

國立交通大學  
電機與控制工程研究所  
博士論文

有效利用資源之低功率數位訊號處理設計  
On Resource-Efficient Low-Power VLSI Signal  
Processing Design

研究生：楊學之  
指導教授：董蘭榮 博士

中華民國九十六年七月

有效利用資源之低功率數位訊號處理設計  
On Resource-Efficient Low-Power VLSI Signal  
Processing Design

研究生：楊學之

Student: Hsueh-Chih Yang

指導教授：董蘭榮

Advisor: Lan-Rong Dung

國立交通大學

電機與控制工程學系

博士論文

A Thesis

Submitted to Department of Electrical and Control Engineering

College of Electrical Engineering and Computer Science

National Chiao Tung University

in Partial Fulfillment of The Requirements

for The Degree of

Doctor of Philosophy

in

Electrical and Control Engineering

July 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年七月

## 推薦函

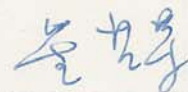
- 一、事由：推薦電機與控制工程系博士班研究生楊學之提出論文以參加國立交通大學博士論文口試。
- 二、說明：本校電機與控制工程系博士班研究生楊學之已完成博士班規定之學科及論文研究訓練。  
有關學科部分，楊君以修必應修學分（請查學籍資料），通過資格考試；有關論文方面，楊君已完成“有效利用資源之低功率數位訊號處理設計”初稿。其論文“Lan-Rong Dung and Hsueh-Chih Yang, "A Parallel-In Folding Technique for High-Order FIR Filter Implementation," IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E89-A, No.12, pp. 3659-3665, Dec. 2006”，“Lan-Rong Dung and Hsueh-Chih Yang "On Multiple-Voltage High-Level Synthesis Using Algorithmic Transformations," IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E87-A, No.12, pp. 3100-3108, Dec. 2004”等期刊接受。另有論文“A Channel-Aware Turbo Decoder with Early Give-Up Technique for Mobile Communications.”，“Algorithmic Transformations and Peak Power Constraint Applied to Multiple-Voltage Low-Power VLSI Signal Processing.”分別發表於研討會或以投稿於期刊正接受審查中（請參閱博士論文著作目錄）。
- 三、總言之，楊君已具備國立交通大學電機與控制工程系博士班研究生應有之教育及訓練水準，因此推薦楊君參加國立交通大學電機與控制工程系博士論文口試。

此致

國立交通大學電機與控制工程學系

電機與控制工程學系教授

董蘭榮



中華民國 96 年 6 月

# 國立交通大學

## 論文口試委員會審定書

本校 電機與控制工程 學系博士班 楊學之 君

所提論文：有效利用資源之低功率數位訊號處理設計  
合於博士資格水準，業經本委員會評審認可。

口試委員：林函峰 \_\_\_\_\_  
林進火 \_\_\_\_\_  
葉經緯 \_\_\_\_\_  
\_\_\_\_\_ \_\_\_\_\_  
\_\_\_\_\_ \_\_\_\_\_

指導教授：葉其華 \_\_\_\_\_

系主任：邱俊誠 \_\_\_\_\_

中華民國 96 年 7 月 5 日

Department of Electrical and Control Engineering  
National Chiao Tung University  
Hsinchu, Taiwan R.O.C.

Date: July 5, 2007

We have carefully read the dissertation entitled On Resource-Efficient Low-Power VLSI Signal Processing Design submitted by Hsueh-Chih Yang in partial fulfillment of the requirements of the degree of **DOCTOR OF PHILOSOPHY** and recommend its acceptance.

J. L. K.

Yang Yang

Ch-Jay L.

Chen

J. P. A.

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

Thesis Advisor: Yang Yang

Chairman: Chou, jin-chen

Copyright

by

Hsueh-Chih Yang

2007

To my family and Ame

# Acknowledgments

First, I would like to thank my advisor, Professor Lan-Rong Dung, for his guidance and support throughout my graduate studies at National Chiao-Tung University. This work would not have been possible without the most inspiring discussions we have had in the past five years.

Also, I would like to thank the Committee members: Professor Youn-Long Steven Lin, Professor Chin-Teng Lin, Professor Jwu-Sheng Hu, and Professor Ching-Wei Yeh for helpful suggestions and comments helping to improve the presentation of this work.

I would like to gratefully acknowledge precedent contributions from Chih-Kai Chang, Shu-Der Lan, and Ming-Feng Yang. I have furthermore to thank all SoCLAB members for sharing with me in distress and joy in the past five years.

Especially, my deepest gratitude goes to my parents and I-I whose patient love enabled me to complete this work. Their love, care and patience words cannot express.

HSUEH-CHIH YANG

*National Chiao-Tung University*

*July 2007*



# On Resource-Efficient Low-Power VLSI Signal Processing Design

Hsueh-Chih Yang, Ph.D.

National Chiao-Tung University, 2007

Advisor: Lan-Rong Dung



The primary design objective of computing and communication systems has been targeting on the following performance metrics: the speed of signal processing, the rate of communications, and the optimization of quality of service. For portable embedded computing systems and wireless systems deployed on a large scale and untethered to power sources, practical considerations dictate a different design regime, one that should be dominated by energy and cost constraints. Batteries are serving as a dedicated energy resource. The requirement of portability places severe restrictions on size and weight, which in turn limits the amount of energy that is continuously available to maintain system operability. For these reasons, a fundamental shift in design paradigm is necessary: from focusing on performance

to focusing on constraints, from maximizing data rate to maximizing resource efficiency. This dissertation illuminates the impact of resource constraints on the design methodologies of VLSI signal processing and communication applications, proposes several design methodologies in the resource-constrained low-power high-level synthesis (HLS), the limited-resource folding techniques, and the power efficient turbo decoder, and tries to stimulate interests in the VLSI signal processing in reformulating and revisiting classic VLSI signal processing problems under new constraints and exploring the role of signal processing in exciting new applications. Based on the proposed techniques, we have designed and implemented two power-efficient chips, a pulse shaping FIR filter for WCDMA and a limited-resource DWT processor.



# 有效利用資源之低功率數位訊號處理設計

## 中文摘要

隨著可攜式電子產品需求的增加，例如筆記型電腦、數位通訊設備、數位視訊設備和音訊播放器等，能量與成本的考量變得非常重要。可攜式電子產品有尺寸上的限制，進而限制了電池電量與使用時間。這些原因讓數位電路設計的目標有了改變，從追求效能到專注在限制上；從追求最大的處理速度到追求更有效地使用資源。此論文闡述在有限的資源情況下，如何有效地去最佳化電路設計的面積與達到低功率的設計，我們提出了一個有限資源情況下之低功率高階合成方法之外，並且提出了在有限資源情況下之硬體折疊架構與一個低功率的渦輪碼解碼方法。本論文中提出之有限資源低功率高階合成方法與有限資源之硬體折疊架構在與相關國際期刊文獻比較之後發現在功率消耗上確實有較佳的表現。有限資源低功率高階合成方法的相關文獻中，極少有使用到演算法轉換去最佳化排程結果的文獻，使用此演算法轉換的效果非常顯著。所提出之有限資源之硬體折疊架構也是目前文獻中最節省暫存器與最少暫存器切換的最佳設計。

# Contents

<b>Acknowledgments</b>	<b>iv</b>
<b>Abstract</b>	<b>v</b>
<b>List of Tables</b>	<b>x</b>
<b>List of Figures</b>	<b>xi</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
<b>Chapter 2 Background Review</b>	<b>7</b>
2.1 Low-Power ASIC Design . . . . .	7
2.1.1 Fundamental Design Decisions . . . . .	8
2.1.2 System and Algorithmic Level . . . . .	9
2.1.3 Architectural Optimization . . . . .	14
2.1.4 Logic Level . . . . .	19
2.1.5 Transistor Level . . . . .	24
2.1.6 Summary . . . . .	27
<b>Chapter 3 Resource-Constrained Low-Power Scheduling in HLS</b>	<b>30</b>
3.1 Introduction . . . . .	30

3.2	Overview of Basic Scheduling Technique and Algorithmic Transformations . . . . .	31
3.2.1	Basic Scheduling Techniques . . . . .	31
3.2.2	Fully-Specified Flow Graph . . . . .	32
3.3	Proposed Approach . . . . .	34
3.3.1	<i>Shrink(graph)</i> . . . . .	36
3.3.2	<i>Minimize_MASP(graph)</i> . . . . .	37
3.3.3	<i>MVS(graph, R<sub>u</sub>, T<sub>u</sub>, L)</i> . . . . .	37
3.3.4	<i>LC_refine(S)</i> . . . . .	39
3.4	Experimental Results . . . . .	42
3.5	Summary . . . . .	43
<b>Chapter 4 Limited-Resource Folding Techniques</b>		<b>48</b>
4.1	Limited-Resource DWT Processor . . . . .	50
4.1.1	Introduction . . . . .	50
4.1.2	Conventional DWT VLSI architecture . . . . .	51
4.1.3	Limited-resource FIR filtering . . . . .	53
4.1.4	Scheduling Algorithm of DWT IP . . . . .	57
4.1.5	Limited-Resource DWT Architecture . . . . .	68
4.1.6	Implementation of DWT IP . . . . .	71
4.1.7	Summary . . . . .	72
4.2	Folding Technique for High-Order FIR Filter Implementation . . . . .	73
4.2.1	Introduction . . . . .	73
4.2.2	Candidates of Folding Techniques . . . . .	75
4.2.3	Existing Folding Techniques . . . . .	75
4.2.4	Comparison Results . . . . .	81
4.2.5	Summary . . . . .	87

<b>Chapter 5 Power Efficient Turbo Decoding</b>	<b>90</b>
5.1 Introduction . . . . .	90
5.2 Early Give-Up Decision . . . . .	91
5.3 State Reuse Mechanism . . . . .	93
5.4 Proposed Turbo Decoding Flow . . . . .	94
5.5 Simulation Results . . . . .	95
<b>Chapter 6 Conclusion</b>	<b>98</b>
<b>Bibliography</b>	<b>102</b>



# List of Tables

3.1	Comparison results of third-order IIR filter with resource constraint RC1 and a timing constraint of 16 control steps. . . . .	43
3.2	Power consumption and reduction of benchmarks by applying retiming transformations only. . . . .	46
3.3	Power consumption and reduction of benchmarks by the proposed scheduling algorithm. . . . .	47
4.1	IS-95 WCDMA pulse shaping FIR filter specification. . . . .	88
4.2	Area and power consumption comparisons.(An IS-95 WCDMA pulse shaping 33-tap FIR) . . . . .	89
4.3	Features of the IS-95 WCDMA pulse shaping FIR filter chip. . . . .	89
5.1	Iteration reduction rate of the proposed turbo decoding (comparing with conventional turbo decoding with the Magic Genie Rule). . . . .	97

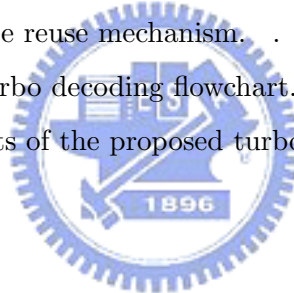
# List of Figures

3.1	FSFG of second-order IIR filter. . . . .	33
3.2	Loop shrinking of second-order IIR. (a) The original FSFG. (b) The equivalent FSFG. . . . .	34
3.3	Unfolding result. (a) An example of FSFG that cannot achieve IPB. (b) A rate-optimal FSFG using unfolding. . . . .	34
3.4	Schedules of second-order IIR before retiming. . . . .	37
3.5	Schedules of second-order IIR after retiming. . . . .	37
3.6	Schedules of second-order IIR after unfolding. . . . .	38
3.7	Flowchart of multiple voltage scheduling. ( <sup>1</sup> In the class $c$ , is the resource with $v(j)$ -voltage available, is the cycle power consumption under the peak power bound, and $T_s + T_c(v(j)) \leq T_L$ ?) . . . . .	40
3.8	Examples of level converters. . . . .	41
3.9	Scheduling results of second-order IIR filter with resource constraint RC2. (a)Without peak power bound. (b)With peak power bound. . . . .	44
3.10	Scheduling results of fifth-order EW filter with resource constraint RC2. (a)Without peak power bound. (b)With peak power bound. . . . .	44
3.11	Cycle power consumption of different benchmarks with resource constraint RC2. . . . .	45
4.1	A waterfall-like algorithm containing 6 processing nodes. . . . .	49



4.2	Resource sharing of grouping. . . . .	49
4.3	Resource sharing of segmentation. . . . .	49
4.4	The design flow of DWT SIP synthesizer. . . . .	52
4.5	A typical filter-based DWT architecture. . . . .	52
4.6	The flow graph of FIR filtering. . . . .	54
4.7	The retimed flow graph of FIR filtering. . . . .	55
4.8	The folded scheduling of FIR filtering. . . . .	55
4.9	The 2-split FSFG of 5-tap FIR filtering. . . . .	57
4.10	The scheduling of FIR filtering. . . . .	58
4.11	The DWT block diagram. . . . .	59
4.12	DM scheduling algorithm. . . . .	61
4.13	The block diagram of IDWT. . . . .	65
4.14	Four scheduling Matrices. . . . .	67
4.15	The data flow in DM. . . . .	67
4.16	The data with rearranged indexing. . . . .	68
4.17	The limited-resource DWT architecture. . . . .	69
4.18	The output register bank. . . . .	69
4.19	The coefficient and input register banks . . . . .	70
4.20	The feedback and Data register banks . . . . .	70
4.21	The Layout of DWT processor . . . . .	73
4.22	A folding example of FIR filter. (a) A 6-tap FIR filter in the transposed form. (b) The folded architecture of Fig. 4.22(a) by using the technique presented in [87]. . . . .	76
4.23	The diagram of the folded bit-plane FIR architecture. . . . .	77
4.24	(a) The SFG of the reformulated $K$ -tap FIR filter. (b) The serial-in folded architecture. . . . .	78
4.25	The $r$ -split FIR filtering. . . . .	80

4.26	The scheduling of FIR coefficients for the parallel-in folded technique.	80
4.27	(a) The architecture of the parallel-in folded FIR filter, and (b) the timing diagram. . . . .	81
4.28	The scheduling of FIR filtering. . . . .	82
4.29	Number of DFFs of folded architectures (in log scale) . . . . .	86
4.30	Access number of DFFs per iteration (in log scale) . . . . .	87
4.31	Number of 1-bit 2-to-1 multiplexers of folded architectures (in log scale)	88
4.32	Photomicrograph of IS-95 WCDMA pulse shaping FIR filter chip. . .	89
5.1	Turbo decoder scheme. . . . .	92
5.2	The trends of the mean of the absolute extrinsic information for solvable and unsolvable packets. . . . .	93
5.3	The average of the required iterations for turbo decoding process with and without state reuse mechanism. . . . .	94
5.4	The proposed turbo decoding flowchart. . . . .	95
5.5	Simulation results of the proposed turbo decoding. . . . .	97



# Chapter 1

## Introduction

A great deal of current research is motivated by the need for decreased power dissipation while satisfying requirement for increased computing capacity. The portable consumer electronics market is constantly demanding more powerful capabilities, smaller and lighter products, and longer battery service lifetime. The battery service lifetime of a mobile embedded system is a major concern for hardware/software designers. Attempts for extending the battery lifetime have traditionally focused on minimizing the power consumption of the circuits. However, even in non-portable systems such as scientific workstations, power is still a serious constraint due to limits on heat dissipation. Another important reason to develop low power techniques is the environmental concerns. According to a U.S. Environmental Protection Agency (EPA) report, 80% of the total office equipment electricity consumption is due to computing equipment, a large part of which is due to such equipment consuming power even when unused [1]. This led to the launching of efforts such as the EPA's Energy Star program [2], which outlines requirements for power-efficient PCs. Therefore, the viewpoint of resource efficient implementation, such as limited-resource issue, low-power issue, and power optimization issue, should be certainly added in modern VLSI signal processing methodologies.

With increasing demand of portable devices, the reduction of power consumption has become the essential issue in VLSI design. [45] and [46] described that decisions during high-level synthesis (HLS) have a profound impact on the power consumption of the final design. Hence, [46], [47], [48], and [49] have addressed on power saving techniques, such as voltage scaling, capacitance reduction and switching minimization for HLS. However, these papers are based on a single voltage supply for power minimization and cannot take full advantage of available schedule slacks to reduce the voltage. Therefore, the use of multiple supply voltages becomes very attractive to low power design recently, such as [50], [51], [52], [53], [54], [55], [57], [58], and [59]. The idea is to assign non-critical tasks to low-voltage components and execute time-critical tasks at higher supply voltage. In [46], [52], [53], [57], and [58], the multiple-voltage scheduling method for power optimization of HLS using either integer linear programming (ILP) or dynamic programming were presented. However, both approaches have pseudo-polynomial or even exponential time complexity. In [55], Shiue and Chakrabarti present a list-based multiple-voltage scheduling algorithm with polynomial-time complexity. The algorithm is driven by three parameters: depth, mobility, and switching capacitance. With considering the level converters, [55] provides effective resource-constrained and latency-constrained schemes for multiple-voltage HLS. From Chakrabarti's group, later on, [59] uses the Lagrange multiplier method to find the optimal solution of multiple-voltage scheduling under both resource and latency constraints.

The papers mentioned above have presented efficient scheduling for multiple-voltage HLS. Yet, few papers have considered the effect of algorithmic transformations on multiple-voltage power minimization. We present a multiple-voltage high-level synthesis methodology that minimizes power dissipation of VLSI signal processing. By applying algorithmic transformations, the proposed approach optimizes the power saving, in terms of the average power and peak power, for DSP

applications when the resources and the latency are constrained. Our approach is motivated by the maximization of task mobilities. The mobility is defined as the distance between its as-late-as-possible (ALAP) schedule time and its as-soon-as-possible (ASAP) schedule time. The increase of mobilities may raise the possibility of assigning tasks to low-voltage components. To earn task mobilities, we use loop shrinking, retiming and unfolding techniques. The loop shrinking can reduce the iteration period bound (IPB), while the others are employed for shortening the minimum achieved sample period (MASP) as much as possible. The minimization of MASP implies high task mobilities. Thereafter, we can assign tasks with high mobilities to low-voltage components and minimize energy dissipation under resource and latency constraints. With considering the overhead of level conversion and the minimization of peak power, the proposed methodology has low complexity and can achieve significant power reduction.

Moreover, this dissertation presents the limited-resource problem which arises when the number of atomic units is constrained. The limited-resource problem has been becoming one of the most important issues in system-on-chip (SOC) design [60]. Roughly speaking, operation scheduling determines the cost-speed tradeoffs of the design. If the design is subject to a speed constraint, the scheduling algorithm will attempt to parallelize the operations to meet the timing constraint. Conversely, if there is a limited one on the cost (area or energy), the scheduler will serialize operations to meet the resource constraint. Once the operations are scheduled and the architectures are determined, the number and types of function units, the lifetimes of variables, and the timing constraints are fixed. Thus a good scheduler is very important to an automated datapath synthesis system [3, 4]. A limited-resource discrete wavelet transformation (DWT) architecture has been proposed as a case of dealing with the limited-resource implementation. Furthermore, because FIR filtering is an essential function in most DSP applications, such as telecommunication and mul-

timedia systems. Its guaranteed stability and simple structure make FIR itself a popular technique for removing unwanted parts of signal. For quality-sensitive applications, the number of FIR taps is normally large, ranges from tens to hundreds. Therefore, long-length (or high-order) FIR filters may result in costly hardware and hence severe power consumption problem. Besides these reasons, FIR filtering is also one of waterfall-like processing procedures which means all processing tasks in the FIR filtering are in a linear progression. Therefore, two proposed folding techniques, demonstrated by a high-order FIR filter, have been presented to illustrate, compare, and conclude the tradeoffs between the limited number of processing elements and performance. Two systematic folding techniques are derived from the idea of resource sharing. One of them involves the resource sharing of grouping, and the other involves the resource sharing of segmentation.

Finally, to achieve power resource efficiency, a power efficient turbo decoder has been designed to reduce redundant iterations in noisy channel and minimize the iteration when the channel becomes better by monitoring the extrinsic information and reuses the *a-prior* LLR of previous decoding process as the initial condition for the resend packet. To efficiently save the notoriously high power dissipation of turbo decoder, literature has presented numbers of early stopping mechanisms. The early stopping mechanisms can be done by early termination or early give-up steps. The early termination ends the search of turbo decoder for solvable packets beforehand, while the early give-up ceases the turbo decoding for unsolvable packets. Many papers have proposed early termination approaches which can be categorized into three classes: soft-bit decision [93, 94], hard-bit decision [95, 96], and extra-checking policy [97, 98]. However, very few papers target on early give-up techniques. The early give-up is particularly important for channels with low signal-to-noise ratio (SNR), in that the early give-up allows the decoder to stop the decoding process for unsolvable packets as early as possible and hence minimizing the number of redundant

iterations. The reduction of MAP iterations implies to save the power dissipation of turbo decoding. It is worth noting that, without early give-up techniques, the ARQ or HARQ protocols will not enable the resend request mechanism for an unsolvable packet until the turbo decoding process reaches the maximum number of iterations. The paper [99] by Buckley and Wicker presents an early give-up technique using a neural network to predict turbo decoding errors. Their technique can improve reliability and throughput performance at a lower average decoding complexity than turbo decoding with CRC-based early termination.

The object of this chapter is to design a channel-aware turbo decoder. The turbo decoder can reduce redundant iterations in noisy channel and minimize the iterations when the channel becomes better. We present an early give-up technique with a state reuse mechanism. The proposed technique detects turbo decoding errors by monitoring the extrinsic information and reuses the *a-priori* LLR of previous decoding process as the initial condition for the resend packet. First, when the extrinsic information oscillates without significant increases as the number of iterations increases, the decoded packet is most likely an unsolvable packet. So, once the oscillation of extrinsic information is detected, the early give-up will stop the decoding process and register the *a-priori* LLR from wasting further power consumption. The ARQ or HARQ protocols will thereafter trigger the resend request. When the turbo decoder starts decoding the resend packet, the registered *a-priori* LLR will be reused as the initial condition. The reuse of the *a-priori* information is called the state reuse mechanism. When the channel status recovers to better situation with higher SNR in a fading environment, the state reuse mechanism can further reduce the required iterations and the computational load.

The rest of this dissertation is organized as follows. In Chapter 2, we briefly review the related work in low-power ASIC design. A resource-constrained low-power scheduling algorithm in HLS has been introduced in Chapter 3. Two limited-

resource folding techniques have been discussed in Chapter 4. Chapter 5 outlines a power efficient turbo decoder. Chapter 6 summarizes the contribution of this dissertation.





# Chapter 2

## Background Review

### 2.1 Low-Power ASIC Design

The development of low power integrated systems requires several fundamental design decisions to be taken and a combination of different power optimization techniques to be applied to the system or to parts thereof. Throughout the last ten years, numerous approaches to low power design have been proposed. These include software as well as hardware optimization strategies. Regarding hardware optimization, further distinction can be made between techniques that are intended for the design of logic circuits and techniques that are specific to memory. There are a large number of low power design techniques frequently discussed in the literature. It is evident that, in a hierarchical design flow of ASIC, power reduction can be achieved at all levels of abstraction. Although high-level power optimization is believed to be most effective, the improvements that can be achieved at the lower levels are none the less significant [8]. Thus, any low power design methodology should include a set of high- and low-level optimization techniques that complement one another. Very few low power design techniques have been established as standard (state-of-the-art) techniques in the development of real applications. Others

have proven to be feasible in experimental designs. Many techniques, however, still are of purely academic importance. For the implementation and evaluation of a new optimization technique, it is important to identify those state-of-the-art techniques (at the same or at a different level of abstraction) that may come into conflict with the new method or may have an impact on the effectiveness of the novel technique.

### 2.1.1 Fundamental Design Decisions

The development of electronic systems usually starts with the specification. At this early stage in the design process, all the information required for developing a working product that fits into a specific market segment is gathered. This includes the functionality, the performance, and the type of power supply. From this information, conclusions regarding the power consumption can be drawn and appropriate constraints can be derived. For instance, if a battery was chosen for power supply, the power consumption must be minimized in order to allow for a reasonable system running time. In the case of very high performance ICs, the power consumption must also be constrained in order to prevent thermal failure. Clearly, the more demanding the specification, the more design and optimization effort is required for meeting the power constraints. Therefore, the specification should always strictly reflect the actual requirements of the application. The fabrication technology also has to be chosen at this stage, i.e. before entering the actual design process. A suitable choice can usually be determined on the basis of the specification and any derived constraints. Mainstream bulk CMOS technologies enable high integration density and high performance at low cost and, at the same time, keep the power consumption at a moderate level. Also, many power optimization techniques can be applied to bulk CMOS designs. For these reasons, bulk CMOS is and will remain to be the technology of choice in the development of most digital electronic systems [9]. Some low power design techniques, however, require enhanced CMOS technolo-

gies. For dual threshold voltage scaling, for instance, low and high threshold voltage transistors must be available, as in so-called multiple threshold voltage CMOS technologies. If the power consumption is extremely critical, silicon on insulator (SOI) technologies can be used instead of bulk CMOS. The expensive SOI wafers and the low yield, however, significantly increase the cost [10]. Once the type of technology has been chosen, the technology level is out of reach for the designer, and all design optimization has to be carried out at the higher levels of abstraction from the transistor level up to the system level. All design and optimization techniques used in this work are compatible with mainstream bulk CMOS fabrication processes.

### 2.1.2 System and Algorithmic Level

#### Partitioning

At the highest levels of abstraction, i.e. the system level and the algorithmic level, the most important task is partitioning. First of all, most systems can be split into logic and memory. The size, the type, the detailed organization, and the management of the memory must then be chosen such that the specified functionality and performance are assured. These choices also have an impact on the power consumption of the system. For further information on low power memory design see the literature [11] [12] [13]. Regarding the logic, which is in the focus of this work, a common approach is to start with functional partitioning, i.e. splitting the specified functionality into less complex subfunctions that can be separately realized by means of different algorithms. The functional partitioning is followed by the actual physical partitioning, where a suitable form of hardware implementation is chosen for each functional partition.

## Implementation Alternatives

Typical hardware implementation alternatives are general purpose microprocessors, DSPs, application specific microprocessors and microcontrollers, configurable logic, and dedicated hardware modules. Each implementation alternative has its own strengths and weaknesses regarding performance, power consumption, flexibility, time to market, and cost. A general purpose microprocessor provides maximum flexibility and sufficient performance for many applications. Since such processors are readily available as separately packaged chips for board-level system development or as intellectual property (IP) blocks for SoC design, even the implementation of complex functionalities takes fairly short time. However, the efficiency of general purpose microprocessors in terms of area and power in proportion to performance is usually low. Digital signal processors (DSP) and application specific processors or controllers are less flexible and, thus, less complex than general purpose processors. If a maximum of flexibility is not absolutely needed, these types of processors lead to more power and area efficient implementations. Configurable logic is a good choice if time to market is critical, the number of pieces to be fabricated is low and the requirements regarding performance and hardware complexity are moderate. Rapid prototyping is another typical field of application. Unfortunately, hardly any power optimization techniques are applicable to configurable logic. Maximum performance and minimum power consumption can be achieved only with dedicated hardware. This comes at the expense of increased time to market and cost. The above statements indicate that the best choice of hardware implementation alternative depends on the specified functionality and performance, the power constraints, and other aspects such as time to market and cost. In modern SoC design, typically some or all components of the system are bought from IP vendors. If the power consumption is critical, it is particularly important to choose IP blocks that have already been designed with the power consumption in mind

or that can at least be further optimized, for instance during logic synthesis. This study is focused on those types of hardware that can be designed and optimized by means of typical ASIC design flows. These are dedicated hardware, application specific processors/controllers and any type of synthesizable IP block (soft macro).

### **Algorithms and Algorithmic Optimization**

A specific functionality can often be realized through several alternative algorithms. Different algorithms usually exhibit different characteristics regarding the performance, the accuracy, and the power consumption. This should be taken into account in system design. On the other hand, the characteristics of the algorithms are often affected by the choice of hardware implementation alternative and vice versa. Thus, a thorough evaluation of algorithms is a complex and time-consuming task for which standard recipes cannot be formulated and that is, therefore, impossible to be automated. System designers often bypass the investigation of different combinations of algorithms and hardware implementation alternatives. Instead, previously published research results are adopted, if available and applicable, which usually results in suboptimal solutions. Once a particular algorithm has been chosen, it can be further optimized with regard to performance or power consumption or both. However, algorithmic optimization techniques are also specific to the type of target hardware. If, for instance, the target is some kind of processor, algorithmic power optimization is a question of software development rather than a hardware design problem. If, on the other hand, the algorithm is to be implemented in dedicated hardware, algorithmic speed-up transformations or multiple supply voltage scheduling can be applied in order to minimize the dynamic power consumption.

## Power Management

Power management reduces the amount of energy wasted whenever parts of a system are not needed at all or not at full speed. With power management schemes the functionality and the performance of a system or circuit are adjusted to time-variant requirements. Examples of such methods are power supply shutdown, dynamic power management, clock gating, and adaptive supply voltage scaling. In a simple embodiment of power management, a system component, e.g. a particular chip, is completely separated from the power supply via an external controllable regulator during idle periods [14]. This is an effective way of avoiding unnecessary static and dynamic power dissipation in inactive components that does not complicate the design of the component to be shut down. The power manager unit (PMU) that controls the regulator is completely external and the power supply pins are the only required interface to the power-managed component. Thus, the component can be designed in the traditional way without the need for any special power management support to be implemented. Major drawbacks of this power supply shutdown approach are the following. Firstly, there is a large power-on delay, which is the time it takes for the supply voltage to stabilize after being switched on again. Secondly, the registers and other non-permanent memory cells lose their content. Power supply shutdown can, in principle, be applied to blocks within an integrated circuit instead of to the entire chip. This, however, requires the power supply infrastructure on the chip to be modified such that the power supply nets of the different blocks are separated from each other and made accessible from the exterior via separate pins. As a consequence, power supply shutdown is restricted to chips in their entirety or to a small number of large blocks on a chip. Complex electronic systems such as personal computers may include advanced dynamic power management (DPM) schemes. Such systems contain various power-manageable components (PMC) controlled by a PMU [15]. Each PMC provides a number of high performance, low

power, and sleep modes/states. The PMU, which may be implemented in hardware or in software, continuously observes the system and puts the PMCs in appropriate states according to the actual requirements at certain points in time. Dynamic power management is widely used in modern notebook computers and, hence, special notebook processors are designed as PMCs. This requires the instruction set, the clock network, the interrupts, etc. to be adapted to the requirements of dynamic power management. Most processors support different low power and sleep modes. In some modes, idle modules within the processors are not separated from the power supply as in the power supply shutdown approach. Instead, the respective parts of the clock network are switched off [15]. If all inputs of the modules to be switched off are registered, there is absolutely no switching activity and, hence no dynamic power dissipation in the idle modules. This technique is called global clock gating. In other modes, certain modules are actually separated from the power supply via internal switches in the power supply nets [15]. Finally, for modules which are not completely idle but also not fully utilized, the clock frequency or the supply voltage or both may be momentarily reduced. Although designing a PMC requires a significant amount of additional design effort, the most challenging task is the development of an effective power management policy (PMP) and its implementation as PMU firm- or software [15]. This software should know about the power characteristics of all modules and be aware of the inevitable performance degradation and power overhead associated with going to and returning from the different low power and sleep modes. An effective PMP should reliably predict the idle time of a module and accurately calculate the net power reduction. The Advanced Power Management (APM) specification was the first industry standard in the field of DPM and has only recently been replaced by the more powerful Advanced Configuration and Power Interface (ACPI) [14], [15], and [16]. Local clock gating is another popular power management technique that requires only moderate additional design effort.

It is frequently used in simple processors such as DSPs, application specific processors, embedded processors and the like, but can be applied to practically any type of circuit. With local clock gating, the control signals that are used to deactivate certain parts of the clock network are locally generated in hardware. In principle, arbitrarily small subcircuits can be deactivated in this way. Since power management based on local clock gating is rather an architectural-level than a high-level technique. A relatively new power management approach is adaptive supply voltage scaling. This is a very attractive technique for dynamic power optimization if the requirements on the performance of a chip vary continuously over time. Instead of just switching off idle components of a system or idle modules on a chip, the clock frequency and the supply voltage are continuously adjusted to the instantaneous performance demand.

### 2.1.3 Architectural Optimization

The two most important methods for power optimization at the architectural level (RTL) are clock gating and architecture-driven supply voltage scaling. Besides clock gating, this subsection covers bus and state encoding and the power characteristics of arithmetic units.

#### Clock Gating

The clock network of a synchronous digital IC normally contains clock buffers and clock nets. The entire clock network, which is frequently called clock tree, is driven by a primary buffer, and subordinate buffers are distributed across the chip. The branches of the clock tree all end at clock input pins of sequential cells such as flip-flops. The large number of driven cells and the large total wire length bring about a large capacitive load on the clock network. Moreover, the switching activity in the clock network is usually the highest of all nets. These are the primary reasons



for the large contribution of the clock tree to the total dynamic power consumption of many chips. In [17], the contribution of the entire clock network including the primary and subordinate clock buffers is quoted at 20% to 45% for different design examples. Thus, clock networks are important targets of low power design.

An effective means of reducing the power consumption in clock networks is clock gating. The concept of clock gating is that logic gates are inserted in the clock tree in a hierarchical manner, either as replacements for or in addition to existing clock buffers. Each of these clock gating cells receives at its input pins a clock signal, which is derived from the primary clock signal CLK, and an enable signal EN, which is generated by global or local control logic, so as to activate or deactivate certain portions of the clock tree. If large portions of the clock tree are deactivated for long periods of time, the power consumption in the clock tree is significantly reduced. Local clock gating is often used in processors, where functional units in the data-path can be deactivated when they are not needed for the execution of a particular instruction [18], [19], [20], [21], and [22]. In this case, the clock enable signals are generated by the instruction decoder. If registers are placed at all inputs of the functional units, clock gating not only affects the power consumption in the clock network itself but suppresses all switching activity within the deactivated data-path units as well. The implementation of gated clocks increases the complexity of the control logic and, hence, creates some power overhead. The overhead is acceptable if it is compensated by the power savings. The correct timing of the enable signals is the most serious issue in the design of clock gating circuitry; glitches at the clock inputs of sequential cells must be avoided in order to assure proper operation of the circuit [22]. Clock gating is often modeled in the HDL code. However, commercial synthesis tools such as BUILDGATES EXTREME (CADENCE) and POWER COMPILER (SYNOPTIS) are also capable of automatic implementation of clock gating.

## Bus Encoding

Low power bus encoding aims at reducing the switching activity and, hence, the dynamic power consumption on long multi-bit interconnects. Bus encoding schemes generally require additional circuitry for the encoding and decoding at the transmitter and receiver side, respectively. This detracts from the overall power reduction. The effectiveness of low power bus encoding also depends on the signal statistics and the knowledge thereof. Particularly important in this respect is the correlation between consecutive data words to be transmitted. Gray coding is often discussed in the context of instruction address encoding in microprocessor systems [12]. Normally, consecutive instructions are stored at consecutive positions in the memory, so that mostly a fixed increment is added to the program counter. If this increment is one, as for byte-addressable memory and a fixed instruction length of one byte, the Gray code may be used instead of the ordinary binary code. The advantage of the Gray code is that an increment of one changes only one bit. Since the Gray code is just a re-ordered binary code, the idea of Gray encoding can be adapted even if the standard increment is different from one. For instance, if the increment is two, as for byte-addressable memory and a fixed instruction length of two byte, the code can simply be re-ordered such that an increment of two changes only one bit. This concept works only for the strictly sequential parts of a program; branch and jump instructions reduce the optimization potential. Also, data memory accesses detract from the optimization potential if the same address bus is used for the instruction and the data addresses. In the case of variable instruction lengths, the advantage of the Gray code vanishes because the increment is not fixed and the signal statistics are no longer predictable. The overhead of Gray address encoding is small. If the program counter and the memory address decoder are already adapted to the optimized coding style, no extra circuitry for the encoding and decoding is needed. If no correlation between data words exists or if the signal statistics are

unknown, redundant codes may be used for reducing the switching activity. The advantages and disadvantages of redundant codes can be illustrated using one-hot coding as an example [23]. In the one-hot code of a decimal value  $M$  only the  $M$ -th bit is set to one while all other bits are zero. Consequently, regardless of the signal statistics, the number of switching bits per cycle is two when the data changes, and zero otherwise. The drawback is that representing  $2^N$  numbers requires  $K = 2^N$  bits as opposed to  $N$  bits required for the ordinary binary coding. The result is an unacceptable overhead for routing, encoding, and decoding. Bus inversion coding (BIC) is an example of redundant bus encoding with low overhead [22], [23], and [24]. In a first embodiment, BIC requires only one additional signal line. The basic idea is to invert a data word prior to transmission if this reduces the number of switching bus lines. The additional line (polarity line) is used for signaling to the receiver whether the data word has been inverted or not. Switching events on this line must, of course, be taken into account when deciding on the polarity of transmissions.

The effectiveness of BIC degrades with increasing bus width. Therefore, broad busses should be split into narrow slices with separate en-/decoders and a separate polarity line for each slice. A maximum switching activity reduction of 25% can be achieved by splitting an  $N$ -bit bus into 2-bit slices at the cost of  $N = 2$  extra wires [22]. This overhead is small compared with one-hot coding. Nevertheless, it is often unacceptable. Thus, four or eight bit are more realistic choices for the width of the slices. The overhead caused by the decoder is small. The encoder, however, can be quite complex and must be taken into account when weighing up advantages and disadvantages of BIC [23] and [25]. Another way of dealing with a lack of knowledge of the signal statistics is adaptive bus encoding, where the incoming data stream is continuously observed and the en-/decoding rules are adapted to the varying statistical properties of the data stream. Recently, an adaptive bus encoding scheme, which is based on the probability based mapping (PBM) technique, was presented

[26]. With PBM, the switching activity on the bus is minimized by minimizing the number of ones to be transmitted. Frequently occurring data words are mapped to code words that contain a small number of ones. A one is transmitted over the bus by inverting the state of the respective bus line. For transmitting a zero, the state of the bus line is maintained. The PBM technique uses a non-redundant data representation and, thus, requires no additional bus lines. The code computation circuitry implemented at both ends of the bus continuously determines a probability of occurrence for each data word in the data stream, computes a new mapping rule in certain intervals, and writes the rule to look-up tables. The PBM scheme can effectively reduce the switching activity if certain data words occur much more frequently in the data stream than others. If, on the other hand, all data words are uniformly distributed, the benefit of PBM vanishes. While a static PBM scheme, where the code mapping rule is optimized for a specific data stream, often yields bad results when used on other data streams, the adaptive PBM scheme can be successfully applied to different data streams or to data streams that exhibit varying statistical properties. Adaptive bus encoding schemes require complex en-/decoding circuitry. The resulting power and area overheads may predominate the possible power savings. Another problem with adaptive bus encoding, which has not been completely solved yet, is the synchronization of the adaption mechanisms at the transmitter and receiver sides of the bus.

### **Low Power Arithmetic Units**

Arithmetic units such as adders and multipliers are critical building blocks in processors and many data-path-dominated ASICs. A variety of concepts for the implementation of such modules can be found in the literature [27]. While, in the past, the design of arithmetic units was driven by the need for sufficient performance at minimum area, their power consumption can now no longer be ignored. [28]has

investigated and compared different types of parallel 16-bit adders and multipliers. Evidently, faster implementations mostly require larger area. For the adder circuits with shorter delay and larger area also translate to higher dynamic power consumption. This is different for the multipliers; the second fastest circuit (Wallace tree) consumes the least dynamic power, while the slowest implementation (array) results in the highest power consumption. The power-delay product (PDP) given in the tables is a possible measure of the trade-off between performance and power. In this respect, the minimum PDP values mark the most efficient implementations of adders and multipliers (variable block width carry skip adder and Wallace tree multiplier). On the basis of this perception, Wallace tree multipliers were, for instance, built into certain StrongARM low power processor derivatives [37]. For a detailed discussion of the structure and the functioning of the different types of adders and multipliers considered in this comparison see the literature [28], [29], and [27].

#### 2.1.4 Logic Level

Standard-cell-based design at the logic level includes logic synthesis, placement, and routing. Logic synthesis can be further divided into technology independent and technology dependent optimization steps.

#### Technology Independent Optimization

Technology independent optimization requires the combinational part of the original design to be separated from the sequential elements. The combinational logic is described in the form of Boolean equations, and the optimization methods operate on these equations. Traditionally, the goal is to find an area efficient, multi-level representation of the combinational logic under timing constraints [30]. A common measure of the area of Boolean networks is the total number of literals<sup>6</sup> in the factored form of the equations [31]. Therefore, the traditional objective of tech-

nology independent optimization is the minimization of the total number of literals. This is usually done with algebraic logic restructuring techniques, e.g. extraction, substitution, factorization, and Boolean minimization. Extraction is the process of identifying a common sub-function of several Boolean equations, introducing a new equation that assigns the common sub-function to a new internal variable, and substituting the common sub-function in the original equations with the new variable [30]. Substitution means substituting a sub-function of a Boolean equation with an existing internal variable [30]. Substitution is applicable if internal variables exist that represent sub-functions of other equations. Another important technique is factorization [30]. The Boolean expression  $a \cdot c + a \cdot d + b \cdot c + b \cdot d$  for instance, can be transformed to  $(a + b) \cdot (c + d)$ . In this example, factorization reduces the number of literals, which is one purpose of the technique. The other purpose is the computation of the cost which is often based on the factored form of the equations, as mentioned above. The same methods can be used for technology independent dynamic power optimization, if the cost function is modified [31]. The cost may be computed as the total sum of the switching activities associated with all literals. This requires the switching activities of the primary inputs to be specified. The switching activities associated with internal variables and primary outputs are then computed by propagating the switching activities at the inputs through the Boolean network using zero delay models for the Boolean operations. Boolean minimization is the process of minimizing Boolean equations using the rules of Boolean algebra, e.g.  $a + \bar{a} = 1$ , and taking into account any dont care conditions [30]. ESPRESSO is a de-facto-standard algorithm for computer aided Boolean minimization of two-level Boolean networks targeting the area of the resulting circuit [30], [32], and [33]. Similar methods can be applied to the set of equations that describes a multi-level Boolean network taking into account additional dont care conditions [34]. Power-aware Boolean minimization, however, requires modification of these methods, as

discussed in [31]. Finally, the optimized Boolean network is prepared for technology mapping in a step called technology decomposition [30] and [31]. A set of primitive Boolean functions such as two-input NAND and NOT is chosen. The Boolean equations are then converted to a graph where each node in the graph is restricted to one of the primitive functions. This process is called technology decomposition and the result is called the subject graph. This graph is the input to technology mapping, the first step in the technology dependent phase of logic synthesis. The quality of the mapping solution depends on the structure of the subject graph. According to [31], a subject graph that minimizes the sum of the switching activities associated with its internal nodes is a good starting point for low power technology mapping.

### Technology Dependent Optimization

The technology dependent phase of logic synthesis starts with a step called technology mapping or cell binding [30] and [31]. In this step, the functionality of each library gate is represented by a graph where each node is restricted to the primitive Boolean functions considered in technology decomposition. These graphs are called pattern graphs. Technology mapping is the process of finding a minimum cost covering of the subject graph by choosing from the collection of pattern graphs that represents the standard cell library. Again, switching activities should be considered when computing the cost in order to obtain a low dynamic power mapping solution [31]. The technology mapping is followed by a post-mapping optimization phase. An important technique applied at this stage is gate sizing. In addition to gate sizing, local transformations are used for altering the structure of the circuit without changing its functionality. Typical examples of local transformations are buffer insertion, complex gate composition and equivalent pin swapping. Gate sizing can affect the dynamic power consumption  $P_{dyn}$  in different ways. Down-sizing, i.e. replacing a cell with a functionally equivalent cell composed of smaller transistors

that have smaller gate input capacitances  $C_G$ , primarily aims at reducing  $C_{node}$  and, thus,  $P_{cap}$  at the input nodes of the sized cell [31] and [22]. In addition, smaller transistor dimensions reduce the short-circuit and the subthreshold currents in the sized cell, thus reducing  $P_{sc}$  and  $P_{sub}$ . On the other hand, down-sizing increases the signal transition time  $t_T$  at the sized cells output, which in turn increases  $P_{sc}$  of the cells driven by the sized cell. For this reason, minimizing the size of cells in non-timing-critical paths does not always result in the lowest dynamic power consumption. At heavily loaded nodes that exhibit very large  $t_T$ , up-sizing may lead to an overall lower  $P_{dyn}$  [35]. Alternatively, extra buffers can be inserted at heavily loaded nodes in order to shorten  $t_T$ , this reduces  $P_{sc}$  at the gates driven by the inserted cell. However, the extra cell introduces extra capacitances and extra short-circuit currents. These overheads must of course be small in comparison with the reduction in short-circuit power at the driven gates in order to make this buffer insertion technique feasible. Standard cell libraries contain so-called complex gates which combine several simple gates in one cell. Complex gate composition replaces a group of simple gates in a gate-level netlist with an equivalent complex gate. [37] and [36]. As a result, some nets no longer connect separate cells. Instead, these nets connect devices within a complex cell which can be accomplished with shorter wires that have less capacitance. This reduces  $P_{cap}$ , especially if many high activity nets can be hidden in complex cells. Another optimization technique, which is called equivalent pin swapping or pin ordering, exploits the different power characteristics of functionally equivalent input pins of the same library cell. These differences in the power characteristics can be due to different input pin capacitances, which leads to different  $P_{cap}$  at the different input nodes. Another possible reason is the exact position of the devices connected to a particular input pin, i.e. the cell-internal circuit structure, which affects the total cell-internal capacitance charged or discharged during a transition of the input node. With pin swapping, high activity nets are



connected to power-efficient input pins with priority [37] and [36].

## Placement and Routing

The traditional objective of placement is to arrange all cells on the chip in such a way that the total wire-length after routing is minimized and, thus, the area is minimal. Since the actual wire-lengths are unknown at this stage, estimates are used for computing the cost function. For power-driven placement, the estimated wire-lengths should be weighted with the switching activities, so that high activity nets are given priority. This way, the total switched capacitance, which determines  $P_{cap}$ , is minimized instead of the total wire-length, which affects the area. In principle, power-driven placement can be carried out with the same algorithms as conventional placement if the cost function is modified appropriately [38]. Routing is the process of making electrical connections between pins of placed cells. In conventional routing, the objective is to minimize the total wire-length. The limitations on the routing resources, i.e. the routing area, the number of metal layers, and the number of feed-throughs between these layers, frequently lead to region congestion. Therefore, it is usually not possible to minimize the length of every single wire. At the beginning of the routing process many resources are available and most wires can be realized with minimum length. As the routing process progresses, congestion problems become more likely and wire-lengths increase. For this reason, critical nets should be routed first. Again, for power-driven routing, the priorities of nets can be determined from the switching activities, so that high activity wires are kept short [38]. The coupling capacitances between neighboring wires are significant sources of power consumption. Therefore, power-driven routing should not only address the wire-length but also reduce the coupling capacitances between high activity wires by increasing their spacing [38].

### 2.1.5 Transistor Level

The standard cell ASIC design style is based on the concept of reusing pre-designed logic gates, 1-bit adders, flip-flops, etc. that are available in so-called standard cell libraries. The following paragraphs cover various aspects related to the development of low power standard cell libraries.

#### Logic Styles

Logic gates can be dynamic or static, i.e. with or without clock control. Dynamic logic is generally faster and, hence, well suited to highest performance circuits. However, the power consumption of dynamic logic is larger than that of static logic because of the additional capacitive load at the clock network(s) and because of unnecessary precharging and discharging of nodes [39]. Moreover, standard tools used for logic and layout synthesis do not support dynamic logic design. The conventional static CMOS logic gates built from n-channel pull-down and p-channel pull-up networks are easy to design, have good driving capabilities which allows high performance, and have good noise margins which makes the circuits robust even at low supply voltages. Static logic gates exploiting cross-coupled p-channel transistors, e.g. cascode voltage switch logic gates, have larger delays and may consume equal or larger amounts of power [61, 106]. Moreover, such gates are difficult to design. Particularly, the design of cells with larger driving strengths is impractical in such logic styles. A third class of static logic, namely the pass transistor logic (PTL), appears to have little or no advantage over the conventional static CMOS gates regarding the power consumption. Moreover, the performance and the robustness of PTL at low supply voltages are insufficient [40], [41], and [39]. For the reasons stated above, only the conventional static logic style can be found in standard cell libraries, except for some pass transistor or transmission gate structures used in XOR gates, multiplexers, flip-flops or full adders.

## Combinational Cells

Standard cell libraries typically contain cells having up to eight inputs. Larger numbers of inputs result in unfeasibly large numbers of transistors connected in series and in parallel. Many transistors connected in series limit the low voltage operation and have either a large total series resistance or large gate capacitances. Many transistors connected in parallel introduce a large total drain diffusion capacitance at the output. Finally, the body effect increases the threshold voltage of transistors connected in series. These effects lead to poor performance [22]. Another important aspect regarding the low power library development is the selection of Boolean functions to be implemented. The number of different Boolean functions of  $N$  input variables is  $M = 2^{2^N}$ . For three inputs, for instance,  $M$  is 256 and for four inputs  $M$  is 65536. It is obvious that only a small collection of all these possible functions can actually be included in a standard cell library. Unfortunately, there is a lack of theoretical analysis of the problem of which functions to implement. Therefore, the actual selection of functions and, hence, types of cells is usually based on human intuition and experience. Typical industrial libraries contain non-inverting buffers, inverters,  $(N)$ AND gates,  $(N)$ OR gates,  $X(N)$ OR gates,  $(N)$ AND- $(N)$ OR complex gates, multiplexer, 1-bit half and full adders and similar cells [22]. Low power libraries should provide a sufficient number of complex gates, i.e.  $(N)$ AND gates,  $(N)$ OR gates, and cells with integrated input inverters. This enables effective complex gate composition for power and area reduction. Also, every type of cell should be provided in sufficiently many different sizes (driving strengths), including minimum sized cells and cells with asymmetrical timing characteristics due to reduced p-channel widths, in order to enable effective gate sizing for timing, power and area optimization. Particularly, non-inverting buffers and inverters, which are frequently used to form optimized cascaded buffers for driving large loads, should be available in a large number of different sizes [42].

## Flip-Flop Cells

Other than dynamic logic gates, dynamic flip-flops may be more power efficient than their static counterparts. This is because dynamic flip-flops can be realized with less transistors and the load presented to the clock network is smaller [43]. However, logic states stored in dynamic circuits need to be periodically refreshed, which prevents dynamic flip-flops from being disabled using clock gating or other means. Moreover, dynamic circuit design is not supported in standard-cell based ASIC design methodologies, as mentioned above. Therefore, only static flip-flops exist in standard cell libraries. The transmission gates are sometimes replaced with tristate buffers, but with this exception, most flip-flop cells in commercial standard cell libraries have this basic structure in common. A disadvantage of this circuit is the large effective load presented to the clock network. This load includes the capacitances that are charged and discharged by the internal clock buffer. Low power flip-flop designs aim primarily at reducing the load presented to the clock network. For instance, has only two transistors driven by the clock input pin. For low to medium switching activity at the data input, this flip-flop consumes less power than the standard flip-flop [42]. At the same time, its delay is significantly shorter. A similar flip-flop comprising a modified master latch is described in [44]. This circuit consumes less power even for high switching activity at the data input.

## Cell Layout Optimization

The means of optimizing the standard cell layouts for low power are limited. Merely optimizing the intra-cell interconnects and the gate structure of very wide transistors can be worthwhile. Large transistors have large drain/source diffusion capacitances if they are realized with a longitudinal gate structure. As explained in [23], the drain capacitance is reduced if the gate is laid out with a finger or ring structure. This technique can be applied, for instance, to large buffer and

inverter cells or to logic cells with large output buffers. For long interconnects within large cells, such as complex flip-flop cells, it is worth considering the area-specific capacitance of different interconnect materials such as polysilicon and metal in order to minimize the wire capacitance.

### 2.1.6 Summary

As mentioned at the beginning of this chapter, low power ASIC design methodologies should include power optimization at all levels of abstraction. Which particular techniques to include in a real-world methodology is determined by their effectiveness, stage of development, versatility, and suitability for automation. These criteria lead to the following assessment of the low power design methods discussed in this chapter. The implementation of power management for static or dynamic power reduction or both is a must, unless it is not applicable to the target application. Dynamic power management can be very effective but requires a tremendous design effort. Therefore, DPM is restricted to the design of complex systems such as personal computers and parts thereof. Clock gating has also proven to be effective and, fortunately, its implementation is simple compared with DPM. Local clock gating is even supported by commercial tools. This technique is state-of-the-art in ASIC design and should be used whenever possible. Regarding the focus of this study, it is important to investigate the impact of clock gating on the effectiveness of voltage scaling in the clock network. Low power bus encoding is a very difficult and conflicting problem. Simple schemes like Gray and one-hot coding are either lacking versatility or are too expensive because of tremendous overheads. Static PBM works well only for the data stream it was designed for and is, thus, only slightly more versatile than Gray coding. Adaptive PBM creates large overhead and suffers from unsolved technical problems. At present, only BIC appears to be useful for a broader range of applications. Low power state encoding is complex, not

well understood and only partially supported by tools. However, if a small subset of transitions can be identified as main contributor to the dynamic power consumption of a particular ASIC, it can be worth encoding the respective states manually while leaving the encoding of the majority of states to a synthesis tool. An impact of bus and state encoding on the effectiveness of the methodology proposed in this study is not expected. The design of optimized arithmetic units from scratch is carried out only in the full-custom design of high performance components such as general purpose microprocessors. For the design of ASICs, technology-independent macro block libraries are available, that contain a variety of pre-designed arithmetic units. Logic synthesis tools revert to these library elements when processing RTL design descriptions subject to timing, power and area constraints. If the constraints cannot be met this way, optimized HDL modeling of arithmetic units can be applied instead of using arithmetic operators in the HDL code. The latter approach is particularly suitable for the design of critical units in the data-path of application specific processor cores; the arithmetic units can be adapted to the target application while preserving synthesizability and, thus, independency of the target fabrication technology. Just as for the aforementioned encoding schemes, an impact on the effectiveness of the methodology proposed in this study is not expected. Logic synthesis is fully automated and relies on standard tools. These tools do not support power optimization in the technology independent phase of logic synthesis. Technology dependent optimization using gate sizing, buffer insertion, complex gate composition, equivalent pin swapping, phase assignment, etc. is state-of-the-art and should be used in any case. About 10% to 20% dynamic power reduction can be expected. These techniques directly compete with the logic-level voltage scaling approach that is in the focus of this study. Therefore, the proposed methodology assures that the effect of state-of-the-art power-driven logic synthesis is taken into account in all investigations. Placement and routing are also automated and are also carried out using

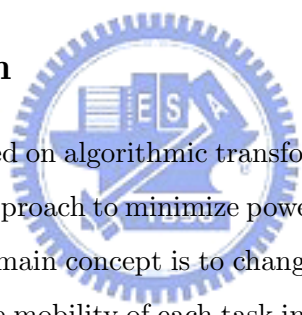
standard tools. In existing design methodologies, the area is usually the only optimization criterion. Timing-driven placement and routing are possible but are not yet standard. Power-driven placement and routing are still under development and cannot be carried out with existing tools. Regarding the library development, many semiconductor vendors avoid the effort to develop completely new libraries. Instead, existing libraries are adapted to newer technology generations with minimum effort. Some companies, e.g. ARTISAN COMPONENTS9 , claim that their libraries are optimized for low power design. However, neither is there any evidence, nor is any information available on how this was achieved. The techniques discussed in this chapter aim at power optimization through power supply shut-down and through optimization of circuit and device parameters such as the switching activity, the device and interconnect capacitances, the signal transition times, and the effective transconductance. Other important parameters are the supply and threshold voltages. However, the simple concept of global supply voltage minimization driven by pipelining or parallelization is usually the only available voltage scaling option.



# Chapter 3

## Resource-Constrained Low-Power Scheduling in HLS

### 3.1 Introduction



Paper [56] exploited on algorithmic transformations for multiple-voltage HLS and present an efficient approach to minimize power consumption under resource and latency constraints. The main concept is to change the computational structures by transformations and make mobility of each task in fully-specified flow graph (FSFG) as high as possible. The mobility means the ability to schedule the starting time of a task. It is defined as the distance between its as-late-as-possible (ALAP) schedule time and its as-soon-as-possible (ASAP) schedule time. Obviously, the increase of mobilities may raise the possibility of assigning tasks to low-voltage components. To earn task mobilities, we use loop shrinking, retiming and unfolding techniques. Furthermore, this chapter provides thorough analysis on different combinations of algorithmic transformations.

In low-power designs for battery-driven portable applications, the peak power drives the transient characteristic of the CMOS circuit. Therefore, in this work, the



minimization of the peak power is another important consideration. Following the optimization of average power dissipation, we suppress the peak power dissipation by the barrier-driven approach. The barrier-driven approach gradually compresses the task schedulability until no further legal scheduling can be found.

As the results, our approach can achieve significant power reduction. In the case of the third-order IIR filter, the proposed methodology can save up to 54.77% of power consumption while the resources running at 5V and 3.3V under the latency constraint of  $1.5T_c$  and resource constraints of  $\{1, 1, 1, 1\}$  (one 3.3V multiplier, one 5V multiplier, one 3.3V adder, and one 5V adder).

The rest of the chapter is organized as follows. In Section 3.2, we introduce algorithmic transformations. Section 3.3 presents the proposed approaches in details. Section 3.4 shows the experimental results and Section 3.5 is the summary of this work.

## 3.2 Overview of Basic Scheduling Technique and Algorithmic Transformations

### 3.2.1 Basic Scheduling Techniques

In this section, some of the basic scheduling techniques are discussed. The simplest scheduling technique is as-soon-as-possible (ASAP) scheduling [5], where the operations (tasks) in the Fully-Specified Flow Graph (FSFG) are scheduled step by step from the first control step to the last. An operation is called ready operation if all of its predecessors are scheduled. This procedure schedules ready operations to the next control step until all the operations are scheduled. As-late-as possible (ALAP) scheduling [6] performs a similar procedure as ASAP scheduling. ALAP scheduling schedules the operations from the last control step toward the first control step. An operation is scheduled to the next control step as all of its successors are

scheduled. Fig. 3.4 shows the ASAP and ALAP scheduling examples.

Due to the constraints of the number of function units, it is not possible to assign too many operations of the same type into one control step. A modified ASAP scheduling involves arbitrarily delaying the ready operations when their number exceeds resource constraints [6].

The list-based scheduling [7], which was originally used in microcode compaction, has been adopted by many high-level synthesis systems. It assigns operations in the FSFG to control steps from the first control step to the last one. The ready operations are given a priority according to heuristic rules and are scheduled into the next control step according to this predefined priority. When the number of scheduled operations exceeds the resource constraints, the remaining operations are delayed [64].

### 3.2.2 Fully-Specified Flow Graph

A deterministic DSP algorithm can be represented by an FSFG. The FSFG describes the relationship between a set of input and output sequences [60]. Fig. 3.1 shows an FSFG for a second-order IIR filter. In FSFG, the IPB is determined by loops [61] and has been used to measure the performance bound of the implementation of FSFG [61, 62, 63]. The iteration period (IP) for a loop is defined as the total computational latency in the loop divided by the total number of delays. The IPB is the maximum value of IPs and represents the lower bound of MASP. For instance, if a multiplication takes 2 time-units and an addition takes one time-unit, the FSFG shown in Fig. 3.1 has an IPB of 4 time-units. However, the IPB is not always achieved without using algorithmic transformations. In Fig. 3.1, for example, the MASP is limited by the critical path  $G-C-A-B$  and so equals to 5 time-units. Thus, to obtain the rate-optimal implementation of FSFG, this chapter introduces three techniques for the IPB reduction and the minimization of MASP. These algo-

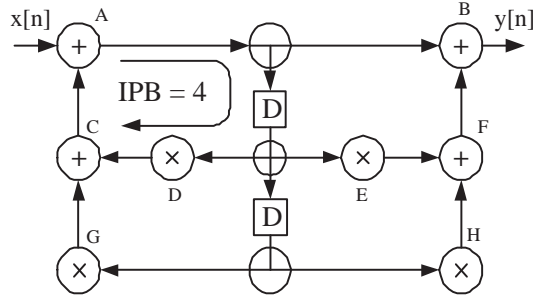


Figure 3.1: FSFG of second-order IIR filter.

rithmic transformation techniques will be explained in the following subsections.

### Loop Shrinking

Loop shrinking can reconstruct the FSFG to obtain the optimal IPB for loops. Fig. 3.2 is an example of loop shrinking. Fig. 3.2(a) has a chain of two additions within the loop. According to the associativity of addition, the function  $a + (b + c)$  in Fig. 3.2(a) is equivalent to the function  $(a + b) + c$  in Fig. 3.2(b). Obviously, the critical loop,  $L_1$ , has been shrunk in Fig. 3.2(b) and IP is reduced as well. Therefore, we can perform loop shrinking on critical loop, which has the maximum IP, to reduce the IPB while the functionality of FSFG keeps the same. In case each task takes one time-unit to execute, the IPB can be reduced from 3 time-units to 2 time-units.

### Retiming and Unfolding

The optimal IPB does not guarantee the optimal rate. Retiming is a process that may help making MASP equal to IPB. With the delay transfer or nodal transfer, it is possible to make MASP optimized. Unfortunately, the retiming technique might not guarantee the optimal MASP. Fig. 3.3(a), for example, the MASP can not be achieved by retiming since node A requires 20 time-units to execute. To achieve

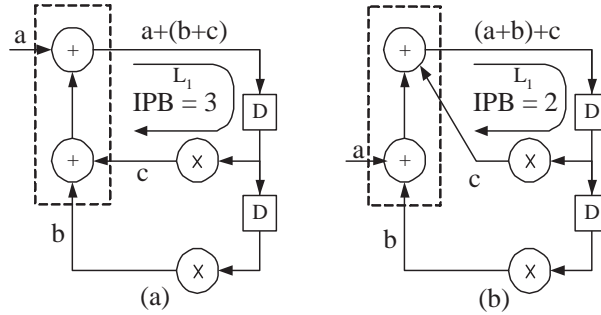


Figure 3.2: Loop shrinking of second-order IIR. (a) The original FSFG. (b) The equivalent FSFG.

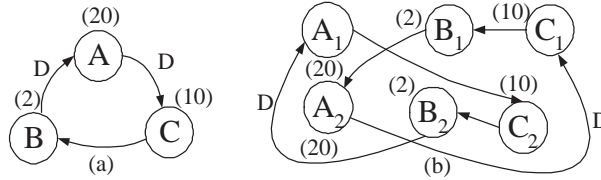


Figure 3.3: Unfolding result. (a) An example of FSFG that cannot achieve IPB. (b) A rate-optimal FSFG using unfolding.

the optimal rate, [63] presents the unfolding technique. Instead of describing one iteration of the computation in the form of a recursive loop, unfolding by a factor  $P$  implies  $P$  consecutive iterations. If the original FSFG has  $N$  tasks, the  $P$ -unfolded FSFG has  $P \times N$  tasks, and the IPB is  $P$  times larger than that of the original FSFG. Fig. 3.3(b) illustrates the result of 2-unfolded FSFG in Fig. 3.3(a). In Fig. 3.3(b), the total number of delays, however, remains unchanged and precedence constraints are also not violated. The unfolding technique can obtain the rate-optimal static schedules.

### 3.3 Proposed Approach

We proposed a multiple-voltage HLS for the low-power DSP realization. The HLS algorithm first applies the loop shrinking technique for IPB reduction, and then

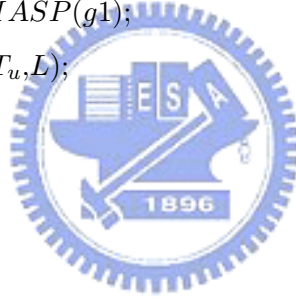
minimize the MASP using the retiming and unfolding. Once the MASP is optimized, the mobilities can be enlarged and the scheduler will have more room to schedule low-voltage components.

### Multi-Voltage HLS Algorithm

```

SCHEDULE(FSFG,  $R_u$ ,  $T_u$ ,  $L$ , EnergyTb, LCTb)
{
   $g = \text{Read}(\text{FSFG}, R_u, T_u, L, \text{EnergyTb}, \text{LCTb});$ 
   $g1 = \text{Shrink}(\text{graph});$ 
  if (MASP = IPB)
     $S = \text{MVS}(g1, R_u, T_u, L);$ 
  else{
     $g2 = \text{Minimize\_MASP}(g1);$ 
     $S = \text{MVS}(g2, R_u, T_u, L);$ 
  }
   $S = \text{LC\_refine}(S);$ 
  Report( $S$ );
}

```



The inputs to our methodology are an FSFG, a resource constraint  $R_u$ , a latency constraint  $T_u$ , a number of voltage levels  $L$ , an energy table of multiple voltages *EnergyTb*, an energy table of level converters *LCTb*, and the outputs are the voltage assignment, start time, and end time of each node and the total power consumption of the scheduling if the legal scheduling exists. In a nutshell, the proposed resource and latency constrained algorithm operates in four passes. In the first pass, the input file specifies  $R_u$ ,  $T_u$ , and the operations within the FSFG. Once having the input information, we use loop shrinking technique to reduce the IPB. In the second pass, we compute the MASP to check whether it matches the

IPB or not. If the MASP matches the IPB, the graph will be sent to the third pass. If the MASP is not equal to the IPB yet, the minimization of MASP can be achieved by  $Minimize\_MASP(graph)$  to obtain optimal mobilities under the given resource constraint. In the third pass,  $MVS(graph, R_u, T_u, L)$ , is used to schedule and assign tasks to the proper scheduling time and components such that the total power/energy consumption is minimum. In the last pass,  $LC\_refine(S)$  refines the schedule of the third pass by considering level converters.

### 3.3.1 $Shrink(graph)$

The follows list the loop shrinking steps.

*Step1 : Calculate initial IPB;*

*Step2 : Search for the critical loop;*

*Step3 : Rearrange edges having relation to the adjacent addition nodes in the critical loop;*

*Step4 : Calculate new IPB; If new IPB < initial IPB, save the rearranged FSFG and let initial IPB equals to new IPB; Otherwise go to Step6;*

*Step5 : Go to Step2;*

*Step6 : Loop shrinking ends;*

$Shrink(graph)$  searches two adjacent addition operations in the critical loop first and then rearrange the associated edges to reduce the number of nodes in the critical loop. The procedure  $Shrink(graph)$  will repeat Step2 to Step4 until the IPB cannot be improved.

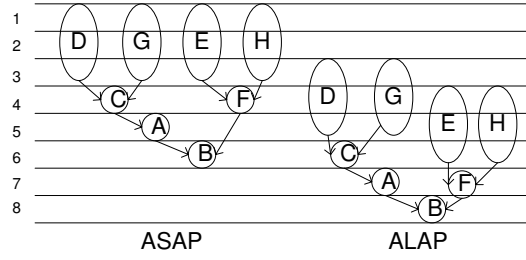


Figure 3.4: Schedules of second-order IIR before retiming.

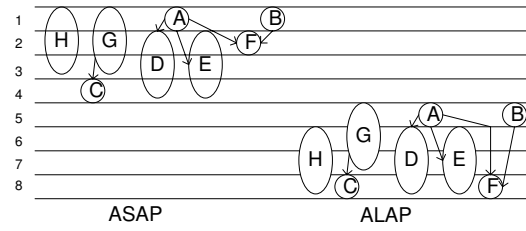


Figure 3.5: Schedules of second-order IIR after retiming.

### 3.3.2 $Minimize\_MASP(graph)$

This subroutine uses retiming or unfolding techniques to obtain MASP and hence optimize mobilities under given timing constraints. The retiming transformation has been implemented by the integer linear programming (ILP) formulation [64]. If the MASP of the retimed FSFG can not achieve the IPB obtained from the  $Shrink(graph)$  stage, then we apply the unfolding technique to the retimed FSFG to guarantee that MASP matches the IPB. Fig. 3.4 shows the ASAP and ALAP scheduling result of the original FSFG of the second-order IIR filter. By applying  $Minimize\_MASP(graph)$  subroutine, Fig. 3.5 and Fig. 3.6 illustrate the scheduling results obtained by the retiming and the unfolding techniques, respectively.

### 3.3.3 $MVS(graph, R_u, T_u, L)$

Fig. 3.7 shows the flowchart of  $MVS(graph, R_u, T_u, L)$ , where the index  $i$  and  $j$  represent the number of classes among all tasks and voltage levels, respectively.

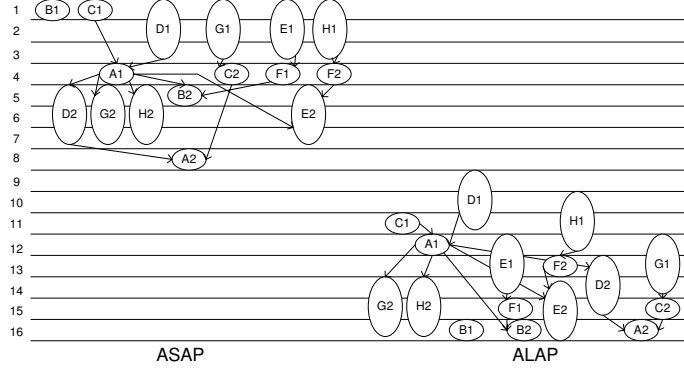


Figure 3.6: Schedules of second-order IIR after unfolding.

The index  $k$  represents the number of tasks in the class  $c$ . In the beginning, all tasks in the FSFG are set to be "unmarked" to represent the un-scheduled status of each node. Then the program will choose a class of operations,  $c$ , such as multiplications, according to the effectiveness among all tasks. To obtain maximally power saving, we determine the number  $M$ , which represents how many number of tasks with the highest effectiveness will be assigned to the lowest voltage resources under the given constraint. The number  $M$  is defined by  $\lfloor \frac{T_u}{T_c(v(j))} \rfloor$ , where  $T_c(v(j))$  represents the execution time of the task with the highest effectiveness operating at  $v(j)$  voltage. Then we can assign tasks by a proposed task-assignment scheme. The scheme is priority-based in that the task with higher priority will have higher opportunity to be assigned to lower voltage resource. So we recursively compute the parameter-list including the value of ALAP and ASAP, depth, and the mobility for each task and assign tasks with higher priority to lower voltage resources. Note that the scheduling order of all tasks does not always follow the data precedence, which means we might deal with all the multiplications before all additions, therefore, we use  $T_s + T_c(v(j)) \leq T_L$  to check the legal scheduling result of each task, where  $T_s$  is the scheduling time and  $T_L$  is the end time in the ALAP scheduling result. Once the timing constraint is illegal, the higher voltage resource will be utilized by the



increment of the number  $j$  to make sure the scheduling result is feasible. In addition, the peak power is bounded between the  $PP_L$  and the  $PP_U$ , where the  $PP_L$  and the  $PP_U$  present the lower bound and the upper bound of peak power, respectively. These two bounds are defined by the average power consumption of using the lowest and the highest voltages without latency constraints. We explore the bounded space from the middle point to find the minimum peak power solution. Moreover, we can reduce the number  $M$  if necessary. For instance, according to the energy chart in [65], the multiplications have higher effectiveness than that of additions. If  $T_u = 10$  and  $R_u = \{1, 1, 1, 1\}$ , we will try to assign ( $\lfloor \frac{10}{4} \rfloor$ ) multiplications to 3.3V resources because one 3.3V multiplier takes 4 time-units for the execution. In case this constraint can not be achieved, we must relieve it by resetting the number with  $M = M - 1$ .

### 3.3.4 $LC\_refine(S)$

After optimizing power consumption using multiple-voltages on data path scheduling, the proposed approach then takes the power consumption of level converters into account and refines the use of level converter. The reason why the optimization of level converters is considered after resource assignment is because the multiple voltage assignment can gain more amount of power saving than does the reduction of level converters. From the power models in [65, 54], obviously, the power difference between high-voltage and low-voltage components is larger than power consumption of level converters. One can optimize the power consumption of multiple voltage scheduling by treating level converters and resource assignment simultaneously. [57], for instance, uses ILP to find the power-optimal solution for multiple voltage levels. Their algorithm exhaustively explores all design space and has  $O(N^3)$  time complexity. The exponential complexity makes the multiple voltage scheduling time-consuming. Instead of considering level converters within resource

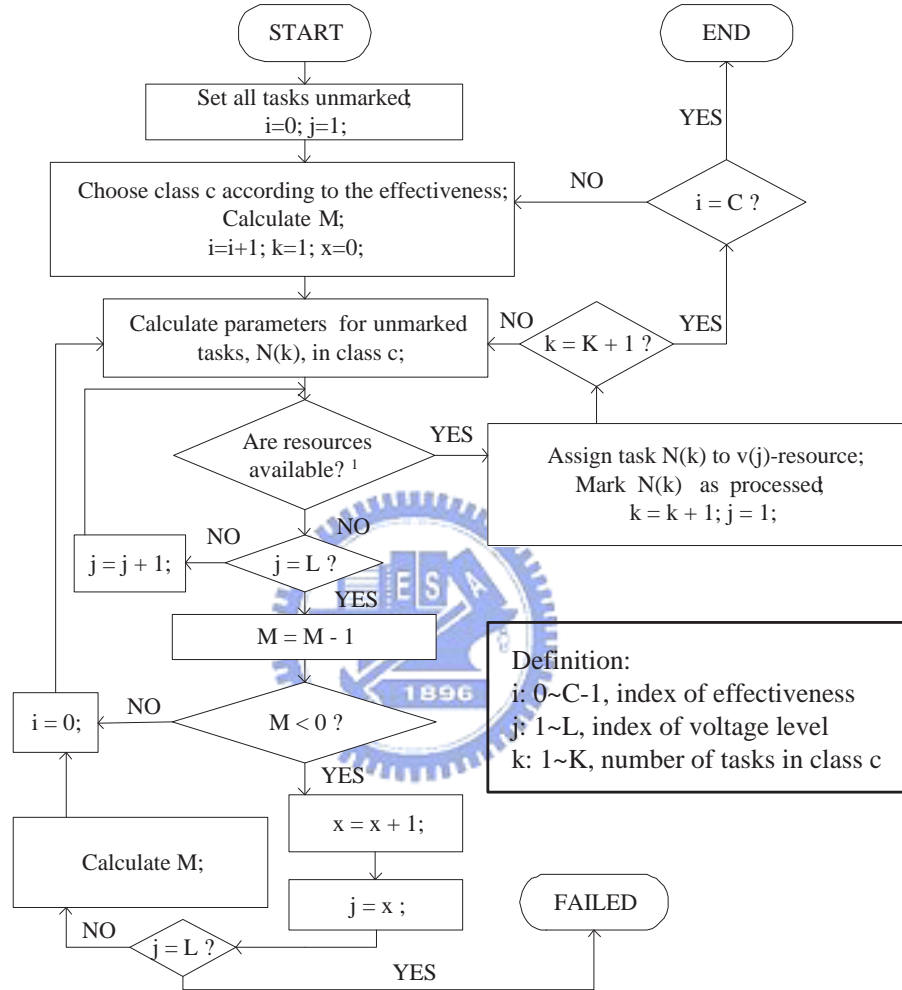


Figure 3.7: Flowchart of multiple voltage scheduling. (<sup>1</sup>In the class  $c$ , is the resource with  $v(j)$ -voltage available, is the cycle power consumption under the peak power bound, and  $T_s + T_c(v(j)) \leq T_L$ ?)

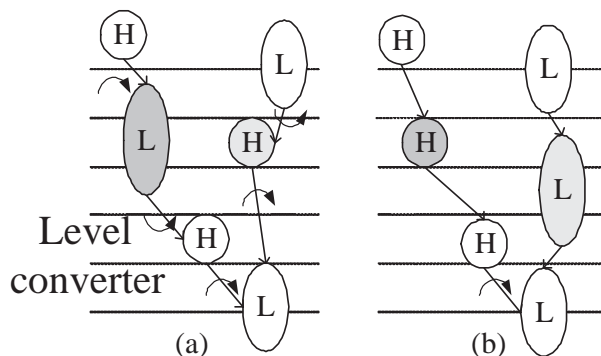


Figure 3.8: Examples of level converters.

assignment, we separate the reduction of level converters from multiple voltage scheduling to produce comparable results with only polynomial complexity  $O(N)$ . Paper [55] gives the level converter introduction the lowest priority in the list-based algorithm and has polynomial complexity. However, they may not be able to refine inefficient resource assignment by removing level converters. Differently from list-based algorithm, our approach is to remove level converters when their associated resource assignments are inefficient. A resource assignment is called inefficient when the power consumption difference between its high voltage and low voltage components is smaller than the power consumption of level converter. For example, in Fig.3.8(a), H and L indicate higher-voltage and lower-voltage components, respectively, there are five level converters are required and the power difference between high-voltage and low-voltage components is smaller than low-to-high level converter. We can switch highlighted high-voltage task with highlighted low-voltage task which has been shown in Fig.3.8(b) to remove requirements of level converters to avoid the power consumption overhead as much as possible.

### 3.4 Experimental Results

The proposed algorithm was implemented in C++ and tested with selected benchmark circuits. We tested the scheduling algorithm using the following sets of resource constraints (RC1, RC2, and RC3):

- 1) number of multipliers: 1 at 5V and 1 at 3.3V; number of adders: 1 at 5V and 1 at 3.3V;
- 2) number of multipliers: 1 at 5V and 2 at 3.3V; number of adders: 1 at 5V and 2 at 3.3V;
- 3) number of multipliers: 1 at 5V, 1 at 3.3V, and 1 at 2.4V; number of adders: 1 at 5V, 1 at 3.3V, and 1 at 2.4V;

Our algorithm has high degree of flexibility and can be applied for other compositions of supply voltages. We also assume that multiple power lines are available, and level converters are needed between resources if they operate at different voltages. The number of level converters is not user defined. Moreover, the proposed algorithm tries to reduce the number of level converters to save the power consumption. The energy consumption and the worst case delays of the different function units have been adopted from [65] and the energy dissipation of level converters adopted from [54]. The delay costs of the level converters are absorbed in the worst case delay values. Because we address the problem under timing constraint, energy consumption can be referred as power consumption. We assume the clock period is 20ns. So the clock cycle of each different function unit can be computed.

The comparison with AR filter (3rd-order IIR filter) has been listed in Table 3.1. In this example, it has been found that our algorithm yielded a greater reduction in power consumption. For instance, for the 3rd order IIR filter with the resource constraint RC1, and a timing constraint of 16, we achieve a 40.20% reduction with the unfolding factor  $P = 3$  compared to the 26.00% reduction by

Table 3.1: Comparison results of third-order IIR filter with resource constraint RC1 and a timing constraint of 16 control steps.

Scheduling algorithm	Power (pJ)	% Reduction
$E_5$	13554	—
[55]	10092	26.00
Retiming applied only	8516	37.16
Proposed	8092	40.20

using the algorithm in [55]. The power reduction from the proposed algorithm compared with  $E_5$  has been tabulated in Table 3.3, where  $E_5$  is the power dissipation corresponding to the supply voltage of 5V.  $E_{alg}$  is the average power dissipation obtained by our algorithm. Table 3.2 lists the power consumption and reduction of benchmarks by applying retiming transformations only. Timing constraints are given for two different values:  $1.5T_c$  and  $2T_c$ , where  $T_c$  is the optimal minimum-computation time (critical-path delay) under the given resource constraint. We also plotted the power consumption per cycle, over all the given number of control steps (clock steps) for different benchmarks in Fig. 3.11. The solid curves correspond to the profile when the scheduling is operated without setting the peak power bound. The profiles with dotted lines correspond to the case when the peak power bound scheme is used. Fig. 3.9 and Fig. 3.10 shows the effect of the peak power bound upon on the scheduling results of the second-order IIR and the fifth-order EW filter, respectively.

### 3.5 Summary

This chapter presents a new HLS methodology under resource and latency constraints. The proposed scheme minimizes the power and the peak power consumption by assigning as many nodes to lower voltage components as possible by applying algorithmic transformations. The loop shrinking transformation can reduce the IPB and the unfolding and retiming techniques guarantee the MASP. Doing so,

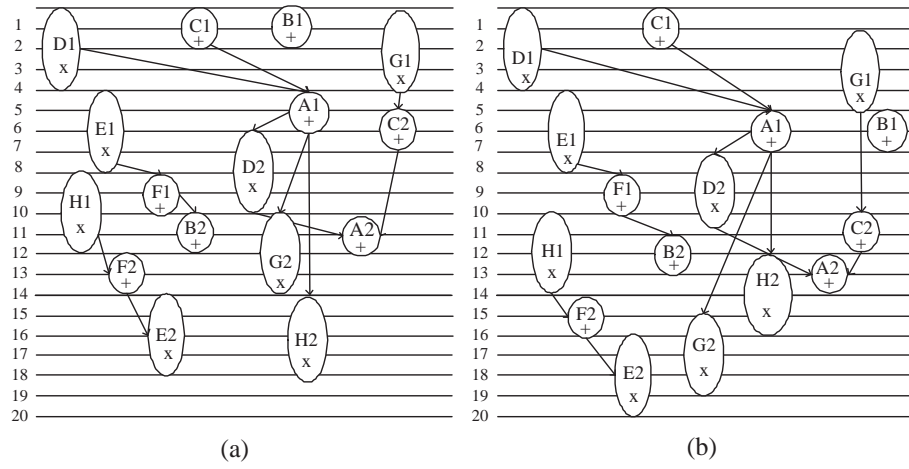


Figure 3.9: Scheduling results of second-order IIR filter with resource constraint RC2. (a)Without peak power bound. (b)With peak power bound.

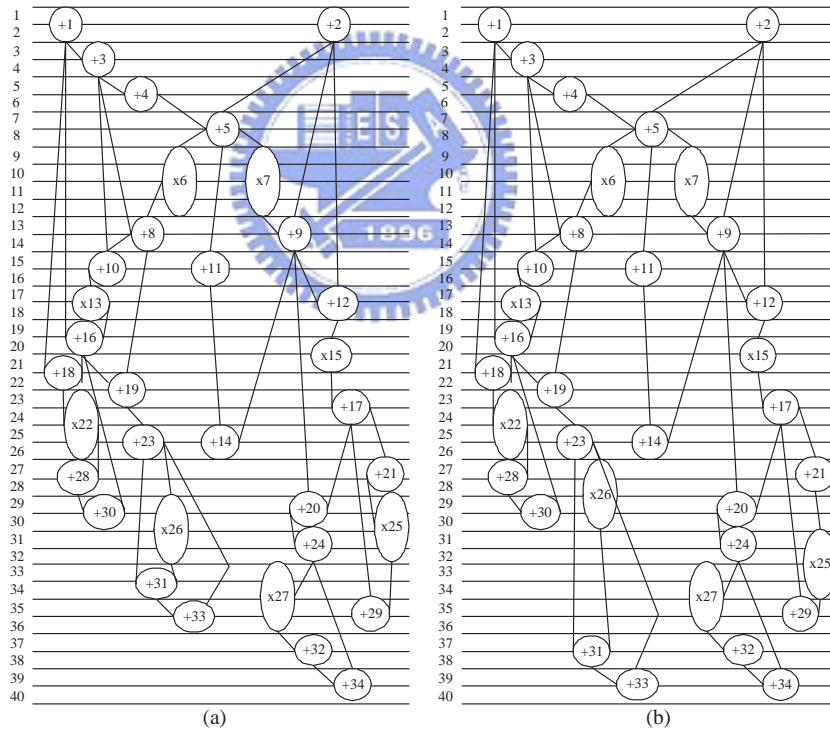


Figure 3.10: Scheduling results of fifth-order EW filter with resource constraint RC2. (a)Without peak power bound. (b)With peak power bound.

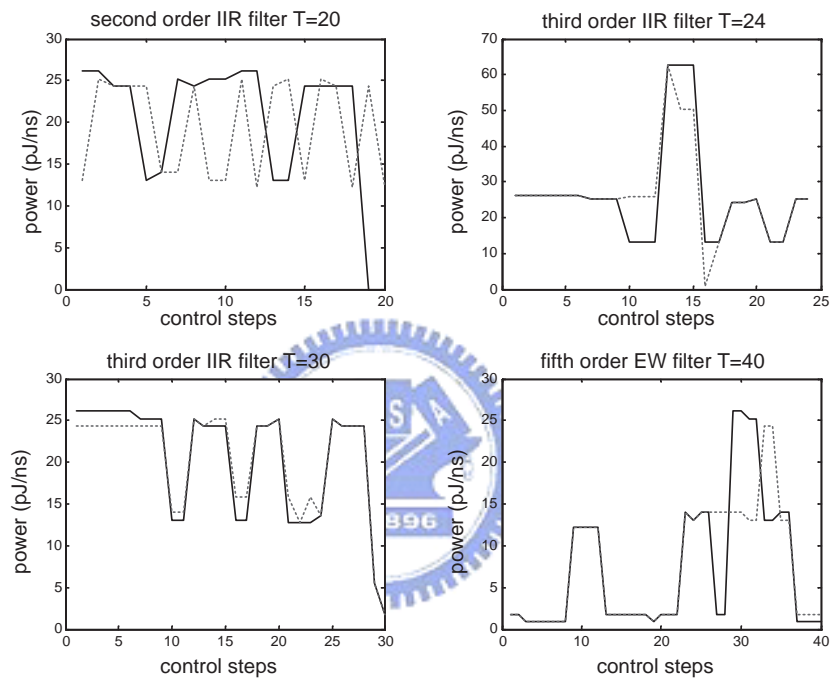


Figure 3.11: Cycle power consumption of different benchmarks with resource constraint RC2.

Table 3.2: Power consumption and reduction of benchmarks by applying retiming transformations only.

Resource constraint	RC1			RC3	
Benchmark	Latency	$E_{alg}$	Reduction	$E_{alg}$	Reduction
second-order IIR filter $E_5=9039\text{pJ}$	$1.5T_c$	6238	30.99%	6096	32.55%
	$2T_c$	5153	44.10%	4900	45.79%
third-order IIR filter $E_5=13554\text{pJ}$	$1.5T_c$	8803	35.05%	8425	37.84%
	$2T_c$	8017	40.85%	8017	40.85%
fifth-order IIR filter $E_5=21694\text{pJ}$	$1.5T_c$	17711	18.36%	17329	20.12%
	$2T_c$	17208	20.68%	16362	24.58%
LMS adaptive filter $E_5=15756\text{pJ}$	$1.5T_c$	12165	22.79%	11918	24.36%
	$2T_c$	10752	31.76%	10240	35.00%
fifth-order EW filter $E_5=1482\text{pJ}$	$1.5T_c$	1330	10.20%	1330	10.20%
	$2T_c$	1188	19.82%	1188	19.82%
2-D fast DCT $E_5=31398\text{pJ}$	$1.5T_c$	23184	26.16%	22543	28.20%
	$2T_c$	21979	30.00%	21693	30.90%

high task mobilities arise the possibility of the assignment of low-voltage resources. As the experimental results, under the timing constraint of  $1.5T_c$ , the power reduction obtained by the proposed methodology is up to 54.77% with two voltage levels and 59.13% with three voltage levels, respectively. The integration of such a scheduler into a low-power datapath synthesis tool will significantly benefit resource efficient low-power DSP applications.



Table 3.3: Power consumption and reduction of benchmarks by the proposed scheduling algorithm.

Resource constraint	RC1			RC3	
	Latency	$E_{alg}$	Reduction	$E_{alg}$	Reduction
second-order IIR filter $E_5=9039\text{pJ}$	$1.5T_c$	5035	44.28%	5035	44.28%
	$2T_c$	3484	61.44%	2767	69.39%
third-order IIR filter $E_5=13554\text{pJ}$	$1.5T_c$	6130	54.77%	5540	59.13%
	$2T_c$	5164	61.90%	3226	76.20%
fifth-order IIR filter $E_5=21694\text{pJ}$	$1.5T_c$	14583	32.78%	14285	34.15%
	$2T_c$	13537	37.60%	9311	57.08%
LMS adaptive filter $E_5=15756\text{pJ}$	$1.5T_c$	10698	32.10%	10197	35.28%
	$2T_c$	8902	43.50%	7651	51.44%
fifth-order EW filter $E_5=1482\text{pJ}$	$1.5T_c$	1330	10.20%	1330	10.20%
	$2T_c$	1188	19.82%	1188	19.82%
2-D fast DCT $E_5=31398\text{pJ}$	$1.5T_c$	18955	39.63%	17978	42.74%
	$2T_c$	17649	43.79%	16917	46.12%

# Chapter 4

## Limited-Resource Folding Techniques

Two folding techniques have been proposed in this chapter. Some preliminary ideas of folding technique have been introduced first.

Fig. 4.1 illustrates a waterfall-like algorithm, in which there are six identical processing nodes in a linear progression. We could fold the waterfall-like algorithm into a resource-constrained architecture by performing resource sharing technique. Fig. 4.2 and Fig. 4.3 show two different resource sharing methods. The color represents the assignment rules. The processing nodes will be processed by the processing element (PE) with the same color. For example, the processing nodes,  $P1$ ,  $P2$ , and  $P3$ , will be processed by  $PE1$  time-multiplexedly in Fig. 4.2, and the processing nodes,  $P1$  and  $P4$ , will be processed by  $PE1$  in Fig. 4.3.

With the decision of resource sharing method, we can try to design the proper folded architecture with specified registers and control unit to perform the algorithm with high resource utilization when the throughput rate is lower than the maximum speed at which a full-length datapath can operate. The design detail and the performance analysis will be addressed in the following sections.



Figure 4.1: A waterfall-like algorithm containing 6 processing nodes.

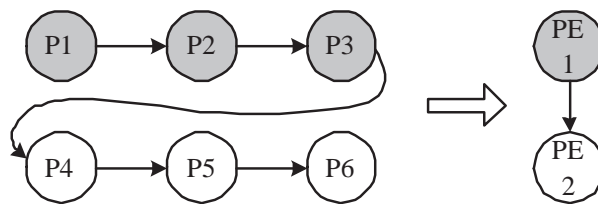


Figure 4.2: Resource sharing of grouping.

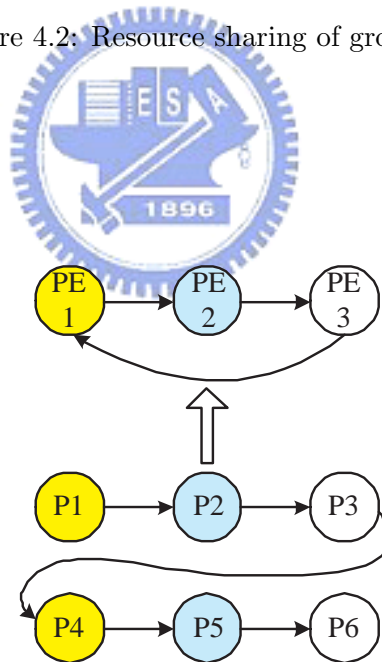


Figure 4.3: Resource sharing of segmentation.

## 4.1 Limited-Resource DWT Processor

### 4.1.1 Introduction

The discrete wavelet transformation (DWT) has been broadly employed in DSP applications in the last decade, particularly in the field of multimedia signal processing, such as video coding, noise analysis, image compression, and so on. Generally, DWT has two major tasks: FIR filtering and 2-folded decimator. FIR filtering is realized by using processing elements(PE), such as Multiply/Accumulate units(MACs), and 2-folded decimator is performed by controlling switching elements. Much research on DWT implementation assumes that the processing and switching elements are not strictly restricted and pays much attention on decreasing the storage size and increasing hardware utilization [66, 67, 68, 69, 70]; however, very few papers deal with the limited-resource implementation. The limited-resource problem arises when the number of atomic units is constrained. It has been becoming one of the most important issues in system-on-chip (SOC) design [60]. Consequently, this section focuses on the scheduling of I/O data streams for limited number of function units (MAC) instead of applying dedicated design for FIR filtering and 2-folded decimator and presented and takes the scheduling is for the MAC-level DWT signal processing.

A number of DWT scheduling algorithms founded on folded architecture have been proposed [71, 72, 73]. Most papers realize DWT processors based on filter-level architecture. In general, they apply unconstrained MACs for the implementation of FIR filtering. We consider these architecture as a filter-level DWT processor. The filter-level DWT processors generate scheduling for intermediate data streams between octaves and require additional inter-octave control units. The extra control units increase hardware overhead and implementation complexity. Besides, most papers focus on the reduction of memory size between octaves, but they did not deal with the memory requirement within a FIR filter. Instead of implementing

DWT processor at filter-level, this section develops the DWT architecture at the MAC-level and the number of MACs is limited. We propose a limited-resource scheduling algorithm that zooms our scope into octaves and flatten the DWT tree into a data flow graph. By doing so, we are at a better position to fully control the timing of data streams and thoroughly consider both inter-octave and inter-MAC communication to decide the all demanded memory. Accordingly, we propose a novel VLSI synthesis for MAC-level DWT architecture and its matched scheduling algorithm, called the limited-resource scheduling algorithm(LRS).

Fig. 4.4 is the design flow of the DWT IP synthesizer that combines DWT SIP generator with the limited-resource scheduler. First, a highly scalable architecture, that consists of an input FIFO, a control unit, embedded memory, registers, and limited number of MACs, is constructed as per the architecture constraints and DWT parameters. The architecture constraints are the data width, the number of MACs, the size of embedded RAM, and the input data volume. The parameters are the number of stages, the number of dimensions, and the type of DWT base. Then, the LRS algorithm produces four scheduling matrices for the control unit synthesis. Since the design process is rather systematic, we have developed an automated generator to synthesize silicon intelligent property (SIP) for DWT processor. The synthesized SIP core can be embedded into a SOC for complete signal processing applications, such as image compression and audio signal restoration, or converted to program codes for commercial off-the-shelf DSP processors with programmable devices such as FPGA and CPLD.

#### 4.1.2 Conventional DWT VLSI architecture

For DWT implementation, several filter-based architectures have been proposed [74, 75, 72, 67] because the basic DWT computation is filtering. Fig. 4.5 illustrates a typical filter-based DWT architecture. The conventional architecture

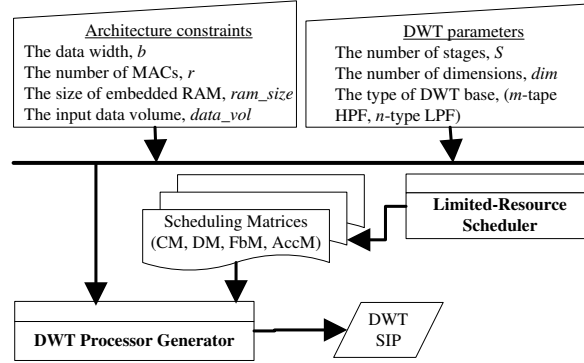


Figure 4.4: The design flow of DWT SIP synthesizer.

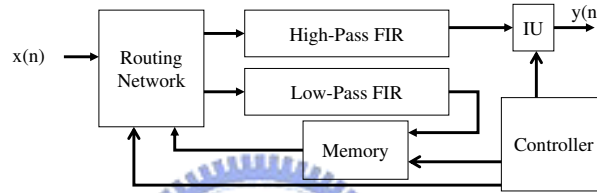


Figure 4.5: A typical filter-based DWT architecture.

is driven by pyramid algorithm. The pyramid algorithm was proposed by Mallat [76]. In the conventional DWT architecture, the lowpass and highpass filters are used to recursively compute each DWT stage, called octave, while the data volume halves after each octave computation. Given the data volume  $N$  and the number of stages  $S$ , the reuse of filters requires a storage unit to hold  $N$  data for each octave. Without parallelizing octave computation, the architecture obviously requires a long latency to finish the DWT computation, because new octave cannot start until the previous octave is completely done. Therefore, paper [77] presents recursive pyramid algorithm (RPA) for scheduling octave computations in parallel.

Traditionally, filter-based architectures address on two challenges: optimizing memory size for intermediate data streams between octaves and increasing the

degree of parallelism in octave scheduling. To deal with these two challenges, researchers spend much effort on design control units and routing network. Paper [72] uses systolic architecture to optimize the computation throughput of filtering. However, the systolic architecture requires extra switching units between filters to orchestrate and synchronize the lowpass and highpass filtering operations. Paper [71] presents the folded wavelet architecture to minimize the interfiltering memory size while carefully designing the routing network. Since the routing network is irregular, the control of routing network becomes complex and the area of routing network is significantly enlarged. Although published literatures have done great job on memory size minimization and parallel scheduling, their works fail to explore optimization possibility on resources within filters in that we believe there is more room for optimization when zooming our scope into octaves and flattening the DWT tree into a data flow graph with fine granularity, such as MAC-level graphes. In addition, the exploration of MAC-level DWT architectures enables us to develop a systematic approach for synthesizing DWT processors with limited-resource constraints. To start with, we initially paid our attention on intrinsic DWT computation: FIR filtering.

### 4.1.3 Limited-resource FIR filtering

FIR filtering is the basic operation for the computation of DWT. Therefore, In this section, the FIR filtering is discussed first and is performed within the limited resources hardware environment. For an input sequence  $x(n)$  and filter coefficients  $h(n)$ , the output sequence  $y(n)$  is given by Equation 4.1.

$$y(n) = \sum_k x(k)h(n - k) \quad (4.1)$$

According to the Equation 4.1, the data path of FIR filtering can be presented

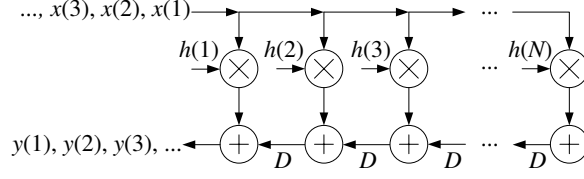


Figure 4.6: The flow graph of FIR filtering.

by a Fully-specified Signal Flow Graph (FSFG) shown in the Fig. 4.6 where  $D$  is the delay element. Because there are now only  $r$  MACs available, the FIR filtering executes parallelly  $r$  MAC-nodes at the maximum within a signal cycle. Due to the degree of parallelism, we retime an  $N$ -tap FIR to the  $r$ -split FSFG shown in the Fig. 4.25. The  $r$ -split FSFG can be obtain by the following retiming steps:(1)Scaling by a scale factor  $F$  determined by Theorem 1, (2)Doing the cut-set retiming[61] at the output edges of the  $(r \cdot i)^{th}$  MAC node,  $i = 1, 2, \dots, \bar{q} - 1$ , where  $\bar{q}$  is determined by Equation 4.2. Therefore, our FIR filtering is rate-optimal for the limited number of MACs. Because the  $r$  MACs can operate parallelly, the computation of FIR filtering can be done in each  $\bar{q}$  cycles called the scheduling period. The scheduling of the FIR filter coefficients in MACs nodes can be folded as shown in the Fig. 4.26 and is called folded scheduling.

$$\bar{q} = \left\lceil \frac{N}{r} \right\rceil \quad (4.2)$$

**Theorem 1** *The scale factor has to satisfy Equation 4.3 for the limited-resource FIR filtering with  $r$  MACs when the folded scheduling is applied.*

$$F \geq \bar{q} = \left\lceil \frac{N}{r} \right\rceil \quad (4.3)$$



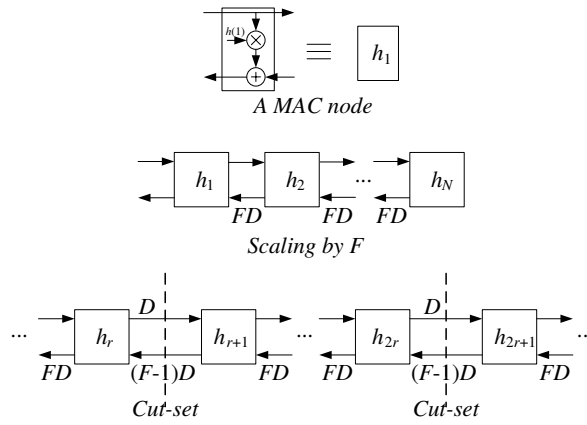


Figure 4.7: The retimed flow graph of FIR filtering.



cycle	MAC1	MAC2	...	MACr
1	$h_1$	$h_2$	...	$h_r$
2	$h_{r+1}$	$h_{r+2}$	...	$h_{2r}$
$\vdots$	$\vdots$	$\vdots$	...	$\vdots$
$n$	$h_{(n-1)r+1}$	$h_{(n-1)r+1}$	...	$h_{nr}$
$\vdots$	$\vdots$	$\vdots$	...	$\vdots$
$q$	$\vdots$	$\vdots$	...	$\vdots$

Figure 4.8: The folded scheduling of FIR filtering.

**Proof:** Assume the scale factor  $F'$  is less than  $F$  at the first. The relationship given by Equation 4.4 is hold, and the FIR filtering can be expanded as shown in Equation 4.5.

$$F' < \bar{q} \quad (4.4)$$

$$y(n) = x(1)h(n) + x(2)h(n-1) + \dots + x(N)h(1) \quad (4.5)$$

According to the r-split FSFG in the Fig. 4.25, the output sequence  $y(n)$  is generated as shown in the Equation 4.6 where we define the notation  $\cdot^{iD}$  or  $[\cdot]^{iD}$  presents the sample that pass through the  $i$  delay elements.

$$\begin{aligned} y(1) &= x(1)h(1) \\ y(2) &= [x(1)h(2)]^{F'D} + x^{iD}(2)h(1) \\ &\vdots \end{aligned} \quad (4.6)$$

Owing to the limitation number of MACs, each input sample  $x(n)$  should be delayed by the factor of  $\bar{q}D$  such that the previous input sample  $x(n-1)$  can multiply with the all FIR filter coefficients as shown in the Fig. 4.25. By the Equation 4.6, the output sample  $y(2)$  will be wrong because that scale factor  $F'$  is smaller than the scheduling period  $\bar{q}$ . The term  $[x(1)h(2)]$  is produced and should be delayed  $\bar{q}$  cycles at least for the later term  $[x(2)h(1)]$  to execute the right addition. The same ill situation will be met in the other output samples. So, in this thesis, the scale factor is chosen to be equal to the scheduling period  $\bar{q}$ .

According to the r-split FSFG of FIR filtering, the scheduling for the limited-

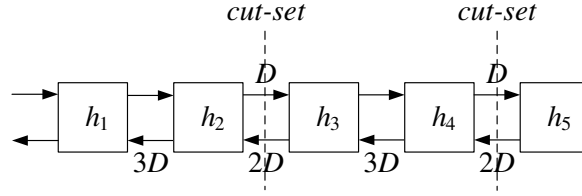


Figure 4.9: The 2-split FSFG of 5-tap FIR filtering.

MAC FIR filtering can be determined and is introduced by an example. Fig. 4.9 shows an example of 5-tap FIR with limited two MACs. The scale factor  $F$  is equal to three and the cut-sets have been done at the output edge of second and fourth MAC nodes to perform 2-split FSFG of FIR filtering. By the 2-split FSFG, the delay elements are replaced by registers to be the output register and input register of each MAC. Therefore, the MAC node can execute the function of Equation 4.7.

$$R_i = x(n)h_i + R_{i+1} \quad i = 1, 2, 3, 4 \quad (4.7)$$

The register can be scheduled following the number of delay elements shown in the  $r$ -split FSFG such that we can get the scheduling of FIR filtering is shown in the Fig. 4.28 where the Coefficients Matrix, Data Matrix, Feedback Matrix, and Accumulated Matrix are called scheduling matrices. The scheduling matrices indicate the  $r$  MACs execute the Equation 4.7 in each cycle respectively and the output sample  $y(n)$  in  $R_1$  will be produced for every three delay.

#### 4.1.4 Scheduling Algorithm of DWT IP

Discrete wavelet transform is a multiresolution analysis tool. It decomposes the signal into detailed and approximation version in each octave and the version of approximation can be decomposed by the same procedure in the next octave

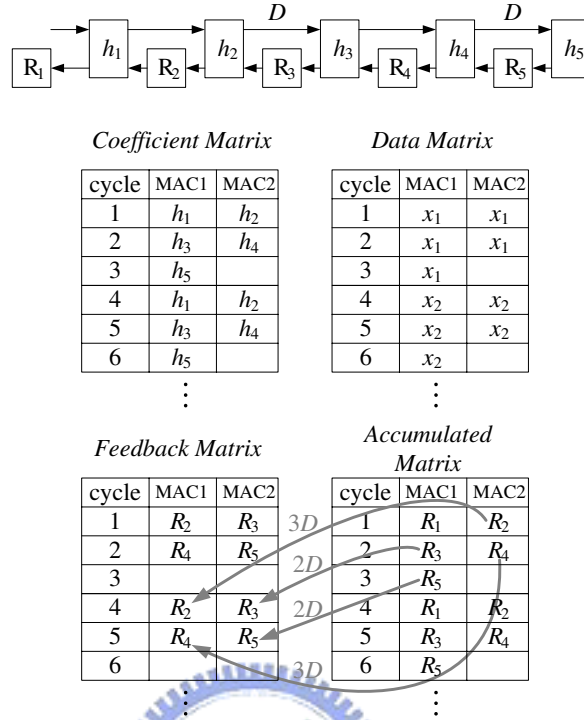


Figure 4.10: The scheduling of FIR filtering.

to perform the multiresolution analysis. Fig. 4.11 shows the part of DWT block diagram where  $L^D$  and  $H^D$  are the lowpass and highpass FIR filter of DWT and the following operation is the two -folded decimator. According to the scheduling of the limited-MAC FIR filtering mentioned in the last section, the filtering operation of the limited-MAC DWT processor can be realized but the 2-folded decimator. The 2-folded decimator could be implemented by the timing of the input samples and the filter coefficients in our r-split FSFG of FIR filtering. Therefore our goal is to find the scheduling matrixces of the DWT signal processing. The algorithm that determined the scheduling matrices of the limited-MAC DWT signal processing is presented by r-split FSFG of limited-MAC FIR and It considers the highpass and lowpass FIR filter at the same time.

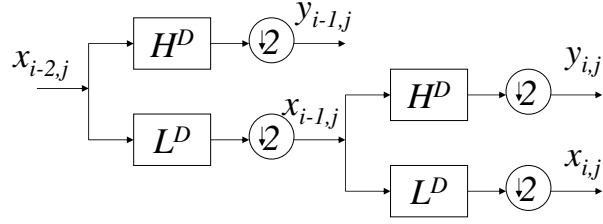


Figure 4.11: The DWT block diagram.

#### Four Scheduling Matrices

The coefficient matrix ( $CM$ ), presents the input schedule of coefficients to the multipliers of the MACs.  $CM$  is a 2-D  $q \times r$  array, where  $q$  is scheduling period that consider the highpass and lowpass FIR filter at the same time and  $r$  is the number of MACs. Assume that the order of wavelet subband filters is  $(m, n)$  where  $m$  is the order of lowpass filter and  $n$  is the order of highpass filter. Given  $r$ ,  $m$ , and  $n$ , the value of scheduling period,  $q$ , can be obtained from Equation 4.8.

$$q = \text{ceil}\left(\frac{m+n}{r}\right) \quad (4.8)$$

It should consider the 2-folded decimator to determine the value of elements in  $CM$ . The scheduling algorithm first finds out the  $CM$  for the single-MAC DWT processor, named as  $CM_1$ , and then fold the  $CM_1$   $r$  times for the DWT processor with  $r$  MACs.  $CM_1$  is a  $q \times 1$  matrix that consists of both lowpass and highpass FIR coefficients. The value of  $CM_1(p)$  is calculated by Equation 4.9, where  $f_m = \text{floor}(\frac{m}{2})$ ,  $f_n = \text{floor}(\frac{n}{2})$ ,  $f = f_m + f_n$ , and  $L_i$  and  $H_i$  are the  $(i-1)^{th}$  order coefficients of lowpass and highpass FIR.

$$CM_1(p) = \begin{cases} L_{2p} + H_{2(p-f_m)} & \text{for } p \leq f \\ L_{2(p-f)-1} \\ + H_{2(p-f+f_m-m)-1} & \text{for } f < p \leq m + n \\ 0 & \text{otherwise} \end{cases} \quad (4.9)$$

To obtain  $CM$  for  $r$ -MAC DWT processor, we first duplicate  $CM_1$   $r$  times to get  $rq \times 1$  matrix  $CM_r$  and then fold  $CM_r$  to the  $CM$ . To fold the  $CM_r$  to the  $CM$  for  $r$ -MAC DWT processor, the scheduling algorithm performs the linear transformation of Equation 4.9 by the index mapping shown in Equation 4.10.

$$CM(i, j) = CM_1(p) \text{ where } p = j + r(i - 1) \quad (4.10)$$

The data matrix( $DM$ ) decides the schedule of input data to the multipliers of MACs. It should still considers the 2-folded decimator for the timing of the input data. Therefore, the calculation of every pair of the output data will consume two input data. Thus, each multiplier will needs two input data within a scheduling period, and the  $DM$  for the single-MAC DWT, named  $DM_1$ , needs provide two addresses for the input data within a scheduling period. The equation 4.11 shows the value of input data indexed by  $b(k)$  and  $a(k)$  during the  $k^{th}$  scheduling period, where  $DM_1^{<k>}$  is the matrix  $DM_1$  in the  $k^{th}$  scheduling period,  $p$  is ranged from 1 to  $q$ ,  $x_{i,j}$  means the  $j^{th}$  input sample of the  $i^{th}$  stage, and  $a(k)$  and  $b(k)$  are calculated by the  $DM$  scheduling algorithm. Fig. 4.12 is the  $DM$  algorithm based on the manner of As-Soon-As-Possible (ASAP), where  $S$  is the stage count of the DWT, and  $Q$  is the integer number. The ASAP scheduling makes the MAC execution priority of the next octave is always higher than that of the previous octave when the input data for the next octave is prepared. The ASAP scheduling will optimize the performance

<i>Initial Conditions: <math>a(1)=0, b(1)=2</math></i>
<i>if <math>[a(k-1) == S-1]</math></i>
<i><math>a(k)=0, b(k)=b(k-1) \times 2^{a(k-1)} + 2;</math></i>
<i>else {</i>
<i>if <math>[b(k-1) == 4Q]</math></i>
<i><math>a(k)=a(k-1)+1, b(k)=b(k-1)/2;</math></i>
<i>else</i>
<i><math>a(k)=0, b(k)=b(k-1) \times 2^{a(k-1)} + 2;</math></i>
<i>}</i>

Figure 4.12: DM scheduling algorithm.

of DWT processor in terms of latency and throughput. Upon calculating  $DM_1$ , the  $DM$  for the multi-MAC DWT can be obtained in the same way as the  $CM$ .

$$DM_1^{<k>}(p) = \begin{cases} (x_{a(k),b(k)-1})_k & \text{for } p \leq f \\ (x_{a(k),b(k)})_k & \text{for } p > f \end{cases} \quad (4.11)$$

The iteration of  $DM$  scheduling algorithm will perform the periodic control strategy for the multi-stage DWT and the control period ( $CP$ ) is  $2^S - 1$  scheduling periods. It means that the processor will costs  $2^S - 1 \times q$  clock cycles to produce the outputs of the  $S^{th}$  stage. In order to perform the periodic timing of the control strategy, the index of the data shown in the  $DM$  scheduling algorithm for  $k = 1, \dots, 2^S - 1$  is rearranged as  $(a(k), u(i)w + b(k))$  by Equation 4.12.

$$\begin{aligned} u(i) &= \max_{a(k)=i} b(k) \\ w &= k \pmod{2^S - 1} \end{aligned} \quad (4.12)$$

The controller implementation of the DWT processor, therefore, will have  $S$  states, and each state represents the source of the data. For example,  $a(k) = 0$  means the data sequence comes from the original input data. When  $a(k)$  is larger than

zero, the data sequence is the output feedback from the  $a(k)^{th}$  stage. Here  $a(k)$  is finite. The output feedback can be either detail or approximation coefficients. Both are allowed to perform the different subband analysis.

The 3<sup>rd</sup> and 4<sup>th</sup> matrices of the scheduling presented here are the feedback matrix ( $FbM$ ) and the accumulated matrix ( $AccM$ ).  $FbM$  shows the location where the MAC should load from and  $AccM$  indicates the location where stores the output data of MACs. Both matrices access the same set of output registers,  $R_{a(k),i}$ , in the output register file and their elements are addresses of associated source registers and destination registers. The  $a(k)$  defined in  $DM$  algorithm represents the output register bank is used at the  $(a(k) + 1)^{th}$  stage. The address calculation is shown in Equation 4.13 and Equation 4.14, where  $a_1 = 2p$ ,  $a_2 = 2(p - f_m)$ ,  $a_3 = 2(p - f) - 1$ ,  $a_4 = 2(p - f + f_m - m) - 1$ ,  $\hat{u}_1(l) = u(l - 1) - u(l - m)$ ,  $\hat{u}_2(l) = u(l - 1) - u(l - n)$ ,  $p = j + r(i - 1)$ , and the function  $u(\cdot)$  is the unit step sequence..

$$FbM(i, j) = \begin{cases} AccM(i, j) + 1 & \text{for } AccM(i, j) \neq m + n \\ & \text{or } m + n + 1 \\ 0 & \text{otherwise} \end{cases} \quad (4.13)$$

$$AccM(i, j) = \begin{cases} a_1 \times \hat{u}_1(a_1) \\ + (a_2 + m) \times \hat{u}_2(a_2) & \text{for } p \leq f \\ a_3 \times \hat{u}_1(a_3) \\ + (a_4 + m) \times \hat{u}_2(a_4) & \text{for } f < p \leq m + n \\ 0 & \text{otherwise} \end{cases} \quad (4.14)$$

After determining the four scheduling matrices, the DWT operation will start



at the first row of the first scheduling period. Equation 4.15 shows the computation in the first scheduling period,  $k = 1$ ,  $a(k) = 0$ .

$$\begin{aligned}
& \text{for } i = 1 \text{ to } q \{ \\
& R_{0,AccM(i,1)} = CM(i, 1) \times DM(i, 1) + R_{0,FbM(i,1)} \\
& R_{0,AccM(i,2)} = CM(i, 2) \times DM(i, 2) + R_{0,FbM(i,2)} \\
& \quad \vdots \\
& R_{0,AccM(i,r)} = CM(i, r) \times DM(i, r) + R_{0,FbM(i,r)} \\
& \}
\end{aligned} \tag{4.15}$$

Then the DWT operation will do the same process for the other scheduling period,  $k$ . After finishing one scheduling period, a detail coefficient and an approximation coefficient of the  $(a(k) + 1)^{th}$  stage will be produced at the output registers labeled  $R_{a(k),1}$  and  $R_{a(k),m+1}$ .

## Performance

The performance of the proposed DWT processor based on the limited resource scheduling is dependent on the number of the MACs. There are four performance metrics we concern in the DWT processor design: latency, the throughput rate, the maximum sampling rate, and the number of registers.

According to the ASAP scheduling, the first output data at the  $s^{th}$  stage needs  $2^s$  original signal samples to generate the first output data. The latency,  $L$ , of the DWT processor is equal to  $\lceil q(2^S - 2) + (\frac{q}{2}) \rceil$  clock cycles, and the throughput rate,  $TH$ , of the DWT processor is equal to  $TH = \frac{2}{(2^S - 1) \times q}$  in that two input data at the  $S^{th}$  stage can be obtained every  $2^S - 1$  iterations.

Now let us consider the maximum sampling rate. To avoid overflowing of the input buffer, the minimum sample period is equal to  $\frac{(2^S - 1) \times q}{2^S}$  clock cycles. Thus,

the sampling rate is bounded at  $\frac{2^S}{(2^S-1) \times q}$  samples per clock cycle.

Regarding the number of registers, we need determine the register requirement for each stage. For each stage, an iteration requires  $m + n$  registers,  $R_1 \sim R_{m+n}$ . Since the number of MACs is limited, data in registers cannot be cleared until finishing the following consecutive stages. Thus, the total number of registers is equal to  $S \times (m + n)$  words.

### The size of memory

According to the MAC-level architecture, the ASAP principle can be realized and conclude that the sequence of inter-octave will not need to save for the computation of the next octave. It introduces that there are only 2-word size of memory between the octaves in our proposed algorithm. Therefore, the memory will include the coefficients register file of  $(m + n)$ -word size, MAC input data register file of  $r$ -word size, output register file of  $S \times (m + n)$ -word size and the inter-octave register of 2-word size . The total size of memory will need  $(m + n) + r + S \times (m + n) + 2$  words. The coefficients register file, MAC input data register file and output register file are needed for iteration operation of filtering. They can be regards as the registers for the iteration operation of filtering instead of the operation of DWT.

### Scheduling Algorithm of IDWT

The inverse discrete wavelet transformation can be performed in the same MAC-level architecture based on the same limited-resource architecture. Its scheduling algorithm is still developed from our r-split FSFG of FIR filter. For the hardware implementation, all we need to change are the controller and two extra adders.

Fig. 4.13 shows the part of the block diagram of IDWT. The reconstruction signal of the  $i^{th}$  stage,  $\tilde{x}_{i,j}$  and the wavelet coefficients,  $y_{i,j}$  are upsampling by two and filtering by lowpass filter, L', and highpass filter, H'. The reconstruction signal

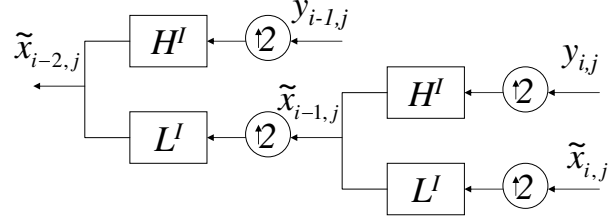


Figure 4.13: The block diagram of IDWT.

of the  $(i - 1)^{th}$  stage,  $\tilde{x}_{i-1,j}$ , will be the summation of the both filtering signals. Notice that the  $\tilde{x}_{S,j} = x_{S,j}$ , and the  $S$  is the stage count of DWT.

To perform the computation of IDWT based on the same architecture of DWT, a little modification of the scheduling algorithm for DWT is needed. The four scheduling matrices,  $CM$ ,  $DM$ ,  $FbM$ , and  $AccM$  are still used here. Both of the  $CM$  and  $AccM$  are the same, but  $DM$  and  $AccM$  are modified for the computation of IDWT. Equation 4.16 shows the  $DM$  scheduling algorithm. Notice that each data here should multiply with every filter coefficients. Therefore, the location of wavelet coefficients  $x_{i,k}$  and  $y_{i,k}$  in  $DM$  is corresponding to the position of lowpass and highpass filter coefficients in  $CM$ . The term in the Equation 4.17 to plus two will ignore the zero input data to perform the interpolation.

$$DM^{<k>}(p) = \begin{cases} x_{i,k} & p \leq f_m, f < p \leq f + m - f_m \\ y_{i,k} & f_m < p \leq f, f + m - f_m < p \leq m + n \\ 0 & otherwise \end{cases} \quad (4.16)$$

$$FbM(i, j) = \begin{cases} & \text{for } AccM(i, j) \neq m+1, \\ AccM(i, j) + 2 & \begin{matrix} m+2, \\ m+n+1 \text{ or} \\ m+n+2 \end{matrix} \\ 0 & \text{otherwise} \end{cases} \quad (4.17)$$

### Example of DWT scheduling

Now we use DWT with Daubechies(9,7) as an example. Given the number of MACs is three, The parameters can be determined as follows:

$$r = 3, q = 6, f_m = 4, f_n = 3, f = 7$$

According to the proposed scheduling algorithm, the scheduler produces four scheduling matrices as shown in the Fig. 4.14. The  $CM$  and  $DM$  are produced to proceed the multiplication process as like  $L_i B_j$ . The  $FbM$  and  $AccM$  will address the output registers for each scheduling period to perform the summation equation.

The  $B_1$  and  $B_2$  in  $DM$  are the FIFO registers that buffer two new samples whose timing are determined by the  $DM$  algorithm for each scheduling period. Fig. 4.15 shows the data that is buffured into the  $B_1$  and  $B_2$  at the first and  $k^{th}$  stage. The periodic control strategy is shown in Fig. 4.16 after rearranging the index of the buffered data.

Both outputs of the  $3^{rd}$  stage will be produced at each  $w$ . For the first scheduling period,  $k = 1$ , The samples in  $B_1$  and  $B_2$  are  $x_{0,1}$  and  $x_{0,2}$ . The first approximation output sample at the first stage,  $y_L(1) = L_2 x_{0,1} + L_1 x_{0,2}$ , will be produced at  $t = 3$  in  $R_{(0,1)}$ , and the first detail sample at the first stage will be

MAC \ time	#1	#2	#3
6t+1	L <sub>2</sub>	L <sub>4</sub>	L <sub>6</sub>
6t+2	L <sub>8</sub>	H <sub>2</sub>	H <sub>4</sub>
6t+3	H <sub>6</sub>	L <sub>1</sub>	L <sub>3</sub>
6t+4	L <sub>5</sub>	L <sub>7</sub>	L <sub>9</sub>
6t+5	H <sub>1</sub>	H <sub>3</sub>	H <sub>5</sub>
6t+6	H <sub>7</sub>	0	0

(a) *CM*

MAC \ time	#1	#2	#3
6t+1	B <sub>1</sub>	B <sub>1</sub>	B <sub>1</sub>
6t+2	B <sub>1</sub>	B <sub>1</sub>	B <sub>1</sub>
6t+3	B <sub>1</sub>	B <sub>2</sub>	B <sub>2</sub>
6t+4	B <sub>2</sub>	B <sub>2</sub>	B <sub>2</sub>
6t+5	B <sub>2</sub>	B <sub>2</sub>	B <sub>2</sub>
6t+6	B <sub>2</sub>	0	0

(b) *DM*

MAC \ time	#1	#2	#3
6t+1	R <sub>s,3</sub>	R <sub>s,5</sub>	R <sub>s,7</sub>
6t+2	R <sub>s,9</sub>	R <sub>s,12</sub>	R <sub>s,14</sub>
6t+3	R <sub>s,16</sub>	R <sub>s,2</sub>	R <sub>s,4</sub>
6t+4	R <sub>s,6</sub>	R <sub>s,8</sub>	0
6t+5	R <sub>s,11</sub>	R <sub>s,13</sub>	R <sub>s,15</sub>
6t+6	0	0	0

(c) *FbM*,  $s=a(k)$

MAC \ time	#1	#2	#3
6t+1	R <sub>s,2</sub>	R <sub>s,4</sub>	R <sub>s,6</sub>
6t+2	R <sub>s,8</sub>	R <sub>s,11</sub>	R <sub>s,13</sub>
6t+3	R <sub>s,15</sub>	R <sub>s,1</sub>	R <sub>s,3</sub>
6t+4	R <sub>s,5</sub>	R <sub>s,7</sub>	R <sub>s,9</sub>
6t+5	R <sub>s,10</sub>	R <sub>s,12</sub>	R <sub>s,14</sub>
6t+6	R <sub>s,16</sub>		

(d) *AccM*,  $s=a(k)$

Figure 4.14: Four scheduling Matrices.

MAC \ time	#1	#2	#3
1	1	2	3
1	$x_{0,1}$	$x_{0,1}$	$x_{0,1}$
2	$x_{0,1}$	$x_{0,1}$	$x_{0,1}$
3	$x_{0,1}$	$x_{0,2}$	$x_{0,2}$
4	$x_{0,2}$	$x_{0,2}$	$x_{0,2}$
5	$x_{0,2}$	$x_{0,2}$	$x_{0,2}$
6	$x_{0,2}$	0	0

(a)  $DM^{(1)}(p)$

MAC \ time	#1	#2	#3
6k-5	1	2	3
6k-4	2	3	1
6k-3	3	1	2
6k-2	4	3	2
6k-1	5	2	3
6k	6	0	0

(b)  $DM^{(k)}(p)$

$p=j+3(i-1)$   
 $m=9, n=7, r=3, q=6$   
 $p=1 \sim rq$

Figure 4.15: The data flow in DM.

Scheduling Period ( $k$ )	Buffer#1	Buffer#2
$7w+1$	$x_{0,8w+1}$	$x_{0,8w+2}$
$7w+2$	$x_{0,8w+3}$	$x_{0,8w+4}$
$7w+3$	$x_{1,4w+1}$	$x_{1,4w+2}$
$7w+4$	$x_{0,8w+5}$	$x_{0,8w+6}$
$7w+5$	$x_{0,8w+7}$	$x_{0,8w+8}$
$7w+6$	$x_{1,4w+3}$	$x_{1,4w+4}$
$7w+7$	$x_{2,2w+1}$	$x_{2,2w+2}$

Stage count : 3  
 Output period :  $2^3 - 1 = 7$   
 $w = k \bmod 7$

Figure 4.16: The data with rearranged indexing.

produced at  $t = 5$  in  $R_{0,10}$ . The first scheduling period is performed until  $t = 6$ , and then the dataflow shown in the Fig.4.16 will be buffered into the  $B_1$  and  $B_2$  in  $k$  order. After executing the  $7^{th}$  scheduling period,  $k = 7$ , the first output of  $3^{rd}$  stage,  $x_{3,1}$  will be generated. Therefore, the DWT with Daubechies(9,7) and 3 MACs can be performed by the results of the scheduling algorithm.

#### 4.1.5 Limited-Resource DWT Architecture

Fig. 4.17 shows the proposed architecture based on the scheduling algorithm mentioned above. The architecture is composed of  $r$  MACs, a scheduler and register banks. The MACs are fed data from the coefficient bus, MAC data bus, and internal feedback bus. As shown in Fig.4.18, the access of output register banks is controlled by the write enable bus,  $Wr\_en$ , and output enable bus,  $O\_en$ . The signals of  $Wr\_en$  bus are produced based on the  $AccM$  to decide the destination registers for storing the output data. The signals of  $O\_en$  bus are used to enable output registers according to the  $FbM$  and select outputs of  $R_{a(k),1}$  and  $R_{a(k),m+1}$  to deliver the coefficients for each stage.

Fig. 4.19 illustrates register banks connecting to the coefficient bus and data

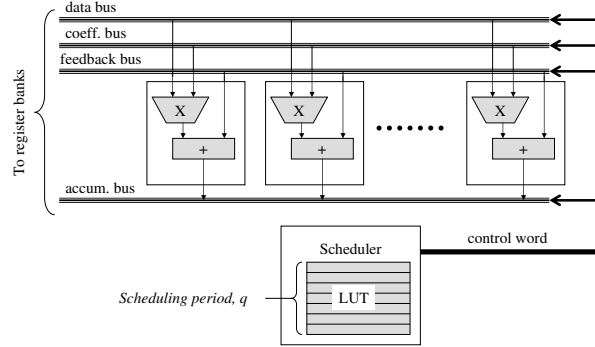


Figure 4.17: The limited-resource DWT architecture.

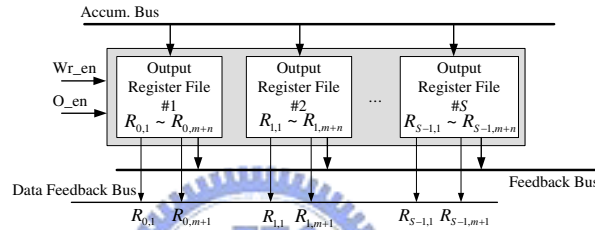


Figure 4.18: The output register bank.

bus. In the coefficient register bank, the  $Wr\_en$  is asserted when setting up wavelet function and the signals of  $O\_en$  bus enable  $r$  coefficient registers based on the  $CM$ . In the input register bank, the signals of  $Wr\_en$  bus decide which input register needs to be updated based on the  $DM$  and the signals of  $O\_en$  bus enable the input of MACs based on the  $DM$ . Fig. 4.20 shows the feedback register banks and the data register bank. The former loads the data from the data feedback bus and the later fetches the input data. The data in both of them are loaded into the input data bus by the  $O\_en$  signals based on the periodic control strategy of  $DM$  algorithm to perform the multi-stage DWT. All of the signals of write and output enable buses are produced by controller according to the proposed scheduling algorithm. The controller design is an important part for implementation of DWT processor.

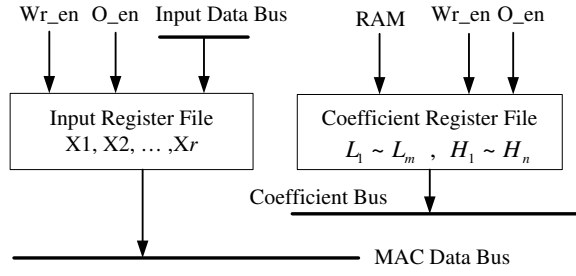


Figure 4.19: The coefficient and input register banks

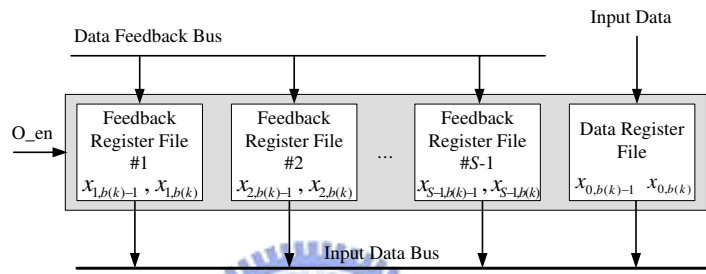


Figure 4.20: The feedback and Data register banks

Because the proposed scheduling algorithm performs the regular behavior for each step, such as periodic feature, it leads to a statistical scheduling. Therefore, the controller can be implemented by look-up-table intuitively. Furthermore, if we want to compute another wavelet function, the work will be just to update the look-up-table (or re-synthesize the controller) and to fetch new coefficients from memory.

Comparing to conventional DWT architectures, the proposed MAC-level architecture can further reduce the overall memory requirement and increasing hardware utilization over filter-based architectures by exploring the redundancy between the lowpass and highpass filters. First, the proposed architecture outperforms conventional architectures in terms of memory requirement because we flatten the DWT computation with fine granularity. The conventional architectures deal with the memory size at filter level, so they have a storage unit to hold  $N$  data for each



octave and the overall memory size is proportional to  $N$ . Second, the proposed architecture has high degree of scalability. Paper [72] has the highest hardware utilization for its systolic implementation, but the irregular memory access makes it lose scalability. Paper [67] has regular implementation, but the hardware utilization is lower than the proposed architecture and their architecture can only apply for certain DWT applications, e.g. the order of FIR has to be less than 8.

#### 4.1.6 Implementation of DWT IP

##### IP Synthesizer

As shown in Fig.4.4, the IP synthesizer generates DWT IP based on the architecture constraints and DWT parameters. The architecture constraints are conditions for implementation of target IP.  $b$  declares the bit-width of input data and determines the precision of function units.  $r$  is the constraint for MAC components. The synthesis will generate  $r$  MACs in the data path.  $ram\_size$  and  $data\_vol$  are parameters for input buffer generator and I/O address generator. Given the architecture constraints, the synthesizer accordingly produces synthesizable VHDL codes for MAC array, input buffer, I/O address generator and interfaces in between. Then, the IP synthesizer calculates four scheduling matrices according to the DWT parameters.

There are four DWT parameters for DWT configuration:  $S$ ,  $dim$ ,  $m$  and  $n$ .  $S$  is the number of octaves which will determine CP for scheduler. The scheduler will use the value of CP to determine when to produce the outputs of the last stage.  $dim$  indicates the dimension of DWT, where 1 for one-dimensional DWT and 2 for two-dimensional DWT. The two-dimensional DWT synthesis requires one more step than one-dimensional DWT synthesis. When synthesizing two-dimensional, the synthesizer will produce transpose memory for row-column DWT computations.  $m$  and  $n$  determine the order of highpass FIR and lowpass FIR, respectively. Given  $m$

and  $n$ , the IP synthesizer calculates four scheduling matrices and then convert the matrices into a look-up table. The look-up table then becomes part of scheduler.

### Synthesis result

The DWT processor with four MACs using the Daubechies (9,7) wavelet basis has been implemented at 50 MHz of the operation frequency. In this section, we use TSMC 0.35um 1P4M process to synthesize the DWT processor. Fig. 4.21 shows the chip layout of the DWT processor. The processor contains eleven components as labeled in Fig. 4.21. The components are: (1) the row stream interface unit that provides the external memory addresses and control signals for 1-D DWT input data or 2-D DWT row data, (2) the ping-pong buffer that stores the two samples of  $DM$  for each scheduling period, (3) the arithmetic unit that contains 4 MACs and one-of-two downsampling operation, (4) the memory manage unit that allocates memory space for the output data, (5)-(6) column stream interface unit that provides the external memory addresses and control signals for 2-D DWT column data, (7) the control unit that controls the operation of data path, (8) the output latches that synchronizes the output data stream with the control signal, (9) glue logic, and (10)-(11) two 16-bit on-chip memory blocks. With four MACs and 16Kbytes SRAM, the core size is  $4238 \times 4238 \mu m^2$ . According to the measurement, the total size of parts (7) and (8) is  $269410 \mu m^2$  and hence the overhead of scheduler is as small as 1.5%.

#### 4.1.7 Summary

A highly scalable VLSI architecture and a limited-resource scheduling algorithm for MAC-level DWT processor has been presented. The scheduling algorithm has been successfully proven that a variety of DWT processors can be efficiently realized by tuning the parameters. Given the architecture constraints and DWT parameters, the scheduling algorithm can generate four schedule matrices and en-

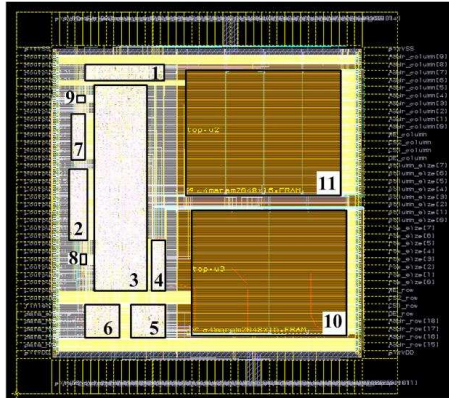


Figure 4.21: The Layout of DWT processor

able the data path to perform the DWT computation. Owing to high degree of scalability and flexibility, an automated DWT processor generator has been developed based on the limited-resource scheduling algorithm. The generated DWT SIP can be embedded into a SOC or mapped to program codes for commercial off-the-shelf DSP processors with programmable devices such as FPGA and CPLD.

## 4.2 Folding Technique for High-Order FIR Filter Implementation

### 4.2.1 Introduction

FIR filtering is an essential function in most DSP applications, such as telecommunication and multimedia systems. Its guaranteed stability and simple structure make FIR itself a popular technique for removing unwanted parts of signal. For quality-sensitive applications, the number of FIR taps is normally large, ranges from tens to hundreds. However, long-length (or high-order) FIR filters may result in costly hardware and hence severe power consumption problem. Furthermore, a long-length FIR architecture may suffer from the clock-skew problem. Besides these

reasons, FIR filtering is also one of waterfall-like processing procedures which means all processing tasks in the FIR filtering are in a linear progression. The simple structure could help to explain two proposed folding techniques. Therefore, two proposed folding techniques, demonstrated by a high-order FIR filter, have been presented to illustrate, compare, and conclude the tradeoffs between the limited number of processing elements and performance. Two systematic folding techniques are derived from the idea of resource sharing. One of them involves the resource sharing of grouping, and the other involves the resource sharing of segmentation.

Many approaches have been proposed to reduce the hardware complexity of FIR filtering [78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92]. They can be classified into three categories: multiplier reduction [86, 81, 82, 89, 85, 88], multiplierless realization [90, 83, 78], and resource sharing (or folding) [87, 79, 84, 91, 92]. The first category can reduce the number of multiplications at the expense of increased number of additions or pre-/post-processing units. The second category mainly uses the distributed arithmetic (DA) technique to dismiss multipliers. Both categories are able to significantly reduce computational complexity but lack flexibility for cost-optimal FIR implementation. The third category is proposed for reducing the datapath cost when the throughput rate is lower than the maximum speed at which a full-length datapath can operate. That is, the folding techniques intend to trade excessive speed for datapath cost with high degree of flexibility.

[87] and [79] present the first systematic transformation technique to fold DSP algorithms. The folding technique was proposed for efficient resource sharing to meet both throughput and resource constraints, while it allows identical operations to be time-multiplexed by the same circuit unit. The technique is attractive not only for its area saving but also for the alleviation of the clock-skew problem. Following [87] and [79], [84] and [91] present folded FIR architectures for variable folding factor and bit-level pipelined implementations, respectively. Both of them take

the advantage of folding technique to meet their objectives. The existed folded architectures mainly focus on the utilization of processing elements or data path; however, few emphasize the cost of registers and control units. Note that when the processing elements have been carefully allocated for performance constraints registers and control units will become the main issues to the efficiency of folding techniques. Thus, this chapter presents two folding techniques based on algebraic and structural transformations. As shown in the estimation results, the proposed techniques can outperform the others in terms of cost and power dissipation.

## 4.2.2 Candidates of Folding Techniques

There are five folding techniques being targeted as candidates. Three of them are published in [84], [87], and [92] while the others are proposed in this dissertation at the first time. Later, we will use the following notations to explain the techniques.

$m$  = the bitwidth of input sample

$b$  = the bitwidth of coefficient

$K$  = the number of filter taps

$r$  = the number of multiplier-adders (MAs)

$f$  = the folding factor ( $f = \lceil \frac{K}{r} \rceil$ )

$T_{MA}$  = the execution time of MA unit

$xput_{spec}$  = the specified throughput

## 4.2.3 Existing Folding Techniques

Article [87] presents a systematic transformation technique to fold DSP algorithms. At first, it maps the nodes of the signal flow graph (SFG) into folding elements; each of them is composed of one or more identical operations. Given the folding factor  $f$  and a  $K$ -tap FIR, there exist  $\lceil K/f \rceil$  (or  $r$ ) processing elements

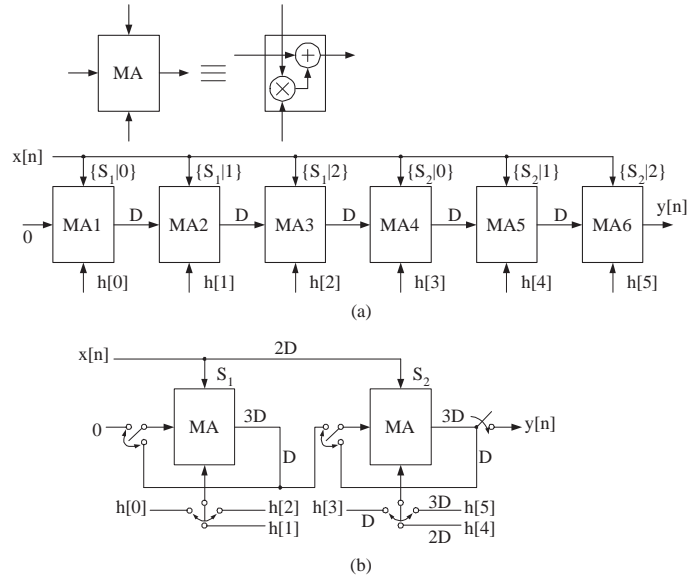


Figure 4.22: A folding example of FIR filter. (a) A 6-tap FIR filter in the transposed form. (b) The folded architecture of Fig. 4.22(a) by using the technique presented in [87].

(PEs) (or MA) in the target architecture, and up to  $f$  equally structured folding elements will be assigned to the same PE. The executing order of folding elements, being assigned to the same PE, is determined by the folding-order number; the folding-order number is set by the task scheduling of folding elements. Finally, the technique requires delay-insertion to synchronize the input of coefficients. The synchronization is realized by registers clocked at the sampling rate, a multiplexer, and a MOD  $f$  counter. Fig. 4.22 illustrates an example with  $K=6$  and  $r=2$ . The paper [92] presents a low power synthesis approach by using the unfolding technique with coefficient reordering. However, it uses the same hardware mapping strategy as [87]. Unfolding a filter, by a factor  $F$  will result in a  $F$ -parallel filter topology. The memory requirement is approximatively proportional to the unfolding factor  $F$ .

Following the systematic folding technique, article [84] presents the folded bit-plane FIR architecture and folds the FIR filter algorithm bitwisely. The folded

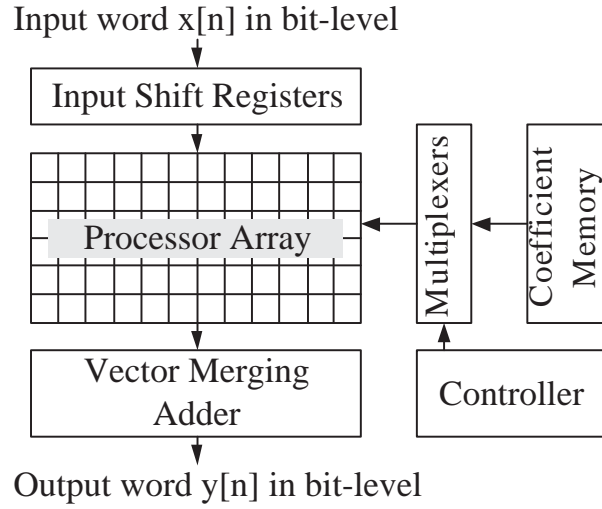


Figure 4.23: The diagram of the folded bit-plane FIR architecture.

bit-plane FIR architecture enables the implementation of changeable folding factor onto a fixed-size systolic array. The systolic array reads the input word  $x[n]$  in the manner of bit-serial, and generates the partial products bit-by-bit with the coefficients. The partial products will then be accumulated for the calculation of the result  $y[n]$ . In the folded bit-plane architecture, the folding factor  $f$  is equal to the programmable bit-width of coefficients and the throughput of the folded architecture can be increased by reducing the bit-width of coefficients. Fig. 4.23 is the diagram of the folded bit-plane FIR presented in [84].

### The architecture of the serial-in folded FIR filter

The transfer function of the  $K$ -tap FIR filter can be reformulated as Eq. 4.18 by recursively calculating  $f$ -tap FIR filtering.

$$\sum_{k=0}^{K-1} h_k \cdot z^{-k} = \sum_{i=0}^{r-1} z^{-fi} \cdot \sum_{j=0}^{f-1} h_{fi+j} \cdot z^{-j} \quad (4.18)$$

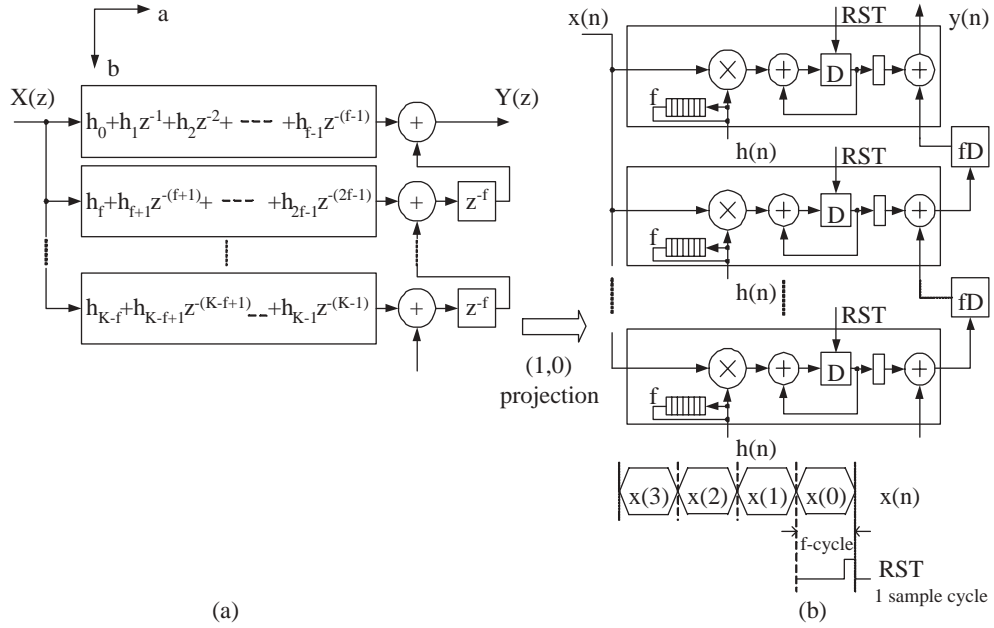


Figure 4.24: (a) The SFG of the reformulated  $K$ -tap FIR filter. (b) The serial-in folded architecture.

where  $f = \frac{K}{r}$ . From Eq. 4.18, the  $K$ -tap FIR filter is composed of  $r$  short-length FIRs ( $\sum_{j=0}^{f-1} h_{fi+j} \cdot z^{-j}$ ), and can be realized by the SFG as shown in Fig. 4.24(a). Given the folding factor  $f$ , the SFG can be then mapped onto  $r$  PEs by (1,0) projection. The folded architecture in Fig. 4.24(b) is called the serial-in folded FIR. In the architecture, each PE serially executes the short-length FIR on input samples and the coefficients  $h_{fi+j}$  are circular in cyclic shift registers of PEs. As shown in Fig. 4.24(b), the clock rate is  $f$  times of the sampling rate and the signal “RST” is asserted every  $f$  clock cycles. The signal “RST” is used to clear the accumulator before the new sample comes. Finally, the  $f$ -stage shift registers  $fD$  are used to buffer the results of short-length FIRs and realize the function of  $z^{-fi}$ .



### The architecture of the parallel-in folded FIR filter

For an input sequence  $x(n)$  and filter coefficients  $h(n)$ , the output sequence  $y(n)$  is given by Eq. 4.1.

According to the Eq. 4.1, the data path of FIR filtering can be presented by an SFG shown in the Fig. 4.6 where  $D$  is the delay element. Because there are only  $r$  MAs available for the folding factor  $f$ , the FIR filtering can execute  $r$  MA-nodes in parallel at the maximum within a signal cycle and requires  $\lceil \frac{K}{r} \rceil$  (or  $f$ ) cycles to finish an iteration. Hence, given  $r$  MAs, the iteration period is bounded by  $f$  cycles and  $f$  is the folding factor. To fold the execution of the  $K$ -tap FIR by  $f$ , we reconstructed a  $K$ -tap FIR to the  $r$ -split SFG as shown in the Fig. 4.25. At first, the delay elements of the original SFG are scaled by the folding factor  $f$  so each iteration of the  $K$ -tap requires  $f$  cycles. Then, we performed the retiming transforms on the edges in the cut-sets as shown in Fig. 4.25. The cut-set is used to segment the retimed graph into  $f$  subgraphs( $r$ -split graph). ( $f = \lceil \frac{K}{r} \rceil$ ). Each subgraph has  $r$  or less than  $r$  MA operations. Afterward those  $f$  subgraphs would be executed in the same  $r$  MAs by turns. The terms  $r$  and  $f$  are defined as the number of hardware resource of MAs and how many equally structured operations will be assigned to the same hardware, respectively. To execute the subgraphs in order, the FIR coefficients are scheduled as illustrated in the Fig. 4.26. Because the input samples are read into the MA array in parallel, the folding technique is called the parallel-in folded FIR.

Fig. 4.27 illustrates the parallel-in folded FIR architecture, where  $w$  is the bitwidth of data bus. Given  $r$  MAs, the folding factor becomes  $f$  and the  $K$ -tap FIR requires  $f$  cycles for an iteration. There are  $K$   $b$ -bit registers in coefficient register bank. The configuration is composed of  $r$  groups. ( $f = \lceil \frac{K}{r} \rceil$ ). Each group contains  $f$  filter coefficients and simultaneously provides them one by one to the corresponding MA unit according to the scheduled order shown in Fig.6. The

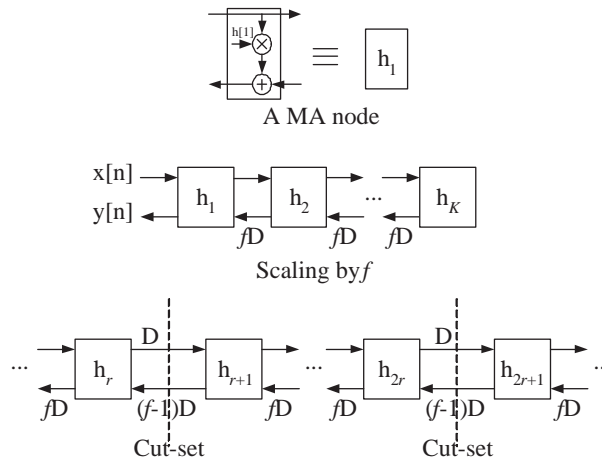


Figure 4.25: The  $r$ -split FIR filtering.

cycle	MA1	MA2	...	MA $r$
1	$h_0$	$h_1$	...	$h_{r-1}$
2	$h_r$	$h_{r+1}$	...	$h_{2r-1}$
$\vdots$	$\vdots$	$\vdots$	...	$\vdots$
$f$	$h_{(f-1)r}$	$h_{(f-1)r+1}$	...	$h_{fr-1}$
$\vdots$	$\vdots$	$\vdots$	...	$\vdots$
$\vdots$	$\vdots$	$\vdots$	...	$\vdots$

Figure 4.26: The scheduling of FIR coefficients for the parallel-in folded technique.

register file is used to buffer the output of MAs so that the subgraphs of  $r$ -split FIR can be executed recursively. The counter is used to schedule the inputs and outputs of MA array for the FIR working correctly. Fig. 4.9 and Fig. 4.28 demonstrate the scheduling for an example of 5-tap FIR with two MAs. Given  $r = 2$ , the folding factor  $f$  is equal to three and we can obtain the 2-split SFG of Fig. 4.9. To map the SFG to the parallel-in architecture, we replaced the delay elements with registers and generated the scheduling as shown in Fig. 4.28. Note that each delay element is not necessary to be realized as a register if one can properly schedule the data storage. Following the scheduling, the MA units can perform the function of Eq. 4.7 in each cycle and produce the output  $y(n)$  in  $R_1$  every three cycles.

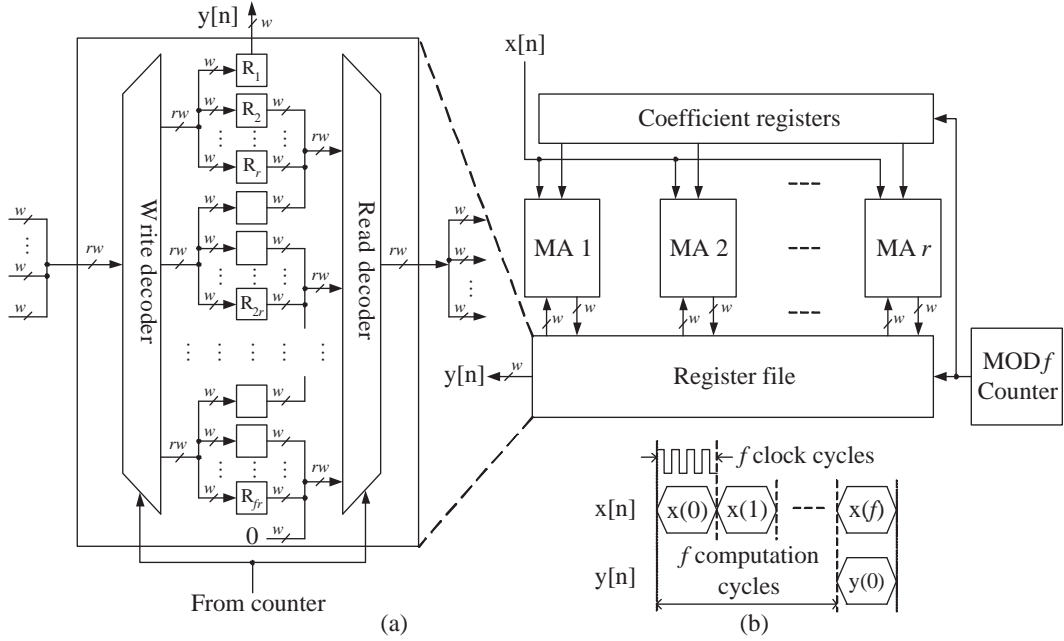
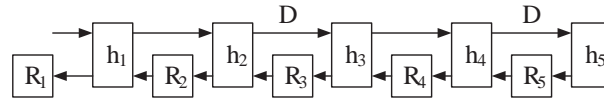


Figure 4.27: (a) The architecture of the parallel-in folded FIR filter, and (b) the timing diagram.

#### 4.2.4 Comparison Results

Given the folding factor  $f$  for  $K$ -tap FIR, all the folding techniques require the same number of MAs and has the same throughput rate and, hence, the efficiency of folding techniques is determined by size and power consumption of memory. At the stage of high-level synthesis, we consider the number of D-type flip-flops (DFFs) as the size of memory and the access number of DFFs per iteration as the power dissipation of memory. When making the comparison, we set the bitwidth of data bus  $w$  as  $m + b + \lceil \log_2 K \rceil$  for full-precision FIR calculation.

Eq. 4.19 formulates the number of DFFs required by [87]. In [87], there are  $r$  MAs located on the accumulated loop. In Eq.4.19, we use the term  $\alpha_0$  to express the number of DFFs for coefficient storage. Because each MA has  $(f + 1)$  registers with full precision to hold the accumulating results, the term  $\alpha_1$  means the total number



Coefficient			Input Sample		
cycle	MA1	MA2	cycle	MA1	MA2
1	$h_1$	$h_2$	1	$x_1$	$x_1$
2	$h_3$	$h_4$	2	$x_1$	$x_1$
3	$h_5$		3	$x_1$	
4	$h_1$	$h_2$	4	$x_2$	$x_2$
5	$h_3$	$h_4$	5	$x_2$	$x_2$
6	$h_5$		6	$x_2$	
$\vdots$			$\vdots$		

Input Register			Output Register		
cycle	MA1	MA2	cycle	MA1	MA2
1	$R_2$	$R_3$	1	$R_1$	$R_2$
2	$R_4$	$R_5$	2	$R_3$	$R_4$
3			3	$R_5$	
4	$R_2$	$R_3$	4	$R_1$	$R_2$
5	$R_4$	$R_5$	5	$R_3$	$R_4$
6			6	$R_5$	
$\vdots$			$\vdots$		

Figure 4.28: The scheduling of FIR filtering.

of DFFs for  $r$  MAs. The term  $\alpha_2$  represents the number of DFFs for multiplexing the input samples to MAs. Because the coefficients are cyclically read by MAs, the number of registers for multiplexing coefficients (in each PE) increasingly varying with the MA changes. Hence the total number of registers for multiplexing coefficients can be calculated by  $q$  and the term  $\alpha_3$  represent the number of DFFs for multiplexing coefficients. Finally, the controller can be implemented as a counter and the term  $\alpha_4$  counts the number of DFFs required in the controller. Eq. 4.20 shows the number of DFFs required by [92]. The unfolding factor,  $F$ , will result in a  $F$ -parallel filter topology. The memory requirement is approximatively proportional to the unfolding factor  $F$ .

$$\begin{aligned}
\#DFF_{[87]} &= \underbrace{Kb}_{\alpha_0} + \underbrace{r(m + b + \lceil \log_2 K \rceil)}_{\alpha_1} (f + 1) \\
&+ \underbrace{(K - f)m}_{\alpha_2} + \underbrace{bq}_{\alpha_3} + \underbrace{\lceil \log_2 f \rceil}_{\alpha_4}, \text{ where } q = \sum_{i=1}^{K-f} i
\end{aligned} \tag{4.19}$$

$$\begin{aligned}
\#DFF_{[92]} &= Kb + F(r(m + b + \lceil \log_2 K \rceil))(f + 1) \\
&+ (K - f)m + bq + \lceil \log_2 f \rceil, \text{ where } q = \sum_{i=1}^{K-f} i
\end{aligned} \tag{4.20}$$

Eq. 4.21 formulates the number of DFFs required by the folded bit-plane FIR architecture [84]. The folded bit-plane FIR architecture requires  $K \times (m + b + \lceil \log_2 K \rceil)$  PEs. Each PE performs 1-bit addition and needs two DFFs for the carry-out and sum signals. In addition, the length of the input shift register is  $(m + b + \lceil \log_2 K \rceil)$ . Thus, the total number of DFFs for PEs can be expressed by the term  $\alpha_5$ . The term  $\alpha_6$  is the total number of DFFs for coefficient registers. Finally, the term  $\alpha_7$  represents the number of DFFs in the controller.

$$\begin{aligned}
\#DFF_{[84]} &= \underbrace{(m + b + \lceil \log_2 K \rceil)(2K + 1)}_{\alpha_5} + \underbrace{Kb}_{\alpha_6} \\
&+ \underbrace{\lceil \log_2 b \rceil}_{\alpha_7}
\end{aligned} \tag{4.21}$$

The formulations of our proposed folded FIR techniques are shown in Eq. 4.22 and Eq. 4.23. The details are as follows. The term  $\alpha_8$  represents the number of DFFs of the coefficient registers. The term  $\alpha_9$  counts the number of DFFs in accumulators and latches of the serial-in folded FIR. The term  $\alpha_{10}$  is the number of DFFs for the

out-loop delays,  $fD$ , as shown in Fig. 4.24. Because there is a counter to generate the signal “RST”, the term  $\alpha_{11}$  expresses the number of DFFs of the counter. For the parallel-in folded FIR, the term  $\alpha_{12}$  gives the number of DFFs in the register file,  $\alpha_{13}$  represents the number of DFFs of the coefficient registers, and  $\alpha_{14}$  expresses the number of DFFs for the MOD- $f$  counter.

$$\begin{aligned} \#DF_{serial-in} = & \underbrace{bfr}_{\alpha_8} + \underbrace{2r(m+b+\lceil\log_2 f\rceil)}_{\alpha_9} \\ & + \underbrace{f(r-1)(m+b+\lceil\log_2 K\rceil)}_{\alpha_{10}} + \underbrace{\lceil\log_2 f\rceil}_{\alpha_{11}} \end{aligned} \quad (4.22)$$

$$\begin{aligned} \#DF_{parallel-in} = & \underbrace{K(m+b+\lceil\log_2 K\rceil)}_{\alpha_{12}} + \underbrace{Kb}_{\alpha_{13}} \\ & + \underbrace{\lceil\log_2 f\rceil}_{\alpha_{14}} \end{aligned} \quad (4.23)$$

We estimated the power consumption of memory by counting the access number of registers of each computation iteration [80]. The following equations list the estimation results for five candidates.

$$\begin{aligned} \#reg\_access_{[87]} = & f(r(m+b+\lceil\log_2 K\rceil) \\ & (f+1) + (K-f)m) \end{aligned} \quad (4.24)$$

$$\begin{aligned} \#reg\_access_{[92]} = & F(f(r(m+b+\lceil\log_2 K\rceil) \\ & (f+1) + (K-f)m)) \end{aligned} \quad (4.25)$$

$$\#reg\_access_{[84]} = m((m + b + \lceil \log_2 K \rceil)(2K + 1) + K) \quad (4.26)$$

$$\begin{aligned} \#reg\_access_{serial-in} &= f(m + (m + b + \lceil \log_2 f \rceil) \\ &\quad (r(f + 1) - f)) \end{aligned} \quad (4.27)$$

$$\#reg\_access_{parallel-in} = 2rf(m + b + \lceil \log_2 K \rceil) + bfr \quad (4.28)$$

Additionally, we formulated the occupancy of multiplexers for five folded architectures by the number of the 1-bit 2-to-1 multiplexer as follows:

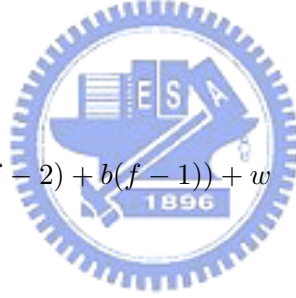
$$\#MUX_{[84]} = 2Kw + Kb \quad (4.29)$$

$$\#MUX_{[87]} = r(w(f - 2) + b(f - 1)) + w \quad (4.30)$$

$$\#MUX_{[92]} = F(r(w(f - 2) + b(f - 1)) + w) \quad (4.31)$$

$$\#MUX_{serial-in} = rw \quad (4.32)$$

$$\#MUX_{parallel-in} = 2rwf \quad (4.33)$$



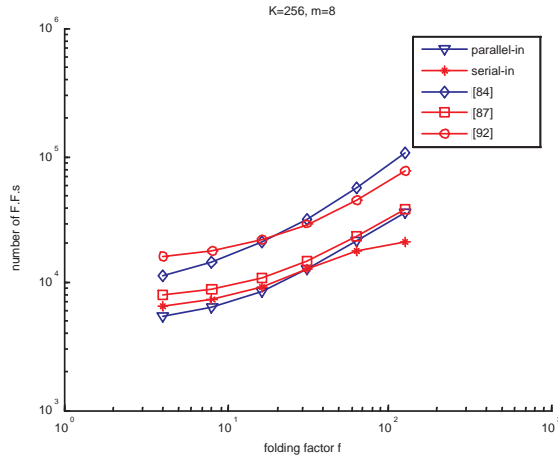


Figure 4.29: Number of DFFs of folded architectures (in log scale)

To graphically compare the candidates, we sketched the results for  $K=255$  and  $m=8$  (in log scale), as shown in Fig. 4.29, Fig. 4.30, and Fig. 4.31. As shown in Fig. 4.29, the serial-in folded FIR has the lowest memory requirement for large  $f$  while the parallel-in folded FIR has the edge for small  $f$ . With regard to the power consumption, the folded FIR of [87] and the serial-in folded FIR exponentially grows with the increasing value of folding factor.

According to Fig. 4.30, the parallel-in folded FIR consumes the least power than others. Taking the IS-95 WCDMA pulse shaping FIR filter, whose specification is tabulated in Table 4.1, as an example, we have implemented five architectures and estimated area requirements and power consumption by the Synopsys Design Analyzer and PrimePower. VLSI design exists a trade-off between operational speed and silicon area occupation. The main argument in this chapter is to save silicon area and power consumption based on the same speed. Basically, each architecture's critical-path delay is the latency between one MA and one multiplexer. We specified the clock constraint of each architecture in the synthesizing stage to let the comparisons make sense. The target report of each architecture is summarized



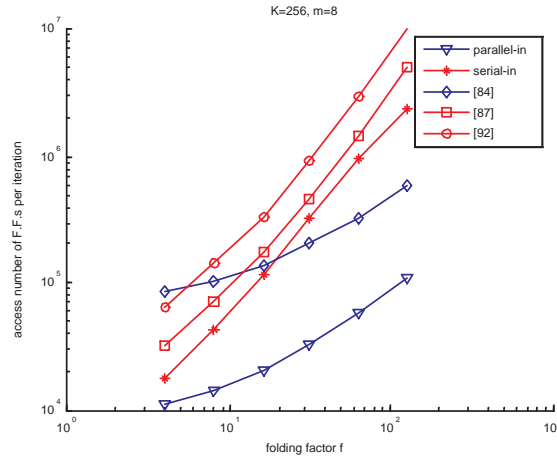


Figure 4.30: Access number of DFFs per iteration (in log scale)

in Table 4.2. As we can see that the folding technique can save the area requirement but maximal resource sharing can lead to an increase in power consumption. However, our proposed folded architectures consume less power among all folded ones. The parallel-in folded FIR filter with the folding factor  $f = 11$  was fabricated using a  $0.18\mu\text{m}$  CMOS technology, packaged in 68-pin LCC, and successfully passed functional testing. The features of the implementation and the chip micrograph are given in Table 4.3 and Fig. 4.32, respectively.

#### 4.2.5 Summary

Two novel systematic hardware-efficient folding techniques for high-order FIR filtering have been presented. The parallel-in folded design methodology was applied to the design of an IS-95 WCDMA pulse shaping FIR filter. It features a sample rate of 168.96 MSPS at a power dissipation of 16.66 mW in a  $0.18\mu\text{m}$  CMOS technology. Under the same throughput rate, the proposed techniques enable the validation of the architecture of the folded FIR filter with minimal storage requirement and less power dissipation when comparing with that of the previous

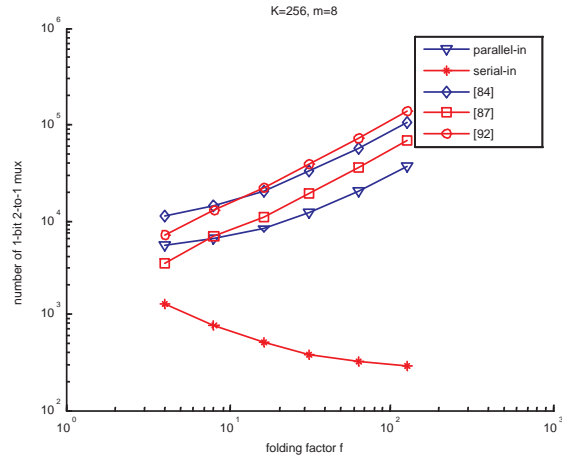


Figure 4.31: Number of 1-bit 2-to-1 multiplexers of folded architectures (in log scale)

filter length	33 tap
throughput	15.36 MSPS
passband edge	$0.1\pi\omega$
stopband edge	$0.28\pi\omega$
passband ripple	1.5 dB
stopband ripple	40 dB
input sample word-length	8 bit
coefficient word-length	16 bit

Table 4.1: IS-95 WCDMA pulse shaping FIR filter specification.

works in the literatures.

FIR Architecture	unfolded	[84]	[87]	[92] ( $F = 2$ )	serial-in	parallel-in
Area ( $\mu m^2$ )	2311048	2231788	865477	1627096	578116	758781
Power ( $mW$ )	6.79	49.9	34.8	70.2	25.45	16.66
Critical-path ( $ns$ )	65.1	4.07	5.92	5.92	5.92	5.92

Table 4.2: Area and power consumption comparisons.(An IS-95 WCDMA pulse shaping 33-tap FIR)

throughput	168.96 MSPS
power dissipation	16.66 mW
chip size	$1.411 \times 1.411 mm^2$
supply voltage(core/ring)	1.8 V/3.3 V

Table 4.3: Features of the IS-95 WCDMA pulse shaping FIR filter chip.

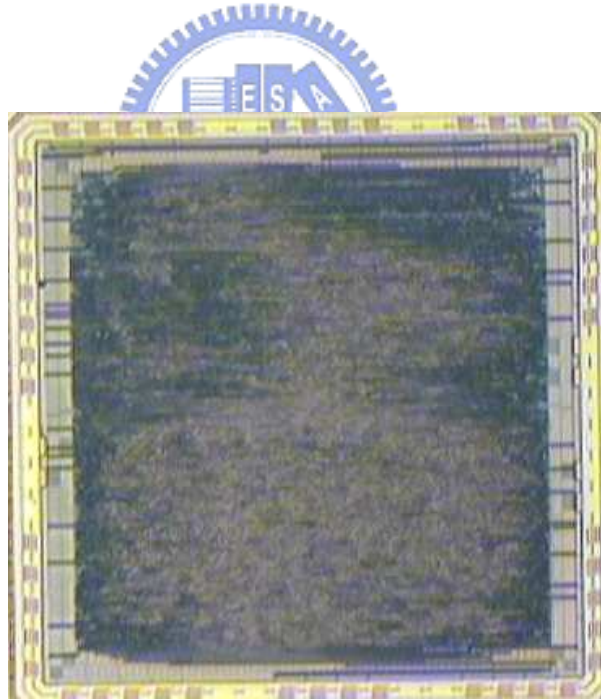


Figure 4.32: Photomicrograph of IS-95 WCDMA pulse shaping FIR filter chip.

## Chapter 5

# Power Efficient Turbo Decoding

### 5.1 Introduction

Turbo codes have been widely used in many telecommunication applications. To efficiently save the notoriously high power dissipation of turbo decoder, literature has presented numbers of early stopping mechanisms. The early stopping mechanisms can be done by early termination or early give-up steps. The early termination ends the search of turbo decoder for solvable packets beforehand, while the early give-up ceases the turbo decoding for unsolvable packets. Many papers have proposed early termination approaches which can be categorized into three classes: soft-bit decision [93, 94], hard-bit decision [95, 96], and extra-checking policy [97, 98]. However, very few papers target on early give-up techniques. The early give-up is particularly important for channels with low signal-to-noise ratio (SNR), in that the early give-up allows the decoder to stop the decoding process for unsolvable packets as early as possible and hence minimizing the number of "redundant" iterations. The reduction of MAP iterations implies to save the power dissipation of turbo decoding. It is worth noting that, without early give-up techniques, the ARQ or HARQ protocols will not enable the resend request mechanism for an unsolvable

packet until the turbo decoding process reaches the maximum number of iterations. The paper [99] by Buckley and Wicker presents an early give-up technique using a neural network to predict turbo decoding errors. Their technique can improve reliability and throughput performance at a lower average decoding complexity than turbo decoding with CRC-based early termination.

The object of this letter is to design a channel-aware turbo decoder. The turbo decoder can reduce redundant iterations in noisy channel and minimize the iterations when the channel becomes better. We present an early give-up technique with a state reuse mechanism. The proposed technique detects turbo decoding errors by monitoring the extrinsic information and reuses the *a-priori* LLR of previous decoding process as the initial condition for the resend packet. First, when the extrinsic information oscillates without significant increases as the number of iterations increases, the decoded packet is most likely an unsolvable packet. So, once the oscillation of extrinsic information is detected, the early give-up will stop the decoding process and register the *a-priori* LLR from wasting further power consumption. The ARQ or HARQ protocols will thereafter trigger the resend request. When the turbo decoder starts decoding the resend packet, the registered *a-priori* LLR will be reused as the initial condition. The reuse of the *a-priori* information is called the state reuse mechanism. When the channel status recovers to better situation with higher SNR in a fading environment, the state reuse mechanism can further reduce the required iterations and the computational load.

## 5.2 Early Give-Up Decision

We use the extrinsic information to detect the syndrome of unsolvable packets and decide when to give error packets up. The unsolvable packets are the received packets which cannot be correctly decoded after the turbo decoder reaches the maximum number of iterations. As shown in Fig. 5.1, for the SISO decoder  $n$ ,

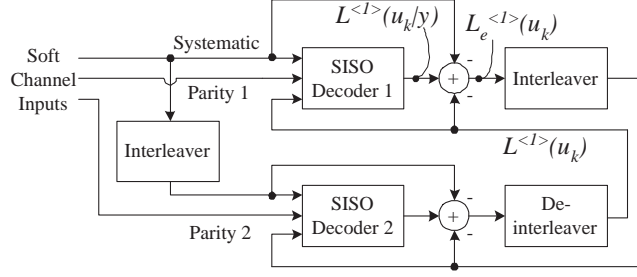


Figure 5.1: Turbo decoder scheme.

the extrinsic information  $L_e^{<n>}(u_k)$  is generated by subtracting the *a-priori* LLR  $L^{<n>}(u_k)$  and the channel value from the *a-posteriori* LLR  $L^{<n>}(u_k|y)$ . In log-MAP algorithm, for each decoded bit  $u_k \in \{-1, +1\}$ , the Eq.(5.1) and Eq.(5.2) expresses the relation of the a-posteriori LLR and the extrinsic information where  $\alpha$  and  $\beta$  represent state metrics and  $\gamma$  stands for branch metrics, and  $y$  is the received symbol sequence [100]:

$$L(u_k|y) = \ln \left( \frac{\sum_{u_k=+1} \alpha_{k-1}(s') \cdot \gamma_k(s', s) \cdot \beta_k(s)}{\sum_{u_k=-1} \alpha_{k-1}(s') \cdot \gamma_k(s', s) \cdot \beta_k(s)} \right) \quad (5.1)$$

$$= L(u_k) + L_c y_{ks} + L_e(u_k)$$

where

$$L_e(u_k) = \ln \left( \frac{\sum_{u_k=+1} \alpha_{k-1}(s') \cdot \chi_k(s', s) \cdot \beta_k(s)}{\sum_{u_k=-1} \alpha_{k-1}(s') \cdot \chi_k(s', s) \cdot \beta_k(s)} \right) \quad (5.2)$$

Fig. 5.2 illustrates the simulation results with solvable and unsolvable packets. Obviously, when the decoder is iteratively decoding an unsolvable packet, the mean of the absolute values of extrinsic information of decoded bits, a.k.a. the mean

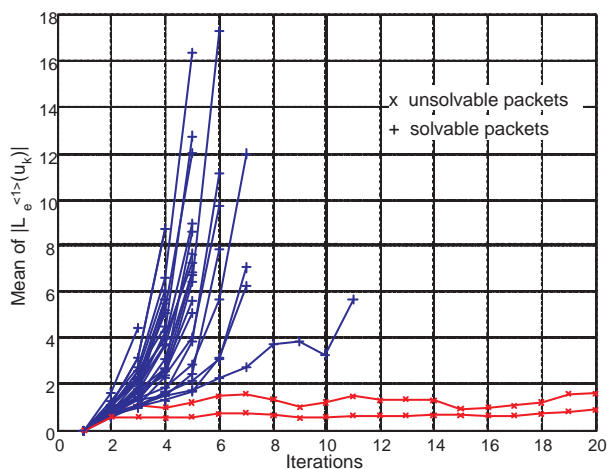


Figure 5.2: The trends of the mean of the absolute extrinsic information for solvable and unsolvable packets.

of  $|L_e^{<1>}(u_k)|$ , oscillates within a bounded range. The proposed early give-up uses the oscillation as the syndrome of an unsolvable packet.

### 5.3 State Reuse Mechanism

Traditionally, the initial *a-priori* LLR for log-MAP decoding algorithm is set to be zero, which is based on the assumption of each decoded bit having equal probability to be decoded as -1 or +1. However, when the same packet is being resent, the last *a-priori* LLR of previous decoding process could be a better guess for initial *a-priori* LLR than zero in that the resent packet should have correlation with the preceding one. If we reuse the last *a-priori* LLR of previous decoding process, it is possible to reduce the number of iterations in the following decoding, and hence save the power consumption. In the proposed approach, the state reuse mechanism is used to further save the power dissipation of the decoding process for resent packets. When the proposed early give-up procedure detects the syndrome of

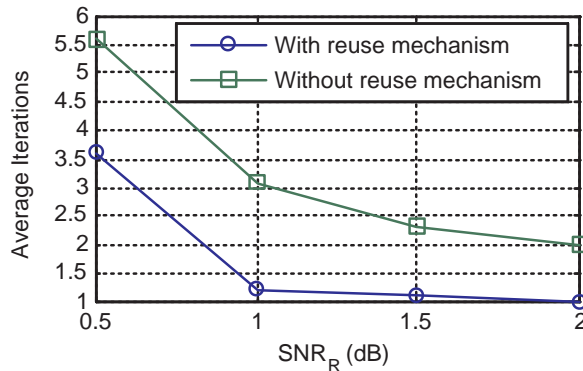


Figure 5.3: The average of the required iterations for turbo decoding process with and without state reuse mechanism.

unsolvable packets, the turbo decoder will register the last *a-priori* LLR value and use it as the initial *a-priori* LLR in the decoding process of the resend packet. Fig.5.3 illustrates the iteration reductions of the state reuse mechanism, where SNRR is the SNR when the packet is being retransmitted. Initially, the given-up packet is transmitted at 0.5 dB SNR. With the reuse mechanism, when the channel status is unchanged, the decoding process of the resent packet requires 3.6 iterations in average, while the one without the reuse mechanism requires 5.6 iterations in average. The efficiency of applying reuse mechanism is even better when the packet is being resent at higher SNR.

## 5.4 Proposed Turbo Decoding Flow

The proposed early give-up is feasible to be embedded into the conventional turbo decoding process. Fig.5.4 shows the flowchart of turbo decoding with the proposed early give-up technique. The technique starts with the initialization of the *a-priori* LLR (step 4-0). If the received packet is a resent packet, the last *a-priori* LLR will be used as the initial condition in the SISO decoding; otherwise, the



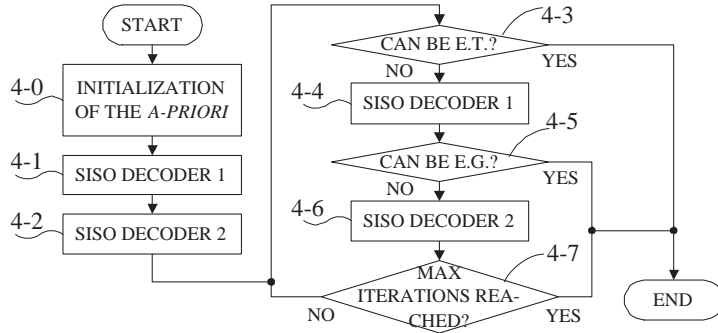


Figure 5.4: The proposed turbo decoding flowchart.

initial *a-priori* LLR will be set to zero. Then, the decoding process performs the SISO decoding of the first constituent code (step 4-1), followed by SISO decoding of the second constituent code (step 4-2). Next, if the decoding result of the second SISO decoding (step 4-2) agrees with the early termination condition (YES path of step 4-3), then the valid decoded packet outputs; otherwise (NO path step 4-3), the iterative decoding continues to the SISO DECODER 1 (step 4-4) followed by the decision of early give-up (step 4-5). If the mean of  $|L_e^{<1>}(u_k)|$  oscillates within a bounded range (YES path of step 4-5), the iterative decoding will halt further activity; otherwise (NO path of step 4-5), the iterative decoding continues to the SISO DECODER 2 (step 4-6). Upon finishing the step 4-6, the decoding process will repeat steps from (step 4-3) to (step 4-7) until one of YES paths of (step 4-3), (step 4-5), or (step 4-7) is enabled.

## 5.5 Simulation Results

Fig. 5.5 shows the simulation results with the proposed early give-up technique in which the average number of iterations required by the proposed decoding process is less than by the decoding with the Magic Genie Rule. Table 5.1 concludes the simulation results for various channel SNRs. The proposed early give-up, when

the channel SNR is low, significantly reduces the required iterations and power dissipation. When the channel SNR is 0.1 dB, the proposed approach can reduce the number of decoding iterations by 51.67%. To understand the hardware overhead of the early give-up implementation, the proposed turbo decoding process has been realized by Application-Specific Integrated Circuit (ASIC) design flow. According to the synthesis report with TSMC  $0.18\mu\text{m}$  cell library, the area overhead of the early give-up is 0.9%. Running at 50MHz, as per the report of PrimePower (by Synopsys Corp.), the decision circuit of the early give-up only consumes 0.262 mW, while the log-MAP decoding unit consumes 46.96 mW per iteration. The power overhead is extremely lower than the power saving of iteration reduction.



Table 5.1: Iteration reduction rate of the proposed turbo decoding (comparing with conventional turbo decoding with the Magic Genie Rule).

Channel SNR	0dB	0.1dB	0.2dB	0.3dB	0.4dB	0.5dB
Reduction Rate	60.66%	51.67%	36.89%	29.41%	12.47%	7.69%

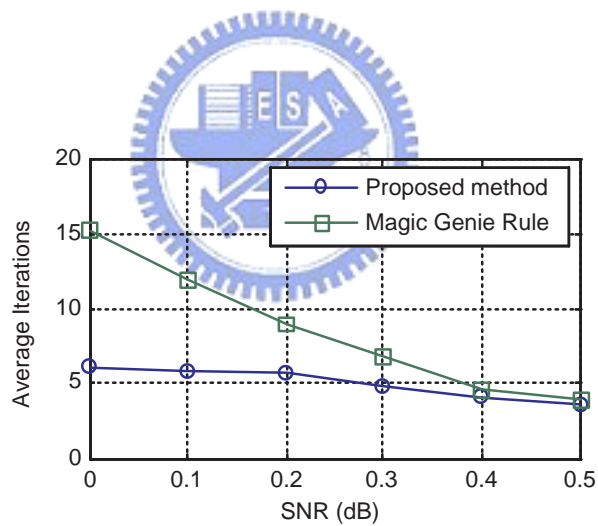


Figure 5.5: Simulation results of the proposed turbo decoding.

## Chapter 6

# Conclusion

Power consumption is an increasingly pressing problem in modern system design, especially for embedded systems and portable devices which are powered by battery. Most previous researches focused on developing power minimization techniques at the lower (transistor and logic) levels of the design hierarchy. Operation scheduling determines the cost-speed tradeoffs of the design. If the design is subject to a speed constraint, the scheduling algorithm will attempt to parallelize the operations to meet the timing constraint. Conversely, if there is a limited on the cost (area or energy), the scheduler will serialize operations to meet the resource constraint.

This dissertation focuses on the limited-resource problem which arises while the number of atomic (function) units is constrained or the power resource is limited. Considerations and techniques on how to meet the constraint requirement have been provided in three different chapters, and have been involved in the perspective on how to utilize the resource efficiently. The contributions of each resource efficient technique go as follows.

## 1. Resource-Constrained Low-Power Scheduling in HLS

- **Maximizing the time frame of each operation**

Algorithmic transformations benefit task mobilities. The algebraic transformation has been used to shrink the loops and hence increase the iteration period bound (IPB). Retiming and unfolding techniques have been employed to reduce the iteration period (IP) as much as possible and thus maximizing the time frame of each task.

- **Maximizing the power saving with consideration on peak power**

In low-power designs for battery-driven portable applications, the peak power drives the transient characteristic of the CMOS circuit. Following the optimization of average power dissipation, we suppress the peak power dissipation by the barrier-driven approach. The barrier-driven approach gradually compresses the task schedulability until no further legal scheduling can be found.

- **Providing flexibility in usage**

The inputs to the proposed multiple-voltage low-power scheduling algorithm are an FSFG, a component library, a resource constraint, and a latency constraint. The algorithm has been implemented in  $C^{++}$  and could be applied for other compositions of supply voltages.

## 2. Limited-Resource Folding Techniques

- **Providing Efficient tradeoffs between cost and speed in folded architectures**

The folding techniques for resource sharing efficiently to meet both throughput and resource constraints have been proposed. The datapath cost could be reduced when the throughput rate is lower than the maximum speed at which a full-length datapath can operate. The proposed folding techniques intend to efficiently trade excessive speed for datapath cost with high degree of flexibility.

- **Outperforming published folded architectures**

The published folded architectures mainly focus on the utilization of processing elements or datapath. The efficient cost of registers and control units have been emphasized by elaborated arrangement. According to the simulation results, under the same throughput rate, the proposed folding techniques outperform the previous works in the literatures with minimal storage requirement and less power dissipation.

### 3. Power Efficient Turbo Decoding

- **Early Give-Up Decision**

The novel early give-up technique of turbo decoding is particularly important for channels with low signal-to-noise ratio (SNR), in that the early give-up allows the decoder to stop the decoding process for unsolvable packets as early as possible and hence minimizing the number of redundant iterations and the power dissipation.

- **State Reuse Mechanism**

The reuse of the *a-priori* information benefits the power saving as well. When the channel status recovers to better situation with higher SNR in

a fading environment, the state reuse mechanism can further reduce the required iterations and the computational load.

Overall, This dissertation illuminates the impact of resource constraints on the design methodologies of VLSI signal processing and communication applications, proposes several design methodologies in the resource-constrained low-power high-level synthesis (HLS), the limited-resource folding techniques, and the power efficient turbo decoder, and tries to stimulate interests in the VLSI signal processing in reformulating and revisiting classic VLSI signal processing problems under new constraints and exploring the role of signal processing in exciting new applications.



# Bibliography

- [1] U.S. Environmental Protection Agency (EPA). [Online]. Available: <http://epa.gov/>
- [2] Energy Star program. [Online]. Available: <http://www.energystar.gov/>
- [3] D. D. Gajski and L. Ramachandran, “Introduction to high-level synthesis,” *IEEE Design & Test of Computers*, vol. 11, no. 4, pp. 44–54, 1994.
- [4] P. G. Paulin and J. P. Knight, “Force-directed scheduling for the behavioral synthesis of ASIC’s,” *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 661–679, June, 1989.
- [5] C. Tseng and D. P. Siewiorek, “Automated synthesis of datapaths in digital systems,” *IEEE Trans. Computer-Aided Design*, vol. 5, pp. 379–395, July, 1986.
- [6] S. Y. Kung, H. J. Whitehouse, and T. Kailath, *VLSI and Modern Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1985.
- [7] S. Davidson, D. Landskov, B. D. Shriver, and P. W. Mallett, “Some experiments in local microcode compaction for horizontal machines,” *IEEE Trans. Comput.*, vol. C-30, no. 7, pp. 460–477, July, 1981.
- [8] K. Keutzer and P. Vanbekbergen, “The impact of CAD on the design of low



- power digital circuits,” *Proc. IEEE Symp. Low Power Electronics*, 1994, pp. 42–45.
- [9] Y. Katsumata et al., “CMOS/BiCMOS technology,” *The VLSI Handbook*, W.-K. Chen (ed.), CRC Press, Boca Raton, 2000, pp. 2/1–2/28
- [10] S. Cristoloveanu, “Silicon on insulator technology,” *The VLSI Handbook*, W.-K. Chen (ed.), CRC Press, Boca Raton, 2000, pp. 4/1–4/15
- [11] L. Benini and G. De Micheli, “System-level power optimization: techniques and tools,” *Proc. Int. Symp. Low Power Electronics and Design*, 1999, pp. 288–293
- [12] S. Gary, “Low-power microprocessor design,” *Low Power Design Methodologies*, J. Rabaey and M. Pedram (eds.), *Kluwer Academic Publishers*, Boston, 1996, pp. 255–288
- [13] K. Itoh, “Low power memory design,” *Low Power Design Methodologies*, J. Rabaey and M. Pedram (eds.), *Kluwer Academic Publishers*, Boston, 1996, pp. 201–251
- [14] L. Benini and G. DeMicheli, “Dynamic Power Management,” *Kluwer Academic Publishers*, Boston, 1998.
- [15] L. Benini, A. Bogliolo, and G. De Micheli, “A survey of design techniques for system-level dynamic power management,” *IEEE Trans. VLSI Systems*, vol. 8, no. 3, pp. 299–316, 2000.
- [16] (Oct. 2006). *Advanced Configuration and Power Interface Specification, Revision 3.0b*. [Online]. Available: <http://www.acpi.info/spec.htm>
- [17] T. Sakurai, H. Kawaguchi, and T. Kuroda, “Low-power CMOS design through VTH control and low-swing circuits,” *Proc. Int. Symp. Low Power Electronics and Design*, 1997, pp. 1–6

- [18] H. Igura et al., “An 800-MOPS, 110-mW, 1.5-V, parallel DSP for mobile multimedia processing,” *IEEE J. Solid-State Circuits*, vol. 33, no. 11, 1998, pp. 1820–1828.
- [19] H. Kubosawa et al., “A 1.2-W, 2.16-GOPS/720-MFLOPS embedded superscalar microprocessor for multimedia applications,” *IEEE J. Solid-State Circuits*, vol. 33, no. 11, 1998, pp. 1640–1647
- [20] W. Lee et al., “A 1-V programmable DSP for wireless communications,” *IEEE J. Solid-State Circuits*, vol. 32, no. 11, 1997, pp. 1766–1776
- [21] C. Piguet et al., “Low-power design of 8-b embedded CoolRisc microcontroller cores,” *IEEE J. Solid-State Circuits*, vol. 32, no. 7, 1997, pp. 1067–1077
- [22] G. Yeap, “Practical Low Power Digital VLSI Design,” *Kluwer Academic Publishers*, Boston, 1998
- [23] A. Chandrakasan and R. Brodersen, “Low Power Digital CMOS Design,” *Kluwer Academic Publishers*, Boston, 1995
- [24] M. Stan and W. Burleson, “Low-power encodings for global communication in CMOS VLSI,” *IEEE Trans. VLSI Systems*, vol. 5, no. 4, 1997, pp. 444–455
- [25] M. Stan and W. Burleson, “Bus-invert coding for low-power I/O,” *IEEE Trans. VLSI Systems*, vol. 3, no. 1, 1995, pp. 49–58
- [26] C. Kretzschmar, R. Siegmund, and D. Mueller, “A low overhead auto-optimizing bus encoding scheme for low power data transmission,” *Proc. Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2002, pp. 342–352
- [27] N. Weste and K. Eshraghian, “Principles of CMOS VLSI Design, 2nd ed.,” *Kluwer Academic Publishers*, Boston, 1994

- [28] T. Callaway and E. Swartzlander, "The power consumption of CMOS adders and multipliers," *Low-Power CMOS Design*, A. Chandrakasan and R. Brodersen (eds.), IEEE Press, Piscataway, 1998, pp. 218–224
- [29] T. Callaway, "Modeling the power consumption of CMOS arithmetic elements," *Application Specific Processors*, E. Swartzlander (ed.), *Kluwer Academic Publishers*, Boston, 1997, pp. 29–61
- [30] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proc. of the IEEE*, vol. 78, no. 2, 1990, pp. 264V299
- [31] S. Iman and M. Pedram, *Logic synthesis for low power VLSI design*, Kluwer Academic Publishers, Boston, 1998
- [32] G. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*, Kluwer Academic Publishers, Boston, 1996
- [33] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, 1994
- [34] J. Monteiro and S. Devadas, "Techniques for power estimation and optimization at the logic level: a survey," *J. VLSI Signal Processing Systems*, vol. 13, no. 2/3, 1996, pp. 259–276
- [35] M. Borah, R. M. Owens, and M. J. Irwin, "Transistor sizing for minimizing power consumption of CMOS circuits under delay constraints," *Proc. Int. Symp. Low Power Design*, 1995, pp. 167–172
- [36] B. Chen and I. Nedelchev, "Power Compiler: a gate-level power optimization and synthesis system," *Proc. IEEE Int. Conf. Computer Design*, 1997, pp. 74–79

- [37] L. Benini, G. De Micheli, and E. Macii, "Designing low-power circuits: practical recipes," *IEEE Circuits and Systems Magazine*, vol. 1, no. 1, 2001, pp. 6–25
- [38] J. Cong, L. He, and C.-K. Koh, "Layout optimization," *Low Power Design in Deep Submicron Electronics*, W. Nebel and J. Mermet (eds.), Kluwer Academic Publishers, Dordrecht, 1997, pp. 205–265
- [39] R. Zimmermann and W. Fichtner, "Low-power logic styles: CMOS versus passtransistor logic," *IEEE J. Solid-State Circuits*, vol. 32, no. 7, 1997, pp. 1079–1090
- [40] M. Kontiala, M. Kuulusa, and J. Nurmi, "Comparison of static logic styles for lowvoltage digital design," *Proc. IEEE Int. Conf. Electronics, Circuits and Systems*, 2001, pp. 1421–1424
- [41] C. Svensson and D. Liu, "Low power circuit techniques," *Low Power Design Methodologies*, J. Rabaey and M. Pedram (eds.), Kluwer Academic Publishers, Boston, 1996, pp. 37–63
- [42] A. Alvandpour and C. Svensson, "Improving cell libraries for low power design," *Proc. Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 1996, pp. 317–325
- [43] C. Svensson and J. Yuan, "Latches and flip-flops for low-power systems," *Low-Power CMOS Design*, A. Chandrakasan and R. Brodersen (eds.), IEEE Press, Piscataway, 1996, pp. 233–238
- [44] N. Dragone et al., "An innovative methodology for the design automation of low power libraries," *Proc. Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 1998, pp. 31–40
- [45] O.S. Unsal and I. Koren, "System-level power-aware design techniques in real-time systems," *Proceedings of the IEEE*, vol.91, pp.1055–1069, July 2003.

- [46] A.P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R.W. Brodersen, "Optimizing power using transformations," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol.14, pp.12–31, Jan. 1995.
- [47] A.P. Chandrakasan, S. Sheng, and R.W. Brodersen, "Low power cmos digital design," *IEEE J. Solid-State Circuits*, vol.27, pp.473–484, April 1992.
- [48] A. Raghunathan and N.K. Jha, "Behavioral synthesis for low power," *IEEE Int. Conf. Computer Design: VLSI in Computer and Processors*, pp.318–322, Oct. 1994.
- [49] A. Raghunathan and N.K. Jha, "An iterative improvement algorithm for low power data path synthesis," *IEEE/ACM Int. Conf. Computer-Aided Design*, pp.597–602, Nov. 1995.
- [50] S. Raje and M. Sarrafzadeh, "Scheduling with two voltages under resource constraints," tech. rep., Dept. Elect. Eng. Comput. Sci., Northwestern Univ. Evanston, IL, 1995.
- [51] M. Takahashi, M. Hamada, T. Nishikawa, H. Arakida, T. Fujita, F. Hatori, S. Mita, K. Suzuki, A. Chiba, T. Terazawa, T. Kuroda, and T. Furuyama, "A 60-mw mpeg4 video codec using clustered voltage scaling with variable supply-voltage scheme," *IEEE J. Solid-State Circuits*, vol.33, pp.1772–1780, Nov. 1998.
- [52] M. Sarrafzadeh and S. Raje, "Scheduling with multiple voltages under resource constraints," *IEEE Int. Sym. on Circuits and Systems*, pp.350–353, May 1999.
- [53] S. Raje and M. Sarrafzadeh, "Variable voltage scheduling," *Proceedings of the International Symposium on Low Power Design*, pp.9–14, 1995.
- [54] J.M. Change and M. Pedram, "Energy minimization using multiple supply voltages," *IEEE Trans. VLSI Syst.*, vol.5, pp.436–443, Dec. 1997.

- [55] W.T. Shiue and C. Chakrabarti, "Low power scheduling with resources operating at multiple voltages," *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol.47, pp.536–543, June 2000.
- [56] Hsueh-Chih Yang and Lan-Rong Dung, "On multiple-voltage high-level synthesis using algorithmic transformations," in *Proceedings of the IEEE ASP-DAC*, pp.872–876, Jan. 2005.
- [57] M.C. Johnson and K. Roy, "Datapath scheduling with multiple supply voltages and level converters," *ACM Trans. Design Automation Electronic Syst.*, pp.227–248, July 1997.
- [58] Y.R. Lin, C.T. Hwang, and A.C.H. Wu, "Scheduling techniques for variable voltage low power design," *ACM Trans. Design Automation Electronic Syst.*, pp.81–97, April 1997.
- [59] A. Manzak and C. Chakrabarti, "A low power scheduling scheme with resources operating at multiple voltages," *IEEE Trans. VLSI Syst.*, vol.10, pp.6–14, Feb. 2002.
- [60] V.K. Madisetti and B.A. Curtis, "A quantitative methodology for rapid prototyping and high-level synthesis of signal processing algorithms," *IEEE Trans. Signal Processing*, vol.42, no.11, pp.3188–3208, November 1994.
- [61] V.K. Madisetti, *VLSI Digital Signal Processors: An Introduction to Rapid Prototyping and Design Synthesis*, IEEE Press, 1996.
- [62] T. Barnwell and C. Hodges, "Optimal implementation of signal flow graphs on synchronous multiprocessors," *IEEE Int. Conf. Parallel Processing*, pp.90–95, August 1982.
- [63] K. Parhi and D. Messerschmitt, "Static rate-optimal scheduling of iterative

- data-flow programs via optimum unfolding,” *IEEE Trans. Computers*, vol.40, pp.178–195, Feb. 1991.
- [64] C.T. Hwang, J.H. Lee, and Y.C. Hsu, “A formal approach to the scheduling problem in high level synthesis,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol.10, pp.464–475, April 1991.
- [65] S.P. Mohanty, N. Ranganathan, and V. Krishna, “Datapath scheduling using dynamic frequency clocking,” *IEEE Computer Society Sym. on VLSI*, pp.58–63, April 2002.
- [66] T. Lin and C.W. Jen, “An efficient 2-d DWT architecture via resource cycling,” *IEEE International Symposium on Circuits and Systems*, vol.4, pp.914–7, May 2001.
- [67] W.S. Peng, “An efficient algorithm and architecture design for two-dimension separable discrete wavelet transform,” *ICIP 99*, vol.2, 1999.
- [68] R.M. Owens and M. Vishwanath, “A very efficient storage structure for DWT and IDWT filters,” *Journal of VLSI Signal Processing*, vol.19, pp.215–25, 1998.
- [69] T.C. Denk and K.K. Parhi, “Systolic VLSI architectures for 1-d discrete wavelet transforms,” *Conference Record of Thirty-Second Asilomar Conference on Signals, Systems and Computers*, vol.2, pp.1220–4, 1998.
- [70] M. S. and M. J.V., “Wavelet packet transforms for system-on-chip applications,” *ICASSP 2000*, vol.6, pp.3287–3290, 2000.
- [71] K. Parhi and T. Nishitani, “VLSI architectures for discrete wavelet transforms,” *IEEE Transactions on VLSI Systems*, vol.1, no.2, pp.191–202, June 1993.
- [72] M. Vishwanath, R. Owens, and M. Irwin, “VLSI architectures for the discrete

- wavelet transform,” *IEEE Trans. Circuit and Systems-II*, vol.42, no.5, pp.305–316, May 1995.
- [73] C. Chakrabarti and C. Mumford, “Efficient realizations of analysis and synthesis filters based on the 2-d discrete wavelet transform,” *ICASSP 96*, pp.3256–3259, 1996.
- [74] C. Chakrabarti and M. Vishwanath, “Efficient realizations of the discrete and continuous wavelet transformsGFrom single chip implementations to SIMD parallel computers,” *IEEE Trans. Signal Processing*, vol.43, pp.759–771, March 1995.
- [75] C. Yu and S.J. Chen, “Efficient VLSI architectures for separable 2-d discrete wavelet transforms,” *IEEE International Symposium on Consumer Electronics*, Oct. 1998.
- [76] S. Mallat., “A theory for multiresolution signal decomposition: The wavelet representation,” *IEEE Trans. Pattern Analysis and Machine Intell.*, vol.11, no.7, pp.674–693, July 1989.
- [77] G. Knowles, “VLSI architectures for the discrete wavelet transform,” *Electronics Letters*, vol.26, no.15, July 1990.
- [78] C. S. Burrus, “Digital filter structures described by distributed arithmetic,” *IEEE Trans. on Circuits Syst.*, pp. 674–680, Dec. 1977.
- [79] T. C. Denk and K. K. Parhi, “Synthesis of folded pipelined architectures for multirate DSP algorithms,” *IEEE Trans. VLSI*, vol. 6, no. 4, pp. 595–607, Dec. 1998.
- [80] Rashindra Manniesing, R. Kleihorst, A. V. D. Avoird, and Emile Hendriks “Power analysis of a general convolution algorithm mapped on a linear processor array,” *Journal of VLSI Signal Processing*, vol. 37, pp. 5–19, May 2004.



- [81] S.-F. Lin, S.-C. Huang, F.-S. Yang, C.-W. Ku, and L.-G. Chen, "Power-efficient FIR filter architecture design for wireless embedded system," *IEEE Trans. Circuits Syst. II*, vol. 51, no. 1, pp. 21–25, Jan. 2004.
- [82] E. Lueder, "Generation of equivalent block parallel digital filters and algorithms by a linear transformation," in *IEEE Int. Symp. on Circuits and Systems*, , pp. 495–498, May 1993.
- [83] M. Mehendale and S. D. Sherlekar, *VLSI SYNTHESIS OF DSP KERNELS-Algorithmic and Architectural Transformations*. KLUWER ACADEMIC PUBLISHERS, 2001.
- [84] I. Milentijevic, V. Ciric, T. Tokic, and O. Vojinovic, "Folded bit-plane FIR filter architecture with changable folding factor," in *IEEE Euromicro Sym. on Digital System Design*, pp. 45–52, Sept. 2002.
- [85] Z. J. Mou and P. Duhamel, "Short-length FIR filters and their use in fast nonrecursive filtering," *IEEE Trans. Signal Processing*, vol. 39, no. 6, pp. 1322–1332, June 1991.
- [86] A. V. Oppenheim and R. W. Schaffer, *Discrete-time signal processing*, 2nd ed. PRENTICE HALL, 1999.
- [87] K. K. Parhi, C.-Y. Wang, and A. P. Brown, "Synthesis of control circuits in folded pipelined DSP architecture," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 1, pp. 29–43, Jan. 1992.
- [88] D. A. Parker and K. K. Parhi, "Low-area/power parallel FIR digital filter implementations," *J. VLSI Signal Processing Syst.*, vol. 17, no. 1, pp. 75–92, 1997.
- [89] D. N. Pearson and K. K. Parhi, "Low-power FIR digital filter architectures," in *IEEE Int. Symp. on Circuits and Systems*, pp. 231–234, Apr. 1995.

- [90] S. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSP Magazine*, pp. 4–19, July 1989.
- [91] P. Bougas, P. Kalivas, A. Tsirikos, and K. Z. Pekmestzi, "Pipelined Array-Based FIR Filter Folding," *IEEE Trans. Circuits and Systems-I*, vol. 52, no. 1, pp. 108–118, Jan. 2005.
- [92] Vijay Sundararajan and K. K. Parhi, "Synthesis of low power folded programmable coefficient FIR digital filters," in *Proc. ASPDAC*, pp. 153–156, Jan. 2000.
- [93] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.
- [94] F. Zhai and I.J. Fair, "Techniques for early stopping and error detection in turbo decoding," *IEEE Trans. Commun.*, vol. 51, no. 10, pp. 1617–1623, Oct. 2003.
- [95] R.Y. Shao, S. Lin, and M.P.C. Fossorier, "Two simple stopping criteria for turbo decoding," *IEEE Trans. Commun.*, vol. 47, no. 8, pp. 1117–1120, Aug. 1999.
- [96] Y. Wu, B.D. Woerner, and J. Ebel, "A simple stopping criteria for turbo decoding," *IEEE Commun. Lett.*, vol. 4, no. 8, pp. 258–260, Aug. 2000.
- [97] A. Shibusatani, H. Suda, and F. Adachi, "A simple stopping criteria for turbo decoding," *IEE Electronics Lett.*, vol. 35, no. 9, pp. 701–702, Apr. 1999.
- [98] J. Heo, K. Chung, and K.M. Chugg, "Simple stopping criterion for min-sum iterative decoding algorithm," *IEE Electronics Lett.*, vol. 37, no. 25, pp. 1530–1531, Dec. 2001.

- [99] M.E. Buckley and S.B. Wicker, “The design and performance of a neural network for predicting turbo decoding error with application to hybrid ARQ protocols,” *IEEE Trans. Commun.*, vol. 48, no. 4, pp. 566–576, Apr. 2000.
- [100] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes (1),” in *Proc. ICC’93*, vol. 2, pp. 1064–1070, May 1993.

