Synthesis and Optimization of DSP Algorithms

George A. Constantinides,
Peter Y.K. Cheung and
Wayne Luk

Kluwer Academic Publishers

SYNTHESIS AND OPTIMIZATION OF DSP ALGORITHMS

This page intentionally left blank

Synthesis and Optimization of DSP Algorithms

by

George A. Constantinides
Imperial College, London

Peter Y.K. Cheung
Imperial College, London

and

Wayne Luk
Imperial College, London

KLUWER ACADEMIC PUBLISHERS
NEW YORK, BOSTON, DORDRECHT, LONDON, MOSCOW

eBook ISBN: 1-4020-7931-1
Print ISBN: 1-4020-7930-3

Created in the United States of America

Visit Kluwer Online at: http://kluweronline.com
and Kluwer's eBookstore at: http://ebooks.kluweronline.com

To all progressive people

This page intentionally left blank

Preface

Digital signal processing (DSP) has undergone an immense expansion since the foundations of the subject were laid in the 1970s. New application areas have arisen, and DSP technology is now essential to a bewildering array of fields such as computer vision, instrumentation and control, data compression, speech recognition and synthesis, digital audio and cameras, mobile telephony, echo cancellation, and even active suspension in the automotive industry.

In parallel to, and intimately linked with, the growth in application areas has been the growth in raw computational power available to implement DSP algorithms. Moore's law continues to hold in the semiconductor industry, resulting every 18 months in a doubling of the number of computations we can perform.

Despite the rapidly increasing performance of microprocessors, the computational demands of many DSP algorithms continue to outstrip the available computational power. As a result, many custom hardware implementations of DSP algorithms are produced - a time consuming and complex process, which the techniques described in this book aim, at least partially, to automate.

This book provides an overview of recent research on hardware synthesis an optimization of custom hardware implementations of digital signal processors. It focuses on techniques for automating the production of area-efficient designs from a high-level description, while satisfying user-specified constraints. Such techniques are shown to be applicable to both linear and nonlinear systems: from finite impulse response (FIR) and infinite impulse response (IIR) filters to designs for discrete cosine transform (DCT), polyphase filter banks, and adaptive least mean square (LMS) filters.

This book is designed for those working near the interface of DSP algorithm design and DSP implementation. It is our contention that this interface is a very exciting place to be, and we hope this book may help to draw the reader nearer to it.

London,
February 2004

George A. Constantinides
Peter Y.K. Cheung
Wayne Luk

This page intentionally left blank

Contents

This page intentionally left blank

1

Introduction

1.1 Objectives

This book addresses the problem of hardware synthesis from an initial, infinite precision, specification of a digital signal processing (DSP) algorithm. DSP algorithm development is often initially performed without regard to finite precision effects, whereas in digital systems values must be represented to a finite precision [Mit98]. Finite precision representations can lead to undesirable effects such as overflow errors and quantization errors (due to roundoff or truncation). This book describes methods to automate the translation from an infinite precision specification, together with bounds on acceptable errors, to a structural description which may be directly implemented in hardware. By automating this step, raise the level of abstraction at which a DSP algorithm can be specified for hardware synthesis.

We shall argue that, often, the most efficient hardware implementation of an algorithm is one in which a wide variety of finite precision representations of different sizes are used for different internal variables. The size of the representation of a finite precision 'word' is referred to as its word-length. Implementations utilizing several different word-lengths are referred to as 'multiple word-length' implementations and are discussed in detail in this book.

The accuracy observable at the outputs of a DSP system is a function of the word-lengths used to represent all intermediate variables in the algorithm. However, accuracy is less sensitive to some variables than to others, as is implementation area. It is demonstrated in this book that by considering error and area information in a structured way using analytical and semi-analytical noise models, it is possible to achieve highly efficient DSP implementations.

Multiple word-length implementations have recently become a flourishing area of research [KWCM98, WP98, CRS+99, SBA00, BP00, KS01, NHCB01]. Stephenson [Ste00] enumerates three target areas for this research: SIMD architectures for multimedia [PW96], power conservation in embedded systems [BM99], and direct hardware implementations. Of these areas, this book

targets the latter, although Chapters 3 to 5 could form the basis of an approach to the first two application areas.

Throughout the book, both the word-length of operations, and the overflow methods used, are considered to be optimization variables for minimizing the area or power consumption of a hardware implementation. At the same time, they impost constraints on possible solutions on the basis of signal quality at the system outputs. The resulting multiple word-length implementations pose new challenges to the area of high-level synthesis [Cam90], which are also addressed in this book.

1.2 Overview

The overall design flow proposed and discussed is illustrated in Fig. 1.1. Each of the blocks in this diagram will be discussed in more detail in the chapters to follow.

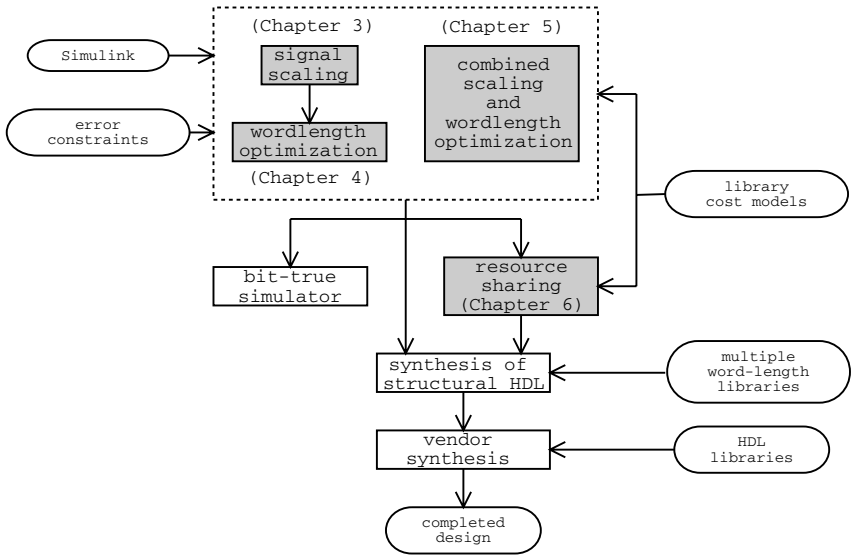


Fig. 1.1. System design flow and relationship between chapters

We begin in Chapter 2 by reviewing some relevant backgroud material, including a very brief introduction to important nomenclature in DSP, digital design, and algorithm representation. The key idea here is that in an efficient hardware implementation of a DSP algorithm, the representation used for each signal can be different from that used for other signals. Our representation consists of two parts: the scaling and the word-length. The optimization of these two parts are covered respectively in Chapters 3 and 4.

Chapter 3 reviews approaches to determining the peak signal value in a signal processing system, a fundamental problem when selecting an appropriate fixed precision representation for signals.

Chapter 4 introduces and formalizes the idea of a multiple word-length implementation. An analytic noise model is described for the modelling of signal truncation noise. Techniques are then introduced to optimize the word-lengths of the variables in an algorithm in order to achieve a minimal implementation area while satisfying constraints on output signal quality. After an analysis of the nature of the constraint space in such an optimization, we introduce a heuristic algorithm to address this problem. An extension to the method is presented for nonlinear systems containing differentiable nonlinear components, and results are presented illustrating the advantages of the methods described for area, speed, and power consumption.

Chapter 5 continues the above discussion, widening the scope to include the ability to predict the severity of overflow-induced errors. This is exploited by the proposed combined word-length and scaling optimization algorithm in order to automate the design of saturation arithmetic systems.

Chapter 6 addresses the implications of the proposed multiple word-length scheme for the problem of architectural synthesis. The chapter starts by highlighting the differences between architectural synthesis for multiple word-length systems and the standard architectural synthesis problems of scheduling, resource allocation, and resource binding. Two methods to allow the sharing of arithmetic resources between multiple word-length operations are then proposed, one optimal and one heuristic.

Notation will be introduced in the book as required. For convenience, some basic notations required throughout the book are provided in Appendix A, p. 151. Some of the technical terms used in the book are also described in the glossary, p. 153. In addition, it should be noted that for ease of reading the box symbol: □ is used throughout this book to denote the end of an example, definition, problem, or claim.

This page intentionally left blank

2

Background

This chapter provides some of the necessary background required for the rest of this book. In particular, since this book is likely to be of interest both to DSP engineers and digital designers, a basic introduction to the essential nomenclature within each of these fields is provided, with references to further material as required.

Section 2.1 introduces microprocessors and field-programmable gate arrays. Section 2.2 then covers the discrete-time description of signals using the z-transform. Finally, Section 2.3 presents the representation of DSP algorithms using computation graphs.

2.1 Digital Design for DSP Engineers

2.1.1 Microprocessors vs. Digital Design

One of the first options faced by the designer of a digital signal processing system is whether that system should be implemented in hardware or software. A software implementation forms an attractive possibility, due to the mature state of compiler technology, and the number of good software engineers available. In addition microprocessors are mass-produced devices and therefore tend to be reasonably inexpensive. A major drawback of a microprocessor implementation of DSP algorithms is the computational throughput achievable. Many DSP algorithms are highly parallelizable, and could benefit significantly from more fine-grain parallelism than that available with general purpose microprocessors. In response to this acknowledged drawback, general purpose microprocessor manufacturers have introduced extra single-instruction multiple-data (SIMD) instructions targetting DSP such as the Intel MMX instruction set [PW96] and Sun's VIS instruction set [TONH96]. In addition, there are microprocessors specialized entirely for DSP such as the well-known Texas Instruments DSPs [TI]. Both of these implementations allow higher throughput than that achievable with a general purpose processor, but there is still a significant limit to the throughput achievable.

The alternative to a microprocessor implementation is to implement the algorithm in custom digital hardware. This approach brings dividends in the form of speed and power consumption, but suffers from a lack of mature high-level design tools. In digital design, the industrial state of the art is register-transfer level (RTL) synthesis [IEE99, DC]. This form of design involves explicitly specifying the cycle-by-cycle timing of the circuit and the word-length of each signal within the circuit. The architecture must then be encoded using a mixture of data path and finite state machine constructs. The approaches outlined in this book allow the production of RTL-synthesizable code directly from a specification format more suitable to the DSP application domain.

2.1.2 The Field-Programmable Gate Array

There are two main drawbacks to designing an application-specific integrated circuit (ASICs) for a DSP application: money and time. The production of state of the art ASICs is now a very expensive process, which can only realistically be entertained if the market for the device can be counted in millions of units. In addition, ASICs need a very time consuming test process before manufacture, as 'bug fixes' cannot be created easily, if at all.

The Field-Programmable Gate Array (FPGA) can overcome both these problems. The FPGA is a programmable hardware device. It is mass-produced, and therefore can be bought reasonably inexpensively, and its programmability allows testing in-situ. The FPGA can trace its roots from programmable logic devices (PLDs) such as PLAs and PALs, which have been readily available since the 1980s. Originally, such devices were used to replace discrete logic series in order to minimize the number of discrete devices used on a printed circuit board. However the density of today's FPGAs allows a single chip to replace several million gates [Xil03]. Under these circumstances, using FPGAs rather than ASICs for computation has become a reality.

There are a range of modern FPGA architectures on offer, consisting of several basic elements. All such architectures contain the 4-input lookup table (4LUT or simply LUT) as the basic logic element. By configuring the data held in each of these small LUTs, and by configuring the way in which they are connected, a general circuit can be implemented. More recently, there has been a move towards heterogeneous architectures: modern FPGA devices such as Xilinx Virtex also contain embedded RAM blocks within the array of LUTs, Virtex II adds discrete multiplier blocks, and Virtex II pro [Xil03] adds PowerPC processor cores.

Although many of the approaches described in this book can be applied equally to ASIC and FPGA-based designs, it is our belief that programmable logic design will continue to increase its share of the market in DSP applications. For this reason, throughout this book, we have reported results from these methods when applied to FPGAs based on 4LUTs.

2.1.3 Arithmetic on FPGAs

Two arithmetic operations together dominate DSP algorithms: multiplication and addition. For this reason, we shall take the opportunity to consider how multiplication and addition are implemented in FPGA architectures. A basic understanding of the architectural issues involved in designing adders and multipliers is key to understanding the area models derived in later chapters of this book.

Many hardware architectures have been proposed in the past for fast addition. As well as the simple ripple-carry approach, these include carry-look-ahead, conditional sum, carry-select, and carry-skip addition [Kor02]. While the ASIC designer typically has a wide choice of adder implementations, most modern FPGAs have been designed to support fast ripple-carry addition. This means that often, 'fast' addition techniques are actually slower than ripple-carry in practice. For this reason, we restrict ourselves to ripple carry addition.

Fig. 2.1 shows a portion of the Virtex II 'slice' [Xil03], the basic logic unit within the Virtex II FPGA. As well as containing two standard 4LUTs, the slice contains dedicated multiplexers and XOR gates. By using the LUT to generate the 'carry propagate' select signal of the multiplexer, a two-bit adder can be implemented within a single slice.

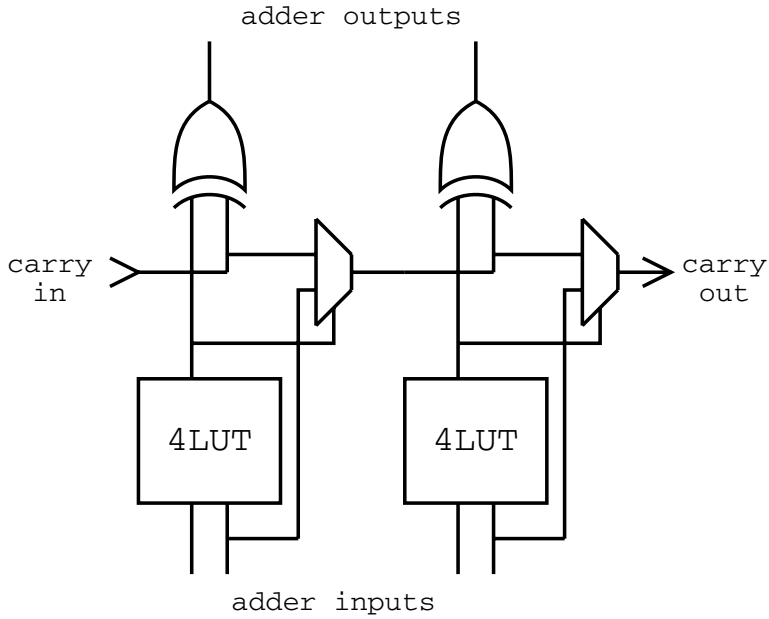


Fig. 2.1. A Virtex II slice configured as a 2-bit adder

In hardware arithmetic design, it is usual to separate the two cases of multiplier design: when one operand is a constant, and when both operands may vary. In the former case, there are many opportunities for reducing the hardware cost and increasing the hardware speed compared to the latter case. A constant-coefficient multiplication can be re-coded as a sum of shifted versions of the input, and common sub-expression elimination techniques can be applied to obtain an efficient implementation in terms of adders alone [Par99] (since shifting is free in hardware). General multiplication can be performed by adding partial products, and general multipliers essentially differ in the ways they accumulate such partial products. The Xilinx Virtex II slice, as well as containing a dedicated XOR gate for addition, also contains a dedicated AND gate, which can be used to calculate the partial products, allowing the 4LUTs in a slice to be used for their accumulation.

2.2 DSP for Digital Designers

A signal can be thought of as a variable that conveys information. Often a signal is one dimensional, such as speech, or two dimensional, such as an image. In modern communication and computation, such signals are often stored digitally. It is a common requirement to process such a signal in order to highlight or supress something of interest within it. For example, we may wish to remove noise from a speech signal, or we may wish to simply estimate the spectrum of that signal.

By convention, the value of a discrete-time signal x can be represented by a sequence $x[n]$. The index n corresponds to a multiple of the sampling period T, thus $x[n]$ represents the value of the signal at time nT. The z transform (2.1) is a widely used tool in the analysis and processing of such signals.

$$X(z) = \sum_{n=-\infty}^{+\infty} x[n]z^{-n} \tag{2.1}$$

The z transform is a linear transform, since if $X_1(z)$ is the transform of $x_1[n]$ and $X_2(z)$ is the transform of $x_2[n]$, then $\alpha X_1(z) + \beta X_2(z)$ is the transform of $\alpha x_1[n] + \beta x_2[n]$ for any real α, β. Perhaps the most useful property of the z transform for our purposes is its relationship to the convolution operation. The output $y[n]$ of any linear time-invariant (LTI) system with input $x[n]$ is given by (2.2), for some sequence $h[n]$.

$$y[n] = \sum_{k=-\infty}^{+\infty} h[k]x[n-k] \tag{2.2}$$

Here $h[n]$ is referred to as the impulse response of the LTI system, and is a fixed property of the system itself. The z transformed equivalent of (2.2), where $X(z)$ is the z transform of the sequence $x[n]$, $Y(z)$ is the z transform

of the sequence $y[n]$ and $H(z)$ is the z transform of the sequence $h[n]$, is given by (2.3). In these circumstances, $H(z)$ is referred to as the transfer function.

$$Y(z) = H(z)X(z) \tag{2.3}$$

For the LTI systems discussed in this book, the system transfer function $H(z)$ takes the rational form shown in (2.4). Under these circumstances, the values $\{z_1, z_2, \ldots, z_m\}$ are referred to as the *zeros* of the transfer function and the values $\{p_1, p_2, \ldots, p_n\}$ are referred to as the *poles* of the transfer function.

$$H(z) = K\frac{(z^{-1} - z_1^{-1})(z^{-1} - z_2^{-1})\ldots(z^{-1} - z_m^{-1})}{(z^{-1} - p_1^{-1})(z^{-1} - p_2^{-1})\ldots(z^{-1} - p_n^{-1})} \tag{2.4}$$

2.3 Computation Graphs

Synchronous Data Flow (SDF) is a widely used paradigm for the representation of digital signal processing systems [LM87b], and underpins several commerical tools such as Simulink from The MathWorks [SIM]. A simple example diagram from Simulink is shown in Fig. 2.2. Such a diagram is intuitive as a form of data-flow graph, a concept we shall formalize shortly. Each node represents an operation, and conceptually a node is ready to execute, or 'fire', if enough data are present on all its incoming edges.

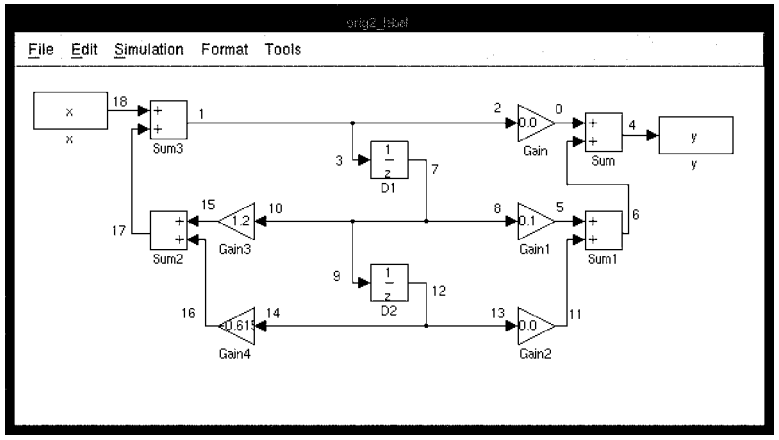


Fig. 2.2. A simple Simulink block diagram

In some chapters, special mention will be made of linear time invariant (LTI) systems. Individual computations in an LTI system can only be one of several types: constant coefficient multiplication, unit-sample delay, addition, or branch (fork). Of course the representation of an LTI system can be of a

hierarchical nature, in terms of other LTI systems, but each leaf node of any such representation must have one of these four types. A flattened LTI representation forms the starting point for many of the optimization techniques described.

We will discuss the representation of LTI systems, on the understanding that for differentiable nonlinear systems, used in Chapter 4, the representation is identical with the generalization that nodes can form any differentiable function of their inputs.

The representation used is referred to as a *computation graph* (Definition 2.1). A computation graph is a specialization of the data-flow graphs of Lee *et al.* [LM87b].

Definition 2.1. A *computation graph* $G(V, S)$ is the formal representation of an algorithm. V is a set of graph nodes, each representing an atomic computation or input/output port, and $S \subset V \times V$ is a set of directed edges representing the data flow. An element of S is referred to as a *signal*. The set S must satisfy the constraints on indegree and outdegree given in Table 2.1 for LTI nodes. The *type* of an atomic computation $v \in V$ is given by $\text{TYPE}(v)$ (2.5). Further, if V_G denotes the subset of V with elements of GAIN type, then $\text{COEF} : V_G \to \mathbb{R}$ is a function mapping the GAIN node to its coefficient.

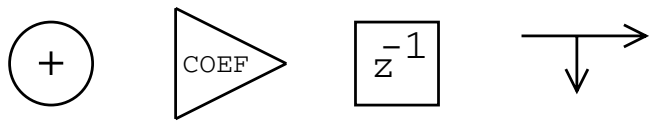
$$\text{TYPE} : V \to \{\text{INPORT}, \text{OUTPORT}, \text{ADD}, \text{GAIN}, \text{DELAY}, \text{FORK}\} \tag{2.5}$$

□

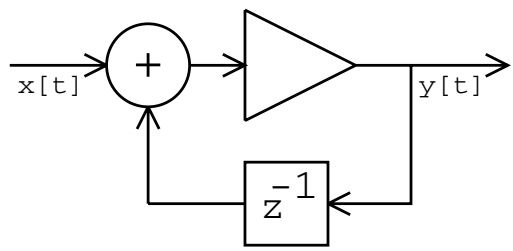
Table 2.1. Degrees of nodes in a computation graph

TYPE(v)	INDEGREE(v)	OUTDEGREE(v)
INPORT	0	1
OUTPORT	1	0
ADD	2	1
DELAY	1	1
GAIN	1	1
FORK	1	≥ 2

Often it will be useful to visualize a computation graph using a graphical representation, as shown in Fig. 2.3. Adders, constant coefficient multipliers and unit sample delays are represented using different shapes. The coefficient of a GAIN node can be shown inside the triangle corresponding to that node. Edges are represented by arrows indicating the direction of data flow. Fork nodes are implicit in the branching of arrows. INPORT and OUTPORT nodes are also implicitly represented, and usually labelled with the input and output names, $x[t]$ and $y[t]$ respectively in this example.



(a) some nodes in a computation graph



(b) an example computation graph

Fig. 2.3. The graphical representation of a computation graph

Definition 2.1 is sufficiently general to allow any multiple input, multiple output (MIMO) LTI system to be modelled. Such systems include operations such as FIR and IIR filtering, Discrete Cosine Transforms (DCT) and RGB to YCrCb conversion. For a computation to provide some useful work, its result must be in some way influenced by primary external inputs to the system. In addition, there is no reason to perform a computation whose result cannot influence external outputs. These observations lead to the definition of a well-connected computation graph (Definition 2.2). The computability property (Definition 2.4) for systems containing loops (Definition 2.3) is also introduced below. These definitions become useful when analyzing the properties of certain algorithms operating on computation graphs. For readers from a computer science background, the definition of a recursive system (Definition 2.3) should be noted. This is the standard DSP definition of the term which differs from the software engineering usage.

Definition 2.2. A computation graph $G(V, S)$ is *well-connected* iff (a) there exists at least one directed path from at least one node of type INPORT to each node $v \in V$ and (b) there exists at least one directed path from each node in $v \in V$ to at least one node of type OUTPORT. □

Definition 2.3. A *loop* is a directed cycle (closed path) in a computation graph $G(V, S)$. The *loop body* is the set of all vertices $V_1 \subset V$ in the loop. A computation graph containing at least one loop is said to describe a *recursive* system. □

Definition 2.4. A computation graph G is *computable* iff there is at least one node of type DELAY contained within the loop body of each loop in G. □

2.4 The Multiple Word-Length Paradigm

Throughout this book, we will make use of a number representation known as the *multiple word-length paradigm* [CCL01b]. The multiple word-length paradigm can best be introduced by comparison to more traditional fixed-point and floating-point implementations. DSP processors often use fixed-point number representations, as this leads to area and power efficient implementations, often as well as higher throughput than the floating-point alternative [IO96]. Each two's complement signal $j \in S$ in a multiple word-length implementation of computation graph $G(V, S)$, has two parameters n_j and p_j, as illustrated in Fig. 2.4(a). The parameter n_j represents the number of bits in the representation of the signal (excluding the sign bit), and the parameter p_j represents the displacement of the binary point from the LSB side of the sign bit towards the least-significant bit (LSB). Note that there are no restrictions on p_j; the binary point could lie outside the number representation, *i.e.* $p_j < 0$ or $p_j > n_j$.

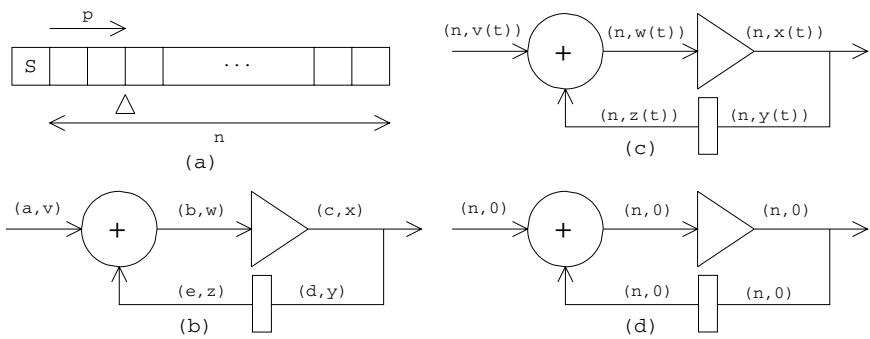


Fig. 2.4. The Multiple Word-Length Paradigm: (a) signal parameters ('s' indicates sign bit), (b) fixed-point, (c) floating-point, (d) multiple word-length

A simple fixed-point implementation is illustrated in Fig. 2.4(b). Each signal j in this block diagram representing a recursive DSP data-flow, is annotated with a tuple (n_j, p_j) showing the word-length n_j and scaling p_j of the signal. In this implementation, all signals have the same word-length and scaling, although shift operations are often incorporated in fixed-point designs, in order to provide an element of scaling control [KKS98]. Fig. 2.4(c) shows a standard floating-point implementation, where the scaling of each signal is a function of time.

A single uniform system word-length is common to both the traditional implementation styles. This is a result of historical implementation on single, or multiple, pre-designed fixed-point arithmetic units. Custom parallel hardware implementations can allow this restriction to be overcome for two reasons. Firstly, by allowing the parallelization of the algorithm so that different operations can be performed in physically distinct computational units. Secondly, by allowing the customization (and re-customization in FPGAs) of these computational units, and the shaping of the datapath precision to the requirements of the algorithm. Together these freedoms point towards an alternative implementation style shown in Fig. 2.4(d). This multiple word-length implementation style inherits the speed, area, and power advantages of traditional fixed-point implementations, since the computation is fixed-point with respect to each individual computational unit. However, by potentially allowing each signal in the original specification to be encoded by binary words with different scaling and word-length, the degrees of freedom in design are significantly increased.

An annotated computation graph $G'(V, S, A)$, defined in Definition 2.5, is used to represent the multiple word-length implementation of a computation graph $G(V, S)$.

Definition 2.5. An *annotated computation graph* $G'(V, S, A)$, is a formal representation of the fixed-point implementation of computation graph $G(V, S)$. A is a pair $(\mathbf{n}, \mathbf{p})$ of vectors $\mathbf{n} \in \mathbb{N}^{|S|}$, $\mathbf{p} \in \mathbb{Z}^{|S|}$, each with elements in one-to-one correspondence with the elements of S. Thus for each $j \in S$, it is possible to refer to the corresponding n_j and p_j. □

2.5 Summary

This chapter has introduced the basic elements in our approach. It has described the FPGAs used in our implementation, explained the desciption of signals using the z-transform, and the representation of algorithms using computation graphs. It has also provided an overview of the multiple word-length paradigm, which forms the basis of our design techniques described in the remaining chapters.

This page intentionally left blank

3

Peak Value Estimation

The physical representation of an intermediate result in a bit-parallel implementation of a DSP system consists of a finite set of bits, usually encoded using 2's complement representation. In order to make efficient use of the resources, it is essential to select an appropriate *scaling* for each signal. Such a scaling should be chosen to ensure that the representation is not overly wasteful, in catering for rare or impossibly large values, and simultaneously that overflow errors do not regularly occur, which would lead to low signal-to-noise ratio.

To determine an appropriate scaling, it is necessary to determine the peak value that each signal could reach. Given a peak value P, a power-of-two scaling p is selected with $p = \lfloor \log_2 P \rfloor + 1$, since power-of-two multiplication is cost-free in a hardware implementation.

For some DSP algorithms, it is possible to estimate the peak value that each signal could reach using analytic means. In Section 3.1, such techniques are discussed for two different classes of system. The alternative, to use simulation to determine the peak signal value, is described in Section 3.2, before a discussion of some hybrid techniques which aim to combine the advantages of both approaches in Section 3.3.

3.1 Analytic Peak Estimation

If the DSP algorithm under consideration is a linear, time-invariant system, it is possible to find a tight analytic bound on the peak value reachable by every signal in the system. This is the problem addressed by Section 3.1.1. If, on the other hand, the system is nonlinear or time-varying in nature, such approaches cannot be used. If the algorithm is non-recursive, i.e. the computation graph does not contain any feedback loops, then data-range propagation may be used to determine an analytic bound on the peak value of each signal. This approach, described in Section 3.1.2, cannot however be guaranteed to produce a tight bound.

3.1.1 Linear Time-Invariant Systems

For linear time-invariant systems, we restrict the type of each node in the computation graph to one of the following: INPORT, OUTPORT, ADD, GAIN, DELAY, FORK, as described in Chapter 2.

Transfer Function Calculation

The analytical scaling rules derived in this section rely on the knowledge of system transfer functions. A transfer function of a discrete-time LTI system between any given input-output pair is defined to be the z-transform of the sequence produced at that output, in response to a unit impulse at that input. These transfer functions may be expressed as the ratio of two polynomials in z^{-1}. The transfer function from each primary input to each signal must be calculated for signal scaling purposes. This section considers the problem of transfer function calculation from a computation graph. The reader familiar with transfer function calculation techniques may wish to skip the remainder of this section and turn to page 20.

Given a computation graph $G(V, S)$, let $V_I \subset V$ be the set of nodes of type INPORT, $V_O \subset V$ be the set of nodes of type OUTPORT, and $V_D \subset V$ be the set of nodes of type DELAY. A matrix of transfer functions $\mathbf{H}(z)$ is required. Matrix $\mathbf{H}(z)$ has elements $h_{iv}(z)$ for $i \in V_I$ and $v \in V$, representing the transfer function from primary input i to the output of node v.

Calculation of transfer functions for non-recursive systems is a simple task, leading to a matrix of polynomials in z^{-1}. The algorithm to find such a matrix is shown below. The algorithm works by constructing the transfer functions to the output of each node v in terms of the transfer functions to each of the nodes u driving v. If these transfer functions are unknown, then the algorithm performs a recursive call to find them. Since the system is non-recursive, the recursion will always terminate when a primary input is discovered, as primary inputs have no predecessor nodes.

Algorithm Scale_Non-Recurse
input: A computation graph $G(V, S)$
output: The matrix $\mathbf{H}(z)$
    initialize $\mathbf{H}(z) = \mathbf{0}$
    foreach $v \in V$
        call Find_Scale_Matrix($G(V, S)$, $\mathbf{H}(z)$, v)

Find_Scale_Matrix
input: A computation graph $G(V, S)$, partially complete $\mathbf{H}(z)$, node $v \in V$
output: Updated $\mathbf{H}(z)$
    if have already called Find_Scale_Matrix for node v, **return**
    foreach $(u, v) \in S$
        call Find_Scale_Matrix($G(V, S)$, $\mathbf{H}(z)$, u)

switch TYPE(v)
case ADD : $\forall i \in V_I,\ h_{i,v} \leftarrow \sum_{u \in \text{pred}(v)} h_{i,u}$
case INPORT : $h_{v,v} \leftarrow 1$
case GAIN : $\forall i \in V_I,\ h_{i,v} \leftarrow \text{COEF}(v) h_{i,\text{pred}(v)}$
case DELAY : $\forall i \in V_I,\ h_{i,v} \leftarrow z^{-1} h_{i,\text{pred}(v)}$
case FORK : $\forall i \in V_I,\ h_{i,v} \leftarrow h_{i,\text{pred}(v)}$
end switch

For recursive systems it is necessary to identify a subset $V_c \subseteq V$ of nodes whose outputs correspond to a system *state*. In this context, a state set consists of a set of nodes which, if removed from the computation graph, would break all feedback loops. Once such a state set has been identified, transfer functions can easily be expressed in terms of the outputs of these nodes using algorithm Scale_Non-Recurse, described above.

Let $\mathbf{S}(z)$ be a z-domain matrix representing the transfer function from each input signal to the output of each of these state nodes. The transfer functions from each input to each state node output may be expressed as in (3.1), where $\mathbf{A}$, and $\mathbf{B}$ are matrices of polynomials in z^{-1}. Each of these matrices represents a z-domain relationship once the feedback has been broken at the outputs of state-nodes. $\mathbf{A}(z)$ represents the transfer function between state-nodes and state-nodes, and $\mathbf{B}(z)$ represents the transfer functions between primary inputs and state-nodes.

$$\mathbf{S}(z) = \mathbf{A}(z)\mathbf{S}(z) + \mathbf{B}(z) \tag{3.1}$$

$$\mathbf{H}(z) = \mathbf{C}(z)\mathbf{S}(z) + \mathbf{D}(z) \tag{3.2}$$

The matrices $\mathbf{C}(z)$ and $\mathbf{D}(z)$ and are also matrices of polynomials in z^{-1}. $\mathbf{C}(z)$ represents the z-domain relationship between state-node outputs and the outputs of all nodes. $\mathbf{D}(z)$ represents the z-domain relationship between primary inputs and the outputs of all nodes.

It is clear that $\mathbf{S}(z)$ may be expressed as a matrix of rational functions (3.3), where $\mathbf{I}$ is the identity matrix of appropriate size. This allows the transfer function matrix $\mathbf{H}(z)$ to be calculated directly from (3.2).

$$\mathbf{S}(z) = (\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} \tag{3.3}$$

Example 3.1. Consider the simple computation graph from Chapter 2 shown in Fig. 2.3. Clearly removal of any one of the four internal nodes in this graph will break the feedback loop. Let us arbitrarily choose the adder node as a state node and choose the GAIN coefficient to be 0.1. The equivalent system with the feedback broken is illustrated in Fig. 3.1. The polynomial matrices $\mathbf{A}(z)$ to $\mathbf{D}(z)$ are shown in (3.4) for this example.

$$\begin{aligned}\mathbf{A}(z) &= 0.1z^{-1} \\ \mathbf{B}(z) &= 1 \\ \mathbf{C}(z) &= [0\ 1\ 0.1\ 0.1\ 0.1\ 0.1z^{-1}]^T \\ \mathbf{D}(z) &= [1\ 0\ 0\ 0\ 0\ 0]^T\end{aligned} \tag{3.4}$$

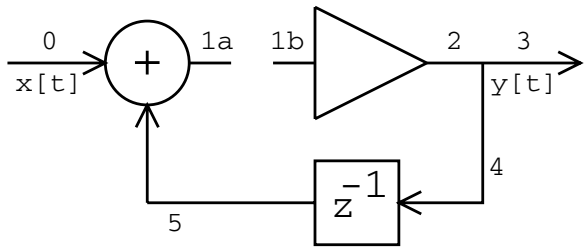


Fig. 3.1. An example of transfer function calculation (each signal has been labelled with a signal number)

Calculation of $\mathbf{S}(z)$ proceeds following (3.3), yielding (3.5). Finally, the matrix $\mathbf{H}(z)$ can be constructed following (3.2), giving (3.6).

$$\mathbf{S}(z) = \frac{1}{1 - 0.1z^{-1}} \tag{3.5}$$

$$\mathbf{H}(z) = [1\ \frac{1}{1-0.1z^{-1}}\ \frac{0.1}{1-0.1z^{-1}}\ \frac{0.1}{1-0.1z^{-1}}\ \frac{0.1}{1-0.1z^{-1}}\ \frac{0.1z^{-1}}{1-0.1z^{-1}}]^T \tag{3.6}$$

□

It is possible that the matrix inversion $(\mathbf{I} - \mathbf{A})^{-1}$ for calculation of $\mathbf{S}$ dominates the overall computational complexity, since the matrix inversion requires $|V_c|^3$ operations, each of which is a polynomial multiplication. The maximum order of each polynomial is $|V_D|$. This means that the number of scalar multiplications required for the matrix inversion is bounded from above by $|V_c|^3|V_D|^2$. It is therefore important from a computational complexity (and memory requirement) perspective to make V_c as small as possible.

If the computation graph $G(V, S)$ is computable, it is clear that $V_c = V_D$ is one possible set of state nodes, bounding the minimum size of V_c from above. If $G(V, S)$ is non-recursive, $V_c = \emptyset$ is sufficient. The general problem of finding the smallest possible V_c is well known in graph theory as the 'minimum feedback vertex set' problem [SW75, LL88, LJ00]. While the problem is known to be NP-hard for general graphs [Kar72], there are large classes of graphs for which polynomial time algorithms are known [LJ00]. However, since transfer function calculation does not require a *minimum* feedback vertex set, we suggest the algorithm of Levy and Low [LL88] be used to obtain a small feedback

vertex set. This algorithm is $O(|S| \log |V|)$ and is given below. The algorithm constructs the cutset V_c as the union of two sets V_c^1 and V_c^2. It works by contracting the graphs down to their essential structure by eliminating nodes with zero- or unit-indegree or outdegree. After contraction, any vertex with a self-loop must be part of the cutset (V_c^1). If no further contraction is possible and no self-loops exist, an arbitrary vertex is added to the cutset (V_c^2). Indeed it may be shown that if, on termination, $V_c^2 = \emptyset$, then the feedback vertex set V_c found is minimum.

Algorithm Levy–Low
input: A computation graph $G(V, S)$
output: A state set $V_c \subseteq V$

```
V_c^1 ← ∅
V_c^2 ← ∅
while V ≠ ∅ do
    V' ← V
    do
        foreach v ∈ V do
            if indegree(v) = 0 do
                V ← V \ {v}
                S ← S \ {(v, v')}
            end if
            if outdegree(v) = 0 do
                V ← V \ {v}
                S ← S \ {(v', v)}
            end if
            if indegree(v) = 1 do
                V ← V \ {v}
                S ← S ∪ {(v', u) : (v', v) ∈ S ∧ (v, u) ∈ S} \
                    ({(v', v)} ∪ {(v, v')})
            end if
            if outdegree(v) = 1 do
                V ← V \ {v}
                S ← S ∪ {(v', u) : (v', v) ∈ S ∧ (v, u) ∈ S} \
                    ({(v', v)} ∪ {(v, v')})
            end if
            if (v, v) ∈ S do
                V ← V \ {v}
                S ← S \ ({(v', v)} ∪ {(v, v')})
                V_c^1 ← V_c^1 ∪ {v}
            end if
        end foreach
    while V ≠ V'
    if V ≠ ∅ do
        select an arbitrary v ∈ V
```

$V \leftarrow V \setminus \{v\}$
$S \leftarrow S \setminus (\{(v', v)\} \cup \{(v, v')\})$
$V_c^2 \leftarrow V_c^2 \cup \{v\}$
end if
end while
$V_c \leftarrow V_c^1 \cup V_c^2$

For some special cases, there is a structure in the system matrices, which results in a simple decision for the vertex set V_c. In these cases, it is not necessary to apply the full Levy–Low algorithm. As an example of structure in the matrices of (3.1), consider the common computation graph of a large-order IIR filter constructed from second order sections in cascade. A second order section is illustrated in Fig. 3.2. Clearly removal of node A1 from each second order section is sufficient to break all feedback, indeed the set of all 'A1 nodes' is a minimum feedback vertex set for the chain of second order sections. By arranging the rows of matrix **A** appropriately, the matrix can be made triangular, leading to a trivial calculation procedure for $(\mathbf{I} - \mathbf{A})^{-1}$.

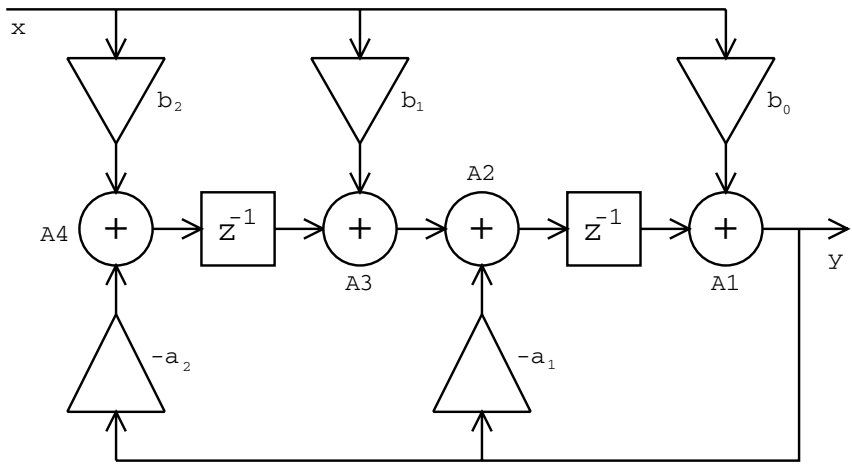


Fig. 3.2. A second order IIR section

Scaling with Transfer Functions

In order to produce the smallest fixed-point implementation, it is desirable to utilize as much as possible of the full dynamic range provided by each internal signal representation. The first step of the optimization process is therefore

to choose the smallest possible value of p_j for each signal $j \in S$ in order to guarantee no overflow.

Consider an annotated computation graph $G'(V, S, A)$, with $A = (\mathbf{n}, \mathbf{p})$. Let $V_I \subset V$ be the set of INPORTs, each of which reaches peak signal values of $\pm M_i$ ($M_i > 0$) for $i \in V_I$. Let $\mathbf{H}(z)$ be the scaling transfer function matrix defined in Section 3.1.1, with associated impulse response matrix $\mathbf{h}[t] = \mathcal{Z}^{-1}\{\mathbf{H}(z)\}$. Then the worst-case peak value P_j reached by any signal $j \in S$ is given by maximizing the well known convolution sum (3.7) [Mit98], where $x_i[t]$ is the value of the input $i \in V_I$ at time index t. Solving this maximization problem provides the input sequence given in (3.8), and allowing $N_{ij} \to \infty$ leads to the peak response at signal j given in (3.9). Here $\mathrm{sgn}(\cdot)$ is the signum function (3.10).

$$P_j = \pm \sum_{i \in V_I} \max_{x_i[t']} \left(\sum_{t=0}^{N_{ij}-1} x_i[t' - t] h_{ij}[t] \right) \tag{3.7}$$

$$x_i[t] = M_i \, \mathrm{sgn}(h_{ij}[N_{ij} - t - 1]) \tag{3.8}$$

$$P_j = \sum_{i \in V_I} M_i \sum_{t=0}^{\infty} |h_{ij}[t]| \tag{3.9}$$

$$\mathrm{sgn}(x) = \begin{cases} 1, & x \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{3.10}$$

This worst-case approach leads to the concept of ℓ_1 scaling, defined in Definitions 3.2 and 3.3.

Definition 3.2. The *ℓ_1-norm* of a transfer function $H(z)$ is given by (3.11), where $\mathcal{Z}^{-1}\{\cdot\}$ denotes the inverse z-transform.

$$\ell_1\{H(z)\} = \sum_{t=0}^{\infty} |\mathcal{Z}^{-1}\{H(z)\}[t]| \tag{3.11}$$

□

Definition 3.3. The annotated computation graph $G'(V, S, A)$ is said to be *ℓ_1-scaled* iff (3.12) holds for all signals $j \in S$. Here V_I, M_i and $h_{ij}(z)$ are as defined in the preceding paragraphs.

$$p_j = \left\lfloor \log_2 \sum_{i \in V_I} M_i \ell_1\{h_{ij}(z)\} \right\rfloor + 1 \tag{3.12}$$

□

3.1.2 Data-range Propagation

If the algorithm under consideration is not linear, or is not time-invariant, then one mechanism for estimating the peak value reached by each signal is to consider the propagation of data ranges through the computation graph. This is only possible for non-recursive algorithms.

Forward Propagation

A naïve way of approaching this problem is to examine the binary point position "naturally" resulting from each hardware operator. Such an approach, illustrated below, is an option in the Xilinx system generator tool [HMSS01].

Consider the computation graph shown in Fig. 3.3. If we consider that each input has a range $(-1, 1)$, then we require a binary point location of $p = \lfloor \log_2 \max |(-1, 1)| \rfloor + 1 = 0$ at each input. Let us consider each of the adders in turn. Adder $a1$ adds two inputs with $p = 0$, and therefore produces an output with $p = \max(0, 0)+1 = 1$. Adder $a2$ adds one input with $p = 0$ and one with $p = 1$, and therefore produces an output with $p = \max(0, 1) + 1 = 2$. Similarly, the output of $a3$ has $p = 3$, and the output of $a4$ has $p = 4$. While we have successfully determined a binary point location for each signal that will not lead to overflow, the disadvantage of this approach should be clear. The range of values reachable by the system output is actually $5*(-1, 1) = (-5, 5)$, so $p = \lfloor \log_2 \max(-5, 5) \rfloor + 1 = 3$ is sufficient; $p = 4$ is an overkill of one MSB.

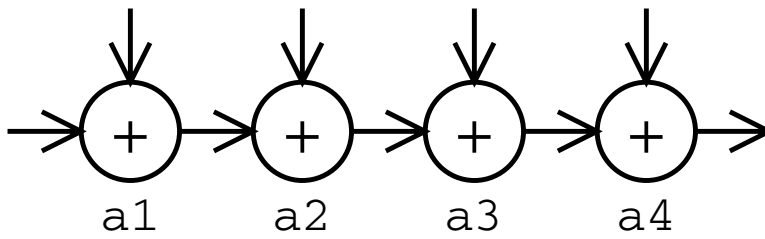


Fig. 3.3. A computation graph representing a string of additions

A solution to this problem that has been used in practice, is to propagate data ranges rather than binary point locations [WP98, BP00]. This approach can be formally stated in terms of interval analysis. Following [BP00],

Definition 3.4. An *interval extension*, denoted by $\mathbf{f}(\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_n)$, of a real function $f(x_1, x_2, \ldots, x_n)$ is defined as any function of the n intervals $\mathbf{x}_1$, $\mathbf{x}_2$, $\ldots$, $\mathbf{x}_n$ that evaluates to the value of f when its arguments are the degenerate intervals x_1, x_2, $\ldots$, x_n, i.e.

$$\mathbf{f}(x_1, x_2, \ldots, x_n) = f(x_1, x_2, \ldots, x_n) \tag{3.13}$$

Definition 3.5. If $\mathbf{x}_i \subseteq \mathbf{y}_i$, for $i = 1, 2, \ldots, n$ and $\mathbf{f}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \subset \mathbf{f}(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n)$, then the interval extension $\mathbf{f}(\mathbf{X})$ is said to be *inclusion monotonic*.

Let us denote by $\mathbf{f}^r(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ the range of function f over the given intervals. We may then use the result that $\mathbf{f}^r(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \subseteq \mathbf{f}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ [Moo66] to find an upper-bound on the range of the function.

Let us apply this technique to the example of Fig. 3.3. We may think of each node in the computation graph as implementing a distinct function. For addition, $f(x, y) = x + y$, and we may define the inclusion monotonic interval extension $\mathbf{f}((x^1, x^2), (y^1, y^2)) = (x^1 + y^1, x^2 + y^2)$. Then the output of adder $a1$ is a subset of $(-2, 2)$ and thus is assigned $p = 1$, the output of adder $a2$ is a subset of $(-3, 3)$ and is thus assigned $p = 2$, the output of adder $a3$ is a subset of $(-4, 4)$ and is thus assigned $p = 3$, and the output of adder $a4$ is a subset of $(-5, 5)$ and is thus assigned $p = 3$. For this simple example, the problem of peak-value detection has been solved, and indeed $\mathbf{f}^r = \mathbf{f}$.

However, such a tight solution is not always possible with data-range propagation. Under circumstances where the DFG contains one or more branches (fork nodes), which later reconverge, such a "local" approach to range propagation can be overly pessimistic. As an example, consider the computation graph representing a complex constant coefficient multiplication shown in Fig. 3.4.

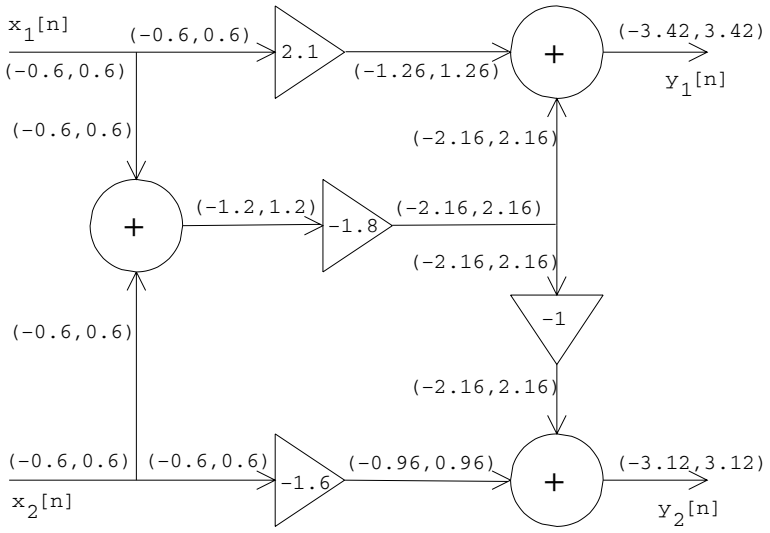


Fig. 3.4. Range propagation through a computation graph

Each signal has been labelled with a propagated range, assuming that the primary inputs have range $(-0.6, 0.6)$. Under this approach, both outputs

require $p = 2$. However such ranges are overly pessimistic. The upper output in Fig. 3.4 is clearly seen to have the value $y_1 = 2.1x_1 - 1.8(x_1 + x_2) = 0.3x_1 - 1.8x_2$. Thus the range of this output can also be calculated as $0.3(-0.6, 0.6) - 1.8(-0.6, 0.6) = (-1.26, 1.26)$. Similarly for the lower output $y_2 = -1.6x_2 + 1.8(x_1 + x_2) = 1.8x_1 + 0.2x_2$, providing a range $1.8(-0.6, 0.6) + 0.2(-0.6, 0.6) = (-1.2, 1.2)$. Thus by examining the global system behaviour, we can see that in reality $p = 1$ is sufficient for both outputs. Note that the analytic scheme described in Section 3.1.1 would calculate the tighter bound in this case.

In summary, range-propagation techniques may provide larger bounds on signal values than those absolutely necessary. This problem is seen in extremis with any recursive computation graph. In these cases, it is impossible to use range-propagation to place a finite bound on signal values, even in cases when such a finite bound can analytically be shown to exist.

3.2 Simulation-based Peak Estimation

A completely different approach to peak estimation is to use simulation: actually run the algorithm with a provided input data set, and measure the peak value reached by each signal.

In its simplest form, the simulation approach consists of measuring the peak signal value P_j reached by a signal $j \in S$ and then setting $p = \lfloor \log_2 kP_j \rfloor + 1$, where $k > 1$ is a user-supplied 'safety factor' typically having value 2 to 4. Thus it is ensured that no overflow will occur, so long as the signal value doesn't exceed $\hat{P}_j = kP_j$ when excited by a different input sequence. Particular care must therefore be taken to select an appropriate test sequence.

Kim and Kum [KKS98] extend the simulation approach by considering more complex forms of 'safety factor'. In particular, it is possible to try to extract information from the simulation relating to the class of probability density function followed by each signal. A histogram of the data values for each signal is built, and from this histogram the distribution is classified as: unimodal or multimodal, symmetric or non-symmetric, zero mean or non-zero mean.

For a unimodal symmetric distribution, Kim and Kum propose the heuristic safety scaling $\hat{P}_j = |\mu_j| + (\kappa_j + 4)\sigma_j$, where μ_j is the sample mean, κ_j is the sample kurtosis, and σ_j is the sample standard deviation (all measured during simulation).

For multimodal or non-symmetric distrubtion, the heuristic safety scaling $\hat{P}_j = P_j^{99.9\%} + 2(P_j^{100\%} - P_j^{99.9\%})$, has been proposed where $P_j^{p\%}$ represents the simulation-measured p'th percentile of the sample.

In order to partially alleviate the dependence of the resulting scaling on the particular input data sequence chosen, it is possible to simulate with several different data sets. Let the maximum and minimum values of the standard deviation (over the different data sets) be denoted $\sigma_{\max}$ and $\sigma_{\min}$ respectively.

Then the proposal of Kim and Kum [KKS98] is to use the heuristic estimate $\sigma = 1.1\sigma_{\max} - 0.1\sigma_{\min}$. A similar approach is proposed for the other collected statistics.

Simulation approaches are appropriate for nonlinear or time-varying systems, for which the data-range propagation approach described in Section 3.1.2 provides overly pessimistic results (such as for recursive systems). The main drawback of simulation-based approaches is the significant dependence on the input data set used for simulation; moreover no general guidelines can be given for how to select an appropriate input.

3.3 Hybrid Techniques

Simulation can be combined with data-range propagation in order to try and combine the advantages of the two techniques [CRS+99, CH02].

A pure simulation, without 'safety factor', may easily underestimate the required data range of a signal. Thus the scaling resulting from a simulation can be considered as a lower-bound. In contrast, a pure data-range propagation will often significantly overestimate the required range, and can thus be considered as an upper-bound. Clearly if the two approaches result in an identical scaling assignment for a signal, the system can be confident that simulation has resulted in an optimum scaling assignment. The question of what the system should do with signals where the two scalings do not agree is more complex.

Cmar *et al.* [CRS+99] propose the heuristic distinction between those simulation and propagation scalings which are 'significantly different' and those which are not. In the case that the two scalings are similar, say different by one bit position, it may not be a significant hardware overhead to simply use the upper-bound derived from range propagation.

If the scalings are significantly different, one possibility is to use saturation arithmetic logic to implement the node producing the signal. When an overflow occurs in saturation arithmetic, the logic saturates the output value to the maximum positive or negative value representable, rather than causing a two's complement wrap-around effect. In effect the system acknowledges it 'does not know' whether the signal is likely to overflow, and introduces extra logic to try and mitigate the effects of any such overflow. Saturation arithmetic will be considered in much more detail in Chapter 5.

3.4 Summary

This chapter has covered several methods for estimating the peak value that a signal can reach, in order to determine an appropriate scaling for that signal, resulting in an efficient representation. We have examined both analytic

and simulation-based techniques for peak estimation, and review hybrid approaches that aim to combine their strengths. We shall illustrate the combination of such techniques with word-length optimization approaches in the next chapter.

4

Word-Length Optimization

The previous chapter described different techniques to find a scaling, or binary point location, for each signal in a computation graph. This chapter addresses the remaining signal parameter: its word-length.

The major problem in word-length optimization is to determine the error at system outputs for a given set of word-lengths and scalings of all internal variables. We shall call this problem *error estimation*. Once a technique for error estimation has been selected, the word-length selection problem reduces to utilizing the known area and error models within a constrained optimization setting: find the minimum area implementation satisfying certain constraints on arithmetic error at each system output.

The majority of this chapter is therefore taken up with the problem of error estimation (Section 4.1). After discussion of error estimation, the problem of area modelling is addressed in Section 4.2, after which the word-length optimization problem is formulated and analyzed in Section 4.3. Optimization techniques are introduced in Section 4.4 and Section 4.5, results are presented in Section 4.6 and conclusions are drawn in Section 4.7.

4.1 Error Estimation

The most generally applicable method for error estimation is simulation: simulate the system with a given 'representative' input and measure the deviation at the system outputs when compared to an accurate simulation (usually 'accurate' means IEEE double-precision floating point [IEE85]). Indeed this is the approach taken by several systems [KS01, CSL01]. Unfortunately simulation suffers from several drawbacks, some of which correspond to the equivalent simulation drawbacks discussed in Chapter 3, and some of which are peculiar to the error estimation problem. Firstly, there is the problem of dependence on the chosen 'representative' input data set. Secondly, there is the problem of speed: simulation runs can take a significant amount of time, and during an optimization procedure a large number of simulation runs may be needed.

Thirdly, even the 'accurate' simulation will have errors induced by finite word-length effects which, depending on the system, may not be negligible.

Traditionally, much of the research on estimating the effects of truncation and roundoff noise in fixed-point systems has focussed on implementation using, or design of, a DSP uniprocessor. This leads to certain constraints and assumptions on quantization errors: for example that the word-length of all signals is the same, that quantization is performed after multiplication, and that the word-length before quantization is much greater than that following quantization [OW72]. The multiple word-length paradigm allows a more general design space to be explored, free from these constraints (Chapter 2).

The effect of using finite register length in fixed-point systems has been studied for some time. Oppenheim and Weinstein [OW72] and Liu [Liu71] lay down standard models for quantization errors and error propagation through linear time-invariant systems, based on a linearization of signal truncation or rounding. Error signals, assumed to be uniformly distributed, with a white spectrum and uncorrelated, are added whenever a truncation occurs. This approximate model has served very well, since quantization error power is dramatically affected by word-length in a uniform word-length structure, decreasing at approximately 6dB per bit. This means that it is not necessary to have highly accurate models of quantization error power in order to predict the required signal width [OS75]. In a multiple word-length system realization, the implementation error power may be adjusted much more finely, and so the resulting implementation tends to be more sensitive to errors in estimation.

Signal-to-noise ratio (SNR), sometimes referred to as signal-to-quantization-noise ratio (SQNR), is a generally accepted metric in the DSP community for measuring the quality of a fixed point algorithm implementation [Mit98]. Conceptually, the output sequence at each system output resulting from a particular finite precision implementation can be subtracted from the equivalent sequence resulting from an infinite precision implementation. The resulting difference is known as the *fixed-point error*. The ratio of the output power resulting from an infinite precision implementation to the fixed-point error power of a specific implementation defines the signal-to-noise ratio. For the purposes of this chapter, the signal power at each output is fixed, since it is determined by a combination of the input signal statistics and the computation graph $G(V, S)$. In order to explore different implementations $G'(V, S, A)$ of the computation graph, it is therefore sufficient to concentrate on noise estimation, which is the subject of this section.

Once again, the approach taken to word-length optimization should depend on the mathematical properties of the system under investigation. We shall not consider simulation-based estimation further, but instead concentrate on analytic or semi-analytic techniques that may be applied to certain classes of system. Section 4.1.2 describes one such method, which may be used to obtain high-quality results for linear time-invariant computation graphs. This approach is then generalized in Section 4.1.3 to nonlinear systems con-

taining only differentiable nonlinear components. Before these special cases are addressed, some general useful procedures are discussed in Section 4.1.1.

4.1.1 Word-Length Propagation and Conditioning

In order to predict the quantization effect of a particular word-length and scaling annotation, it is necessary to propagate the word-length values and scalings from the inputs of each atomic operation to the operation output, as shown in Table 4.1. The 'q' superscript is used to indicate a word-length before quantization, *i.e.* truncation or rounding. The only non-trivial case is the adder, which has its various types illustrated in Fig. 4.1. The entries in Table 4.1 correspond to the common types of adders encountered in practice, as illustrated in Fig. 4.1(a-d), where $n_a > p_a - p_b$ or $n_b > p_b - p_a$. The rare cases, typically only encountered when severe output noise is tolerable, are shown in Fig. 4.1(e-f). Note that due to the commutativity of addition, these six types are reduced to three implementation cases.

Table 4.1. Propagation of word-lengths

TYPE(v)	Propagation rules for $j \in$ OUTEDGE(v)
GAIN	For input (n_a, p_a) and coefficient (n_b, p_b): $p'_j = p_a + p_b$ $n_j^{q\prime} = n_a + n_b$
ADD	For inputs (n_a, p_a) and (n_b, p_b): $p'_j = \max(p_a, p_b) + 1$ $n_j^{q\prime} = \max(n_a, n_b + p_a - p_b) - \min(0, p_a - p_b) + 1$ (for $n_a > p_a - p_b$ or $n_b > p_b - p_a$)
DELAY or FORK	For input (n_a, p_a): $p'_j = p_a$ $n_j^{q\prime} = n_j$

The word-length values derived through format propagation may then be adjusted according to the known scaling of the output signal (described in Chapter 3). If the scaled binary point location at signal j is p_j, whereas the propagated value derived is p'_j ($> p_j$), then this corresponds to a Most Significant Bit (MSB)-side width-reduction. An adjustment $n_j^q \leftarrow n_j^{q\prime} - (p'_j - p_j)$ must then be made, where $n_j^{q\prime}$ is the propagated word-length value, as illustrated in Fig. 4.2. Conceptually, this is inverse sign-extension, which may occur after either a multiplication or an addition. This analysis allows correlation information derived from scaling (Chapter 3) to be used to effectively take advantage of a type of 'don't-care condition' not usually considered by synthesis tools.

Example 4.1. A practical example can be seen from the computation graph illustrated in Fig. 4.3(a), which represents an algorithm for the multiplication

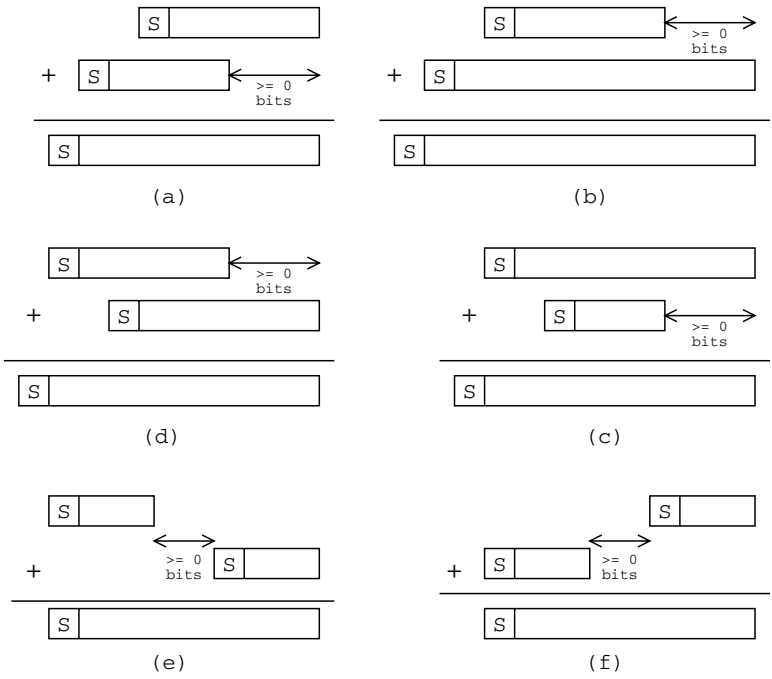


Fig. 4.1. Multiple word-length adder types. (a-d) are common adders, (e,f) are uncommon

of a complex input by a complex coefficient. Assume that each of the constant coefficient multipliers has coefficient of format $(8, 1)$ (except for the coefficient value -1 which has format $(1, 1)$).

Applying the propagations listed in Table 4.1 for the rightmost addition operations would predict the formats $(10, 3)$ and $(11, 3)$ for the outputs of the upper and lower addition, respectively. However, it can easily be seen through ℓ_1 scaling that binary point locations of $p = 1$ are sufficient to represent the peak values on both outputs. Consider the uppermost adder as an example. Although its two inputs can reach peak values of 1.26 and 2.16, it is known that the output can never exceed 1.26. Thus at the output of the adder, $n^{q'} = 10$ can be reduced to $n^q = 8$ through inverse sign-extension.

The computation graph shown in Fig. 4.3(a) is redrawn in Fig. 4.3(b) with the propagated and adjusted word-lengths n^q explicitly shown for all signals where a quantization (signal rounding or truncation) takes place. □

When designing a multiple word-length implementation $G'(V, S, A)$, it is important to avoid sub-optimal implementations which may arise through a bad choice of annotation A. Certain choices of A are clearly sub-optimal. Consider, for example, a GAIN node which multiplies signal j_1 of word-length n_{j_1} by a coefficient of format (n, p). If the output signal j_2 has been assigned

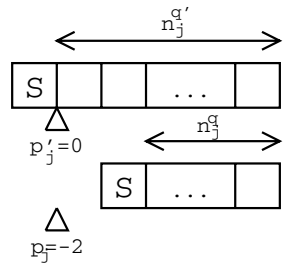
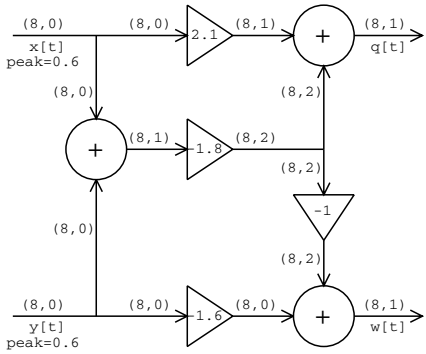
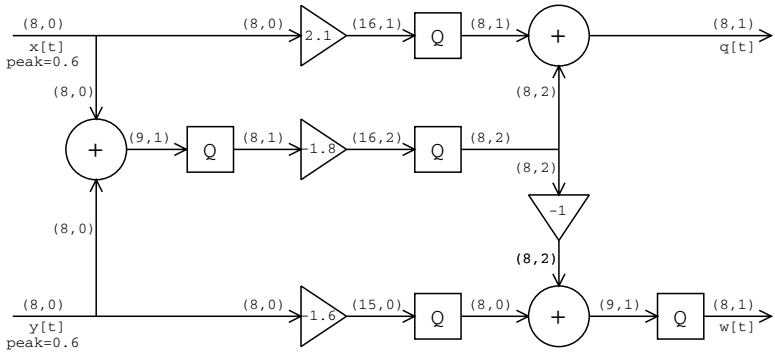


Fig. 4.2. Word-Length and Scaling adjustment



(a) an annotated computation graph



(b) adjusted propagated wordlengths n^q and quantizers "Q" explicitly shown

Fig. 4.3. Word-Length propagation through an annotated computation graph

word-length $n_{j_2} > n_{j_1} + n$, then this assignment is sub-optimal, since at most $n_{j_1} + n$ bits are necessary to represent the result to full precision. Ensuring that such cases do not arise is referred to as 'conditioning' the annotated computation graph [CCL01b]. Conditioning is an important design step, as it allows the search space of efficient implementations to be pruned, and ensures that the most efficient use is made of all bits of each signal. It is now possible to define a *well-conditioned* annotated computation graph to be one in which there are no superfluous bits representing any signal (Definition 4.2).

Definition 4.2. An annotated computation graph $G'(V, S, A)$ with $A = (\mathbf{n}, \mathbf{p})$ is said to be *well-conditioned* iff $n_j \leq n_j^q$ for all $j \in S$. □

During word-length optimization, ill-conditioned graphs may occur as intermediate structures. An ill-conditioned graph can always be transformed into an equivalent well-conditioned form in the iterative manner shown in Algorithm WLCondition.

Algorithm 4.1
Algorithm WLCondition
Input: An annotated computation graph $G'(V, S, A)$
Output: An annotated computation graph, with well-conditioned word-lengths and identical behaviour to the input system
begin
Calculate p'_j and $n_j^{q\prime}$ for all signals $j \in S$ (Table 4.1)
Form n_j^q from $n_j^{q\prime}$, p'_j and p_j for all signals $j \in S$
while $\exists j \in S : n_j^q < n_j$
Set $n_j \leftarrow n_j^q$
Update $n_j^{q\prime}$ for all affected signals (Table 4.1)
Re-form n_j^q from $n_j^{q\prime}$, p'_j and p_j for all affected signals
end while
end

4.1.2 Linear Time-Invariant Systems

We shall first address error estimation for linear time-invariant systems. An appropriate noise model for truncation of least-significant bits is introduced below. It is shown that the noise injected through truncation can be analytically propagated through the system, in order to measure the effect of such a noise on the system outputs. Finally, the approach is extended in order to provide detailed spectral information on the noise at system outputs, rather than simply a signal-to-noise ratio.

Noise Model

A common assumption in DSP design is that signal quantization (rounding or truncation) occurs only after a multiplication or multiply-accumulate operation. This corresponds to a uniprocessor viewpoint, where the result of

an n-bit signal multiplied by an n-bit coefficient needs to be stored in an n-bit register. The result of such a multiplication is a $2n$-bit word, which must therefore be quantized down to n bits. Considering signal truncation, the least area-expensive method of quantization [Fio98], the lowest value of the truncation error in two's complement with $p = 0$ is $2^{-2n} - 2^{-n} \approx -2^{-n}$ and the highest value is 0. It has been observed that values between these ranges tend to be equally likely to occur in practice, so long as the $2n$-bit signal has sufficient dynamic range [Liu71, OW72]. This observation leads to the formulation of a uniform distribution model for the noise [OW72], of variance $\sigma^2 = \frac{1}{12}2^{-2n}$ for the standard normalization of $p = 0$. It has also been observed that, under the same conditions, the spectrum of such errors tends to be white, since there is little correlation between low-order bits over time even if there is a correlation between high-order bits. Similarly, different truncations occurring at different points within the implementation structure tend to be uncorrelated.

When considering a multiple word-length implementation, or alternative truncation locations, some researchers have opted to carry this model over to the new implementation style [KS98]. However there are associated inaccuracies involved in such an approach [CCL99]. Firstly quantizations from n_1 bits to n_2 bits, where $n_1 \approx n_2$, will suffer in accuracy due to the discretization of the error probability density function. Secondly in such cases the lower bound on error can no longer be simplified in the preceding manner, since $2^{-n_2} - 2^{-n_1} \approx -2^{-n_1}$ no longer holds. Thirdly the multiple outputs from a branching node may be quantized to different word-lengths; a straight forward application of the model of [OW72] would not account for the correlations between such quantizations. Although these correlations would not affect the probability density function, they would cause inaccuracies when propagating these error signals through to primary system outputs, in the manner described below.

The first two of these issues may be solved by considering a discrete probability distribution for the injected error signal. For two's complement arithmetic the truncation error injection signal $e[t]$ caused by truncation from (n_1, p) to (n_2, p) is bounded by (4.1).

$$-2^p(2^{-n_2} - 2^{-n_1}) \leq e[t] \leq 0 \tag{4.1}$$

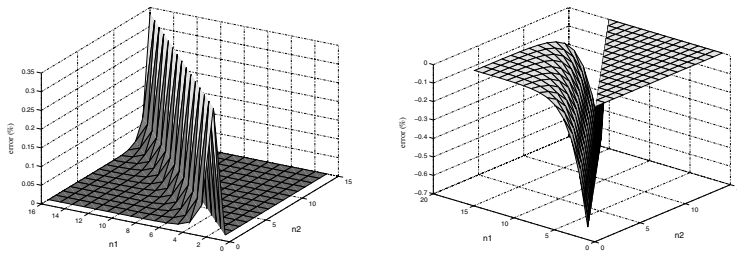
It is assumed that each possible value of $e[t]$ has equal probability, as discussed above. For two's complement truncation, there is non-zero mean $E\{e[t]\}$ (4.2) and variance σ_e^2 (4.3).

$$\begin{aligned} E\{e[t]\} &= -\tfrac{1}{2^{n_1-n_2}} \sum_{i=0}^{2^{n_1-n_2}-1} i \cdot 2^{p-n_1} \\ &= -2^{p-1}(2^{-n_2} - 2^{-n_1}) \end{aligned} \tag{4.2}$$

$$\begin{aligned}\sigma_e^2 &= \frac{1}{2^{n_1-n_2}} \sum_{i=0}^{2^{n_1-n_2}-1} (i \cdot 2^{p-n_1})^2 - E^2\{e[t]\} \\ &= \frac{1}{12} 2^{2p} (2^{-2n_2} - 2^{-2n_1})\end{aligned} \tag{4.3}$$

Note that for $n_1 \gg n_2$ and $p = 0$, (4.3) simplifies to $\sigma_e^2 \approx \frac{1}{12} 2^{-2n_2}$ which is the well-known predicted error variance of [OS75] for a model with continuous probability density function and $n_1 \gg n_2$.

A comparison between the error variance model in (4.3) and the standard model of [OS75] is illustrated in Fig. 4.4(a), showing errors of tens of percent can be obtained by using the standard simplifying assumptions. Shown in Fig. 4.4(b) is the equivalent plot obtained for a continuous uniform distribution in the range $[2^{-n_1} - 2^{-n_2}, 0]$ rather than $[2^{-n_2}, 0]$, which shows even further increases in error. It is clear that the discretization of the distribution around $n_1 \approx n_2$ may have a very significant effect on σ_e^2. While in a uniform word-length implementation this error may not be large enough to impact on the choice of word-length, multiple word-length systems are much more sensitive to such errors since finer adjustment of output signal quality is possible.



(a) (b)

Fig. 4.4. Error surface for (a) Uniform $[2^{-n_2}, 0]$ and (b) Uniform $[2^{-n_1} - 2^{-n_2}, 0]$

Accuracy is not the only advantage of the proposed discrete model. Consider two chained quantizers, one from (n_0, p) to (n_1, p), followed by one from (n_1, p) bits to (n_2, p) bits, as shown explicitly in Fig. 4.5. Using the error model of (4.3), the overall error variance injected, assuming zero correlation, is $\sigma_{e_0}^2 + \sigma_{e_1}^2 = \frac{1}{12} 2^{2p} (2^{-2n_1} - 2^{-2n_0} + 2^{-2n_2} - 2^{-2n_1})$. This quantity is equal to $\sigma_{e_2}^2 = \frac{1}{12} 2^{2p} (2^{-2n_2} - 2^{-2n_0})$. Therefore the error variance injected by considering a chain of quantizers is equal to that obtained when modelling the chain as a single quantizer. This is not the case with either of the continuous uniform distribution models discussed. The standard model of [OW72] gives $\sigma_{e_0}^2 + \sigma_{e_1}^2 = \frac{1}{12} (2^{2(p-n_1)} + 2^{2(p-n_2)})$. This is not equal to $\sigma_{e_2}^2 = \frac{1}{12} 2^{2(p-n_2)}$, and diverges significantly for $n_2 \approx n_1$. This consistency is another useful advantage of the proposed truncation model.

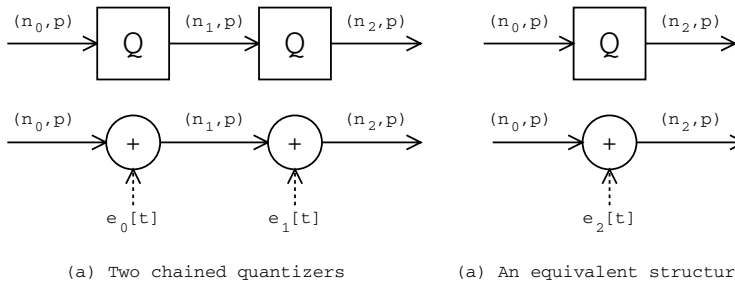


Fig. 4.5. (a) Two chained quantizers and their linearized model and (b) an equivalent structure and its linearized model

Noise Propagation and Power Estimation

Given an annotated computation graph $G'(V, S, A)$, it is possible to use the truncation model described in the previous section to predict the variance of each injection input. For each signal $j \in \{(v_1, v_2) \in S : \text{TYPE}(v_1) \neq \text{FORK}\}$ a straight-forward application of (4.3) may be used with $n_1 = n_j^q$, $n_2 = n_j$, and $p = p_j$, where n_j^q is as defined in Section 4.1.1. Signals emanating from nodes of FORK type must be considered somewhat differently. Fig. 4.6(a) shows one such annotated FORK structure, together with possible noise models in Figs. 4.6(b) and (c). Either model is valid, however Fig. 4.6(c) has the advantage that the error signals $e_0[t]$, $e_1[t]$ and $e_2[t]$ show very little correlation in practice compared to the structure of Fig. 4.6(b). This is due to the overlap in the low-order bits truncated in Fig. 4.6(b). Therefore the cascaded model is preferred, in order to maintain the uncorrelated assumption used to propagate these injected errors to the primary system outputs. Note also that the transfer function from each injection input is different under the two models. If in Fig. 4.6(b) the transfer functions from injection error inputs $e_0[t]$, $e_1[t]$ and $e_2[t]$ to a primary output k are given by $t_{0k}(z)$, $t_{1k}(z)$ and $t_{2k}(z)$, respectively, then for the model of Fig. 4.6(c), the corresponding transfer functions are $t_{0k}(z) + t_{1k}(z) + t_{2k}(z)$, $t_{1k}(z) + t_{2k}(z)$ and $t_{2k}(z)$.

By constructing noise sources in this manner for an entire annotated computation graph $G'(V, S, A)$, a set $F = \{(\sigma_p^2, \mathbf{R}_p)\}$ of injection input variances σ_p^2, and their associated transfer function to each primary output $\mathbf{R}_p(z)$, can be constructed. From this set it is possible to predict the nature of the noise appearing at the system primary outputs, which is the quality metric of importance to the user. Since the noise sources have a white spectrum and are uncorrelated with each other, it is possible to use L_2 scaling to predict the noise power at the system outputs. The L_2-norm of a transfer function $H(z)$ is defined in Definition 4.3. It can be shown that the noise variance E_k at output k is given by (4.4).

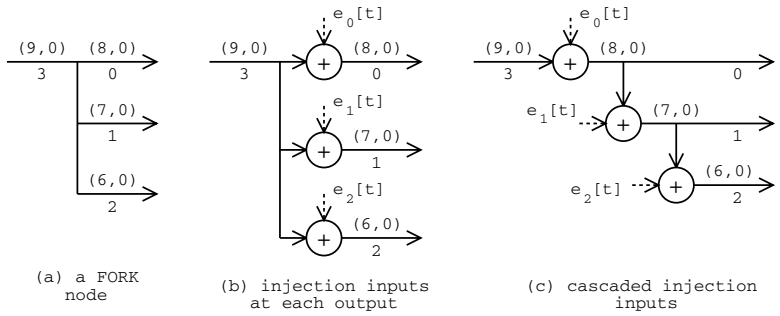


Fig. 4.6. Modelling post-FORK truncation

$$E_k = \sum_{(\sigma_p, \mathbf{R}_p) \in F} \sigma_p^2 L_2^2\{R_{pk}\} \tag{4.4}$$

Definition 4.3. The *L_2-norm* [Jac70] of a transfer function $H(z)$ is given by (4.5), where $\mathcal{Z}^{-1}\{\cdot\}$ denotes the inverse z-transform.

$$\begin{aligned} L_2\{H(z)\} &= \left(\frac{1}{2\pi} \int_{-\pi}^{\pi} |H(e^{j\theta})|^2 d\theta \right)^{\frac{1}{2}} \\ &= \left(\sum_{n=0}^{\infty} |\mathcal{Z}^{-1}\{H(z)\}[n]|^2 \right)^{\frac{1}{2}} \end{aligned} \tag{4.5}$$

□

Noise Spectral Bounds

There are some circumstances in which the SNR is not a sufficient quality metric. One such circumstance is if the user is especially concerned about noise in particular frequency bands. Consider, for example, the four output noise spectra illustrated in Fig. 4.7. All spectra have identical power, however environments sensitive to noise at roughly half the Nyquist frequency are unlikely to favour the top-left spectrum. Similarly environments sensitive to high-frequency or low-frequency noise are unlikely to favour the bottom-left or top-right spectra, respectively. Under these circumstances it is useful to explore fixed-point architectures having error spectra bounded by a user specified spectrum function. The two approaches of noise power and noise spectrum bounds complement each other, and may be used either together or separately to guide the word-length optimization procedures to be discussed in this chapter.

This can be achieved by predicting the entire power spectral density (PSD). The noise PSD $Q_k(z)$ at an output $k \in V$ of the algorithm may

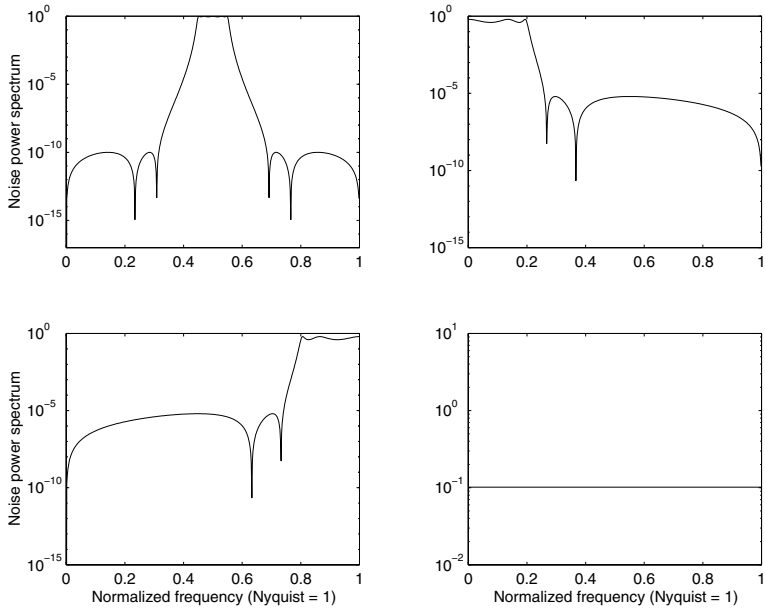


Fig. 4.7. Some different noise spectra with identical noise power

be estimated using the transfer functions $R_{pk}(z)$ from each noise source to the output as in (4.6), since each noise source has a white spectrum.

$$Q_k(z) = \sum_{(\sigma_p, \mathbf{R}_p) \in F} \sigma_p^2 |R_{pk}(z)|^2, \ (z = e^{j\theta}) \tag{4.6}$$

Once the noise PSD of a candidate implementation $G'(V, S, A)$ has been predicted, it is necessary to test whether the implementation satisfies the user-specified constraints. The proposed procedure tests an upper-bound constraint $|C_k(e^{j\theta})|^2$ for $Q_k(z)$, defined for all $\theta \in [0, 2\pi)$ by a real and rational function $C_k(z)$. The feasibility requirement can be expressed as (4.7).

$$Q_k(e^{j\theta}) < |C_k(e^{j\theta})|^2, \text{ for all } \theta \in [0, \pi] \tag{4.7}$$

If there is an output k such that $Q_k(e^{j0}) \geq |C_k(e^{j0})|^2$ or $Q_k(e^{j\pi}) \geq |C_k(e^{j\pi})|^2$ then clearly (4.7) does not hold and the feasibility test is complete. If neither of these conditions are true, and neither $Q_k(z)$ nor $C_k(z)$ have poles on the unit circle $|z| = 1$, then we may partition the set $[0, \pi]$ into $2n + 1$ subsets $\{\Theta_1 = [0, \theta_1), \Theta_2 = [\theta_1, \theta_2), \ldots, \Theta_{2n+1} = [\theta_{2n}, \pi]\}$ such that $Q_k(e^{j\theta}) < |C_k(e^{j\theta})|^2$ for $\theta \in \Theta_1 \cup \Theta_3 \cup \ldots \cup \Theta_{2n}$ and $Q_k(e^{j\theta}) \geq |C_k(e^{j\theta})|^2$ for $\theta \in \Theta_2 \cup \Theta_4 \cup \ldots \cup \Theta_{2n-1}$. Since $Q_k(z)$ and $C_k(z)$ have no poles on the unit circle, it may be deduced that $Q_k(e^{j\theta_1}) - |C_k(e^{j\theta_1})|^2 = Q_k(e^{j\theta_2}) - |C_k(e^{j\theta_2})|^2 = Q_k(e^{j\theta_{2n}}) - |C_k(e^{j\theta_{2n}})|^2 = 0$, and indeed that $Q_k(e^{j\theta}) - |C_k(e^{j\theta})|^2 = 0 \Leftrightarrow \theta =$

θ_i for some $1 \leq i \leq 2n$ and $\theta \in (0, \pi)$. Thus the problem of testing (4.7) in the general case has been reduced to the problem of locating those roots of the numerator polynomial of $F_k(z) = \text{numerator}(Q_k(z) - |C_k(z)|^2)$ that lie on the unit circle. In practice, locating roots can be a computation highly sensitive to numerical error [Act90]. The proposed approach is therefore to locate those 'approximate' roots lying in a small annulus around the unit circle, and then to test a single value of θ between the arguments of each successive pair of these roots to complete the feasibility test.

If $Q_k(z)$ or $C_k(z)$ have large order, there are well-known problems in locating roots accurately through polynomial deflation [Act90]. Since only those roots near the unit-circle need be located, it is proposed to use a procedure based on root moment techniques [Sta98] to overcome these problems. Root moments may be used to factor the numerator polynomial $F_k(z) = F_k^1(z)F_k^0(z)$ into two factors, $F_k^1(z)$ containing roots within the annulus of interest, and $F_k^0(z)$ containing all other roots. Once $F_k^1(z)$ has been extracted, Laguerre's method [PFTV88] may be applied to iteratively locate a single root z_0. The factor $(z - z_0)$, $(z - z_0)(z - z_0^*)$, $(z - z_0)(z - 1/z_0)$ or $(z - z_0)(z - z_0^*)(z - 1/z_0)(z - 1/z_0^*)$, depending on the location of z_0, may then be divided from the remaining polynomial before continuing with extraction of the next root.

This test can be used by the word-length optimization procedures described in Section 4.4 to detect violation of user-specified spectral constraints, and help guide the choice of word-length annotation towards a noise spectrum acceptable to the user.

4.1.3 Extending to Nonlinear Systems

With some modification, some of the results from the preceding section can be carried over to the more general class of nonlinear time-invariant systems containing only differentiable nonlinearities. In this section we address one possible approach to this problem, deriving from the type of small-signal analysis typically used in analogue electronics [SS91].

Perturbation Analysis

In order to make some of the analytical results on error sensitivity for linear, time-invariant systems [CCL01b] applicable to nonlinear systems, the first step is to linearize these systems. The assumption is made that the quantization errors induced by rounding or truncation are sufficiently small not to affect the macroscopic behaviour of the system. Under such circumstances, each component in the system can be locally linearized, or replaced by its "small-signal equivalent" [SS91] in order to determine the output behaviour under a given rounding scheme.

We shall consider one such n-input component, the differentiable function $Y[t] = f(X_1[t], X_2[t], \ldots, X_n[t])$, where t is a time index. If we denote by $x_i[t]$

a small perturbation on variable $X_i[t]$, then a first-order Taylor approximation for the induced perturbation $y[t]$ on $Y[t]$ is given by $y[t] \approx x_1[t]\frac{\partial f}{\partial X_1} + \ldots + x_n[t]\frac{\partial f}{\partial X_n}$.

Note that this approximation is linear in each x_i, but that the coefficients may vary with time index t since in general $\frac{\partial f}{\partial X_i}$ is a function of $X_1, X_2, \ldots, X_n$. Thus by applying such an approximation, we have produced a linear time-varying small-signal model for a nonlinear time-invariant component.

The linearity of the resulting model allows us to predict the error at system outputs due to *any* scaling of a small perturbation of signal $s \in S$ analytically, given the simulation-obtained error of a *single* such perturbation instance at s. Thus the proposed method can be considered to be a hybrid analytic / simulation error analysis.

Simulation is performed at several stages of the analysis, as detailed below. In each case, it is possible to take advantage of the static schedulability of the synchronous data-flow [LM87a] model implied by the algorithm representation, leading to an exceptionally fast simulation compared to event-driven simulation.

Derivative Monitors

In order to construct the small-signal model, we must first evaluate the differential coefficients of the Taylor series model for nonlinear components. Like other procedures described in this section, this is expressed as a graph transformation.

In general, methods must be introduced to calculate the differential of each nonlinear node type. This is performed by applying a graph transformation to the DFG, introducing the necessary extra nodes and outputs to calculate this differential. The general multiplier is the only nonlinear component considered explicitly in this section, although the approach is general; the graph transformation for multipliers is illustrated in Fig. 4.8. Since $f(X_1, X_2) = X_1X_2$, $\frac{\partial f}{\partial X_1} = X_2$ and $\frac{\partial f}{\partial X_2} = X_1$.

After insertion of the monitors, a (double-precision floating point) simulation may be performed to write-out the derivatives to appropriate data files to be used by the linearization process, to be described below.

Linearization

The construction of the small-signal model may now proceed, again through graph transformation. All linear components (adder, constant-coefficient multiplier, fork, delay, primary input, primary output) remain unchanged as a result of the linearization process. Each nonlinear component is replaced by its Taylor model. Additional primary inputs are added to the DFG to read the Taylor coefficients from the derivative monitor files created by the above large-signal simulation.

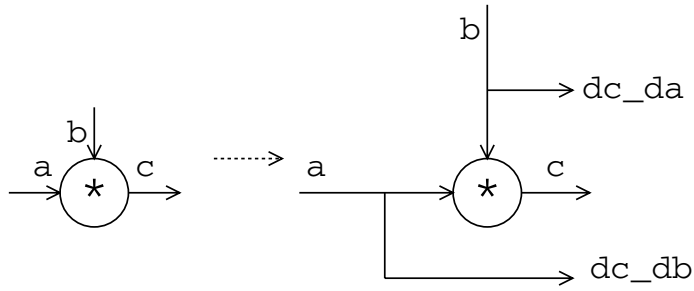


Fig. 4.8. Local graph transformation to insert derivative monitors

As an example, the Taylor expansion transformation for the multiplier node is illustrated in Fig. 4.9. Note that the graph portion in Fig. 4.9(b) still contains multiplier 'nonlinear' components, although one input of each multiplier node is now external to the model. This absence of feedback ensures linearity, although not time-invariance.

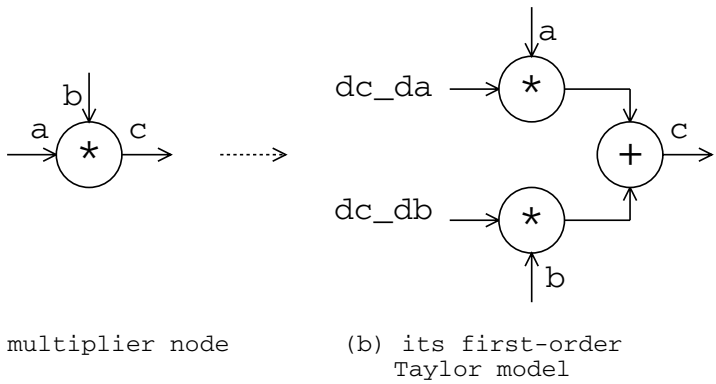


Fig. 4.9. Local graph transformation to produce small-signal model

Noise Injection

In Section 4.1.2, so-called L_2-scaling was used to analytically estimate the noise variance at a system output through scaling of the (analytically derived) noise variance injected at each point of quantization. Such a purely analytic technique can be used only for linear time-invariant systems; however in this section we suggest an extension of the approach for nonlinear systems.

Since the small-signal model is linear, if an output exhibits variance V when excited by an error of variance σ^2 injected into any given signal, then the output will exhibit variance αV when excited by a signal of variance $\alpha\sigma^2$ injected into the same signal ($0 < \alpha \in \mathbb{R}$). Herein lies the strength of the proposed linearization procedure: if the output response to a noise of known variance can be determined *once only* through simulation, this response can be scaled with analytically derived coefficients in order to estimate the response to *any* rounding or truncation scheme.

Thus the next step of the procedure is to transform the graph through the introduction of an additional adder node, and associated signals, and then simulate the graph with a known noise. In our case, to simulate truncation of a two's complement signal, the noise is independent and identically distributed with a uniform distribution over the range $[-2\sqrt{3}, 0]$. This range is chosen to have unit variance, thus making the measured output response an unscaled 'sensitivity' measure.

The graph transformation of inserting a noise injection is shown in Fig. 4.10. One of these transformations is applied to a distinct copy of the linearized graph for each signal in the DFG, after which zeros are propagated from the *original* primary-inputs, to finalize the small-signal model. This is a special case of constant propagation [ASU86] which leads to significantly faster simulation results for nontrivial DFGs.

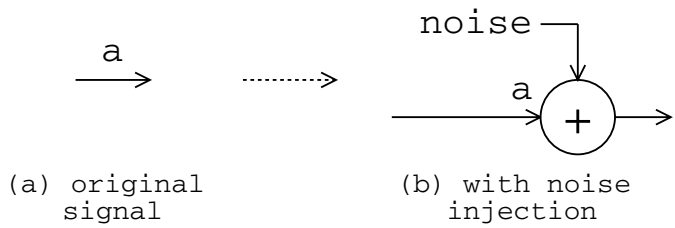


Fig. 4.10. Local graph transformation to inject perturbations

The entire process is illustrated for a simple DFG in Fig. 4.11. The original DFG is illustrated in Fig. 4.11(a). The perturbation analysis will be performed for the signals marked (*) and (**) in this figure. After inserting derivative monitors for nonlinear components, the transformed DFG is shown in Fig. 4.11(b). The linearized DFG is shown in Fig. 4.11(c), and its two variants for the signals (*) and (**) are illustrated in Figs. 4.11(d) and (e) respectively. Finally, the corresponding simplified DFGs after zero-propagation are shown in Figs. 4.11(f) and (g) respectively.

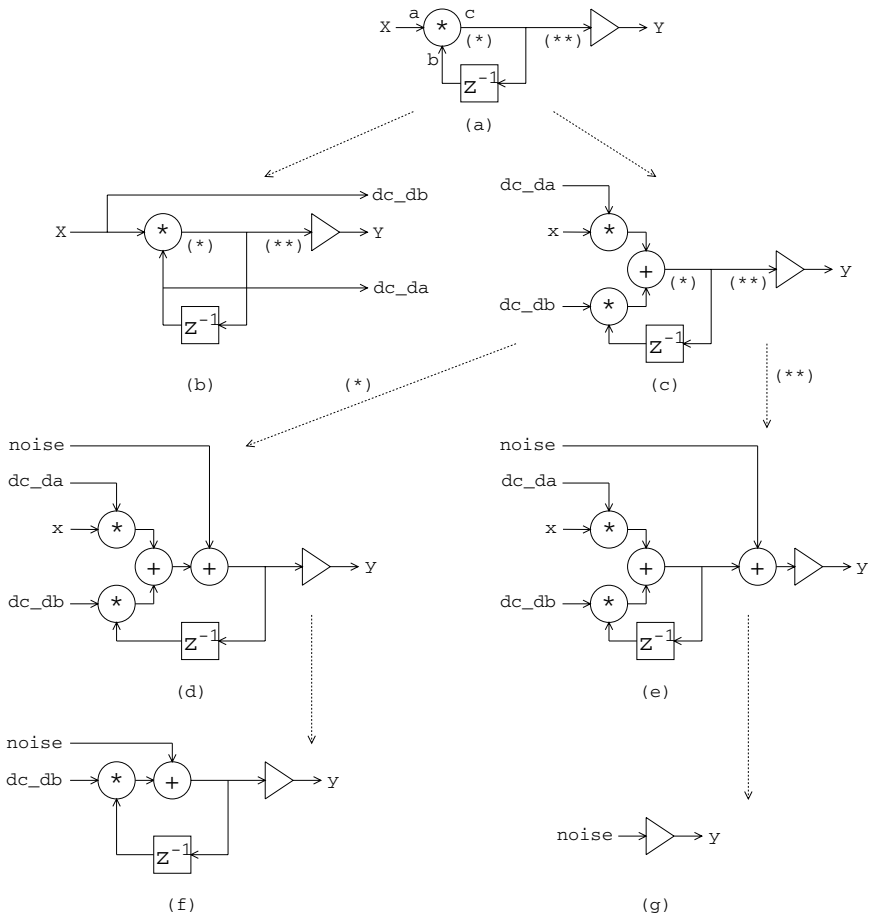


Fig. 4.11. Example perturbation analysis

4.2 Area Models

In order to implement a multiple word-length system, component libraries must be created to support multiple word-length arithmetic. These libraries can then be instantiated by the synthesis system to create synthesizable hardware description language, and must be modelled in terms of area consumption in order to provide the word-length optimization procedure with a cost metric.

Since an available target platform for Synthesis is the Altera-based SONIC reconfigurable computer [HSCL00], these component libraries have been built from existing Altera macros [Alt98]. Altera provides parameterizable macros for standard arithmetic functions operating on integer arithmetic, which form the basis of the multiple word-length libraries for the two arithmetic functions

of constant coefficient multiplication and addition. Integer arithmetic libraries are also available from many other FPGA vendors and ASIC library suppliers [Xil03, DW]. Multiple word-length libraries have also been constructed from the Synopsys DesignWare [DW] integer arithmetic libraries, for use in ASIC designs. Blocks from each of these vendors may have slightly different cost parameters, but the general approach described in this section is applicable across all vendors. The external interfaces of the two multiple word-length library blocks for GAIN and ADD are shown below in VHDL format [IEE99].

```
ENTITY gain IS
  GENERIC( INWIDTH, OUTWIDTH, NULLMSBS, COEFWIDTH : INTEGER;
           COEF : std_logic_vector( COEFWIDTH downto 0 ) );
  PORT( data : IN std_logic_vector( INWIDTH downto 0 );
        result : OUT std_logic_vector( OUTWIDTH downto 0 ) );
END gain;

ENTITY add IS
  GENERIC( AWIDTH, BWIDTH, BSHL, OUTWIDTH, NULLMSBS : INTEGER );
  PORT( dataa : IN std_logic_vector( AWIDTH downto 0 );
        datab : IN std_logic_vector( BWIDTH downto 0 );
        result : OUT std_logic_vector( OUTWIDTH downto 0 ) );
END add;
```

As well as individually parameterizable word-length for each input and output port, each library block has a `NULLMSBS` parameter which indicates how many most significant bits (MSBs) of the operation result are to be ignored (inverse sign extended). Thus each operation result can be considered to be made up of zero or more MSBs which are ignored, followed by one or more data bits, followed by zero or more least significant bits (LSBs) which may be truncated, depending on the `OUTWIDTH` parameter. For the adder library block, there is an additional `BSHL` generic which accounts for the alignment necessary for addition operands. `BSHL` represents the number of bits by which the `datab` input must conceptually be shifted left in order to align it with the `dataa` input. Note that since this is fixed-point arithmetic, there is no physical shifting involved; the data is simply aligned in a skew manner following Fig. 4.1. `dataa` and `datab` are permuted such that `BSHL` is always non-negative.

Each of the library block parameters has an impact on the area resources consumed by the overall system implementation. It is assumed, when constructing a cost model, both that a dedicated resource binding is to be used [DeM94], and that the area cost of wiring is negligible, *i.e.* the designs are *resource dominated* [DeM94]. A dedicated resource binding is one in which each computation node maps to a physically distinct library element. This assumption (relaxed in Chapter 6) simplifies the construction of an area cost model. It is sufficient to estimate separately the area consumed by each computation node, and then sum the resulting estimates. Of course in reality the logic synthesis, performed after word-length optimization, is likely to result

in some logic optimization between the boundaries of two connected library elements, resulting in lower area. Experience shows that these deviations from the area model are small, and tend to cancel each other out in large systems, resulting in simply a proportionally slightly smaller area than predicted.

It is extremely computationally intensive to perform logic synthesis each time an area metric is required for feedback into word-length cost estimation in optimization. It therefore is advisable to model the area consumption of each library element at a high level of abstraction using simple cost models which may be evaluated many times during word-length optimization with little computational cost. The remainder of this section examines the construction of these cost models.

The area model for a multiple word-length adder is reasonably straight forward. The ripple-carry architecture is used [Hwa79] since FPGAs provide good support for fast ripple-carry implementations [Alt98, Xil03]. The only area-consuming component is the core (integer) adder constructed from the vendor library. This adder has a width of $\max(\texttt{AWIDTH}-\texttt{BSHL}, \texttt{BWIDTH})-\texttt{NULLMSBS}+2$ bits. Each bit may not consume the same area, however, because some bits are required for the `result` port, whereas others may only be needed for carry propagation; their sum outputs remain unconnected and therefore the sum circuitry will be optimized away by logic synthesis. The cost model therefore has two parameters k_1 and k_2, corresponding to the area cost of a sum-and-carry full adder, and the area cost of a carry-only full adder respectively. The area of an adder is expressed in (4.8).

$$A_{\mathrm{ADD}}(\texttt{AWIDTH}, \texttt{BWIDTH}, \texttt{BSHL}, \texttt{NULLMSBS}, \texttt{OUTWIDTH}) = k_1(\texttt{OUTWIDTH}+1)+ k_2(\max(\texttt{AWIDTH}-\texttt{BSHL}, \texttt{BWIDTH})-\texttt{NULLMSBS}-\texttt{OUTWIDTH}+1) \tag{4.8}$$

Area estimation for constant coefficient multipliers is significantly more problematic. A constant coefficient multiplier is typically implemented as a series of additions, through a recoding scheme such as the classic Booth technique [Boo51]. This implementation style causes the area consumption to be highly dependent on coefficient value. In addition, the exact implementation scheme used by the vendor integer arithmetic libraries is known only to the vendor. Although an ideal area model would account for any recoding-based implementation, this currently remains unimplemented. Instead a simple area model has been constructed (4.9) and the coefficient values k_1 and k_2 have been determined through the synthesis of several hundred multipliers of different coefficient value and width. The model has then been fitted to these data using a least-squares approach. Note that the model does not account for `NULLMSBS` because for a properly scaled coefficient, $\texttt{NULLMSBS} \leq 1$ for a GAIN block, and therefore has little impact on area consumption.

$$A_{\text{GAIN}}(\texttt{INWIDTH}, \texttt{OUTWIDTH}, \texttt{COEFWIDTH}) = k_3\texttt{COEFWIDTH}(\texttt{INWIDTH}+1)+ k_4(\texttt{INWIDTH}+\texttt{COEFWIDTH}-\texttt{OUTWIDTH}) \tag{4.9}$$

4.3 Problem Definition and Analysis

Given a computation graph $G(V, S)$, Chapter 3 has described how a scaling vector $\mathbf{p}$ may be derived. Combining the area models presented in Section 4.2 into a single area measure on G gives a cost metric $A_G(\mathbf{n}, \mathbf{p})$. Combining the error variance model (Section 4.1) into a vector $\mathbf{E}_G(\mathbf{n}, \mathbf{p})$, with one element per output, allows the word-length optimization problem to be formulated as below. Here $\mathcal{E}$ denotes the user specified bounds on error variance.

Problem 4.4 (WORD-LENGTH OPTIMIZATION). Given a computation graph $G(V, S)$, the WORD-LENGTH OPTIMIZATION problem may be defined as to select $\mathbf{n}$ such that $A_G(\mathbf{n}, \mathbf{p})$ is minimized subject to (4.10).

$$\begin{aligned} &\mathbf{n} \in \mathbb{N}^{|S|} \\ &\mathbf{E}_G(\mathbf{n}, \mathbf{p}) \leq \mathcal{E} \end{aligned} \tag{4.10}$$

□

4.3.1 Convexity and Monotonicity

In this section some results are presented on the nature of the constraint space defined by (4.10). It is demonstrated that the error observable at primary system outputs may not be a monotonically decreasing function in each internal word-length. Moreover it is illustrated that error non-convexity may occur, causing the constraint space to be non-convex in $\mathbf{n}$.

It is often generally assumed that the greater the precision to which each internal variable is known, the greater the precision of the computation output. Indeed Sung and Kum explicitly state this assumption [SK95]. However a simple example is sufficient to illustrate that to make this assumption for multiple word-length arithmetic may be fallacious. Consider performing the operation $x = y - y$, using possibly different finite precision representations for the two y values. If the same representation of y is used for both inputs to the subtraction, then no matter how poor their precision, the result will have zero error. Using different representations for the two y values will often lead to a finite error value x. Of course this is a contrived example, and in practice such a calculation would not be performed. However this section illustrates that such a result is generalizable to practical applications and can lead to non-convexity in the constraint space.

Let us first consider signals produced by a single output computation node, which in turn forms the input to a GAIN node. An example is shown

in Fig. 4.12(a) with the truncation quantizers shown explicitly as 'Q' blocks. Consider reducing the word-length immediately before multiplication, from n_2 to $n_2 - 1$. Such a reduction will cause the word-length immediately after multiplication to decrease from n_3 to $n_3 - 1$. Let ΔQ_1 denote the change in quantization error at the pre-multiplier quantizer and ΔQ_2 denote the change in quantization error at the post-multiplier quantizer. Let the constant coefficient multiplier have coefficient β of word-length n and scaling p. The observable error variance cannot decrease at any output if $\beta^2 \Delta Q_1 + \Delta Q_2 \geq 0$. $\Delta Q_1 = 3 \cdot 2^{2(p_1 - n_2)}$ and $\Delta Q_2 = -3 \cdot 2^{2(p_2 - n_3)}$. The condition therefore reduces to $n_3 \geq n_2 + p_2 - p_1 - \log_2 |\beta|$. But $n_3 = n_2 + p_2 - p_1 + n - p$, and so the condition becomes $n \geq p - \log_2 |\beta|$. For an optimally scaled coefficient, $p = \lfloor \log_2 |\beta| \rfloor + 1$, and so the condition is satisfied for all positive n.

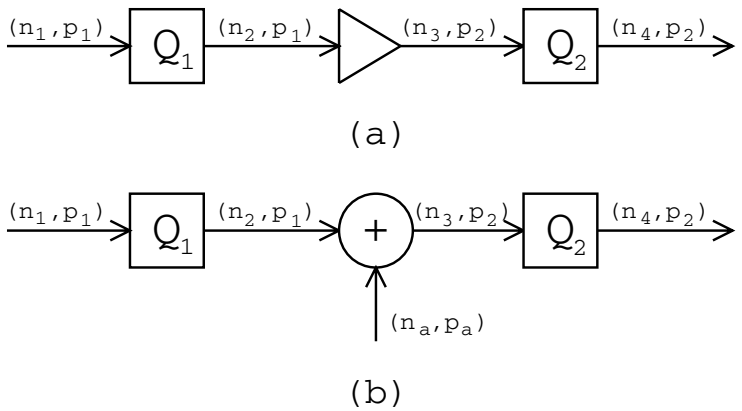


Fig. 4.12. Error behaviour for GAIN and ADD nodes

The preceding analysis extends directly to the case of signals produced by single-output computation nodes and driving DELAY or FORK nodes. For adders, the output word-length could either decrease to $n_3 - 1$ or remain at n_3, depending on the additional adder input with format (n_a, p_a) shown in Fig. 4.12(b). If the output word-length remains at n_3, $\Delta Q_2 = 0$ and the error variance at any primary output cannot decrease. If the word-length decreases to $n_3 - 1$, the picture becomes slightly more complex. In order to ensure the error variance does not decrease at any output, it is required that $\Delta Q_1 + \Delta Q_2 \geq 0$, *i.e.* $n_3 \geq n_2 + p_2 - p_1$. However from word-length propagation through the adder, it can be seen that $n_3 = \max(n_a, n_2 + p_a - p_1) - \min(0, p_a - p_1) + 1 - (\max(p_a, p_1) + 1 - p_2)$ and $n_3 - 1 = \max(n_a, n_2 - 1 + p_a - p_1) - \min(0, p_a - p_1) + 1 - (\max(p_a, p_1) + 1 - p_2)$ (Table 4.1). Combining these expressions results in $n_3 = n_2 + p_2 - p_1$, which satisfies the condition.

Thus far it has been shown that, so long as a system remains well-conditioned, reducing the word-lengths of signals which are the output of

nodes of type GAIN, ADD, or DELAY, cannot lead to an increase in the observed error variance at any primary output. The same cannot be said for FORK nodes, as will be demonstrated below.

The source of the difference between FORK nodes and other nodes can be illustrated with the help of Fig. 4.13. The transfer function from each FORK output to each primary system output may be distinct, unrelated, and arbitrary rational functions in z^{-1}. Recall from Section 4.1.2 that the chained model shown in the lower half of Fig. 4.13 is used for noise estimation. This means that if the order of the chained quantizations is changed due to a change in any of the word-lengths $n_2, n_3, \ldots, n_k$, new transfer functions may result, leading to different noise performance and possible non-convexity.

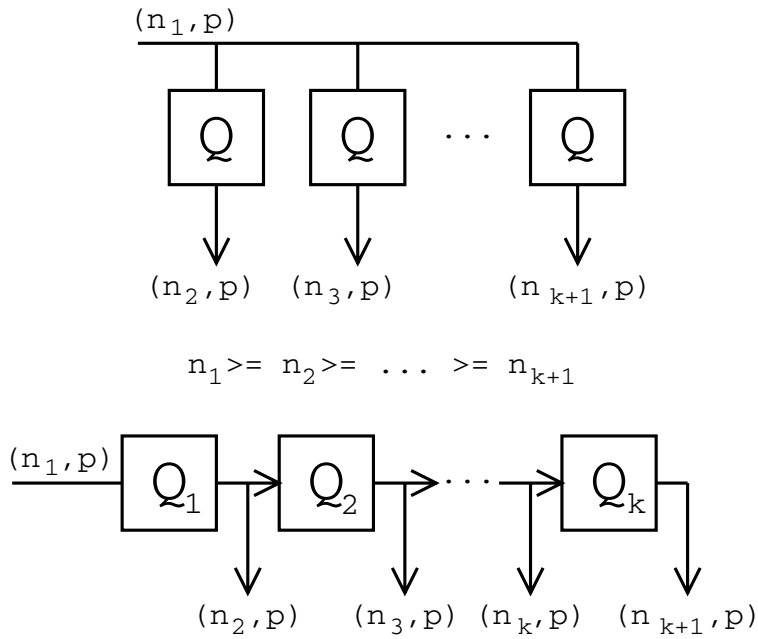


Fig. 4.13. Error estimation for FORK nodes

Claim. A computation graph containing a 2-way FORK node may exhibit error behaviour that is not monotonic in the word-length vector.

Proof:

Consider the FORK shown in Fig. 4.14(a). The transfer function from each quantizer injection input to a particular output is shown underneath the appropriate quantizer. From the uppermost to the lower-most diagram, the word-length of one fork output has been changed from n_2 to $n_2 - 1$ to $n_2 - 2$. The second of these reductions has caused a reversal in the order of the cas-

caded model. Performing an error analysis, the output errors in the uppermost to lower-most cases are given by (4.11–4.13), respectively. The difference between (4.12) and (4.11) is given in (4.14) and the difference between (4.13) and (4.12) is given in (4.15). The second of these two differences, (4.15) is clearly positive. The first of these two differences may be positive or negative, depending on the transfer functions involved. Thus non-monotonicity may result, however non-convexity cannot result from such an arrangement.

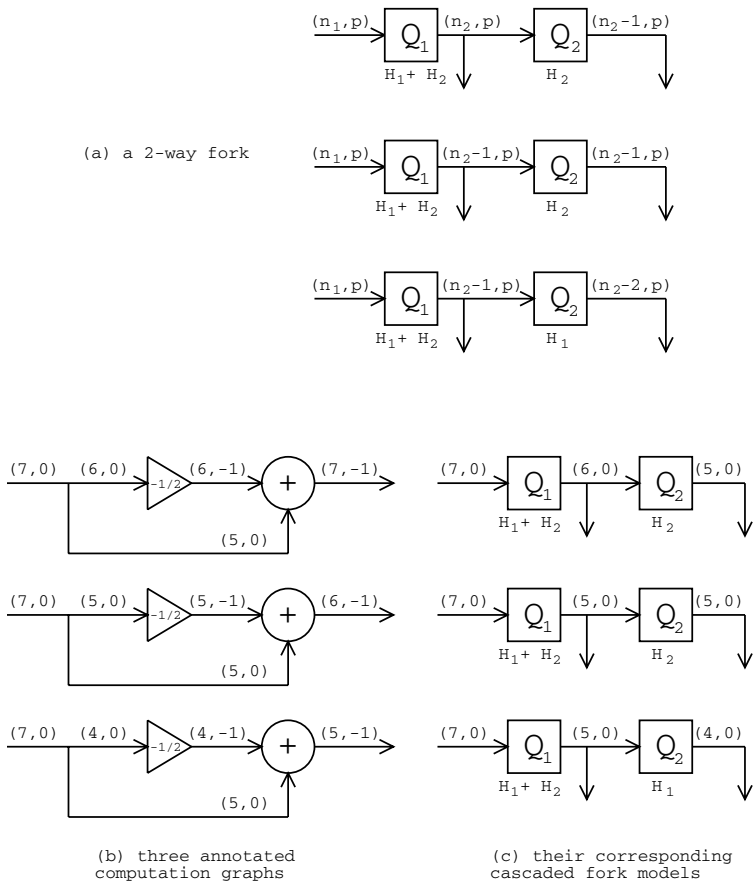


Fig. 4.14. Non-monotonic error behaviour in a computation graph

$$2^{2p}\left[(2^{-2n_2}-2^{-2n_1})L_2^2\left\{\sum_{i=1}^{2}H_i(z)\right\}+3\cdot 2^{-2n_2}L_2^2\{H_2(z)\}\right] \quad (4.11)$$

$$2^{2p}(4 \cdot 2^{-2n_2} - 2^{-2n_1})L_2^2 \left\{ \sum_{i=1}^{2} H_i(z) \right\} \quad (4.12)$$

$$2^{2p} \left[(4 \cdot 2^{-2n_2} - 2^{-2n_1})L_2^2 \left\{ \sum_{i=1}^{2} H_i(z) \right\} + 12 \cdot 2^{-2n_2} L_2^2\{H_1(z)\} \right] \quad (4.13)$$

$$3 \cdot 2^{-2(p-n_2)} \left[L_2^2 \left\{ \sum_{i=1}^{2} H_i(z) \right\} - L_2^2\{H_2(z)\} \right] \quad (4.14)$$

$$12 \cdot 2^{2(p-n_2)} L_2^2\{H_1(z)\} \quad (4.15)$$

□

Example 4.5. As an example, consider the annotated computation graph shown in Fig. 4.14(b). Going from the top of the figure to the bottom, the word-length immediately preceding the multiplier is reduced by one bit each time. This 2-way FORK is the only error source in this system. Shown in Fig. 4.14(c) is the FORK model used for error estimation, including the transfer functions $H_1(z)$, $H_2(z)$ through the different paths. $H_1(z) = -\frac{1}{2}$, and corresponds to the path through the GAIN block. $H_2(z) = 1$, and corresponds to the other path. In the uppermost system, the error injected by Q_1 and Q_2 have variance $2^{-12} - 2^{-14}$ and $2^{-10} - 2^{-12}$, respectively. Applying L_2 scaling predicts an output error variance of $2^{-14} - 2^{-16} + 2^{-10} - 2^{-12} = 51 \cdot 2^{-16}$. For the system in the centre of Fig. 4.14(b), the errors injected by Q_1 and Q_2 have variance $2^{-10} - 2^{-14}$ and 0, respectively. Applying L_2 scaling predicts and output error variance of $2^{-12} - 2^{-16} = 15 \cdot 2^{-16}$, a lower overall variance. Reducing the word-length once again, considering the lowermost system, the injected error variances at Q_1 and Q_2 are $2^{-10} - 2^{-14}$ and $2^{-8} - 2^{-10}$, respectively. Applying L_2 scaling results in an output error variance of $2^{-12} - 2^{-16} + 2^{-10} - 2^{-12} = 63 \cdot 2^{-16}$, a increase over the previous value. So the computation graph illustrated in Fig. 4.14(b) has the property that the error variance at the system output is not monotonic in the word-length of each signal. □

Such systems can arise in practice, for example in the parallel second order section implementation of a large IIR filter.

Non-monotonicity has been illustrated, however the constraint space of the system in Fig. 4.14 is still convex for all possible user-defined error specifications. For systems incorporating a 3-way FORK, non-convexity may arise.

Claim. An computation graph containing a 3-way FORK node may exhibit error behaviour which is non-convex in the word-length vector.

Proof:

Consider the FORK shown in Fig. 4.15(a). Once more, from the uppermost to the lower-most diagram, the word-length of one fork output has been changed

from n_2 to $n_2 - 1$ to $n_2 - 2$. Performing an error analysis, the output errors in the uppermost to lower-most cases are given by (4.16–4.18), respectively. The difference between (4.17) and (4.16) is given in (4.19) and the difference between (4.18) and (4.17) is given in (4.20). Either of these differences may be positive or negative, depending on the transfer functions involved. Thus non-convexity may result in this case.

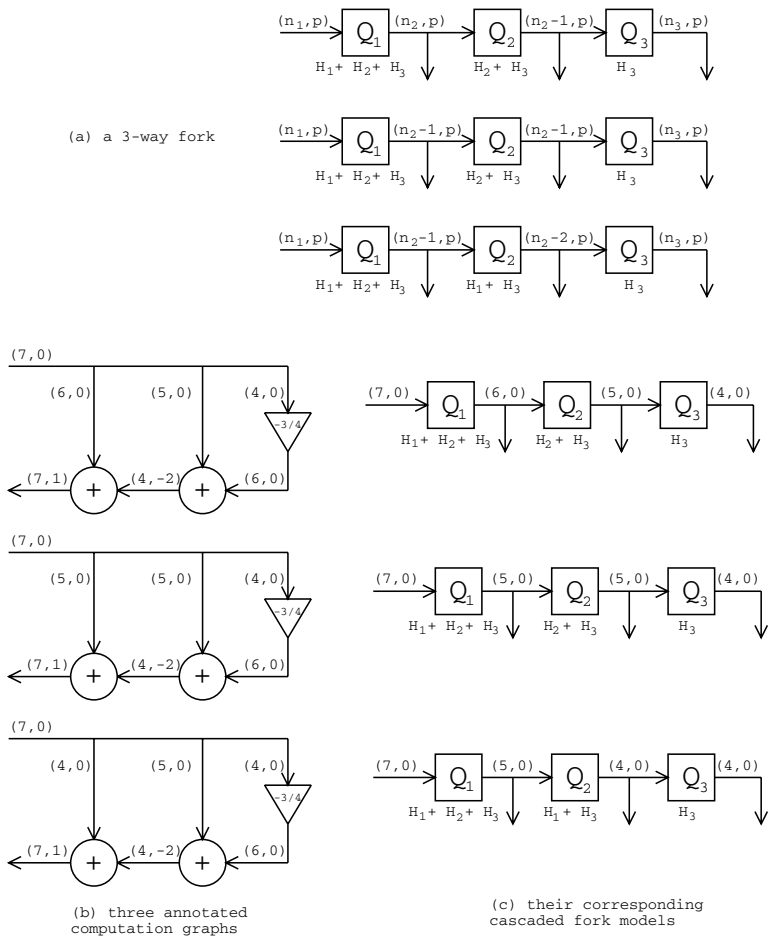


Fig. 4.15. Non-convex error behaviour in a computation graph

$$2^{2p}\Big[(2^{-2n_2}-2^{-2n_1})L_2^2\left\{\sum_{i=1}^{3}H_i(z)\right\}+3\cdot 2^{-2n_2}L_2^2\left\{\sum_{i=2}^{3}H_i(z)\right\}+ \\ (2^{-2n_3}-4\cdot 2^{-2n_2})L_2^2\{H_3(z)\}\Big] \tag{4.16}$$

$$2^{2p}\left[(4\cdot 2^{-2n_2}-2^{-2n_1})L_2^2\left\{\sum_{i=1}^{3}H_i(z)\right\}+(2^{-2n_3}-4\cdot 2^{-2n_2})L_2^2\{H_3(z)\}\right] \tag{4.17}$$

$$\begin{aligned}&2^{2p}\Bigg[(4\cdot 2^{-2n_2}-2^{-2n_1})L_2^2\left\{\sum_{i=1}^{3}H_i(z)\right\}+12\cdot 2^{-n_2}L_2^2\{H_1(z)+H_3(z)\}+\\&(2^{-2n_3}-16\cdot 2^{-2n_2})L_2^2\{H_3(z)\}\Bigg]\end{aligned} \tag{4.18}$$

$$3\cdot 2^{2(p-n_2)}\left[L_2^2\left\{\sum_{i=1}^{3}H_i(z)\right\}-L_2^2\left\{\sum_{i=2}^{3}H_i(z)\right\}\right] \tag{4.19}$$

$$12\cdot 2^{2(p-n_2)}\left[L_2^2\{H_1(z)+H_3(z)\}-L_2^2\{H_3(z)\}\right] \tag{4.20}$$

□

Example 4.6. As an example, consider the annotated computation graphs shown in Fig. 4.15(b). In the uppermost graph, the errors injected by quantizers Q_1, Q_2 and Q_3 have variance $2^{-12}-2^{-14}$, $2^{-10}-2^{-12}$ and $2^{-8}-2^{-10}$, respectively. Applying L_2 scaling results in a predicted error variance of $25\cdot(2^{-16}-2^{-18})+2^{-14}-2^{-16}+9\cdot(2^{-12}-2^{-14})=519\cdot 2^{-18}$. For the system in the centre of Fig. 4.15(b), the errors injected by quantizers Q_1, Q_2 and Q_3 have variance $2^{-10}-2^{-14}$, 0, and $2^{-8}-2^{-10}$, respectively. Applying L_2 scaling results in a predicted error variance of $25\cdot(2^{-14}-2^{-18})+9\cdot(2^{-12}-2^{-14})=807\cdot 2^{-18}$, an increase on the previous error variance. Finally considering the lower-most system in Fig. 4.15(b), the errors injected by quantizers Q_1, Q_2 and Q_3 have variance $2^{-10}-2^{-14}$, $2^{-8}-2^{-10}$ and 0, respectively. Applying L_2 scaling results in a predicted error variance of $25\cdot(2^{-14}-2^{-18})+2^{-12}-2^{-14}=423\cdot 2^{-18}$, a reduction over the previous error variance. Thus if a user-specified constraint on the output noise power were set between $519\cdot 2^{-18}$ and $807\cdot 2^{-18}$, then the uppermost and lower-most structures would be feasible but the centre structure would be infeasible. □

It has been shown that the noise model derived in Section 4.1.2 leads to a constraint space which, under well defined conditions, may be non-convex in the word-length optimization variables. It should be noted that this property is not simply an artifact of the noise model, but has been observed in practice using bit-true simulation. This non-convexity makes the constraint space a harder space to search for optimum solutions [Fle81].

4.4 Optimization Strategy 1: Heuristic Search

Since the word-length optimization problem is NP-hard [CW01], a heuristic approach has been developed to find feasible word-length vectors having small,

though not necessarily optimal, area consumption. The heuristic algorithm used is shown in Algorithm Word-LengthFalling. After performing an ℓ_1 scaling, the algorithm determines the minimum uniform word-length satisfying all error constraints. The design at this stage corresponds to a standard uniform word-length design with implicit power-of-two scaling, such as may be used for an optimized uniprocessor-based implementation. Each word-length is then scaled up by a factor $k > 1$, which represents a bound on the largest value that any word-length in the final design may reach. In the Synoptix implementation [CCL00a, CCL01b], $k = 2$ has been used. At this point the structure may be ill-conditioned, requiring reduction to a well-conditioned structure, as described in Section 4.1.1.

The resulting well-conditioned structure forms a starting point from which one signal word-length is reduced by one bit on each iteration. The signal word-length to reduce is decided in each iteration by reducing each word-length in turn until it violates an output noise constraint. At this point there is likely to have been some pay-off in reduced area, and the signal whose word-length reduction provided the largest pay-off is chosen. Each signal's word-length is explored using a binary search.

Although Algorithm Word-LengthFalling is a greedy algorithm, both the constraints and the objective function play a role in determining the direction of movement towards the solution. As a result, this algorithm is less dependent on local information than a pure steepest-descent search.

Algorithm 4.2
Algorithm Word-LengthFalling
Input: A Computation Graph $G(V, S)$
Output: An optimized annotated computation graph $G'(V, S, A)$, with $A = (\mathbf{n}, \mathbf{p})$
begin
Let the elements of S be denoted as $S = \{j_1, j_2, \ldots j_{|S|}\}$
Determine $\mathbf{p}$ through ℓ_1 scaling
Determine u, the minimum uniform word-length satisfying (4.10) with $\mathbf{n} = u \cdot \mathbf{1}$
Set $\mathbf{n} \leftarrow ku \cdot \mathbf{1}$
do
Condition the graph $G'(V, S, A)$
Set currentcost $\leftarrow A_G(\mathbf{n}, \mathbf{p})$
foreach signal $j_i \in S$ **do**
Set bestmin $\leftarrow$ currentcost
Determine $w \in \{2, \ldots, n_{j_i}\}$, if such a w exists, such that (4.10) is satisfied for annotation $([n_{j_1} \ldots n_{j_{i-1}}\ w\ n_{j_{i+1}} \ldots n_{j_{|S|}}]^T, \mathbf{p})$ but not satisfied for annotation $([n_{j_1} \ldots n_{j_{i-1}}\ (w-1)\ n_{j_{i+1}} \ldots n_{j_{|S|}}]^T, \mathbf{p})$
If such a w exists, set minval $\leftarrow A_G([n_{j_1} \ldots n_{j_{i-1}}\ w\ n_{j_{i+1}} \ldots n_{j_{|S|}}]^T, \mathbf{p})$
If no such w exists, set minval $\leftarrow A_G([n_{j_1} \ldots n_{j_{i-1}}\ 1\ n_{j_{i+1}} \ldots n_{j_{|S|}}]^T, \mathbf{p})$
if minval $<$ bestmin **do**

```
        Set bestsig ← j, bestmin ← minval
      end if
    end foreach
    if bestmin < currentcost
      n_bestsig ← n_bestsig − 1
  while bestmin < currentcost
end
```

Algorithm Word-LengthFalling will, in general, provide better results under a convex constraint space. However the non-convexities described in Section 4.3.1 should not affect its operation too severely. The binary search mechanism will result in 'jumping over' infeasible portions of non-convex constraint space in some cases, and will remain stuck on one side of an infeasible region in other cases. In either case, since the word-lengths are only reduced by one bit each time, the next iteration will be facing the non-convexity from a different direction. Non-convexity can manifest itself in intermediate word-length vectors having infeasible error properties. However the final word-length vector will always be feasible, as moves are only ever performed in a direction leading to a feasible solution and the cost metric $A_G(\mathbf{n}, \mathbf{p})$ is monotonic in $\mathbf{n}$.

The overwhelming proportion of execution time in Algorithm Word-LengthFalling is spent in the iterative refinement of the initial scaled-up uniform word-length solution. The average-case execution time will be discussed in Section 4.6.1.

4.5 Optimization Strategy 2: Optimum Solutions

The contribution of section is to present a technique for *optimum* word-length allocation, for the case where the DSP algorithm to be synthesized is a linear, time-invariant (LTI) system [Mit98].

The Mixed Integer Linear Programming (MILP) technique described in this section has been applied to several small benchmark circuits, and the results compared to the heuristic presented in Section 4.4. Modelling as a MILP permits the use of industrial-strength MILP solvers such as BonsaiG [Haf]. Although MILP solution time can render the synthesis of large circuits intractable, optimal results even on small circuits are valuable as benchmarks with which to compare heuristic optimization procedures.

The proposed MILP model contains several variables, which may be classified as: integer signal word-lengths and signal word-lengths before quantization, binary auxiliary signal word-lengths and auxiliary signal word-lengths before quantization, binary decision variables, real adder costs, and real fork node errors.

Note that only adders, gains, and delays cost area resources (forks are considered free). However adders have an inherently complex area model and

thus while gains and delays are included directly in the objective function, the cost of each adder $v \in V_A$ is represented by a distinct variable A_v.

We are now in a position to formulate an area-based objective function for the MILP model (4.21), where $\text{CW}(v)$ represents the coefficient word-length of gain node v.

$$\begin{aligned}\text{min: } \sum_{v \in V_A} A_v + \sum_{v \in V_G} \Big\{ &(k_3 \text{CW}(v) + k_4) n_{\text{in}(v)} - \\ &k_4 n_{\text{out}(v)} + (k_3 + k_4) \text{CW}(v) \Big\} + \sum_{v \in V_D} k_5 n_{\text{in}(v)} \end{aligned} \tag{4.21}$$

Constraints on quantization error propagation are much harder to cast in linear form due to the exponentiation, shown in Section 4.1. In order to overcome this nonlinearity, we propose to use an additional binary variables, $\bar{n}$, one for each possible word-length that a signal could take. This is expressed in (4.22), and (4.23) ensures that each signal can only have a single word-length value. Here $\setminus$ is used to denote set subtraction. Note that in order to apply this technique, it is necessary to know upper-bound word-lengths $\hat{n}_s$ for each $s \in S$. Techniques to derive these will be discussed in Section 4.5.1. Note that signals which drive fork nodes are not considered in this way, as fork node error models are considered separately (see Section 4.5.3).

$$\forall s \in S \setminus \text{pred}(V_F), n_s - \sum_{b=1}^{\hat{n}_s} b \cdot \bar{n}_{s,b} = 0 \tag{4.22}$$

$$\forall s \in S \setminus \text{pred}(V_F), \sum_{b=1}^{\hat{n}_s} \bar{n}_{s,b} = 1 \tag{4.23}$$

Using these binary variables it is possible to re-cast expressions of the form 2^{-2n_j}, which appear in error constraints (see Section 4.1), into linear form as $\sum_{b=1}^{\hat{n}_s} 2^{-2b} \bar{n}_{j,b}$. Similarly it is necessary to linearize the exponentials in word-lengths before quantization (4.24)–(4.25).

$$\forall s \in S \setminus \text{pred}(V_F) \setminus \text{succ}(V_F), n_s^q - \sum_{b=1}^{\hat{n}_s^q} b \bar{n}_{s,b}^q = 0 \tag{4.24}$$

$$\forall s \in S \setminus \text{pred}(V_F) \setminus \text{succ}(V_F), \sum_{b=1}^{\hat{n}_s^q} \bar{n}_{s,b}^q = 1 \tag{4.25}$$

For each system output, we propose to use an error constraint of the form given in (4.26). $\mathcal{E}$ represents the user-defined bound on the error power at the system output, and hence on the signal quality. Note that in this section

we only consider single-output systems, for simplicity of explanation, however the technique is easily extensible to multiple-input, multiple-output (MIMO) systems.

$$\sum_{v \in V_F} E_v + \sum_{s \in S \setminus \mathrm{pred}(V_F) \setminus \mathrm{succ}(V_F) \setminus \mathrm{succ}(V_I)} 2^{2p_s} L_2^2\{H_s(z)\}\Big(\sum_{b=1}^{\hat{n}_s} 2^{-2b}\bar{n}_{s,b} - \sum_{b=1}^{\hat{n}_s^q} 2^{-2b}\bar{n}_{s,b}^q\Big) + \sum_{s \in \mathrm{succ}(V_I)} 2^{2p_s} L_2^2\{H_s(z)\}\Big(\sum_{b=1}^{\hat{n}_s} 2^{-2b}\bar{n}_{s,b} - 2^{-2n_s^q}\Big) \leq 12\mathcal{E} \tag{4.26}$$

Note that those signals driven by system inputs are considered separately, since there is no need for Boolean variables representing the pre-quantization word-length of a variable, as this parameter is defined by the system environment. Place-holders E_v are used for the contribution from fork nodes; these will be defined by separate constraints in Section 4.5.3.

4.5.1 Word-Length Bounds

Upper bounds on the word-length of each signal, before and after quantization, are required by the MILP model in order to have a bounded number of binary variables corresponding to the possible word-lengths of a signal.

Our bounding procedure proceeds in three stages: perform a heuristic word-length optimization on the computation graph (Section 4.4); use the resulting area as an upper-bound on the area of each gain block within the system, and hence on the input word-length of each gain block; 'condition' the graph, following the procedure described in Section 4.1.1. The intuition is that typically the bulk of the area consumed in a DSP implementation comes from multipliers. Thus reasonable upper-bounds are achievable by ensuring that the cost of each single multiplier cannot be greater than the heuristically achieved cost for the entire implementation.

Of course this only bounds the word-length of signals which drive gain blocks. In addition, the word-length of signals driven by primary inputs is bounded by the externally-defined precision of these inputs. Together this information can be propagated through the computation graph, resulting in upper bounds for all signals under the condition that any closed loop must contain a gain block.

In the remainder of this paper, we denote by $\hat{n}_s$ the so-derived upper bound on the word-length of signal $s \in S$ and by $\hat{n}_s^q$ the upper bound on the word-length of the same signal before LSB truncation.

4.5.2 Adders

It is necessary to express the area model of Section 4.2 as a set of constraints in the MILP. Also a set of constraints describing how the word-length at an adder output varies with the input word-lengths is required.

In the objective function, the area for each adder $v \in V_A$ was modelled by a single variable A_v. It will be demonstrated in this section how this area can be expressed in linear form.

Let us define β_v for an adder $v \in V_A$ with input signals a and b (4.27), where the inputs 'a' and 'b' are chosen to match with Fig. 4.1 (reproduced with more detail as Fig. 4.16) so that it is b which needs to be left-shifted for alignment purposes. s_v is also illustrated in Fig. 4.16, and models the number of bits by which input b must be shifted.

$$\beta_v = \max(n_a - s_v, n_b) \tag{4.27}$$

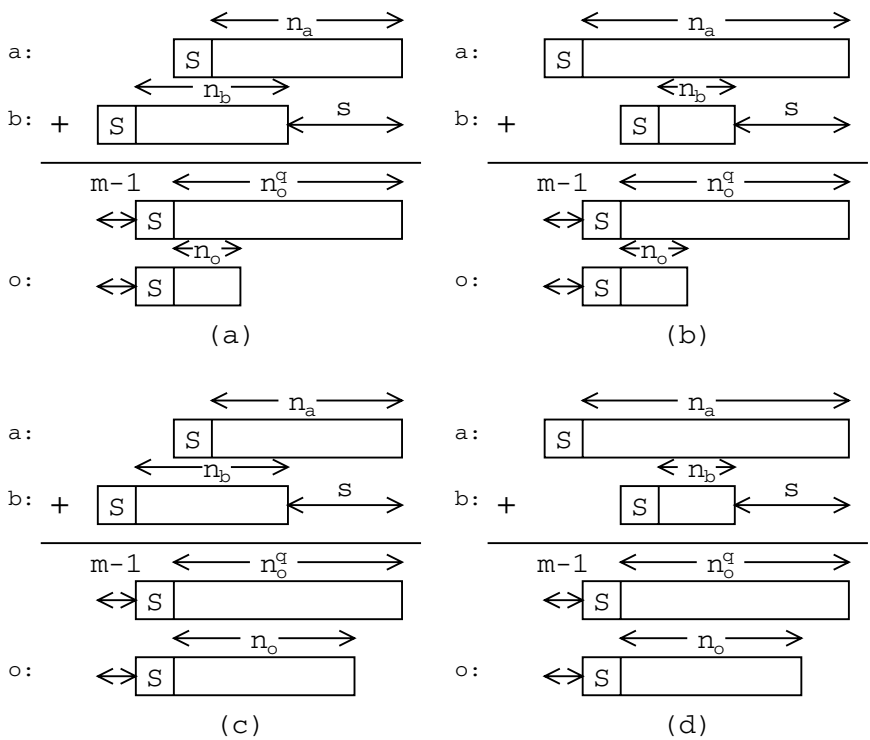


Fig. 4.16. Multiple Word-Length Adder Types

We may then express the area of an adder as (4.28). Signal o is the output signal for the adder and m_v models the number of MSBs of the addition

which are known through scaling to contain no information, as described in Chapter 3 and illustrated in Fig. 4.1. This value is independent of the word-lengths, and for an adder can be expressed as $m_v = \max(p_a, p_b) + 1 - p_o$.

$$A_v = \begin{cases} k_1(n_o + 1) + k_2\left[\beta - m_v - n_o + 1\right], \\ \qquad n_o + m_v \leq \beta + 1 \\ k_1\left[\beta - m_v + 2\right], \\ \qquad \text{otherwise} \end{cases} \tag{4.28}$$

The non-linearities due to the max operator in (4.27) and the decision in (4.28) must be linearized for the MILP model. This is achieved through the introduction of four binary decision variables δ_{v1}, δ_{v2}, δ_{v3} and δ_{v4} for each adder $v \in V_A$.

For the remainder of this section, we consider a general adder with inputs i and j and output o, to distinguish from the more specific case considered above, where input b was used to denote the left-shifted input to an adder. In order to model (4.27), if $p_j \leq p_i$ then (4.29)–(4.32) are included in the MILP. Otherwise (4.33)–(4.36) are included in the MILP. The right-hand side of each inequality consists of a trivial bound on the left-hand side, multiplied by a decision variable.

$$n_i - n_j + p_j - p_i < \delta_{v1}(\hat{n}_i + p_j - p_i) \tag{4.29}$$
$$\beta_v - n_j + p_j - p_i \geq (1 - \delta_{v1})(-\hat{n}_j - p_i + p_j) \tag{4.30}$$
$$n_i - n_j + p_j - p_i \geq \delta_{v2}(-\hat{n}_j + p_j - p_i) \tag{4.31}$$
$$\beta_v - n_i \geq (1 - \delta_{v2})(-\hat{n}_i) \tag{4.32}$$

$$n_j - n_i + p_i - p_j < \delta_{v1}(\hat{n}_j - p_j + p_i) \tag{4.33}$$
$$\beta_v - n_i + p_i - p_j \geq (1 - \delta_{v1})(1 - \hat{n}_i - p_j + p_i) \tag{4.34}$$
$$n_j - n_i + p_i - p_j \geq \delta_{v2}(-\hat{n}_i + p_i - p_j) \tag{4.35}$$
$$\beta_v - n_j \geq (1 - \delta_{v2})(-\hat{n}_j) \tag{4.36}$$

Note that β_v and α_v are only bounded from below by the constraints given. Inequalities are used in order to allow disjunctions and thus implications, for example selecting $\delta_{v1} = 0$ in (4.29) gives $n_i - n_j + p_j - p_i < 0$, whereas selecting $\delta_{v1} = 1$ gives $\beta_v - n_j + p_j - p_i \geq 0$. Allowing δ_{v1} as an optimization variable results in $n_i \geq n_j - p_j + p_i \Rightarrow \beta_v \geq n_j + p_j - p_i$. Equality of A_v is guaranteed through its positive coefficient in the objective function.

In order to model (4.28), (4.37)–(4.40) are included in the MILP. These terms model the choice in (4.28) as a pair of implications, in an identical manner to that described above.

$$n_o - \beta_v + m_v - 1 \geq \delta_{v3}(m_v - \hat{\beta}_v) \tag{4.37}$$

$$A_v + (k_2 - k_1)n_o - k_2\beta_v + k_2(m_v - 1) - k_1 \geq (1 - \delta_{v3})\left[(k_2 - k_1)\hat{n}_o - k_2\hat{\beta}_v + k_2(m_v - 1) - k_1\right] \tag{4.38}$$

$$n_o - \beta_v + m_v - 1 < \delta_{v4}(\hat{n}_o + m_v - 2) \tag{4.39}$$

$$A_v + k_1(m_v - \beta_v - 2) \geq (1 - \delta_{v4})k_1(m_v - \hat{\beta}_v - 2) \tag{4.40}$$

The pre-quantization output word-length of an adder with inputs i and j and output o is given by $n_o^q = \max(n_i - p_i, n_j - p_j) + p_o$. We may express this as (4.41)–(4.42), since before-quantization word-lengths only appear with negative coefficient in the error so the error constraints can be relied upon to reduce n_o^q to achieve equality.

$$n_o^q \geq n_i - p_i + p_o \tag{4.41}$$

$$n_o^q \geq n_j - p_j + p_o \tag{4.42}$$

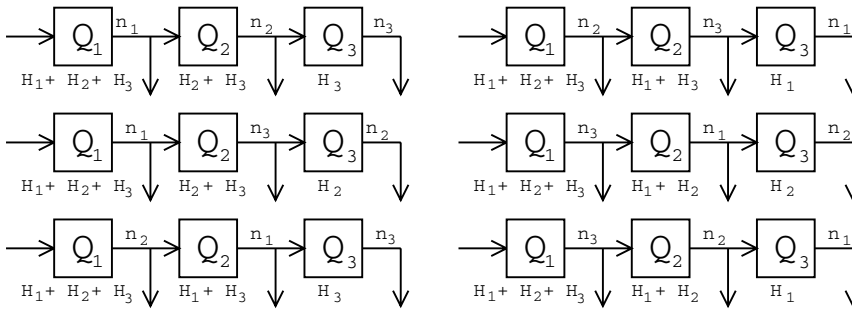
4.5.3 Forks

As demonstrated in Section 4.3.1, fork nodes can lead to unusual error behaviour due to the different possible orderings of word-length at their outputs, which are required in order to guarantee freedom from statistical correlation and hence an accurate error model. Fig. 4.17 illustrates the six different possible configurations of a 3-way fork with outputs n_1, n_2 and n_3. For example, the top left figure corresponds to $n_1 \geq n_2 \geq n_3$ and the bottom right to $n_3 \geq n_2 \geq n_1$. Each of the 'Q' blocks is a truncation of least-significant bits in a signal. The z-domain transfer function from the truncation error injected, to the system output, is shown underneath the relevant 'Q' block.

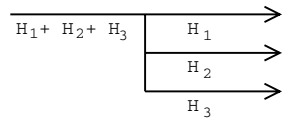
In order for the MILP to fully model this behaviour it is necessary to consider each of the possible orderings. Let σ_v be a w-tuple, representing an order $(a, b, \ldots, f)$ on a w-way fork node $v \in V_F$ with input signal i. Thus, for example, $\sigma_v(2)$ is the second largest signal width. We may express the error resulting from truncation of those signals driven by node v as (4.43), with one constraint per possible σ, a total of $w!$. Here $\wedge$ represents Boolean conjunction.

$$\begin{aligned} &\bigwedge_{r=1}^{w-1}(n_{\sigma_v(r)} \geq n_{\sigma_v(r+1)}) \Rightarrow \\ &E_v = 2^{2p_i}\left(\sum_{r=1}^{w-1} L_2^2\left\{\sum_{h=1}^{w-r} H_{\sigma_v(h)}\right\}(2^{-2n_{\sigma_v(r+1)}} \right. \\ &\left. -2^{-2n_{\sigma_v(r)}}) + L_2^2\left\{\sum_{h=1}^{w} H_{\sigma_v(h)}(2^{-2n_i^q} - 2^{-2n_w})\right\}\right) \end{aligned} \tag{4.43}$$

Applying DeMorgan's theorem and linearizing the resulting disjunction gives (4.44)–(4.48). Each exponential is then further linearized through the



(a) possible output permutations in a 3-way FORK



(b) the original FORK specification

Fig. 4.17. Possible output permutations in a 3-way fork

procedure described in Section 4.5. The ϵ and η variables in (4.44)–(4.48) are additional binary decision variables and the right-hand side of each inequality consists of a trivial bound on the left-hand side, multiplied by a decision variable. At least one inequality is non-trivial, a property ensured by (4.48).

$$n_{\sigma(1)} - n_{\sigma(2)} < \epsilon_{v\sigma(1),\sigma(2)} \hat{n}_{\sigma(1)} \tag{4.44}$$

$$n_{\sigma(2)} - n_{\sigma(3)} < \epsilon_{v\sigma(2),\sigma(3)} \hat{n}_{\sigma(2)} \tag{4.45}$$

$$\dots$$

$$n_{\sigma(w-1)} - n_{\sigma(w)} < \epsilon_{v\sigma(w-1),w} \hat{n}_{\sigma(w-1)} \tag{4.46}$$

$$E_v - 2^{2p_i} \left(\sum_{r=1}^{w-1} L_2^2 \left\{ \sum_{h=1}^{w-r} H_{\sigma(h)} \right\} (2^{-2n_{\sigma_{v(r+1)}}} - 2^{-2n_{\sigma_{v(r)}}}) + L_2^2 \left\{ \sum_{h=1}^{w} H_{\sigma(h)} \right\} (2^{-2n_{\sigma_{v(w)}}} - 2^{-2n_o^q}) \right) \geq -\eta_{v\sigma} 2^{2(p_i-1)} \sum_{r=0}^{w-1} L_2^2 \left\{ \sum_{h=1}^{w-r} H_{\sigma(h)} \right\} \tag{4.47}$$

$$\sum_{r=1}^{w-1} \epsilon_{v\sigma(r),\sigma(r+1)} + \eta_{v\sigma} \leq w - 1 \tag{4.48}$$

It is not necessary to explicitly consider quantization of the input signal to a fork node, since the above constraints use the pre-quantization word-length of the fork input n_i^q. It is necessary, however, to guarantee that the input signal provides enough word-length for the largest of its outputs (4.49).

$$\begin{aligned} n_i &\geq n_a \\ n_i &\geq n_b \\ &\dots \\ n_i &\geq n_f \end{aligned} \tag{4.49}$$

4.5.4 Gains and Delays

In contrast to adders and fork nodes, gain nodes are straight-forward. The area of a gain node has already been modelled in the objective function (Section 4.5). The only remaining constraint required is to model the pre-quantization output word-length of a gain $v \in V_G$ with input signal a, output signal o and coefficient of word-length $\text{CW}(v)$ and scaling $\text{SC}(v)$. This constraint is already in linear form (4.50).

$$n_o^q = n_a + \text{CW}(v) - p_a - \text{SC}(v) + p_o \tag{4.50}$$

Delay nodes also have a simple relationship between their input word-length and their output word-length before quantization, shown in (4.51) for the case of a delay node with input i and output o.

$$n_o^q = n_i \tag{4.51}$$

4.5.5 MILP Summary

A MILP model for the word-length optimization problem has been proposed. It remains to quantify the number of variables (4.52) and constraints (4.53) present in the model. Note that the number of constraints given does not include integrality constraints, the unit upper bounds on Boolean variables, or the trivial fork constraints in (4.49) which do not form part of the optimization problem.

$$\begin{aligned} \text{vars} = &\sum_{s \in S \setminus \text{pred}(V_F)} (\hat{n}_s + 1) + \\ &\sum_{s \in S \setminus \text{pred}(V_F) \setminus \text{succ}(V_F)} (\hat{n}_s^q + 1) + \\ &|V_F| + \\ &6|V_A| + \\ &\sum_{v \in V_F} \text{od}(v)(\text{od}(v) - 1)\left\{1 + (\text{od}(v) - 2)!\right\} \end{aligned} \tag{4.52}$$

$$\begin{aligned}\text{cons} = {} & 2|S \setminus \text{pred}(V_F)|+ \\ & 2|S \setminus \text{pred}(V_F) \setminus \text{succ}(V_F)|+ \\ & 1+ \\ & 10|V_A|+ \\ & \sum_{v \in V_F} \text{od}(v)(\text{od}(v)-1)\left\{1+2(\text{od}(v)-2)!\right\}+ \\ & |V_G|+|V_D|\end{aligned} \tag{4.53}$$

It can be seen that so long as the number of large-fanout fork nodes are limited, the number of constraints in the MILP model grows approximately linearly in the number of nodes and signals. Under the same conditions the number of variables can grow up to quadratically with the number of signals because the upper bounds on each signal word-length will vary approximately linearly with the number of large area-consuming nodes. Both parameters are dominated by any large-fanout fork nodes, since the number of η variables and their associated constraints grow combinatorially with fanout.

4.6 Some Results

Synoptix, a complete synthesis system incorporating the algorithms in this chapter, has been developed for implementation of multiple word-length systems in FPGAs. The input to Synoptix is a Simulink [SIM] block diagram, and the output is a structural description in VHDL [IEE99] or AHDL [MAX]. FPGA vendor tools are then used to perform the low-level logic synthesis, placement, and routing of the designs. The design-flow for implementation on the Sonic platform [HSCL00] is illustrated in Fig. 4.18, with the Sonic-specific parts shaded.

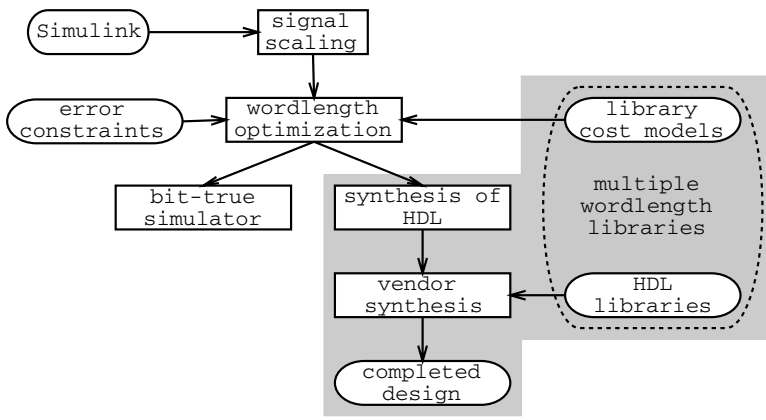


Fig. 4.18. Synoptix design flow

The system has been tested on several benchmark circuits, including finite impulse response (FIR) and infinite impulse response (IIR) filters, a discrete cosine transform (DCT), a polyphase filter bank (PFB), and an RGB to YCrCb convertor. It is important to note that all results presented in this section are measured from placed and routed designs, rather than estimated from the library cost estimation procedure described in Section 4.2.

4.6.1 Linear Time-Invariant Systems

Mixed Integer Linear Programming Results

Fig. 4.19 illustrates area-error tradeoff curves for both a second and a third order linear phase FIR filter [Mit98]. For the second order filter, results for both 4-bit and 8-bit inputs are given. For the third order filter, only results for a 4-bit input have been obtained. Three curves are present in each plot: the optimum uniform word-length implementation, the heuristically derived multiple word-length implementation from (Section 4.4), and the optimum multiple word-length implementation achieved by solving the MILP presented in Section 4.5.

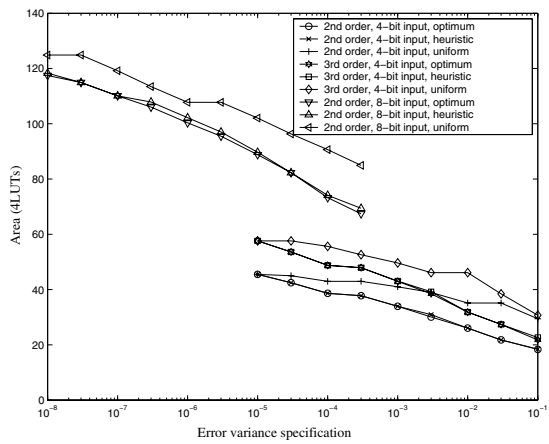


Fig. 4.19. Area / Error tradeoffs compared for a 2nd and 3rd order FIR filter

The results clearly illustrate the high-quality solutions achievable by the heuristic solution, averaging only 0.7% with a maximum of 3.9% worse than the optimum result.

An optimum word-length allocation for an RGB to YCrCb convertor with 4-bit inputs has also been performed. This result shows an optimal cost of 78.61 LUTs, equal to the result achieved by the heuristic.

Fig. 4.20 illustrates the structure [Eva] and optimum word-lengths of the RGB to YCrCb converter for 4-bit inputs (of range ±112), 4-bit coefficients, and with an error-free Y, whereas a bounded error power of up to 10^{-2} has been allowed for Cr and Cb.

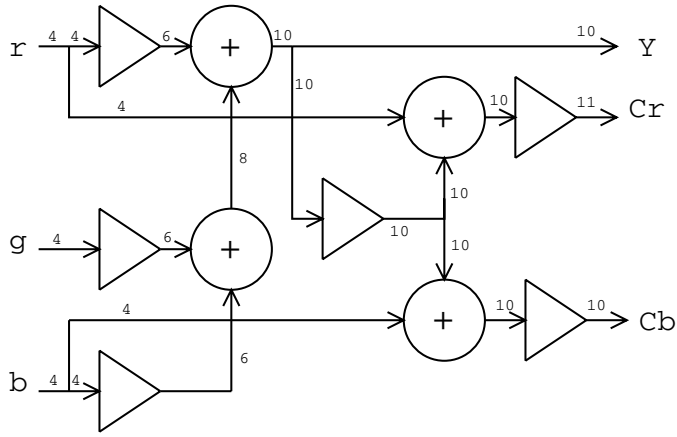


Fig. 4.20. Optimal word-length allocations for the ITU RGB to YCrYb converter

The BonsaiG MILP solver [Haf] was used to solve the MILP models: execution time ranged from 2 seconds to 6 minutes on an AMD Athlon 1.2 GHz with 512 MB RAM. This compares to less than 0.2 second for the heuristic solutions on the same machine. Limits on the scale of the MILP solvable are due to both excessive run-time and numerical instabilities in the MILP solver.

Heuristic Results

Shown in Fig. 4.21 is a graph of placed-and-routed resource usage (measured in Altera Flex10k logic cells [Alt98]) against specified error variance. This plot is representative in terms of the general shape of the plots obtained for all designs. The benchmark is a simple second order (biquadratic) IIR digital filter. Both the multiple word-length design and the optimized uniform word-length structure are shown. The plot of area for a uniform word-length decreases in steep steps. This is because there is a sudden change when the next-lowest word-length becomes feasible with respect to the error constraints. This is not the case for the optimized multiple word-length structures, since there are many more optimization variables and hence many different error powers are achievable. In addition, the heuristic line lies consistently below the uniform line (by 2% to 15%), showing a consistent area saving for this design.

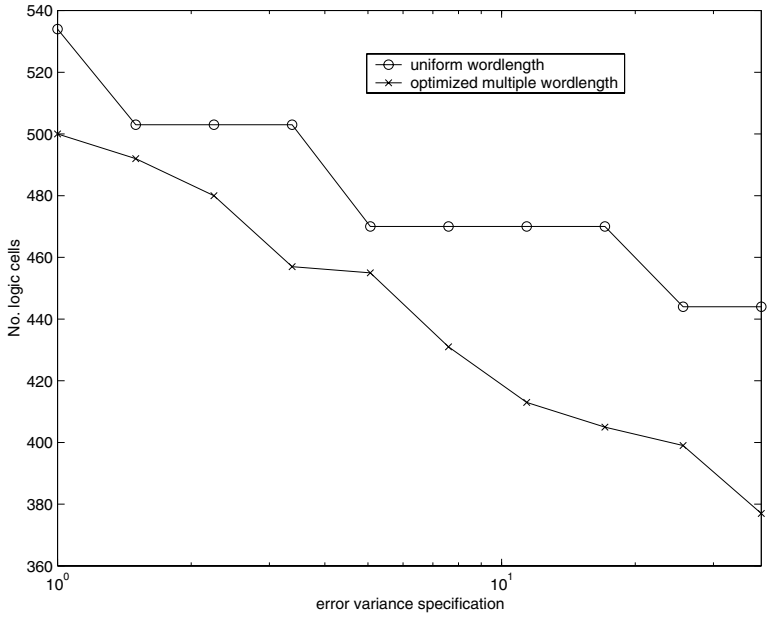


Fig. 4.21. Circuit area against specified error power for an IIR biquadratic filter

Table 4.2 illustrates some further results from larger benchmark circuits. Both the number of logic cells (LCs) and maximum clock frequency are reported. Each of these results corresponds to a single point on the area-error tradeoff curve for the circuit, and have been placed and routed in an Altera Flex10k70RC240-3 device (as used in the Sonic [HSCL00] platform) except where otherwise stated. The FIR filter is a 126-tap linear-phase low-pass Direct Form II transposed [Mit98] structure, suggested by [LKHP97] as a representative DSP design. The DCT is an 8-point, 1-dimensional decimation in time structure from [Par99] which has also been suggested as a benchmark by [LKHP97]. Two versions of this benchmark have been synthesized, one (DCT^1) with equal error tolerance on all outputs, and the other (DCT^2) with required signal-to-noise ratio (SNR) reducing by 3dB per DCT coefficient, so that low frequency coefficients are less noisy than high-frequency ones. The IIR filter is of 4th order, as used by [KKS00] and is of interest since it has a recursive (feedback) structure. The polyphase filter bank (PFB) is the design given in [FGL01] for evaluation of the Streams-C compiler. The RGB to YCrCb convertor is of the form suggested by the ITU [Eva], and allows some quantization error in the Cr and Cb outputs whereas the Y output is guaranteed to be error-free. This design is of particular interest since the multiple word-length approach can clearly be used to customize the datapath in order to achieve these differential specifications. Each of these circuits has

been synthesized twice, once using an optimal uniform word-length structure, and once using the multiple word-length structure generated by the Synoptix tool. The DCT designs have been synthesized on a device with a larger number of I/O pins, due to the I/O-limited nature of the designs, whereas the FIR filter has been synthesized on a device with a significantly larger logic capacity.

Table 4.2. Lossy synthesis results

Design	Uniform wl			
	Area (#LCs)		f_{clk} (MHz)	width (bits)
FIR†	6125		29.15	16
DCT^{1*}	1394		12.95	13
DCT^{2*}	1367		13.03	12
IIR	701		9.57	12
PFB	321		30.03	15
RGB–YCrCb	438		11.58	18
Design	Multiple wl			
	Area (#LCs)	% improvement	f_{clk} (MHz)	% improvement
FIR†	3356	42.5%	36.23	2.5%
DCT^{1*}	1311	6.0%	13.67	5.6%
DCT^{2*}	1164	14.9%	13.53	3.8%
IIR	623	11.1%	9.32	-2.6%
PFB	273	15.0%	31.34	4.4%
RGB–YCrCb	272	37.9%	16.15	39.5%

* implemented on Flex10k70GC503-3
† implemented on Flex10k200SRC240-1

It should be noted that even for the uniform word-length structures, Synoptix has been used to automatically insert power-of-two scaling [Jac70], which is good practice in DSP design. Also note that for both uniform and multiple word-length structures, these circuits represent a completely unpipelined implementation of the specification, in order to aid direct comparison of maximum clock rate f_{clk} reported.

Table 4.2 illustrates that area reductions of between 6% and 45% (mean 22%) have been achieved by using the multiple-word-length synthesis approach described in this chapter. These area reductions have been accompanied by a speedup in maximum clock frequency between -3% and 39% (mean 12%), even though the estimated speed is not considered by the cost function used for optimization. Interestingly, the only benchmark to have been slowed down slightly as a result of the optimization is the IIR filter. This is due to the increase of some signal word-lengths on the critical path around the feedback loops in this filter. Importantly, the largest area reductions and speedups have occurred in both the FIR filter, which is the largest design shown, and the RGB to YCrCb convertor, which has a structure ideally suited to multiple

word-lengths since the error-free Y is calculated first, from which Cr and Cb are derived [Eva].

With the exception of the IIR filter, all benchmarks have also been synthesized with a specification of zero error. This is a degenerate case of the optimization procedure, corresponding to *lossless* synthesis, and has been performed to enable possible comparisons with other lossless approaches. (Note that the IIR filter cannot be synthesized in a lossless way due to the feedback.) Table 4.3 illustrates these results. The FIR result for area is only an estimate reported by the synthesis tools since the design was unable to be placed in the largest device supported by Altera MaxPlusII, and for the same reason there is no clock frequency result for this benchmark under lossless synthesis. The most important trend to be gleaned from these results is that the correctness preserving (lossless) approach to high-level synthesis of DSP structures from floating-point specifications is insufficient by itself to achieve results matching or improving on traditional DSP design techniques, when some output round-off error can be tolerated. For as long as output error is not considered as a design variable by high-level synthesis systems, design specification languages must explicitly consider word-length or sub-optimal designs will often result.

Table 4.3. Lossless synthesis results

Design	Multiple wl	
	Area (#LCs)	f_{clk} (MHz)
FIR†	11110	†
DCT*	1530	12.77
PFB	332	38.31
RGB–YCrCb	547	11.99
* implemented on Flex10k70GC503-3		
† design too large for Flex10k200 device		

Table 4.4 presents the execution times of an unoptimized C implementation of Algorithm Word-LengthFalling (without spectral constraints) running on a Pentium III 450MHz, for each of the benchmark circuits discussed in the previous section.

In order to provide an insight into the way in which algorithm execution time scales with the size of the synthesized system, Direct Form II FIR filters with between 6 and 74 taps have been generated and optimized using the heuristic from Section 4.4. Fig. 4.22 illustrates the results. Let n be the number of filter taps. The **do–while** refinement loop of the algorithm will be executed a number of times approximately proportional to n, since only one word-length will be reduced by one bit on each iteration. The **foreach** statement will be executed $|S|$ times, which is proportional to n for an FIR filter structure. Finally the error estimation phase execution time will be generally linear in n. Overall, the expected average-case execution time is proportional to n^3.

Table 4.4. Lossy synthesis execution times

Design	Time
FIR	42min 32.13sec
DCT[1]*	9.99sec
DCT[2]*	16.62sec
IIR	1.06sec
PFB	0.08sec
RGB–YCrCb	0.06sec

Examining Fig. 4.22 reveals an execution time that scales between n^3 and n^4. This is because the preceding analysis did not consider that each of these filters has been synthesized with the same error specification. The result is that filter order growth causes word-length growth. The uniform word-length necessary to satisfy the error constraints grows approximately as $\log n$. Taking this into account causes the **do**–**while** loop execution count to increase to $n \log n$ and the search for an appropriate w to be completed in $\log \log n$ attempts, leading to an average execution time model of $n^3 \log n \log \log n$ for FIR filters.

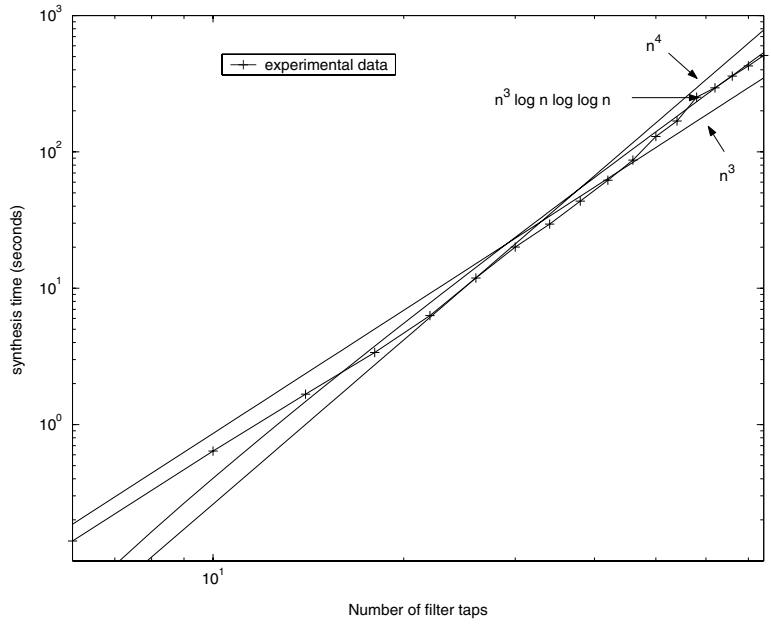


Fig. 4.22. Variation of execution time with number of filter taps

Noise Shaping

Thus far, all results presented in this chapter have concentrated on noise variance error specifications. In this section, some results from the spectral bounds approach described in Section 4.1.2 will be presented.

The simple block diagram of Fig. 2.2, reproduced in Fig. 4.23 with all signals labelled, may be used to demonstrate the 'noise shaping' capability of the multiple word-length optimization procedure. The transfer function from each signal to each primary output has been calculated (the matrix $\mathbf{T}(z)$ of Section 3.1.1), and a resulting normalized power spectrum from each of these transfer functions is illustrated in Fig. 4.24, where a normalized frequency of 1 corresponds to the Nyquist frequency. These power spectra correspond to 'spectral profiles' of the different paths that exist from each point within this filter's structure to the filter output. Each achievable output noise spectrum consists of a linear combination of these profiles. The Synoptix synthesis system maintains an efficient internal representation of the profiles and uses them to construct an estimated roundoff-noise spectrum for a given fixed-point implementation of the filter.

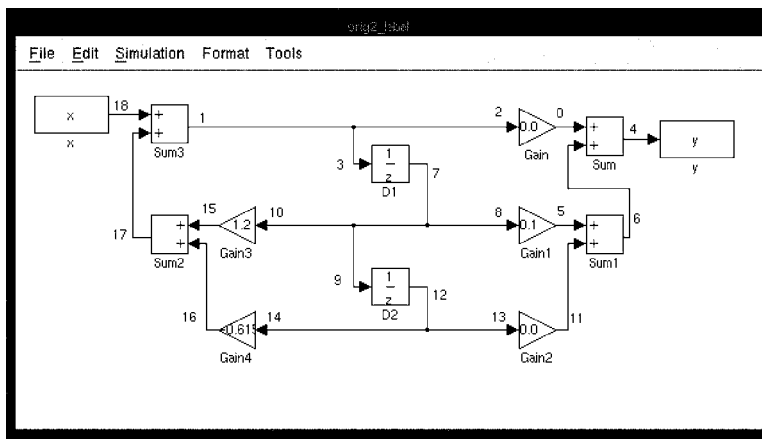


Fig. 4.23. A simple Simulink block diagram

Shown in Figs. 4.25(a,b) are two spectral noise specifications (upper curve) and the corresponding noise spectra of the optimized filters produced by Algorithm Word-LengthFalling (lower curve). Figs. 4.25(c,d) illustrate the corresponding optimal uniform word-length implementations. The multiple word-length design paradigm has been exploited in order to achieve a tight-fitting implementation. Comparing Figs. 4.25(a) and (c) and Figs. 4.25(b) and (d) demonstrates that the optimization has been able to 'stretch' the output noise PSD to closely meet the specification. This in turn translates into significant

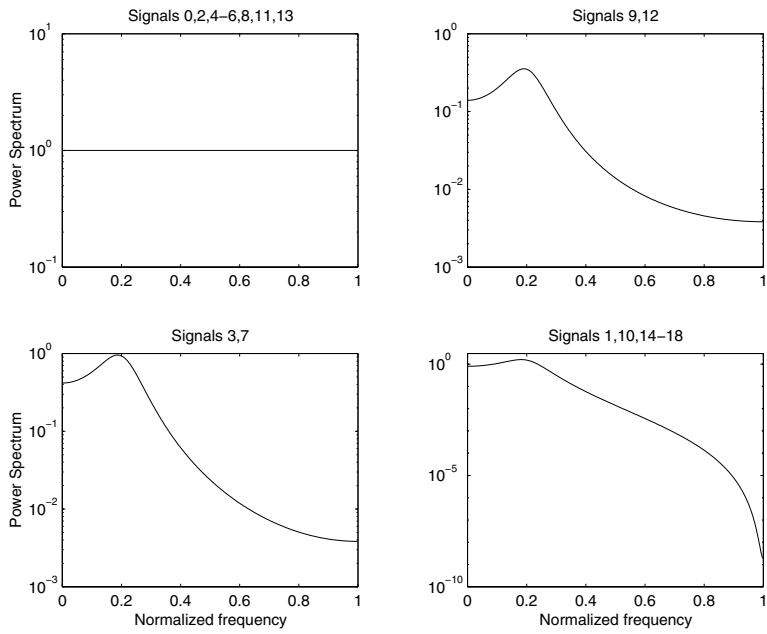


Fig. 4.24. Spectral profiles through the filter

area savings: for this example the uniform word-length design requires 810 logic cells in an Altera Flex10K device, compared to 636 and 663 logic cells for the shaped designs, a 22% and 18% area reduction respectively.

The spectral specifications are only slightly different in the two cases of Figs. 4.25(a) and (b): Fig. 4.25(b) has a somewhat reduced bound on high-frequency noise. The optimization procedure has successfully incorporated the modified constraint by reducing the high-frequency noise in the implemented structure. In contrast, there has been no change to the uniform word-length system between Figs. 4.25(c) and (d), since with a uniform word-length structure only a limited range of output noise spectra are possible.

4.6.2 Nonlinear Systems

Case Study: Adaptive Filtering

Adaptive filtering is a common DSP application, especially in the field of communications where it is widely used, for example, to compensate for multi-path distortion in mobile communication systems [Hay96].

In addition to its practical significance, adaptive filtering has some interesting algorithmic features:

- All adaptive filtering algorithms contain feedback, limiting the applicability of several existing word-length optimization techniques [WP98,

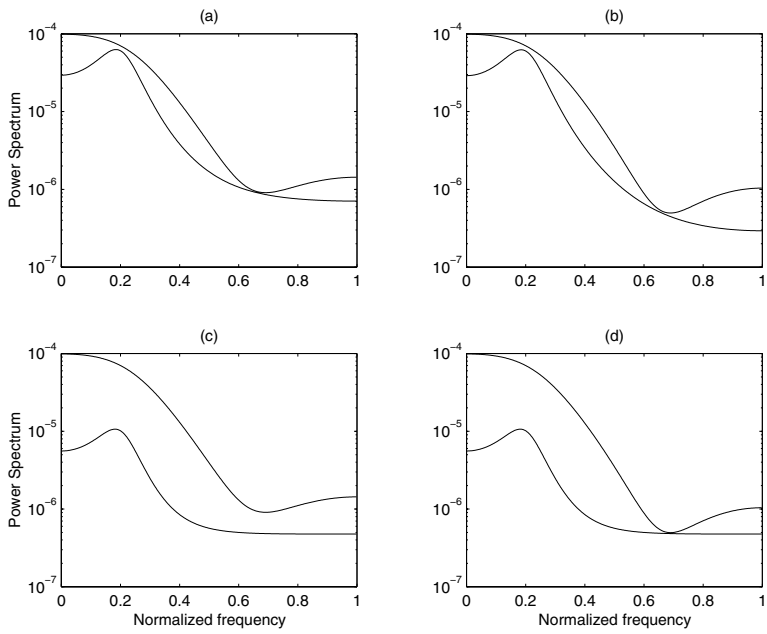


Fig. 4.25. Two specifications (upper curve) and their optimized (a,b) multiple word-length and (c,d) uniform word-length implementation noise spectra

NHCB01, BP00, SBA00, CRS+99] and limiting the performance achievable through pipelining.

- Adaptive filters contain general multipliers, rather than the constant coefficient multipliers present in static filters. This means that adaptive filters are nonlinear systems, limiting the applicability of purely analytic techniques such as that presented in Section 4.1.2 [CCL01b, CCL02].
- The coefficients of an adaptive filter are updated by accumulating (usually small) correction terms. Such 'integration loops' make the outputs of an adaptive filter very sensitive to errors induced around such loops.

The so-called least-mean-square (LMS) adaptive filter [Hay96] will be considered in this section, due to its widespread use in practice. For the unfamiliar reader, a brief review of LMS filters will now be provided.

Consider an input signal $x[t]$ and a desired filter response $d[t]$. (The desired response could be known *a-priori*, for example from a 'training sequence' used in GSM mobile telephony). Let n denote the order of the filter, and $\mathbf{u}[t]$ denote the vector $\mathbf{u}[t] = (x[t]\ x[t-1]\ x[t-2]\ \ldots\ x[t-n])^T$, where ^{T} represents vector/matrix transpose. An LMS filter with real input and coefficients has the following algorithm, where $\mathbf{0}$ represents a column vector with each element equal to 0, and μ is a user-chosen scalar adaptation coefficient.

$\mathbf{w}[0] = \mathbf{0}$
for $t \geq 0$ **do**
    $y[t] = \mathbf{w}^T[t]\mathbf{u}[t]$
    $e[t] = d[t] - y[t]$
    $\mathbf{w}[t+1] = \mathbf{w}[t] + \mu\mathbf{u}[t]e[t]$
end do

A DFG for a first-order LMS filter is shown in Fig. 4.26. The DFG for an nth order filter is easily derived through a replication of the taps and the use of an adder-tree to sum the partial results.

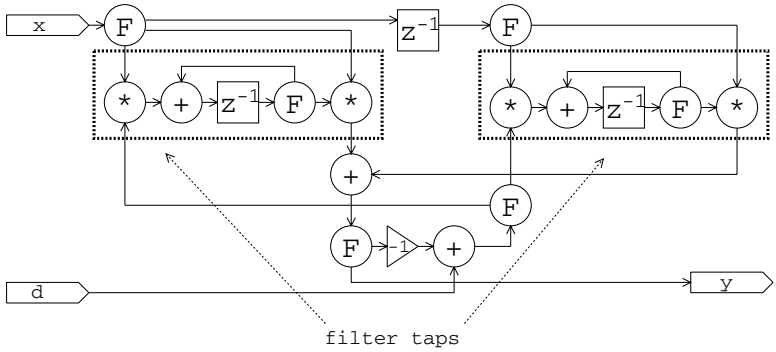


Fig. 4.26. First order LMS adaptive filter

Area, Power, and Speed

In order to demonstrate the area, power, and delay advantages of the proposed method, 90 filters of between 1st and 10th order have been constructed and synthesised. In each case the 'desired' input $d[t]$ to the adaptive filter is a well-known 100,000 sample voice clip from [FRE93]. The filter input $x[t]$ is a version of the same signal, corrupted by three different 12th order autoregressive filters, operating on three disjoint and equally sized portions of the input signal. Each distortion filter has constant coefficients randomly chosen such that the filter poles occur in complex conjugate pairs and have independent, identically distributed uniform distribution in magnitude range $(0, 1)$ and in phase range $(0, \pi/2)$.

The filter designs and input sequences have then been passed to the synthesis tool, and for each design three different optimization procedures have been followed. Firstly, the design has been synthesized with the optimum uniform scaling and the optimum uniform word-length for all signals. This design choice reflects the simplest form of optimized DSP design. Secondly, the design has been synthesized with scaling individually optimized for each signal (see

Chapter 3) and the optimum uniform word-length. This design choice reflects the use of a tool such as [SBA00] which focuses on optimizing signals from the MSB-side. The final design procedure has been to use an individually optimized scaling combined with an individually optimized word-length, as proposed by this chapter.

From the filter designs in Simulink and the representative input sequences, the synthesis tool automatically generates a combination of structural VHDL and Xilinx Coregen scripts, together with a makefile to synthesize the Virtex bit-stream. Each design has been fully placed and routed in a Xilinx Virtex 1000 (XCV1000BG560-6), after which an area, power consumption, and timing analysis has been performed. Due to memory and run-time constraints imposed by large value-change-dump simulation files, power analysis could only be performed for a 100-sample portion of the 100,000-sample input sequence used by the tool.

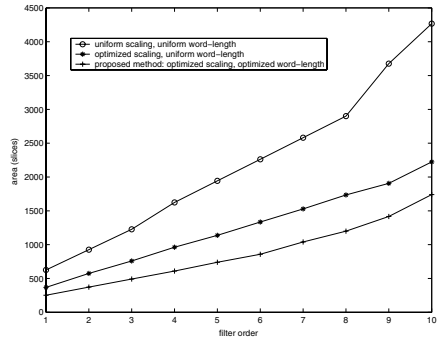
The first set of results is concerned with the variation of design metrics with the order of the filter to be synthesized. For each of these results, the filters have been synthesized using the same lower-bound on output SNR of 34dB.

The results are illustrated in Fig. 4.27(a), (b) and (c) for area, power, and clock period, respectively. Area savings of up to 37% (mean 32%) have been achieved over scaling optimization alone, and up to 63% (mean 61%) over neither scaling nor word-length optimization. This is *combined* with a power reduction of up to 49% (mean 43%) and speed-up of up to 18% (mean 10%) over scaling optimization alone, and a power reduction of up to 84.6% (mean 81.2%) and speed-up of up to 29% (mean 18%) over neither scaling nor word-length optimization.

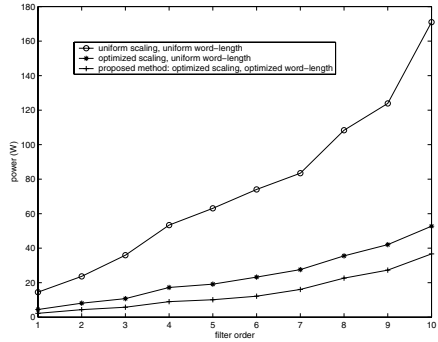
The second set of results is concerned with the variation of design metrics with the user-specified lower-bound on allowable SNR. For these results, a 5th order LMS filter has been synthesized with SNR bound varying between −6dB and 64dB. These results are illustrated in Fig. 4.28(a), (b) and (c) for area, power, and clock period, respectively. As well as demonstrating the useful capability to trade-off numerical accuracy for area, power and speed, these results also illustrate significant improvements in all three metrics.

Area savings have been achieved of up to 75% (mean 45%) over scaling optimization alone, and up to 80% (mean 66%) over neither scaling nor word-length optimization. This is *combined* with a power reduction of up to 96% (mean 58%) and speed-up of up to 29% (mean 11%) over scaling optimization alone, and a power reduction of up 98% (mean 87%) and speed up of up to 36% (mean 20%) over neither scaling nor word-length optimization.

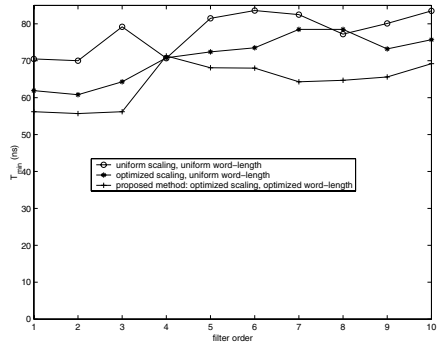
It should be expected that on average the power savings are no smaller than the area savings of this approach. However in practice, the power savings are often significantly greater. This can be explained by two observations relating to the switching activity of signals. Firstly, if the scaling of each signal is not individually optimized, then a significant number of signals will contain unnecessary sign-extension. When a two's complement signal changes from a



(a) variation of area with filter order

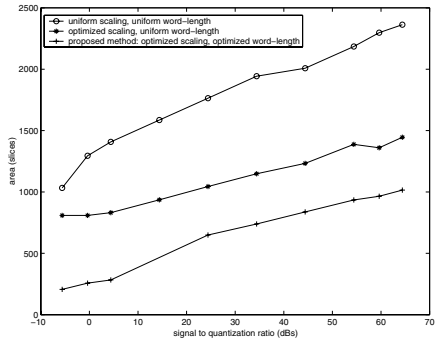


(b) variation of power consumption with filter order

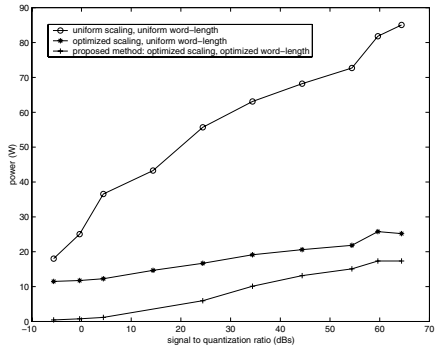


(c) variation of minimum realizable clock period with filter order

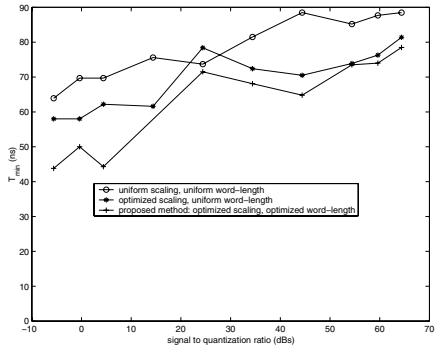
Fig. 4.27. Synthesis results for LMS adaptive filters (fixed SNR bound of 34dB)



(a) variation of area with SNR bound



(b) variation of power consumption with SNR bound



(c) variation of minimum realizable clock period with SNR bound

Fig. 4.28. Synthesis results for LMS adaptive filters (5th order filter)

positive to a negative value, or vice-versa, *all* of these MSBs will toggle. Thus the overall switching activity in a realization can be reduced dramatically by applying scaling optimization. Secondly, when a sampled signal is in a period of relatively low-frequency (with respect to the Nyquist rate), the activity amongst low-order bits is, on average, likely to be significantly larger than that amongst high-order bits due to the slowly changing signal value. Thus word-length optimization, which specifically targets the low-order bits of each signal, is likely to lead to a significant reduction in the overall activity level. In addition, it is likely that a large portion of the power consumption due to logic activity in DSP systems derives from multiplier cores. In multipliers, the power consumption is far more sensitive to reductions in the switching activity of low-order input bits than that of high-order input bits [MS01]. These explanations are supported by the plot of Fig. 4.28(b) which shows the power saving of the proposed method over scaling optimization alone increasing rapidly for low SNR. This is because the low SNR allows word-length optimization to aggressively target more low-order bits.

4.6.3 Limit-cycles in Multiple Word-Length Implementations

The multiple word-length design paradigm, combined with a word-length optimization technique, has been shown to be highly effective at optimizing system area for a given user-specified bound on truncation noise. However, a finite precision implementation can additionally suffer from certain types of noise not considered in Section 4.1.2. A finite precision implementation of an IIR filter is essentially a finite state machine (FSM). Under any unchanging input vector, an FSM may exhibit one of two steady-state behaviours: it may either settle in an 'attractor state', or it may cycle around a finite number of states. The latter of the two behaviours can result in output oscillations in a finite precision implementation, which would not be present for the infinite precision case. In Digital Signal Processing, this inherently non-linear behaviour is referred to as limit-cycle behaviour [Mit98]. There have been several studies into limit cycles [LMV88, BB90, PKBL96], generally focussing on conditions for non-existence of limit cycles in uniform word-length implementations, and indentifying regions of the coefficient space guaranteed to be limit-cycle free. While limit cycle behaviour is not considered by the optimization procedure developed in this chapter, it is nevertheless important from a user's perspective that the limit-cycle behaviour of the optimized multiple word-length systems is not generally worse than that of more traditional implementation schemes.

In order to compare the limit-cycle behaviour of uniform word-length and optimized multiple word-length systems, the following experimental procedure has been followed. Fifty thousand second order auto-regressive filters have been generated, with coefficients uniformly selected from the coefficient regions likely to result in limit cycles of period one or two [LMV88]. Each point in Fig. 4.29 illustrates a single such coefficient vector. Each of these filters has

then been synthesized using the optimum uniform word-length for a range of specified maximum error variances, and the resulting truncation error has been estimated. Each of the truncation errors forms the specification for a multiple word-length implementation of the same filter. Comparison of uniform and multiple word-length implementations may be achieved by exciting each filter with a large impulse and measuring statistics on the output signal once the transient effects have died away. The peak and the power of each limit cycle are shown in Fig. 4.30.

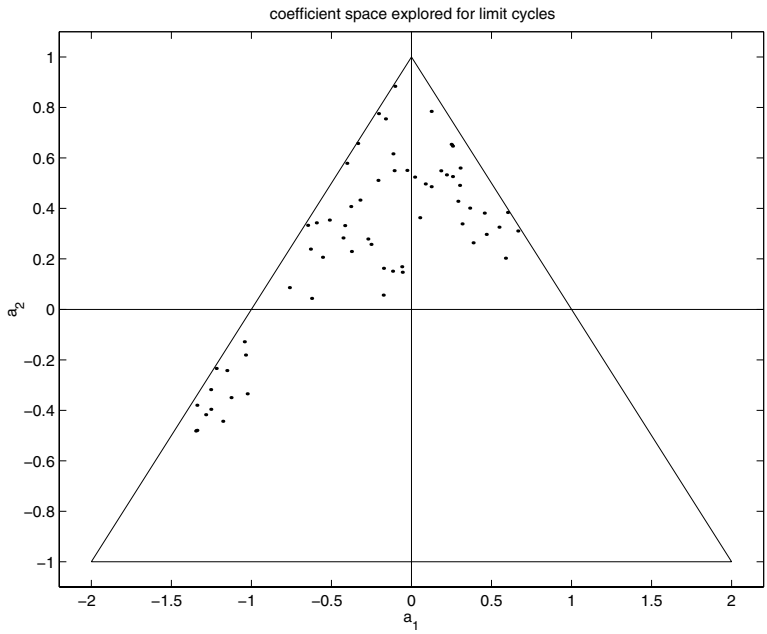


Fig. 4.29. Coefficient space searched for limit cycles

The results of Fig. 4.30 are provided both on a log-log scale in order to observe the spread of results, and on a linear-linear scale in order for the cases without limit-cycles to be observable. A total of 17170 out of 50407 (34%) of uniform word-length implementations exhibited no limit cycle behaviour, whereas 20030 out of 50407 (40%) of multiple word-length implementations exhibited no limit cycle behaviour. For those cases where both implementations exhibited limit cycle behaviour, a histogram of the relative power of the two limit cycles is shown in Fig. 4.31. For these cases, a multiple word-length implementation has on average a 0.8dB lower limit cycle power than the equivalent uniform word-length implementation.

It can be concluded that multiple word-length implementations are somewhat more likely than their uniform word-length equivalent to be free of period

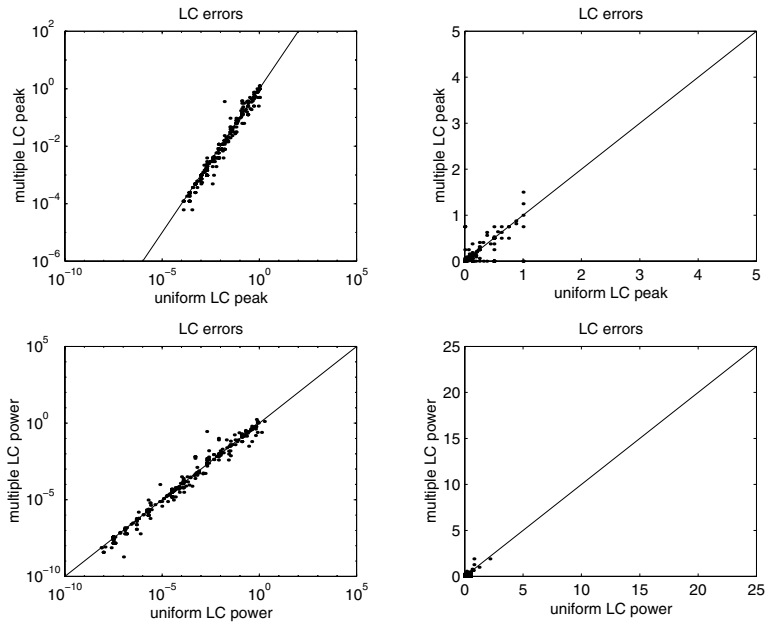


Fig. 4.30. Limit cycle peak and power

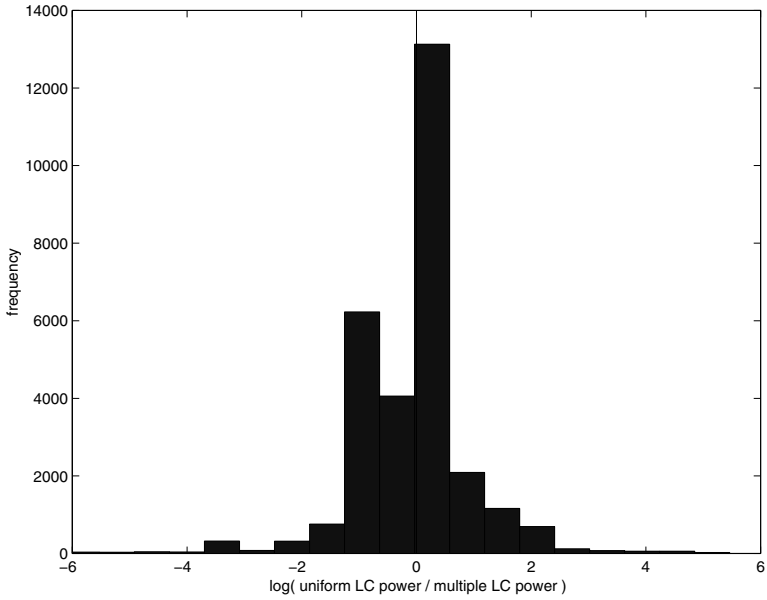


Fig. 4.31. Relative error power of limit cycles

one or two zero-input impulse induced limit cycle behaviour. When both implementation structures exhibit limit cycle behaviour, the multiple word-length limit cycles tend, on average, to have very slightly lower power than the equivalent optimum uniform word-length implementation. These should not be surprising results, since it is feedback loops that are often the most sensitive to the truncation error modelled in Section 4.1.2. Therefore multiple word-length optimizations will generally prefer to reduce word-lengths in other locations than those around such sensitive loops, which are also the cause of limit cycle behaviour.

4.7 Summary

This chapter has introduced several methods for automating the design of bit-parallel multiple word-length implementations of DSP systems. A lossy synthesis approach has been described, based on optimizing the area consumption of the resulting implementation, subject to constraints on the finite precision errors.

Two special cases have been dealt with in detail: linear time-invariant systems, and nonlinear systems containing only differentiable nonlinearities. Two optimization methods have been proposed: an application specific heuristic, and an optimum approach based on integer linear programming.

It has been demonstrated that the multiple word-length design paradigm allows a broad design space to be searched by the synthesis tools, leading to high quality results.

In this chapter, system area estimation was performed using the assumption of a dedicated resource binding. However the synthesis of large systems may not be possible using this design methodology. It is often necessary to trade off the number of signal samples processed per clock period against system area through the use of operation scheduling and binding techniques. Chapter 6 addresses these problems for the case of multiple word-length systems.

5

Saturation Arithmetic

This chapter explains how saturation arithmetic can be used to optimize multiple word-length designs. Section 5.1 first motivates our approach and covers some background material. Section 5.2 contains a discussion of the overheads in terms of system area and speed associated with saturation arithmetic implementations. After introducing some necessary definitions in Section 5.3, a technique for analytic estimation of saturation arithmetic noise is presented in detail in Section 5.4. This noise estimation procedure forms the basis of an extension, presented in Section 5.5, to the optimization heuristic discussed in the previous chapter, to jointly optimize signal word-lengths and scalings. This algorithm has been implemented as part of the Synoptix system, and the results obtained are presented and discussed in Section 5.6. The chapter ends with conclusions in Section 5.7.

5.1 Overview

In a standard implementation of a DSP system, when adding two numbers using two's complement representation, there is a possibility of overflow. In the two's complement representation, overflow results in a 'wrap-around' phenomenon, where attempts to represent positive numbers just outside the representable range result in their interpretation as large negative numbers, and vice versa. The result can be a catastrophic loss in signal-to-noise ratio. Signals in digital signal processing designs are therefore usually either scaled appropriately to avoid overflow for all but the most extreme input vectors, or saturation arithmetic is used. Saturation arithmetic introduces extra hardware to avoid the wrap-around, replacing it with saturation to either the largest positive number or the largest negative number representable at the adder output.

In microprocessor based DSP, hardware support for saturation modes is usually available, and the decision on whether saturation arithmetic should be used is typically left to the programmer, for example in the TMS320C6200

DSP [TI] or the MMX extensions to the Pentium [PW96]. For direct implementation in hardware, the choice of whether or not to use saturation arithmetic is particularly important due to the associated cost of the extra saturation logic in area and delay.

For LTI systems, if ℓ_1 scaling is used, as in Chapter 4, overflow is not an issue and therefore standard arithmetic is to be preferred. In some cases ℓ_1 scaling can be overly pessimistic, catering for situations that are extremely rarely encountered with practical input signals. Overly pessimistic signal scaling can lead to a number of most significant bits (MSBs) in a datapath being left unused. This is true particularly in the case of filters with long impulse responses. Under these circumstances an alternative scaling scheme could be used to reduce the datapath width and therefore save implementation area and power consumption, however overflow becomes a distinct possibility. It is in these situations that saturation arithmetic becomes a useful implementation scheme, to limit the impact of these overflows on the overall system signal-to-noise ratio.

A reasonable design approach, when creating a saturation arithmetic implementation of a DSP system, is to determine signal scaling through simulation. Input vectors are supplied to the system, and the peak value reached by each internal signal is recorded. Signals are then scaled to ensure that the full dynamic range afforded by the signal representation would be used under excitation with the given input vectors. In contrast, this chapter presents an optimization technique suitable for LTI systems, based on analytic noise models.

5.2 Saturation Arithmetic Overheads

Saturation arithmetic is not a cost-free design methodology. The implementation of a saturation arithmetic component has some area and delay overheads associated with it, when compared to the equivalent non-saturation component.

Traditionally, saturation arithmetic has been associated with the addition operator [Mit98]. However there are also circumstances where saturation may be appropriate following a multiplication. Consider for example a $(7, 0)$ signal which uses its full dynamic range, and therefore has a peak of $\pm(1 - 2^{-7})$. We may multiply this signal by a $(7, 1)$ coefficient with value $1 + 2^{-6}$. The resulting signal will therefore have a peak of $\pm(1+2^{-6}-2^{-7}-2^{-13})$, although the $(14, 1)$ format for this signal could represent peaks of up to $\pm(2^1 - 2^{-13})$, approximately double the range. It is intuitive, therefore, that a saturation by one bit after the multiplication would rarely saturate to its peak values, and choosing a $(13, 0)$ representation for the result rather than $(14, 1)$ may result in a more efficient implementation at only a small cost in saturation error. The technique described in this chapter generalizes the approach, to

allow saturators to be inserted after any operator: primary input, addition, delay or constant coefficient multiplication.

Another standard approach to saturation arithmetic is to saturate only a single most significant bit (MSB). A standard saturation would therefore convert an (n, p) representation into an $(n-1, p-1)$ representation. This design approach derives from having a single n-bit accumulator. The approach taken in this chapter does not constrain the design space in this way, and indeed can saturate different signals to different degrees, in order to trade off implementation area with implementation error.

The key component introduced and studied in this chapter is therefore a saturator, as defined in Definition 5.1 and illustrated in Fig. 5.1 for a k-bit saturator applied to an $(n+1)$-bit two's complement signal. The symbols used follow IEEE standard [IEE86]: specifically '>=1' refers to an 'or' gate, 'G1' to the common select line of the two-input multiplexers, and inversion is indicated by a triangle. From this architecture, a simple area cost model can be deduced for the saturator: $A(n,k) = c_1 k + c_2 n$, where c_1 and c_2 are empirically derived constants which may vary with target architecture. The cost function used in Section 4.2 must therefore be modified to incorporate this additional cost. Of course, the output of every operator need not be saturated: following the notation of Section 4.1.1, saturation is never needed if $p_j = p'_j$ for a signal j, because the full output range is required. In addition for an LTI system, if there is no signal j such that p_j is less than the value suggested by ℓ_1 scaling, then the system is free from overflow and the saturation area model need not be used, even if there is a signal j with $p_j < p'_j$.

Definition 5.1. A *saturator* is a circuit component, as illustrated in Fig. 5.1, for converting from format (n, p) to format $(n-k, p-k)$. The parameter k is referred to as the *degree* of the k-bit saturator.

Another difference to the area model used in Chapter 4, is that the previously cost-free computation nodes INPORT and FORK may now have a cost associated with them due to any saturation nonlinearities at their outputs. FORK is of particular interest, since for a minimal area implementation, a FORK node should be implemented as a cascade of saturators as shown in Fig. 5.2.

In addition to the area overhead of saturation arithmetic, there will also be some delay penalty associated with the saturator. Fig. 5.3 illustrates how the placed and routed propagation delay across a saturation arithmetic constant coefficient multiplier varies with input word-length for a fixed coefficient value and for 1-bit saturation. This figure shows a very significant timing overhead for saturation arithmetic, up to 73%.

To summarize, saturation arithmetic provides advantages in terms of allowing controlled overflows that may not dramatically affect the output signal-to-noise ratio. However these advantages are provided at the cost of a somewhat larger and slower circuit compared to standard two's complement arithmetic for the same binary point locations and word-lengths. In a given imple-

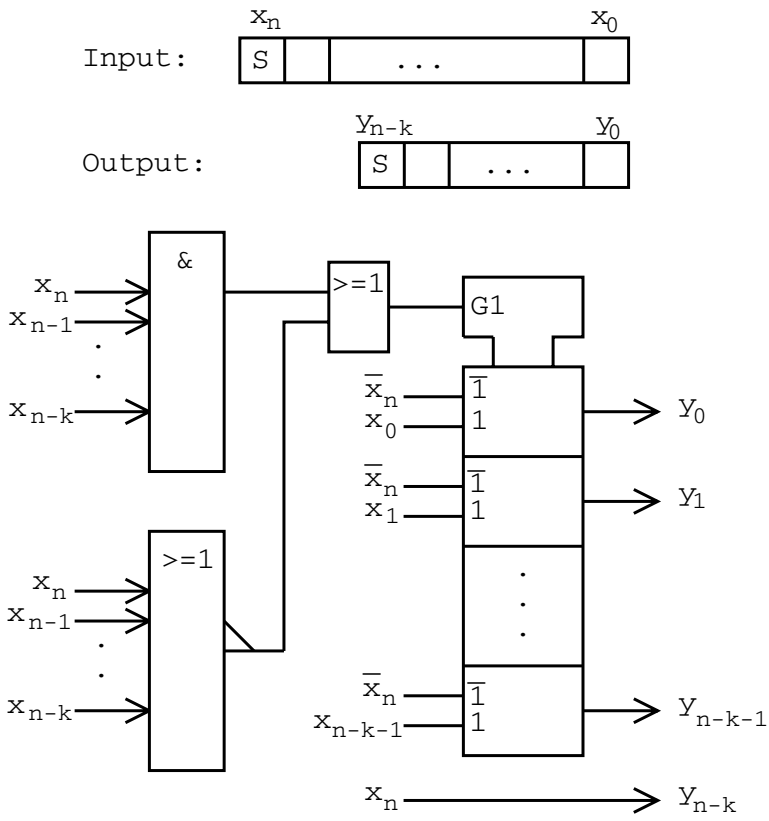


Fig. 5.1. A two's complement saturator from (n, p) to $(n-k, p-k)$

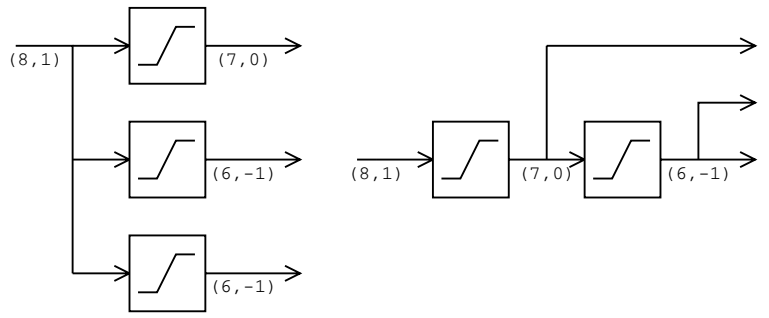


Fig. 5.2. Cascaded minimal area implementation of a fork saturation

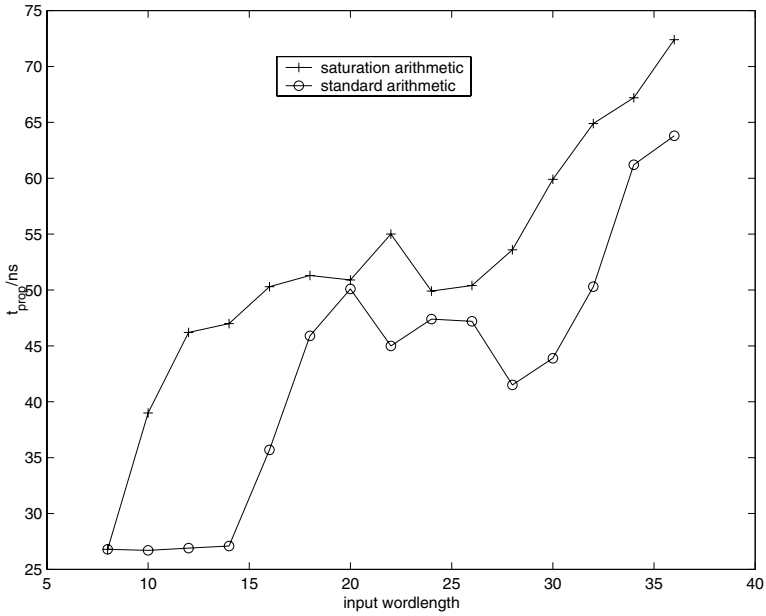


Fig. 5.3. Propagation delay across standard and saturation arithmetic constant coefficient multipliers implemented in Altera Flex10k70RC240-3

mentation, it is not necessary to use saturation arithmetic for all operators, so care must be taken to select the appropriate places and the appropriate degree of saturation to apply at those points. For systems with long impulse responses it may be possible, through judicious choice of saturator location and degree, to create a smaller implementation of the system using saturation arithmetic.

5.3 Preliminaries

The noise model for saturation arithmetic, presented in the following section, will involve the concepts of saturation nonlinearities, saturation systems and cross-correlations, which are defined below.

Definition 5.2. A *saturation nonlinearity* is a function of the form shown in (5.1). The parameter $c > 0$ is referred to as the *cut-off* of the saturation nonlinearity. ($\mathrm{sgn}(\cdot)$ is the signum function, which has value -1 for negative argument, and +1 otherwise).

$$s_c(x) = \begin{cases} x, & |x| \leq c \\ c\,\mathrm{sgn}(x), & \text{otherwise} \end{cases} \tag{5.1}$$

A saturation nonlinearity can be considered as a generalized model for a two's complement saturator. The nonlinearity is more general as the cut-off used to model a two's complement saturator will be an integral power of two, however there is no such restriction on saturation nonlinearity cut-off.

Definition 5.3. A *saturation system* is a system constructed from a Linear Time Invariant system by introducing at least one saturation nonlinearity at an output of an operation. At least one of these nonlinearities must have a cut-off less than the ℓ_1 peak value at that output. The Linear Time Invariant system from which the saturation system is constructed is referred to as the *underlying LTI system.*

The formal representation of a saturation system is as a saturation computation graph $G_S(V, S, C)$, as defined below.

Definition 5.4. A *saturation computation graph* $G_S(V, S, C)$ is an annotated form of a computation graph $G(V, S)$.
The set C takes the form $C = \{(j_1, c_1), (j_2, c_2), \ldots, (j_p, c_p)\}$, where $j_i \in S$ and $c_i \in (0, \infty)$. C models the position j_i and cut-off c_i of each saturation nonlinearity $1 \leq i \leq p$ in the saturation system. □

Note that in some graphical representations of a saturation system, the saturation nonlinearities will be explicitly shown. These are not to be considered nodes in the saturation computation graph $G_S(V, S, C)$, but are rather implicit in the choice of annotation C. Similarly the formal representation of a fixed-point realization of a saturation system is simply an annotated computation graph $G'(V, S, A)$ (Definition 2.5). The saturator locations are implicit in the choice of annotation A.

Definition 5.5. The *cross-correlation function* $r_{xy}[\tau]$ between two statistically stationary random processes x and y is defined to be $r_{xy}[\tau] = E\{x[t]y[t-\tau]\}$.

5.4 Noise Model

A fixed-point approximate realization of a saturation system is a nonlinear dynamical system containing two types of nonlinearities: saturations and truncations. Saturations are large-scale nonlinearities affecting the most significant bits of a word, whereas truncations are small-scale nonlinearities affecting a few of the least significant bits of a word. There is therefore good reason to consider the two effects separately, and also good reason to assume that these errors are approximately uncorrelated since correlation between high-order and low-order bit patterns is unlikely. The exception is if the saturation nonlinearities are so extreme (have such low cut-offs compared to the dynamic range of the saturated signal) that the probability density function used in

Section 4.1.2 becomes invalid. However this would require a saturation nonlinearity to spend a considerable portion of its time in the 'saturated state', an unlikely situation in practical cases. For this reason, only saturation error modelling is considered in this section. The overall error estimate is then formed by summing the predicted error variance due to saturation with that due to the truncation model described in Section 4.1.2.

5.4.1 Conditioning an Annotated Computation Graph

The concept of a well-conditioned structure (Section 4.1.1) can easily be extended to systems involving saturation. In such systems there is an extra source of ill-conditioning, arising from the possibility that for a signal j, $p'_j < p_j$ (notation as defined in Section 4.1.1). Recall that this was not a possibility when signal scalings were determined by an ℓ_1 approach, because ℓ_1 scaling ensures that the entire dynamic range of the signal is required if an appropriate input stimulus is provided. Note from Table 4.1 that although the word-length $n_j^{q'}$ resulting from an operation can depend both on the scalings and word-lengths of the operation inputs, the scaling p'_j depends only on the input scalings. This means it is possible to separate the conditioning of an annotated computation graph into two phases: first the scalings are conditioned, and then the word-lengths are conditioned following the procedure in Section 4.1.1. The scaling conditioning algorithm is given in Algorithm ScaleCondition.

Algorithm 5.1
Algorithm ScaleCondition
Input: An annotated computation graph $G'(V, S, A)$
Output: An annotated computation graph, with well-conditioned scalings and identical behaviour to the input system
begin
 Calculate p'_j for all signals $j \in S$ (Table 4.1)
 while $\exists j \in S : p'_j < p_j$
  Set $p_j \leftarrow p'_j$
  Update p'_j for all affected signals (Table 4.1)
 end while
end

5.4.2 The Saturated Gaussian Distribution

In order to estimate the error incurred through the introduction of one or more saturation nonlinearities, a model is required for the Probability Density Function (pdf) of a signal undergoing saturation. The first simplifying assumption made is that these pdfs may be approximated by a zero-mean Gaussian distribution. Gaussianity is a useful assumption from the modelling perspective, since the addition of two (arbitrarily correlated) zero-mean Gaussian variables forms another zero-mean Gaussian variable, and the scaling of a

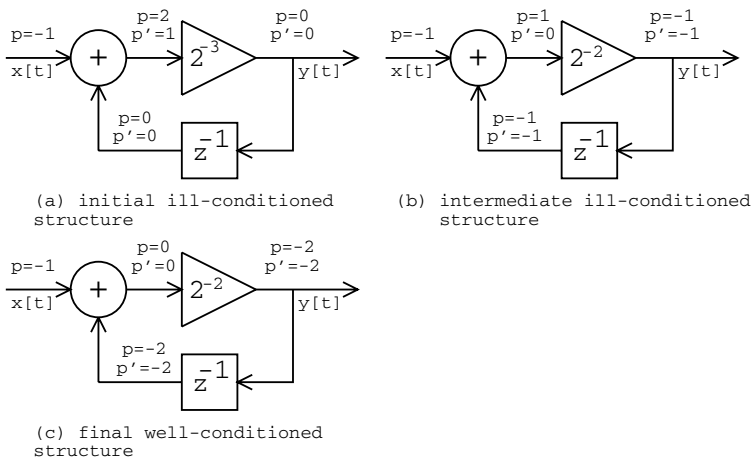


Fig. 5.4. An example of scaling conditioning

zero-mean Gaussian variable also forms another zero-mean Gaussian variable. It therefore follows that all internal signals in an LTI system driven by zero-mean Gaussian inputs will themselves be zero-mean Gaussian, since all signals can be expressed as a weighted sum of present and past inputs. The Gaussian assumption is also useful because the joint pdf $f_{XY}(x, y)$ of two zero-mean Gaussian variables X and Y is completely known from their respective variances and correlation coefficient, defined as $\gamma = E\{XY\}/\sqrt{E\{X^2\}E\{Y^2\}}$.

In reality, however, inputs may follow a large variety of distributions which will cause the intermediate signals in the modelled system to deviate to some extent from their idealized Gaussian form. The assumption is that such a deviation will be small enough for practical cases and for the purposes to which this model will be put. Often the largest deviation from the Gaussian model is likely to be at the primary inputs to the system, since the internal nodes are formed by a weighted sum of present and past input values. In the most extreme example, where the LTI system under investigation approaches a normalized integrator (transfer function $H(z) = \lim_{n\to\infty} n^{-1}(1 - z^{-n})/(1 - z^{-1})$) and the input is made up of a stream of independent identically distributed (iid) random variables, the Gaussian approximation will clearly hold no matter what the input distribution, by the central limit theorem. In more general cases there are an abundance of extensions to the central limit theorem for specific relaxations of the constraints on independence and identical distribution [Chu74]. While there is no general theoretical result for all cases, it is reasonable to assume that bell-shaped distributions are common in practice, and evidence to support the assumption is available in Section 5.4.6, where modelling results are compared to simulations of 'real-world' speech input data.

The introduction of saturation nonlinearities into the system significantly complicates the picture. The *saturated Gaussian* distribution, as defined below and illustrated in Fig. 5.5, is used to model the pdf at each internal signal within a saturation system.

Definition 5.6. A random variable X follows a *saturated Gaussian* distribution with parameters $(\sigma,\, c)$ where $\sigma \geq 0$ and $c \geq 0$ iff its probability density function $f_X(x)$ has the form given in (5.2). The Gaussian distribution with mean 0 and standard deviation σ is referred to as the *underlying distribution*.

$$f_X(x) = \begin{cases} Q(c/\sigma)\,(\delta(x-c) + \delta(x+c)) + \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{x^2}{2\sigma^2}), & \text{if } |x| \leq c \\ 0, & \text{otherwise} \end{cases} \tag{5.2}$$

Here $\delta(\cdot)$ is the Dirac delta function [Gar90] and $Q(\cdot)$ represents the 'upper tail' function of the standard Gaussian distribution.

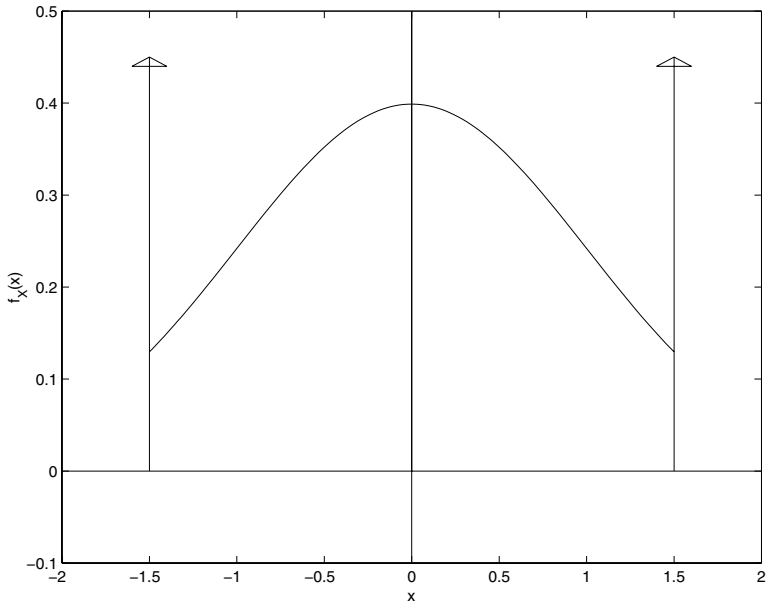


Fig. 5.5. The saturated Gaussian distribution with parameters ($\sigma = 1$, $c = 1.5$)

Since the pdf is an even function, all odd moments of the probability distribution vanish to zero. Expressions are provided below for the second and fourth moment of the saturated Gaussian, which will be used for modelling purposes.

The second moment (variance) of a saturated Gaussian distribution can be calculated though (5.3).

$$\mu_2 = \sigma^2 - 2\left\{Q(c/\sigma)(\sigma^2 - c^2) + \frac{c\sigma}{\sqrt{2\pi}}\exp(-\frac{c^2}{2\sigma^2})\right\} \tag{5.3}$$

The fourth moment (related to Kurtosis) of a saturated Gaussian distribution can be calculated through (5.4).

$$\mu_4 = 3\sigma^4 - 2\left\{Q(c/\sigma)(3\sigma^4 - c^4) + \frac{1}{\sqrt{2\pi}}c\sigma(c^2 + 3\sigma^2)\exp(-\frac{c^2}{2\sigma^2})\right\} \tag{5.4}$$

Multiplication of a saturated Gaussian random variable of parameters (σ, c) by a constant factor k results in a random variable with saturated Gaussian distribution of parameters ($k\sigma$, kc). Clearly the multiple outputs of a branching node whose input has parameters (σ, c) will all have the same parameters (σ, c), and the output of a delay node will behave in a similar manner.

Let X be one such post-multiplication, post-branching, or post-delay saturated Gaussian signal. This signal may then itself be saturated by a nonlinearity with cut-off $\bar{c}$. The variance of the associated error e injected at the point of saturation can be calculated as in (5.5), where $f_{\hat{X}}(x)$ is the pdf of the underlying Gaussian distribution of X.

$$\begin{aligned} E\{e^2\} &= \begin{cases} 2\int_{\bar{c}}^{c}(\bar{c}-x)^2 f_{\hat{X}}(x)dx + (\bar{c}-c)^2 Q(c/\sigma), & \bar{c} < c \\ 0, & \text{otherwise} \end{cases} \\ &= \begin{cases} 2(c-\bar{c})^2 Q(c/\sigma) + 2(\bar{c}^2+\sigma^2)\left[Q(\bar{c}/\sigma) - Q(c/\sigma)\right] + \\ \frac{2\sigma}{\sqrt{2\pi}}\left[(2\bar{c}-c)\exp(-\frac{c^2}{2\sigma^2}) - \bar{c}\exp(-\frac{\bar{c}^2}{2\sigma^2})\right], & \bar{c} < c \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{5.5}$$

For addition the situation is more complex, and is addressed in the following section.

5.4.3 Addition of Saturated Gaussians

While the addition of two Gaussian random variables follows a Gaussian distribution, it is not true that the addition of two saturated Gaussian random variables follows a saturated Gaussian distribution. In fact, the distribution formed by their addition may follow a number of forms depending on the correlation between the two inputs to the addition, and their respective 'c' parameters. Fig. 5.6 illustrates an addition followed by a saturation nonlinearity.

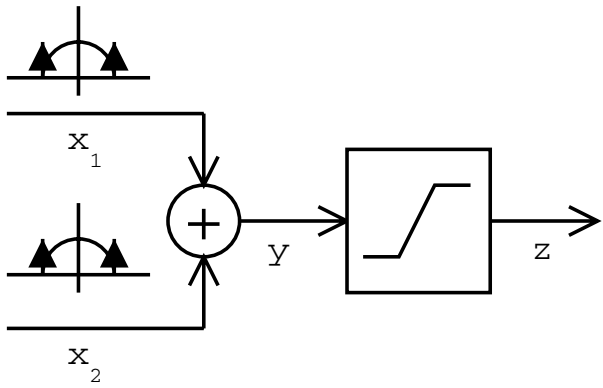


Fig. 5.6. An addition followed by a saturation

Example 5.7. Fig. 5.7 illustrates one example pdf (before the post-adder saturation) formed from the summation of two uncorrelated saturated Gaussian random variables with parameters $(\sigma_1 = 1.6e-02, c_1 = 2^{-5})$ and $(\sigma_2 = 2.2e-01, c_2 = 2^{-2})$ respectively.

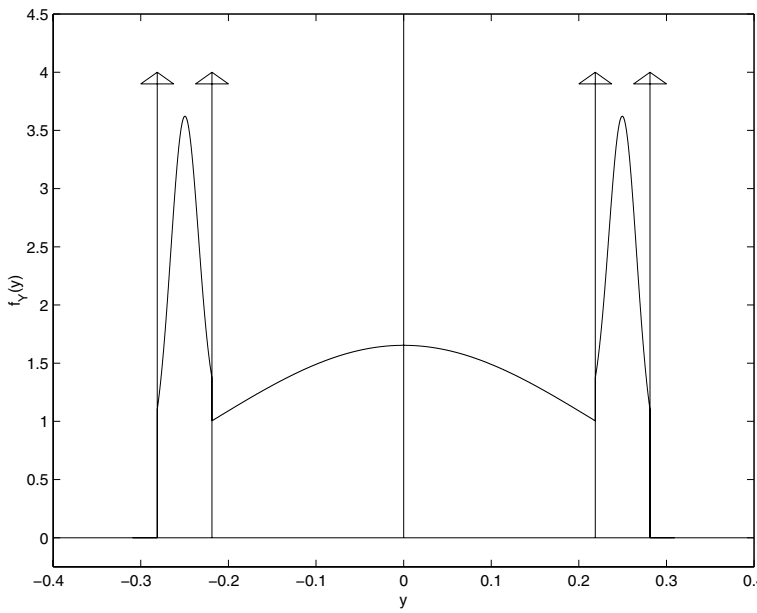


Fig. 5.7. A probability density function for the sum of two saturated Gaussian variables

The pdf of the summed saturated Gaussians can be visualized as deriving directly from the underlying joint Gaussian pdf of the of the two inputs. This is illustrated in Fig. 5.8, showing the portions of the underlying joint probability space corresponding to particular values of the sum. Shaded regions indicate entire regions of the plane resulting in the same sum, and black lines link the locus of single points resulting in this sum, as well as indicating the borders of shaded regions.

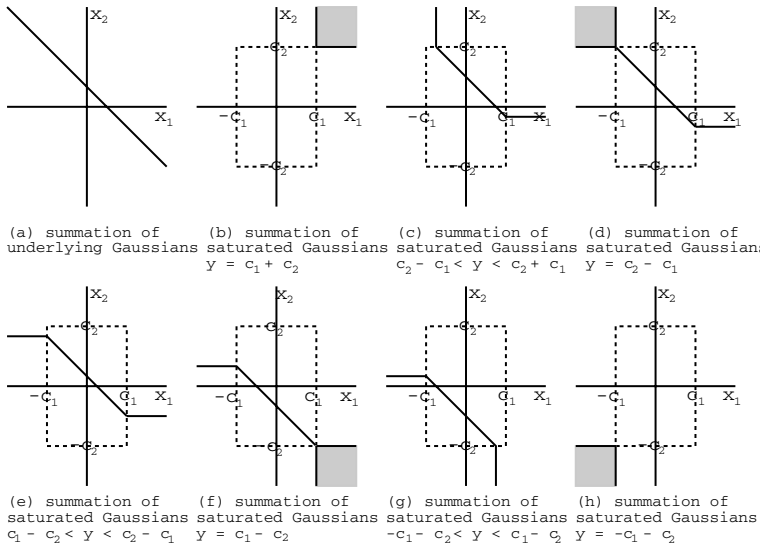


Fig. 5.8. The pdf of the sum of two saturated Gaussians (b)–(h) compared to the pdf of the sum of the underlying Gaussians (a)

While theoretically these complex distributions could be used as models for the signals in a saturation system, in practice the computational complexity associated with error estimation using these models is exponential in the number of additions in the system. In order to use error estimation within the tight inner loop of an optimization procedure, a linear-time estimation procedure is required. For this reason, the distribution of the random variable after both the addition and its corresponding saturation is approximated by a saturated Gaussian. The parameters of the saturated Gaussian approximation can be tuned to best approximate the more complex pdf. This approach allows a simple linear-time estimation procedure to be used, while sacrificing some accuracy. Note that the estimate of the error caused by the saturation nonlinearity immediately following the addition is based on the full distribution; it is only the propagation of this distribution through the saturation system that is based on the simplified model.

The 'tuning' of model parameters can be performed through matching all statistical moments of the two distributions, up to and including the fifth moment. The procedure is as follows: firstly the 'true' second and fourth moment of the saturated sum are calculated, and secondly the model parameters (σ_m, c_m) are chosen to match these moments. In order to calculate the 'true' moments and correlation coefficients for each addition, the transfer functions from each primary input to each adder input must be known.

Example 5.8. Since saturation arithmetic is particularly useful for IIR filters, we consider here a second order Direct Form II transposed IIR section, typically used as a building block for larger order IIR filters [Mit98]. Such an IIR section is illustrated in Fig. 3.2, reproduced in Fig. 5.9 for convenience where each addition has been labelled A1 to A4.

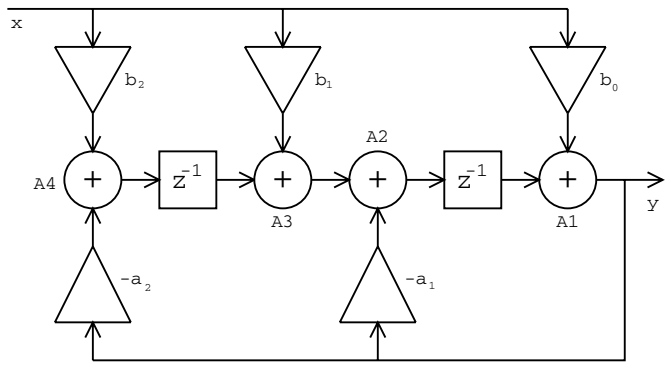


Fig. 5.9. A second order IIR section

The covariance calculations at each adder are provided in (5.6) for completeness. In order to be able to calculate the correlations between inputs to adders for a second order section, we therefore require $r_{xy}[\tau]$ for $\tau = -1, 0, 1, 2$, $r_{xx}[\tau]$ for $\tau = 1, 2$ and $r_{yy}[\tau]$ for $\tau = 1$. These values can be calculated knowing r_{xx} and the transfer functions to each adder input in the system.

$$
\begin{aligned}
&\text{A4: } E\{-b_2x[n]a_2y[n]\} = -b_2a_2r_{xy}[0] \\
&\text{A3: } E\{b_1x[n](b_2x[n-1] - a_2y[n-1])\} = b_1b_2r_{xx}[1] - b_1a_2r_{xy}[1] \\
&\text{A2: } E\{-a_1y[n](b_1x[n] + b_2x[n-1] - a_2y[n-1])\} = \\
&\qquad -a_1b_1r_{xy}[0] - a_1b_2r_{xy}[-1] + a_1a_2r_{yy}[1] \\
&\text{A1: } E\{b_0x[n](-a_1y[n-1] + b_1x[n-1] + b_2x[n-2] - a_2y[n-2])\} = \\
&\qquad -a_1b_0r_{xy}[1] + b_0b_1r_{xx}[1] + b_0b_2r_{xx}[2] - b_0a_2r_{xy}[2]
\end{aligned}
\tag{5.6}
$$

It is important to note that these calculations do not depend in any way on the actual cut-off values for the saturation nonlinearities since they are

calculated from the underlying LTI system. This means that correlation coefficients need only be calculated once for a given system, before entering the optimization loop.

The next phase in the algorithm is to match the calculated moments to a saturated Gaussian model. The matching is a relatively computationally inexpensive operation, because there already exist good intuitive starting guesses for the two parameters, namely $\sigma_m^{(0)} = \sqrt{\sigma_1^2 + \sigma_2^2 + 2\gamma\sigma_1\sigma_2}$, and $c_m^{(0)} = \bar{c}$, where γ is the correlation coefficient of the underlying Gaussian inputs and $\bar{c}$ is the cut-off of the post-adder saturation nonlinearity. Since μ_2 and μ_4 for the model, (5.3) and (5.4) respectively, are differentiable it is straightforward to use a 2-dimensional extension of the Newton-Raphson method [OR70] to select the model parameters. The Jacobian is shown in (5.7).

$$J = \begin{bmatrix} \frac{\partial \mu_2}{\partial \sigma} & \frac{\partial \mu_2}{\partial c} \\ \frac{\partial \mu_4}{\partial \sigma} & \frac{\partial \mu_4}{\partial c} \end{bmatrix} = 2 \begin{bmatrix} \sigma(1 - 2Q(c/\sigma)) - \frac{2c}{\sqrt{2\pi}} \exp(-\frac{c^2}{2\sigma^2}) & 2cQ(c/\sigma) \\ 6\sigma^3(1 - 2Q(c/\sigma)) - \frac{c}{\sqrt{2\pi}} \exp(-\frac{c^2}{2\sigma^2}) & 4c^3Q(c/\sigma) \end{bmatrix} \quad (5.7)$$

Since a saturation system is a nonlinear dynamical system, estimation of γ for each addition is based on the equivalent correlation coefficient in the underlying LTI system. This is a reasonable assumption when the overall level of saturation in the system is small, which will generally be the case unless extremely low SNR is allowable.

A standard iterative update scheme is used, as shown in (5.8).

$$\begin{bmatrix} \sigma_m^{(n+1)} \\ c_m^{(n+1)} \end{bmatrix} = \begin{bmatrix} \sigma_m^{(n)} \\ c_m^{(n)} \end{bmatrix} - J^{-1} \begin{bmatrix} \mu_2^{(n)} \\ \mu_4^{(n)} \end{bmatrix} \quad (5.8)$$

5.4.4 Error Propagation

As with the truncation error estimation described in Section 4.1.2, in order to propagate the saturation error to the outputs of the system, the saturation nonlinearities are linearized as shown for an example in Fig. 5.10. This allows the use of linear system theory to predict the effect of this saturation error on the output signal-to-noise ratio. However unlike the truncation case, it cannot be assumed that the saturation errors injected at various points within the structure are uncorrelated. Recall that for the truncation noise model, this approximation is reasonably valid because it is the least significant bits that are being quantized, which have little relationship to other sets of least significant bits being truncated at other points. The same cannot be said for the saturation nonlinearities. Taking the example in Fig. 5.10, it is clear that when $e_1[t] < 0$, there will be a greater probability of $e_2[t]$ being negative than if $e_1[t] \geq 0$ because there will be a greater probability of the post-adder signal value being large and positive if the primary input is large and positive. In addition, the white assumption on each individual error input is not valid for saturation errors, since the spectrum of the error sequence will clearly depend

heavily on the (possibly coloured) spectrum of the input sequence. These two dimensions of dependence, between pairs of error inputs and over time, require a more sophisticated error estimation model.

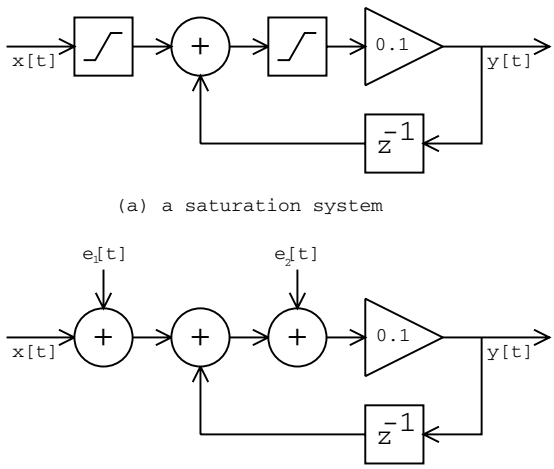


Fig. 5.10. Linearization of the saturation nonlinearities

Estimating the cross-correlation function between saturation inputs is possible but computationally intensive. Although in general there is no solution to this problem in linear time, a bound can certainly be placed on the error power at the system outputs, and this bound can be calculated in linear time.

Claim. In a saturation system let there be a total of p saturation nonlinearities, with corresponding linearized error inputs $e_1, \ldots, e_p$ of standard deviations $\sigma_{e_1}, \ldots, \sigma_{e_p}$. Let us concentrate on a single primary output with error y, and let $H_i(z)$ denote the transfer function from error input e_i to this output. Then (5.9) holds.

$$E\{y^2[t]\} \leq \left(\sum_{i=1}^{p} \sigma_{e_i} \ell_1\{H_i(z)\} \right)^2 \tag{5.9}$$

Proof:
The value of the error y at time index t (at any specific output) is then given by the convolution in (5.10). Here b_i represents the impulse response from error input i to the output in question.

$$y[t] = \sum_{i=1}^{p} \sum_{k=0}^{\infty} e_i[t-k] b_i[k] \tag{5.10}$$

Assuming statistically stationary input signals, we obtain (5.11) for the general saturation-induced error power at the output. Here $r_{e_{i_1}e_{i_2}}$ is the cross-correlation function $r_{e_{i_1}e_{i_2}}[\tau] = E\{e_{i_1}[t]e_{i_2}[t-\tau]\}$.

$$E\{y^2[t]\} = \sum_{i_1=1}^{p}\sum_{i_2=1}^{p}\sum_{k_1=0}^{\infty}\sum_{k_2=0}^{\infty} b_{i_1}[k_1]b_{i_2}[k_2]r_{e_{i_1}e_{i_2}}[k_2-k_1] \tag{5.11}$$

The Cauchy-Schwartz inequality (5.12) [Gar90] may be invoked, allowing (5.11) to be separated, as in (5.13). Here σ_{e_i} represents the standard deviation of error input e_i and $H_i(z)$ is the transfer function from that input to the output in question.

$$\begin{gathered} E^2\{XY\} \le E\{X^2\}E\{Y^2\} \\ \text{for zero-mean random variables } X \text{ and } Y \end{gathered} \tag{5.12}$$

$$\begin{aligned} E\{y^2[t]\} &\le \sum_{i_1=1}^{p}\sum_{i_2=1}^{p}\sum_{k_1=0}^{\infty}\sum_{k_2=0}^{\infty} |b_{i_1}[k_1]b_{i_2}[k_2]|\sigma_{e_{i_1}}\sigma_{e_{i_2}} \\ &\le \left(\sum_{i=1}^{p}\sigma_{e_i}\ell_1\{H_i(z)\}\right)^2 \end{aligned} \tag{5.13}$$

□

Of course such a bound may be more or less close to the empirical error power. The true measure of usefulness for this bound in real applications, is that the circuits synthesized using this measure of error are at least of comparable area to those synthesized using traditional simulation-based average case scaling, preferably smaller. This measure will be tested in Section 5.6.

5.4.5 Reducing Bound Slackness

There are certain transformations which may be performed on the graph representation of a saturation system without affecting the global system behaviour. A saturation nonlinearity may be moved through a constant coefficient multiplication, a move accompanied by a corresponding scaling in the saturation cut-off parameter (Fig. 5.11(a)). In addition, two consecutive nonlinearities can be merged (Fig. 5.11(b)), and multiple nonlinearities following a branching node can be reconfigured (Fig. 5.11(c)). Although these transformations do not result in different system behaviour from an external perspective, the estimated saturation error resulting from the procedure described above can differ, due to different slackness in the Cauchy-Schwartz derived upper bound. It is useful to minimize this slackness. Note that the transformation in Fig. 5.11(c) has been used in Section 5.2 to minimize the implementation cost of a FORK node. However the two applications of this transformation

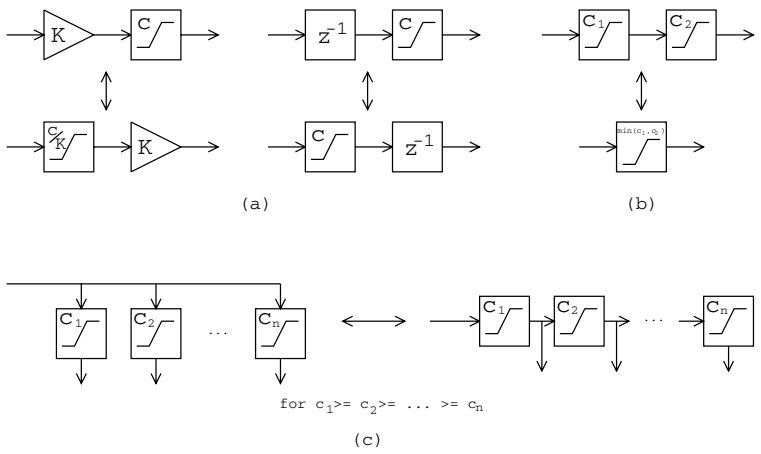


Fig. 5.11. Useful saturation transformations

are entirely separate: applying the transformation for saturation error modelling in no way suggests altering the physical realization of the saturation nonlinearity.

Example 5.9. Applying these transformations to the second order section shown in Fig. 5.12(a) results in the improved saturation system in Fig. 5.12(b); *W1* and *W2* are wiring constructs where the outputs are a permuted version of the inputs, and *S1* and *S2* are saturation constructs with possible forms illustrated in Figs. 5.12(c) and (d) respectively. Applying the saturation noise model to this construction can result in a tighter Cauchy-Schwartz bound.

Claim. For the transformation shown in Fig. 5.11(b), it is always desirable to merge the two nonlinearities.

Proof:

Let the linearized error inputs at the saturation nonlinearities with cut-offs c_1 and c_2 be $e_1[t]$ and $e_2[t]$ of variance σ_{e_1} and σ_{e_2}, respectively. Similarly, let the error input at the saturation nonlinearity with cut-off $\min(c_1, c_2)$ be $e[t]$ of variance σ_e. Since there are no branches between nonlinearities, (5.13) will not be increased by merging iff the error standard deviations obey $\sigma_{e_1} + \sigma_{e_2} \geq \sigma_e$. However it is clear that $e[t] = e_1[t] + e_2[t]$ and therefore application of (5.12) reveals that this is indeed the case since $\sigma_e^2 = \sigma_{e_1}^2 + \sigma_{e_2}^2 + 2E\{e_1[t]e_2[t]\}$ and $\sigma_{e_1} + \sigma_{e_2} \geq \sigma_e \Leftrightarrow \sigma_{e_1}^2 + \sigma_{e_2}^2 + 2\sigma_{e_1}\sigma_{e_2} \geq \sigma_e^2$. □

The transformation shown in Fig. 5.11(c) will not always improve the error bound, since the ℓ_1 scaling from saturation error with cut-off c_i, $1 \leq i < n$, is different in the two cases. Although the error injected due to the saturations with cut-offs c_j, $2 \leq j \leq n$ may be reduced, whether this is offset by the change in scaling is dependent on the system transfer functions. For a general n-way

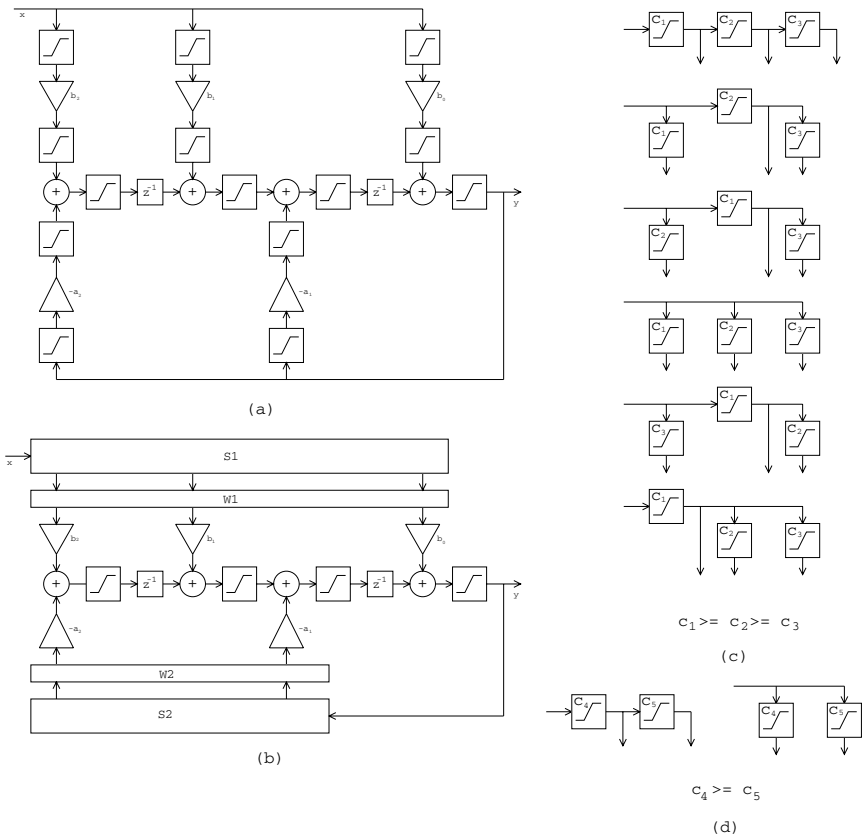


Fig. 5.12. Reducing estimate slackness in a second order IIR filter through saturation transformations: (a) original model (b) transformed model (c) possible forms of *S1* (d) possible forms of *S2*

branching node, the situation becomes more complex still; any tree structure of nonlinearities may be used, so long as the partial order implied by the tree does not violate the numerical order of nonlinearity cut-offs. In practical cases it is often true that the right hand side of Fig. 5.11(c) represents the best error performance. Often when considering saturation arithmetic one is interested in recursive filters, which are typically constructed as a cascade of second order sections. In each second order section, there is one three-way and one two-way branching node. These nodes have sufficiently small outdegree that the search for an optimal ordering can be performed by exhaustive search; the different configurations possible are illustrated in Figs. 5.12(c) and (d) for the three-way and two-way branches, respectively. For more general structures, simple search procedures could be used to determine a reasonable modelling configuration of nonlinearities, or the default configuration of Fig. 5.11(c)

could be used. In any case it is important to note that the configuration does not affect the correctness of the bound in (5.13), only the quality of the bound.

These transformations are combined in Algorithm SlackReduce for a general saturation computation graph. Saturation nonlinearities are propagated 'backwards' through the computation graphs, in the reverse direction to the data-flow. Back propagation allows nonlinearities to come together from different branches of a FORK, and perhaps propagate back through the FORK reducing the Cauchy-Schwartz bound. In contrast forward propagation would not allow merging of nonlinearities, as nonlinearities do not cross adder nodes. In Algorithm SlackReduce the details of FORK nodes are omitted for brevity. Note that for a FORK node a simple heuristic is used: the nonlinearities are sorted in order and the transformation illustrated in Fig. 5.11(c) is applied. This approach is used instead of the exhaustive search (described above for the special case of IIR filters in second order sections), for three reasons. Firstly it has been observed in practice that with respect to the error contribution of the FORK node, this arrangement is the most common arrangement to result in minimal Cauchy-Schwartz bound. Secondly by moving the saturation nonlinearity with maximum cut-off to the inedge of the FORK node it is possible to further apply slack-reducing saturation transformations to predecessors of the FORK node. Thirdly this arrangement is independent of the ℓ_1 scalings, and so the entire computation graph can be treated as one rather than considering each output separately. However because of this heuristic decision it cannot be guaranteed that $\mathbf{E_2} \leq \mathbf{E_1}$ for all input graphs.

Algorithm 5.2
Algorithm SlackReduce
Input: An saturation computation graph $G_S(V, S, C)$,
having Cauchy-Schwartz bound $\mathbf{E_1}$
Output: An equivalent saturation computation graph,
having Cauchy-Schwartz bound $\mathbf{E_2}$
begin
do
foreach $v \in V : \exists((v, v'), c) \in C$ **do**
switch TYPE(v)
case GAIN:
Set $C \leftarrow C \cup \{(\text{inedge}(v), c/\text{COEF}(v))\} - \{((v, v'), c)\}$
case DELAY:
Set $C \leftarrow C \cup \{(\text{inedge}(v), c)\} - \{((v, v'), c)\}$
case FORK:
Apply transformation shown in Fig. 5.11(c),
modify V, S and C accordingly
end switch
end foreach
$C \leftarrow C - \{(j, c) \in C : \exists(j, c') \in C, c' < c\}$
while C has changed during current iteration

end

5.4.6 Error estimation results

This section presents some results from using the error model discussed thus far to predict saturation error variance. Approximately 6000 IIR filters have been generated, each filter having between second and tenth order. The feedback coefficients are generated by randomly choosing complex conjugate pole locations in the range $0 < |z| < 0.998$ and $0 < \arg(z) < \pi$ (uniformly distributed), and the feed-forward coefficients are generated by randomly distributing them between 0 and 1 (uniformly distributed). ℓ_1 scaling is then applied to each filter, and the binary point locations are decided by choosing either those through ℓ_1 scaling or one or two bits beneath this value, each location independently of the other. Probabilities are skewed such that there is a mean of two saturation nonlinearities in each section, in order to agree with typical synthesized circuits. The choice of saturator degree is decided with uniformly distributed probability and independently of all other saturation locations. The predicted error variance is then compared to the observed variance arrived at through a bit-true simulation of the system. Two types of input data are used: independent identically distributed Gaussian input samples, and real speech data [FRE93].

Fig. 5.13 presents several plots of the error bound, derived through the technique presented in this chapter, against a simulation run. Fig. 5.14 presents the data as a histogram of over-estimation ratios, and also illustrates how this histogram changes with the order of the filter modelled for the iid case. It is clear that for the large majority of designs the Cauchy-Schwartz bound provides a value between 0dB and 50dB greater than the simulated result. Although both the slackness of the bound derived, and the mismatch between the saturated Gaussian model and post-addition pdf grow with the order of the filter, it is not a rapid growth. From fourth to tenth order the bulk of overestimation ratios lie in the same range, with little change in the spread of the distribution. Recall that the error estimation is for average-case behaviour, and so the bound is on the *expected* error variance, not the error variance for each specific case. In addition the bound is only exact if the standard deviations of the injected errors are known exactly, whereas in reality they are estimated through modelling signals as saturated Gaussians, as discussed previously. There is some mismatch between the input probability density function and the Gaussian assumption for speech, leading to marginally worse performance in the speech case (speech pdf falls off as $\exp(-\lambda_1|x|)$ rather than the Gaussian $\exp(-\lambda_2 x^2)$ [P50]).

Whether the noise model is sufficient for optimization purposes can only be measured by the quality of the circuits produced by the optimization procedure, which will be discussed in the following section.

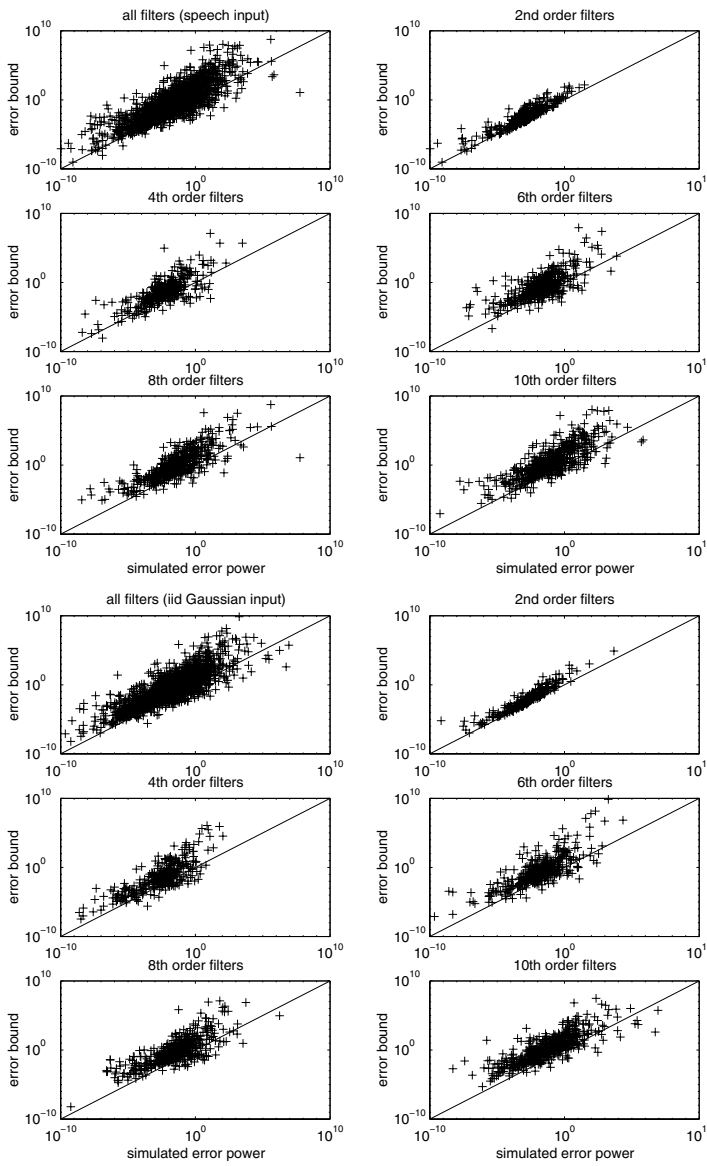


Fig. 5.13. Saturation error model: estimated against simulated error variance for (upper) speech input and (lower) iid Gaussian input

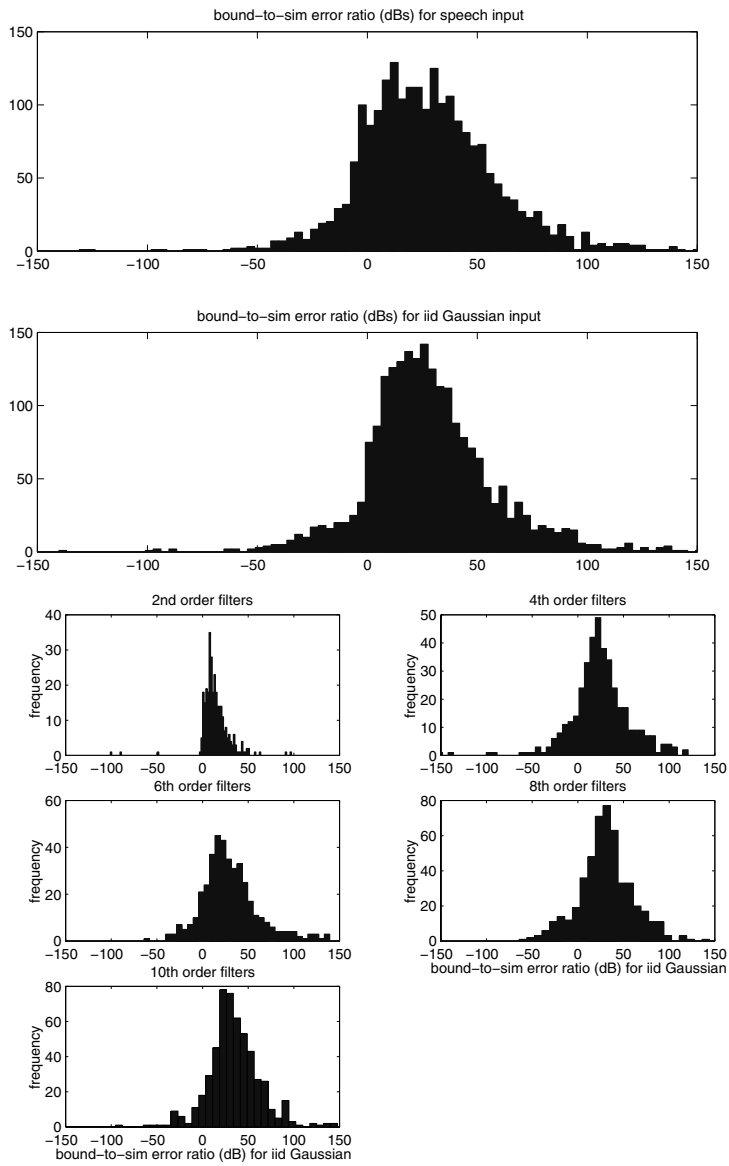


Fig. 5.14. Overestimation ratios for saturation error model (upper) over all filter orders and (lower) for specific filter orders

5.5 Combined Optimization

The method presented in Section 5.4 can be used to form an estimate $\mathbf{E}_G(\mathbf{n}, \mathbf{p}, \mathbf{R})$ of the error variances incurred through the implementation of computation graph G with word-lengths $\mathbf{n}$, binary point locations $\mathbf{p}$, and input correlation matrix $\mathbf{R}[\tau] = E\{\mathbf{x}[t]\mathbf{x}[t-\tau]^T\}$, where $\mathbf{x}$ is the vector of system primary inputs. This estimate may then be compared to the user-specified bounds $\mathcal{E}$ on error variance at each output.

Since both the scaling and the word-length of each signal can have an impact on system error and area, the problem of finding a suitable annotation for the computation graph must now be treated as the combined optimization problem formulated below *c.f.* the Word-length Optimization Problem from Chapter 4.

Problem 5.10 (Combined Word-length and Scaling Optimization). Given a computation graph $G(V, S)$ and correlation matrix $\mathbf{R}$, the COMBINED WORD-LENGTH AND SCALING OPTIMIZATION problem may be defined as to select $(\mathbf{n}, \mathbf{p})$ such that $A_G(\mathbf{n}, \mathbf{p})$ is minimized subject to (5.14).

$$\begin{aligned} &\mathbf{n} \in \mathbb{N}^{|S|} \\ &\mathbf{p} \in \mathbb{Z}^{|S|} \\ &\mathbf{E}_G(\mathbf{n}, \mathbf{p}, \mathbf{R}) \leq \mathcal{E} \end{aligned} \tag{5.14}$$

To solve this optimization problem, it is necessary to modify Algorithm Word-LengthFalling from Section 4.4 to incorporate the optimization of binary point locations. This is performed by Algorithm CombOptAlg, where $\mathbf{1}$ represents a vector of ones of appropriate size, and k is the scaling factor as described in Section 4.4. B_j is a lower bound on the binary point location of signal j. Typically B_j is set to be a fixed, but reasonably large, number of bits beneath the binary point location implied by ℓ_1 scaling. Although B_j is rarely reached in practical designs with realistic error constraints, these bounds are required to theoretically ensure termination of Algorithm CombOptAlg. Reaching $p_j = B_j$ is considered equivalent to $p_j \sim -\infty$. The interpretation of this value is that it is unnecessary to calculate signal j in order to satisfy the error constraints, and so the entire cone of logic creating signal j may be optimized away.

Unlike the error estimation used in Algorithm Word-LengthFalling, the error estimation subroutine used in Algorithm CombOptAlg contains some computationally expensive calculations, namely the post-adder saturation error and pdf estimation, which involves numerical integration and series approximations [AS70]. However the calls to adder saturation error estimation routines in the above algorithm exhibit a high degree of temporal locality. It is highly probable that a given p_j will not change from one iteration to the next, and therefore the same error estimations are often required. Rather than re-calculate these each time, new error estimates for a given adder are only

calculated when the c parameter of either input saturated Gaussian distribution is changed, or the $\bar{c}$ post-adder saturation nonlinearity cut-off is changed (see Sections 5.4.2 and 5.4.3).

Algorithm 5.3
Algorithm CombinedOptHeur
Input: A Computation Graph $G(V, S)$
input correlation matrix $\mathbf{R}$
Output: An Optimized Annotated Computation
Graph $G'(V, S, A)$, $(A = (\mathbf{n}, \mathbf{p}))$
begin
Calculate the variance of each signal and the correlation coefficient between inputs to adders, to be used in all calls to the error estimation subroutine
Set $\mathbf{p} \leftarrow \ell_1$ scaling vector (as described in Section 3.1.1)
Determine u, the minimum uniform word-length satisfying $\mathbf{E}_G(u \cdot \mathbf{1}, \mathbf{p}, \mathbf{R}) \leq \mathcal{E}$
Set $\mathbf{n} \leftarrow ku \cdot \mathbf{1}$
do
Condition the graph $G'(V, S, A)$
Set currentcost $\leftarrow A_G(\mathbf{n}, \mathbf{p})$
foreach signal $j \in S$ **do**
Set bestmin $\leftarrow$ currentcost
Determine $w \in \{2, ..., n_j\}$, if such a w exists, such that
$\mathbf{E}_G([n_1 \dots n_{j-1}\ w\ n_{j+1} \dots n_{|S|}]^T, \mathbf{p}, \mathbf{R}) \leq \mathcal{E}$ and
$\mathbf{E}_G([n_1 \dots n_{j-1}\ (w-1)\ n_{j+1} \dots n_{|S|}]^T, \mathbf{p}, \mathbf{R}) \not\leq \mathcal{E}$
If such a w exists, set minval $\leftarrow A_G([n_1 \dots n_{j-1}\ w\ n_{j+1} \dots n_{|S|}]^T, \mathbf{p})$
If no such w exists, set minval $\leftarrow A_G([n_1 \dots n_{j-1}\ 1\ n_{j+1} \dots n_{|S|}]^T, \mathbf{p})$
if minval $<$ bestmin **do**
Set bestsig $\leftarrow j$, bestmin $\leftarrow$ minval, vartype $\leftarrow$WORD-LENGTH
end if
Determine $x \in \{B_j + 1, ..., p_j - 1, p_j\}$, if such an x exists, such that
$\mathbf{E}_G(\mathbf{n}, [p_1 \dots p_{j-1}\ x\ p_{j+1} \dots p_{|S|}]^T, \mathbf{R}) \leq \mathcal{E}$ and
$\mathbf{E}_G(\mathbf{n}, [p_1 \dots p_{j-1}\ (x-1)\ p_{j+1} \dots p_{|S|}]^T, \mathbf{R}) \not\leq \mathcal{E}$
If such an x exists, set minval $\leftarrow A_G(\mathbf{n}, [p_1 \dots p_{j-1}\ x\ p_{j+1} \dots p_{|S|}]^T)$
If no such x exists, set minval $\leftarrow A_G(\mathbf{n}, [p_1 \dots p_{j-1}\ B_j\ p_{j+1} \dots p_{|S|}]^T)$
if minval $<$ bestmin **do**
Set bestsig $\leftarrow j$, bestmin $\leftarrow$ minval, vartype $\leftarrow$SCALING
end if
end foreach
if bestmin $<$ currentcost
if vartype $=$ WORD-LENGTH
$n_{\text{bestsig}} \leftarrow n_{\text{bestsig}} - 1$
else
$p_{\text{bestsig}} \leftarrow p_{\text{bestsig}} - 1$
while bestmin $<$ currentcost
end

5.6 Results and Discussion

Algorithm CompOptAlg has been implemented in MATLAB [MAT] in order to leverage its numerical integration routines, at the cost of increased execution time over an extension to the C implementation used in Chapter 4. However all execution runs discussed in this section are completed within 10 minutes on a Pentium III 450MHz, compared to 4 minutes for a Matlab implementation of Algorithm Word-LengthFalling and 10 seconds for calculation of the optimum uniform word-length. In addition, the execution time of Algorithm CombOptAlg will grow at the same rate as that of Algorithm Word-LengthFalling.

To illustrate the applicability of saturation arithmetic optimization to different types of design, two 4^{th} order IIR filers are generated. One is a narrow bandpass elliptic filter, and one is a lowpass elliptic filter. Both filters are to be driven with a speech input [FRE93]. Clearly the narrow bandpass filter will have very high theoretical peak values at internal signals, as would be determined by ℓ_1 scaling. However an input signal that would cause internal signals to reach these peaks is unlikely to be present in a speech signal, which contains a wide range of frequency components. Thus the ratio between ℓ_1 peak value at any signal and the peak value reached during a simulation run is likely to be relatively large. In contrast the equivalent ratios in the lowpass filter are likely to be much more modest.

Fig. 5.15 shows an execution trace of Algorithm CombOptAlg executing on the bandpass filter, illustrating the change in system area and error variance as the algorithm refines the solution. Two plots are superimposed in the error variance trace: these are the error due to truncation alone, and the overall error. It is clear that until iteration 42, the contribution of saturation to the overall error is negligible. At this iteration the decision is made to reduce a binary point location of one of the signals, resulting in a significant saturation error. However by the end of the optimization run it is clear that the truncation error yet again dominates the saturation error.

5.6.1 Area Results

The standard approach of using simulation to determine signal scaling aims to avoid overflow for the specific input sequences provided, though not necessarily for all input sequences. The scaling of each signal through the use of simulation, and hence the area of a simulation-scaled system, therefore depends on the length of input sequence used for simulation. The longer the input sequence, the more the 'tails' of the pdf of an internal signal are likely to be encountered. Simulating the system on a short input sequence may result in a smaller area at the cost of a larger saturation error on unencountered input sequences, when compared to lengthy simulation runs.

In contrast, Algorithm CombOptAlg tolerates overflow errors if these errors help to achieve a small implementation cost, and is able to estimate the

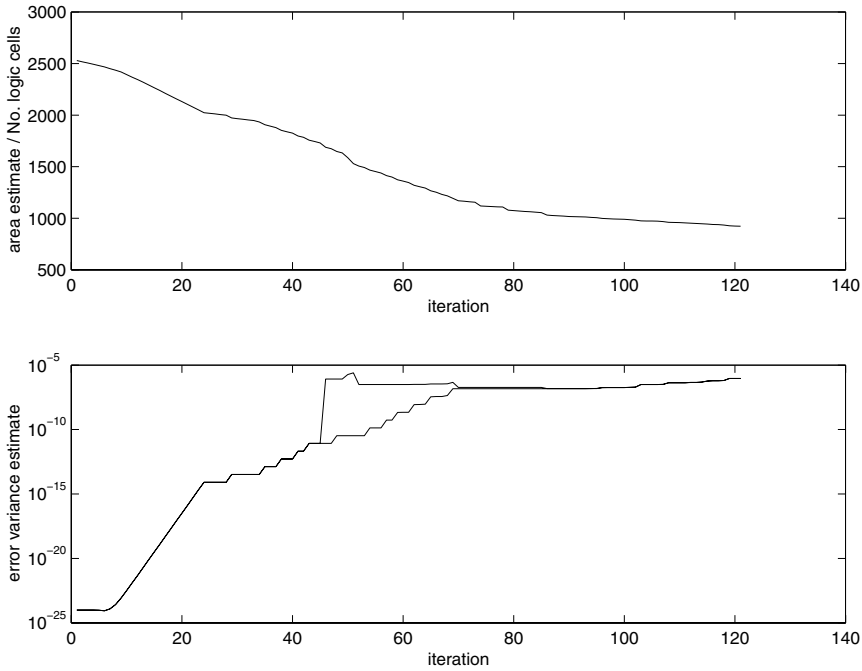


Fig. 5.15. An execution trace of Algorithm CombOptAlg executed on a 4^{th} order narrow bandpass elliptic IIR filter

severity of such errors for the average input sequence. For this reason, it is to be expected that Algorithm CombOptAlg may generate systems of smaller overall area than those generated through a simulation-based scaling followed by Algorithm Word-LengthFalling for word-length optimization.

Fig. 5.16 shows a comparison of different approaches to area / error tradeoffs for the bandpass filter example. Plots (e) and (f) correspond to the approach detailed in Section 4.4 based on ℓ_1 scaling followed by word-length optimization. Simulation-based scaling results are illustrated as regions (a) and (c): the upper curve in the region corresponds to simulation with a relatively long input sequence (10^5 samples at 8kHz, a spoken announcement), whereas the lower curve corresponds to simulation with a relatively short input sequence ($3 \cdot 10^3$ samples at 8kHz, a spoken word). Region (a) illustrates a system that has been simulation-scaled, and with the optimum uniform word-length. Region (c) illustrates a system that has been simulation-scaled and then word-length optimized using Algorithm Word-LengthFalling. Region (b) illustrates the overlap between regions (a) and (c). Finally plot (d) corresponds to Algorithm CombOptAlg, a combined scaling and word-length optimization procedure.

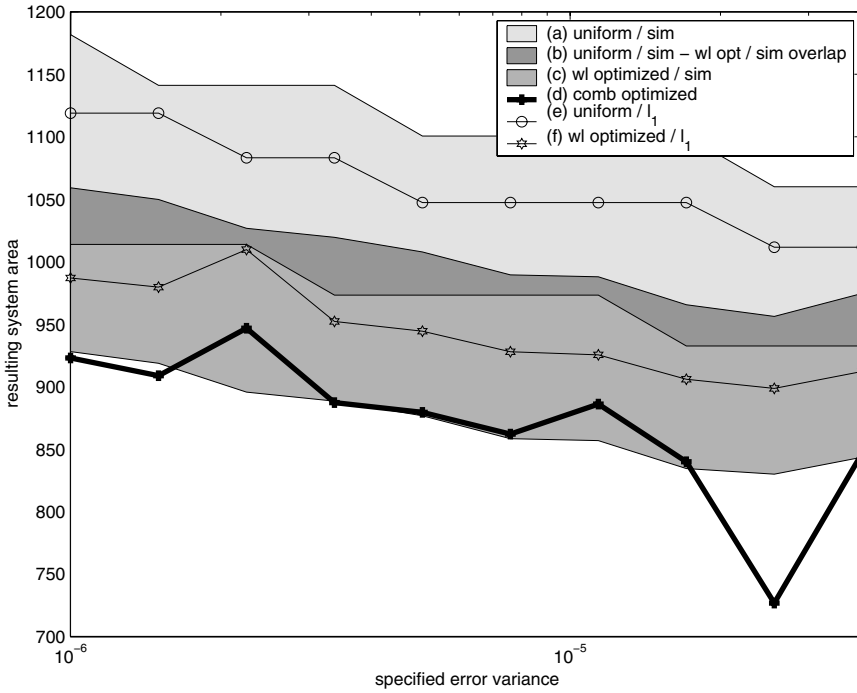


Fig. 5.16. Comparison of different design approaches for trading-off system area and error, for a 4^{th} order narrow bandpass elliptic IIR filter

There are several important pieces of information revealed by Fig. 5.16. It may seem somewhat surprising that the upper line of region (a) has a larger area than plot (e), since (e) represents *worst case* ℓ_1-scaled results. (Similarly comparing region (c) to plot (f)). However recall that there is an area overhead associated with saturation arithmetic, and when performing simulation-based scaling saturators must be introduced when a signal is converted from (n, p) to $(n-k, p-k)$ (Section 5.2). For the ℓ_1 case, the essentially cost-free operation of inverse sign-extension (Section 4.1.1) will suffice. This overhead is sufficient to make the simulation-scaled system larger than the equivalent ℓ_1-scaled system, when a long simulation run is used. The overhead appears to be a price worth paying when the short simulation run is used, however for alternative input sequences the saturation error may violate the user-specified error constraint. Plot (d) demonstrates that superior area results can be achieved through Algorithm CombOptAlg, approximately matching the best-case of region (c), but not limited to the specific small input sequence used. In summary, Algorithm CombOptAlg has resulted in average system area for Altera Flex10k between 10.6% and 21.8% less than the standard saturation arithmetic approach of using a single uniform word-length and simulation to de-

termine signal scaling. In addition average area reductions of 0.3% to 13.3% have been achieved compared to simulation-scaling followed by application of Algorithm Word-LengthFalling. The area has been reduced by 18% on average compared to uniform word-length and ℓ_1 scaling, and 7.9% compared to ℓ_1 scaling followed by application of Algorithm Word-LengthFalling. These data are summarized in Table 5.1.

Table 5.1. Average percentage improvement of Algorithm CombOptAlg over alternative approaches for a 4th order narrow bandpass elliptic IIR filter

	simulation scaling		ℓ_1 scaling
	short input sequence	long input sequence	
uniform word-length	10.6%	21.8%	18.0%
optimized word-length	0.3%	13.3%	7.9%

Fig. 5.17 shows the equivalent comparison for the lowpass filter example. In this case plot (d) lies consistently beneath region (a) and plot (e) lies consistently beneath region (b), illustrating that the saturation arithmetic area overhead is very significant for this example. Indeed, high quality solutions can be obtained through the procedure described in Section 4.4 alone, without the need for saturation arithmetic. Plot (c), representing Algorithm CombOptAlg performs consistently well, matching Algorithm Word-LengthFalling in most cases and improving upon it in other cases. Thus by *judicious* placement of saturators, it is even possible to use saturation arithmetic to improve the area consumption of this example. In summary, Algorithm CombOptAlg has resulted in average system area between 12.2% and 20.0% less than the standard saturation arithmetic approach of using a single uniform word-length and simulation to determine signal scaling. In addition average reductions of 5.7% to 13.0% have been achieved compared to simulation-scaling followed by application of Algorithm Word-LengthFalling. The area has been reduced by 11% on average compared to uniform word-length and ℓ_1 scaling, and 0.3% compared to ℓ_1 scaling followed by application of Algorithm Word-LengthFalling. These data are summarized in Table 5.2.

Table 5.2. Average percentage improvement of Algorithm CombOptAlg over alternative approaches for a 4th order lowpass elliptic IIR filter

	simulation scaling		ℓ_1 scaling
	short input sequence	long input sequence	
uniform word-length	12.2%	20.0%	11.0%
optimized word-length	5.7%	13.0%	0.3%

Area results have thus far been illustrated for fixed system function, while varying the specification on maximum error variance. It has been demon-

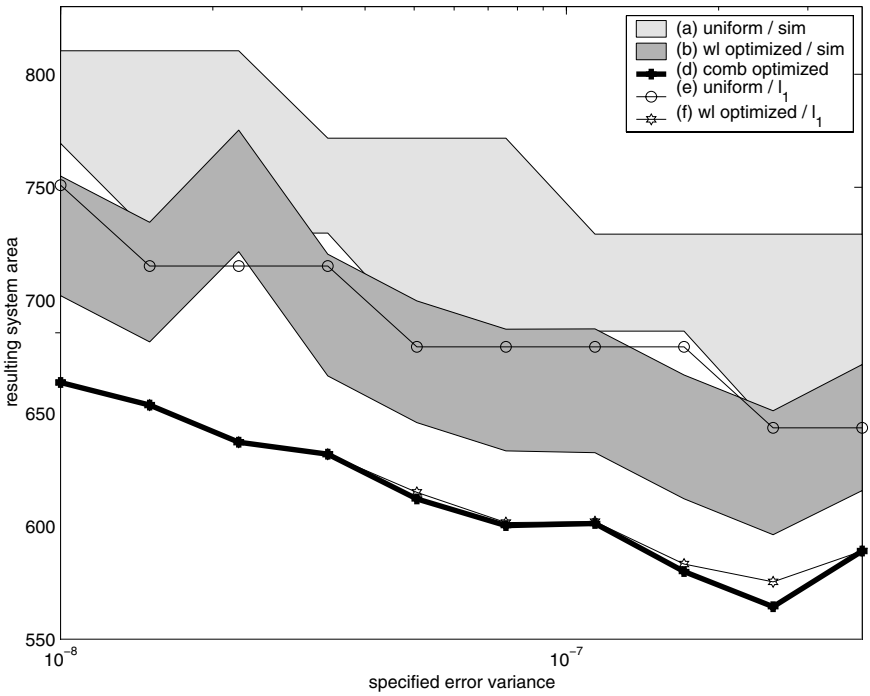


Fig. 5.17. Comparison of different design approaches for trading-off system area and error, for a 4^{th} order lowpass elliptic IIR filter

strated that significant area reductions can be achieved for the narrow bandpass filter example. Fig. 5.18 illustrates how the area consumption of second order autoregressive filters varies with the location of their complex conjugate pole on the z-plane, for fixed error specification. Compared to ℓ_1 scaling, area savings have been achieved using the techniques developed in this chapter for systems with poles with magnitude greater than approximately 0.9.

5.6.2 Clock frequency results

As noted in Section 5.2, there are also significant timing overheads associated with the use of saturation arithmetic. While Algorithm CombOptAlg does not explicitly consider circuit speed, it is instructive to place the points on Fig. 5.16 on a speed / area design-space diagram. This is shown in Fig. 5.19, where the short simulation run results are used for representation of simulation-scaled systems. There are ten graphs, corresponding to the ten error variance specifications in Fig. 5.16. Speed estimates are obtained from Altera MaxPlus II [MAX] on the fully placed and routed design in an Altera Flex10kRC240-3 device. A Pareto-optimal point [DeM94] is a point in

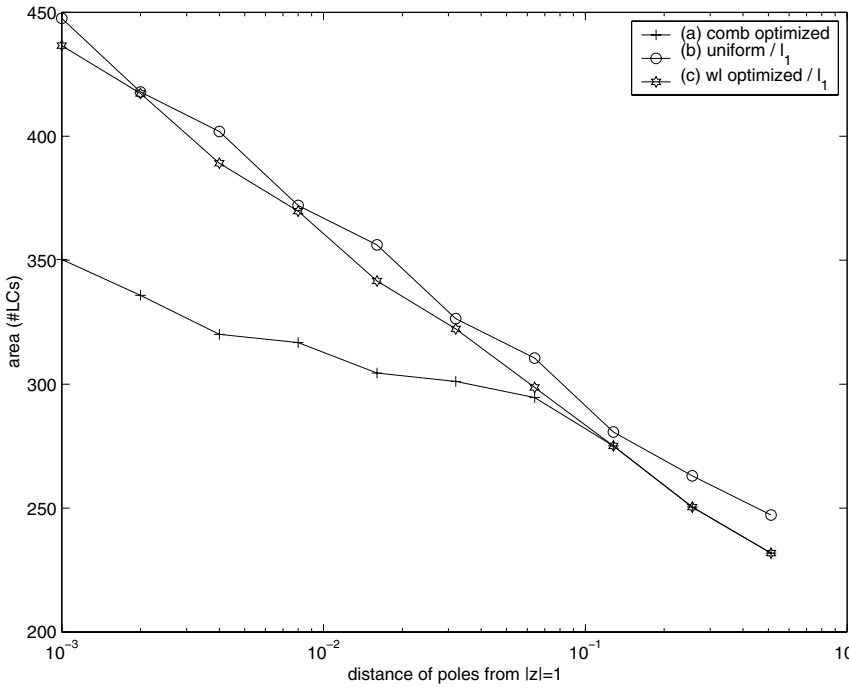


Fig. 5.18. Variation of system area with pole location

the design space that is not dominated in all design objectives by any other design-space point. The Pareto optimal points in Fig. 5.19 are joined by solid lines.

The results of Fig. 5.19 demonstrate that although the difference in area is small between short-run simulation-based scaling word-length-optimized systems and those resulting from Algorithm CombOptAlg, there is a significant speed difference. The source of this consistent speedup, averaging 27.6% over uniform word-length and 23.7% over optimized word-length structures, is illustrated in Fig. 5.20 where the saturator locations and degrees are illustrated for a single optimization example.

Comparing Figs. 5.20(a) and (b), simulation-based scaling has resulted in a large number of low degree saturators. In contrast the optimized saturators are few in number, but are generally of higher degree. Although aiming to reduce system implementation area, Algorithm CombOptAlg has also resulted in significant speedup over simulation-based approaches by using only a small number of saturators. The Cauchy-Schwartz bound tends to drive the solution towards using fewer saturators in order to minimize the potential error cross-correlation effects.

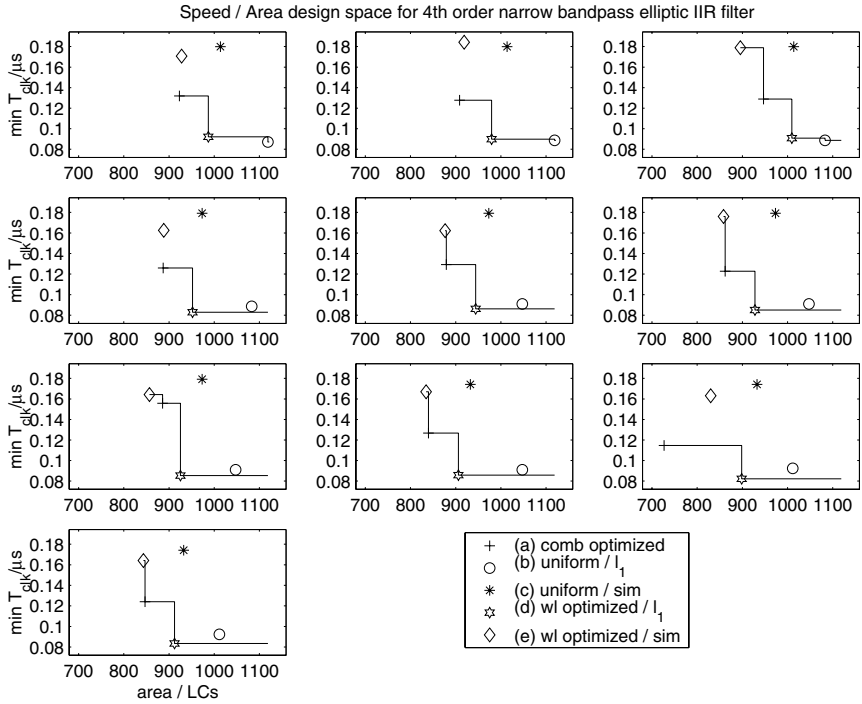
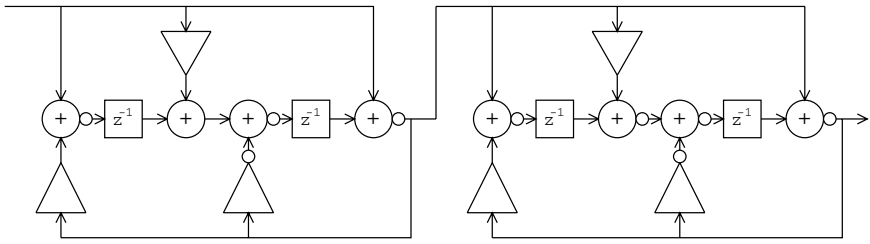


Fig. 5.19. Alternative design approaches, and their speed / area design-space locations

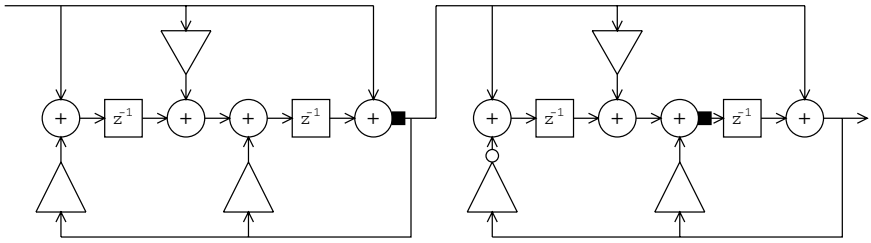
5.7 Summary

This chapter has presented a novel technique for design automation of saturation arithmetic systems. An analytic saturation noise estimation method has been presented, based on the introduction of the saturated Gaussian distribution and a linearization of the saturation nonlinearities. In contrast to truncation and rounding, auto- and cross-correlations between linearized saturation nonlinearities have been accounted for using a bound derived through the Cauchy-Schwartz inequality. Techniques have been presented to reduce the slackness associated with such a bound.

The heuristic presented in Section 4.4 has been extended to incorporate combined scaling and word-length optimization. The results of such an optimization have been discussed for real examples of DSP systems and contrasted with more traditional approaches to scaling optimization. It has been shown that allowing rare saturation errors can result in fast and small implementations of IIR filters when the poles of the filter are close to the unit circle. Improvements have been achieved of up to 8% in area and 24% in speed over and above the improvements generated through the techniques of Chapter 4.



(a) scaling determined by simulation



(b) scaling determined by combined optimization

○ **Saturator, degree 1**
● **Saturator, degree 2**
■ **Saturator, degree 3**

Fig. 5.20. Saturator locations and degrees for the 4^{th} order narrow bandpass elliptic IIR filter

This page intentionally left blank

6

Scheduling and Resource Binding

This chapter considers the problem of architectural synthesis for multiple word-length systems. The constraint of a dedicated resource for each operation, implicit in the previous chapters, is relaxed so as to allow resource sharing to be considered.

Section 6.1 contains an overview of how the material in this chapter contributes to an overall design flow for our approach. Section 6.2 provides a concrete formulation of the problem to be addressed. Section 6.3 introduces an Integer Linear Programming (ILP) approach to the problem. However the ILP approach is not practical for large problems, due to the computational complexity associated with solving the ILP model. This drawback is the motivation for a polynomial-time heuristic algorithm presented in Section 6.4. Synthesis results are reported and discussed in Section 6.5 before summarizing the chapter in Section 6.6.

6.1 Overview

The work in this chapter may be considered as a 'post-processing' step to the word-length and scaling determination procedures discussed in Chapters 4 and 5, as illustrated in Fig. 6.1. Ideally these two sub-problems should be optimized as a single step, however the approach taken in this book is to solve the two sub-problems separately. There are some advantages from breaking the problem in this way. Firstly it means that the work described in this chapter applies to all systems, irrespective of their special properties required for the approaches detailed in Chapters 3–5. Secondly the algorithmic complexity of the synthesis algorithms is reduced by breaking the problem into manageable pieces. The dashed line in Fig. 6.1 indicates that one possible approach to improve on straight-forward application of architectural synthesis, when considering the combined problem, would be to use feedback from architectural results to put further constraints on the word-length determination phase.

Such constraints could emphasize the sharing of particular resources by reducing the number of free word-length variables in the word-length optimization problem. However that approach is not addressed in this book. The present chapter is concerned only with the architectural synthesis block in Fig. 6.1.

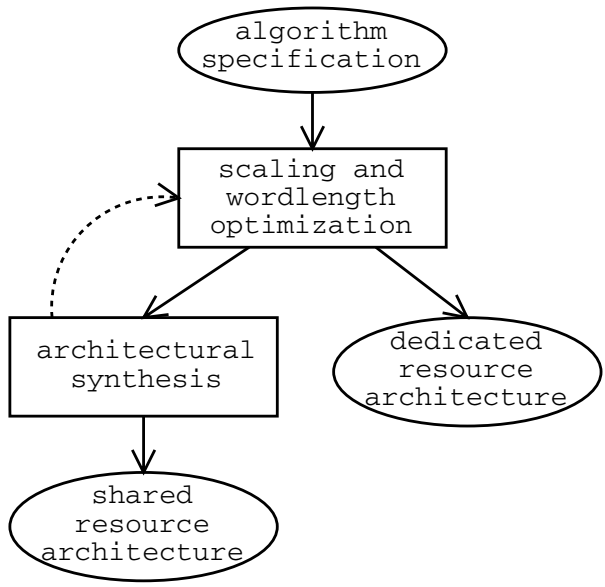


Fig. 6.1. Overall design flow for general resource binding architectures

6.2 Motivation and Problem Formulation

The synthesis problems addressed in Chapters 4 and 5 both assume that the architecture resulting from the synthesis process utilizes a dedicated resource binding. In a dedicated resource binding, each unit sample delay is mapped to a distinct register, each addition to a distinct adder, and each constant-coefficient multiplication to a distinct constant-coefficient multiplier. One of the problems with this approach is that for area-limited architectures such as FPGAs, very large designs may not be feasible. A solution to this problem is to share some of the resources between operations, multiplexing the inputs to these resources over time, and thus allowing a speed / area tradeoff.

The problems of resource allocation (deciding how many resources of a particular type should be used), resource binding (deciding which operation is to be executed on which resource), and scheduling (deciding at which clock period, or 'time step', each operation should execute) have all been well studied [Cam90, McF90, DeM94, Lin97]. However such work in the open literature

has invariably considered all operations of a particular type, such as 'multiply' or 'add', to have identical implementations or at least an identical library of possible implementations [JPP88, Jai90, IM91, HO93].

There has been very little previous or concurrent research [KS98, CCL00b, KS01] on architectural synthesis for multiple word-length systems. The multiple word-length paradigm has a significant impact on the traditional problems of high-level synthesis, arising from two factors. Firstly, each computational unit of a specific type, for example 'multiply', cannot be assumed to have equal cost in a multiple word-length implementation, since area scales with operator word-length. This issue has been considered by both [KS98, KS01] and [CCL00b]. Secondly, the choice of word-length for an operation can impact on the latency of that operation. For instance, larger bit-parallel multipliers may have longer latency than smaller bit-parallel multipliers. The existence of multiple word-lengths therefore complicates the resource binding problem, and also increases the interaction between binding and scheduling of operations.

Only a single iteration of the specified algorithm, and simplified cost models are considered in this chapter. Each arithmetic operation v is associated with a word-length. For an adder, a word-length is a positive integer $b^A(v)$, representing the bit-width of the core (integer) adder required in order to implement the multiple word-length addition (see Section 4.2). For a multiplier, a word-length is a pair $b^M(v)$, representing the two input bit-widths of the core (integer) multiplier required (also see Section 4.2). The elements of the pair are arranged such that the first element is always greater than or equal to the second element. Thus for the remainder of this chapter, it is sufficient to refer to a '10-bit addition' or a '23×12-bit multiplication'.

These concepts are formalized in the definition of a sequencing graph given in Definition 6.1.

Definition 6.1. A *sequencing graph* $P(V, D)$ is a directed acyclic graph (DAG), representing the data flow during a single iteration of an algorithm. The set V is in one-to-one correspondence with the set of operations. The directed edge set $D \subset V \times V$ is in correspondence with the flow of data from one operation to another.

As with Definition 2.1, a TYPE function exists for elements of V (6.1). Each node $v \in V$ with $\text{TYPE}(v) = \text{MULT}$ has a word-length tuple $b^M(v) = (p_v, q_v) \in \mathbb{N}^2$ with $p_v \geq q_v$ and each node $v \in V$ with $\text{TYPE}(v) = \text{ADD}$ has a single word-length $b^A(v) \in \mathbb{N}$.

$$\text{TYPE} : V \rightarrow \{\text{ADD}, \text{MULT}\} \tag{6.1}$$

Example 6.2. A simple sequencing graph is illustrated in Fig. 6.2. The node set consists of five multiplications and four additions.

The MULTIPLE WORD-LENGTH ARCHITECTURAL SYNTHESIS problem may now be defined in Problem 6.3. Note that the creation of structural hardware

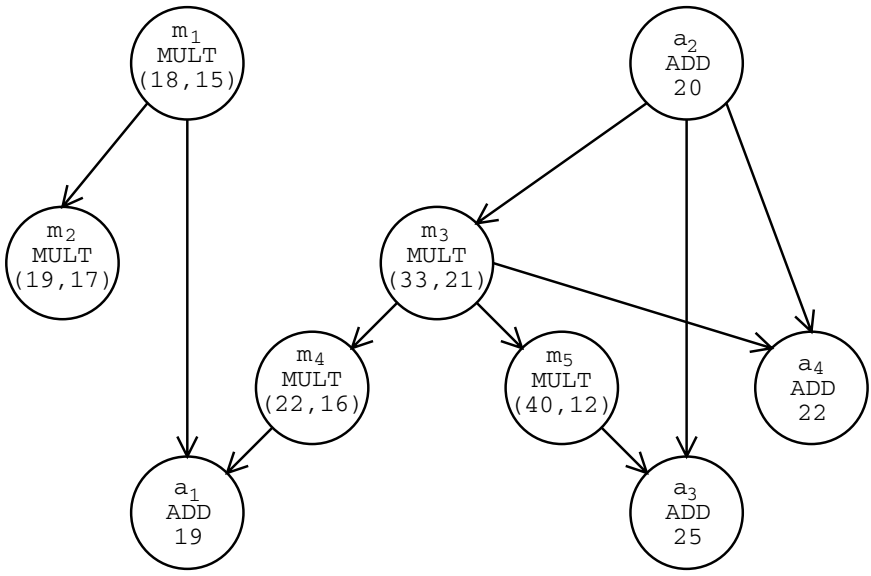


Fig. 6.2. A simple sequencing graph: node labels indicate node name, node type, and word-length

description language from the resulting information is not considered in this chapter, as such techniques are well known [PWB92].

Problem 6.3 (MULTIPLE WORD-LENGTH ARCHITECTURAL SYNTHESIS). Given a sequencing graph $P(V, D)$, and a specified maximum latency λ, determine

- a set of resources and their associated word-lengths Y
- a mapping from operation to resource $\mathcal{R} : V \to Y$
- a mapping from operation to time-step $\mathcal{S} : V \to \mathbb{N} \cup \{0\}$

such that the area consumed by the set of resources is minimal, all data-dependencies are preserved, no resource executes more than one operation in each clock cycle, and the entire sequencing graph completes within λ cycles.

Each resource $y \in Y$ has a word-length $b(y) \in \mathbb{N}$ for an adder and $b(y) \in \mathbb{N}^2$ for a multiplier resource. For each target architecture, it is necessary to construct an empirically derived function which determines the required number of clock cycles for each multi-cycle resource word-length r. This construction has been performed for the Sonic architecture [HSCL00] for word-lengths up to 64-bits, the results of which are given in (6.2).

$$L(r) = \begin{cases} \lceil (p+q)/8 \rceil, & r = (p,q) \in \mathbb{N}^2 \\ 2, & r \in \mathbb{N} \end{cases} \tag{6.2}$$

The cost function used for each resource is given in terms of its word-length in (6.3), where $k \in \mathbb{R}$ is a technology-dependent constant representing the relative cost of adder and multiplier implementations.

$$\text{cost}(r) = \begin{cases} p \cdot q, \, r = (p, q) \in \mathbb{N} \times \mathbb{N} \\ k \cdot r, \, r \in \mathbb{N} \end{cases} \tag{6.3}$$

Example 6.4. As a motivational example, consider again the sequencing graph representing data-dependencies shown in Fig. 6.2. An area-optimal schedule, binding and word-length selection for this sequencing graph is illustrated in Fig. 6.3 for the case $k = 1$ and no operation pipelining. This resource allocation consists of two adders: one of 25-bits and one of 19-bits, and three multipliers: one is a 19×17-bit multiplier, one a 33×17-bit multiplier, and one a 40×12-bit multiplier. The graphical matrix illustrates which resource is being used by which operation at which time step.

Note that in Fig. 6.3 resources can perform operations up to the word-length of the resource, even if implementation in a larger resource leads to a longer latency than a 'tight-fitting' resource would require. For example operation m_4 is implemented in a resource of latency 7 cycles, although its word-length only requires a 22×16-bit multiplier which would take only 5 cycles to complete. This 'stretching' of operations that are not on the critical path can conceivably lead to significantly reduced area, by exposing possibilities for resource sharing.

6.3 Optimum Solutions

Integer Linear Programming (ILP) [GN72] has been used in high-level synthesis for some time [HLH91, Ach93, LP93, DeM94, LMD94]. This section presents an extension to these ILP formulations in order to solve Problem 6.3 [CCL00c]. Formulation as an ILP is useful from an analytical perspective, because it formalizes the problem and its constraints. In addition, for small problem instances, ILP solvers such as `lp_solve` [Sch97] may be used to obtain globally optimum solutions to the synthesis problem. These optimum solutions are valuable references for comparison with heuristic approaches.

6.3.1 Resources, Instances and Control Steps

Before presenting the ILP formulation of Problem 6.3, it is necessary to define certain quantities and notations, to be used in the following sections.

The starting point for the ILP approach is a sequencing graph $P(V, D)$ and a target overall latency constraint λ. The latency constraint corresponds to a user-specified upper bound on the number of clock cycles which may elapse

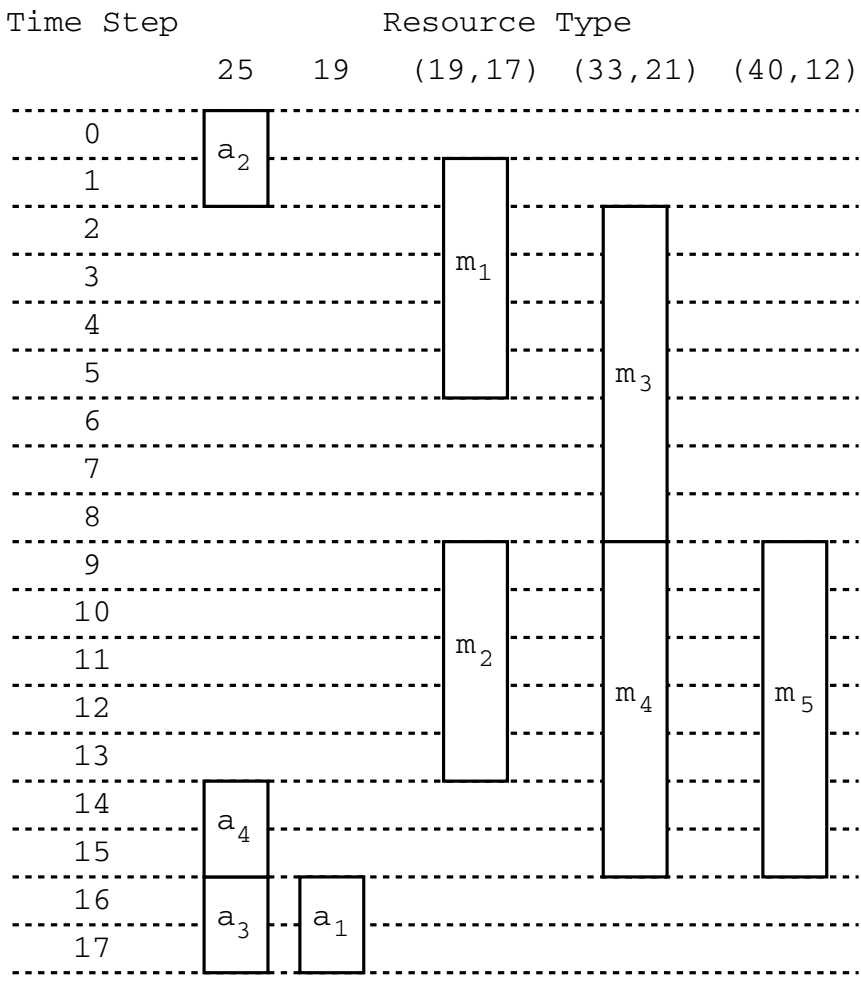


Fig. 6.3. An optimum scheduling, resource binding, and word-length selection for the sequencing graph illustrated in Fig. 6.2

between the start of the first operation in the sequencing graph, and the end of the last operation in the sequencing graph.

Let $V_m = \{v \in V : \text{TYPE}(v) = \text{MULT}\}$ and similarly let $V_a = \{v \in V : \text{TYPE}(v) = \text{ADD}\}$.

Any resource of the correct type, and large enough in word-length, can perform an operation. For example a resource type (p, q) can perform any $p' \times q'$-bit multiplication, so long as $p' \leq p$ and $q' \leq q$. However the search-space for area-efficient implementations may be trimmed significantly by observing that area-optimal resource bindings will only ever use the resource word-length that is just large enough to cover all operations assigned to that

resource. For an adder to which operations $V' \subseteq V$ have been assigned, this corresponds to a word-length of $\max_{a \in V'} b^A(a)$. For a multiplier to which operations $V' \subseteq V$ have been assigned, this corresponds to a word-length of $(\max_{m \in V'} \pi_1\left(b^M(m)\right), \max_{m \in V'} \pi_2\left(b^M(m)\right))$, where $\pi_1(\cdot)$ and $\pi_2(\cdot)$ are the projection operators (6.4).

$$\begin{aligned} \pi_1(p,q) &= p \\ \pi_2(p,q) &= q \end{aligned} \tag{6.4}$$

There are therefore only certain resource types which can arise from the optimal sharing of resources between operations. Let $R^A(a)$ denote the set of adders which could implement the addition $a \in V_a$ and $R^M(m)$ denote the set of multipliers which could implement the multiplication $m \in V_m$. Then $R^A(a)$ and $R^M(m)$ are given by (6.5) and (6.6) respectively. Together, these resource types form a resource-set $R(v)$ for each operation $v \in V$ (6.7). R denotes the set of all such resource types (6.8).

$$R^A(a) = \{p \in b^A(V_a) : p \geq b^A(a)\} \tag{6.5}$$

$$\begin{aligned} R^M(m) = \{(p,q) | \exists (p,b) \in b^M(V_m), \exists (c,q) \in b^M(V_m) : \\ p \geq c \ \wedge \ q \geq b \ \wedge \ p \geq d \ \wedge \ q \geq e \text{ where } (d,e) = b^M(m)\} \end{aligned} \tag{6.6}$$

$$R(v) = \begin{cases} R^A(v), & v \in V_a \\ R^M(v), & v \in V_m \end{cases} \tag{6.7}$$

$$R = \bigcup_{v \in V} R(v) \tag{6.8}$$

An upper bound $I(r)$ may be placed on the number of instances of each resource type that could arise. For an adder resource, there can be as many instances of a w-bit adder as there are w-bit addition operations (6.9). For a multiplier resource, each $p \times q$-bit resource can only arise due to resource sharing of a $p \times b$-bit and a $c \times q$-bit multiplication with $p \geq c$ and $q \geq b$. The number of these pairings is bounded by (6.10).

$$I(r) = |\{a \in V_a : b^A(a) = r\}|, \text{ for } r \in R(V_a) \tag{6.9}$$

$$\begin{aligned} I(p,q) = \min \{ \ & |\{m \in V_m : q \geq e \text{ where } (p,e) = b^M(m)\}|, \\ & |\{m \in V_m : p \geq d \text{ where } (d,q) = b^M(m)\}| \ \}, \\ & \text{for } (p,q) \in R(V_m) \end{aligned} \tag{6.10}$$

From (6.2) it is possible to define the maximum latency $\ell_{\max}(v)$ and minimum latency $\ell_{\min}(v)$ of each operation $v \in V$ according to (6.11, 6.12).

$$\ell_{\min}(v) = \min_{r \in R(v)} L(r) \tag{6.11}$$

$$\ell_{\max}(v) = \max_{r \in R(v)} L(r) \tag{6.12}$$

In order to bound the possible execution control steps of each operation, it is necessary to utilize as-soon-as-possible (ASAP) and as-late-as-possible (ALAP) scheduling [Par99], provided in Algorithms ASAP and ALAP for completeness.

Algorithm 6.1
Algorithm ASAP
Input: A sequencing graph $P(V, D)$ and a latency $\ell(v)$ for each operation $v \in V$
Output: A schedule $\mathcal{S} : V \to \mathbb{N} \cup \{0\}$
begin
 mark$(v) \leftarrow$ FALSE for all $v \in V$
 foreach $v \in V : \nexists(v', v) \in D$ **do**
  $\mathcal{S}(v) \leftarrow 0$
  mark$(v) \leftarrow$ TRUE
 end foreach
 do
  foreach $v \in V : \forall(v', v) \in D, \text{mark}(v') =$ TRUE **do**
   $\mathcal{S}(v) \leftarrow \max_{(v',v) \in D} \{\mathcal{S}(v') + \ell(v')\}$
   mark$(v) \leftarrow$ TRUE
  end foreach
 while $\exists v \in V : \text{mark}(v) =$ FALSE
end

Algorithm 6.2
Algorithm ALAP
Input: A sequencing graph $P(V, D)$, latency constraint λ, and
 a latency $\ell(v)$ for each operation $v \in V$
Output: A schedule $\mathcal{S} : V \to \mathbb{N} \cup \{0\}$
begin
 mark$(v) \leftarrow$ FALSE for all $v \in V$
 foreach $v \in V : \exists(v, v') \in D$ **do**
  $\mathcal{S}(v) \leftarrow \lambda - \ell(v)$
  mark$(v) \leftarrow$ TRUE
 end foreach
 do
  foreach $v \in V : \forall(v, v') \in D, \text{mark}(v') =$ TRUE **do**
   $\mathcal{S}(v) \leftarrow \min_{(v,v') \in D} \{\mathcal{S}(v')\} - \ell(v)$
   mark$(v) \leftarrow$ TRUE
  end foreach
 while $\exists v \in V : \text{mark}(v) =$ FALSE
end

Consider performing ASAP and ALAP scheduling of the operations, using a latency $\ell = \ell_{\min}$ for all operations. Let $\text{ASAP}(v)$ denote the resulting ASAP control step for each operation $v \in V$. Similarly let $\text{ALAP}(v, \lambda)$ denote the ALAP control step for each operation $v \in V$ given a user-specified latency bound of λ and under the same operation latencies.

Each operation $v \in V$, executing on resource type $r \in R(v)$, can only start its execution during one of the time steps in the set $T(v, r)$ (6.13).

$$T(v, r) = \{t \in \mathbb{N} \cup \{0\} : t \geq \text{ASAP}(v) \wedge t \leq \text{ALAP}(v, \lambda) - L(r) + \ell_{\min}(v)\} \quad (6.13)$$

It is useful to enumerate all possible start times $T(v)$ for each operation $v \in V$, according to (6.14), and indeed the complete set of time-steps T (6.15).

$$T(v) = \{t | \exists r \in R(v) : t \in T(v, r)\} \quad (6.14)$$

$$T = \{t | \exists v \in V : t \in T(v)\} \quad (6.15)$$

6.3.2 ILP Formulation

Extending the notation used by Landwehr, et al. [LMD94], the ILP may be formulated as follows. Let $b_{i,r}$ define a Boolean variable with $b_{i,r} = 1$ iff instance number i of resource type r has at least one operation bound to it. This allows the objective function to be formulated in linear form (6.16).

$$\text{Minimize} \sum_{r \in R} \text{cost}(r) \sum_{i=1}^{I(r)} b_{i,r} \quad (6.16)$$

In order to introduce the constraints, let $x_{v,t,i,r}$ be defined as in (6.17).

$$x_{v,t,i,r} = \begin{cases} 1, \text{ if operation } v \text{ is scheduled at time-step } t \text{ on the} \\ \quad i\text{th instance of resource type } r \\ 0, \text{ otherwise} \end{cases} \quad (6.17)$$

The minimization is performed subject to three types of constraint. The first are the binding constraints, to ensure that each operation is executed on exactly one instance (6.18). The second are the resource constraints, to ensure that no resource instance is executing more than one operation at a time (6.19). The final set are the precedence constraints, to ensure that all operations obey the dependencies in the sequencing graph (6.20).

$$\forall v \in V, \sum_{r \in R(v)} \sum_{i=1}^{I(r)} \sum_{t \in T(v,r)} x_{v,t,i,r} = 1 \quad (6.18)$$

$$\forall t \in T, \forall r \in R, \forall i \in \{1, ..., I(r)\}, \\ \sum_{v \in V : r \in R(v)} \sum_{t_1 \in \{t, ..., t+L(r)-1\} \cap T(v,r)} x_{v,t_1,i,r} \leq b_{i,r} \tag{6.19}$$

$$\forall (v_1, v_2) \in D, \\ \forall t \in T(v_2) \cap \{\text{ASAP}(v_1) + \ell_{\min}(v_1) - 1, \ldots, \text{ALAP}(v_1) + \ell_{\max}(v_1) - 1\}, \\ \sum_{r \in R(v_2)} \sum_{i=1}^{I(r)} \sum_{t_2 \in T(v_2,r) : t_2 \leq t} x_{v_2,t_2,i,r} \; + \\ \sum_{r \in R(v_1)} \sum_{i=1}^{I(r)} \sum_{t_1 \in T(v_1,r) : t_1 > t - L(r)} x_{v_1,t_1,i,r} \leq 1 \tag{6.20}$$

Example 6.5. Recall the simple sequencing graph of Fig. 6.2. The ILP formulation for this sequencing graph contains 164 variables and 166 constraints for $\lambda = 18$, the lowest achievable latency. Fig. 6.3 illustrates an optimal solution corresponding to this latency constraint, which has the following optimization variables taking the value 1: $x_{a_2,0,1,25}$, $x_{a_4,14,1,25}$, $x_{a_3,16,1,25}$, $x_{a_1,16,1,19}$, $x_{m_1,1,1,(19,17)}$, $x_{m_2,9,1,(19,17)}$, $x_{m_3,2,1,(33,21)}$, $x_{m_4,9,1,(33,21)}$, $x_{m_5,9,1,(40,12)}$, $b_{1,25}$, $b_{1,19}$, $b_{1,(19,17)}$, $b_{1,(33,21)}$, $b_{1,(40,12)}$. All other variables are equal to zero.

After solution, the values of the ILP optimization variables $x_{v,t,i,r}$ and $b_{i,r}$ encode a solution to Problem 6.3 given in (6.21–6.23).

$$Y = \{(i, r) : b_{i,r} = 1\} \tag{6.21}$$

$$\mathcal{R}(v) = \sum_{r \in R(v)} \sum_{i=1}^{I(r)} \sum_{t \in T(v,r)} (i, r) \cdot x_{v,t,i,r}, \text{ for } v \in V \tag{6.22}$$

$$\mathcal{S}(v) = \sum_{r \in R(v)} \sum_{i=1}^{I(r)} \sum_{t \in T(v,r)} t \cdot x_{v,t,i,r}, \text{ for } v \in V \tag{6.23}$$

Optimal solutions can only be found for relatively small examples using ILP, due to the large number of variables and constraints. Moreover, the number of variables and constraints increases linearly with the relaxation of λ. It is these drawbacks that have motivated the search for efficient heuristic solutions to this problem, as presented in Section 6.4.

6.4 A Heuristic Approach

This section presents a heuristic approach to Problem 6.3 [CCL01a]. The proposed algorithm iteratively refines word-length information while using

resource-constrained scheduling and a combined resource binding and word-length selection procedure, in order to steer the solution towards feasibility with respect to the user-specified latency constraint.

The following description starts with an overview of the heuristic in Section 6.4.1 and a description of the word-length compatibility graph in Section 6.4.2. Each of the algorithm steps is then described: calculation of resource bounds (Section 6.4.3) and latency bounds (Section 6.4.4), scheduling using incomplete word-length information (Section 6.4.5), combined binding and word-length selection (Section 6.4.6) and word-length refinement (Section 6.4.7).

6.4.1 Overview

A high-level overview of the proposed heuristic is shown in Algorithm ArchSynth. The algorithm arrives at a solution through an iterative refinement of word-length information in order to reach the user-specified latency target λ. An initial solution is constructed by allowing each operation to be scheduled using the longest latency of all resources which could perform that operation. Scheduling in this manner guarantees that any resource binding will not violate the schedule, and it is expected that a great deal of resource sharing can be achieved. However, using the upper bound latency of each operation may result in a violation of the overall latency target λ. At each iteration of Algorithm ArchSynth, these upper bounds are refined by selecting an operation and reducing its upper-bound latency and hence the range of different word-length resources which could implement that operation.

In most implementation cases, the area consumed by a multiplier is significantly larger than that consumed by an adder. It is for this reason that Algorithm ArchSynth calculates bounds on the number of *multipliers* required and constructs the solution accordingly, searching for solutions with between $m_{\min}$ and $m_{\max}$ multipliers. The corresponding bound on the number of adders is determined through a simple scaling with a factor β. By performing the optimization in this manner, the bounds on the number of each resource type need not be optimized individually, leading to an improvement in algorithm execution time at the possible penalty of a few extra adders in the resulting architecture. For our current implementation, we use the empirically derived $\beta = 4$. Of course if the set of available resource types were expanded beyond adders and multipliers in a way that destroys this imbalance in implementation area, this approach could no longer be used. In the most general case, it would be necessary to extend Algorithm ArchSynth, introducing a new loop similar to steps 1–2 for each resource type.

Algorithm 6.3
Algorithm ArchSynth
Input: A data flow graph $P(V, D)$ and a latency constraint λ

Output: Scheduling, binding, and word-length
for each operation
begin
1. calculate $m_{\min}$ and $m_{\max}$, bounds on #multipliers
2. **for** $n \in \{m_{\min}, \ldots, m_{\max}\}$ **do**
2.1 **do**
2.1.1 calculate the resource set covering each operation
2.1.2 search for upper-bounds on latency of each operation
2.1.3 search for a feasible schedule using latency upper bounds and no more than n mults and $\beta \cdot n$ adds
2.1.4 perform combined binding and word-length selection
2.1.5 **if** binding violates the latency constraint λ **do** try to refine operation word-length information
else do record this feasible solution
end if
while refinement (step 2.1.5) is possible
end for
end

The calculation of resource bounds (step 1) and each of the steps 2.1.1–2.1.5 will be discussed in detail in the following sections. In addition, failure conditions can arise in finding upper-bounds (step 2.1.2), deadlocks in scheduling with incomplete word-length information (step 2.1.3), and refining upper bounds (step 2.1.5). Each of these cases will also be considered in the following sections.

6.4.2 Word-Length Compatibility Graph

A fundamental model that underlies the majority of the proposed heuristic is the *word-length compatibility graph*.

Definition 6.6. A *word-length compatibility graph* $G(V \cup R, C \cup H)$ is a representation of information about the *type* of each operation, the *word-length* of each operation, and schedule-derived information on time-compatibility between operation pairs. The vertex set can be partitioned into two subsets V and R, where V denotes the set of operations, and R denotes the set of resource types (6.8). The set of edges can also be partitioned into two subsets C and H. H is a set of undirected edges $\{v, r\}$, where $v \in V$ and $r \in R$, representing the information that operation v could be performed by resource type r. C is a set of directed edges (v_1, v_2), where $v_1, v_2 \in V$, representing the information that operation v_1 is scheduled to complete execution before operation v_2 is scheduled to start execution.

The scheduling algorithm to be described in Section 6.4.5 utilizes the operation – resource-type compatibility encoded in the edge set H and implicitly

creates the edge set C in the process. The combined binding and word-length selection algorithm to be described in Section 6.4.6 utilizes both the time-compatibility edge set C and the operation resource-type compatibility set H.

It is important to note that the edge set C has been chosen to ensure that subgraph $G^+(V, C)$ exhibits a transitive orientation [Gol80], since if $v_1 \in V$ finishes before v_2 starts, and v_2 finishes before v_3 starts, it follows that v_1 finishes before v_3 starts. This orientation will be used in Section 6.4.6 to aid fast resource binding. Note also that the set C of directed edges in the graph need not be constructed explicitly in a software implementation, but can be inferred from the scheduled times of the operations.

Example 6.7. A simple word-length compatibility graph is shown in Fig. 6.4(c), corresponding to the data flow graph and schedule shown in Figs. 6.4(a) and (b), respectively.

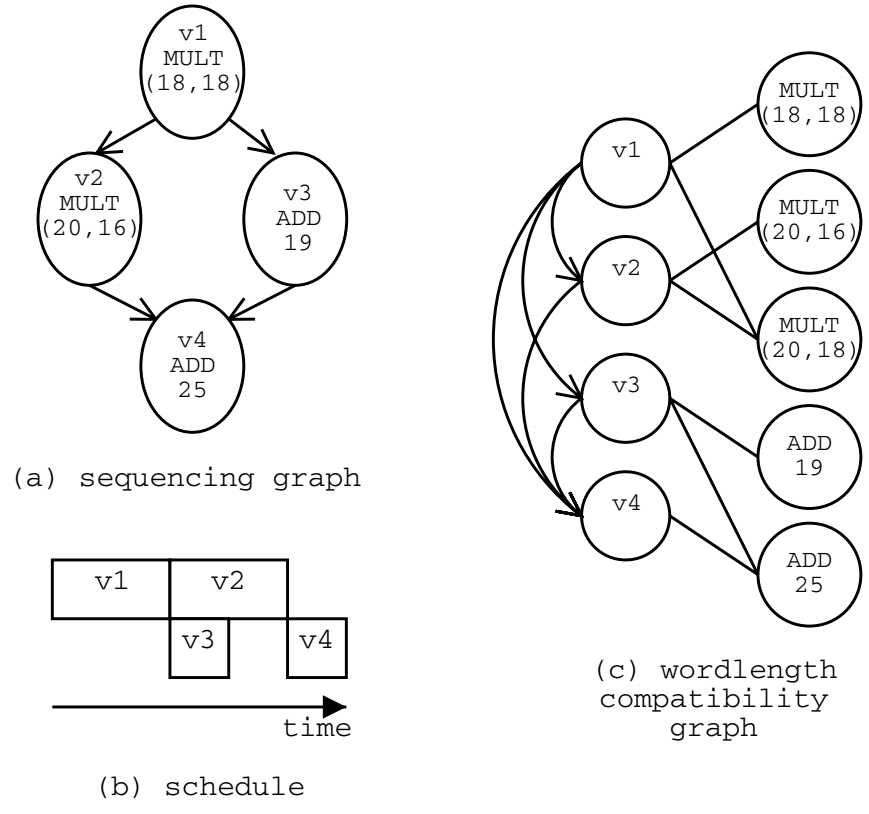


Fig. 6.4. A Word-Length Compatibility Graph

The initial word-length compatibility graph is constructed in the following manner: A resource set R is constructed following (6.5–6.8). Edge set H is initialized to the set $\{\{v \in V, r \in R\} : r \in R(v)\}$. C is initialized to the empty set. As Algorithm ArchSynth executes, word-length refinement will result in the deletion of edges from the set H.

6.4.3 Resource Bounds

The first stage of the heuristic is to find the smallest and largest sensible upper bounds to place on the number of multipliers required. These values are obtained from a study of how the iteration latency achieved by list-scheduling decreases with the number of multipliers allowed.

A standard resource-constrained list scheduling algorithm [DeM94, NT86] can be used to heuristically obtain these bounds. Standard ALAP urgency-based list scheduling with a bound $\mathbf{c} \in \mathbb{N}^2$ on the number of resources of each type is used. $\mathbf{c}$ is a 2-vector of integer elements, the first corresponding to the bound on the number of multipliers and the second corresponding to the bound on the number of adders.

The bounds $m_{\min}$ and $m_{\max}$ used in Algorithm ArchSynth can now be defined, assuming that the given latency bound is realizable. Bound $m_{\min}$ is the smallest value such that the list-scheduled latency is within the constraint for all schedules with $\mathbf{c} \geq (m_{\min}, \beta m_{\min})$ and $\ell(v) = \ell_{\min}(v)$ for all $v \in V$. Similarly bound $m_{\max}$ is the smallest value such that the list-scheduled latency is within the constraint for all schedules with $\mathbf{c} \geq (m_{\max}, \beta m_{\max})$ and $\ell(v) = \ell_{\max}(v)$ for all $v \in V$. For tight latency constraints there may be no such $m_{\max}$ value, in which case $m_{\max}$ is set to $|V_m|$. In each of these cases, a binary search is used to determine the bounds.

The rationale behind these bounds is the following. If an algorithm cannot be scheduled to meet the imposed latency constraint under a resource constraint $\mathbf{c}$, even when all operations have their minimum possible latency, then the algorithm cannot meet this latency constraint for any $\mathbf{c}' \leq \mathbf{c}$. This provides a lower bound on the number of each type of resource required. Similarly, if the imposed timing constraint can be met under a resource constraint $\mathbf{c}$, even when all operations have their maximum latency, then the algorithm can meet this latency constraint for all $\mathbf{c}' \geq \mathbf{c}$. This provides an upper bound on the number of each type of resource required. Fig. 6.5 illustrates the way in which the achieved latency varies with the bound on the number of multipliers supplied to the list-scheduler.

Example 6.8. An example derivation of resource bounds is illustrated in Fig. 6.6. Fig. 6.6(a) illustrates a simple data flow graph, with corresponding initial word-length compatibility graph shown in Fig. 6.6(b). The latency curves resulting from list scheduling for different resource bounds are plotted in Fig. 6.6(c). The points corresponding to the minimum possible number and maximum necessary number of multiplier resources have been highlighted.

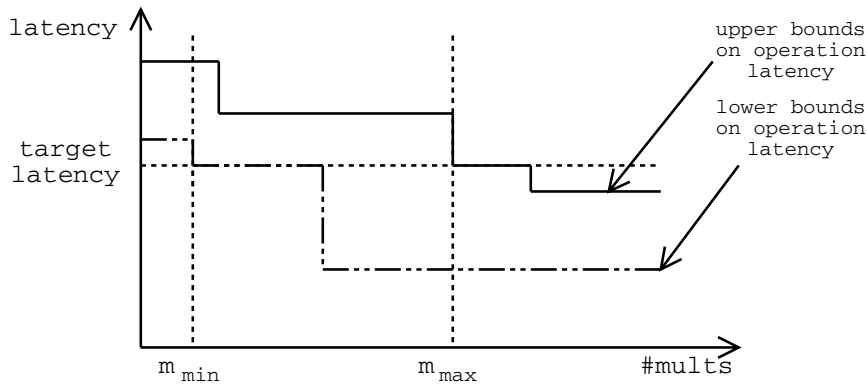


Fig. 6.5. Calculating bounds on the number of multipliers $m_{\min}$ and $m_{\max}$

6.4.4 Latency Bounds

Before entering the main refinement loop in Algorithm ArchSynth, it is possible to significantly reduce the number of iterations by pruning the operation latency search space. If it is not possible to list-schedule a data flow graph $P(V, D)$ when all operations $V \setminus \{v\}$ have their minimum possible latency while operation v has latency $\ell(v)$, then it is assumed that a feasible schedule will equally not be possible if operation v has any latency $\ell'(v) > \ell(v)$. This allows the edge set H of an initially constructed word-length compatibility graph to be refined. The approach is illustrated in Algorithm LatencyBounds. After first checking that a feasible solution exists in steps 1–2 (by trying to schedule when all operations have minimum latency), the algorithm proceeds by deleting edges from the set H. Each operation node v is tested in turn (step 3) to find the maximum latency the operation could have (out of those corresponding to resource types which could implement that operation) while not violating the overall latency constraint. Once this value is found (step 3.1), any edges connecting node v to a resource type with a greater latency are removed from the word-length compatibility graph (step 3.2).

Algorithm 6.4
Algorithm LatencyBounds
Input: A data flow graph $P(V, D)$, initial
word-length compatibility graph $G(V \cup R, C \cup H)$,
resource constraint vector $\mathbf{c}$ and latency constraint λ
Output: A refined word-length compatibility graph
begin
1. $\ell(v) \leftarrow \min_{\{v,r\} \in H} L(r)$ for all $v \in V$
2. **if** ListSchedule(P, ℓ, $\mathbf{c}$) returns a schedule violating
latency constraint λ **do**

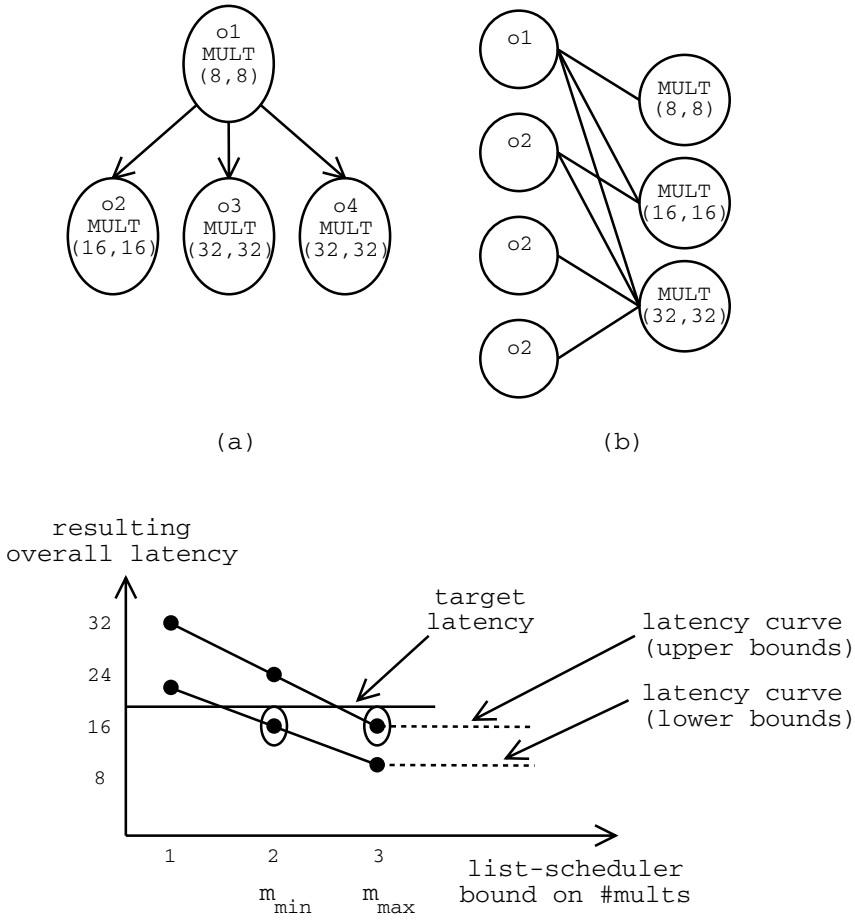


Fig. 6.6. Example multiplier bounds

return failure case
end if
3. **foreach** $v \in V$ **do**
3.1 Search for the maximum $\ell(v) \in \{L(r) : \exists\{v, r\} \in H\}$ such that ListSchedule(P, ℓ, **c**) returns a schedule satisfying latency constraint λ
3.2 $H \leftarrow H \setminus \{\{v, r\} \in H : L(r) > \ell(v)\}$
4. return $\ell(v)$ to its original value $\ell(v) \leftarrow \min_{\{v,r\} \in H} L(r)$
end foreach

end

Using Algorithm LatencyBounds, an upper bound on the latency of each operation can be established. Once this has been done, the set of edges H of the Word-Length Compatibility Graph, representing the possible design decisions, has been pruned.

6.4.5 Scheduling with Incomplete Word-Length Information

At the time of scheduling in Algorithm ArchSynth, the word-length of each operation may not be fixed. Indeed each operation could be implemented using any resource type to which the operation is linked by an edge in the word-length compatibility graph. The scheduling problem therefore has incompletely defined constraints [CSH00], and a technique must be developed to incorporate these constraints into directing the search for a solution. The following paragraphs illustrate the need for such a technique, before introducing the proposed solution.

Traditional resource-constrained scheduling techniques such as force-directed list scheduling [PK89], require resource constraints to be expressed in terms of a bound on the number of resources of each type. During standard list scheduling, these constraints are tested at each time step before deciding whether to schedule a new operation. The constraints may be formally expressed as follows. Let $e_{v,t}$ be defined as in (6.24). Thus $e_{v,t} = 1$ iff operation v is *executing* during time-step t. Given a set of control steps T (6.15), a set of operations V, and the maximum number of resources c_k of type k, the traditional resource constraints may be expressed as (6.25).

$$e_{v,t} = \begin{cases} 1, \text{ if } (\mathcal{S}(v) \leq t) \wedge (\mathcal{S}(v) + \ell(v) > t) \\ 0, \text{ otherwise} \end{cases} \tag{6.24}$$

$$\forall k \in \{\text{ADD}, \text{MULT}\}, \max_{t \in T} \sum_{v \in V : \text{TYPE}(v) = k} e_{v,t} \leq a_k \tag{6.25}$$

In the case of multiple word-length systems, these constraints tend to be too relaxed to guarantee that no more than a_k resources of type k will be used by the given schedule.

Example 6.9. Consider the schedules and corresponding word-length compatibility graphs shown in Fig. 6.7. Such graphs could arise during the execution of Algorithm ArchSynth. Fig. 6.7(a) has fully defined word-length information for each operation. It is clear that even though v_1, v_2 and v_3 are all multiplications and do not overlap in execution, three distinct multiplier resources will still be required for their implementation. However the standard scheduling constraint (6.25) would be satisfiable for $a_{\text{MULT}} = 1$.

Fig. 6.7(b) has an incomplete specification (there is at least one operation which could be implemented in more than one possible resource type). However a 32×32-bit multiplier could conceivably implement every operation.

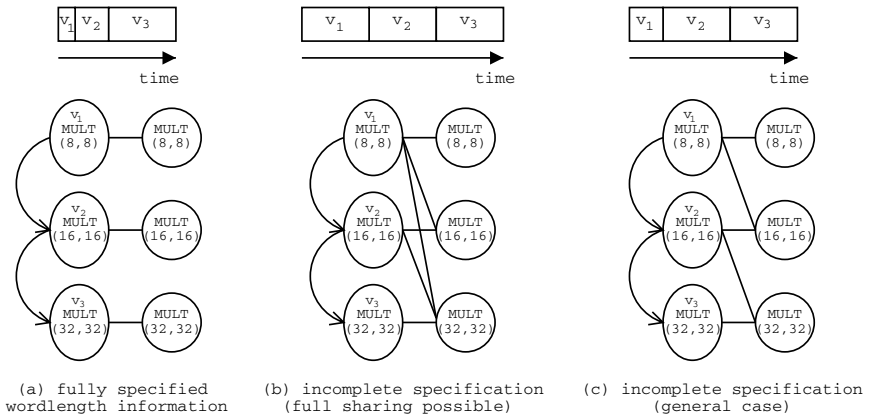


Fig. 6.7. Some schedules and word-length compatibility graphs

Thus it is possible to implement the entire system using a single multiplier resource.

Fig. 6.7(c) illustrates a general case, corresponding to the deletion of a single edge from the word-length compatibility graph of Fig. 6.7(b). Using traditional methods, it is unclear in this case how to incorporate such constraints into the search for an appropriate schedule.

These examples demonstrate that a more sophisticated approach to scheduling is required to take word-length information into account. In general it is necessary to consider the *incomplete* word-length specification provided by an edge set H.

The scheduling algorithm proposed is a modification of standard list scheduling [DeM94]. The modification lies in the resource constraint calculation. Before any scheduling takes place, a small cardinality subset $S \subseteq R$ is found such that $\forall v \in V, \exists s \in S : \{v, s\} \in H$. Conceivably, a resource binding could consist only of resource of types represented in S. Define $O(r)$ to be the set of operations performable by resource type $r \in R$, *i.e.* $O(r) = \{v \in V : \exists\{v, r\} \in H\}$. Similarly let $S(v)$ denote the subset of resource types in S which could implement operation $v \in V$, *i.e.* $S(v) = \{s \in S : \exists\{v, s\} \in H\}$. Then the proposed constraint function to be used in the algorithm can be expressed as in (6.26).

$$\forall k \in \{\text{ADD}, \text{MULT}\},$$
$$\sum_{s \in S:\text{TYPE}(s)=k} \max_{t \in T} \left\{ \sum_{v \in O(r)} e_{v,t} |S(v)|^{-1} \right\} \leq a_k \tag{6.26}$$

This is a heuristic measure with the following justification. Firstly (6.26) is at least as strict as (6.25), which is a special case of the former under the condition $|\text{TYPE}(V)| = |S|$, the smallest sized S possible. This represents the

case where each multiplication could be performed by a single large multiplier, and each addition could be performed by a single large adder. As the possibilities for the implementation of each operation are reduced during execution of Algorithm ArchSynth, the balance on the left hand side of (6.26) shifts from the $\max_{t \in T}$ to the $\sum_{s \in S}$ to reflect the stricter constraints. The small cardinality S is used in order to relax the constraint as much as possible, since any two operations in $O(s)$ could *possibly* be eventually bound to the same resource. Operations belonging to more than one $O(s)$, *i.e.* those $v \in V$ with $|S(v)| > 1$, are accounted for by 'sharing' equally their usage between each of the elements $S(v)$.

Algorithm IncompSched illustrates this scheduling based on incomplete information. Two auxilliary data structures are used in the algorithm to keep track of the scheduling constraint (6.26), usage(s) and maxusage(s). Respectively, these keep track of the instantaneous and peak usage of resource type $s \in S$. The algorithm starts by setting the latency $\ell(v)$ of each operation to its maximum (step 1). After so doing, a standard ALAP-based urgency measure [DeM94] is calculated for each node (step 2), and the time step index is initialized (step 3). The set S described above (step 4), and its related function $S(v)$ (step 5), to be used in the scheduling constraint (6.26) are then calculated. Step 6 ensures that the peak usage maxusage(s) for each element of that set is initialized. The algorithm then enters its main scheduling loop, with one iteration per time step (step 7).

At the start of each iteration, the instantaneous usage of resources is initialized (step 7.1), the ready-list is calculated (step 7.2), and the prime candidate for scheduling is selected (step 7.3). The algorithm then enters a secondary loop (step 7.4), which tries to schedule this and any other operation of the same type. The current left-hand side of (6.26) is first calculated (step 7.4.1), and then updated (step 7.4.2) for any $s \in S$ for which scheduling in the current control step would use more than the current peak usage for that s. If the updated (6.26) is still satisfied, then the scheduling of the operation is accepted (step 7.4.3), and the peak usage is updated (step 7.4.4). Deadlocks, to be discussed below, may occur in the scheduling process. These are detected by step 7.5.

Example 6.10. An example execution of Algorithm IncompSched is shown in Fig. 6.8. The data flow graph and word-length compatibility graph are shown in Figs. 6.8(a) and (b) respectively. Fig. 6.8(c) enumerates the $S(v)$ sets for this example, and Fig. 6.8(d) shows how the usage and maxusage variables evolve as the algorithm executes for $a_{\text{ADD}} = 1$, $a_{\text{MULT}} = 2$. The resulting schedule is shown in Fig. 6.8(e), and could be resource-bound as a single 16-bit adder together with both a 16×16-bit multiplier and a 32×32-bit multiplier. Details on how such a resource binding can be found for general graphs are discussed in the following section.

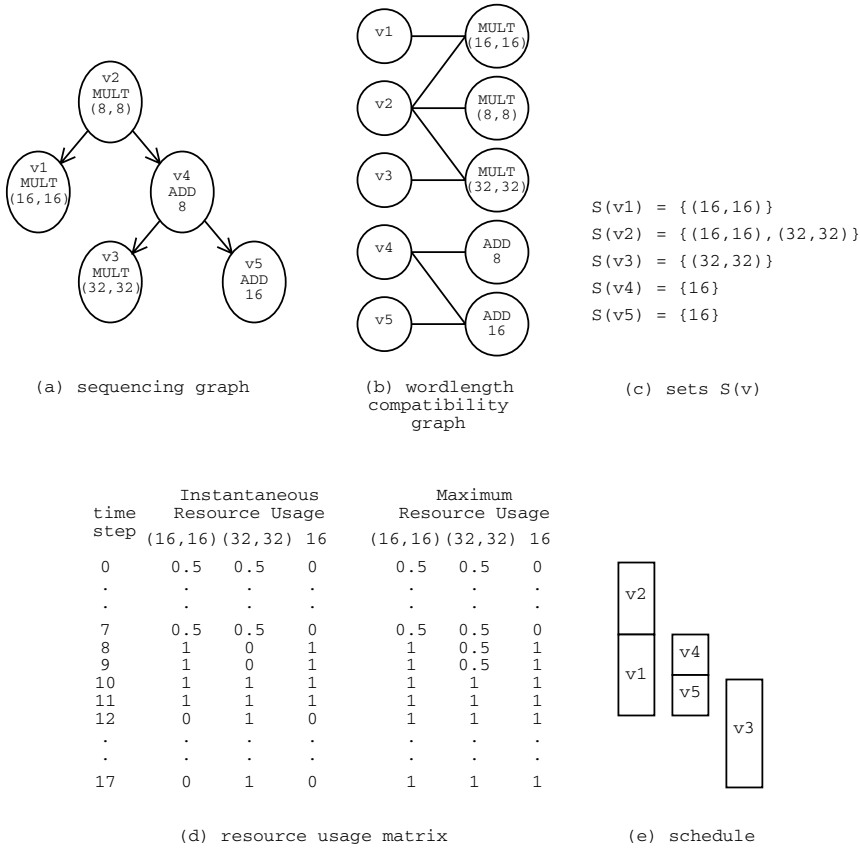


Fig. 6.8. An example of scheduling a data flow graph with incomplete word-length specifications

Algorithm 6.5
Algorithm IncompSched
Input: A data flow graph $P(V, D)$, word-length compatibility graph $G(V \cup R, C \cup H)$ and maximum number $\mathbf{c}$ of each resource type
Output: A schedule $\mathcal{S} : V \rightarrow \mathbb{N} \cup \{0\}$ for each $v \in V$
begin
1. $\ell(v) = \max\limits_{\{v,r\} \in H} L(r)$
2. Determine the 'urgency' of each operation $v \in V$ through ALAP scheduling
3. $t \leftarrow 0$
4. Find $S \subseteq R$ of smallest size such that $\forall v \in V, \exists s \in S : \{v, s\} \in H$

5. Let $S(v) = \{s \in S : \exists\{v, s\} \in H\}$
6. maxusage$(s) \leftarrow 0$ for all $s \in S$
7. **do**
7.1 usage$(s) \leftarrow 0$ for all $s \in S$
7.2 $E \leftarrow$ the list-schedule 'ready list' [DeM94], sorted by urgency
7.3 $e \leftarrow$ most urgent element of E
7.4 **do**
7.4.1 total $\leftarrow \sum_{s \in S(\{v \in V : \mathrm{TYPE}(v) = \mathrm{TYPE}(e)\})} \mathrm{maxusage}(s)$
7.4.2 **foreach** $s \in S(e)$:
usage$(s) + |S(e)|^{-1} >$ maxusage(s) **do**
total $\leftarrow$ total $-$ maxusage$(s)+$
usage$(s) + |S(e)|^{-1}$
end foreach
7.4.3 **if** total $\leq a_{\mathrm{TYPE}(e)}$ **do**
$\mathcal{S}(e) \leftarrow t$
usage$(s) \leftarrow$ usage$(s) + |S(e)|^{-1}$
7.4.4 **foreach** $s \in S(e)$: usage$(s) >$ maxusage(s) **do**
maxusage$(s) \leftarrow$ usage(s)
end foreach
end if
7.4.5 $e \leftarrow$ next most urgent element of E, if one exists
while such an e exists $\wedge \exists k \in \{\mathrm{ADD}, \mathrm{MULT}\}$:
$\sum_{s \in S : \mathrm{TYPE}(s) = k} \mathrm{maxusage}(s) < a_i$
$t \leftarrow t + 1$
7.5 **if** deadlock detected **do**
return failure case
end if
while there remains at least one unscheduled operation
end

There are a number of significant differences between standard list scheduling and Algorithm IncompSched. Information on resource usage is accumulated over control steps in Algorithm IncompSched, rather than each step being constraint-function-independent of each other step. There are two related drawbacks from this: Firstly, it is possible for the proposed list-scheduler to deadlock, by scheduling operations belonging to $O(s_1)$ for the some $s_1 \in S$ early in the schedule and then having no remaining resources to schedule operations belonging to $O(s_2)$ for some $s_2 \in S, s_2 \neq s_1$ later in the schedule. Such deadlocks can be easily detected: if all operations have finished by the current time-step and yet no operation has been scheduled by the end of that time-step, deadlock has occurred. Secondly, although the scheduler may not deadlock, greedy allocation of parallel $O(s_1)$ operations early-on in the schedule may cause schedules of longer than optimal latency. Thus Algorithm IncompSched has a greedy bias towards earlier time steps.

The subset $S \subseteq R$ used by Algorithm IncompSched can be found easily through Algorithm SSet. Starting from an empty set S, this algorithm simply iteratively adds those resource types from the set H which could implement the most (thus far uncovered) operations.

Algorithm 6.6
Algorithm SSet
Input: Word-Length compatibility graph $G(V \cup R, C \cup H)$
Output: Set $S \subseteq R$ required by Algorithm IncompSched
begin
$S \leftarrow \emptyset$
$V' \leftarrow V$
$H' \leftarrow H$
while $|V'| > 0$ **do**
Find $r \in R$ such that $|\{\{v, r\} \in H'\}|$ is maximum
$S \leftarrow S \cup \{r\}$
$V' \leftarrow V' \setminus \{v \in V : \exists \{v, r\} \in H\}$
$H' \leftarrow H' \setminus \{\{v, r\} \in H\}$
end while
end

6.4.6 Combined Binding and Word-Length Selection

Once a data flow graph has been scheduled, resource binding and word-length selection can be performed. No resource binding can violate the scheduling latency constraint, since latency upper bounds have been used when performing the scheduling (Algorithm ArchSynth). The COMBINED BINDING AND WORD-LENGTH SELECTION problem (Problem 6.11) is therefore a subproblem of Problem 6.3.

Problem 6.11 (Combined Binding and Word-Length Selection). Given a scheduled word-length compatibility graph $G(V \cup R, C \cup H)$, the COMBINED BINDING AND WORD-LENGTH SELECTION problem is to select a set of resources and their associated word-lengths Y and a mapping from operation to resource $\mathcal{R} : V \rightarrow Y$ such that the area consumed by the set of resources is minimal and no resource executes more than one operation in each clock cycle.

Definition 6.12. A clique k is a *maximal clique* of graph G iff k is not a subgraph of any other clique of graph G.

Definition 6.13. A clique $k(V_k, E_k)$ is a *maximum clique* of graph G iff there is no clique $k'(v'_k, e'_k)$ of G with $|v'_k| > |v_k|$.

Definition 6.14. A clique $k(v', c')$ of the subgraph $G^+(V, C)$ of word-length compatibility graph $G(V \cup R, C \cup H)$ is a *feasible clique* iff $\exists r \in R : \forall v \in v', \{v, r\} \in H$.

The COMBINED BINDING AND WORD-LENGTH SELECTION problem is approached by partitioning the subgraph $G^+(V, C)$ into a set K of feasible cliques. The feasibility constraint captures the requirement that there must be a single resource capable of performing all operations in the clique. The cost of this resource binding is then given by (6.27).

$$\sum_{k(v_k, e_k) \in K} \min_{r \in R : \forall v \in v_k, \exists \{v, r\} \in H} \text{cost}(r) \tag{6.27}$$

This problem is a special case of the set-covering or weighted unate covering problem.

Problem 6.15 (SET COVERING, [Chv79]). Consider a set of sets $U = \{u_1, u_2, \ldots, u_n\}$ and associated positive costs $c_{u_1}, c_{u_2}, \ldots, c_{u_n}$. Let $I = \bigcup_{u \in U} u$. A subset $U' \subseteq U$ is a *cover* iff $\bigcup_{u \in U'} u = I$. The *cost* of this cover is $\sum_{u \in U'} c_u$. The problem is to find a cover of minimum cost.

The combined binding and word-length selection problem can be cast as a set covering problem in the following manner. Let U denote the set of node sets of all feasible cliques in the graph $G^+(V, C)$ (6.28). The cost c_k associated with clique $k \in U$ is given by the corresponding term in the summation of (6.27).

$$U = \{V' \subseteq V : V' \text{ induces a feasible clique in } G^+(V, C)\} \tag{6.28}$$

The proposed approach is to extend a known heuristic for solving the unate covering problem [Chv79] to the combined resource binding and word-length selection application. In order to present the proposed extensions to this simple heuristic, it is first reviewed below.

Intuitively, for a greedy algorithm it becomes more desirable to include a set u_j in the cover U' as the number of elements covered by u_j and not already covered by any previously chosen set increases. This is tempered by the cost of set u_j, and thus the ratio of these two quantities forms an appropriate measure of desirability. This observation leads to the following heuristic proposed in [Chv79].

Algorithm 6.7
Algorithm ChvatalHeur
Input: An instance of the set covering problem (problem 6.15)
Output: A cover $U' \subseteq U$
begin
$U' \leftarrow \emptyset$

```
 while ∃i : |u_i| ≠ 0 do
     find i such that |u_i|/c_i is maximum
     U' ← U' ∪ {u_i}
     foreach j ∈ {1, ..., n} do
         u_j ← u_j \ u_i
     end foreach
 end while
end
```

The first, and simplest, extension to [Chv79] is to include some compensation for the greedy nature of the original algorithm. If a clique is chosen during one iteration of the algorithm, it is checked whether this clique could be extended to cover all operations covered by any of the cliques chosen at previous iterations. If such an extension is possible, the selected clique is grown accordingly and the previously chosen clique is deleted from the cover set.

The more important distinction is that the set U is never calculated, since its size can be very large (exponential in $|V|$). Instead, an implicit approach is used, which is polynomial in $|V|$.

Consider the set of clique node-sets $U_r \subseteq U$ that may be implemented using a resource $r \in R$, *i.e.* $U_r = \{u \in U : \forall v \in u, \{v, r\} \in H\}$. It is clear that it only makes sense to select those cliques induced by maximal subsets $u \in U_r : \nexists u' \in U_r : u \subset u'$ for implementation in resource type r. Non-maximal cliques correspond to so-called 'column domination' in unate covering [GN72]. However a stronger statement can be made, that only maximum feasible cliques need to be considered as candidates, *i.e.* $u \in U_r : \nexists u' \in U_r : |u| < |u'|$. This is because Chvatal's heuristic [Chv79] will always return a higher score for a maximum feasible clique than for a non-maximum clique of the same resource type, and so a maximum clique will always be chosen in preference to a non-maximum clique. Incorporating this knowledge leads to the proposed resource binding and word-length selection algorithm presented below as Algorithm ResBindWLSel.

The algorithm starts by initializing certain values (steps 1–4). In step 1, C is set to correctly reflect its definition (Table A). V_1 is initialized to the full set of operations (step 2), and will be iteratively reduced as operations are bound to resources (step 5.7). n_r is a counter of how many resources of type $r \in R$ have thus far been allocated by the binding, and is initialized to 0 (step 3). Y, the final set of resources, is initialized to be empty (step 4).

After initialization, the algorithm enters its main loop (step 5), where one resource type is selected, and operations bound to an instance of that resource type, on each iteration. In order to choose which resource type to select, Chvatal's heuristic is applied (steps 5.1–5.2). Compensation for the greedy nature of this heuristic is provided by step 5.3, which can backtrack on previous decisions, as described above. Finally, steps 5.4–5.7 perform the binding: step 5.4 adds a new resource to the existing set, step 5.5 binds each operation in the clique selected by steps 5.1–5.2 to this new resource. Finally,

the number of resources of that type is incremented (step 5.6) and the set of unbound operations is reduced (step 5.7).

Algorithm 6.8
Algorithm ResBindWLSel
Input: A Word-Length Compatibility Graph
$G(V \cup R, C \cup H)$ and schedule $\mathcal{S} : V \to \mathbb{N} \cup \{0\}$
Output: A resource set Y and a binding $\mathcal{R} : V \to Y$
begin
1. $C \leftarrow \{(v_1, v_2) : v_1, v_2 \in V \wedge \mathcal{S}(v_1) + \max_{\{v_1, r\} \in H} L(r) \leq \mathcal{S}(v_2)\}$
2. $V_1 \leftarrow V$
3. $n_r \leftarrow 0$ for all $r \in R$
4. $Y \leftarrow \emptyset$
5. **while** $|V_1| > 0$ **do**

5.1 **foreach** $r \in R$ **do**
$V' \leftarrow \{v \in N : \exists \{v, r\} \in H\}$
Let $G'(V', E')$ be the subgraph of $G^+(V, C)$
induced by vertex set V'
Search $G'(V', E')$ for a maximum clique with
node set $p_r \subseteq V'$
end foreach

5.2 Choose $r \in R$ such that $|p_r|(\text{cost}(r))^{-1}$ is maximum

5.3 **foreach** $y \in Y$
$V' \leftarrow \{v \in V : \mathcal{R}(v) = y\}$
if $\forall v \in V' : \exists \{v, r\} \in H$ and $V' \cup p_r$ induces a
clique in $G^+(V, C)$ **do**
$Y \leftarrow Y \setminus \{y\}$
$\mathcal{R}(v) \leftarrow (r, n_r)$ for all $v \in y$
end if
end foreach

5.4 $Y \leftarrow Y \cup \{(r, n_r)\}$

5.5 $\mathcal{R}(v) \leftarrow (r, n_r)$ for all $v \in p_r$

5.6 $n_r \leftarrow n_r + 1$

5.7 $V_1 \leftarrow V_1 \setminus p_r$

end while
end

Because the graph $G^+(V, C)$ (and any subgraph induced by a vertex subset) is a transitively oriented graph, finding the maximum clique is a simple linear-time operation [Gol80].

Example 6.16. Consider the data flow graph illustrated in Fig. 6.9(a). An example execution of Algorithm ResBindWLSel is illustrated in Fig. 6.9(d-f) for the schedule and word-length compatibility graph shown in Fig. 6.9(b) and (c). Three iterations are required. During the first iteration, a 30-bit adder is selected to perform operations $v2$ and $v3$. The second iteration selects a

20×10-bit multiplier to perform operation $v1$ and the final iteration selects a 16×16-bit multiplier to perform operation $v4$. The two possibilities faced by the first iteration: a 30-bit adder for operations $v2$ and $v3$ or a 15-bit adder to perform operation $v2$ only, have equal heuristic scores. However if the latter possibility were selected, the clique covering $v3$ (selected on the following iteration) would be grown to cover $v2$, resulting in the same binding.

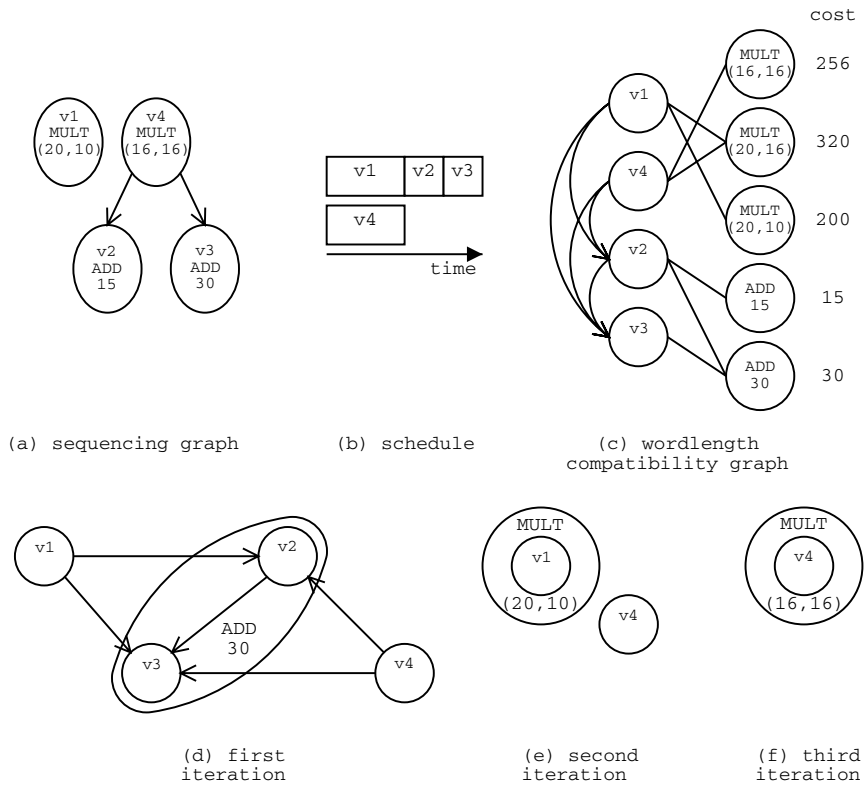


Fig. 6.9. Example execution of Algorithm ResBindWLSel

6.4.7 Refining Word-Length Information

On each iteration of Algorithm ArchSynth, if the latency constraint is violated, the word-length information in the word-length compatibility graph is refined in order to guide the algorithm towards a feasible solution. The *bound critical path*, defined below, is calculated in order to provide an insight into which operations may be blocking the creation of a feasible solution. Then a single operation on this bound critical path is selected, and its latency is

refined, leading to the deletion of one or more edges from the word-length compatibility graph.

The Bound Critical Path

As a first step for refining latency upper bounds, the concept of the *bound critical path* is introduced by extension of the critical path.

Definition 6.17. Consider a data flow graph $P(V, D)$. The *critical path* $V_c \subseteq V$ of a data flow graph $P(V, D)$, given a latency $\ell(v)$ for each node $v \in V$ is defined to be the subset of nodes with equal ASAP and ALAP scheduling times with respect to the minimum possible latency constraint (6.29).

$$v \in V_c \Leftrightarrow \text{ASAP}(v) = \text{ALAP}(v, \max_{v' \in V}\{\text{ASAP}(v') + \ell(v')\}) \tag{6.29}$$

Given a data flow graph $P(V, D)$, a word-length compatibility graph $G(V \cup R, C \cup H)$, a schedule $\mathcal{S} : V \rightarrow \mathbb{N} \cup \{0\}$, a resource set Y and a resource binding $\mathcal{R} : V \rightarrow Y$, it is possible to construct a set of edges D^b representing operations abutting in time on the same resource (6.30). Nodes v_1 and v_2 are thus connected by an edge in D^b iff node v_1 finishes execution on a resource *the cycle before* node v_2 starts execution on the same resource.

$$\begin{aligned} D^b = \{(v_1 \in V, v_2 \in V) : \mathcal{S}(v_1) + \\ L(\pi_1(\mathcal{R}(v_1))) = \mathcal{S}(v_2) \wedge \mathcal{R}(v_1) = \mathcal{R}(v_2)\} \end{aligned} \tag{6.30}$$

Definition 6.18. The *bound critical path* V_b of a scheduled and resource-bound algorithm of data flow graph $P(V, D)$ is defined to be the critical path of the augmented data flow graph $P'(V, D \cup D^b)$.

It is possible in this way to capture information about which operations may be responsible for failure to meet the user-specified iteration latency. Once this critical subset of operations has been determined, methods can be applied to refine compatibility information present in the edge set H.

Refining Latency Upper Bounds

The reduction of the latency of an operation $v \in V_b$ in the bound critical path could possibly lead to the reduction of the overall latency. Indeed, in order for Algorithm IncompSched to meet the latency constraint λ, at least one of the operations in the subset V_b' (6.31) *must* have its latency reduced (V_b' is the subset of the bound critical path consisting of operations whose latency *could* be reduced and then complete within the latency constraint).

$$V_b' = \{v \in V_b : \mathcal{S}(v) + \min_{\{v,r\} \in H} L(r) \leq \lambda \wedge \ell_{\min}(v) \neq \ell_{\max}(v)\} \tag{6.31}$$

Reducing the latency of operations that are not members of this set but are nevertheless members of V_b may be necessary, but will clearly not be sufficient to schedule the entire data flow graph within the required latency bound.

On each iteration of Algorithm ArchSynth, one of the operations $v \in V_b'$ is chosen, and the edge set H is adjusted to reduce the upper bound on the latency of v. In the case that $|V_b'| > 1$, the following empirically derived heuristic tie-break rules are applied.

By reducing the upper bound on the latency of operation v, edges $\{v, r\} \in H : L(r) = \ell_{\max}(v)$ will be deleted from H. Considering word-length and type information alone, the potential set of operations $J(v)$ which could share a resource with operation $v \in V$ is given by $J(v) = \{v' \in V | \exists r \in R : \{v, r\}, \{v', r\} \in H\}$. A simple heuristic measure would be to select the operation $v \in V$ for which this set is least 'affected' by the resultant loss of the edges in H. Thus the node $v \in V$ minimizing measure (6.32) is selected. The set $J'(v)$, corresponding to $J(v)$ after removal of the edges in H, is defined as $J'(v) = \{v' \in V | \exists r \in R : (\{v, r\}, \{v', r\} \in H \wedge L(r) \neq \ell_{\max}(v))\}$.

$$|J(v) \setminus J'(v)| \cdot |J(v)|^{-1} \tag{6.32}$$

Once again, in case of tie break on the above measure, a further heuristic can be applied: those operations currently bound to resources utilizing less than the upper-bound latency of that operation are preferred candidates. Thus an arbitrary node $v \in V_b'$ maximizing (6.32) and satisfying $L(\pi_1(\mathcal{R}(v))) < \ell_{\max}(v)$ is selected, if one exists. Otherwise simply an arbitrary node maximizing (6.32) is selected.

This procedure is illustrated in Algorithm WLRefine. After constructing the abuttal edges (6.30) in step 1, the bound critical path is extracted (step 2), and the subset V_b' of the bound critical path (6.31) is found (step 3). If this set is empty, no refinement of word-length information can help the search for a feasible solution, and the failure case is returned (step 4). Otherwise a search for an appropriate operation v to refine is conducted (steps 5-6), according to the heuristics discussed above. Once a node has been found the edge set H, representing the which resource types can perform which operations, is refined by removing all edges connecting the chosen operation to resources of latency equal to the maximum of all resource types for that operation.

Algorithm 6.9
Algorithm WLRefine
Input: A data flow graph $P(V, D)$, word-length compatibility graph $G(V \cup R, C \cup H)$, latency constraint λ, resource set Y and resource binding $\mathcal{R} : V \rightarrow Y$
Output: A refined word-length compatibility graph
begin
1. Construct the abuttal edges D^b (6.30)
2. Perform ASAP and ALAP scheduling on $P(V, D \cup D^b)$

3. Extract the subset $V_b' \subseteq V$ of nodes on the bound critical path which could complete within the latency constraint (6.31)
4. **if** $V_b' = \emptyset$ **do**
   return failure case
5. Find $V_p \subseteq V_b'$ of nodes maximizing the measure (6.32)
6. **if** $\exists v \in V_p : L(\pi_1(\mathcal{R}(v))) < \ell_{\max}(v)$ **do**
   Select one such node $v \in V_p$
   else do
   Select an arbitrary node $v \in V_p$
   end if
7. $H \leftarrow H \setminus \{\{v, r\} \in H : L(r) = \ell_{\max}(v)\}$

end

Example 6.19. Fig. 6.10 illustrates an example refinement phase corresponding to the data flow graph introduced in Fig. 6.2 and reproduced in Fig. 6.10(a) for convenience. Nodes that are on the computation critical path with respect to the data flow graph $P(V, D)$ are highlighted in Fig. 6.10(a), $V_c = \{a_2, m_3, m_4, m_5, a_1, a_3\}$. This data flow graph has been scheduled and resource-bound in Fig. 6.10(b). The portions of node execution time between $\ell_{\min}(v)$ and $\ell_{\max}(v)$ have been shaded in the figure.

The augmented data flow graph $P'(V, D \cup D^b)$ obtained from time-abutment edges D^b is illustrated in Fig. 6.10(c) and consists of a single extra edge from operation m_2 to operation m_5. The resulting change in critical path is significant. The bound critical path is given by $V_b = \{m_1, m_2, m_5, a_3\}$.

For $\lambda = 18$, the lowest achievable latency constraint, the subset V_b' is given by $V_b' = \{m_1, m_2\}$. The heuristic measures described above may then be applied to decide which of these two nodes is to have its upper bound latency reduced.

6.5 Some Results

Before considering in detail the performance of the methods discussed in this chapter, it is instructive to follow through the sequencing graph of Fig. 6.2 which has been used as an example throughout this chapter where appropriate. Both optimal and heuristic schedules, resource allocations, bindings and word-length selections have been performed for this example in order to explore the area / latency tradeoff achievable. Fig. 6.11 plots these results. Not all ILP results are shown, due to excessive ILP solver execution time. The ILP and heuristic solutions are identical for all cases except $\lambda = 26$, where there is a slight difference caused by the presence of an extra adder in the heuristic solution.

Figure 6.12 illustrates the results for the benchmark circuits introduced in Section 4.6. For comparison, not only are the optimal (ILP) results and

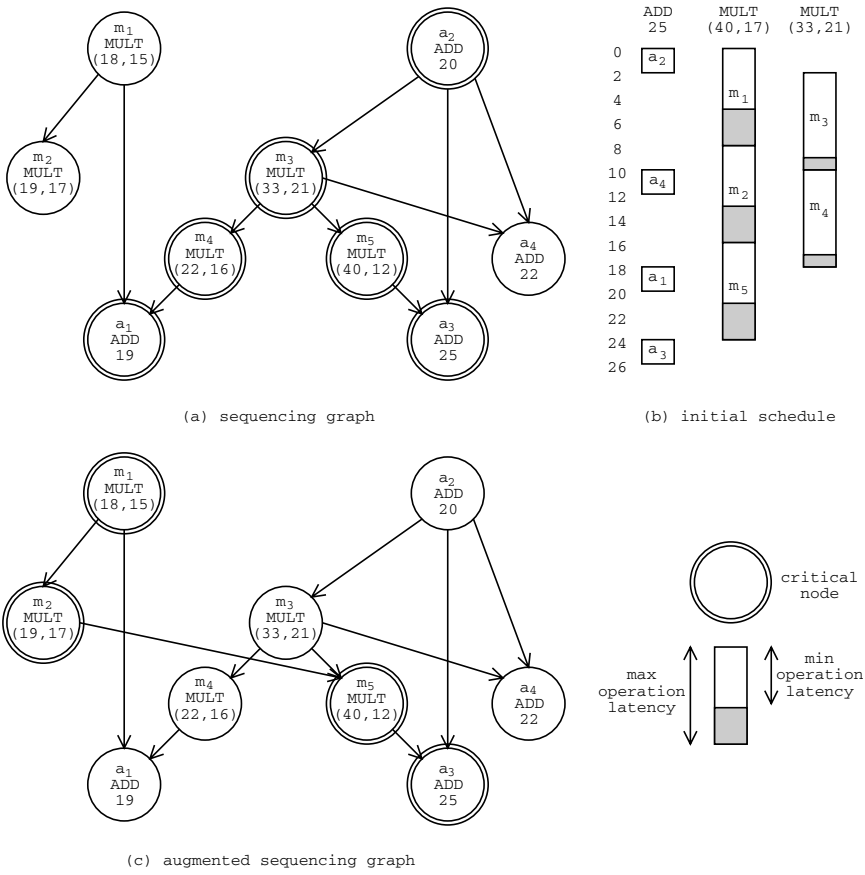


Fig. 6.10. Example use of the bound critical path for word-length refinement

the heuristic results provided, but also the solutions corresponding to word-length-blind scheduling followed by an *optimal* resource binding [CCL00b]. All three sets of results are only provided for the IIR filter, the polyphase filter bank and the RGB to YCrYb convertor, due to the excessive execution time of both the ILP solver and the optimal binding.

These results illustrate that for the benchmark circuits, the heuristic presented in this chapter provides a significant improvement in area (between −16% and 46%, average 15%) over a two-stage approach of scheduling and then binding, even when the binding is optimal. The heuristic results have between 0% and 62% (average 14%) worse area than the optimum combined solution, for cases where the optimum is known.

In order to fully characterize the heuristic performance, further results have been obtained using artificially generated examples. For statistically significant data on solution quality, 200 random sequencing graphs have been

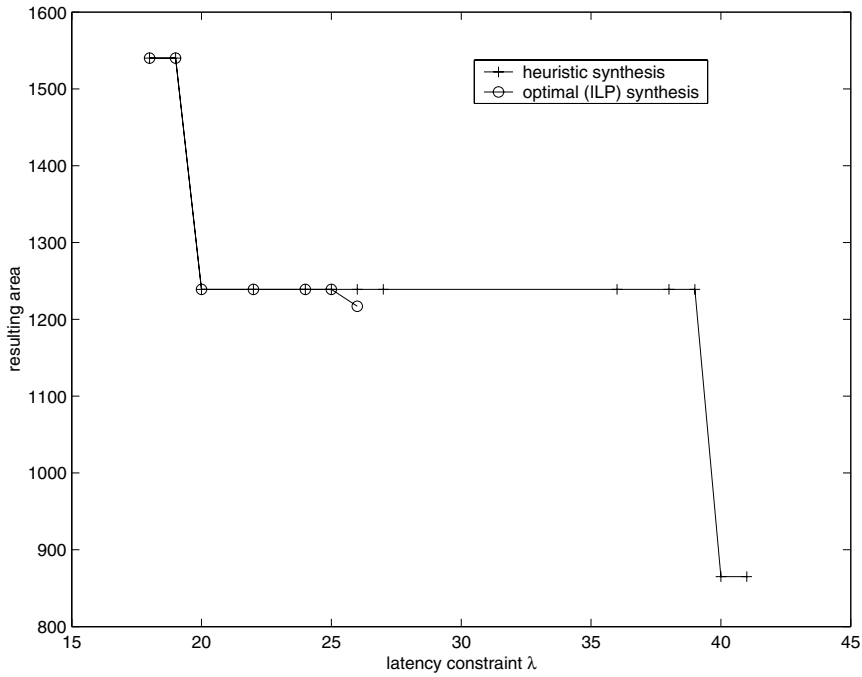
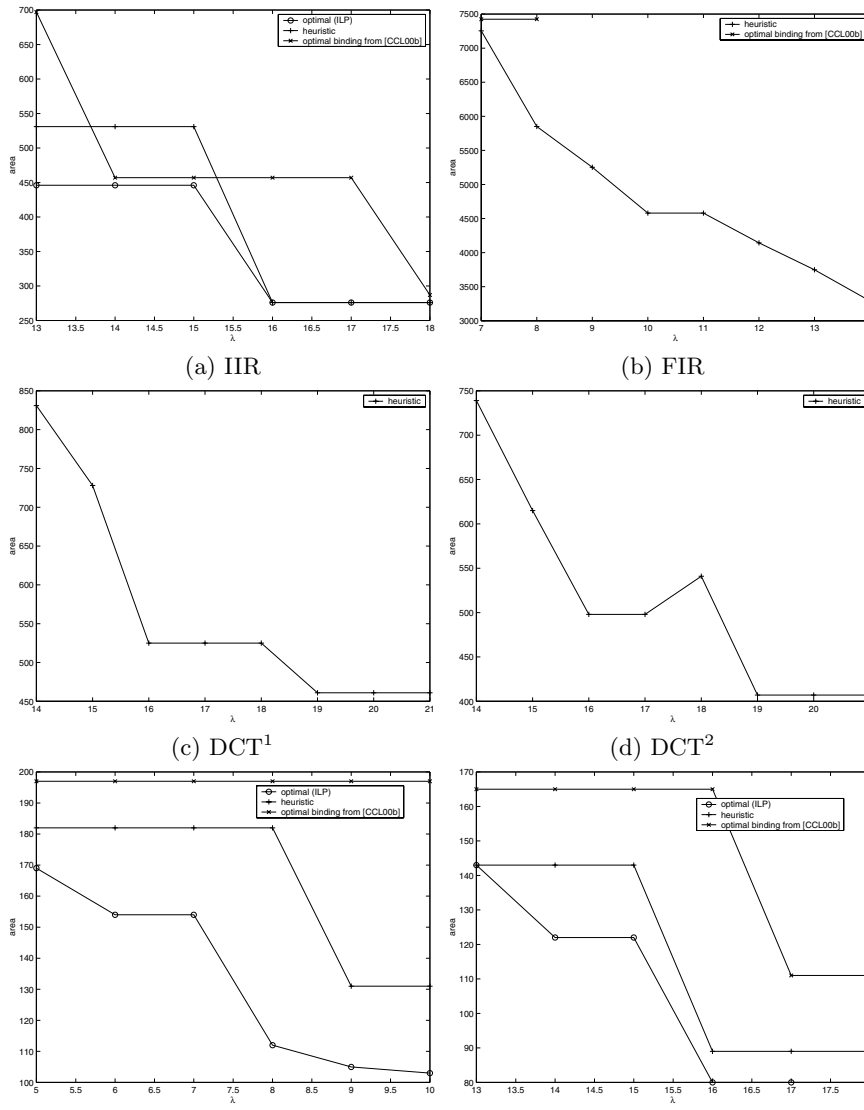


Fig. 6.11. Design-space exploration for the simple sequencing graph of Fig. 6.2

generated for each (problem size $|V|$, latency constraint λ) pair, using a variant of the TGFF algorithm [DRW98]. The first set of detailed quality results that has been collected, measures the variation of the heuristic solution quality with both the problem size and the tightness of the supplied latency constraint. For each sequencing graph, the minimum possible latency constraint $\lambda_{\min}$ has been found using ASAP scheduling, from which latency constraints have been generated corresponding to a 0% to 30% relaxation of $\lambda_{\min}$. Algorithm Arch-Synth has then been executed on the graph / latency constraint combination. The resulting area has been normalized with respect to the optimal solution resulting from [CCL00b] where operations may only share resources when the implemented resource has latency no longer than the minimum required to implement the given operations. These results are plotted in Fig. 6.13, as a percentage area penalty for using the approach of [CCL00b] over the heuristic presented in this chapter. Each point represents the mean of two hundred representative designs.

The results illustrate that for designs with even a small 'slack' in terms of latency constraints, significant area improvements of up to 30% can be made by performing the scheduling, binding, and word-length selection in the intertwined manner proposed. The area improvements come from increased resource sharing due to implementing small word-length operations in larger



(a) IIR (b) FIR

(c) DCT[1] (d) DCT[2]

(e) PFB (f) RGB–YCrCb

Fig. 6.12. Design-space exploration for the benchmark circuits of Section 4.6

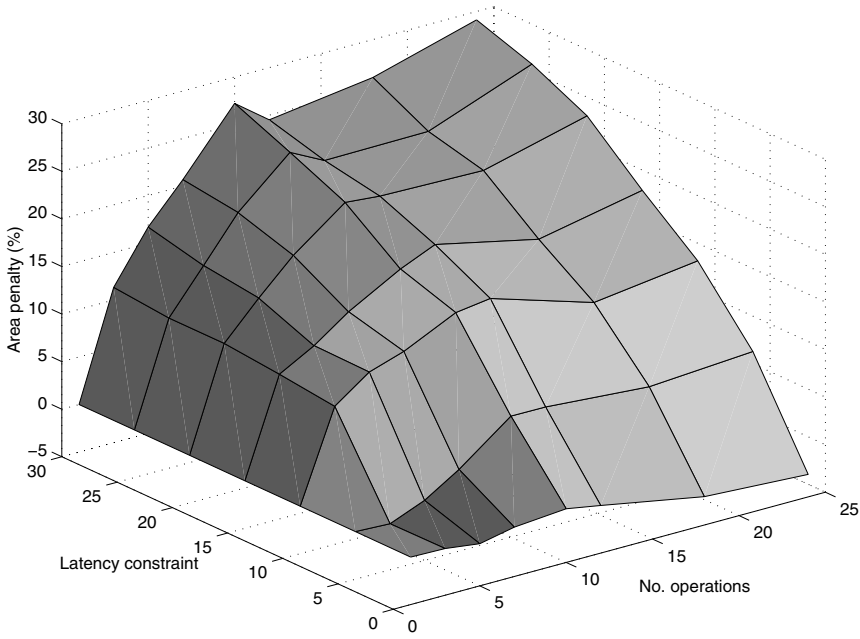


Fig. 6.13. Variation with number of operations and latency constraint of area penalty for [CCL00b] over the proposed heuristic

word-length resources with longer latency. Even for relatively small graphs, area improvements of tens of percent are possible.

Fig. 6.14 illustrates the increase in implementation area from using the heuristic presented in Section 6.4 over the optimum combined problem presented in Section 6.3. These results can only be provided for modest problem size and a minimum latency constraint $\lambda = \lambda_{\min}$, as the ILP solution execution time scales rapidly with problem size and as the latency constraint is relaxed.

The variation of minimum-latency execution time with problem size for 200 graphs using the ILP model (solved with `lp_solve` [Sch97] on a Pentium III 450MHz) and the heuristic algorithm (implemented in C on the same machine) is shown in Fig. 6.15, illustrating the polynomial complexity of the heuristic against the exponential complexity of the ILP. Over the range of 1 to 10 operations, the relative increase in area ranges from 0% to 16% whereas the ILP solution takes between one and three orders of magnitude greater time to execute. An important point not brought out by these results is the scaling of execution time with overall latency constraint. The number of variables in the ILP model scales with the latency constraint, making the execution time highly dependent on this parameter. This is illustrated in Table 6.1 for 200 9-operation sequencing graphs. By contrast, the execution time of Algorithm ArchSynth does not scale with the latency constraint. Thus the

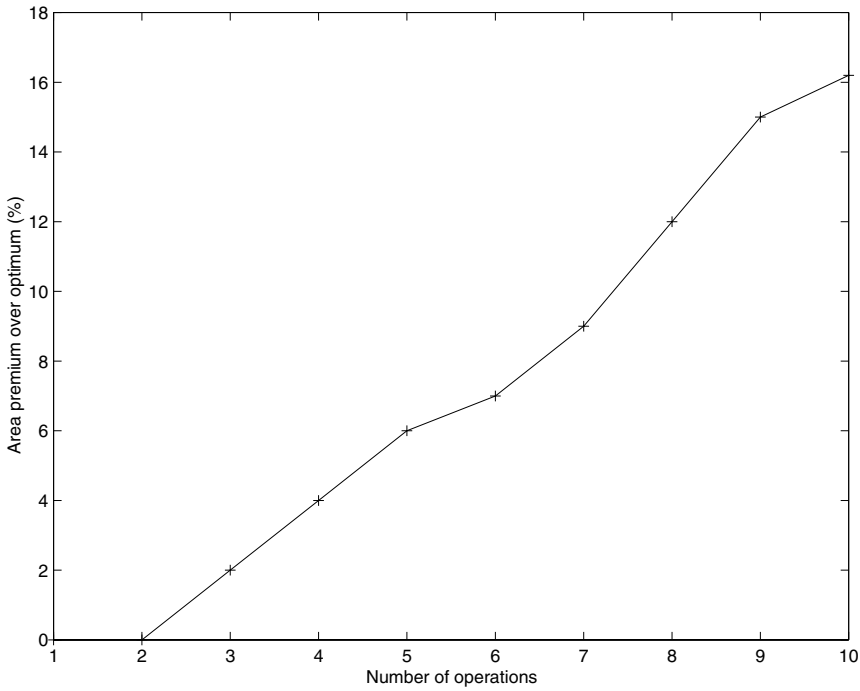


Fig. 6.14. Variation of area premium for Algorithm ArchSynth over the optimum solution

one to three orders of magnitude illustrated in Fig. 6.15 are under conditions most favourable to the ILP-based solution.

Table 6.1. Variation of execution time for 200 graphs with $\lambda/\lambda_{\min}$ for heuristic and ILP solution

$\lambda/\lambda_{\min}$	heuristic (secs)	ILP (mins:secs)
1.00	3.02	2:07.09
1.05	3.51	4:05.21
1.10	3.73	15:55.56
1.15	3.52	>30:00.00

The results presented clearly indicate the practical nature of the heuristic presented in this chapter, whose execution results in a speedup of up to three orders of magnitude over ILP solution, even for modest graph sizes, at an area-penalty of between 0% and 16%.

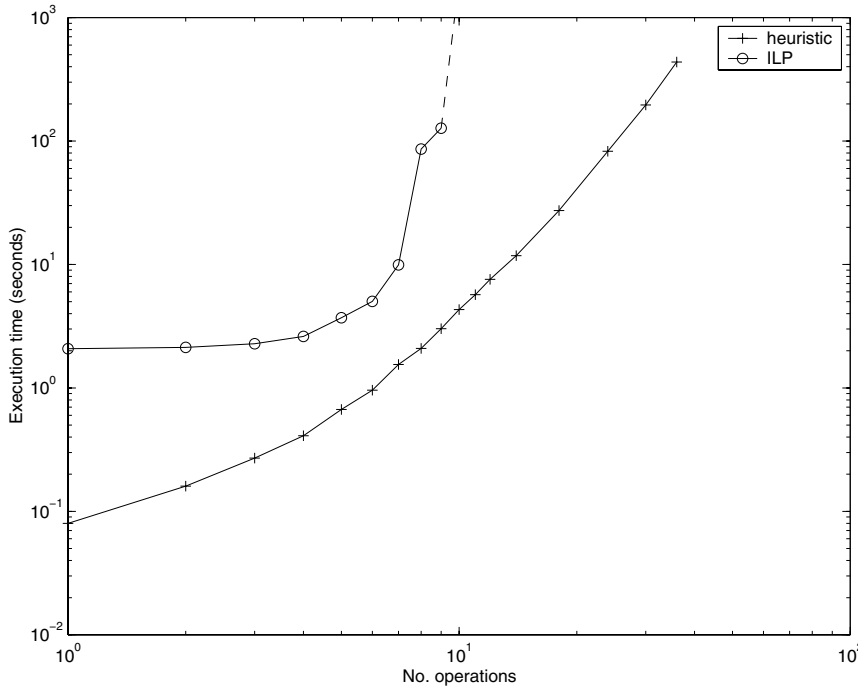


Fig. 6.15. Variation with number of operations of execution time for 200 graphs (heuristic and ILP solutions)

6.6 Summary

This chapter has presented work addressing the problem of architectural synthesis for multiple word-length systems. It has been demonstrated that the traditional architectural synthesis problems of operation scheduling, resource allocation, and resource binding, can be significantly complicated by the presence of multiple word-length arithmetic.

An Integer Linear Programming (ILP) construction for the multiple word-length architectural synthesis problem has been formulated. The ILP formulation provides solutions which are optimal with respect to the area-based cost function, but suffers from long run-times which scale up rapidly with relaxation of the latency constraint.

A heuristic solution has been developed based on intertwined scheduling, resource binding / word-length selection, and word-length refinement. This involves techniques for scheduling with incompletely defined word-length information, combining binding and word-length selection, and latency-based word-length refinement based on critical path analysis.

The results from an implementation of both the ILP and the heuristic approach show that significant improvements to system area can be made

by allowing non-critical operations to share large word-length resources. The heuristic solution has been shown to be within 16% of the optimal area, over a range of minimum-latency problem sizes for which a three orders of magnitude speedup over an ILP solver has been achieved. For non-minimum-latency problems, the execution speedup is even greater.

7

Conclusion

This final chapter contains two sections. Section 7.1 summarizes the key ideas in this book, while Section 7.2 contains suggestions for future work.

7.1 Summary

This book has examined the problems surrounding the synthesis of hardware from initial infinite precision specifications of DSP algorithms. Such a synthesis procedure raises the level of abstraction at which a DSP algorithm can be specified, to the level typically used by DSP algorithm designers.

As a result of investigating efficient DSP implementation architectures, it has been argued that the multiple word-length paradigm can result in efficient implementations, incorporating fine control in the trade-off of implementation area and power consumption against acceptable arithmetic error. The term *lossy synthesis* has been coined to describe the incorporation such a trade-off within the synthesis process itself.

The design and development of multiple word-length systems is not a simple process; indeed it has been proven that the search for optimal architectures is an NP-hard problem [CW01]. However an heuristic algorithm has been presented, which achieves significant improvements of up to 45% in area and 39% in speed, compared to traditional design techniques which consider only a single uniform system word-length for all the signals in the system.

Different approaches to the problem of overflow have been considered. Implementations making use of saturation arithmetic have been compared to systems scaled to ensure that no overflow is possible. A technique has been introduced to automate the design of saturation arithmetic systems. The proposed technique automates the placement of each saturation nonlinearity and the degree of saturation at each nonlinearity. Results indicate that up to a further 8% reduction in system area and 24% speedup can be achieved by applying saturation automation techniques together with the multiple word-length approach described above. The relative advantage of saturation arith-

metic is shown, however, to depend heavily on the nature of the algorithm to which it is applied.

Finally, the impact of multiple word-length designs on architectural synthesis has been assessed. Optimal and heuristic techniques have been proposed for the combined problem of scheduling, resource allocation, word-length selection, and resource binding for multiple word-length synthesis. The proposed approaches allow a wide design-space to be searched, trading off the number of samples processed per clock period against implementation area. Compared to previous techniques which do not take into account the variation of operation latency with word-length, our approach can achieve area savings of up to 46%.

7.2 Future Work

There are many ways in which the work presented in this book could be taken forward and expanded to new domains, new architectures, and new objectives. In this section some of the possibilities will be expanded upon.

The objective functions used throughout this book have concentrated on the minimization of implementation area. Often the resulting circuits have displayed a significant increase in maximum clock frequency and reduction in power consumption as desirable side-effects of the optimization techniques used. However it may be useful in the future to explicitly consider clock frequency or power consumption within the objective function or as a constraint on the optimizations performed. In synchronous systems the maximum allowable clock frequency is determined by worst-case propagation delay, whereas area has, in this book, been modelled by a weighted sum of different component areas. It is likely that this difference in the nature of the objective function would require further development of the optimization algorithms.

As discussed in Chapter 6, performing word-length optimization before architectural synthesis can lead to sub-optimal designs. A future direction of research would be to investigate the interdependence between these two steps and to develop a combined architectural synthesis and word-length optimization approach. Some steps in this direction have recently been made by Kum and Sung [KS01].

Dynamic reconfiguration is an area of particular interest in the FPGA community [SLC98]. This term refers to 'on-the-fly' modifications to the hardware encoded in an FPGA configuration. Bondalapati, *et al.* have proposed a technique to vary the word-length of a loop variable over *time*, as the loop executes [BP99]. Another avenue for future research would be to concentrate on the extension of the framework described in this book to time-evolution of precision requirements.

A

Notation

This appendix introduces some of the background notation and definitions, which are used throughout the book.

A.1 Sets and functions

$\mathbb{Z}$ is used to denote the set of all integers. $\mathbb{N}$ is similarly the set of all natural numbers (positive integers). $\mathbb{R}$ is used to denote the set of all reals.
$f : X \rightarrow Y$ denotes a function mapping elements of X to elements of Y. In this context $f(X) \subseteq Y$ denotes the *range* of function f.
Following a notation commonly used in DSP texts, square brackets ([and]) are used throughout the thesis when it is useful to distinguish a function with a discrete domain from a function with a continuous domain (for which round brackets are used). Thus for $f_1 : \mathbb{R} \rightarrow \mathbb{R}$ we write $f_1(x)$, but for $f_2 : \mathbb{Z} \rightarrow \mathbb{R}$ we write $f_2[k]$.
$\cup$ is used to denote the union of sets and $\cap$ to denote their intersection.
$|X|$ is used to denote the size (cardinality) of a set X
$X \times Y$ denotes the Cartesian product of the sets X and Y

A.2 Vectors and Matrices

Vectors are denoted with bold face and their elements with subscripts, thus a_i is the i'th element of vector $\mathbf{a}$
A^T denotes the transpose of a matrix A, whereas A^H denotes the conjugate transpose.

Definition A.1. Let $\mathbf{a}$ and $\mathbf{b}$ be two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$. The vector relation $\mathbf{a} \leq \mathbf{b} \Leftrightarrow a_i \leq b_i$ for all $1 \leq i \leq n$. Similarly $\mathbf{a} \not\leq \mathbf{b} \Leftrightarrow \exists i : a_i > b_i$. $\square$

A.3 Graphs

Definition A.2. A *directed graph* $G(V, E)$ consists of a finite set V and an irreflexive binary relation $E \subset V \times V$ on V. An element of V is referred to as a *vertex* or *node*. An element of E is referred to as an *edge*. □

Definition A.3. For a graph $G(V, E)$, inedge(v) (outedge(v)) is used to denote the set of in-edges (out-edges) $\{(v', v) \in E\}$ ($\{(v, v') \in E\}$). □

Definition A.4. The *indegree (outdegree)* of the node $v \in V$ of a graph $G(V, E)$ is the size of the node's inedge (outedge) set. □

Definition A.5. For a graph $G(V, E)$, pred(v) (succ(v)) is used to denote the set of predecessor (successor) nodes $\{v' : (v', v) \in E\}$ ($\{v' : (v, v') \in E\}$).

A.4 Miscellaneous

z^* denotes the conjugate of a complex number z.
$\wedge$ is used to denote the logical (Boolean) *and* function
$\vee$ is used to denote the logical (Boolean) *or* function

Definition A.6. The *z-transform* of a sequence $h[t]$ is written $\mathcal{Z}\{h[t]\}$ as defined in (A.1). The *inverse z-transform* of a function $H(z)$ is written $\mathcal{Z}^{-1}\{H(z)\}$.

$$\mathcal{Z}\{h[t]\} = \sum_{t=-\infty}^{\infty} h[t] z^{-t} \tag{A.1}$$

□

A.5 Pseudo-Code

This section provides a brief description of the pseudo-code used. Keywords are given in **bold**. There are 14 keywords: **Input**, **Output**, **begin**, **while**, **end**, **do**, **foreach**, **if**, **switch**, **case**, **for**, **goto**, **and**, and **return**.

Each algorithm begins with an **Input** and **Output** line describing the pre-requisites for and results of algorithm execution, respectively. The body of the algorithm is enclosed within **begin**–**end** delimiters.

The **while**, **do**, **if**, **switch**, **case**, **for**, **goto** and **return** keywords are taken directly from C [KR78]. **foreach** allows iteration over each member of a set. A label 'L1' for a **goto** is indicated by 'L1:' preceding the labelled line of code.

while, **if**, or **foreach** constructs containing more than one line of code are enclosed within **do**–**end** delimiters.

Glossary

$\ell_1\{\cdot\}$: ℓ_1 norm. The sum of absolute values of the impulse response corresponding to the transfer function argument.

$L_2\{\cdot\}$: L_2 norm. The square-root of the sum of squared values of the impulse response corresponding to the transfer function argument.

$O(\cdot)$: So-called "Big-Oh" representation is used to define the asymptotic worst-case behaviour of an algorithm. An algorithm is said to be $O(f(n))$ if its execution time, operating on an instance of size $n > n_1$, is bounded from above by $k \cdot f(n)$ for some constant k.

$Q(\cdot)$: Upper tail of the standard Gaussian (Normal) distribution.

$\mathrm{sgn}(\cdot)$: The signum function. -1 for negative argument, $+1$ for positive argument.

$\delta(\cdot)$: The Dirac delta function.

$\mathbb{R}$: The set of all reals.

$\mathbb{Z}$: The set of all integers.

$\mathbb{N}$: The set of all positive integers.

$\mathcal{Z}\{\cdot\}$: The z-transform. See Definition A.6, p. 152).

$\mathcal{Z}^{-1}\{\cdot\}$: The inverse z-transform.

annotated computation graph: A formal representation of the fixed-point implementation of a computation graph. See Definition 2.5, p. 13

auto-regressive filter: A filter whose present output value is formed from a weighted sum of its past output values.

behavioural description: A description of a circuit in terms of the way the circuit behaves in response to input stimuli. This description need not contain any information about the way the circuit is actually implemented.

blocking: A blocking read operation is one which must wait for a token to be present before reading it. A blocking write operation is one which must wait for space on a communication channel before writing to that channel.

computable: See Definition 2.4, p. 12.

computation graph: A formal representation of an algorithm. See Definition 2.1, p. 10.

control step: The basic unit of time used in scheduling. A control step is equal to a state in finite state machine-based controllers or to a microprogram step in microprogrammed control units [Cam90].

cross-correlation: See Definition 5.5, p. 84.

DAG: Directed acyclic graph.

data-dependencies: An operation x is data-dependent on an operation y if y produces a result which is consumed by x. For example the code fragment `a = b + c; d = a * b;` contains a data-dependency between the multiplication operation and the addition operation [ASU86].

data path: The part of a design that deals with computation of data. Data paths are often characterized by regular logic and bit slices, for example ALUs and adders [Cam90].

DCT: Discrete Cosine Transform [Mit98].

direct form: Several filter implementation structures come under the banner of 'direct form' implementations. These are discussed in detail in [Mit98], Chapter 6.

directed graph: See Definition A.2, p. 152.

DSP: Digital Signal Processing.

finite state machine: A machine defined by a finite set of states S, a set of possible input vectors I, a set of possible output vectors O, a state transition function $f : I \times S \rightarrow S$, and an output function $g : I \times S \rightarrow O$.

FIR: Finite Impulse Response.

fixed-point: A binary representation for numbers [Kor02].

floating-point: A binary representation for numbers [Kor02].

genetic algorithm: A heuristic search technique based on biological evolution [RSORS96].

iff: If and only if.

iid: Independent, Identically Distributed.

IIR: Infinite Impulse Response.

injection input: A conceptual device used to model finite precision representation errors through a linearization process. Injection inputs are discussed in Sections 3.1.1 and 4.1.2.

intractable: An intractable problem is one that is unsolvable by any polynomial time algorithm [GJ79].

kurtosis: The fourth order statistical moment [Chu74].

latency: The latency of a resource is equal to the number of clock cycles elapsing between presentation of the inputs to the resource, and the result appearing at the output of the resource. The latency of an algorithm is the number of clock cycles elapsing between the start of the first operation in the algorithm and the end of the last operation in the algorithm.

loop: See Definition 2.3, p. 11.

LSB: Least Significant Bit.

LTI: Linear Time Invariant.

MSB: Most Significant Bit.

NP-hard: Any problem which to which an NP-complete problem may be reduced in polynomial time [GJ79].

Nyquist frequency: The frequency corresponding to half the sampling frequency.

Pareto-optimal point: A point in the design space not dominated by *all* other design objectives.

pdf: Probability Density Function.

pole: A root of the denominator polynomial in a z-domain transfer function.

polynomial time algorithm: An algorithm whose execution time is bounded by a polynomial in problem instance size [GJ79].

PSD: Power Spectral Density [Mit98].

recursive: See Definition 2.3, p. 11.

resource binding: The mapping of operations on to physical computational units.

resource dominated: A resource dominated circuit is one in which the overwhelming area consumption is due to the resources required to perform operations rather than the interconnect between these operations.

saturated Gaussian: See Definition 5.6, p. 87.

saturation arithmetic: The use of arithmetic components which saturate to the maximum positive (or negative) value on detection of a positive (or negative) overflow condition.

saturation computation graph: See Definition 5.4, p. 84.

saturation nonlinearity: See Definition 5.2, p. 83.

saturation system: See Definition 5.3, p. 84.

saturator: See Definition 5.1, p. 81.

scaling: The scaling of a signal is determined by the position of the binary point in the signal representation. The terms 'binary point position' and 'scaling' are used interchangeably.

SDF: Synchronous Data Flow [LM87b].

sequencing graph: See Definition 6.1, p. 6.1.

sign-extension: The process of extending the number of bits used to represent a signed number by duplication of the most significant bit [TW01].

SIMD: Single Instruction Multiple Data. A processor organization where a single instruction stream controls many processing elements [Fly72].

skewness: A third order statistical moment [Chu74].

structural description: A circuit description defining the structure of the circuit in terms of the blocks from which it is built together with the interconnections between those blocks *c.f.* behavioural description.

throughput: A measure of the rate at which data are produced and consumed.

transfer function: A z-domain function representing the input-output behaviour of a linear time invariant (LTI) system. The transfer function is the z-transform of the impulse response.

transitive orientation: A graph $G(V, E)$ is transitively oriented iff $(a, b), (b, c) \in E \Rightarrow (a, c) \in E$.

truncation: The process of reducing the number of bits used to represent a number by ignoring one or more least significant bits.

uniprocessor: A computing system designed around a single processing element.

unit impulse: A signal consisting entirely of zeros at all but one time index (usually time zero), at which the impulse has unit value.

von Neumann processor: A sequential processor which executes a stored program.

VHDL: VHSIC (Very High Speed Integrated Circuit) Hardware Description Language [Per91].

well-connected: See Definition 2.2, p. 11.

white spectrum: A spectrum consisting of all frequencies with equal amplitude.

z-transform: See Definition A.6, p. 152.

References

[Ach93] H. Achatz. Extended 0/1 LP formulation for the scheduling problem in high-level synthesis. In *Proc. EURODAC with EURO-VHDL*, 1993.

[Act90] F. S. Acton. *Numerical Methods that Work*. Mathematical Assoc. of America, Washington, 1990.

[Alt98] Altera Corporation, San Jose. *Altera Databook*, 1998.

[AS70] M. Abramowitz and I. A. Stegun, editors. *Handbook of Mathematical Functions*. Dover Publications, New York, 9th edition, 1970.

[ASU86] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, Reading, Mass., 1986.

[BB90] T. Bose and D. P. Brown. Limit cycles in zero input digital filters due to two's complement quantization. *IEEE Trans. Circuits and Systems*, 37(4), April 1990.

[BM99] D. Brooks and M. Martonosi. Dynamically exploiting narrow width operands to improve processor power and performance. In *Proc. 5th International Symposium on High Performance Computer Architecture*, January 1999.

[Boo51] A. D. Booth. A signed binary multiplication technique. *Quarterly J. Mechan. Appl. Math.*, 4(2):236–240, 1951.

[BP99] K. Bondalapati and V. K. Prasanna. Dynamic precision management for loop computations on reconfigurable architectures. In *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, 1999.

[BP00] A. Benedetti and P. Perona. Bit-width optimization for configurable DSP's by multi-interval analysis. In *Proc. 34th Asilomar Conference on Signals, Systems and Computers*, 2000.

[Cam90] R. Camposano. From behavior to structure: High-level synthesis. *IEEE Design and Test of Computers*, 7(5):8–19, October 1990.

[CCL99] G. A. Constantinides, P. Y. K. Cheung, and W. Luk. Truncation noise in fixed-point SFGs. *IEE Electronics Letters*, 35(23):2012–2014, November 1999.

[CCL00a] G. A. Constantinides, P. Y. K. Cheung, and W. Luk. Multiple precision for resource minimization. In B. Hutchings, editor, *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, April 2000.

[CCL00b] G. A. Constantinides, P. Y. K. Cheung, and W. Luk. Multiple-wordlength resource binding. In H. Gruenbacher and R. Hartenstein, editors, *Field-Programmable Logic: The Roadmap to Reconfigurable Systems*, Lecture Notes in Computer Science. Springer-Verlag, 2000.

[CCL00c] G. A. Constantinides, P. Y. K. Cheung, and W. Luk. Optimal datapath allocation for multiple-wordlength systems. *IEE Electronics Letters*, 36(17):1508–1509, August 2000.

[CCL01a] G. A. Constantinides, P. Y. K. Cheung, and W. Luk. Heuristic datapath allocation for multiple-wordlength systems. In *Proc. Design Automation and Test In Europe*, München, March 2001.

[CCL01b] G. A. Constantinides, P. Y. K. Cheung, and W. Luk. The multiple wordlength paradigm. In *Proc. IEEE Symposium on Field Programmable Custom Computing Machines*, Rohnert Park, CA, April–May 2001.

[CCL02] G. A. Constantinides, P. Y. K. Cheung, and W. Luk. Optimum wordlength allocation. In *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, CA, April 2002.

[CH02] M. Chang and S. Hauck. Précis: A design-time precision analysis tool. In *Proc. IEEE Symposium on Field Programmable Custom Computing Machines*, 2002.

[Chu74] K.-L. Chung. *A Course in Probability Theory*. Academic Press, New York, 1974.

[Chv79] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, August 1979.

[CRS+99] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens. A methodology and design environment for DSP ASIC fixed point refinement. In *Proc. Design Automation and Test in Europe*, München, 1999.

[CSH00] C. Chantrapornchai, E. H.-M. Sha, and X. S. Hu. Efficient design exploration based on module utility selection. *IEEE Trans. Computer Aided Design*, 19(1):19–29, January 2000.

[CSL01] M.-A. Cantin, Y. Savaria, and P. Lavoie. An automatic word length determination method. In *Proc. IEEE International Symposium on Circuits and Systems*, pages V–53 – V–56, 2001.

[CW01] G. A. Constantinides and G. J. Woeginger. The complexity of multiple wordlength assignment. *To appear in Applied Mathematics Letters*, 2001.

[DC] Synopsys design compiler. http://www.synopsys.com/products/logic/logic.html.

[DeM94] G. DeMicheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, New York, 1994.

[DRW98] R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: Task graphs for free. In *Proc. CODES/CASHE'98*, pages 97–101, 1998.

[DW] Designware technical bulletin. http://www.synopsys.com/news/pubs/designware_tb.html.

[Eva] B. L. Evans. Raster image processing on the TMS320C7X VLIW DSP. http://www.ece.utexas.edu/~bevans/hp-dsp-seminar/07_C6xImage2/sld001.htm.

[FGL01] J. Frigo, M. Gokhale, and D. Lavenier. Evaluation of the Streams-C C-to-FPGA compiler: An applications perspective. In *Proc. ACM/SIGDA International Symposium on FPGAs*, 2001.

[Fio98] P. D. Fiore. Lazy rounding. In *Proc. IEEE Workshop on Signal Processing Systems*, pages 449–458, 1998.

[Fle81] R. Fletcher. *Practical Methods of Optimization, Vol. 2: Constrained Optimization*. Wiley and Sons, New York, 1981.

[Fly72] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, September 1972.

[FRE93] FREETEL. Esprit project 6166: FREETEL database, 1993.

[Gar90] W. Gardner. *Introduction to Random Processes*. McGraw-Hill, New York, 1990.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[GN72] R. S. Garfinkel and G. L. Nemhauser. *Integer Programming*. John Wiley and sons, New York, 1972.

[Gol80] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

[Haf] L. Hafer. BonsaiG. http://www.cs.sfu.ca/∼lou/BonsaiG.

[Hay96] S. S. Haykin. *Adaptive Filter Theory*. Prentice Hall, Englewood Cliffs, New Jersey, 1996.

[HLH91] 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*, 10(4):464–475, April 1991.

[HMSS01] J. Hwang, B. Milne, N. Shirazi, and J. Stroomer. System level tools for DSP in FPGAs. In R. Woods and G. Brebner, editors, *Proc. Field Programmable Logic*. Springer-Verlag, 2001.

[HO93] I. G. Harris and A. Orailoğlu. Intertwined scheduling, module selection and allocation in time-and-area constrained synthesis. In *Proc. IEEE International Conference on Circuits and Systems*, pages 1682–1685, 1993.

[HSCL00] S. D. Haynes, J. Stone, P. Y. K. Cheung, and W. Luk. Video image processing with the Sonic architecture. *IEEE Computer*, 33(4):50–57, April 2000.

[Hwa79] K. Hwang. *Computer Arithmetic: Principles, Architecture and Design*. Wiley and Sons, New York, 1979.

[IEE85] IEEE standard for binary floating-point arithmetic, 1985. IEEE Std 754-1985.

[IEE86] IEEE standard for logic circuit diagrams, 1986. ANSI/IEEE Std 991-1986.

[IEE99] IEEE standard for VHDL register transfer level (RTL) synthesis, 1999. IEEE Std 1076.6-1999.

[IM91] M. Ishikawa and G. De Micheli. A module selection algorithm for high-level synthesis. In *Proc. IEEE International Symposium on Circuits and Systems*, pages 1777–1780, 1991.

[IO96] C. Inacio and D. Ombres. The DSP decision: Fixed point or floating? *IEEE Spectrum*, 33(9):72–74, September 1996.

[Jac70] L. B. Jackson. On the interaction of roundoff noise and dynamic range in digital filters. *Bell Syst. Tech. J.*, 49:159–184, February 1970.

[Jai90] R. Jain. MOSP: Module selection for pipelined designs with multi-cycle operations. In *Proc. IEEE International Conference on Computer Aided Design*, pages 212–215, 1990.

[JPP88] R. Jain, A. Parker, and N. Park. Module selection for pipelined synthesis. In *Proc. 25th ACM/IEEE Design Automation Conference*, pages 542–547, 1988.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Tatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, New York, 1972.

[KKS98] S. Kim, K. Kum, and W. Sung. Fixed-point optimization utility for C and C++ based digital signal processing programs. *IEEE Trans. on Circuits and Systems II*, 45(11):1455–1464, November 1998.

[KKS00] K.-I. Kum, J. Kang, and W. Sung. AUTOSCALER for C: An optimizing floating-point to integer C program convertor for fixed-point digital signal processors. *IEEE Trans. Circuits and Systems II*, 47(9):840–848, September 2000.

[Kor02] I. Koren. *Computer Arithmetic Algorithms*. A K Peters, Massachusetts, 2002.

[KR78] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[KS98] K. Kum and W. Sung. Word-length optimization for high-level synthesis of digital signal processing systems. In *Proc. IEEE International Workshop on Signal Processing Systems SIPS'98*, pages 569–678, 1998.

[KS01] K.-I. Kum and W. Sung. Combined word-length optimization and high-level synthesis of digital signal processing systems. *IEEE Trans. Computer Aided Design*, 20(8):921–930, August 2001.

[KWCM98] H. Keding, M. Willems, M. Coors, and H. Meyr. FRIDGE: A fixed-point design and simulation environment. In *Proc. Design Automatation and Test in Europe*, 1998.

[Lin97] Y.-L. Lin. Recent developments in high-level synthesis. *ACM Trans. Design Automation of Electronic Systems*, 2(1):2–21, January 1997.

[Liu71] B. Liu. Effect of finite word length on the accuracy of digital filters – a review. *IEEE Trans. Circuit Theory*, CT-18(6):670–677, 1971.

[LJ00] H.-M. Lin and J.-Y. Jou. On computing the minimum feedback vertex set of a directed graph by contraction operations. *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems*, 19(3):295–307, March 2000.

[LKHP97] C. Lee, D. Kirovski, I. Hong, and M. Potkonjak. DSP QUANT: Design, validation, and applications of DSP hard real-time benchmark. In *Proc. IEEE International Conference on Acoustics Speech and Signal Processing*, volume 1, pages 671–682, 1997.

[LL88] H. Levy and D. W. Low. A contraction algorithm for finding small cycle cutsets. *J. Algorithms*, 9:470–493, 1988.

[LM87a] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous dataflow programs for digital signal processing. *IEEE Trans. Computers*, January 1987.

[LM87b] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *IEEE Proceedings*, 75(9), September 1987.

[LMD94] B. Landwehr, P. Marwedel, and R. Dömer. OSCAR: Optimum simultaneous scheduling, allocation and resource binding. In *Proc. European Design Automation Conference*, pages 90–95, 1994.

[LMV88] A. Lepschy, G. A. Mian, and U. Viaro. Effects of quantization in second-order fixed-point digital filters with two's complement truncation quantizers. *IEEE Trans. Circuits and Systems*, 35(4), April 1988.

[LP93] L. E. Lucke and K. K. Parhi. Generalized ILP scheduling and allocation for high-level DSP synthesis. In *IEEE Custom Integrated Circuits Conference*, pages 5.4.1–5.4.4, 1993.

[MAT] MATLAB. http://www.mathworks.com.

[MAX] MaxPlus II. http://www.altera.com.

[McF90] M. C. McFarland. The high-level synthesis of digital systems. *IEEE Proceedings*, 78(2):301–318, 1990.

[Mit98] S. K. Mitra. *Digital Signal Processing*. McGraw-Hill, New York, 1998.

[Moo66] R. E. Moore. *Interval Analysis*. Prentice-Hall, 1966.

[MS01] M. Mehendale and S. D. Sherleker. *VLSI Synthesis of DSP Kernels*. Kluwer Academic, 2001.

[NHCB01] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee. Precision and error analysis of MATLAB applications during automated hardware synthesis for FPGAs. In *Proc. Design Automation and Test in Europe*, pages 722–728, Munich, Germany, 2001.

[NT86] J. Nestor and D. Thomas. Behavioral synthesis with interfaces. In *Proc. IEEE International Conference on Computer Aided Design*, pages 112–115, 1986.

[OR70] J. Ortega and W. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, New York, 1970.

[OS75] A. V. Oppenheim and R. W. Schafer. *Digital Signal Processing*. Prentice-Hall, New Jersey, 1975.

[OW72] A. V. Oppenheim and C. J. Weinstein. Effects of finite register length in digital filtering and the fast fourier transform. *IEEE Proceedings*, 60(8):957–976, 1972.

[P50] International Telecommunications Union P.50 standard.

[Par99] K. Parhi. *VLSI Digital Signal Processing Systems*. Wiley and sons, New York, 1999.

[Per91] D. L. Perry. *VHDL*. McGraw-Hill, New York, 1991.

[PFTV88] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, Cambridge, 1988.

[PK89] P. G. Paulin and J. P. Knight. Force-directed scheduling for the behavioral synthesis of ASICs. *IEEE Trans. Computer-Aided Design*, 8:661–679, June 1989.

[PKBL96] K. Premaratne, E. C. Kulasekere, P. H. Bauer, and L.-J. Leclerc. An exhaustive search algorithm for checking limit cycle behavior of digital filters. *IEEE Trans. Signal Processing*, 44(10), October 1996.

[PW96] A. Peleg and U. Weiser. MMX technology extension to the Intel architecture. *IEEE Micro*, 16:42–50, August 1996.

[PWB92] K. K. Parhi, C.-Y. Wang, and A. P. Brown. Synthesis of control circuits in folded pipelined DSP architectures. *IEEE J. Solid-State Circuits*, 27:29–43, January 1992.

[RSORS96] V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith. *Modern Heuristic Search Methods*. Wiley and Sons, 1996.

[SBA00] M. Stephenson, J. Babb, and S. Amarasinghe. Bitwidth analysis with application to silicon compilation. In *Proc. SIGPLAN Programming*

Language Design and Implementation, Vancouver, British Columbia, June 2000.

[Sch97] H. Schwab. lp_solve. ftp://ftp.es.ele.tue.nl/pub/lp_solve, 1997.

[SIM] Simulink. http://www.mathworks.com.

[SK95] W. Sung and K. Kum. Simulation-based word-length optimization method for fixed-point digital signal processing systems. *IEEE Trans. Signal Processing*, 43(12):3087–3090, December 1995.

[SLC98] N. Shirazi, W. Luk, and P. Y. K. Cheung. Run-time management of dynamically reconfigurable designs. In *Proc. IEEE Symposium on FPGAs for Custom Computing Machines*, 1998.

[SS91] A. S. Sedra and K. C. Smith. *Microelectronic Circuits*. Saunders, 1991.

[Sta98] T. Stathaki. Root moments: A digital signal-processing perspective. *IEE Proc. Vis. Image Signal Process.*, 145(4):293–302, August 1998.

[Ste00] M. W. Stephenson. Bitwise: Optimizing bitwidths using data-range propagation. Master's thesis, Massachusetts Institute of Technology, Dept. Electrical Engineering and Computer Science, May 2000.

[SW75] G. W. Smith and R. B. Walford. The identification of a minimal feedback vertex set of a directed graph. *IEEE Trans. Circuits and Systems*, CAS-22(1):9–15, January 1975.

[TI] TMS320C6000 high performance DSPs. http://dspvillage.ti.com.

[TONH96] M. Trembley, M. O'Connor, V. Narayan, and L. He. VIS speeds new media processing. *IEEE Micro*, 16(4):10–20, August 1996.

[TW01] R. J. Tocci and N. J. Widmer. *Digital Systems: Principles and Applications*. Prentice-Hall, New Jersey, 8th edition, 2001.

[WP98] S. A. Wadekar and A. C. Parker. Accuracy sensitive word-length selection for algorithm optimization. In *Proc. International Conference on Computer Design*, pages 54–61, Austin, Texas, October 1998.

[Xil03] Xilinx, Inc., San Jose. *Field Programmable Gate Arrays*, 2003.

Index