bipartite_matching (available_regs, edges);
19: bind (available_regs, edges, matching, step);
20: }
21: }

```

The functions used in Algorithm 3.4 are described below.

- (1) *profile_life_time* (*s*, *b*) profiles the life time information of every edge in the scheduled DFG.
- (2) *last_used_regs* (*module*) finds all the registers that have fed the module before the current cstep.
- (3) *get_reg* (*edge*) gets the register to which the DFG edge is bound.
- (4) *allocate_new_regs* (*b*, *available_regs*) allocates new registers to the binding so that there are enough registers to perform the binding.
- (5) *bipartite_matching* (*available_regs*, *edges*) finds the optimal matching between registers *available_regs* and DFG edges in terms of total switching activity using the weighted bipartite matching algorithm. In this case, for the bipartite graph $WB = (V, U, E, W)$, V is the set of available registers, U is the set of edges and w_{ij} is the SA when $u_j \in U$ is bound to $v_i \in V$.
- (6) *bind* (*available_regs*, *edges*, *matching*, *step*) performs the binding from DFG edges to registers *available_regs* according to their optimal matching.

Spurious SA on modules is effectively removed by reserving registers that will introduce spurious SA. Note that this method intrinsically increases the number of registers needed. From the experiment result shown in Section 3.5.5, it is observed that the number of registers is increased by 11%. Constraints on the number of registers can also be set to obtain a trade-off between registers used and SA reduction. Note that the proposed

spurious SA technique targets on removing the unnecessary SA on RTL level structure. In terms of lower level design (logic level and physical level), spurious SA can still occur because of bad timing when the circuit is not well designed.

3.3.4 Operand Matching Optimization

Usually, an operation has many operands, and accordingly, the module on which the operation is executed also has several inputs. Therefore, suppose there are n operands for an operation and n inputs of the module, there would be $n!$ possibilities for the assignment from operands to inputs.

Operand matching has a great impact on the total SA. Consider the case when a two-operand ($c1$ and $c2$) multiplication is bound to a two-input multiplier ($i1$ and $i2$). Suppose the old values on $i1$ and $i2$ are $c1'$ and $c2'$ respectively. When $c1$ is bound to $i1$ and $c2$ is bound to $i2$, the total SA would be $SA1 = SA(c1', c1) + SA(c2', c2)$; when $c1$ is bound to $i2$ and $c2$ is bound to $i1$, the total SA would be $SA2 = SA(c1', c2) + SA(c2', c1)$. $SA1$ and $SA2$ could differ a lot, especially when two consecutive operations executed on the same module share certain operands.

In the experiments of this work, all operations have two operands and all modules have two inputs, i.e. $n = 2$. Therefore an exhaustive search through the $n! = 2! = 2$ possibilities is used. However, for a large n , $n!$ could be extremely large, making an exhaustive search not practical. It is observed that this operand matching problem can be addressed by the weighted bipartite matching algorithm. In this case, for the bipartite graph $WB = (V, U, E, W)$, V is the set of operands of the operation, U is the set of inputs of the module and w_{ij} is the SA when input $u_j \in U$ is bound to input $v_i \in V$.

3.4 An Illustrational Example

In this section, an illustrational example based on the DiffEq is used to show the main processes of the proposed RC-LAS, with the resources available being two multipliers and one ALU. Multiplications are executed on multipliers, while ALU performs additions, subtractions and comparisons. The SA table is given in Table 3.1. Suppose the look ahead

csteps are two, and backtrack step is one, i.e. $T_A = 2$, and $T_B = 1$ in Algorithm 3.1. The following are the main steps.

(a) Initially $cstep = 1$ and the candidate operations for scheduling are $\{m1, m2, m4, m6, a1\}$. When the SA between $cstep = 1$ and the previous steps are considered, it is the same to schedule any operation, because there is no previous cstep. However, the SA between $cstep = 1$ and $cstep = 2$ could vary a lot when selecting different operations in $cstep = 1$. Suppose there are two scheduling and binding patterns $B1$ and $B2$ for cstep 1 and 2 as shown in Table 3.5.

Table 3.5 Bindings Considered for Cstep 1

Binding		Cstep 1	Cstep 2
B1	M1	m6	m1
	M2	m2	m4
	ALU1	a1	a2
B2	M1	m1	m6
	M2	m4	m2
	ALU1	a1	a2

Clearly Binding $B1$ and Binding $B2$ have the same cost, which is also much less than any other bindings. Suppose $B1$ is selected. As backtracking step $T_B = 1$, only the scheduling of cstep 1, e. g. $\{m6, m2, a1\}$, is selected as part of the final scheduling.

(b) $cstep = 2$ and the candidate operations for this step are $\{m1, m4, a5, a2\}$. By bipartite matching, the minimum SA between the previous csteps and $cstep = 2$ occurs when $\{m1, m4, a2\}$ are selected, because $SA(a1, a2) = 10.7 < SA(a1, a5) = 15.32$. However, as the difference is not much, Binding $\{m1, m4, a5\}$ is also allowed to predict the binding in cstep 3 to minimize the SA from cstep 1 to cstep 3. The patterns found are shown in Table 3.6.

Table 3.6 Bindings Considered for Cstep 2

Binding		Cstep 1	Cstep 2	Cstep 3
B3	M1	m6	m1	m5
	M2	m2	m4	m3
	ALU1	a1	a2	a5
B4	M1	m6	m1	m5
	M2	m2	m4	m3
	ALU1	a1	a5	a2

As the cost of $B3$ is less than the cost of $B4$, $B3$ is selected for cstep 2, which is $\{m1, m4, a2\}$. $B4$ and cstep 3 of binding $B3$ are discarded.

(c) $cstep = 3$ and the candidates are $\{m3, m5, a5\}$. As there are sufficient resources, looking ahead to $cstep = 4$ is not needed. Bipartite matching algorithm is used to find the optimal binding for this cstep. The same situation is encountered when scheduling $cstep = 4$ and $cstep = 5$. The final result is shown in Table 3.7.

Table 3.7 Final Resource Binding

	Cstep 1	Cstep 2	Cstep 3	Cstep 4	Cstep 5
M1	m6	m1	m5		
M2	m2	m4	m3		
ALU1	a1	a2	a5	a3	a4

The LC-LAS is a simplified version of the LAS technique with lower complexity. In LC-LAS, always two steps are looked ahead and one step is backtracked ($T_A = 2, T_B = 1$). Besides, instead of repetitively searching for synthesis patterns for local T_A steps, it always compares two patterns and selects the less expensive one. The main procedure is illustrated in Section 3.3.2.

3.5 Experimental Results

Several benchmarks were used to test RC-LAS and LC-LAS. IIR is a standard 4th-order Butterworth IIR filter. FIR is a 20th-order Equiripple Symmetric FIR filter. DiffEq is a Differential Equation Solver. Lattice is a 3rd-order Normalized Lattice filter [CHU96]. EWF is a 5th-order Elliptical Wave Filter [GEB92]. TFIR is a 10th-order FIR filter using a transposed implementation. FDCT is an implementation of Fast Discrete Cosine Transformation using FFT. DCT is an implementation of standard 5th-order Discrete Cosine Transformation [BEL00].

3.5.1 Data Flow Graph Construction

To carry out the experiment on a benchmark, first the benchmark must be converted into a DFG. Consider a simple example of a second-order IIR filter, as illustrated in Figure 3.5.

Figure 3.5 The Structure of A Second-Order IIR Filter

This filter consists of a primary input i , a primary output o , 4 multiplications, 4 additions, and two delays ($d1$ and $d2$). To convert this structure into a DFG representation, every

delay node is split into an intermediate output and an intermediate input. The DFG of Figure 3.5 is represented in Figure 3.6.

Figure 3.6 The DFG Representation of the Second-Order IIR Filter

In this DFG representation, there are 3 inputs (i , $i1$, $i2$), 4 constant inputs (the coefficients of the four multiplications), and 3 outputs (o , $o1$, $o2$). Statistically, there are 8 DFG nodes (4 multiplications and 4 additions) and 15 DFG edges.

A very important feature of this DFG is the intrinsic relation between $o1$ and $i1$ ($o2$ and $i2$). Because $o1$ and $i1$ ($o2$ and $i2$) are constructed from a delay element $d1$ ($d2$), the value of $o1$ ($o2$) in a certain iteration must feed the value of $i1$ ($i2$) in the next iteration. This feature must be included in simulation program for SA table construction, as described in Section 3.2.1, in order to obtain a realistic and accurate result to build the SA table and estimate the power consumption.

The DFG information of the experimental benchmarks is listed in Table 3.8. The largest benchmark is the DCT, which has 44 DFG nodes and 60 DFG edges. The smallest one is the Differential Equation Solver, with 11 DFG nodes and 17 DFG edges.

The width of the data path is 16 bits, and Q-15 fixed-point format is used to represent the data value possessed by every DFG edge during the simulation of SA table construction. Two types of resources are used. One is multiplier, capable of performing multiplications; the other is ALU, capable of performing addition, subtraction and comparison. When the total cost is calculated, the power factor of ALU is normalized to 1. The power factor of a multiplier is 21.2, which is the power consumption ratio of multiplier over ALU reported by Synopsys Design Compiler.

Table 3.8 Information of the Experimental Benchmarks

Benchmark	#nodes	#edges	#inputs	#constants	#outputs
IIR	9(*)8(+)	29	5	7	5
FIR	11(*)20(+)	63	21	11	21
DiffEq	6(*)2(+)2(-)1(<)	17	4	2	4
Lattice	14(*)6(+)2(-)	36	4	10	4
EWF	8(*)26(+)	50	8	8	8
TFIR	11(*)10(+)	43	11	11	11
FDCT	16(*)13(+)13(-)	51	8	1	8
DCT	18(*)23(+)3(-)	60	6	10	6

3.5.2 Experiments on RC-LAS

RC-LAS employs the LAS concept on a resource-constrained synthesis problem. In this experiment, the look ahead step T_A is set to 2, and the backtrack step T_B is 1. Table 3.9 shows the comparisons on the total SA occurs on both modules and registers in RC-LAS, which are also illustrated in Figure 3.7.

Table 3.9 Experimental Results of Total SA for RC-LAS

Benchmark	Clique Partition (1)	Optimized (2)	RC-LAS	$\Delta\%$ over (1)	$\Delta\%$ over (2)
IIR	2572	1394	1178	54.2	15.5
FIR	3573	2660	2304	35.5	13.4
DiffEq	1896	1170	673	64.5	42.5
Lattice	4096	3100	2373	42.1	23.5
EWF	2513	1854	1808	28.1	2.5
TFIR	3184	1170	970	69.5	17.1
FDCT	3926	1589	1419	63.9	10.7
DCT	6601	4575	2091	68.3	54.3
Average	3545	2189	1602	54.8	26.8

Figure 3.7 Total SA Reductions for RC-LAS

The column “Clique Partition” indicates the average SA obtained when list scheduling with the module resource constraint is used for scheduling and clique partition with register constraint is used for module and register binding, without any attempt to optimize SA. The column “Optimized” is the average SA obtained from list scheduling and weighted bipartite matching binding with optimization on SA in only the current

control step. Compared to the list scheduling with clique partition binding, the proposed RC-LAS can reduce the SA significantly in all the benchmarks with an average reduction of 54.8%. This reduction shows the great impact of scheduling and binding on SA. The reduction over “Optimized” shows the influence of scheduling on SA. For the 8 benchmarks tested, the reduction varies from 2.5% to 54.3% with an average of 26.8%, which means scheduling also contributes significantly for reducing SA. Generally, the more possible ways of scheduling a DFG (e.g. DCT), the greater SA reduction can be achieved. In cases like EWF, the critical path is relatively long compared to the number of operations, so there are less possible schedules and hence, less space for the proposed RC-LAS to optimize the SA. The DCT benchmark has many scheduling possibilities and thus offers greater space for SA optimization.

Table 3.10 Latency Comparison for RC-LAS

Benchmark	Resources	List Scheduling	RC-LAS
IIR	3(MULT) 1(ALU)	9	11
FIR	2(MULT) 3(ALU)	9	10
DiffEq	2(MULT) 1(ALU)	5	5
Lattice	2(MULT) 1(ALU)	9	12
EWF	1(MULT) 3(ALU)	16	17
TFIR	3(MULT) 2(ALU)	6	6
FDCT	2(MULT) 3(ALU)	12	12
DCT	2(MULT) 3(ALU)	12	14

In resource-constrained synthesis, a common concern is the latency. Table 3.10 presents the latency comparison between the result of critical-path list scheduling and the result of the proposed RC-LAS. From Table 3.10, it can be seen that the latency increased is quite small. This is because RC-LAS is based on list scheduling. The only difference is that it schedules operation based on another criterion – the SA. As a result, in every control step, operations will be scheduled as many as possible. This increase can easily be minimized by incorporating the critical path information into the scheduling procedure, e.g.

operations in the critical path can be selected arbitrarily. Therefore, it can be concluded that the SA reduction at the cost of slightly increased latency is acceptable.

3.5.3 Influence of Look-Ahead and Backtracking

In the previous section, the results from a typical configuration of Look-Ahead and Backtracking ($T_A = 2$ and $T_B = 1$) are compared with the results from a power-unconscious synthesis process. Here some results from some typical look-ahead and backtracking configurations are compared, as shown in Table 3.11. The unit is the total SA. The fluctuation is clearly shown in Figure 3.8. It can be seen from Table 3.11 that SA decreases as look-ahead steps and backtracking steps increase.

Table 3.11 Influence of Look-Ahead and Backtracking

Benchmark	$T_A=1$	$T_A=2$	$T_A=2$	$T_A=3$	$T_A=3$	$T_A=3$
	$T_B=0$	$T_B=0$	$T_B=1$	$T_B=0$	$T_B=1$	$T_B=2$
IIR	1308	1308	1178	1203	1178	1178
FIR	2522	2403	2304	2286	2286	2286
DiffEq	897	673	673	673	673	673
Lattice	2668	2516	2373	2407	2366	2366
EWF	1896	1808	1808	1799	1808	1808
TFIR	1211	1058	970	1058	958	958
FDCT	1482	1434	1419	1452	1404	1404
DCT	2988	2186	2091	2121	2076	2066

Figure 3.8 Fluctuation of Total SA

When the influence of look-ahead steps is analyzed, the results of $T_A = 2$ show decent improvement compared to the results of $T_A = 1$. However, the results of $T_A = 3$ are negligibly advantageous over the results of $T_A = 2$ at the cost of much higher algorithmic complexity. Therefore, it is concluded that for DFG with moderate size, $T_A = 2$ is sufficient to obtain a reasonable satisfactory optimization result as shown in Section 3.5.2.

When the influence of backtracking is analyzed, for both $T_A = 2$ and $T_A = 3$, increasing T_B exhibits SA reduction. Although averagely the reduction is not significant, there are some cases where increasing T_B leads to decent SA reduction. The results suggest that T_B should be set to a fairly large value, because although it incurs some speed degradation, the increased algorithmic complexity is only proportional to the latency, making it unimportant in high-order algorithms.

3.5.4 Experiments on LC- LAS

LC-LAS is a latency-constrained power-conscious high-level synthesis system employing the LAS concept. It is a simplified version of the LAS algorithm, in which T_A is always 2 and T_B is always 1. It schedules operations based on their mobility. In every control step, operations that are critical in this step are forced to be scheduled. Initial resource allocation allocates certain number of resources based on either common latency-

constrained scheduling algorithm or arbitrary assignment. Additional resources are allocated if necessary during the binding process. In this section, results of RC-LAS and LC-LAS are compared. The result of RC-LAS is used as the latency constraint for LC-LAS. Besides, the resource constraint used in RC-LAS will guide the initial resource allocation of LC-LAS. Table 3.12 shows the comparison.

Table 3.12 Comparison of RC-LAS and LC-LAS

Benchmark	RC-LAS			LC-LAS			SA
	Resources	Latency	SA	Resources	Latency	SA	$\Delta\%$
IIR	3(*) 1(+)	11	1178	3(*) 1(+)	11	1177	0.0
FIR	2(*) 3(+)	10	2304	2(*) 3(+)	10	2406	-4.4
DiffEq	2(*) 1(+)	5	673	2(*) 1(+)	5	674	0.0
Lattice	2(*) 1(+)	12	2373	2(*) 1(+)	12	2541	-7.1
EWf	1(*) 3(+)	17	1808	1(*) 3(+)	17	1917	-6.0
TFIR	3(*) 2(+)	6	970	3(*) 2(+)	6	1097	-13.1
FDCT	2(*) 3(+)	12	1419	2(*) 3(+)	11	1583	-11.6
DCT	2(*) 3(+)	14	2091	2(*) 3(+)	14	1923	8.0

In Table 3.12, the resource usage and latency are basically the same (except for the FDCT benchmark) for RC-LAS and LC-LAS. This shows that LC-LAS yields a satisfactory resource distribution and usage. The SA for LC-LAS is marginally worse than that for RC-LAS. However, compared with (1) and (2) of Table 3.9, it still produces significant SA reduction.

3.5.5 Influence of Register Binding

The proposed register binding algorithm uses the weighted bipartite matching algorithm as well as a register reservation and management scheme to achieve both register SA reduction and spurious SA removal. The influence of register binding is two-fold. First, the SA consumed by registers is reduced. Second, the spurious SA on modules due to unnecessary register loading is eliminated. Table 3.13 shows the influence of register binding on both the SA on registers and the SA on modules. SA reduction on registers is

illustrated in Figure 3.9, while total SA reduction due to the optimization of registers is shown in Figure 3.10.

Table 3.13 Influence of Register Binding on Total SA and Register SA

Benchmark	Cliques Partition Register Binding			Optimize Register Binding			Reg SA $\Delta\%$	Total SA $\Delta\%$
	#Reg	Reg SA	Total SA	#Reg	Reg SA	Total SA		
IIR	12	24.73	2215	14	21.95	1178	11.2	46.8
FIR	32	45.56	2838	32	36.90	2304	19.0	18.8
DiffEq	9	10.68	1044	10	8.60	673	19.5	35.5
Lattice	11	35.36	3115	12	30.32	2373	14.3	23.8
EWf	11	47.68	2241	13	37.18	1808	22.0	19.3
TFIR	14	41.85	2007	17	30.62	970	26.8	51.7
FDCT	13	53.59	1580	17	43.04	1419	19.7	10.2
DCT	20	60.72	2675	21	46.65	2091	23.2	21.8
Average	-	40.02	2214	-	31.91	1602	20.3	27.7

Figure 3.9 SA Reduction on Registers

Figure 3.10 Total SA Reductions Due to Register Optimization

In LAS, one-step register binding is used, in which register binding is optimized for each control step. In this way, register binding optimization and spurious SA reduction can be performed simultaneously. Therefore the proposed algorithm not only optimizes the SA on registers, but also reduces the SA on modules (by eliminating spurious SA). As can be seen in Table 3.13, the register SA is reduced by 20.3% in average compared to the register binding based on clique partition method. The total SA is reduced by 27.7% in average due to the removal of spurious SA. This reduction is at the cost of extra registers. However, it is found from the experimental results that only few extra registers (sometimes zero) are needed, which is justifiable by the SA reduction.

3.5.6 Algorithmic Complexity

The proposed LAS technique is very practical when computation time is concerned. Suppose there are N control steps, and in every look-ahead step, the maximum total number of nodes is M , then the algorithmic complexity is in the order of $O(N*M)$. Note that in the worst case, M is exponential to the number (L) of nodes that are ready to be scheduled within the look-ahead steps. However, as L is often small, M is seldom large. In the rare cases when M is large, an effective way is to limit the number of bad schedules and bindings. In this way, computation time can be parameterized to trade off

optimization results for large designs. Our experiments show that optimization result degrades by only a negligible factor.

LAS is implemented in Java and runs on a 2.4GHz Pentium PC with 256MBytes of RAM. The computation time is shown in Table 3.14.

Table 3.14 Computation Time of LAS

Benchmark	# Nodes	# Edges	RC-LAS Run Time (s)	LC-LAS Run Time (s)
IIR	17	29	1.72	1.66
FIR	31	63	3.05	2.87
DiffEq	11	17	1.66	1.06
Lattice	22	36	3.36	2.65
EWF	34	50	2.21	2.07
TFIR	21	43	3.17	2.54
FDCT	42	51	5.23	4.86
DCT	44	60	15.24	7.22

In Table 3.14, the RC-LAS uses 2 look-ahead steps, and the limit of modules (M) is set to 2 times of the number of nodes that are ready to be scheduled in the look-ahead steps. It is observed that in addition to the size of the DFG, the critical path delay also influences the computation time. For example, the computation time of DCT is relatively long while that of EWF is relatively short. The reason is that DFGs with short critical path delay tend to have more possible schedules, thus more synthesis possibilities to be evaluated. The LC-LAS algorithm is more computational efficient than the RC-LAS algorithm, with very small degraded performance, as can be seen in Table 3.12.

3.5.7 Comparison

As pointed out by the authors of [MOH05], different algorithms have different objectives, constraints and overhead, a direct fair comparison is not quite possible. In general, the power characterization, the switching activity simulation result, the formulation of

benchmarks, many algorithm parameters and even coding styles all make a fair comparison very difficult. Here attempt is made to perform some comparisons using the same approach in [MOH05] and the results are summarized in Table 3.15 and Table 3.16, with the purpose to provide a general idea on the performance of LAS. Table 3.15 compares the average improvement over power-unconscious techniques with some published results. In [MUR03], a game-theoretic technique combining scheduling and binding achieves an average improvement of 39.2% over 7 benchmarks. In [RAG94], a heuristic grouping technique achieves an average improvement of 9.8% over 10 benchmarks. In [LUO04], 28% improvement over 7 benchmarks is achieved by a max-cost multi-commodity network algorithm. The proposed RC-LAS algorithm obtains 54.8% improvement over 8 benchmarks. The game theoretic algorithm in [MUR03] and RC-LAS both adopt a combined scheduling and binding technique. The computation time of RC-LAS is also better than that of [MUR03].

Table 3.15 Average Improvement (%) Comparisons for RC-LAS

	[MUR03]	[RAG94]	[LUO04]	RC-LAS
Average Improvement (%)	39.2	9.8	28	54.8

Table 3.16 Individual Improvement (%) Comparisons for RC-LAS

Benchmark	[MUR03]	[RAG94]	[LUO04]	RC-LAS
IIR	38.8	-	-	54.2
FIR	47.3	-	-	35.5
DiffEq	39.8	8.1	23.47	64.5
Lattice	50.5	-	-	42.1
EWF	39.7	5.6	23.48	28.1
FDCT	-	-	23.87	63.9

Table 3.16, where “-” indicates the results are unavailable in the published literature, shows the comparison over some benchmarks. It can be observed that in most cases, RC-LAS obtains a better result. The only exception is the EWF, for which RC-LAS can only

achieve a 28.1% reduction, while [MUR03] achieves a 39.7% reduction, which confirms that RC-LAS is less efficient for circuits with a long critical path delay compared to the number of operations, as the design space is small and RC-LAS is not able to take the advantage of the look-ahead technique.

3.6 Summary

For the same inputs and same functionality in VLSI, different circuit structures give different switching activities and hence consume different amounts of power. In high level synthesis, both scheduling and binding have a major influence on the circuit structures, and thus on the power consumption. Binding (modules, registers and interconnects) optimization for power has been widely explored in the literature of high level synthesis and in many systems, re-scheduling has been used to optimize scheduling. However, re-scheduling can easily be influenced by local effect due to the complexity of scheduling. Therefore a good start-point is essential for iterative re-scheduling techniques.

In the proposed Look-Ahead Synthesis technique, scheduling and binding are performed simultaneously to explore the influence of both scheduling and binding on SA. To reduce the effects of local minimum, LAS adopts a branch and bound approach to perform concurrently scheduling and binding for not just one cstep but also several csteps ahead and uses the optimal one for backtracking. Register binding is then performed not only to optimize the SA on registers, but also to eliminate spurious SA on modules.

Two high-level synthesis systems, RC-LAS (Resource-Constrained Look-Ahead Synthesis) and LC-LAS (Latency-Constrained Look-Ahead Synthesis) are presented in this chapter. Experimental results show that both scheduling and binding have a great impact on power consumption. The proposed look-ahead technique can optimize both of them and reduce the SA substantially. The experimental results obtained showed that the look-ahead technique is most effective for complex designs with many possible scheduling and binding combinations. The results also demonstrate that the “look-ahead technique” is able to escape from local minima, which gives much better results for the benchmarks with large design space for exploration.

Chapter 4 Voltage Scaling and Frequency Scaling for Energy Optimization

4.1 Introduction

In high level synthesis, switching activity reduction can be realized on functional units, registers, memories and interconnects, and has been one of the power optimization dimension and addressed in Chapter 3. Recently research focus has been extended to the optimization of other factors for power and energy reduction. A major reason is that the benefit of switching activity reduction techniques strongly depends on the structure of the circuit. It means that for a given DFG, it is possible that the gain from reducing switching activity may be limited even if latency and area are heavily traded off. This happens when the average switching between every pair of edges does not differ much in the DFG.

It is stated in Chapter 2 that power and energy reduction can be effectively achieved by reducing the supply voltage but incurs tradeoff between performance and power/energy because reducing voltage will increase circuit delay. A popular method to address the tradeoff is to use multi-voltage datapath, in which resources in the critical path are supplied with higher voltage to compensate performance, while resources not in the critical path are supplied with lower voltage to save power.

In this chapter, a technique called MuVoF (Multi-Voltage Multi-Frequency Synthesis) is proposed to address the low-energy synthesis problem for functionally pipelined datapath, under resource and throughput constraint. Functionally pipelined datapath [DEM94] refers to the type of datapath that consists of multiple stages, each of which processes a different set of data. As it can provide high throughput compared to conventional datapath, it is widely used in high speed applications. Functionally pipelined structure has existed for a long time, but has not been explored for low energy synthesis. In MuVoF, the datapath is partitioned into several pipeline stages, each of which operates at its own frequency, under a global throughput constraint. Multi-voltage scheme is also used.

Unlike other proposed multi-voltage design, the proposed technique allocates the resources without any initial voltage assignment. Voltage assignment to resources is updated on-the-fly, depending on the pipeline stage partitioning, frequency and scheduling. Throughput constraint is given as real time unit instead of number of csteps. To the best knowledge of the author, this is the first technique addressing the low-energy problem for functionally pipelined circuits using multi-voltage and multi-frequency technique.

MuVoF consists of two steps of processing. The first step (core process) assumes some initial information, such as initial single clock period, initial single supply voltage and initial non-pipeline scheduling. It optimally partitions the initial datapath into several pipeline stages, scales frequencies in each stage, and then performs multi-voltage assignment, under resource and throughput constraint. The second step is an iterative process. It iteratively performs several local moves to refine the final datapath to reduce energy.

The rest of this chapter is organized as follow. Section 4.2 introduces the basic idea of multi-frequency and multi-voltage behavioral synthesis for functionally pipelined datapath. Section 4.3 presents the core process and the iterative process of the proposed algorithm in details. This is followed by experimental results in Section 4.4. Finally, a summary is given in Section 4.5.

4.2 MuVoF Synthesis for Functionally Pipelined

Datapath

The proposed MuVoF technique generates functionally pipelined datapath with multiple supply voltages and multiple frequencies. Multi-voltage datapath is described in Section 2.4. Unlike traditional single-voltage datapath, resources in a multi-voltage datapath can operate at different voltage levels. When the output of a resource drives the input of another resource with higher supply voltage, a level converter is placed between them to enhance the signal voltage. The datapath is also functionally pipelined with multiple voltages, which is introduced below.

4.2.1 Multi-Frequency Functionally Pipelined Datapath Model

Pipeline is a common technique to increase the throughput of digital circuits. In a non-pipelined circuit, all inputs must wait for the outputs of the last iteration to be completed, while in a pipelined circuit, inputs can feed the circuit before the calculation of the last iteration is finished. In other words, in a non-pipelined circuit, the input data introduction time, δ , must satisfy $\delta \geq \text{clock period} * \text{latency}$, while in pipelined circuits, this condition is not necessary. Actually, for a fully pipelined circuit, $\delta = \text{clock period}$, i.e. that new data can be introduced in every clock cycle.

In terms of high level synthesis, there are two types of pipeline [DEM94]. The first type is resource pipeline, in which pipelined resources are used. The second type is functional pipeline, in which the circuit is partitioned into multiple pipeline stages, i.e. the whole circuit is a pipelined resource. In this chapter, high level synthesis techniques for energy optimization based on functional pipeline are proposed.

During the high level synthesis for functionally pipelined datapath, the DFG is partitioned into multiple stages, as illustrated in the example in Figure 4.1.

Figure 4.1 A Partitioned DFG for Functionally Pipelined Datapath Synthesis

This DFG, which has 4 csteps, is partitioned into two pipeline stages. The first stage consists of the first and second csteps, and the second stage consists of the other two steps. The two stages can operate concurrently, with their data communication controlled by a controller. The input data can be introduced after the completion of the the first stage, instead of the completion of the whole circuit (which is the case in a non-pipelined structure). The output of the first stage will feed the second stage, which is controlled by the controller.

Suppose the clock cycle is 5ns, and the throughput constraint is 1/20ns (the circuit is required to complete the processing of every sample in 20ns). Note that it only takes $5\text{ns} * 2 = 10\text{ns}$ to complete both stages, thus 10ns is wasted on the waiting for the new input data. From this observation, it is proposed here that the clock period in both stages be extended to 10ns. As the clock cycle is extended, resources can tolerate a longer delay. Therefore the supply voltage may also be scaled down if the delay under the new supply voltage does not incur any timing violation. In a word, frequency scaling introduces opportunities for supply voltage scaling and thus reduces the energy indirectly. In a pipelined structure where certain stages have different latencies, frequency scaling could lead to multi-frequency functionally pipelined datapath. Figure 4.2 illustrates such a pipelined datapath model.

Figure 4.2 Multi-Frequency Functionally Pipelined Datapath Model

The datapath consists of multiple pipeline stages, each of which is synchronized by a scaled clock. All the pipeline stages work concurrently and the data communication between consecutive stages is controlled by a global FSM controller. This controller is clocked by a master clock, of which the clock period is typically the inverse of the throughput constraint.

It is worth noting that although pipeline is a powerful technique to enhance the performance, it comes at a cost. Pipelined circuit typically requires more resources. Consider the DFG of Figure 4.1, Stages 1 and 2 typically do not share resources, because they are running concurrently. Although there exist some binding algorithms that consider cross-stage resource sharing, they usually suffer from defects such as high complexity, extremely complicated controlling logic, and the sharing resource is usually very limited.

4.2.2 Multi-Voltage and Multi-Frequency Synthesis

Multi-voltage synthesis involves the process of identifying the criticalness of operations. Operations in the critical paths need to be assigned with higher voltage supply to maintain performance, while non-critical operations are assigned with voltages as low as possible. A scheduled DFG of an IIR filter in Figure 4.3, with a single supply voltage 5.0V is used here to illustrate the concept of multi-voltage multi-frequency functionally pipelined design, which forms the basis of the multi-voltage multi-frequency behavioral synthesis for energy reduction.

Figure 4.3 DFG of IIR Filter with Supply Voltage = 5.0V

Given ALU and multiplier as the available library components, the characterization of these resources is adopted from [MOH03] and listed in Table 4.1 and Table 4.2, assuming the average switching on a signal is 0.5.

In Table 4.1, worst case delay estimation is used, so the delay is calculated by equation (4.1):

$$d = d_{FU} + d_{MUX} + d_{Reg} + d_{Conv} \quad (4.1)$$

where d_{FU} , d_{MUX} , d_{Reg} and d_{Conv} are the delay of functional unit, multiplexer, register and level converter respectively.

Table 4.1 Delay and Eenergy Characterization of Library Components

Library Component	5.0V		3.3V		2.4V	
	Delay (ns)	Energy (pJ)	Delay (ns)	Energy (pJ)	Delay (ns)	Energy (pJ)
ALU	25.7	57	45.5	25	76.8	13
Multiplier	54	2202	96.6	960	163.7	507
MUX	-	9	-	4	-	2

Table 4.2 Energy Characterization of Level Converter

V1 \ V2	5.0V	3.3V	2.4V
5.0V	-	-	-
3.3V	178.1	-	-
2.4V	139.4	53.04	-

For convenience, the examples in this section only consider energy of FUs. For the DFG in Figure 4.3, the functional units consist of two multipliers performing multiplication, and one ALU performing addition, subtraction and comparison. Suppose the original power supply is 5.0V, and voltages available are 5.0V, 3.3V and 2.4V. In a simple case, if the clock period is 50ns, the supply voltage of the adder can be directly reduced from 5.0V to 3.3V because the delay of an ALU under 3.3V is still less than 50ns. In this case, the datapath becomes a multi-voltage design, with 5.0V for the two multipliers and 3.3V for the adder.

However, if the clock period is 30ns, there is no simple way to reduce the voltage of any functional units. In this case, the total delay of the design is 30ns * 16 = 480ns. If the input data arrives faster than this, a non-pipelined datapath derived from this DFG cannot sustain the throughput constraint. One feasible solution is to allocate additional resources to construct a functionally pipelined datapath. Suppose the total available resources are 4 multipliers and 2 ALUs, and the input data arrives every 300ns, which means the throughput constraint is 1/300ns. There are several ways to partition the original datapath into multiple pipeline stages under both resource constraint and throughput constraint. Figure 4.4 gives a feasible partition with two pipeline stages.

Figure 4.4 A Two-Stage Pipeline Partition of Figure 4.3

Consider the first pipeline stage of Figure 4.4, which consists of 6 csteps and with a total delay of $30\text{ns} * 6 = 180\text{ns}$. Obviously, as the input data arrives every 300ns , the clock period of this stage can actually be expanded to $300\text{ns} / 6 = 50\text{ns}$.

In a functionally pipelined datapath, the total energy E can be modeled as (4.2), where N is the number of stages and M_i is the number of resources in Stage i .

$$E = \sum_{i=1}^N E_i = \sum_{i=1}^N \sum_{j=1}^{M_i} C_{i,j} V_{i,j}^2 \alpha_{i,j} \quad (4.2)$$

If the clock period of the first pipeline stage of Figure 4.4 remains 30ns , using the same library components and the same average switching activity in every transition (0.5) in Table 4.1, the energy of Figure 4.4 is calculated as:

$$EI = 2202 * 9 + 57 * 8 = 20274\text{pJ}$$

Obviously the clock period of Stage 1 can be extended to 50ns . Note that this multi-frequency scheme cannot directly reduce energy, because energy consumption does not depend on frequency. However, when the clock period of a pipeline stage is extended, there is a longer slack in the stage. This promotes the opportunity for performing multi-voltage technique on the stage. A multi-voltage multi-frequency datapath version of Figure 4.4 is given in Figure 4.5.

Figure 4.5 A Multi-Voltage Multi-Frequency Functionally Pipelined Datapath
(Operations without Voltage Notation Are Assigned with 5.0V)

It can be observed that in the first pipeline stage, all operations can be assigned with 3.3V, without any violation in operation dependency, throughput constraint, or resource constraint. Note that the supply voltage of one of the multipliers in the second stage is also reduced to 3.3V. This is because there are less csteps in the second stage than in the original DFG, which reduces the chance of resource constraint violation if the supply voltage of some resource is scaled down. Consider the original DFG in Figure 4.3, if operation 13 is assigned with 3.3V, there would be one 3.3V multiplier and one 5.0V multiplier. In this case, either operation 20 or operation 14 has to extend into cstep 7, which breaks resource constraint. This is because now two 3.3V multipliers and one 5.0V multiplier are needed in cstep 7. This situation does not happen after the partition, as shown in Figure 4.5 (b). The energy of Figure 4.5 is:

$$E2 = 960*6 + 25*2 + 960*1 + 2202*2 + 57*6 = 11516\text{pJ}$$

This energy $E2$ yields a 43.2% reduction over $E1$, indicating that careful pipeline stage partitioning, multi-frequency and multi-voltage design can achieve significant energy reduction for functionally pipelined datapath.

4.2.3 Influence of State Partitioning and Scheduling

Different stage partitions result in different frequency scaling in the partitions, which will also change the multi-voltage assignment. A different partition scheme is shown in Figure 4.6 for comparison with Figure 4.4.

Figure 4.6 An Alternative Two-Stage Pipeline Partition of Figure 4.3

With this way of partition, the clock period can be extended to $300\text{ns} / 8 = 37.5\text{ns}$ for both pipeline stages. Voltage scaling can be performed as shown in Figure 4.7, where there are one 3.3V multiplier, two 5.0V multipliers and one 5.0V ALU in the first

pipeline stage, and one 5.0V multiplier and one 5.0V ALU in the second pipeline stage. The total resources are still four multipliers and two ALUs. The energy of Figure 4.7 is:

$$E3 = (960*2 + 2202*6 + 57*2) + (2202*1 + 57*6) = 17790pJ$$

This only achieves 12.3% reduction in energy when compared to $E1$, which are not as significant as that achieved by $E2$. The partitioning process is especially important when relatively large amount of resources are available or the throughput constraint is relatively slack compared to the original delay. This is because in both cases there are many partitioning possibilities, and it becomes more important to select an optimal one.

Figure 4.7 Third Multi-Voltage Multi-Frequency Functionally Pipelined Datapath of Figure 4.3 (Operations without Voltage Notation Are Assigned with 5.0V)

In addition to the stage partitioning process, another factor that influences the energy reduction is the original scheduling before stage partitioning and multi-voltage assignment. This is because different schedules lead to different stage partitions and

voltage assignment. Consider an alternative scheduling scheme of the IIR filter as shown in Figure 4.8.

Figure 4.8 An Alternative Scheduled DFG of IIR Filter

The datapath requires three multipliers and one ALU, and a delay of $30\text{ns} * 14 = 420\text{ns}$. As before, the throughput constraint is set to be $1 / 300\text{ns}$, and the allocated resources are four multipliers and two ALUs. One possible multi-voltage multi-frequency functionally pipelined datapath with 3 pipeline stages is shown in Figure 4.9.

Figure 4.9 A Multi-Frequency Multi-Voltage Functionally Pipelined Datapath of Figure 4.8

The first pipeline stage is occupied by three 3.3V multipliers, and has a clock period of $300\text{ns}/3 = 100\text{ns}$. The second pipeline stage has one 2.4V ALU, and a clock period of 100ns. The third pipeline stage has one 2.4V multiplier and one 3.3V ALU, and a clock period of 60ns. Therefore this datapath satisfies the resource constraint (4 multipliers and 2 ALU) and throughput constraint (300ns), but the datapath is quite different from Figure 4.5, because they are derived from two different non-pipeline scheduled DFGs. The energy of Figure 4.9 is:

$$E4 = (960*8) + (13*3) + (507*1 + 25*5) = 8351\text{pJ}$$

This achieves 58.8% energy reduction over $E1$, which is even better than the energy reduction of 43.2% obtained in Figure 4.5.

The above analysis indicates that given the resource and throughput constraint, throughput constraint, there are three major factors that influence the final energy

consumption in a functionally pipelined datapath. They are the initial scheduling, the stage partitioning, and the multi-voltage assignment. In the technique proposed and described in the next section, an initial stage partitioning and voltage assignment are first obtained by assuming an initial scheduling, which is achieved in the core process algorithm described in Section 4.3.1. Then an iterative algorithm is performed to find an optimal scheduling and correspondent stage partitioning and multi-voltage assignment to achieve low energy, as described in Section 4.3.2.

4.3 Algorithm Formulation of MuVoF

From the analysis and discussions in Section 4.2, it can be seen that the multi-frequency and multi-voltage scheme is very effective for energy optimization in functionally pipelined datapath. The three important factors in this technique are initial scheduling, stage partitioning, and multi-voltage assignment. Optimizing only one of them or optimizing them separately will give less superior results. However, an optimal result can be achieved only by NP-hard algorithms, which are not feasible in practice. In this situation, iterative algorithms are desired to find a required tradeoff between algorithmic complexity and optimal result. In this section, a technique called MuVoF is proposed to address the low energy synthesis for functionally pipelined datapath using multiple voltages and multiple frequency approach. MuVoF consists of two steps: a first-step heuristic algorithm (core process) and a second-step iterative algorithm.

4.3.1 Core process of the Algorithm

Given a scheduled DFG, resource constraint and throughput constraint, the core process of the algorithm first searches for an optimal partition of the DFG into several pipeline stages, then assigns every operation a voltage supply as low as possible, while satisfying resource constraint, throughput constraint and operation dependency. Therefore the core process consists of a partitioning process and a multi-voltage assignment process and they are launched in sequence.

4.3.1.1 Stage Partition

The objective of the stage partitioning is to assign the operations to a number of pipeline stages such that each stage has the maximum clock period, i.e. minimum clock frequency, while satisfying the resource and throughput constraints. Note that a partition itself cannot reduce energy, but it facilitates the opportunities for reducing energy through voltage reduction. An optimal partition is the one that can produce an optimal clock period extension solution so that energy has the best opportunity to be reduced via voltage reduction. Note that when extending a clock, the new cycle time is selected from a predefined clock set. In this way, the number of clocks and cycle times can be bounded by the user. The current implementation of MuVoF does not consider multi-voltage assignment in the partitioning stage in order to keep the algorithmic complexity low, but it can be easily incorporated into the partitioning stage by redefining the cost function. The partitioning process is given in Algorithm 4.1.

// Algorithm 4.1: Stage Partition

```
// Input: a scheduled DFG (G), resource constraint (R), cycle-time (T),
// throughput constraint (TP) and number of csteps (N).
// Output: a functionally pipelined scheduled DFG, with different clock frequencies in
// different stages.

1: current_cost =  $\infty$ ;
2: s0 = initial_partition (); // initial empty partition
3: s_first = append_to_partition (s0, 1);
4: S = {s_first};
5: while (S  $\neq$   $\Phi$ ) {
6:   select an element s  $\in$  S;
7:   S = S \ {s};
   // first branch
8:   s1 = append_to_partition (s, s.step+1);
   // if satisfies throughput constraints
9:   if (s1.current_stage_len * T  $\leq$  1 / TP) {
```

```

10:      if (s1.low_bound < current_cost) {
11:          S = S ∪ {s1};
12:          if (s1.step == N) {
13:              current_best = s1;
14:              current_cost = s1.low_bound;
15:          }
16:      }
17:  }

  // second branch
18:  s2 = start_new_partition (s, s.step+1);
  // if satisfies resource constraints
19:  if (s.resources + resource_usage (s.step+1) ≤ R) {
20:      if (s2.low_bound < current_cost) {
21:          S = S ∪ {s2};
22:          if (s2.step == N) {
23:              current_best = s2;
24:              current_cost = s2.low_bound;
25:          }
26:      }
27:  }
28: }

29: update_stage_clock (current_best); // include clock period extension

```

The partitioning algorithm adopts a binary branch-and-bound approach. Suppose there are N csteps in the original un-partitioned DFG. During the partitioning process, first an empty partition is constructed by the function *initial_partition* () in Line 2, in which no control steps are processed yet, and then control steps are processed from the first to the last sequentially. A solution is defined as an intermediate partitioning scheme, in which the pipeline stages of some csteps have been determined while the pipeline stages of the remaining csteps are not fixed yet. Consider a solution when Figure 4.3 is processed:

$$s = \{(17, 16), (), (19, 15), ()\}$$

$$s.step = 4$$

$$s.resources = 2 \text{ multipliers}$$

This solution has one pipeline stage, which includes the first and second csteps (operation 17 and 16), and the third and fourth csteps (operation 19 and 15), with the operations in the same cstep enclosed by a pair of brackets “()”. The pipelining stages of csteps 5 to 16 are not yet determined.

Given a solution, for every cstep that has not been processed, there are two options, which form the two branches of the binary branch-and-bound algorithm.

The first option (Line 8 to Line 17) is to put the cstep into the current pipeline stage of the solution, which is realized by the function *append_to_partition (solution, cstep)* in Line 8. This option increases the latency of the current stage, and thus might violate the throughput constraints. This is first bound condition for this first branch (Line 9). If this option is performed for cstep 5, the solution becomes:

$$s1 = \{(17, 16), (), (19, 15), (), (14, 20, 21)\}$$

$$s1.step = 6$$

$$s1.resources = 2 \text{ multipliers} + 1 \text{ ALU}$$

The second option (Line 18 to Line 27) is to put the cstep into a new pipeline stage, which is realized by the function *start_new_partition (solution, cstep)* in Line 18. This option increases the resources that are needed, because resource sharing among stages is not allowed in the proposed technique, and different pipeline stages consume different set of resources, and thus may violate the resource constraints. This is the first bound condition for this second branch (Line 19). If this option is chosen, the solution becomes:

$$s2 = \{(17, 16), (), (19, 15), ()\} + \{(14, 20, 21)\}$$

$$s2.step = 4$$

$$s2.resources = 4 \text{ multipliers} + 1 \text{ ALU}$$

If no constraint is violated when either the first option or the second option is chosen, the new solution is put into the solution set and a cost lower bound of the new solution is calculated. When calculating the lower bound of a *solution* (s), a dummy solution (s') is derived from s , assuming that every un-processed cstep occupies one separate pipeline stage. The cost of this dummy solution s' is the lower bound of the original solution s , and this lower bound is the second bound condition for both options. For example, when the lower bound of $s2$ is calculated, $s2'$ is derived as:

$$s2' = \{(17, 16), ()\} + \{(19, 15), ()\} + \{(14, 20, 21)\} + \{(23)\} + \{(18, 13), ()\} + \{(22)\} + \{(24)\} + \{(26)\} + \{(25)\} + \{(27)\} + \{(28), ()\} + \{(29)\}$$

Suppose there are M pipeline stages in a solution, the cost function of this solution is:

$$\text{cost}(\text{SOLUTION}) = \sum_{i=1}^M \text{cost}(\text{STAGE}_i)$$

For the calculation of the cost of a stage, a so-called clock period extension technique is performed on this stage to extend the clock period under throughput constraints. As above-mentioned, the current implementation does not incorporate voltage reduction to reduce algorithmic complexity. Instead, it uses a cost function defined (4.3), where N is the number of stages, M_i is the number of resources in Stage i , E_{ij} is the energy required for the operation, and f_i is the frequency of Stage i .

$$\text{cost} = \sum_{i=1}^N \sum_{j=1}^{M_i} E_{i,j} f_i \quad (4.3)$$

This cost function indicates that, whenever possible, energy-intensive operations (such as multiplications) shall be assigned to stages that have lower frequency, such as they have the best opportunity to be assigned to low voltage resources.

If a new solution derived from option 1 or option 2 does not break any constraints, it will be evaluated. If the solution is complete (all operations are assigned to some stage of the solution) and its lower bound cost is less than the current best cost, the solution is stored and the current best is updated (Line 12 and Line 22). Note that when a solution is complete, its lower bound cost is equal to the cost of the solution. If the lower bound of the solution is accepted but there are unprocessed operations left, the solution is added as a new branch. The search stops when all csteps are processed, and the solution with the best cost recorded becomes the final solution, of which the clock period for every stage is extended under various constraints in the function *update_stage_clock (current_best)* in Line 29. The clock period extension is further explained in Section 4.4.3.

The stage partitioning problem is formulated as a binary branch-and-bound algorithm. It aims to find the optimal stage partitioning such that the clock period for every partition is the maximum (i.e. minimum clock frequency) under the resource and throughput constraints, and the cost, as defined in Equations (4.3), is the minimum.

4.3.1.2 Multi-Voltage Assignment

After the DFG is partitioned into multiple stages, multi-voltage assignment is performed on each pipeline stage. The multi-voltage assignment technique refers to the process of assigning voltages selected from a voltage domain to every operation under resource, operation dependency and throughput constraints. The algorithm is outlined as Algorithm 4.2.

// Algorithm 4.2: Multi-Voltage Assignment

// Input: Resource Types (RT), Resource Constraint (RC), Latency Constraint (L)

- 1: for each $t \in RT$ {
- 2: $V = \{v_i | i \in [1, M], v_1 > v_2 > \dots > v_M\}$; // M = number of available voltages
- 3: $R = RC(t)$;
- 4: for each $r \in R$ { $r.voltage = v_1$ };

```

5:   for ( $i = 1$ ;  $i < M$ ;  $i++$ ) reassign ( $v_i$ ,  $v_{i+1}$ ,  $R$ ,  $L$ ,  $t$ ); // See Algorithm 4.3
6: }

```

With a set of resource types (RT) and under the resource constraints (RC) and latency constraints (L), Algorithm 4.2 tries to reduce the resources for a particular pipeline stage in a step-by-step manner (Line 5) for every resource type (multiplications, additions, etc.). Note that for a pipeline stage, the latency constraints are the same as the throughput constraints for the whole datapath. This algorithm is also applicable for normal DFG instead of a pipeline stage. With a set of voltages for a particular resource type, supply voltage is reduced gradually from the highest voltage to the lowest voltage by the function *reassign* (v_H , v_L , R , L , t) in Line 5. For example, if the voltage set is [5.0V, 3.3V, 2.4V]. In this first step, the voltage of some resources of 5.0V is to be reduced to 3.3V. In the second step, the voltage of some resources of 3.3V is to be reduced to 2.4V.

Function *reassign* (v_i , v_{i+1} , R , L , t), which tries to reduce the voltage of some resources of type t from v_i to v_{i+1} under resource constraint R and latency constraint L is described as Algorithm 4.3.

// Algorithm 4.3: Voltage Reassignment

// Function *reassign* (v_H , v_L , R , L , t)

```

1:  $PR = \Phi$ ; // resources which have already been processed
2: Level1: while ( $r \in R$  &&  $r \notin PR$  &&  $r.voltage == v_H$ ) {
3:    $r.voltage = v_L$ ;
4:    $PR = PR \cup \{r\}$ ;
5:    $RN = \Phi$ ; // set of nodes whose voltages are reassigned
6:   for ( $cstep = 1$ ;  $cstep \leq L$ ,  $cstep++$ ) {
7:      $N = get\_match\_nodes(cstep, r, v_H)$ ;
       // try to scale down the voltage of a node
8:      $success = false$ ;

```

```

9:      if ( $n \in N$  &&  $\text{min\_voltage}(n) \leq v_L$ ) {
10:           $n.\text{voltage} = v_L$ ;
11:          if ( $\text{!check\_resource}(R, \text{cstep}, \text{cstep} + \text{latency}(n, v_L) - 1)$ )
12:               $n.\text{voltage} = v_H$ ;
13:          else {
14:               $RN = R \cup \{n\}$ ;
15:               $\text{success} = \text{true}$ ;
16:          }
17:      }
18:      if ( $\text{!success}$ ) { // check resource of  $v_H$ 
19:          if ( $\text{!check\_resource}(R, \text{cstep}, \text{cstep})$ ) {
20:              for each ( $rn \in RN$ )  $rn.\text{voltage} = v_H$ ;
21:               $r.\text{voltage} = v_H$ ;
22:              continue Level1;
23:          }
24:      }
25:  }
26: }
```

Generally, the algorithm tries to reduce the supply voltage from v_H to v_L for as many resources as possible (as in the Level1 loop). In Line 3, the voltage of a resource r is reassigned to v_L . This may break the latency constraint L because the latency of the resource is increased due to lower voltage. Also, as there are now one more resource with voltage v_L and one less resource with voltage v_H , resource constraint R may be violated in one way or another in some csteps. Therefore the constraints R and L must be checked in every cstep, as indicated in Line 6. The checking is explained below.

Suppose initially the resource r is used by an operation which starts at cstep A . Resource constraint violation will occur in cstep A if initially all resources with v_H are operating in cstep A (which means every resource in this cstep is assigned with an operation node).

This is because after reducing the voltage of r from v_H to v_L , there is one less resource with v_H , but the number of operations with v_H remains the same. This condition is checked by the function *check_resource* (R , $cstep$, $cstep$) in Line 19. If it fails, the process must backtrack all nodes whose voltage has been reassigned due to the voltage reduction of resource r , because it proves that the voltage of r cannot be reduced. Therefore the next iteration is launched, trying to reduce the voltage of another resource.

The solution to the above possible resource constraints violation is to try to reduce the voltage of an operation node from v_H to v_L in the current $cstep$ as well. In Line 7, the function *get_match_nodes* ($cstep$, r , v_H) returns all the nodes N that match with resource r in $cstep$ with voltage v_H . Line 9 checks if there exists a node n in N whose voltage can be reduced to v_L without breaking dependency constraints and latency constraints, in which the function *min_voltage* (n) returns the minimum voltage under dependency constraints and latency constraints. If so, the node is reassigned with voltage v_L , as in Line 10. Note that resource constraint may also be introduced in the new life time [$cstep$, $cstep + \text{latency}(n, v_L) - 1$] of n . This is because in some $csteps$ in the new life time of n , all resources of v_L may have already been used, leaving no resource of v_L to execute n . The function *latency* (n , v_L) returns the latency of n under voltage v_L . This possible violation is checked in Line 11. If failed, the voltage of n must be rolled back. If successful, there is no need to perform Line 19 for the current $cstep$. This is because now there is one less resource of v_L but also one less node of v_L , therefore no resource violation will occur. Note that Line 19 only checks resources in the current $cstep$, and the following $csteps$ will be checked in the following iterations. In this manner, voltage reduction is performed on as many resources and nodes as possible, with resource constraints and latency constraints maintained throughout.

In general, the core process consists of a pipeline stage partitioning process and a voltage scaling process. Algorithm 4.1 uses a branch-and-bound approach to find an optimal stage partition and a clock period extension technique is used to maximize the clock frequency for each pipeline stage. Algorithm 4.2 uses a heuristic algorithm to perform multi-voltage assignment for each pipeline stage under the resource and throughput

constraints. These two algorithms work together to form the core process, in which a multi-frequency multi-voltage functionally pipelined datapath is synthesized from the original scheduled DFG. To reduce the energy further, an iterative process is proposed in the next section.

4.3.2 The Iterative Process of the Algorithm

After the core process is completed, a multi-voltage multi-frequency functionally pipelined datapath is obtained. To further reduce energy, the datapath is refined by an iterative algorithm, which based on the simulated annealing technique [GER99]. The necessity of the iterative algorithm is twofold: (1) to optimize the energy consumption or other cost functions. In the core process, when performing stage partitioning in Algorithm 4.1, single-voltage design is assumed, because it yields lower complexity for the cost estimation in the branch-and-bound algorithm. This makes it impractical to set energy as the optimization target in Algorithm 4.1, because multi-frequency itself does not affect energy. Therefore a realistic way is to set the Equation (4.3) as the optimization target in the core process and set the energy consumption as the optimization target in the iterative algorithm. (2) to find the best scheduling for energy. In Section 4.2.4, it is proved that different schedules yield different stage partitions and multi-voltage assignments, and thus different power and energy consumption. The iterative algorithm is given as Algorithm 4.4 and explained below.

// Algorithm 4.4: Iterative Algorithm

```

1:  $s$  = initial solution from the core process;
2:  $T$  = initial temperature;
3: Level1: while (!stop ()) {
4:   Level2: while (!thermal_equilibrium ()) {
5:      $s' = \text{local\_move}(s)$ ;
6:      $\Delta c = \text{cost}(s') - \text{cost}(s)$ ;
7:     if ( $\Delta c \leq 0$ )  $s = s'$ ;
8:     else if ( $e^{\frac{-\Delta c}{T}} > \text{random}(1)$ )  $s = s'$ ;
```

```

9:    }
10:   T = new_temperature (T);
11: }
*****

```

Algorithm 4.4 consists of two loops. The Level2 loop count is controlled by the function *thermal_equilibrium ()* (Line 4), which ensures that enough number of local moves (Line 5) have been performed before the temperature is updated (Line 10). This number is the summation of number of nodes and number of edges in current implementation. Current implementation uses $T = R * T$ ($0 \leq R \leq 1$) to realize the function *new_temperature (T)*. The temperature controls whether a new solution is accepted or not (Line 8). Temperature is reduced in each iteration, which gradually makes it more and more difficult for a new solution with higher cost to be accepted. Larger value of R usually leads to longer convergence time. However, from extensive experiments, it is found that the factor must be relatively large to achieve a good result (due to the high complexity of the problem), and 0.84 gives a good tradeoff between computational complexity of quality of results. The function *stop ()* in Line 3 determines the termination of the annealing process. The process stops when an upper bound of the number of iterations is reached or no local move is accepted in the inner loop.

A new solution is obtained by performing a local move on a known solution. In this technique, four types of local moves are proposed and described below.

Type 1: push up. This move pushes the first cstep of a pipeline stage into the preceding stage. Consider the datapath of Figure 4.4. Suppose the partition solution is given as:

$$s = \{(17, 16), (), (15, 19), (), (21, 20, 14), (23), (13, 18), ()\} + \{(22), (24), (26), (25), (27), (28), (), (29)\}$$

After push-up is performed on the second stage, operation 22 is moved into the second stage, and the solution becomes:

$$s' = \{(17, 16), (), (15, 19), (), (21, 20, 14), (23), (13, 18), (), (22)\} + \{(24), (26), (25), (27), (28), (), (29)\}$$

Type 2: push down. This move pushes the last cstep of a pipeline stage into the succeeding stage. For example, after push-down is performed on the first stage of solution s , operation 18 is pushed down to the second stage and the solution becomes:

$$s' = \{(17, 16), (), (15, 19), (), (21, 20, 14), (23)\} + \{(13, 18), (), (22), (24), (26), (25), (27), (28), (), (29)\}$$

Type 3: assign extra resources and split stage. There are some cases that more resources are available than the current assigned resources. This happens either when more resources are manually allocated for optimization or when after serials of local moves, total resource usage is changed. In this case, extra resources are allocated to the stage which yields the maximum cost reduction. When a pipeline stage is assigned with extra resources, it may be split into two stages while the new resource constraint is still satisfied. For example, suppose one extra ALU is allocated, the second stage of s can be split into two stages, and s becomes:

$$s' = \{(17, 16), (), (15, 19), (), (21, 20, 14), (23), (13, 18), ()\} + \{(22), (24), (26), (25), (27)\} + \{(28), (), (29)\}$$

Type 4: reschedule a pipeline stage. This move reschedules all the operations of a certain pipeline stage. Incorporated with other types of moves, the move can change the whole scheduling scheme of the DFG, and totally alter the influence of the original scheduling, which is the input of Algorithm 4.1.

The four types of moves are randomly selected in the function *local_move* (). On the other hand, the stage on which a local move is performed in the last iteration has lower priority to be altered in this iteration compared to the other stages. After a move is

performed on a pipeline stage, clock period extension (described in the next subsection) and multi-voltage assignment are performed on the new stage, aiming to find an optimal datapath for the new partitioning or new scheduling. The cost is evaluated by the total energy consumption of the new solution. In this way, the simulated-annealing-based iterative algorithm, together with the four types of local moves, further reduces energy in an iterative manner.

4.3.3 Clock Period Extension

Clock period extension is essential in MuVoF to reduce energy. In the process, the clock period of a pipeline stage is extended under the resource and throughput constraints. Consider a pipeline stage shown in Figure 4.10.

Figure 4.10 A Pipeline Stage

The first method to perform clock period extension is to directly extend the clock period while throughput constraint is not violated. Suppose the clock period of Figure 4.10 is 30ns, and the throughput constraint is 1 / 300ns, then the clock period can be extended to $300\text{ns} / 4 = 75\text{ns}$.

Another straightforward way to perform clock period extension is to merge consecutive csteps while resource constraint is still maintained. Consider Figure 4.10, it is obvious that cstep 1 and cstep 2 can be merged into one step, while cstep 3 and cstep 4 can be merged into another step. This extends the clock period to $30\text{ns} * 2 = 60\text{ns}$, and the resource usage is the same.

By combining these two methods, the new clock period can be extended to $300\text{ns} / 2 = 150\text{ns}$.

Clock period extension may introduce some deserted csteps, in which there is no operation. By removing these deserted csteps, clock period can be further extended. Suppose the clock period of Figure 4.10 has been extended to 75ns , and the delay of all multipliers is less than 75ns . The datapath becomes the one as shown in Figure 4.11.

Figure 4.11 A Pipeline Stage with Deserted Steps

Cstep 2 and 4 have no operation, therefore become deserted and can be removed. After the removal, there are two csteps and the clock period can be extended again to $300\text{ns} / 2 = 150\text{ns}$. This achieves the same result as the step merging does.

The three types of moves, direct extension, step merging and deserted step removal, form the basis of the clock period extension. Whenever the clock is extended, the three methods are performed again, until no further extension is possible.

4.4 Experimental Results

To carry out experiments, characterization of library components must first be established. In this chapter, the characterization used is taken from [MOH03], and is shown in Table 4.1 and Table 4.2. The characterization includes delay and energy consumption. Two types of functional unit are used: ALU and Multiplier. ALU performs addition, subtraction and comparison, while multiplier performs multiplication. The multi-voltage domain includes 5.0V, 3.3V and 2.4V. Switching activity of 0.5 is assumed when calculating the energy of an FU. The delay is computed as the summation of functional unit delay, register delay, multiplexer delay and level converter delay. Energy is calculated on FUs and level converters. Experiments were carried out on several benchmarks, including IIR, FIR, EWF (Elliptic Wave Filter), Lattice Filter, DCT, Fast DCT (using FFT), Transposed FIR filter, Differential Equation Solver and Elliptical Filter. The characteristics (number of nodes, edges, etc.) of these benchmarks can be found in Section 3.5.

The result of the MuVoF strongly depends on the throughput and resource constraints. When the throughput and/or resource constraints are loose, MuVoF can fully exploit the design space and reduce energy most efficiently. In the experiments, different constraints were applied to show the influence of resource and throughput constraints on energy consumption. Experiments under various constraints were carried out in order to illustrate the influence of resource constraints and throughput constraints, as presented in the following subsections.

4.4.1 Experiments under Tight Constraints

In this experiment, relatively tight constraints were applied to the benchmarks to examine the effectiveness of MuVoF. These constraints were obtained by manually estimating the lower bound of the constraints and extending it by a small amount to create some search space for optimization. The resource and throughput constraints used are given in Table 4.3.

Table 4.3 Relatively Tight Constraints for MuVoF

Benchmark	IIR	FIR	EWF	Lattice	DCT	FDCT	TFIR	DiffEq	Elliptical
FU	2(+)	5(+)	5(+)	2(+)	6(+)	6(+)	3(+)	2(+)	4(+)
	3(*)	5(*)	2(*)	4(*)	4(*)	4(*)	3(*)	2(*)	4(*)
Throughput (ns)	300	300	400	350	400	400	300	250	450

The experimental results obtained under the tight constraints are reported in Table 4.4, and illustrated in Figure 4.12. Column 2 is the energy consumption of single-voltage single-frequency design. Columns 3 to 5 list the results of MoVoF. Column 3 is the number of clocks synthesized. Column 4 is the energy consumption. Column 5 is the energy reduction over Column 2. Column 6 is the computation time with MuVoF implemented in Java and running on a 2.4GHz Pentium PC with 256MBytes of RAM.

Table 4.4 shows the effectiveness of MuVoF even under tight constraints. Averagely, energy is reduced by 23.5%. Note that the number of clocks synthesized is maximally 3. In a real design, multiple-frequency scheme has an overhead in clock generating logic and clock tree routing. Therefore it is advisable to limit the number of frequencies as well as the values. As described in Section 4.3.1.1, this is realized by defining the frequency set used in clock period extension. In the experiments, the frequency set allows frequencies that are multiples of ten, starting from 30ns. The maximum allowed number of synthesized frequencies is 5.

Table 4.4 Energy Optimization under Relatively Tight Constraints for MuVoF

Benchmark	SVSF (pJ)	MuVoF			Run Time (s)
		# of Clocks	Energy (pJ)	$\Delta(\%)$	
IIR	20274	3	18457	9.0	1.34
FIR	25362	3	11610	54.2	5.88
EWF	19098	3	13668	28.4	5.95
Lattice	31284	2	23516	24.8	2.39
DCT	41118	3	30822	25.0	12.00
FDCT	36714	2	28671	21.9	10.31

TFIR	24792	2	22993	7.3	2.17
DiffEq	13497	2	11052	18.1	0.28
Elliptical	49584	3	39429	20.4	13.29
Average				23.5

Figure 4.12 Energy Reductions under Relatively Tight Constraints for MuVoF

4.4.2 Experiments under Loose Constraints

As a CMOS gate occupies less area and has shorter delay in modern DSM technologies, area and speed become less critical in many applications. On the other hand, with the increasing complexity (size and frequency) of portable devices, energy consumption is a major concern in many systems to lengthen battery life. Therefore situations emerge in which area and speed are compromised for energy optimization. In this experiment, relatively loose constraints are applied on resource usage and throughput, to study the performance of the multi-frequency and multi-voltage functionally pipelined technique. To be realistic, the constraints reported in Table 4.3 are loosened by a small amount (at most one more FU for each FU type and throughput is loosened by 50ns) and the new constraints are listed in Table 4.5. The experimental results obtained are shown in Table 4.6, also illustrated in Figure 4.13.

Table 4.5 Relatively Loose Constraints for MuVoF

Benchmark	IIR	FIR	EWF	Lattice	DCT	FDCT	TFIR	DiffEq	Elliptical
FU	2(+)	6(+)	6(+)	2(+)	6(+)	6(+)	4(+)	2(+)	5(+)
	4(*)	6(*)	2(*)	5(*)	5(*)	5(*)	4(*)	3(*)	5(*)
Throughput (ns)	350	350	450	400	450	450	350	300	500

Table 4.6 Energy Optimization under Relatively Loose Constraints for MuVoF

Benchmark	SVSF (pJ)	MuVoF			Run Time (s)
		# of Clocks	Energy (pJ)	Δ (%)	
IIR	20274	2	8407	58.5	1.74
FIR	25362	2	8936	64.8	6.21
EWF	19098	2	8423	55.9	6.13
Lattice	31284	3	11458	63.4	2.96
DCT	41118	2	17924	56.4	12.22
FDCT	36714	3	15836	56.9	10.17
TFIR	24792	2	10998	55.6	2.69
DiffEq	13497	2	5833	56.8	0.48
Elliptical	49584	3	20649	58.4	13.63
Average				58.6

Figure 4.13 Energy Reductions under Relatively Loose Constraints for MuVoF

An average energy reduction of 58.6 is observed. The maximum number of clocks is still 3.

Comparing Table 4.4 to Table 4.6, it is observed that the relaxation of resource and throughput constraints results in significant energy reduction. With only a small amount of increase in resource and throughput, the energy reduction improves from 23.5% to 58.6%. This observation enables an easy way to explore design alternatives. For example, initially tight constraints can be set on resources and throughput. If the energy is unacceptable, constraints can be relaxed to obtain more energy-efficient designs. As a result, energy can be reduced more significantly, at the cost of a little more resources and/or a little slower design. It is also noted that when constraints are relaxed, the number of synthesized clocks does not tend to increase. This is because the relaxation of resource constraints is not significant, which in turn imposes a limit on how many pipelining stages can be used (as different stages cannot share resources). As a result, although the maximum allowed number of synthesized frequencies is 5, in both tight-constraint and loose-constraint experiments, the maximum number is 3. This indicates that the overhead introduced by multi-frequency scheme is controllable and not significant.

4.4.3 Influence of Resource Constraints and Throughput Constraints

It is known that both resource constraints and throughput constraints have substantial impact on energy consumption, and that relaxation of resource and throughput constraints can produce better optimized results. In this section, the influence of resource constraints and throughput constraints is examined separately.

4.4.3.1 Influence of Resource Constraints

In this subsection, the influence of resource constraints is studied. Experiments were carried out on different resource constraints while the throughput (TP) constraint was fixed, as shown in Table 4.7. The first column is the list of benchmarks. The second column is the fixed throughput constraint. Column 3 to Column 8 are the various resource constraints, labeled as A to F. The energy consumptions are reported in Table 4.8, also illustrated in Figure 4.14.

Table 4.7 Various Resources Constraints for MuVoF

Benchmark	TP	A	B	C	D	E	F
IIR	350ns	1(+)3(*)	2(+)3(*)	3(+)3(*)	2(+)4(*)	3(+)4(*)	2(+)5(*)
FIR	350ns	5(+)4(*)	5(+)5(*)	5(+)6(*)	6(+)6(*)	6(+)7(*)	7(+)7(*)
EWF	450ns	4(+)2(*)	5(+)2(*)	6(+)2(*)	6(+)3(*)	7(+)3(*)	8(+)4(*)
Lattice	400ns	2(+)3(*)	2(+)4(*)	2(+)5(*)	3(+)5(*)	3(+)6(*)	4(+)6(*)
DCT	450ns	5(+)4(*)	6(+)4(*)	6(+)5(*)	7(+)5(*)	7(+)6(*)	8(+)6(*)
FDCT	450ns	5(+)4(*)	6(+)4(*)	6(+)5(*)	7(+)5(*)	7(+)6(*)	8(+)6(*)
TFIR	350ns	3(+)2(*)	3(+)3(*)	4(+)3(*)	4(+)4(*)	5(+)4(*)	5(+)5(*)
DiffEq	300ns	2(+)2(*)	2(+)3(*)	3(+)3(*)	2(+)4(*)	3(+)4(*)	4(+)4(*)
Elliptical	500ns	4(+)4(*)	4(+)5(*)	5(+)5(*)	6(+)5(*)	6(+)6(*)	7(+)6(*)

Table 4.8 Influence of Resource Constraints on Energy (pJ) for MuVoF

Benchmark	A	B	C	D	E	F
IIR	18634	18405	18405	8407	8407	5103
FIR	11025	10526	10218	8936	8936	7201
EWF	13856	13668	8423	6791	6483	4831
Lattice	26257	13822	11458	10997	10997	10012
DCT	30667	30694	17924	17924	16523	16074
FDCT	22583	16157	15836	15674	14215	14057
TFIR	22886	22993	22740	10998	10998	8637
DiffEq	6097	5833	5833	5121	4563	4563
Elliptical	32014	20802	20649	20598	20598	20598

Figure 4.14 Energy Reductions under Different Resource Constraints for MuVoF

Table 4.8 (Figure 4.14) clearly shows how the energy decreases as the resource constraints are loosen. For example, the energy consumption of IIR decreases from 18634pJ to 5103pJ when the resource constraint loosens from 1(+)-3(*) to 2(+)-5(*). The proposed MuVoF technique is capable of achieving substantial energy reduction when resource constraints are allowed to be relaxed.

It is also shown that adding extra resources of different types may lead to considerably different results. Take IIR as an example and suppose the current resource constraint (Case B) is $2(+)3(*)$, the results are different if different additional resource is allowed. If an extra ALU is added (Case C), no improvement is resulted. However, if an extra Multiplier is added (Case D), substantial reduction in energy (from 18405pJ to 8407) are achieved. This nonlinearity usually occurs when the resource constraints are tight, and will disappear at some point during the process of loosening resource constraints. This is because when the resource constraints are tight, the voltages of very few resources can be scaled down, and when adding a resource leads to a successful voltage-scaling move, the energy can be reduced a lot, resulting in a sharp decreasing curve. On the other hand, when resource constraints are loose, the voltages of many resources have already been scaled down.

The benefit of loose resource constraints relies on both the stage partitioning process and the voltage scaling process. For stage partitioning, when more resources are available, a DFG tends to be partitioned into more pipeline stages, which gives more space for clock period extension and thus voltage scaling. For voltage scaling, there are situations in which if an operation is assigned to a low-voltage resource, the throughput constraint is maintained but the resource constraint is violated. In these cases, allocating additional low-voltage resources for these operations can produce additional energy saving because there are more low-voltage operations. However, it should be noted that energy cannot be infinitely reduced by relaxing resource constraints because there are limited number of operations in a DFG. For example, in an extreme case, when the number of resources is equal to the number of operations, there is no need to allocate more resources.

4.4.3.2 Influence of Throughput Constraints

To study the influence of throughput, the resource constraint was fixed and experiments were carried out under different throughput constraints, as shown in Table 4.9. Column 1 is the list of benchmarks. Column 2 is the fixed resource constraint. Column 3 to Column 7 are the throughput constraints. The energy consumptions under various throughput constraints are reported in Table 4.10, also illustrated in Figure 4.15.

Table 4.9 Various Throughput Constraints (ns) for MuVoF

Benchmark		A	B	C	D	E
IIR	2(+) ³ (*)	250	300	350	400	450
FIR	5(+) ⁵ (*)	250	300	350	400	450
EWF	5(+) ² (*)	350	400	450	500	550
Lattice	2(+) ⁴ (*)	300	350	400	450	500
DCT	6(+) ⁴ (*)	350	400	450	500	550
FDCT	6(+) ⁴ (*)	350	400	450	500	550
TFIR	3(+) ³ (*)	250	300	350	400	450
DiffEq	2(+) ² (*)	200	250	300	350	400
Elliptical	4(+) ⁴ (*)	400	450	500	550	600

Table 4.10 Influence of Throughput Constraints on Energy (pJ) for MuVoF

Benchmark	A	B	C	D	E
IIR	18639	18457	18405	8503	8503
FIR	21417	11610	10526	9775	8963
EWF	19013	13668	13668	8411	8309
Lattice	23741	23516	13822	12920	12001
DCT	40917	30822	30694	17936	17741
FDCT	28711	28671	16157	15806	15732
TFIR	23017	22993	22993	10896	10735
DiffEq	13623	11052	5833	5833	5098
Elliptical	41400	39429	32014	31558	21681

Figure 4.15 Energy Reductions under Different Throughput Constraints for MuVoF

Table 4.10 (Figure 4.15) shows how energy decreases when throughput constraint is relaxed. As can be seen from the figure, the reduction is extremely nonlinear and difficult to predict, because it strongly depends on the structure of the DFG. Take IIR as an example, loosening throughput constraint from 1/300ns to 1/350ns does not improve the results much, but loosening it from 1/350ns to 1/400ns yields great improvement. As in the case of resource constraints relaxation process, when throughput constraints are very loose, the voltages of most resources have already been scaled down, and the curves become more and more linear.

The benefit of loosening throughput constraints mainly relies on the clock period extension and voltage scaling process, although it does influence the stage partitioning process. Intuitively, clock period can be extended if throughput constraint is loosened. As stated before, extended clock period creates more space for voltage scaling, which reduces energy. However, relaxing throughput constraints directly degrade the circuit performance; therefore usually throughput constraints cannot be compromised excessively.

4.4.4 Comparison

The above experimental results perform comparisons with energy-unconscious synthesis techniques. As this is the first attempt on energy optimization multi-voltage multi-frequency functionally pipelined datapath, a direct and fair comparison with existing works is not achievable. Besides, as the authors of [MOH05] pointed out, it is difficult to achieve a fair comparison, because different schemes have different objectives, constraints and overheads. In this work, the target is to explore the design space in a functionally pipelined datapath using multi-voltage multi-frequency techniques. As the datapath is functionally pipelined, more resources must be used to achieve a much faster throughput, compared to non-pipelined structures. Therefore in the following comparison, resource usage and speed are not listed, because MuVoF typically uses more resources and provides faster throughput. The justification is that in modern Deep Sub-Micro technology, area occupied by FUs is usually not a major concern; while speed and energy consumption is a much more important optimization target in many applications, especially in portable and mobile applications. The purpose of the comparison is to provide a general idea of the performance of MuVoF, not to directly compare with other published techniques.

4.4.4.1 Average Improvement Comparison

Table 4.11 lists the average energy reduction comparison between MuVoF under both tight and loose constraints and other published techniques. While the constraints are loose, the performance of MuVoF surpasses all the others. When the constraints are tight, MuVoF still achieves a satisfactory 23.5% energy reduction, but less than the others.

Table 4.11 Average Improvement (%) Comparisons for MuVoF

	[CHA97]	[MOH04a]	[MOH05]	[MAN02]	[PAP99]	[CHE06]	Tight	Loose
Energy Δ(%)	40.19	43.29	23.3	up to 50	39	up to 46.7	23.5	58.6

As stated before, typically MuVoF requires more resources, but produces faster throughput due to the feature of functionally pipelined datapath. One of the most distinct advantages of MuVoF is that, when more resources are available, it can easily utilize

them and generates a better design in terms of energy consumption. This feature provides an easy and effective way to explore the design space, and is especially useful when energy consumption is a much more important concern than circuit size.

4.4.4.2 Benchmark Comparison

An energy improvement comparison over some benchmarks was also performed and the results are reported in Table 4.12. It can be observed that under tight constraints, MuVoF is better than other techniques over some benchmarks, but worse in the others. While under loose constraints, MuVoF performs better. Although it is not feasible to perform a direct and fair comparison and it is impossible to define the “tightness” of constraints over different techniques, the comparison shows that MuVoF is effective, especially under relatively loose constraints, and provides an easy and effective method to explore design space for energy reduction.

Table 4.12 Benchmark Improvement (%) Comparisons for MuVoF

Benchmark	[CHA97]	[MOH04a]	[MOH05]	[MAN02]	Tight	Loose
IIR	-	-	24.7	-	9.0	58.5
FIR	-	41.55	18.0	33.2	54.2	64.8
EWf	44.47	43.01	-	44.2	28.4	55.9
DCT	-	43.58	-	30.4	25.0	56.4
FDCT	43.44	-	-	-	21.9	56.9
DiffEq	40.87	-	-	53	18.1	58.4

4.5 Summary

Research in low energy high level synthesis has been focusing on parameters like switching activity, supply voltage and operating frequency. As the precise switching activity is difficult to establish at the behavioral level and the switching activity reduction has its own bounds, the voltage and frequency scaling techniques are gaining popularity recently. The advantage of these techniques is that the tradeoff between the energy

consumption and the traditional optimization parameters such as area and speed can be easily accomplished, and results depend less on the structure of the DFG.

In this chapter, MuVoF, a multi-voltage multi-frequency behavioral synthesis technique for functionally pipelined datapath is proposed. Given a DFG, resource constraint and throughput constraint, the technique consists of two levels. The core process finds an optimal pipeline stage partitioning and performs clock period extension and multi-voltage assignment for each pipeline stage. The iterative process uses a simulated-annealing approach and performs several types of local moves to iteratively refine the final results. The results obtained from the experiments show that the multi-frequency multi-voltage synthesis scheme is very effective in reducing energy consumption for a functionally pipelined datapath. It is also shown that both the resource and throughput constraints have a major influence on the amount of energy that can be reduced. It is observed that if area and speed are not the primary concerns, energy can drastically be reduced at the cost of larger area and slower speed.

Chapter 5 Floorplan-Driven Low Energy High Level Synthesis for Multi-Voltage Datapath

5.1 Introduction

The advent of Deep Submicron (DSM) technology has inspired many new research topics in high level synthesis. This is because in this new semiconductor fabrication technology, interconnects, specifically connecting wires, dominate many design parameters such as area, delay as well as power/energy consumption. On the other hand, conventional synthesis flow in IC design is a top-down flow, with high level synthesis preceding physical synthesis. Therefore in the high level synthesis tasks, little information of physical level is available. This leaves a gap between the conventional synthesis flow and the DSM technology, making optimal results in the high level more likely to fail in the physical level and requiring more backtracking and iteration in the design flow.

As stated in Chapter 2, many research works [PRA99, STA03, DAV05] have been shifted to the integration of high level synthesis and physical synthesis to close this gap. Currently, integration of floorplan information into high level synthesis is a popular approach due to relatively low increase in computational complexity.

In this chapter, a technique named FloM (Floorplan-driven Multi-voltage synthesis) is proposed. This technique targets a multi-voltage datapath because it is very effective in power reduction due to the quadratic dependency of power on supply voltage. Many multi-voltage synthesis techniques [JOH97, CHA97, MAN02, MOH05] have been proposed, including the MuVoF technique proposed in Chapter 4. In this chapter, a new multi-voltage synthesis technique is proposed using simulated annealing algorithm that can be seamlessly integrated into the proposed floorplan-driven synthesis system.

It is well-known that floorplan can be classified into two categories, slicing [STO83, TAN06] and non-slicing [GUO01]. The slicing structure is a rectangle dissection that can

be obtained by recursively cutting a rectangle vertically or horizontally. [WON86] proposed a flexible floorplanning algorithm for slicing-based floorplanning that has many variations [CHA95a, FAN94, PRA99, STA03] in the literature to handle different problems. The proposed technique also uses a modified approach of the algorithm to optimize the wire energy consumption under multi-voltage environment.

This chapter is organized as follow. Section 5.2 discusses the influence of floorplanning on power optimization of multi-voltage datapath. Section 5.3 presents the proposed floorplan-driven technique in details. Experimental results and comparisons are presented in Section 5.4, which is followed by a summary in Section 5.5.

5.2 Interconnect Power Optimization and Multi-voltage Design

In modern DSM technology, interconnects, such as wire, play a more and more important role in area, performance, as well as power consumption. As interconnects consume a considerable amount of the total power, it now becomes a very important design dimension. In terms of high level synthesis, interconnect information must be included in the synthesis phase to achieve an accurate power estimation and thus, a better optimization result.

5.2.1 Impact of Floorplanning on Power

Floorplanning is a very important step in physical synthesis. It determines the relative positions of all blocks, including FUs, registers, etc. Although a floorplan of a circuit does not provide detailed physical information, it guides lower-level physical synthesis steps, such as placement, routing and layout, and thus has a major impact on the final physical implementation. Consider the datapath example, which consists of 1 multiplier, 2 ALUs, and 5 registers, as shown in Figure 5.1. A feasible floorplan is shown in Figure 5.2.

Figure 5.1 A Datapath Example

Figure 5.2 A Floorplan for Figure 5.1

The floorplan provides a more accurate estimation of the circuit area, which is the rectangular area occupied by the layout, rather than the area summation of all the resources. This fact makes floorplanning very important for area optimization, because floorplan determines the relative positions of resources, and thus influences the connections among resources. Consider another floorplan shown in Figure 5.3.

Figure 5.3 A Floorplan for Figure 5.1 that Takes More Space

Obviously the floorplan in Figure 5.3 occupies much more space than the floorplan of Figure 5.2 does, because more space is wasted due to bad positioning. As floorplan plays a critical role in physical synthesis and is crucial for placement, routing and layout, a large number of algorithms have been proposed for floorplan optimization in the physical synthesis literature. These algorithms mainly target area optimization, trying to group resources together as compact as possible [WON86, GER99]. There also exist some algorithms which consider some other parameters, such as the total length of connecting wires [GUO01, TAN06].

The fact that interconnections play an unprecedented important part in modern DSM technology imposes more stringent requirements on the optimization of interconnects in the floorplanning process. This has impact on not only the area, as discussed above, but also the power and energy.

As stated in Chapter 2, the power of wire can be modeled as

$$P_{wire} = \alpha \cdot c \cdot f \cdot V^2 \cdot \sum L_i \quad (5.1)$$

In which α is the average SA occurred on the wire, V the voltage of the signal that flows through the wire, f is the operating frequency, c is the wire capacitance per unit, and $\sum L_i$ is the total length. Furthermore, in the floorplanning stage, detailed routing and layout information are not applicable, thus wire length can be modeled as

$$\sum L_i = \sum_{i=1}^n \sum_{j=1}^n N_{i,j} \cdot M_{i,j} \cdot \text{distance}(p_i, p_j) \quad (5.2)$$

Where $N_{i,j}$ is the number of connections from the outputs of resource i to the inputs of resource j , $M_{i,j}$ is the bit width, and $\text{distance}(p_i, p_j)$ is the distance of the central points of the two resources.

5.2.2 Multi-voltage Design

The multi-voltage design has been discussed in Section 2.4. In the high level synthesis literature, power optimization is usually performed under a resource constraint in which every resource has a pre-assigned supply voltage. In Chapter 4, a multi-voltage assignment algorithm is proposed, which does not pre-define the voltage of each resource but intelligently assign the optimal supply voltage to each resource. In this chapter, the same approach that the supply voltages of sources are determined on-the-fly is adopted. An example of a multi-voltage datapath is illustrated in Figure 5.4.

Figure 5.4 A Multi-Voltage Datapath

In this datapath, there are two functional units (*FU1* and *FU2*). *FU1* is supplied with 5.0V and *FU2* with 3.3V. The register (*Reg1/Reg2*) that stores the output of a functional unit (*FU1/FU2*) is supplied with the same voltage as the functional unit so that they can have close connection. A functional unit can take input signals with voltage equal to or higher than the supply voltage of itself. For example, *FU2* (3.3V) takes a 5.0V signal (output of *Reg1*) as one input and a 3.3V signal (output of the level converter *LV2*) as the other. This is the same for *MUX* (2.4V) taking a 2.4V signal as one input and a 3.3V signal as the other. However, whenever a lower voltage signal drives a higher voltage signal, a level converter (*LV1/LV2*) must be adopted. Therefore the output of *MUX* (2.4V) cannot be connected directly to the input of *FU1* (5.0V). Instead, a 2.4V to 5.0V level converter (*LV1*) must be placed between them to convert the 2.4V signal to the 5.0V signal that can go directly into *FU1*. *LV2* (2.4V to 3.3V) also functions as a converter for the 2.4V input signal to 3.3V signal. In such cases, level converters play a very important role in multi-voltage designs. Generally it converts a lower-level signal into a higher-level signal so that the signal can act as a driver to a module. Otherwise the logic may not be functional. There are obviously two parameters to be considered in a level converter: the input signal level and the output signal level. Note that the power consumption of a level converter can be considerably large and may vary a lot depending on the two

parameters. A Differential Cascade Voltage Switch (DCVS) is illustrated as Figure 2.2 in Section 2.4.

In this chapter, a power estimator based on Equation (5.3) is developed to evaluate the power cost during synthesis, which considers the power consumed by registers, MUXes, level converters, wires, as well as modules.

$$P_{total} = P_{module} + P_{reg} + P_{wire} + P_{level_converter} + P_{MUX} \quad (5.3)$$

Where

$$P_{module} = \sum COEFF_{module,i} \cdot V_i^2 \quad (5.4)$$

$$P_{reg} = \sum COEFF_{reg,i} \cdot V_i^2 \quad (5.5)$$

$$P_{wire} = \sum COEFF_{wire,i} \cdot L_i \cdot V_i^2 \quad (5.6)$$

$$P_{level_converter} = \sum P_{level_converter,i} \quad (5.7)$$

$$P_{MUX} = \sum COEFF_{MUX,i} \cdot V_i^2 \quad (5.8)$$

The COEFF is the product of switching activity (α), capacitance (c), and frequency (f). Again assuming that the capacitance of a wire does not change with its voltage level, (5.6) is rewritten as (5.9).

$$P_{wire} = COEFF_{wire} \cdot \sum L_i \cdot V_i^2 \quad (5.9)$$

From Equation (5.9), it can be concluded that the optimization of power on interconnections is no longer the same as the optimization of total wire length in a multi-voltage design, because different wires can carry signals with different voltage.

5.2.3 Floorplan Optimization for Multi-voltage Design

In a multi-voltage design, the optimization of wire power is more difficult than the optimization of total wire length. This is due to the fact that the supply voltage of a resource is not fixed, but is determined by the behavioral synthesis algorithm. The scheme of determining voltage supply on-the-fly creates a dilemma for both the behavioral synthesis algorithm and the floorplan algorithm. On one hand, in the behavioral synthesis process, the detailed floorplan information is not available, so the power consumed by wire is impossible to estimate. On the other hand, for the floorplan algorithm, the actual supply voltage of a resource is not determined and the wire power cannot be calculated with (5.9), because V_i is unknown. Therefore a single sequential invocation of the behavioral synthesis algorithm and the floorplan algorithm may not achieve an optimal result even if the behavioral synthesis algorithm and the floorplan algorithm independently achieve the optimal results. Based on this observation, an integrated algorithm is proposed in the next section to address this dilemma, using an iterative approach.

On top of the difficulty of integration of the behavioral synthesis process, the multi-voltage model itself complicates the floorplan process in another manner. As stated in Chapter 4, multi-voltage design imposes a challenge on the circuit layout. It is desirable to partition the circuit into several regions, each operating at their own voltages. Some circuit isolation is necessary to mitigate the crosstalk of those regions, by increased use of substrate contacts and slightly increased spacing between wells. This makes existing design tools hard to incorporate, thus considerably increases the work of layout design. Therefore in real world, multi-voltage design is not an easy task. A real-world multi-voltage media processor is presented in [USA98], in which two supply voltages are used. It is shown that considerable power consumption has been saved.

Suppose there are two behavioral synthesis solutions, A and B. In the behavioral synthesis process, only the power consumed by resources (modules, registers, MUXes, and level converters) can be estimated. Suppose this cost is called as behavioral synthesis

cost, and $behavioral_synthesis_cost(A) < behavioral_synthesis_cost(B)$. Since A and B can lead to totally different floorplan and thus wire connection, it is possible that $wire_cost(A) > wire_cost(B)$. Note that:

$$overall_cost = behavioral_synthesis_cost + wire_cost \quad (5.10)$$

Conclusively, an optimal solution in terms of behavioral synthesis does not necessarily mean an optimal solution in terms of overall synthesis (behavioral synthesis + floorplan synthesis) result. An optimization algorithm must be able to integrate the two processes in one way or another, so that the overall cost is estimated relatively accurately during the synthesis process.

5.3 Algorithm Formulation of FloM

An iterative algorithm, FloM, is proposed for the integration of behavioral synthesis and floorplan synthesis. It also takes into account of multi-voltage design for datapath power optimization.

5.3.1 Algorithm Flow

Basically FloM iteratively refines the behavioral synthesis result and the floorplan result so that the overall power consumption is minimized. The algorithm flow is given in Figure 5.5.

Figure 5.5 Overall Algorithm Flow of FloM

The inputs are a DFG, library components characterization, and various constraints, such as resource constraint, delay constraint (in real time unit), clock cycle time and the voltage domain (supply voltage levels that are allowed). After an initial scheduling and binding is performed, the algorithm goes into an outer-level simulated annealing algorithm. In each iteration of the simulated annealing algorithm, a floorplanner, a power/energy estimator, and a local move engine are performed in sequence, until an acceptable synthesis result is obtained.

5.3.2 Outer-Level Iterative Algorithm

The outer-level algorithm is a major part of the complete algorithm flow. It takes an initial scheduling and binding, and produces an optimized result for scheduling, binding, as well as floorplanning, in terms of total power/energy consumption, including modules, registers, MUXes, level converters and wires. This simulated-annealing-based iterative algorithm is given as Algorithm 5.1.

// Algorithm 5.1: Outer-Level Iterative Algorithm

```

1:  $s$  = initial solution;
2:  $T = cost(s)$ ;
3:  $stop = false$ ;
4: Level1: while ( $!stop$ ) {
5:    $total\_moves = 0$ ;
6:    $accepted\_moves = 0$ ;
7:   Level2: while ( $accepted\_moves < N \ \&\& \ total\_moves < 2 * N$ ) {
8:      $s' = local\_move(s)$ ;
9:      $total\_moves++$ ;
10:     $accepted\_moves++$ ;
11:     $\Delta c = cost(s') - cost(s)$ ;
12:    if ( $\Delta c \leq 0$ )  $s = s'$ ;
13:    else if ( $e^{-\Delta c/T} > random(1)$ )  $s = s'$ ;
14:    else  $accepted\_moves--$ ;
15:  }
16:  if ( $accepted\_moves < total\_moves * 5\%$  ||  $T < T\_threshold$ )  $stop = true$ ;
17:   $T = r * T$ ; //  $r = 0.84$  in current implementation
18: }
```

A solution is defined as a composite of scheduling, binding and floorplan result. The function *cost* (*s*) in Line 11 estimates the full power consumption cost of a solution *s*. Inside this function, an optimized floorplan is obtained in order to do the estimation. This floorplan is obtained by the inner level algorithm, which is described in Section 5.3.4. The function *local_move* (*s*) in Line 8 performs a local move on the solution, as described in the following section. Good local moves allow the transition from solution *s*₁ to solution *s*₂ within a finite number of moves.

5.3.3 Local Moves in the Outer-Level Algorithm

A local move is defined here as a move to change the voltage assignment of a resource, to change the resource binding, or to change the operation scheduling. Whenever a local move is performed, an optimized floorplan is generated. Therefore, a local move has the effect of changing the scheduling, binding, voltage assignment, as well as the floorplanning (thus a solution, as defined in the previous section). The following five move types are adopted in the outer-level algorithm.

(1) Voltage Scaling

This move changes the supply voltage of a functional unit, FU, from v_0 to v_1 . If $v_0 < v_1$, the timing will certainly still be met, because the latency of the resource is decreased under the new higher voltage supply. Therefore rescheduling is not necessary. If $v_0 > v_1$, rescheduling is performed on operations executed on FU and all their successors. A valid Voltage Scaling move must not incur violation on either the latency constraint or the resource constraint. Consider the DFG of an IIR filter as shown in Figure 5.6. This figure is redrawn from Figure 4.3, and shows the scheduling under 5.0V. The following describes how voltage scaling affects the scheduling.

Figure 5.6 DFG of an IIR Filter

Suppose operations 20 and 28 are executed on resource *R1*. If the voltage of *R1* is scaled up from 2.4V to 3.3V, then the latency of *R1* is reduced, so the time taken to complete operations 20 and 28 is reduced, and thus timing will not be violated. Note that this move needs reallocation and rebinding of registers. This is because the voltage of all the registers that stores the value of *R1* must change accordingly, resulting in possible register sharing conflict. Suppose a register *REG1* (2.4V) stores the both the data of *R1* (2.4V) and *R2* (2.4V). If the voltage of *R1* is scaled to 3.3V, *REG1* can no longer be

shared by $R1$ and $R2$, because their voltages are different now. The number of registers needed may also change, and thus a register reallocation process must be performed. Scaling up the voltage of a resource will generally increase the power, but as it will change other power factors, such as registers, MUXes, level converters and floorplan, the overall power consumption may decrease. Even if the overall power consumption increases, negative moves are allowed in simulated annealing algorithm in order to introduce new positive moves.

On the other hand, if the voltage of a resource is scaled down, timing violation may occur. Now it is supposed that the voltage of $R1$ is scaled from 3.3V to 2.4V, the latency of the all the operations (say, operations 20 and 28) is increased. Suppose now the latency spans three clock cycles, then there would be a timing violation on operation 28, because its operating time overlaps with its successor, operation 29. Note that a resource violation also takes place in operation 20, because it now operates as late as in control step 7, making three multiplications (20, 13, 18) running in the same clock cycle. This would need three multipliers (possibly with different voltages), breaking the resource constraint, because originally only two multipliers are needed. Therefore, both timing violation and resource violation are possible when the voltage of a resource is scaled down. When a violation does occur, resource-constrained rescheduling must be performed to balance the operation distribution. If it fails, the move is considered as an invalid move and is discarded. There are two main steps of the rescheduling.

- (i) Update the delay of all the operations, N , on which the Voltage Scaling move is performed, and sort the operations in ascending order of their start time.
- (ii) Sequentially, move each n_i in N upwards as much as possible, so that operation dependency is maintained, and the start time of n_i is larger than the end time of n_{i-1} . If the new end time of n_i is larger than one of its successors, all the operations whose start time is larger than the end time of n_i are pushed downwards to satisfy the dependency constraints.

(2) FU Rebinding

This move rebinds a single operation from resource $R1$ to another compatible resource $R2$ whose supply voltage is different from that of $R1$. Similarly, if the voltage of $R1$ is lower than the voltage of $R2$, no timing violation will occur, because the speed of the operation is higher on the new resource. Therefore only reallocation and rebinding of registers are needed. Otherwise, rescheduling may be needed to resolve any timing violation. Note that an operation cannot be rebound to a resource that is operating within the life time of the operation, otherwise resource conflict occurs. When this condition is satisfied, resource violation will not happen. This move also requires reallocation of registers.

(3) FU Swap

This move swaps the bound FUs ($FU1$ and $FU2$) of two operations ($OP1$ and $OP2$). A valid FU swap move must not incur any rescheduling move, which means that the two FUs must have the same supply voltage and there is no operation lifetime conflict after the swap. The operation lifetime conflict occurs in a situation like this: suppose an operation $OP3$, which is bound to $FU2$, operates in the same control steps as $OP1$. After the swap, $OP1$ would also be executing on $FU2$, so now $OP1$ and $OP3$ are bound to the same resource ($FU2$) and operate in the same control steps, resulting in a resource violation.

(4) Register Rebinding

This move rebinds an edge E from register $REG1$ to another register $REG2$ with the same supply voltage. No rescheduling is required in this move. A valid register rebinding move must incur no violation on variable lifetime. The variable lifetime violation occurs in a situation like this: after reassigning E to $REG2$, there is another edge $E1$ which is also bound to $REG2$, and the lifetimes of E and $E1$ overlap.

(5) Register Swap

This move swaps the bound registers (*REG1* and *REG2*) of two variables (*E1* and *E2*). The two registers must have the same supply voltage, and no variable lifetime conflict should occur after the swap. The variable lifetime conflict happens when after the swap, the lifetime of *E1* overlaps with the lifetime of an edge that is assigned to *REG2*, or the lifetime of *E2* overlaps with the lifetime of an edge that is assigned to *REG1*.

In each iteration, the five local moves are randomly selected in the function *local_move* () to generate a new solution. A set is defined to keep the FUs/registers that are selected to perform a move in an iteration. In an iteration, the FUs/registers in the set have lower priority to be selected to perform a move, and the set is updated with the newly selected FUs/registers. The local moves work together to explore synthesis alternatives to relieve local effects. These moves include the changes of resource voltages, FU binding and register binding, which also result in changes in interconnects, including MUXes and wires so that better solutions can be found for power/energy optimization. The experimental results show that they are adequate to achieve considerable optimization results.

5.3.4 Inner-level Floorplanning Algorithm for Multi-Voltage Design

Whenever a synthesis solution is found, a floorplanner generates an optimized floorplan for the multi-voltage datapath. The floorplanner partitions the datapath into different regions, each of which is occupied by floorplan blocks with the same voltage. Floorplan blocks include FUs, registers and MUXes. The target is to optimize the power consumed by the wires. The power consumed by FUs, registers, MUXes and level converters is not considered here, because it is determined not by the floorplan, but by the synthesis result.

The floorplanning algorithm is based on a modified simulated annealing algorithm that is first proposed in [WON86]. The modifications are: (1) The proposed algorithm intends to optimize the total power consumed by the wires, while the original algorithm intends to optimize area and the total wire length. These are different optimization targets according

to (5.6), because different wires could carry signals of different voltages on multi-voltage design. (2) The proposed algorithm partitions the datapath into multiple regions, each of which consists of resources of the same voltage, while the original algorithm is applied to the floorplan with single voltage supply.

As stated before, a floorplan can be categorized as a slicing floorplan or a non-slicing floorplan. Slicing structures have several advantages [OTT82] and thus been widely explored. One of the most popular representations of slicing floorplan is normalized Polish expression, first proposed in [WON86] and briefly summarized below.

Definition 5.1: A Slicing Tree is an oriented rooted binary tree to represent the hierarchical structure of a slicing structure, in which every internal node is labeled as either “*” or “+”, corresponding to either a vertical or a horizontal cut of the rectangle.

A slicing floorplan and its corresponding slicing tree are shown in Figure 5.7.

Figure 5.7 A Slicing Floorplan and Its Slicing Tree

Definition 5.2: A balloting sequence is a binary sequence $b_1b_2...b_m$ which satisfies that for any $k \in [1, m]$, the number of the 0's in $b_1b_2...b_k$ is less than the number of the 1's in $b_1b_2...b_k$.

Definition 5.3: Let σ be a function defined as (5.11).

$$\sigma(x) = \begin{cases} 1, x \in [1, n] \\ 0, x \in \{*, +\} \end{cases} \quad (5.11)$$

Where $*$ and $+$ are operators and $1, 2, \dots, n$ are operands. A sequence $\alpha_1 \alpha_2 \dots \alpha_{2n-1}$ is a Polish Expression of length $2n - 1$ iff every $i \in [1, n]$ appears exactly once in the sequence, and $\sigma(\alpha_1) \sigma(\alpha_2) \dots \sigma(\alpha_{2n-1})$ is a balloting sequence. A Normalized Polish Expression is a Polish Expression with no consecutive $*$'s or $+$'s in the sequence.

In [WON86], it is shown that a slicing floorplan can be represented by a unique skewed slicing tree (a type of slicing tree in which no node and its right son has the same label in $\{*, +\}$) by performing the cuts always from right to left and from top to bottom, and by a unique normalized Polish expression.

Note that the slicing tree in Figure 5.7 is not suitable for normalized Polish expression representation because it is not a skew tree. A skewed slicing tree representation of the floorplan is given in Figure 5.8.

Figure 5.8 A Floorplan and Its Skewed Slicing Tree Representation

The skewed slicing tree in Figure 5.8 can be represented as a normalized Polish expression: $PE1 = 1\ 6 + 3\ 5 * 2 + * 7\ 4 + *$.

Summarily, slicing tree and Polish expression are two ways to represent a slicing floorplan structure. A skewed slicing tree and a normalized Polish expression give unique representations of a slicing floorplan structure. A normalized Polish expression is very suitable for simulated annealing algorithm because it is unique (therefore eliminates redundancy), its data structure is simple and it provides easy local moves, as explained in the following.

In the proposed Multi-voltage floorplanning, a floorplan consists of several super blocks, each of which consists of blocks with the same voltage. Therefore there are two levels of Polish expressions. The first level has one normalized Polish expression, which represents the floorplan of the super blocks. The second level has several normalized Polish expressions, each of which represents the floorplan of a super block. An optimal floorplan solution regards to the solution yielding the minimal cost for the flattened floorplan.

The floorplanning algorithm also uses the simulated annealing algorithm as in Algorithm 5.1. As there are two levels of Polish expressions, a local move can perform on either the outer level Polish expression or one of the inner level Polish expressions.

Definition 5.4: A chain is a sequence $b_1\ b_2\ \dots\ b_k$, of operators $+$ and $*$, where for $i \in [1, k - 1]$, $b_i \neq b_{i+1}$. A complement of a chain is the chain obtained by interchanging the operators $+$ and $*$. For example, the complement of a chain $++**+$ is $*+*++$.

In the simulated annealing algorithm, three types of local moves are adopted as in [WON86].

- (1) Swap two adjacent operands.

Suppose operands 6 and 3 in $PE1$ are swapped, $PE2 = 1\ 3 + 6\ 5 * 2 + * 7\ 4 + *$ is obtained. This move always produces a normalized Polish expression, because the balloting property is not violated, and the operators do not change.

(2) Complement some chain of nonzero length.

By performing this move on $PE1$, $PE3 = 1\ 6 * 3\ 5 + 2 * + 7\ 4 * +$ is obtained. Note that the full chain must be complemented; otherwise the resulting Polish expression may not be normalized. For example, $PE4 = 1\ 6 * 3\ 5 + 2 * + 7\ 4 * *$, obtained by complementing the chain except the last operator, is not normalized and thus not regarded as a valid move.

(3) Swap two adjacent operand and operator.

By swapping operand 5 and operator $*$ in $PE1$, $PE5 = 1\ 6 + 3 * 5\ 2 + * 7\ 4 + *$ is obtained. Note that this move may produce a non-balloting sequence or a sequence that contains identical consecutive operators. For example, by swapping operand 6 and operator $+$ in $PE1$, the sequence $1 + 6\ 3 * 5\ 2 + * 7\ 4 + *$ is produced, which is not a balloting sequence. Similarly, by swapping operand 3 and operator $+$ of a normalized Polish expression $PE6 = 1\ 2 + 3 + 4 *$, the sequence $1\ 2 ++ 3\ 4 *$ is obtained, which not only violates the balloting property, but also contains two consecutive operator $+$'s.

As this move is performed frequently in the simulated annealing algorithm, an efficient method to check whether this move is valid must be available. Fortunately, it is very obvious to check whether a move produces a sequence that contains identical consecutive operators; and [WON86] gives an efficient method to check whether a move produces a non-balloting sequence. Suppose the number of 0's in $\sigma(\alpha_1)\ \sigma(\alpha_2)\ \dots\sigma(\alpha_k)$ is d_k , and consider a move that swaps α_i and α_{i+1} . Note that only the case when α_i is an operand and α_{i+1} is an operator needs to be considered, because otherwise the swap always generates a balloting sequence, according to Definition 5.3. It can be seen that the swap will not violate the balloting property iff $2d_{i+1} < i$.

The three types of moves are sufficient to go from a normalized Polish expression to another normalized Polish expression. Further details on the Polish Expression can be found in [WON86]. In the proposed multi-voltage floorplanner, these moves could be performed on either the outer level Polish expression or one of the inner level Polish expressions. By randomly selecting these moves, the simulated annealing algorithm optimizes the floorplan for each super block of a specific supply voltage as well as for the whole datapath.

One of the major advantages of iterative algorithms is that the optimization target is easily adjusted by simply changing the cost function. In conventional floorplanning algorithms, the target is to optimize area occupation of the floorplan rectangle and the total wire length. However, in the proposed technique, the target is power/energy optimization. As shown in (5.9), in a multi-voltage design, the floorplan with the shortest wire length does not necessarily means a solution with the minimum power/energy consumption in wire, because different wires can carry signals of different voltages. Therefore, for power/energy optimization, the cost function of the iterative algorithm is the power/energy consumption in (5.9) instead of the total wire length or the area. If power/energy consumption, total wire length and area are all important, a composite cost function can be defined.

5.4 Experimental Results

In this chapter, FloM, a floorplan-driven low-energy high level synthesis technique for multi-voltage design, is proposed. FloM uses two levels of simulated annealing algorithm. The outer level optimizes the total energy consumption, including the energy consumed by FUs, registers, MUXes, level converters and wires. The inner level optimizes the energy specifically for wires by performing the floorplanning. To evaluate the effectiveness of the FloM, experiments have been carried out. The benchmarks and their constraints are listed in Table 5.1. The features of the benchmarks can be found in Section 3.5.

Table 5.1 Benchmarks and Constraints for FloM

Benchmark	# of MULT	# of ALU	Latency
EWF	1	3	25
IIR	2	1	16
DCT	2	3	25
DiffEq	2	1	10
FDCT	2	3	25
FIR	2	2	20
TFIR	2	2	15
Lattice	2	1	20

The delay and energy characterization of FUs, MUX and level converters are reported in Table 4.1 and Table 4.2. For registers, the same ALU to register energy consumption ratio as reported by the Synopsys Design Compiler is used, and thus the energy consumption of a register is 57/3, 25/3 and 13/3 under 5.0V, 3.3V and 2.4V respectively. In the experiments, the clock period is 30ns. The voltages allowed are 5.0V, 3.3V and 2.4V.

5.4.1 Experimental Results on Synthesis

In this section, experiments are carried out to evaluate the multi-voltage low power synthesis algorithm, specifically, the five local moves described in Section 5.3.3. As these moves do not affect the floorplan directly, the floorplanning algorithm (the inner level algorithm) is not invoked in this experiment, and the cost function of the outer level simulated annealing algorithm is defined as the total energy consumed by the datapath, e.g. Multipliers, ALUs, registers, MUXes and level converters. The results are shown in Table 5.2 and illustrated in Figure 5.9.

In Table 5.2, the second column is the energy consumption when list scheduling and single voltage (5.0V) design are used. The third and fourth columns are the energy consumption and energy reduction percentage over the second column when list scheduling and multi-voltage design are used. It adopts the multi-voltage assignment algorithm from Chapter 4. The fifth and sixth columns are the energy consumption and

energy reduction percentage over the second column when the proposed outer level simulated annealing algorithm is used.

Table 5.2 Experimental Results on Datapath Energy Consumption for FloM (pJ)

Benchmark	List Sch + Single Vol Energy	List Sch + Multi-V		FloM	
		Energy	Δ (%)	Energy	Δ (%)
EWF	20525	18723	8.8	13634	33.6
IIR	21077	17656	16.2	13285	37.0
DCT	42834	29123	32.0	23663	44.8
DiffEq	13964	10545	24.5	7852	43.8
FDCT	38169	32821	14.0	21773	43.0
FIR	27045	23774	12.1	9191	66.0
TFIR	25834	21161	18.1	16442	36.4
Lattice	30129	26719	11.3	20637	31.5
Average	-	-	17.8	-	42.4

Figure 5.9 Reductions on Datapath Energy for FloM

It is observed that by applying multi-voltage assignment, an average improvement of 17.8% is achieved. This reduction is purely caused by reducing the voltage supply of some resources (scheduling is not changed). The proposed algorithm refines the solution (including scheduling) iteratively and achieves an average energy reduction of 42.4%. This further reduction is due to the fact scheduling is refined in some of the five local moves, and as different scheduling provides different space for multi-voltage assignment, more design space is explored in the proposed algorithm.

5.4.2 Experimental Results on Synthesis and Floorplanning

In this section, the floorplanning process is taken into consideration in the experiments. For the estimation of floorplan area and wire energy consumption, the area characterization of modules and wire capacitance model must be available. In the literature, there exist many modeling technique for area and capacitance. However, in this work, research has been focusing on the synthesis algorithm. Therefore a simple model is used for standardization. The dimension of resources is defined in “unit”. The wire capacitance is modeled as 0.01 pf per unit. Note that the proposed technique could be easily adjusted to suit new and more accurate models, simply by changing the cost function in the iterative algorithm. That is to say, the proposed technique does not depend on area and power models. The possible dimensions of multipliers, ALUs and registers are shown in Table 5.3.

Table 5.3 Dimensions of Resources (Unit)

Resource	Multiplier	ALU	Register
Dimensions	5*20, 10*10	3*8, 4*6	2*4

Unlike traditional synthesis flow, in which high level synthesis is performed before floorplanning, FloM seamlessly integrates high level synthesis and floorplanning for low power/energy. To estimate the effectiveness of the proposed low power floorplanning technique, Table 5.4 gives the wire energy consumption under different optimization scheme, which is also illustrated in Figure 5.10. Column 2 is the wire energy consumption using the traditional synthesis flow. In this flow, first power-conscious

synthesis is performed using the proposed iterative algorithm under the same assumption in Section 5.4.1 that the floorplan cost is zero. Then a power-unconscious floorplan is performed to optimize the total wire length. The Column 3 and Column 4 are the wire energy consumption and its reduction over Column 2 when the total wire length is optimized using the proposed integrated flow. Column 5 and Column 6 are the wire energy consumption and its reduction over Column 2 when energy is optimized using the proposed integrated flow. Summarily, Column 2 reports the result of traditional flow, in which high level synthesis is performed before floorplanning. Column 3 and Column 5 report the results of the proposed integrated flow, in which Column 3 uses wire length as the cost function and Column 5 uses energy as the cost function.

Table 5.4 Experimental Results on Wire Energy Consumption for FloM (pJ)

Benchmark	Traditional Flow	Optimize Wire Length		Optimize Energy	
		Energy	Δ (%)	Energy	Δ (%)
EWF	7286	5421	25.6	4138	43.2
IIR	6892	4818	30.1	4287	37.8
DCT	10275	6915	32.7	5620	45.3
DiffEq	4100	2821	31.2	2476	39.6
FDCT	11236	7292	35.1	5315	52.7
FIR	7967	5832	26.8	5035	36.8
TFIR	8360	6713	19.7	5743	31.3
Lattice	8261	5973	27.7	4337	47.5
Average	-	-	28.9	-	42.6

Figure 5.10 Reductions on Wire Energy for FloM

Using the proposed integrated flow, the energy consumed by wires is reduced by a factor of 28.9%. This reduction is due to fact that in the proposed technique, the behavioral synthesis and floorplanning are performed in an integrated manner, and the synthesis algorithm finds an optimized resource usage for the floorplanning process. Meanwhile, when the multi-voltage issue is taken into consideration, the proposed technique can achieve an average reduction of 42.6%. This reduction is due to the fact that in multi-voltage design, different wires could carry signals with different voltages, and the voltages must be taken into account to estimate the power consumption, as demonstrated by (5.9).

5.4.3 Floorplan-unaware VS Floorplan-aware Low Power Synthesis

In this section, experiments are performed on total energy consumption to compare the result of floorplan-unaware low power synthesis and floorplan-aware low power synthesis, as shown in Table 5.5. The comparison on total energy is also shown in Figure 5.11. The column “Traditional Flow” reports the wire energy, datapath energy and total energy using the traditional flow. In this flow, first power-conscious synthesis is performed using the proposed iterative algorithm under the same assumption in Section

5.4.1 that the floorplan cost is zero. Then a power-unconscious floorplan is performed to optimize total wire length. The column “Proposed” reports the wire energy, datapath energy and total energy using the proposed integrated flow in FloM. From the “Improvement” column, it is observed that an average wire energy reduction of 42.6% is obtained, which is already shown in Table 5.4. This is at the cost of 7.3% increase in the datapath energy consumption. This increase is because the cost function in the traditional flow is the datapath energy consumption (i.e. energy consumption by wire is not counted), while the cost function in the integrated flow is the total energy consumption. Averagely, the integrated flow achieves a reduction of 9.5% in total energy reduction compared to the traditional flow. In the next generation of semiconductor technology, the wire energy dominates the total energy, and the energy reduction of the integrated flow over traditional flow would be much higher.

Table 5.5 Experimental Results on Total Energy Consumption for FloM (pJ)

Benchmark	Traditional Flow			Proposed			Improvement (%)		
	Wire	Datapath	Total	Wire	Datapath	Total	Wire	Datapath	Total
EWF	7286	13634	20920	4138	14547	18685	43.2	-6.7	10.7
IIR	6892	13285	20177	4287	13843	18130	37.8	-4.2	10.1
DCT	10275	23663	33938	5620	25509	31129	45.3	-7.8	8.3
DiffEq	4100	7852	11952	2476	8152	10628	39.6	-3.9	11.1
FDCT	11236	21773	33009	5315	23754	29069	52.7	-9.1	12.0
FIR	7967	9191	17158	5035	9384	14419	36.8	-2.1	16.0
TFIR	8360	16442	24802	5743	16985	22728	31.3	-3.3	8.4
Lattice	8261	20637	28898	4337	23567	27904	47.5	-14.2	3.4
Average	-	-	-	-	-	-	42.6	-7.3	9.5

Figure 5.11 Reductions on Total Energy for FloM

5.4.4 Algorithmic Complexity

A major issue for physical-driven high level synthesis is the performance. This is because in an integrated physical synthesis and high level synthesis, physical information is usually estimated multiple times during high level synthesis. As floorplanning is the first stage of physical synthesis and thus has relatively low computational cost, floorplan-driven high level synthesis gives a good tradeoff between performance and quality of results.

In Table 5.6, a computational time comparison between traditional synthesis flow and the proposed integrated flow (FloM) is performed. The algorithms are implemented in Java and runs on a 2.4GHz Pentium PC with 256Mbytes of RAM. The row “Traditional Flow” reports the run time of the traditional flow. In this flow, power-conscious synthesis is first performed using the proposed iterative algorithm under the same assumption in Section 5.4.1 that the floorplan cost is zero. Then a power-unconscious floorplan is performed to optimize wire energy. The row “Proposed” reports the run time of the proposed integrated flow in FloM.

Table 5.6 Computational Time (in sec.)

Benchmark	EWF	IIR	DCT	DiffEq	FDCT	FIR	TFIR	Lattice
Traditional Flow (s)	5.34	1.69	9.01	0.21	9.30	4.69	1.84	1.98
Proposed (s)	16.22	4.77	62.04	1.05	55.64	30.21	12.37	13.08

Figure 5.12 Increase of Computational Time for FloM

From Table 5.6 and Figure 5.12, it is clearly shown that the better results of the proposed integrated flow are obtained at a cost. The run time is significantly increased. This is due to the fact that for every move to be evaluated, floorplanning must be performed. Therefore, efficient floorplanning algorithm must be available. The floorplanning algorithm used in FloM is seen to be efficient, and the run time of FloM is acceptable. This fact is regarded as a proof that integrating floorplanning into high level synthesis is a legitimate method to perform physical-driven high level synthesis for better optimization results.

5.4.5 Comparison

In this section, FloM is compared with some published results. Different techniques have different objectives and constraints, and results are affected by many parameters. The

objective of this comparison is to provide a general idea of the effectiveness of the proposed technique.

In Table 5.7, the average improvements on the multi-voltage datapath energy consumption are compared. FloM can achieve results comparable to those published techniques. The compared literatures are reviewed in Chapter 2.

Table 5.7 Average Improvement (%) Comparisons on Datapath for FloM

	[CHA97]	[MOH04]	[MAN02]	FloM
Energy Δ(%)	40.19	43.29	39	42.4

In Table 5.8, comparisons on the multi-voltage datapath energy consumption are performed on some benchmarks. It can be seen that FloM obtains comparable results on all benchmarks except the EWF case.

Table 5.8 Benchmark Improvement (%) Comparisons on Datapath for FloM

Benchmark	[CHA97]	[MOH04]	[MAN02]	FloM
FIR	-	41.55	33.2	66.0
EWF	44.47	43.01	44.2	33.6
DCT	-	43.58	30.4	44.8
FDCT	43.44	-	-	43.0
DiffEq	40.87	-	53	43.8

Since this is the first work on the floorplan-driven low power high level synthesis for multi-voltage design, in Table 5.9, we compare the wire energy and datapath energy reduction over traditional interconnect-unaware flow between FloM and [STA03], which is also based on simulated-annealing method. It should be noted that [STA03] does not perform any multi-voltage optimization; therefore the results achieved in the traditional flow in this work are normally better than those of IUO (Interconnect Unaware Optimization) in [STA03].

Table 5.9 Energy Reduction (%) Comparisons for FloM

Benchmark	[STA03]		FloM	
	Wire	Datapath	Wire	Datapath
FDCT	35.63	-0.79	52.7	-9.1
FIR	37.79	-0.07	36.8	-2.1
DiffEq	29.05	-0.02	39.6	-3.9
Average	41.24	-7.70	42.6	-7.3

Note that the average reduction is over all the experimental benchmarks instead of the three listed in Table 5.9. For the three benchmarks, FloM reduces the wire energy better than [STA03], but at a larger cost in the datapath energy. The total energy reduction means little in the comparison, because different wire energy models are used. Averagely, FloM yields a slightly better result in both wire energy reduction and datapath energy increase. As the reduction is over the results obtained by traditional floorplan-unaware low power synthesis flow, FloM achieves better results in this flow because multi-voltage optimization is considered. The reason of the worse datapath of FloM compared to [STA03] is also due to the use of multiple voltages.

5.5 Summary

In the traditional synthesis flow, physical synthesis is performed only after high level synthesis. As a result, no physical information is available during the high level synthesis. This problem becomes very critical in today’s DSM technology, in which the interconnects, including wires, occupy a large circuit area and contribute significantly to the speed as well as the power consumption. When no physical information can be used, the high level synthesis could generate results that are optimal at the for behavioral level, but fail at the physical level.

To close this gap, many research works have been shifted to the integration of physical information in the high level synthesis. Currently, the integration of floorplanning and high level synthesis gains much popularity, mainly because the integration of even lower

level physical information (placement, routing and layout) is too computation costly for the current computer hardware.

In this chapter, floorplan information is integrated in low-power high level synthesis targeting multi-voltage design using a two-level simulated annealing algorithm. The proposed technique considers the floorplanning of resources with different voltages and the wire energy consumption in addition to the energy consumption of FUs, MUXes, registers, as well as level converters. Using the proposed efficient local moves, FloM achieves 42.4% energy reduction in datapath compared to the result obtained from the power-unconscious synthesis. Compared to the traditional power optimization flow, the proposed integration flow achieves 42.6% energy reduction in wires at the cost of 7.3% energy increase in datapath. This percentage of energy reduction in wires is a significant amount in the future fabrication technology, where the energy consumed by wires dominates the total energy consumption.

Chapter 6 Application of Symbolic Computer Algebra in High Level Synthesis

6.1 Introduction

DSP and multimedia have become widespread applications in the field of digital circuits. In such data-path intensive applications, design optimization should be performed at a higher level than at logic level in order to satisfy the more and more aggressive time-to-market requirements, problem sizes and increasing design space dimensions. As an optimization method, high level transformation techniques are usually performed before high level synthesis. Such transformations often change or reconstruct the DFG of the input behavioral level description so as to reduce some design parameters, such as area and critical path delay (CPD), as well as energy consumption.

Unlike traditional mathematical computation on computers, which is based on arithmetic of fixed-length integers and fixed-precision floating-point numbers, modern Symbolic Computer Algebra (SCA) supports exact rational arithmetic arbitrary precision floating-point arithmetic, and algebraic manipulations of expressions containing symbols. As a powerful tool, computer algebra has witnessed great success in both science and engineering. In VLSI, it has been used for high-level formal verification problems [DEV91, SHE05]. However, its possible applications in high level synthesis have not yet been widely explored.

As an extension of distributivity which is used in traditional Tree Height Reduction (THR) [DEM94], polynomial factorization has the potential to reduce area, CPD as well as energy consumption. However, as far as the author is aware, hardly any research work has focused on the application of polynomial factorization in high level synthesis. It is believed that one of the reasons is that, unlike in logic synthesis, factorization cannot be done in high level synthesis without the concept and tool of SCA. Besides, polynomial factorization requires the input to be polynomials. Fortunately, nowadays there are many

commercial Computer Algebra Systems on the market, e.g. Maple, MATHCAD, and OCTAVE, in which Maple is one of the most successful ones. Moreover, polynomial formulation for different kinds of input circuit descriptions has also been established. Supported by these, factorization can be included as a high level transformation technique.

Pre-optimized complex components, such as square, MAC (Multiplier/Accumulator), sine, cosine, etc., have been widely used in digital circuit design to reduce cost. These components can be designed by designers as well as offered as a commercial library, such as the Synopsys DesignWare library. Unfortunately, hitherto no such complex component mapping automation tool is available. Contrarily, the mapping of these components is completely in hands of designers, and must be done manually. Now that a Computer Algebra System, such as Maple, is available, the concept of polynomial decomposition (simplification, reduction) can be applied to automate the mapping of complex components to designs.

In this chapter, many possible applications of SCA, such as factorization, expansion, term merging, and polynomial decomposition, in high level synthesis are explored. Two of the most important techniques, polynomial factorization and decomposition are given special attention.

In the proposed Subexpression Factorization technique, the best combinations of subexpressions for factorization to optimize area, CPD and energy consumption are explored. Factorization proves effective to reduce area, CPD and energy. In most cases, factorization is very effective in reducing area and energy. However, it is not so proficient in reducing CPD. In fact, sometimes it may increase the CPD. Therefore, factorization is regarded as a method to reduce area and energy consumption.

In the proposed Complex Components Mapping technique, polynomial decomposition is used to automate the mapping of pre-optimized complex components. The idea of using polynomial decomposition first appears in [PEY03a]. However, due to the feature of polynomial decomposition, sometimes it is better to decompose a subexpression of a

polynomial instead of the whole polynomial. This is because a Computer Algebra System will simplify the input polynomial in terms of mathematics, which has very different criteria from that of high level synthesis.

This chapter is organized as follow. Section 6.2 gives a survey of SCA and related works on the two techniques proposed in this chapter. Section 6.3 formulates the first technique named as Subexpression Factorization. Section 6.4 focuses on the second technique named as Complex Components Mapping. Other transformation techniques are introduced in Section 6.5. Experimental results are presented in Section 6.6, followed by a summary in Section 6.7.

6.2 High Level Transformation Techniques Based on SCA

In this chapter, two high level transformation techniques, Subexpression Factorization and Complex Components Mapping, are proposed. These techniques are based on the concept of SCA, which is described in Chapter 2 and further discussed here.

6.2.1 Symbolic Computer Algebra

The major purpose of a Computer Algebra System (CAS) is to manipulate a formula symbolically using computer. For example, expanding, factorizing, root finding, or simplifying an algebraic polynomial are some of the common uses of CAS. Nowadays there are many CASs in the market, in which Maple [MAP] is one of the most widely used tools.

The input of the proposed techniques is a polynomial extracted from a basic block (or a portion of it) of a design, as described in Chapter 2. Then necessary functions in Maple are called and the results are obtained. Two Maple functions are crucial in the proposed techniques: “factor” and “simplify”.

Some examples of the two functions have been given in Section 2.6. The function “factor” tries to factorize the input polynomial. If it fails, it outputs a functionally equivalent polynomial. The function “simplify” will decompose the input polynomial using equations called *siderels* in Maple.

The function “factor” is used in the subexpression factorization procedure, while the function “simplify” is the core of mapping complex components. Consider the polynomial $y - 1/6 y^3 - 1/5040 y^7$, which is a Taylor series expansion of sine function.

```
> sine:=y-1/6*y^3+1/120*y^5-1/5040*y^7:
```

```
> f:=y-1/6*y^3-1/5040*y^7:
```

```
> result:=simplify(f,{sine=z});
```

$$result := -\frac{y^5}{120} + z$$

These functions will perform their job in the context of mathematics. However, in high level synthesis, the factorized and simplified results may not be better for hardware implementation. That is one of the most important considerations in the proposed techniques. For factorization, global or local information must be kept to determine whether the factorization result yields a better mapping. For decomposition, because the feature of the “simplify” function, subexpression decomposition may be needed for a better decomposition result.

6.2.2 Factorization of Subexpressions

Factorization has been playing an important role in logic synthesis [BER90, STA94]. However, polynomial factorization in high level synthesis is much more complicated than Boolean expression factorization, and not many existing literatures have explored it. In [PEY01a, PEY03a], polynomial factorization is used as guideline in the complex components mapping procedure. Actually, distributivity, as used in THR, is a simple example of factorization.

If the whole polynomial cannot be factorized, some subexpressions of it may be. The proposed technique tries to find the best subexpression factorization combinations of a polynomial, i.e. the best way the polynomial can be partially factorized in terms of the cost function, such as energy, area and CPD. That is because some polynomials can be partially factorized in several ways. For example, the subexpression factorization results of $a c + a d + b c + b d$ can be $(a + b)(c + d)$, $a(c + d) + b(c + d)$, $(a + b)c + (a + b)d$ and so on. There are also nested subexpression-factorization forms. For example:

$$a^3 + a^2 c + 2 a^2 b + 2 a b c + a b^2 + b^2 c + a c + c^2 = ((a + b)^2 + c) (a + c)$$

Not only can subexpression factorization expose the hidden potential factorization ability of polynomials, can it also be used to explore the best subexpression-factorization form of a given polynomial which has already been factorized. For example, if the input polynomial is $((a + b)^2 + c) (a + c)$, it can be expanded into $a^3 + a^2 c + 2 a^2 b + 2 a b c + a b^2 + b^2 c + a c + c^2$. The purpose is to find whether the original form is the best factorization form. Another form, for example, $(a + b)^2 (a + c) + a c + c^2$ may yield a better result in terms of some cost functions.

Figure 6.1 DFGs before and After Factorization

Subexpression factorization may lead to savings for both area and CPD. Consider the DFGs in Figure 6.1 of the polynomial $ac + ad + bc + bd = (a + b)(c + d)$.

In this case, both number of components and CPD are reduced after factorization. However, it is found that factorization is much better at reducing area than reducing CPD. That is because the polynomial (or part of it) will become a product-of-sum form after factorization. Although both multiplication and addition are associative operators, the length of the associative sub-polynomials may probably be shortened after factorization, and hence the potential for THR is reduced. On the other hand, as an extension of the associativity operation, subexpression factorization can reduce the area in most cases. The effect of factorization on energy consumption is similar to that on area, because energy consumption is also directly related to the number of operations. The difference is that energy consumption of an operation is used instead of its area.

6.2.3 Mapping Complex Components

Commercial design tools offer high-level complex components other than multipliers and adders, including MAC, square, sine, cosine, etc. The users can also design their own complex components. As these complex components have been optimized beforehand, fully mapping them may yield significant advantages on area, performance and energy consumption. Unfortunately, the existing tools rely on explicit synthesis directives from designers to map complex components. The dependence on the intelligence of the designers reduces the possibility of using complex components widely.

As far as the author is aware, there are now few papers on the mapping problem of complex components. In [SMI98, SMI99, SMI01], a technique is proposed to match a portion of the data flow to library cells. However, if such match cannot be found, there is no automatic way to transform the data flow in order to map the library cells. This issue is addressed in [PEY01a, PEY03a], in which a novel technique is proposed to map complex library elements and code segments automatically in the proposed systems called Symsyn and Symsoft. In the technique, the input polynomial is decomposed into polynomials that can be represented by the available library components (hardware or

software). This decomposition strongly relates to a concept of SCA, which is called simplification modulo set of polynomials.

The inputs of complex components mapping are the input polynomial representing a portion of data flow extracted from the design, and the polynomial representation of the available library components, such as adder, multiplier, MAC, sine, cosine, etc. The polynomial formulation of library components is the same as the polynomial formulation of the input data flow. For example:

$$\begin{aligned}\cos(x) &\approx 1 - 1/2 x^2 + 1/24 x^4 - 1/720 x^6 + 1/40320 x^8 \\ \sin(y) &\approx y - 1/6 y^3 + 1/120 y^5 - 1/5040 y^7\end{aligned}$$

Assume that the input data flow is:

$$Z = \sin(y) - 1/120 y^5$$

A straightforward mapping is to use a sine module, subtractors, multipliers and squares. However, as sine module is large and slow, it may be better to avoid using it. In the proposed complex components mapping technique, firstly the input data flow is formulated as a polynomial:

$$Z \approx y - 1/6 y^3 - 1/5040 y^7$$

By means of simplification modulo set of polynomials, a sine module can be mapped in the polynomial, although it has no explicit sine function any longer. The method is to use a SCA System. Obviously the polynomial can also be mapped without the sine function. By comparing the mapping results with and without the sine function, it can be determined whether it is beneficial to use a sine component. Note that this comparison procedure needs no interaction of the designers, and can be performed automatically.

Here the complex components mapping procedure is regarded as a high level transformation technique, because actually the input data flow will be transformed into another data flow, which is functionally equivalent or approximately equivalent to the input. For example, the input $\sin(y) - 1/120 y^5$ is transformed into $y - 1/6 y^3 - 1/5040 y^7$ if using a sine function is not beneficial in terms of some predefined cost function. Note that as approximation is performed, simulation must be done to ensure that the precision is sufficient.

6.3 Proposed Subexpression Factorization

In this section, the first transformation technique, Subexpression Factorization, is presented. This technique depends on the Maple function “factor” to perform symbolic factorization on carefully chosen subexpressions, and aims to achieve a best subexpression-factorization in terms of area, CPD and energy consumption.

6.3.1 Problem Formulation

By transforming the input polynomial, factorization can reduce the number of operations (and potentially energy consumption and area) and sometimes CPD as well.

As stated previously, if a polynomial cannot be factorized, subexpressions of it may be. The two issues faced are discussed below.

(1) How to determine whether a factorization is beneficial or not?

The benefit of factorization can be determined with local or global information, which are called local determination and global determination respectively. If a polynomial is split into two subexpressions, that is:

$$P = P_s + P_r = P_f + P_r$$

where P_f is factorization result of P_s .

Given $\text{cost}(P)$ is the cost to implement the polynomial P in hardware, the following terms are defined.

Definition 6.1: Local Determination. If $\text{cost}(P_f) < \text{cost}(P_s)$, it is said that P_f is better than P_s and P_f is kept as factorization result. Otherwise, P_s is kept as the result.

Definition 6.2: Global Determination. if $\text{cost}(P_f + P_r) < \text{cost}(P_s + P_r)$, it is said that P_f is better than P_s and P_f is kept as factorization result. Otherwise, P_s is kept as the result.

The above definitions are illustrated with the following example:

$$a^2 + 2ab + b^2 + x + y + z = (a + b)^2 + x + y + z$$

In local determination, the DFGs of $P_f = (a + b)^2$ and $P_s = a^2 + 2ab + b^2$ are compared. In global determination, the DFGs of $(a + b)^2 + x + y + z$ and $a^2 + 2ab + b^2 + x + y + z$ are compared instead.

Generally, if the aim is to optimize energy or area, local determination is often sufficient. But global determination may be needed to optimize the CPD, because local determination excludes the impact of THR, which is discussed later. Global determination is never worse than local determination in terms of estimation quality. However, global determination may cause a drastic increase in computation time, especially when the subexpressions are much shorter than the whole polynomial. Alternatively a semi-global determination is used as in this work.

Definition 6.3: Semi-Global Determination. If $\text{length}(P_r) > \text{MAXLEN}$, local determination is used. Otherwise global determination is used. The function $\text{length}(P_r)$ computes the number of terms in P_r . MAXLEN is a predefined integer, which is often small ($1 \sim 5$).

Semi-Global Determination is a simple tradeoff between computation time and estimation quality.

(2) Which subexpressions and combinations should be selected?

In many situations there are many subexpressions of a polynomial that can all be factorized. Therefore the selection of subexpressions is an important issue in Subexpression Factorization. An exhaustive algorithm must try out all the subexpressions in all the combinations as factorization candidates to determine which subexpressions should be selected. However, the complexity is exponential.

To reduce the exploration space, the number of subexpression candidates for factorization needs to be reduced. As stated previously, there are many algorithms to test the irreducibility of polynomials [GAO01, KAL03]. Unfortunately, to the best knowledge of the author, all of them have restrictions or with high complexity, sometimes more complex than the factorization itself. Consequently it is neither efficient nor effective to implement them.

In this work a heuristic algorithm is proposed. It is assumed that the longest subexpression has the most possibility to achieve the best result, so longer subexpressions have higher priority to be handled. To reduce the subexpression candidates without too much additional cost, some criteria are applied on the selection process. These criteria are based on a graph defined as Term Dependency Graph (TDG).

Definition 6.4: A Term Dependency Graph $G (V, E)$ is an undirected graph, in which each vertex $v_i \in V$ accords to a term of the input polynomial. There is an edge $(v_i, v_j) \in E$ if and only if there exist one or more common variables in vertex v_i and v_j . The set of the common variables called CV is the weight of (v_i, v_j) .

For example, the TDG of a polynomial $a^2 + 2 a b + b^2 + a^2 b + x^2 y + x^2 z + w$ is shown in Figure 6.2.

Figure 6.2 TDG of $a^2 + 2ab + b^2 + a^2b + x^2y + x^2z + w$

The problem of selecting a subexpression now becomes the selection of a subgraph of the input TDG. As stated previously, the proposed heuristic algorithm assumes that the longest subexpression has the highest possibility to yield the best result. Accordingly, subgraphs are selected in the order of decreasing size for subexpression factorization. If there are several subgraphs that have the same size and can all be factorized, the one with the best result is selected.

Limiting the subexpression candidates for factorization is to reduce the number of selected subgraphs. Some criteria are applied on the subgraph selection. Figure 6.3 shows the TDG of the polynomial $ac + bc + bd + abcd$.

Figure 6.3 TDG of $ac + bc + bd + abcd$

Definition 6.5:

- (a) AV of a TDG is a set of all the variables appear in the graph (input polynomial). For example, the AV of Figure 6.3 is $\{a, b, c, d\}$.
- (b) ACV of a TDG is a set of all the variables in the CV s of all edges, i.e. $ACV = \bigcup_{vi} CV_i$. For example, the ACV of Figure 6.3 is $\{a, b, c, d\}$.
- (c) CVG of a TDG is a set of variables, each of which appears in the CV of every edge, i.e. $CVG = \bigcap_{vi} CV_i$. For example, the CVG of Figure 6.3 is Φ .

The applied criteria are as below.

Criteria 6.1: If a subgraph is not connected, i.e. there exists a node that is not connected to any other node in the subgraph, it will not be selected as candidate for factorization. In Figure 6.3, the subgraph $[ac, bd]$ will never be selected.

Criteria 6.2: If a subgraph has a $CVG \neq \Phi$, it will be selected as candidate for factorization. In Figure 6.3, the subgraph $[ac, bc]$ will be selected while selecting two-vertex subgraphs.

Criteria 6.3: If a subgraph has an $ACV \neq AV$, it will not be selected as candidate for factorization. In Figure 6.3, the subgraph $[ac, ad, bc]$ will not be selected.

Actually, Criteria 6.1 is a special case of Criteria 6.3. Unfortunately, Criteria 6.3 is not true in terms of mathematics. A typical example is:

$$x^3 + y^3 z^3 = (x y + z) (x^2 y^2 - x y z + z^2)$$

However, Criteria 6.3 is regarded as a satisfactory criterion for high level synthesis with very few exceptions.

6.3.2 Factorization Algorithms

In the proposed subexpression factorization technique, longer subexpressions have higher priority to be handled first. To deal with nested factorization, the proposed algorithm is split into two hierarchies: the subexpression factorization and the polynomial factorization.

6.3.2.1 Subexpression Factorization

Given a TDG $G(V, E)$, the function *subexpression_factorization* (), as described in algorithm 6.1, tries to find the best subexpression-factorization form of a polynomial with no consideration of nested factorization.

// Algorithm 6.1: Subexpression Factorization

// Function: *subexpression_factorization* ($G(V, E)$)

// Input: a Term Dependency Graph $G(V, E)$ that represents the input polynomial.

// Output: the subexpression-factorization form of the input polynomial.

```

1: final_result =  $\Phi$ ;
2: Level1: while ( $E \neq \Phi$ ) {
3:   find a connected subgraph  $G_s(V_s, E_s) \subseteq G(V, E)$ ;
   // Find factorized subgraphs in the order of decreasing size.
4:   Level2: for ( $i = |V_s|; i > 1; i--$ ) {
       // find subgraphs that can be factorized and their factorized results
       // using TDG and Maple
5:      $S = \{[G_i(V_i, E_i), \text{factor\_result}] \mid G_i \text{ is a connected Graph}$ 
                                      $\&\& G_i \subseteq G_s \&\& |V_i| = i$ 
                                      $\&\& G_i \text{ can be factorized}\}$ ;
6:     if ( $S = \Phi$ ) continue Level2;
       // Some subgraphs may have common vertices. In that case, if one
       // subgraph is selected as a factorized part, others must not be selected.
7:     results =  $\Phi$ ;
8:     while ( $S \neq \Phi$ ) {
```



```

9:       $[G_b(V_b, E_b), best\_result] = get\_best\_result(S);$ 
10:      $results = results \cup \{G_b(V_b, E_b)\};$ 
11:     for each  $[G_i(V_i, E_i), factor\_result] \in S$ 
12:       if  $(V_b \cap V_i \neq \Phi)$   $S = S \setminus [G_i(V_i, E_i), factor\_result];$ 
13:     }
14:      $final\_result = final\_result \cup results;$ 
15:     for each  $G_b(V_b, E_b) \in results$  {
16:        $V = V \setminus V_b;$  // remove vertices
17:        $E = E \setminus E_b;$  // remove edges
18:     }
19:     break Level2;
20:   }
21: }
22: return combine (final_result);

```

The algorithm selects subgraphs in the order of descending size (Line 4). If there is any subgraph that can be factorized (determined by TDG) and the factorization result (obtained by Maple function “factor”) yields a less cost in terms of local determination or global determination or semi-global determination, it will be stored as candidate (Line 5). When there are more than one candidates with the same size, the one with the greatest benefit is selected and those that have some common vertices (terms) with the best candidate are removed (Line 8 to Line 13). If there are some disconnected subgraphs, partition is needed. This selection and removal will be repeated until all the candidates have been processed. After this, intermediate factorization results will be stored (Line 14) and the graph is updated (Line 15 to Line 18). Finally, all intermediate factorization results are combined to form the final result (Line 22).

A simple illustration of the subexpression factorization process of the polynomial $a^2 + 2ab + b^2 + a^2b + x^2y + x^2z + w$ is presented, the TDG of which is shown in Figure 6.3. With different cost functions and estimation methods, the results may be different.

(1) Step 1:

Stored *final_result*: $(a + b)^2$

Remaining TDG:

Figure 6.4 Remaining TDG of Step 1

(2) Step 2:

Stored *final_result*: $(a + b)^2 + x^2 (y + z)$

Remaining TDG:

Figure 6.5 Remaining TDG of Step 2

(3) Step 3:

Stored *final_result*: $(a + b)^2 + x^2 (y + z) + a^2 b + w$

6.3.2.2 Polynomial Factorization

As the subexpression factorization algorithm handles the longest subexpression first, the polynomial factorization algorithm, which is the upper level of subexpression factorization, performs the factorization on the whole polynomial, as described in Algorithm 6.2. It calls Algorithm 6.1 to perform the subexpression factorization and each factorized expression is recursively passed to the polynomial factorization, hence achieving the nested factorization.

// Algorithm 6.2: Polynomial Factorization

```

1: Polynomial polynomial_factorization (Polynomial poly) {
2:   result = subexpression_factorization (TDG (poly));
3:   subs = polynomials in all parentheses of result;
4:   if (subs ≠ ∅) {
5:     for each polynomial s ∈ subs {
6:       p = polynomial_factorization (s); // recursive call
7:       update (result, s, p);
8:     }
9:   }
10:  return result;
11: }
```

By integrating the subexpression factorization algorithm (Line 2), the polynomial factorization algorithm further factorizes the sub-polynomials (Line 5) inside the result obtained by the function *subexpression_factorization* (). The function *update* (*result*, *s*, *p*) updates the *result* by substituting the expression *s* with the factorized result *p*. The factorization algorithm tries to find the best subexpression factorization and combination in terms of some predefined cost function and estimation method.

Consider the polynomial formulation of Phase Shift Keying (PSK) modulator used in digital communication [PEY03a]:

$$psk := 1 - \frac{1}{2}x^2 - xy - \frac{1}{2}y^2 + \frac{41667}{1000000}x^4 + \frac{41667}{250000}x^3y + \frac{125001}{500000}x^2y^2 + \frac{41667}{250000}xy^3 + \frac{41667}{1000000}y^4$$

The polynomial factorization process may be (also depends on cost functions and estimation methods):

Step1:

$$\text{result} := \left(\frac{41667}{1000000}x^2 + \frac{41667}{500000}xy - \frac{1}{2} + \frac{41667}{1000000}y^2 \right) (y+x)^2 + 1$$

Step 2:

$$\text{result} := \left(\frac{41667(y+x)^2}{1000000} - \frac{1}{2} \right) (y+x)^2 + 1$$

As stated before, if the input is a partially factorized polynomial, e.g. the input is the result of step 1, it can be expanded first to find whether the input is the best subexpression-factorization form.

6.4 Proposed Complex Components Mapping

In this section, the second transformation technique, Complex Components Mapping, is presented. This technique depends on the Maple function “simply” to perform symbolic decomposition for the optimization of area, CPD and energy consumption.

6.4.1 Problem Formulation

The inputs of Complex Components Mapping are the polynomial formulation of a (portion of) DFG as well as the polynomial formulation of the complex components. The mapping is based on polynomial decomposition, i.e. to decompose a polynomial using equations. The corresponding function in Maple is “simplify”. Similar to factorization, as Maple is a mathematical tool, it will decompose the polynomial as completely as possible. However, when High Level Synthesis is concerned, subexpression decomposition may be better than the whole polynomial decomposition. For example:

```
> sine:=x-1/6*x^3+1/120*x^5-1/5040*x^7;
```

$$\text{sine} := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7$$

```
> f:=x^8+x-1/6*x^3+121/120*x^5-1/5040*x^7;
```

$$f := x^8 + x - \frac{1}{6}x^3 + \frac{121}{120}x^5 - \frac{1}{5040}x^7$$

> **simplify(f,{sine=z});**

$$x^5 + 5040x^2 - 840x^4 + 42x^6 - 5040xz + z$$

Actually, what is preferred as the final result is another form: $f = x^8 + \text{sine} + x^5$. The complication of the Maple output is because Maple will decompose any term with order not lower than the highest order of the side relations (In this example, the side relation is “ $\text{sine} = z$ ”). As a result, the term x^8 is also decomposed by the side relations.

Due to this feature of Maple, an algorithm based on the decomposition of subexpressions is proposed, in which complex components are mapped to certain subexpressions of the input polynomial.

Similar to Subexpression Factorization, an important issue in subexpression decomposition is how to select good subexpressions, because selecting all subexpressions of a polynomial is of exponential complexity. Fortunately, as polynomial simplification will only decompose the term whose order is not lower than the highest order of the side relations, the low-order terms of a subexpression has no influence on the decomposition result. As a result, only the terms with orders higher than or equal to the highest order of the side relation are the candidates for subexpression selection.

Subexpression decomposition is more rigorous than Subexpression Factorization on the determination of whether a decomposition is better. In most cases, global determination is indispensable to get a satisfactory estimation of the upper bound of any intermediate results. This is because if the highest order of a variable, e.g. x , of the side relations is n , polynomial decomposition may generate certain new term, e.g. x^{n_i} , whose order n_i satisfies $0 \leq n_i < n$. If there are already such terms, a good estimation requires them to be merged, which makes the global information essential. This is named Term Merging and described in Section 6.5.1.

6.4.2 Mapping Algorithm

The proposed Complex Components Mapping algorithm, which is based on subexpression decomposition, uses a modified branch-and-bound approach. In a typical branch-and-bound algorithm, the lower bound of an intermediate result is calculated and compared with a bound set as the cost of best result found so far. However, in the case of Complex Components Mapping, it is difficult to calculate a sharp lower bound, thus the upper bound of an intermediate result is calculated instead. The upper bound is then compared with the upper bound of result obtained by mapping only simple components. Algorithm 6.3 performs the Complex Components Mapping.

// Algorithm 6.3: Complex Components Mapping

```

1:  $S = S_0$ ; // initial solution, in which polynomial is not mapped
2:  $FULL\_MAP = \Phi$ ;
3:  $R = \Phi$ ;
4: Level1: while ( $S \neq \Phi$ ) {
5:   select  $s \in S$ ;
6:    $S = S \setminus \{s\}$ ;
7:    $SUB = select\_subexpressions(s)$ ;
8:    $USED = \Phi$ ;
9:   Level2: for each  $sub \in SUB$  {
10:     $LIB = allocate\_complex\_library\_elements(sub)$ ;
11:     $can\_map = false$ ;
12:    Level3: for each  $lib \in LIB$  {
13:      if ( $lib \in USED$ ) continue Level3;
14:       $r = simplify(sub, lib, s)$ ;
15:      if ( $upper\_bound(r) < upper\_bound(sub)$ ) {
16:         $S = S \cup \{r\}$ ;
17:         $USED = USED \cup \{lib\}$ ;
18:         $can\_map = true$ ;

```

```

19:      }
20:  }
21:  if (!can_map) FULL_MAP = FULL_MAP  $\cup$  {s};
22:  }
23: }
24: for each  $r \in FULL\_MAP$  {
25:    $r = MAC\_mapping(r)$ ;
26:    $R = R \cup \{r\}$ ;
27: }
28: report best of R;
*****

```

In Algorithm 6.3, complex components are mapped into a subexpression of the input polynomial. Subexpressions are calculated by the function *select_subexpressions ()* in Line 7. Section 6.4.1 explains that only the terms with orders higher than or equal to the highest order of the side relation are the candidates for subexpression selection. The algorithm tries to map complex components for each subexpression (Line 9). If it succeeds, the new solution is added as a new branch of the solution tree (Line 15). Otherwise the solution will be stored (Line 21) and eventually will be further mapped by function *MAC_mapping ()* (Line 25). To increase flexibility, whenever a subexpression is to be mapped, a set of complex library components is allocated (Line 10), each of which will be evaluated (Line 12) by calling the Maple function *simplify ()* in Line 14. If the mapping leads to savings according to the cost function, other subexpressions will not map this component in order to avoid repetition. Note that MAC mapping is only performed in function *MAC_mapping ()*, because MAC does not have a static form, making it not straightforward to call the Maple function “simply”. MAC mapping is discussed later in Section 6.5.4.

The process is demonstrated with the example below.

```
> sine:=x-1/6*x^3+1/120*x^5-1/5040*x^7;
```

$$\text{sine} := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7$$

> f:=x^8+x-1/6*x^3+121/120*x^5-1/5040*x^7;

$$f := x^8 + x - \frac{1}{6}x^3 + \frac{121}{120}x^5 - \frac{1}{5040}x^7$$

The function f can map sine, cosine, square, etc. It is found that mapping sine with the subexpression: $\text{subexpr} = -1/5040 x^7$ yields the best result as illustrated below.

> subexpr:=-1/5040*x^7;

$$\text{subexpr} := -\frac{x^7}{5040}$$

> result:=simplify(subexpr,{sine=z});

$$\text{result} := -x + \frac{1}{6}x^3 - \frac{1}{120}x^5 + z$$

> f:=f-subexpr+result;

$$f := x^8 + x^5 + z$$

Then the result $f = x^8 + \text{sine} + x^5$ is obtained. This intermediate result can be mapped further with the function square.

This example also illustrates why global determination is needed. When simplifying $-1/5040 x^7$ using sine, the following is obtained.

$$\text{result} := -x + \frac{1}{6}x^3 - \frac{1}{120}x^5 + z$$

The simplified result of $-1/5040 x^7$ contains terms with new orders like $-x$, $1/6 x^3$, and $-1/120 x^5$, which also appear in the remaining subexpression of the input polynomial f . So the upper bound estimation is rather bad without combining the simplification result with the remaining subexpression of the input polynomial.

6.5 Relevant Transformation Techniques

During Subexpression Factorization and Complex Components Mapping, some other transformation techniques for optimization are embedded. These techniques are discussed in the following sections.

6.5.1 Term Merging and Coefficient Expansion

Term Merging is described previously in Section 6.4.1. It merges the terms with the same core. For example, $a+2a$ can be merged to $3a$. As polynomial decomposition may generate new terms, Term Merging is essential for a satisfactory estimation in the Complex Components Mapping, as stated before.

Just as its name implies, Coefficient Expansion is to expand the coefficient of a product-of-sum polynomial. It may reduce area and/or CPD. For example, the transformation of $5(2a+3b)$ into $10a+15b$ can save area or CPD, depending on the scheduling and binding.

Coefficient Expansion is useful in factorization due to the feature of Maple. Maple will factorize coefficients sometimes as in the example shown below.

```
> f:=10*a*x+10*a*y+15*b*x+15*b*y:
> factor(f);
5 (x + y) (2 a + 3 b)
```

The reason is still that Maple is a mathematical tool. The factorization result may be better in terms of hardware implementation if it is $(10a+15b)(x+y)$.

6.5.2 Tree Height Reduction

As stated in Chapter 2, THR is a software technique which witnesses its success in high level synthesis as a method to reduce the CPD. A typical THR uses commutativity, associativity and distributivity to reduce the arithmetic expression tree height. In high

level synthesis, THR [NIC91, KOL94] can be used after the DFG is constructed, to move operators up and down to see whether the result is better. To reduce complexity, guidelines are used which are only suitable for adder and multiplier. In the proposed techniques, THR is embedded as a compilation technique during the establishment of the DFG. The following two simple criteria are applied.

(1) When mapping a single term with a squarer or multiplier, if the term consists of variables $x_0, x_1, x_2, \dots, x_n$, two variables x_i, x_j are found such that,

$$x_i.delay \leq x_k.delay, \text{ for every } 0 \leq k \leq n, \text{ and } k \neq j.$$

$$x_j.delay \leq x_k.delay, \text{ for every } 0 \leq k \leq n, \text{ and } k \neq i.$$

and the multiplication of x_i and x_j is mapped.

The delay of a variable is defined as the delay of critical path from the primary inputs to the variable in the DFG. By mapping the two variables with the least delay in a term, more parallelism in the DFG is achieved so that tree height is reduced.

(2) When mapping several sum-of-product terms $T_0, T_1, T_2 \dots T_n$, with adder, MAC or other complex components, the two terms (if map two-operand operators, e.g. adder and MAC) T_i, T_j are found such that,

$$T_i.delay \leq T_k.delay, \text{ for every } 0 \leq k \leq n, \text{ and } k \neq j.$$

$$T_j.delay \leq T_k.delay, \text{ for every } 0 \leq k \leq n, \text{ and } k \neq i.$$

and the two terms are mapped. After mapping two terms, the number of terms is reduced by one. Thus eventually the multi-term can be fully mapped by repeating the two-term mapping. When mapping two terms, the two terms with the least delays are selected. However, when there are terms with exactly the same delay, the selection is random.

Similarly, the delay of a term is defined as the delay of critical path from the primary inputs to the variables of the term in the DFG. By mapping the two terms with the least delay, more parallelism is achieved so that tree height is reduced.

6.5.3 Term Factorization

Different from the polynomial factorization, Term Factorization is a technique to factorize a term. For example, the term $a^2 b^2$ needs two squares (If there are no square module in the library, multipliers are used instead) and one multiplier. After term-factorize it into $(a b)^2$, only one square and one multiplier are needed.

Term factorization will dramatically influence the operation sharing. Consider the polynomial $a^2 b^2 + a^2$, although $(a b)^2$ is better than $a^2 b^2$, it loses the opportunity to reuse a^2 , which is used in another term of $a^2 b^2 + a^2$. One method is to use the global determination to judge whether to use the term-factorized form, which turns out to be rather time-consuming. Fortunately, most of the time it is not a problem while applying Term Factorization to the result of polynomial factorization, because polynomial factorization can perform similar operation sharing itself. Therefore Term Factorization is used as a subset of polynomial factorization.

As the number of variables and their orders of a term increase, Term Factorization becomes more complicated. Similar to the polynomial factorization, the objective of Term Factorization is to find the best combinations of variables. Algorithm 6.4 gives the main steps of Term Factorization.

// Algorithm 6.4: Term Factorization

Step (1): Find all combinations of two high-order variables.

Step (2): Term-factor each combination and see which one yields the best result.

Step (3): Compile the best combination in the DFG.

Step (4): Update the variables information and return to step (1).

The term $a^2 b^3 c^4 d^6$ is used as an example to illustrate the steps. When the optimization target is the area, the following is obtained.

$$a^2 b^3 c^4 d^6 \rightarrow (a c^2)^2 b^3 d^6 \rightarrow (a c^2)^2 (b d^2)^3 \rightarrow b d^2 (a c^2 b d^2)^2$$

If the optimization concern is area * CPD, the following is obtained.

$$a^2 b^3 c^4 d^6 \rightarrow b (a b)^2 c^4 d^6 \rightarrow b (a b)^2 (c d)^4 d^2$$

Like polynomial factorization, Term Factorization may decrease the number of components used, and hence reduce energy and area. It is not so effective at reducing CPD.

6.5.4 MAC Mapping

MAC implements the functionality of $a b + c$, in which a, b, c can be signal or constants. MAC is especially useful in DSP. As MAC does not have a static form like cosine and sine, it is difficult to call Maple function “simplify” to map MACs because suitable siderels are difficult to obtain. In [PEY03a], the authors use some guidelines, such as factorization and Horner Form to find the siderels for MAC mapping. However, as global determination is required (an MAC will always be favorable if local determination is used), the computation may explode as the problem size increases. An alternative method is proposed to map MACs separately and implant the algorithm into the Complex Components Mapping algorithm, as shown in Algorithm 6.3.

Before the mapping, the input polynomial of the MAC mapping algorithm is pre-processed into an interim polynomial to suit the MAC mapping process. Each term of the interim polynomial has at most one multiplication. For example, $a b c + b d + e$ is partially mapped into $x0 + b d + e$, where $x0 = b c$. Algorithm 6.5 describes the MAC Mapping algorithm.

// Algorithm 6.5: MAC Mapping

// Input: a pre-processed polynomial poly

```

1: SL = no_mult_terms (poly);
2: ML = one_mult_terms (poly);
3: while (!(|SL| == 1 && |ML| == 0)) {
4:   t1 = least_delay_term (SL);
5:   t2 = least_delay_term (ML);
   // Map the two terms with one MAC
6:   r1 = map_mac (t1, t2);
   // Map the two terms with one multiplier and one adder
7:   r2 = map_mult_adder (t1, t2);
8:   if (cost (r1) < cost (r2))
9:     update (SL, ML, r1);
10:  else
11:    update (SL, ML, r2);
12: }
```

Algorithm 6.5 tries to map MAC as many as possible, as long as the mapping is beneficial. Firstly, the terms with no multiplication (Line 1) and the terms with only one multiplication (Line 2) are obtained. These terms are then mapped step by step (Line 3) using a MAC (Line 6) or one adder plus one multiplier (Line 7). The terms with shorter delays in the expression is mapped first in order to achieve maximum parallelism (Line 4 and Line 5). After the mapping, the solution with less cost (Line 8) is used to update the two lists of terms (Line 9 and Line 11). For example, consider the polynomial $a+b+c d+e f$. When Algorithm 6.5 is performed, the following is obtained.

$$a+b+c d+e f \rightarrow \text{MAC}(a, c d)+b+e f \rightarrow \text{MAC}(a, c d)+\text{MAC}(b, e f)$$

Typically, mapping MAC is rather useful to decrease the energy/area. However, sometimes it may increase the CPD. This occurs in such a situation: if a MAC is used, it will be in the critical path; but if an adder and a multiplier are used instead, only the multiplier is in the critical path. That is to say, using MACs will decrease the paths in the DFG and thus decreases the possible way that the critical path can go through. Algorithm 6.5 tries to balance the energy/area and the CPD.

6.6 Experimental Results

In this chapter, two novel high level transformation techniques, Subexpression Factorization and Complex Components Mapping, are proposed. Experiments were carried out to evaluate the two algorithms in terms of area, CPD, as well as energy consumption.

6.6.1 Experiments on Subexpression Factorization

In the experiments, the library contains only adders and multipliers. The characteristics of an adder and a multiplier are obtained by the synthesis result of Synopsys Design Compiler and reported in Table 6.1. The technology used is TSMC 0.6 μm .

Table 6.1 Characterization of Library Components for Subexpression Factorization

Library Component	Area (unit)	Delay (ns)	Energy (pJ)
Adder	177	10.24	48
Multiplier	3140	18.56	521

Experiments were carried out with some typical data flow examples. These data flow examples are formulated as polynomials, as shown in Table 6.2. The first five polynomials are from [PEY03a]. The last four polynomials are typical polynomial for testing the competence of the proposed algorithms.

Table 6.2 Benchmarks for Subexpression Factorization

P1	$a^2 - b^2$
P2	$b^3 + b a^2 c$
P3	$1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 + x + yz$
P4	$1 - a^2 - b^2 + a^2 b^2 + \frac{1}{2}a^4 + \frac{1}{2}b^4 - \frac{1}{6}a^6 - \frac{1}{2}a^4 b^2 - \frac{1}{2}a^2 b^4 -$ $\frac{1}{6}b^6 + \frac{1}{24}a^8 + \frac{1}{6}a^6 b^2 + \frac{1}{4}a^4 b^4$ $+ \frac{1}{6}a^2 b^6 + \frac{1}{24}b^8$
P5	$1 - \frac{1}{2}x^2 - xy - \frac{1}{2}y^2 + \frac{41667}{1000000}x^4 + \frac{41667}{250000}x^3y +$ $\frac{125001}{500000}x^2y^2 + \frac{41667}{250000}xy^3 +$ $\frac{41667}{1000000}y^4$
P6	$ac + ad + bc + bd + b^2d^2$
P7	$a^2 + 2ab + b^2 + x^2 + 2xy + y^2 + 1$
P8	$ax + ay + a + x + y + 1$
P9	$a^2 - b^2 + a - b$

The results are shown in Table 6.3 and illustrated in Figures 6.6 to 6.8. Area is estimated by adding the area of all the operations, regardless of resource sharing. CPD is estimated by finding the longest path from inputs to output, regardless of clock period and scheduling information. As subexpression factorization is not designed for CPD optimization, area is the optimization cost function in the columns “Area” and “CPD”, and energy consumption is the optimization cost function in the column “Energy”.

Table 6.3 Experimental Results for Subexpression Factorization

Benchmark	Without Factorization			With Factorization			Improvement (%)		
	Area	CPD	Energy	Area	CPD	Energy	Area	CPD	Energy
P1	6457	28.80	1090	3494	28.80	617	45.9	0.0	43.4
P2	15877	47.36	2653	12737	65.92	2132	19.8	− 39.2	19.6
P3	16408	65.92	2797	13268	76.16	2276	19.1	− 15.5	18.6
P4	87258	106.88	14739	23219	133.76	3983	73.4	− 25.1	73.0
P5	54796	86.40	9241	9951	86.40	1707	81.8	0.0	81.5
P6	22688	49.28	3839	9951	47.36	1707	56.1	3.9	55.5
P7	26182	59.52	4456	6988	49.28	1234	73.3	17.2	72.3
P8	7165	39.04	1282	3671	39.04	665	48.8	0.0	48.1
P9	6811	39.04	1186	3671	39.04	665	46.1	0.0	43.9
Average							64.3	− 8.3	63.7

Figure 6.6 Reductions on Area by Subexpression Factorization

Figure 6.7 Influences on CPD by Subexpression Factorization

Figure 6.8 Reductions on Energy by Subexpression Factorization

On average, the area is reduced by 64.3%, energy is reduced by 63.7% and CPD is increased by 8.3%. The conclusion is that Subexpression Factorization is very effective in reducing area and energy, but not CPD. The reason is that factorization can reduce the number of operations, and thus area and energy. However, the critical path is not determined by the number of operations, but the delay accumulation of operations. Therefore, factorization is not suitable for CPD optimization. The proposed algorithms

have the ability to find the best subexpression-factorization form (including nested form) in terms of a predefined cost function. If the input polynomial cannot be factorized, the proposed algorithms will only take a negligible extra time to process it.

6.6.2 Experiments on Complex Components Mapping

To evaluate the effectiveness of Complex Components Mapping, a different set of benchmarks is used, as listed in Table 6.4.

Table 6.4 Benchmarks for Complex Components Mapping

B1	$a^2 - b^2$
B2	$b^3 + b a^2 c$
B3	$1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 + x + yz$
B4	$1 - a^2 - b^2 + a^2 b^2 + \frac{1}{2}a^4 + \frac{1}{2}b^4 - \frac{1}{6}a^6 - \frac{1}{2}a^4 b^2 - \frac{1}{2}a^2 b^4 - \frac{1}{6}b^6 +$ $\frac{1}{24}a^8 + \frac{1}{6}a^6 b^2 + \frac{1}{4}a^4 b^4 + \frac{1}{6}a^2 b^6 + \frac{1}{24}b^8$
B5	$1 - \frac{1}{2}x^2 - xy - \frac{1}{2}y^2 + \frac{41667}{1000000}x^4 + \frac{41667}{250000}x^3y + \frac{125001}{500000}x^2y^2 +$ $\frac{41667}{250000}xy^3 + \frac{41667}{1000000}y^4$
B6	$\cos(x) + x^5 + x^3 + x$
B7	$\sin(x) + x^3 + x$
B8	$2 \cos(x) + 3 \sin(x)$
B9	$\sin(x) \cos(x)$
B10	$x^{10} + \cos(x)$

The characteristics of the library components are reported in Table 6.5. The library components used here are adder, square, multiplier, MAC, sine and cosine. The listed characteristics are normalized with respect to those of an adder. From the table it is found that an MAC does not yield too much advantage over an adder plus a multiplier. This

limits the usage of MACs in the mapping procedure. However, MAC is quite efficient in practice. If more precise estimation is available, the number of MACs used will be much more.

Table 6.5 Normalized Delay, Area and Energy of Library Components

Library Element	Normalized Delay	Normalized Area	Normalized Energy
Adder	1.00	1.00	1.00
Square	1.05	5.95	6.76
Multiplier	1.35	8.84	10.85
MAC	2.29	9.45	8.27
Sine	6.00	41.43	52.36
Cosine	6.02	41.28	51.98

The experimental results are shown in Table 6.6, and illustrated in Figures 6.9 to 6.11. For the columns “Area” and “CPD”, the optimization cost function is Area*CPD, which is a simple tradeoff between area and delay. For the column “Energy”, optimization cost function is the energy consumption. Like the Subexpression Factorization, the Complex Components Mapping technique achieves considerable improvements in area and energy, but not in CPD. Although averagely the Complex Components Mapping yields an average improvement of 2.6% in CPD, there are cases in which CPD is increased. On the other hand, average reductions of 23.7% in area and 29.5% in energy are achieved.

Generally, Complex Components Mapping achieves area, CPD and energy optimization by using pre-optimized modules instead of common multipliers and adders. For example, MACs are widely used in DSP applications to construct the datapath consisting of a multiplier plus an adder. MACs provide better characteristics than a multiplier plus an adder in area, delay as well as energy consumption.

Table 6.6 Experimental Results of Complex Components Mapping

Benchmark	Without Complex Components			With Complex Components			Improvement (%)		
	Area	CPD	Energy	Area	CPD	Energy	Area	CPD	Energy
B1	18.68	2.35	22.7	12.90	2.05	14.52	30.9	12.8	36.0
B2	45.20	3.70	55.25	39.42	3.70	42.86	12.8	0.0	22.4
B3	48.20	5.05	58.25	42.42	4.45	45.86	12.0	11.9	21.3
B4	252.68	9.05	306.95	234.56	8.45	277.61	7.2	6.6	9.6
B5	158.28	7.05	192.45	145.16	6.75	164.8	8.3	4.3	14.4
B6	95.40	7.40	115.5	73.08	7.02	82.66	23.4	5.1	28.4
B7	91.40	6.05	111.5	57.83	7.00	68.39	36.7	15.7	38.7
B8	140.60	7.40	170.75	101.39	8.37	127.04	27.9	18.0	25.6
B9	201.48	8.40	245.4	91.55	7.37	115.19	54.6	12.3	53.1
B10	84.56	7.40	102.65	68.97	7.02	84.11	18.4	5.1	18.1
Average							23.7	2.6	29.5

Figure 6.9 Reductions on Area by Complex Components Mapping

Figure 6.10 Influences on CPD by Complex Components Mapping

Figure 6.11 Reductions on Energy by Complex Components Mapping

6.6.3 Comparison

The proposed algorithms are also compared with some existing works. The results are reported in Table 6.7. The column “existing work” refers to [PEY03a], and the column “proposed technique” refers to a simple integration of the transformation techniques proposed in this chapter. In the integrated technique, first Subexpression Factorization is

performed, and then Complex Components Mapping is performed on the result. As the energy consumption was not reported in [PEY03a], it is not compared here.

Table 6.7 Comparisons

Benchmark	Existing Work		Proposed Technique		Improvement (%)	
	Area	CPD	Area	CPD	Area	CPD
B1	10.84	2.35	10.84	2.35	0.0	0.0
B2	30.19	4.69	30.19	4.69	0.0	0.0
B3	41.24	5.58	35.30	5.63	14.4	-0.9
B4	96.61	9.41	51.09	9.97	47.1	-6.0
B5	42.28	7.02	25.85	6.63	38.9	5.6
Average					30.7	-0.76

From Table 6.7, an average area improvement of 30.7% over the existing work can be seen, with only 0.76% increase in CPD. This achievement is mainly due to the fact the proposed techniques optimize on subexpressions of the input polynomial, instead of the whole polynomial.

6.7 Summary

In this chapter, two new high level transformation techniques, namely the Subexpression Factorization and the Complex Components Mapping, which are based on the concept of SCA, are proposed. Their effects on area, CPD and energy consumption are studied. The experimental results obtained show that the proposed techniques are effective and yield better results than published techniques.

In general, polynomial factorization can be regarded as an extension of distributivity which is used in the conventional THR techniques. Unlike in logic factorization, it is almost impossible to utilize polynomial factorization without a Computer Algebra System. Polynomial factorization has the ability to reduce the number of operations, and thus potentially area, CPD and energy. In this chapter, a heuristic algorithm is proposed

to find the best subexpression-factorization form of the input polynomial. From the experimental results, it can be concluded that the polynomial factorization is very useful in reducing area and energy. However it is less capable of reducing the CPD.

Using pre-optimized complex components can yield great benefit in area, CPD and energy consumption. Unfortunately, no EDA tool is available to automatically map them. Instead, the mapping process relies on the designers. The concept of polynomial decomposition has been introduced as a technique of automating the process of mapping complex components. In this chapter, the subexpression decomposition is proposed to find the best mapping. The experimental results show that the proposed technique can reduce area, CPD as well as energy.

As a popular research area in mathematics, theorem proving and high-level verification, SCA is demonstrated to be efficient also in high level synthesis in this chapter. Besides the polynomial factorization and the polynomial decomposition, other algorithms in SCA may also be explored for high level synthesis.

Chapter 7 Conclusions and Future Works

7.1 Conclusions

Recent years have witnessed an explosion in the size of digital applications, especially digital signal processing and multimedia applications. To optimize the designs for the applications with such large scale, optimization techniques must be performed at a higher level of abstraction. For this reason, high level synthesis has been an active research topic for the past two decades. Traditionally, area and performance are the two major optimization targets in digital system designs. However, with the ever-increasing demands in portable and wireless applications, power/energy has been gaining more attention and now becomes the major optimization objective. This is because the current battery technology cannot sustain the complexity of modern digital applications very well. On the other hand, with the modern DSM process technology, the area and delay of cells are decreasing, making power consumption the most important design parameter to be considered in many systems.

Research works on power optimization in high level synthesis can be classified into two categories, power modeling and power optimization. Power modeling aims to accurately represent the power/energy consumption of a circuit in mathematics, usually without the presence of detailed implementation information of the circuit. Power optimization aims to optimize the structure of the circuit in order to achieve better implementation in terms of power and energy consumption. In this thesis, the existing power modeling and optimization techniques are reviewed and discussed in Chapter 2. Several power optimization techniques are proposed in Chapter 3 to Chapter 6 to explore different methods for reducing the power/energy consumption in VLSI systems.

As power consumption has a linear dependency on switching activity, SA-based power optimization techniques have gained much attention in the history of low power synthesis. In Chapter 3, a SA-based technique named LAS is proposed. This technique utilizes the concept of look-ahead, backtracking and weighted bipartite matching to perform

integrated scheduling and binding for SA reduction. During the list-scheduling-based scheduling process, several control steps are looked ahead and the bipartite-matching-based binding is performed to obtain the optimal binding for these control steps with the minimal SA cost. Then backtracking is performed to limit the local minima. This process is repeated until a complete synthesized system is obtained. A register optimization technique is also integrated in LAS. The technique adopts a register management technique to eliminate spurious SA and uses bipartite matching to perform the register binding to reduce the SA on registers. The experimental results show that an average of 54.8% power reduction can be achieved using the proposed LAS technique.

The voltage-scaling-based low power techniques have gained more popularity recently, because power has a quadratic dependency on supply voltage. To compensate the performance degradation caused by scaling down supply voltage, multi-voltage synthesis and frequency scaling have been proposed. In multi-voltage synthesis, the resources in the critical path are assigned with higher voltage to maintain performance, while the resources not in the critical path assigned with lower voltage to reduce power. Frequency scaling techniques have also been explored, in which frequency is scaled up to compensate the performance degradation by other techniques such as voltage scaling. In Chapter 4, a technique named MuVoF is proposed. This technique targets energy optimization of functionally pipelined datapath using multi-voltage and multi-frequency synthesis under resource and throughput constraints. Unlike other frequency scaling techniques, in MuVoF, frequency is scaled up in certain pipeline stages under throughput constraints so that the supply voltages of certain resources can be scaled down. MuVoF consists of two steps. The first step performs stage partition and multi-voltage assignment to obtain an initial schedule. The second step iteratively refines the result. Techniques such as Clock Period Extension are also incorporated in MuVoF to achieve better results. The experimental results show an average 58.6% energy reduction when the constraints are relatively loose.

In modern DSM technology, interconnects, such as wires, contribute a large amount of total power and energy consumption. However, in high level synthesis, physical level

information is not available. Many works have been proposed to address the integration of high level synthesis and physical synthesis to achieve better power estimation during power optimization. A technique named FloM is proposed in Chapter 5 to perform floorplan-driven synthesis for multi-voltage datapath. The technique adopts a two-level iterative approach. In the inner level, the floorplanning for multi-voltage datapath is performed by iteratively refining the floorplan for optimizing power on wires. In the outer level, the multi-voltage datapath is iteratively optimized in terms of power consumption on both the datapath and the connecting wires. Several local moves are developed to refine the datapath in the outer level of the algorithm, by performing rescheduling, rebinding and voltage reassignment. The two nested iterative processes work together in FloM to achieve an optimized datapath and floorplan so that the total power consumption is minimized. An average wire energy reduction of 42.6% is observed in the experiments.

Finally, two low power high level transformation techniques, namely the Subexpression Factorization and the Complex Components Mapping, are proposed in Chapter 6. Performed before high level synthesis tasks, high level transformations aim to modify the DFG to give the high level synthesis a good start point. These two techniques are based on the modern Symbolic Computer Algebra and use several functions from the famous tool Maple. By finding the best subexpressions and calling the Maple function “factor”, the Subexpression Factorization finds the best factorization form of an input polynomial. On the other hand, the Complex Components Mapping finds the best subexpressions and calls the Maple function “simply” in order to map the input polynomial using pre-optimized complex modules, such as square, sine and cosine. By integrating with other proposed transformation techniques, such as the Term Merging, Coefficient Expansion, THR, Term Factorization and MAC mapping, the two techniques are able to achieve considerable improvements on mapping of the input polynomial to DFG. Averagely, the energy is reduced 63.7% by the Subexpression Factorization, and 29.5% by the Complex Components Mapping.

As the Subexpression Factorization and the Complex Components Mapping target DFG optimization, they can be easily integrated with the LAS, MuVoF and FloM. However, the integration and interaction of LAS, MuVoF and FloM are not straightforward. The problem is discussed in Section 7.2.4. In the three proposed techniques, MuVoF gives the best optimization results. Therefore if multi-voltage and multi-frequency designs are applicable, MuVoF is ideal, especially when there are enough throughput and resource slacks. In the fabrication technologies where interconnects are dominant, FloM is able to achieve satisfactory results. Thus it is preferable when longer computation time is allowed. If multi-voltage datapath is not applicable, LAS is efficient for exploring the optimization space based on switching activity.

In summary, several techniques are proposed in this work for low power/energy high level synthesis. These techniques target power/energy reduction by optimizing various parameters such as the switching activity, supply voltage, clock frequency as well as switched capacitance. Different techniques, such as the SA reduction, spurious SA elimination, multi-voltage, frequency scaling, functional pipelining, floorplan-driven as well as high level transformations, are explored. The experiments on test benchmarks have been carried out, and the results obtained show that power/energy can be effectively optimized using the proposed techniques.

7.2 Suggestions for Future Work

This work tries to explore low power/energy high level synthesis in several aspects. Although several techniques are proposed and developed, much more work can be done. The following areas are suggested for future research.

7.2.1 Adopt More Accurate Power/Energy Models

In this work, the proposed techniques use popular but simple power models. To have a high level synthesis platform for real designs, more accurate models must be available. This is because power models are used for power estimation during power optimization process. If the power model is inaccurate, optimization techniques will only generate results that are optimal for the power model used, but may be inaccurate in real designs.

For the SA-based techniques, the construction of SA table is rather important. The two most popular methods to construct SA table is to use simulation or statistics. For the simulation-based methods, accurate stimulating vectors must be available. For the statistics-based methods, accurate statistical models of the input signals must be obtained. Besides, the load capacitance of a module depends on not only the module itself, but also the connections from/to the module. This information must be included to more accurately estimate the switched capacitance of a module during its operation.

In multi-voltage synthesis, the existing works including MuVoF usually assume an average SA of 0.5 and thus do not consider the influence of binding. To more accurately model the datapath, SA must also be taken into consideration. This also engages the consideration of binding, because binding will influence the total SA more when SA is not 0.5.

For the transformation-based techniques, the information of scheduling and binding may be considered. As high level transformations are performed before high level synthesis, no scheduling or binding information are available during the transformations. This may result in rather bad results generated when high level synthesis is performed. To reduce this discrepancy, scheduling and binding information must be estimated during high level transformations.

Apart from the above considerations, the power/energy of the whole circuit may also be more accurately modeled instead of simply summing up the estimated power/energy of all the resources. Note that more accurate power models are usually more time consuming to compute. A balance must be maintained between the computation time and the quality of results. On the development of the proposed techniques, the power models are used in the cost functions. Hence, if more accurate power models are developed, the cost functions can be easily modified accordingly without changing the proposed techniques.

7.2.2 Close the Gap between High Level Synthesis and Physical Synthesis

The modern DSM technology imposes a serious risk on the closure of IC chips. That is to say, even if a chip perfectly passes all functional verification and timing analysis before layout, it may still fail to work after layout. This is because of the fact that the interconnects such as wires become to dominate the area, latency as well as power consumption. On the other hand, current high level synthesis is performed before physical synthesis, thus detailed floorplanning, placement and routing information is not available during synthesis. To close this gap, physical information must be integrated into high level synthesis.

One of the most important issues is how to estimate the wire length. In terms of timing, wires have considerable delays in DSM, which will be accumulated in the delay of datapath. Failure to include the delay of wires leads to inaccurate delay calculation and probable post-layout timing violation. In terms of power, wires also contribute a large amount of the total power in DSM. To achieve power optimization after layout, the power consumed by wires must be taken into account.

Unfortunately, estimation of wire length is not a simple task, because it needs the routing information, which can be obtained only in the later stage of physical synthesis. Researchers have been using floorplan information for wire length estimation in high level synthesis. However, the information may not be sufficient in practice. On the other hand, as wire length is usually calculated many times in high level synthesis, performing detailed routing during high level synthesis is too time consuming. Therefore, it is recommended here that to close the gap between high level synthesis and physical synthesis, global routing [GER99] can be integrated into the high level synthesis process to obtain more accurate wire length estimation. By this means, the delay and energy consumption of wires are well accounted. Modern computers are powerful enough to handle the multiple computations of global routing which is much less time consuming than detailed routing.

7.2.3 Low Power High Level Synthesis for Control Intensive Applications

In this work, the targeted applications are data-intensive applications, in which the input behavioral description is formulated as a DFG. On the other hand, there are also many other applications which contain a large amount of control logic. A typical example is a behavioral description containing many loops and branches. These applications are named the Control Intensive Applications. Typically, the input behavioral description is formulated as a Control-Data Flow Graph (CDFG).

Many DSP and multimedia applications contain a lot of loops. Although many loops can be unrolled and treated as normal descriptions that can be formulated as DFGs, there also exist many loops that cannot be unrolled, e.g. loops of which the boundary is not fixed. On the other hand, unrolling loops usually achieve better latency, but at the cost of more resources and higher power consumption. As a result, many researchers tend to keep the loops and add control logic to control the loops, thus forming a CDFG representation. Optimization in high level synthesis is challenging for CDFGs, because the control logic is difficult to handle, and there are many probability issues in CDFGs. For example, a branch in a CDFG may have a much higher probability to be executed, and its influence, e.g. on switching activity, must be taken into account.

As such, there are many research works in the literature on low power high level synthesis for control intensive applications [CHE02, KHO99, LAK99b, LUO04]. Due to the wide existence of control intensive applications in today's digital applications, and the high complexity of optimizations in low power high level synthesis for these applications, this research area is rather important and prospective.

The proposed multi-voltage and floorplan-driven synthesis techniques can be applied to CDFG with minimal modifications. However, loops need to be carefully handled, using unrolling and partially unrolling techniques [WEI01a, WEI01b, KIT03], which will generate much different structures. On the other hand, SA-based algorithms may behave

differently on CDFGs. For example, it is not beneficial to put a lot of effort to optimize the switching activity of a branch which is seldom executed. As a result, when estimating the switching activity of a solution in the switching activity optimization algorithms, such as the LAS technique, the probability of branches should be considered.

7.2.4 An Integrated Low Power/Energy High Level Synthesis Framework

There has been intensive research work in low power/energy high level synthesis and many techniques have been proposed, including those in the thesis. These techniques usually act as stand-alone, aiming to use different algorithms to address different low power problems. Typically, one technique works well in one condition, but does not necessarily generate sufficiently good results in another condition. In order to setup a synthesis framework, an intuitive idea is to integrate these algorithms.

However, the integration of these techniques is not as simple as it may seem to be, even if they are optimizing different parameters. The reasons are two folds. First, the output of a technique is usually not the input of another technique. Therefore they cannot be simply launched in sequence. Secondly, when searching the design space, a technique may discard the space which is optimal for other techniques, making it difficult to integrate the techniques into a single one. Consider the LAS and the MuVoF techniques. When performing look-ahead, LAS selects the scheduling and binding which is optimal for several local control steps. However, the schedules that are suitable for MuVoF could have been discarded in LAS. Furthermore, it is very difficult to justify which scheduling is better in the subsequent processes.

To achieve a good integration, two steps are required. First, the individual algorithm should be tested more rigorously. More and larger benchmarks and real designs should be tested and the results analyzed before integration. Second, during design space exploration, the solution space must be carefully examined, and a good estimation must be derived to determine which technique should have the priority to choose what it wants.

Furthermore, a complicated algorithm flow may be indispensable to coordinate these techniques.

To integrate the proposed Subexpression Factorization and Complex Components Mapping techniques, the easiest way is to launch them and the proposed synthesis techniques (LAS, MuVoF, FloM) in sequence. However, better results can be achieved if the synthesis techniques are integrated into the transformation techniques for better estimation of energy consumption. To reduce complexity, some low-complexity versions of the proposed synthesis techniques can be developed and used. For example, the LC-LAS can be used instead of the RC-LAS, and the core process of MuVoF can be used instead of the complete algorithm. The LAS technique can be integrated into MuVoF and FloM by introducing some SA-based moves in the iterative algorithms. When a move is performed on the solution (may include only several local control steps) obtained by LAS, it is evaluated with the consideration of estimated switching activity, instead of assuming all the switching activity to be 0.5. FloM can also be integrated in MuVoF. One approach is to adopt the integrated flow proposed in FloM and take the wire energy consumption into account in the iterative process of MuVoF. It is noted that in an integrated algorithm, it should not be expected that all techniques achieve the full potential of their own capability, because the overall optimization effect is the objective of integration.

As power optimization in high level synthesis is a complex problem, it is impossible to have one single technique that provides the best solution. Several techniques are proposed in this work, but more new approaches as well as integration of some of the techniques should be further explored and developed.

Reference

- [AKR89] A. Akritas, *Elements of Computer Algebra with Applications*, John Wiley & Sons Inc, 1989.
- [ALS02] L. Alston and B. Madahar, "From C to netlists: hardware engineering for software engineers," *Electronic & Communication Engineering Journal*, Vol. 14, No. 4, Aug. 2002.
- [ALU02] S. Alupoaei and S. Katkoori, "Net-Based Force-Directed Macrocell Placement for Wirelength Optimization," *IEEE Trans. VLSI*, Vol. 10, No. 6, Dec. 2002.
- [BEL00] M. Bellanger, *Digital Processing of Signals*, John Wiley & Sons, 2000.
- [BEL95] A. Bellaouar and M. Elmasry, *Low-Power Digital VLSI Design*, Kluwer Academic Publishers, 1995.
- [BEN00] L. Benini and G. De Micheli, "System-level power-optimization: Techniques and tools," *ACM Trans. Design Automation of Electronic Systems*, Vol. 5, No. 2, Apr. 2000.
- [BEN02] L. Benini and G. De Micheli, "Networks on a chip: a new SoC paradigm," *Computer*, Vol. 35, No. 1, 2002.
- [BER90] C. Berman, N. Maeda and S. Malik, "Boolean Factoring with Kernels and Rectangle Covering," *ISCAS 1990*, Vol. 4, May 1990.
- [BRI01] M. Bright and T. Arslan, "Synthesis of Low-Power DSP Systems Using a Genetic Algorithm," *IEEE Trans. Evolutionary Computation*, Vol. 5, No. 1, Feb. 2001.
- [CAR03] J. Cardoso and H. Neto, "Compilation for FPGA-Based Reconfigurable hardware," *Design & Test of Computers, IEEE*, Mar.-Apr. 2003.
- [CAS05] M. Casu and L. Macchiarulo, "Throughput-Driven Floorplanning With Wire Pipelining," *IEEE Trans. CAD*, Vol. 24, No. 5, May 2005.
- [CHA00] C. Chantrapornchai, E. Sha and X. Hu, "Efficient Design Exploration Based on Module Utility Selection," *IEEE Trans. CAD*, Vol. 19, No. 1, Jan. 2000.
- [CHA92] A. Chandrakasan, S. Sheng and R. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, Vol. 27, Apr. 1992.

-
- [CHA94] A. Chandrakasan, R. Allmon, A. Stratakos and R. Brodersen, "Design of portable systems," in *Proc. Custom Int. Circuits Conf.* 1994.
- [CHA95a] K. Chao and D. Wong, "Floorplanning for Low Power Designs," *ISCAS 1995*, Vol.1, 1995.
- [CHA95b] J. Chang and M. Pedram, "Register allocation and binding for low power," *32nd ACM/IEEE Design Automation Conference*, 1995.
- [CHA95c] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey and R. Brodersen, "Optimizing Power Using Transformations," *IEEE Trans. CAD*, Vol. 14, No. 1, Jan. 1995.
- [CHA96] J. Chang and M. Pedram, "Module assignment for low power," in *Proc. Design Auto. Conf.*, Sep. 1996.
- [CHA97] J. Chang and M. Pedram, "Energy Minimization Using Multiple Supply Voltages," *IEEE Trans. VLSI*, Vol. 5, No. 4, Dec. 1997.
- [CHA98] C. Chantrapornchai, E. Sha and X. Hu, "Efficient scheduling for imprecise timing based on fuzzy theory," in *Proc. Midwest Symp. Circuits and Systems*, 1998.
- [CHE00] Z. Chen, K. Roy and E. Chong, "Estimation of Power Dissipation Using a Novel Power Macromodeling Technique," *IEEE Trans. CAD*, Vol. 19, No. 11, Nov. 2000.
- [CHE02] C. Chen and M. Sarrafzadeh, "Power-Manageable Scheduling Techniques for Control Dominated High-Level Synthesis," in *Proc. DATE '02*, 2002.
- [CHE03] D. Chen, J. Cong and Y. Fan, "Low-Power High-Level Synthesis for FPGA Architectures," *ISLPED '03*, Aug. 2003.
- [CHE06] D. Chen, J. Cong and J. Xu, "Optimal Simultaneous Module and Multi-voltage Assignment for Low Power," *ACM Trans. Design. Auto. Elec. Sys.*, vol. 11, no. 2, pp. 362-386, Apr. 2006.
- [CHO02] Y. Choi and T. Kim, "An Efficient Low-Power Binding Algorithm in High-Level Synthesis," *ISCAS 2002*, Vol. 4, Jul. 2002.
- [CHU96] J. Chung and K. Parhi, *Pipelined Lattice and Wave Digital Recursive Filters*, Kluwer Academic, 1996.
-

-
- [CON01] G. Constantinides, P. Cheung and W. Luk, "Heuristic Datapath allocation for multiple wordlength systems," in *Proc. Design, Auto. Test*, Mar. 2001.
- [DAS98] A. Dasgupta and R. Karri, "High-Reliability, Low-Energy microarchitecture synthesis," *IEEE Trans. CAD*, Vol. 17, No. 12, Dec. 1998.
- [DAV04] A. Davoodi, V. Khandelwal and A. Srivastava, "Empirical Models for Net-Length Probability Distribution and Applications," *IEEE Trans. VLSI*, Vol. 12, No. 10, Oct. 2004.
- [DAV05] A. Davoodi and A. Srivastava, "Power-Driven Simultaneous Resource Binding and Floorplanning: A Probabilistic Approach," *IEEE Trans. VLSI*, Vol. 13, No. 8, Aug. 2005.
- [DAV88] J. Davenport, Y. Siret, and E. Tournier, *Computer Algebra - Systems and algorithms for algebraic Computation*, Academic Press, 1988.
- [DEM94] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, New York: McGraw-Hill, 1994.
- [DEV91] S. Devadas, K. Keutzer and A. S. Krishnakumar, "Design verification and reachability analysis using algebraic manipulation", in *Proc. ICCD'91*, 1991.
- [DIE00] O. Diessel, H. ElGindy, M. Middendorf, H. Schmeck and B. Schmidt, "Dynamic scheduling of tasks on partially reconfigurable FPGAs," in *Proc. IEE Comput. Digit. Tech.*, Apr. 2000.
- [ELE98] P. Eles, K. Kuchcinski and Z. Peng, *System Synthesis with VHDL*, Kluwer Academic Publishers, 1998.
- [FAN94] Y. Fang and D. Wong, "Simultaneous Functional-Unit Binding and Floorplanning," in *Proc. Int. Conf. CAD*, Nov. 1994.
- [FER89] D. Ferry, L. Akers and E. Greeneich, *Ultra Large Scale Integrated Microelectronics*, Prentice Hall, 1989.
- [FRE00] V. Fresse, O. Deforges and M. Assouil, "Rapid prototyping for mixed architectures," *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, Jun. 2000.
- [GAI98] S. Gailhard, N. Julien, J. Diguët and E. Martin, "How to transform an architectural synthesis tools for lower power designs," in *Proc. Great Lakes Symp. VLSI*, 1998.
-

-
- [GAO00] S. Gao, "Factoring multivariate polynomials via partial differential equations," *Mathematics Subject Classification*, 2000.
- [GAO01] S. Gao, "Absolute Irreducibility of Polynomials via Newton Polytopes," *J. of Algebra*, 2001.
- [GEB92] C. Gebotys and M. Elmasry, *Optimal VLSI architectural synthesis*, Kluwer Academic Publishers, 1992.
- [GER99] S. H. Gerez, *Algorithms for VLSI Design Automation*, Wiley & Sons, 1999.
- [GUO01] P. Guo, T. Takahashi, C. K. Cheng and T. Yoshimura, "Floorplanning Using a Tree Representation," *IEEE Trans. CAD*, Vol. 20, No. 2, Feb. 2001.
- [GUP00] S. Gupta and F. Najm, "Power Modeling for High-Level Power Estimation," *IEEE Trans. VLSI*, Vol. 8, No. 1, Feb. 2000.
- [GUP02] S. Gupta, M. Reshadi, N. Savoiu, N. Dutt, R. Gupta and A. Nicolau, "Dynamic common sub-expression elimination during scheduling in high-level synthesis," *15th Int. Symp. System Synthesis*, Oct. 2002.
- [GUS02] O. Gustafsson and L. Wanhammar, "ILP modelling of the common subexpression sharing problem," *9th Int. Conf. Electronics, Circuits and Systems*, Vol. 3, 2002.
- [HAM01] J. Hammes, A. Bohm, C. Ross, M. Chawathe, B. Draper, R. Rinker and W. Najjar, "Loop fusion and temporal common subexpression elimination in window-based loops," in *Proc. 15th Int. Symp. Parallel and Distributed Processing*, Apr. 2001.
- [HAR00] J. Harkin, T. McGinnity and L. Maguire, "Partitioning methodology for dynamically reconfigurable embedded systems," in *Proc. IEE Comput. Digit. Tech.*, Vol. 147, Aug. 2000.
- [HEN02] J. Henkel and Y. Li, "Avalanche: An environment for design space exploration and optimization of low-power embedded systems," *IEEE Trans. VLSI*, Vol. 10, No. 4, Aug. 2002.
- [HON99] I. Hong, D. Kirovski, G. Qu, M. Potkonjak and M. Srivastava, "Power Optimization of Variable-Voltage Core-Based Systems," *IEEE Trans. CAD*, Vol. 18, No. 12, Dec. 1999.
-

-
- [HUA00] J. Huang, J. Jou and W. Shen, "ALTO: An iterative area/performance tradeoff algorithm for LUT-Based FPGA technology mapping," *IEEE Trans. VLSI*, Vol. 8, No. 4, Aug. 2000.
- [JHA01] N. Jha, "Low Power System Scheduling and Synthesis," *ICCAD 2001*, pp. 259-263, Nov. 2001.
- [JOH97] M. Johnson and K. Roy, "Datapath Scheduling with Multiple Supply Voltages and Level Converters," *ACM Trans. Design Automat. Electron. Syst.*, Vol. 2, No. 3, Jul. 1997.
- [JON95] C. C. Jong and Y. H. Lam, "Using time zones for data path allocation in High-Level synthesis of digital systems," *Int. J. Eelectronics*, 1995.
- [KAL03] E. Kaltofen, "Polynomial Factorization: a Success Story," *ISSAC'03*, Aug. 2003.
- [KHO02] K. Khouri and N. Jha, "Leakage Power Analysis and Reduction During Behavioral Synthesis," *IEEE Trans. VLSI*, Vol. 10, No. 6, Dec. 2002.
- [KHO99] K. Khouri, G. Lakshminarayana and N. Jha, "High-Level synthesis of Low-Power Control-Flow intensive circuits," *IEEE Trans. CAD*, Vol. 18, No. 12, Dec. 1999.
- [KIM96] J. Kim and S. Chae, "New MPEG2 decoder architecture using frequency scaling," in *Proc. IEEE Int. Symp. Circuits and Systems*, 1996.
- [KIT03] S. Kittitornkun and Y. Hu, "Mapping deep nested Do-Loop DSP algorithms to large scale FPGA array structures," *IEEE Trans. VLSI*, Vol. 11, No. 2, Apr. 2003.
- [KNA92] D. Knapp, "Fasolt: A program for feedback-driven data-path optimization," *IEEE Trans. CAD*, Vol. 11, No. 6, Jun. 1992.
- [KOL94] D. Kolson, A. Nicolau and N. Dutt, "Integrating Program Transformations in the Memory-Based Synthesis of Image and Video Algorithms," in *Proc. Int. Conf. CAD*, Nov. 1994.
- [KRU01] L. Kruse, E. Schmidt, G. Jochens, A. Stammermann, A. Schulz, E. Macii and W. Nebel, "Estimation of lower and upper bounds on the power consumption from scheduled data flow graphs," *IEEE Trans. VLSI*, Vol. 9, No. 1, Feb. 2001.
-

- [KUC90] K. Kucukcakar and A. Parker, "Data path tradeoffs using MABAL," in Proc. 27th DAC, Jun. 1990.
- [KUM01] K. Kum and W. Sung, "Combined word-length optimization and high-level synthesis of digital signal processing systems," *IEEE Trans. CAD*, Vol. 20, No. 8, Aug. 2001.
- [KUR02] E. Kursun, A. Srivastava, S. Memik and M. Sarrafzadeh, "Early Evaluation Techniques for Low Power Binding," *ISLPED'02*, Aug. 2002.
- [KUT02] T. Kutzschebauch and L. Stok, "Layout Driven Decomposition with Congestion Consideration," in Proc. *Design, Auto. Test*, 2002.
- [LAK04] G. Lakshminarayana, A. Raghunathan and K. Khouri, "Common-Case Computation: A High-Level Energy and Performance Optimization Technique," *IEEE Trans. CAD*, Vol. 23, No. 1, Jan. 2004.
- [LAK99a] G. Lakshminarayana, A. Raghunathan, N. Jha and S. Dey, "Power Management in High-Level Synthesis," *IEEE Trans. VLSI*, Vol. 7, No. 1, Mar. 1999.
- [LAK99b] G. Lakshminarayana and N. Jha, "FACT: A Framework for Applying Throughput and Power Optimizing Transformations to Control-Flow-Intensive Behavioral Descriptions," *IEEE Trans. CAD*, Vol. 18, No. 11, Nov. 1999.
- [LAK99c] G. Lakshminarayana, K. Khouri and N. Jha, "Wavesched: A Novel Scheduling Technique for Control-Flow Intensive Designs," *IEEE Trans. CAD*, Vol. 18, No. 5, May 1999.
- [LEE89] J. Lee, Y. Hsu, and Y. Lin, "A new Integer Linear Programming formulation for the scheduling problem in data path synthesis," in Proc. *Int. Conf. CAD*, Nov. 1989.
- [LIN97] Y. Lin, C. Hwang and A. Wu, "Scheduling Techniques for Variable Voltage Low Power Designs," *ACM Trans. Design Automat. Electron. Syst.*, Vol. 2, No. 2, Apr. 1997.
- [LIU94] D. Liu and C. Svensson, "Power Consumption Estimation in CMOS VLSI Chips," *IEEE J. Solid-State Circuits*, Vol. 29, No. 6, Jun. 1994.

-
- [LUN01] M. Lundberg, L. Muhammad, K. Roy and S. Wilson, "A Novel Approach to High-Level Switching Activity Modeling With Applications to Low-Power DSP System Synthesis," *IEEE Trans. Signal Processing*, Vol. 49, No. 12, Dec. 2001.
 - [LUO04] J. Luo, L. Zhong, Y. Fei and N. Jha, "Register Binding-Based RTL Power Management for Control-Flow Intensive Designs," *IEEE Trans. CAD*, Vol. 23, No. 8, Aug. 2004.
 - [LYU03] C. Lyuh and T. Kim, "High-Level Synthesis for Low Power Based on Network Flow Method," *IEEE Trans. VLSI*, Vol. 1, No. 3, Jun. 2003.
 - [LYU04] C. Lyuh, T. Kim and K. Kim, "Coupling-Aware High-Level Interconnect Synthesis," *IEEE Trans. CAD*, Vol. 23, No. 1, Jan. 2004.
 - [MAC00] D. MacMillen, M. Butts, R. Camposano, D. Hill and T. Williams, "An Industrial View of Electronic Design Automation," *IEEE Trans. CAD*, Vol. 19, No. 12, Dec. 2000.
 - [MAC98] E. Macii, M. Pedram and F. Somenzi, "High-Level Power Modeling, Estimation, and Optimization," *IEEE Trans. CAD*, Vol. 17, No. 11, Nov. 1998.
 - [MAE01] R. Maestre, F. Kurdahi, M. Fernandez, R. Hermida, N. Bagherzadeh and H. Singh, "A framework of reconfigurable computing: task scheduling and context management," *IEEE Trans. VLSI*, Vol. 9, No. 6, Dec. 2001.
 - [MAK03] W. Mak and E. Young, "Temporal Logic Replication for Dynamically Reconfigurable FPGA partitioning," *IEEE Trans. CAD*, Vol. 22, No. 7, Jul. 2003.
 - [MAN02] A. Manzak and C. Chakrabarti, "A Low Power Scheduling Scheme with Resources Operating at Multiple Voltages," *IEEE Trans. VLSI*, Vol. 10, No. 1, Feb. 2002.
 - [MAR95a] R. Martin and J. Knight, "Power-Profiler: Optimizing ASICs Power Consumption at the Behavioral Level," *ACM/IEEE Design Auto. Conf.*, 1995.
 - [MAR95b] D. Marculescu, R. Marculescu and M. Pedram, "Information theoretic measures for energy consumption at the register-transfer level," in *Proc. Int. Symp. Low-Power Designs*, Apr. 1995.
 - [MAP] Website: <http://www.maplesoft.com>.
-

-
- [MAS03] K. Masselos, P. Merakos, S. Theoharis, T. Stouraitis and C. Goutis, "Power efficient data path synthesis of sum-of-products computations," *IEEE Trans. VLSI*, Vol. 11, No. 3, Jun. 2003.
 - [MCF90] M. McFarland and T. Kowalski, "Incorporation bottom-up design into hardware synthesis," *IEEE Trans. CAD*, Vol. 9, No. 9, Sep. 1990.
 - [MEH97] R. Mehra, L. Guerra and J. Rabaey, "A Partitioning Scheme for Optimizing Interconnect Power," *IEEE J. Solid-State Circuits*, Vol. 32, Mar. 1997.
 - [MIG92] M. Mignotte (Translated by C. Mignotte), *Mathematics for Computer Algebra*, Springer-Verlag New York, Inc, 1992.
 - [MOH03] S. Mohanty and N. Ranganathan, "Energy efficient scheduling for datapath synthesis," in *Proc. Int. Conf. VLSI Design*, Jan. 2003.
 - [MOH04a] S. Mohanty and N. Ranganathan, "A Framework for Energy and Transient Power Reduction During Behavioral Synthesis," *IEEE Trans. VLSI*, Vol. 12, No. 6, Jun. 2004.
 - [MOH04b] S. Mohanty, N. Ranganathan and S. Chappidi, "ILP Models for Energy and Transient Power Minimization during Behavioral Synthesis," in *Proc. IEEE Int. Conf. VLSI*, 2004.
 - [MOH05] S. Mohanty and N. Ranganathan, "Simultaneous Peak and Average Power Minimization During Datapath Scheduling," *IEEE Trans. Circuits and Systems*, Vol. 52, No. 6, Jun. 2005.
 - [MUR03] A. Murugavel and N. Ranganathan, "A Game Theoretic Approach for Power Optimization During Behavioral Synthesis," *IEEE Trans. VLSI*, Vol. 11, No. 6, Dec. 2003.
 - [NAR98] U. Narayanan, P. Pan and C. Liu, "Low power logic synthesis under a general delay model," in *Proc. Low Power Electron. Design*, Aug. 1998.
 - [NEM96] M. Nemani and F. Najm, "Toward a high-level power estimation capability," *IEEE Trans. CAD*, Vol. 15, No. 6, Jun. 1996.
 - [NEM99] M. Nemani and F. Najm, "High-Level Area and Power Estimation for VLSI Circuits," *IEEE Trans. CAD*, Vol. 18, No. 6, Jun. 1999.
 - [NIC91] A. Nicolau and R. Potasman, "Incremental Tree Height Reduction for High Level Synthesis," in *Proc. Design Auto. Conf.*, Jun. 1991.
-

-
- [OTT82] R. Otten, "Layout Structures," in Proc. *IEEE Large Scale Systems Symp.* 1982.
 - [PAN99] P. Panda and N. Dutt, "Low-Power memory mapping through reducing address bus activity," *IEEE Trans. VLSI*, Vol. 7, No. 3, Sep. 1999.
 - [PAP82] C. Papadimitriou and K. Steiglitz, *Combinational Optimization: Algorithms and Complexity*, Prentice-Hall, 1982.
 - [PAP99] C. Papachristou, M. Nourani and M. Spining, "A Multiple Clocking Scheme for Low-Power RTL Design," *IEEE Trans. VLSI*, Vol. 7, No. 2, Jun. 1999.
 - [PAS99] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde and D. Durackova, "A new algorithm for elimination of common subexpressions," *IEEE Trans. CAD*, Vol. 18, No. 1, Jan. 1999.
 - [PAU89] P. Paulin and J. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," *IEEE Trans. CAD*, Vol. 8, No. 6, 1989.
 - [PEY01a] A. Peymandoust and G. De Micheli, "Using Symbolic Algebra in Algorithmic Level DSP Synthesis," in Proc. *Design Auto. Conf.*, 2001.
 - [PEY01b] A. Peymandoust and G. De Micheli, "Symbolic Algebra and Timing Driven Data-Flow Synthesis," *ICCAD 2001*, Nov. 2001.
 - [PEY03a] A. Peymandoust and G. De Micheli, "Application of Symbolic Computer Algebra in High-Level Data-Flow Synthesis," *IEEE Trans. CAD*, Vol. 22, No. 9, Sep. 2003.
 - [PEY03b] A. Peymandoust, T. Simunic and G. De Micheli, "Complex Instruction and Software Library Mapping for Embedded Software Using Symbolic Algebra," *IEEE Trans. CAD*, Vol. 22, No. 8, Aug. 2003.
 - [PRA99] P. Prabhakaran P. Banerjee, J. Crenshaw and M. Sarrafzadeh, "Simultaneous Scheduling, Binding and Floorplanning for Interconnect Power Optimization," in Proc. *Int. Conf. VLSI Design*, 1999.
 - [RAG01] V. Raghunathan, S. Ravi, A. Raghunathan and G. Lakshminarayana, "Transient Power Management Through High Level Synthesis," *ICCAD 2001*, Nov. 2001.
 - [RAG03] A. Raghunathan, S. Dey and N. Jha, "High-Level Macro-Modeling and Estimation Techniques for Switching Activity and Power Consumption," *IEEE Trans. VLSI*, Vol. 11, No. 4, Aug. 2003.
-

- [RAG94] A. Raghunathan and N. Jha, "Behavioral synthesis for low power," in Proc. *ICCD'94*, Oct. 1994.
- [RAG97] A. Raghunathan and N. Jha, "SCALP: An Iterative-Improvement-Based Low-Power Data Path Synthesis System," *IEEE Trans. CAD*, Vol. 16, No. 11, Nov. 1997.
- [ROB99] X. Roblot "Polynomial Factorization Algorithms over Number Fields," *J. Symbolic Computation*, 1999.
- [ROT00] J. Rotman, *A First Course in Abstract Algebra*, 2nd edition, Prentice-Hall Inc, 2000.
- [SCH05] M. Schmitz, B. Al-Hashimi and P. Eles, "Cosynthesis of Energy-Efficient Multimode Embedded Systems with Consideration of Mode-Execution Probabilities," *IEEE Trans. CAD*, Vol. 24, No. 2, Feb. 2005.
- [SHE05] N. Shekhar, P. Kalla, F. Enescu and S. Gopalakrishnan, "Exploiting vanishing polynomials for equivalence verification of fixed-size arithmetic datapaths," in Proc. *ICCD 2005*, 2005.
- [SHE97] Z. Shen and C. C. Jong, "Exploring module selection space for architectural synthesis of low-power designs," in Proc. *Int. Symp. Circuits and Systems*, 1997.
- [SHI00a] W. Shiue and C. Chakrabarti, "Low-Power Scheduling with Resources Operating at Multiple Voltages," *IEEE Trans. Circuits and Systems*, Vol. 47, No. 6, Jun. 2000.
- [SHI00b] W. Shiue and C. Chakrabarti, "ILP-based Scheme for Low-Power Scheduling and Resource Binding," in Proc. *IEEE Int. Symp. Circuits and Systems*, Vol. 3, 2000.
- [SMI01] J. Smith and G. De Micheli, "Polynomial Circuit Models for Component Matching in High-Level Synthesis," *IEEE Trans. VLSI*, Vol. 9, No. 6, Dec 2001.
- [SMI98] J. Smith and G. De Micheli, "Polynomial methods for component matching and verification," in Proc. *ICCAD*, Nov. 1998.
- [SMI99] J. Smith and G. De Micheli, "Polynomial methods for allocating complex components," in Proc. *Design Auto. Test Eur. Conf.*, Mar. 1999.

-
- [SPR94] D. Springer and D. Thomas, "Exploiting the special structure of conflict and compatibility Graphs in High-Level Synthesis," *IEEE Trans. CAD*, Vol. 13, No. 7, Jul. 1994.
 - [SRI01] V. Srinivasan, S. Govindarajan and R. Vemuri, "Fine-Grained and Coarse-Grained Behavioral Partitioning with effective utilization of memory and design space exploration for multi-FPGA architectures," *IEEE Trans. VLSI*, Vol. 9, No. 1, Feb. 2001.
 - [STA03] A. Stammermann, D. Helms, M. Schulte, A. Schulz and W. Nebel, "Binding, Allocation and Floorplanning in Low Power High-Level Synthesis," *ICCAD'03*, Nov. 2003.
 - [STA94] T. Stanion and C. Sechen, "Boolean Division and Factorization Using Binary Decision Diagrams," *IEEE Trans. CAD*, Vol. 13, No. 9, Sep. 1994.
 - [STO83] L. Stockmeyer, "Optimal Orientations of Cells in Slicing Floorplan Designs," *Information and Control*, Vol. 59, 1983.
 - [SYN] Website: <http://www.synopsys.com>.
 - [TAK98] M. Takahashi, M. Hamada, T. Nishikawa, H. Arakida, T. Fujita, F. Hatori, S. Mita, K. Suzuki, A. Chiba, T. Terazawa, F. Sano, Y. Watanabe, K. Usami, M. Igarashi, T. Ishikawa, M. Kanazawa, T. Kuroda and T. Furuyama, "A 60mW MPEG-4 video codec using clustered voltage scaling with variable supply voltage scheme," *IEEE J. Solid-State Circuits*, Vol. 33, No. 11, Nov. 1998.
 - [TAN06] X. Tang, R. Tian and M. Wong, "Minimizing Wire Length in Floorplanning," *IEEE Trans. CAD*, Vol. 25, No. 9, Sep. 2006.
 - [TSE86] C. Tseng and D. Siewiorek, "Automated synthesis of data paths in digital systems," *IEEE Trans. CAD*, Vol. 5, No. 3, Jul. 1986.
 - [USA98] K. Usami, M. Igarashi, F. Minami, T. Ishikawa, M. Kanazawa, M. Ichida and K. Nogami, "Automated Low-Power Technique Exploiting Multiple Supply Voltages Applied to a Media Processor," *IEEE J. Solid-State Circuits*, Vol. 33, No. 3, Mar. 1998.
 - [WAN04] W. Wang, A. Raghunathan, G. Lakshminarayana and N. Jha, "Input Space Adaptive Design: A High-Level Methodology for Optimizing Energy and Performance," *IEEE Trans. VLSI*, Vol. 12, No. 6, Jun. 2004.
-

- [WAN96] H. Wang, A. Nicolau and K. Siu, "The strict time lower bound and optimal schedules for parallel prefix with resource constraints," *IEEE Trans. Comput.*, Vol. 45, No. 11, Nov. 1996.
- [WEI99] M. Weinhardt and W. Luk, "Pipeline vectorization for reconfigurable systems," in Proc. *IEEE Symp. FCCAM'99*, Apr. 1999.
- [WEI01a] M. Weinhardt and W. Luk, "Memory access optimization for reconfigurable systems," in Proc. *IEE Comput. Digit. Tech.*, Vol. 148, No. 3, May. 2001.
- [WEI01b] M. Weinhardt and W. Luk, "Pipeline Vectorization," *IEEE Trans. CAD*, Vol. 20, No. 2, Feb. 2001.
- [WIL00] A. Williams, A. Brown and M. Zwolinski, "Simultaneous optimization of dynamic power, area and delay in behavioural synthesis," in Proc. *IEE Comput. Digit. Tech.*, Vol. 147, No. 6, Nov. 2000.
- [WOL98] W. Wolf, *Modern VLSI Design, Systems on Silicon*, Prentice Hall, 1998.
- [WON86] D. Wong and C. Liu, "A new Algorithm for Floorplan Design," in Proc. *IEEE/ACM Design Auto. Conf.*, 1986.
- [WU03] D. Wu, B. Al-Hashimi and P. Eles, "Scheduling and mapping of conditional task graph for the synthesis of low power embedded systems," *IEE Proc. Comput. Digit. Tech.*, Vol. 150, No. 5, Sep. 2003.
- [XU97] M. Xu and F. Kurdahi, "Layout-Driven RTL Binding Techniques for High-Level Synthesis Using Accurate Estimators," *ACM Trans. Design Automation of Electronic Systems (TODAES)*, Vol. 2, Oct. 1997.
- [ZHO02] L. Zhong and N. K. Jha, "Interconnect-aware High-level Synthesis for Low Power," *ICCAD'02*, Nov. 2002.
- [ZHO05] L. Zhong and N. K. Jha, "Interconnect-Aware Low-Power High-Level Synthesis," *IEEE Trans. CAD*, Vol. 24, No. 3, Mar. 2005.