DEVELOPMENT OF A HIGH LEVEL SYNTHESIS TOOL SPECIALIZED ON FIR-BASED MULTIRATE SYSTEMS

by
Arda Yurdakul
B.S. in E.E., Boğaziçi University, 1992
M.S. in E.E., Boğaziçi University, 1994

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor
of
Philosophy

Boğaziçi University
1999

ACKNOWLEDGEMENTS

I would like to thank my advisor Assoc. Prof. Dr. Günhan Dündar for his invaluable guidance and friendship during this thesis. Also I would like to express my gratitude to Prof. Dr. Sabih Tansal and Prof. Dr. Ömer Cerid as they have continually supported my studies in Boğaziçi University. Similarly, I am grateful to the jury members, Assoc. Prof. Dr. Kuban Altınel and Assoc. Prof. Dr. Yusuf Leblebici, for reading my thesis and sparing time to come to listen my presentation.

Special thanks go to a PhD candidate like me, Tuna Tarım, who has proved to be my real friend by her long e-mail messages caring about my problems and anxieties, and sharing hers with me in spite of several miles between us.

Above all, I am especially indebted to my family, including my three-year-old nephew Oğuz who has waited for me to finish writing this thesis so that we would play games again.

Finally, this thesis is dedicated to my dear friend Alper Atalay who passed away so young...

ABSTRACT

Digital Signal Processing (DSP) is the most studied area in design automation, because it is one of the most well-established branches of electrical engineering for several years. In the last few years, it is stimulated by the progression of multirate techniques. The rapid development on multirate digital signal progressing is complemented by the emergence of new applications. The key property of multirate algorithms is their computational efficiency.

In this thesis, a silicon compiler is developed to reduce design time for the hardware realization of FIR-based multirate DSP algorithms. This is a brand new study, because there does not exist a silicon compiler of this type according to our knowledge. Although multirate algorithms contain decimators and interpolators changing the effective sample rate, the design of synchronous systems using a single-clock signal is possible by this newly developed tool. The designer can achieve this by folding nodes of similar type into a single node. Additionally, the FIR filters followed by a decimator or following an interpolator can be entered as a single node while defining a system at the input of the tool. Also multiplications with the tap coefficients in FIR-based nodes in a fold are handled at the same time to exploit common terms so as to realize those multiplications without multipliers. As a result, the tool produces very efficient layouts in terms of area, power and clock signals. It can also determine the quantization levels of tap coefficients in FIR-based nodes and fractional parts of data bus if the system output error is specified. It also handles module selection under given power, area and delay constraints and scheduling like other well-known silicon compilers. The compiler is programmed to process bit-parallel-digit-serial architectures.

KISA ÖZET

Sayısal işaret işleme, elektrik mühendisliğinin en yerleşik alanlarından biri olduğundan tasarım otomasyonunda en çok çalışılan konulardan biridir. Son yıllarda geliştirilen çoklu hızlı tekniklerle yeni uygulama alanları ortaya çıkmıştır. Çoklu hızlı algoritmaların en önemli özelliği işlemsel verimliliktir.

Bu tezde, Sonlu Dürtü Yanıtlı (SDY) tabanlı çoklu hızlı sayısal sistemlerin donanım olarak gerçeklenirken geçen tasarım zamanını kısaltmak için bir silikon derleyici geliştirilmiştir. Bu çalışma, bundan sonra geliştirilecek bu tip derleyicilerin, bildiğimiz kadarıyla, ilk örneğidir. Çoklu hızlı sistemlerde bulunan seyrelticiler ve genleştiriciler etkin örnek hızını değiştirmektedir. Bu da donanım olarak gerçeklenirken farklı saat sinyallerini zorunlu kılar. Oysa benzer işlemleri yapan düğümlerin tasarımcı tarafından doğru bir şekilde tek bir düğüme katlanmasıyla derleyici tek bir saat sinyaliyle bütün sistemi gerçekleyebilmektedir. Ayrıca bir SDY süzgeci, bir SDY süzgecini izleyen bir seyreltici, bir genleştirciyi izleyen bir SDY süzgeci tipindeki işlemler tek bir düğüm olarak derleyicinin girişinde kullanılan sistem tanım dosyasına yazılabilmektedir. Bir katta bulunan SDY süzgeçlerinin katsayılarındaki benzer terimler kullanılarak bu çarpım işlemleri çarpıcısız olarak gerçeklenmektedir. Sonuç olarak derleyici alan, güç ve saat işaretleri bakımından verimli devre planları üretmektedir. Bütün bu özelliklerinin yanında derleyicimiz, herhangi bir SDY-tabanlı çoklu hızlı sistemin çıkışındaki hata verildiğinde, her SDY-tabanlı düğümdeki katsayıların ve veriyolunun tamsayı olmayan kısmının sınırlama seviyelerini hesaplayabilmektedir. Ayrıca diğer tüm silikon derleyicilerinin yaptığı gibi kullanıcı tarafından belirlenen alan, zaman ve güç kısıtlarına göre kütüphanelerden eleman seçer ve bunların işlemsel zamanlamasını yapar. Derleyicimiz bit-paralel-basamak-seri tipindeki işaretleri işlemek üzere proglanmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

LIST OF SYMBOLS

a	ancestor in ancestor set A_n, A_{N_f}.
a_n	area of node n.
a_{sys}	area of the system.
$a_{\max}$	maximum area in set $\Lambda \cup Q$.
A_n	ancestor set for node n.
A_{N_f}	ancestor set for node set N_f.
b	fractional data bus.
b_{bit}	number of bits for fractional data bus quantization.
B	fractional bus quantizaiton set.
c	coefficient.
c_{bit}	number of bits for coefficient quantization.
c_q	quantized coefficient.
$clock$	operational clock period of the system.
C	coefficient set.
C_Θ	the set for the coefficients with odd number of non-zero entries in CSD representation.
d_n	delay of node n.
d_{sys}	delay of the system.
$d_{\max}$	maximum delay in set $\Lambda \cup Q$.
D	decimation factor.
e	root-mean-squared error.
$E[.]$	expected value.
f	fold.
F	fold set.
g	function.
G, H	FIR filters.
i, j, m	general purpose variable.
I	Interpolation factor.
J	level.

k	tap order in FIR-based nodes.
K	number of taps in FIR-based nodes.
L	band.
M	module selection array.
n	node.
n_l	wordlength of node n.
n_{l_r}	remaining wordlength of node n during module selection.
N	node set as given in DFG.
N_f	node set as given in DFG for fold f.
$O(.)$	"big oh", order of growth.
p_n	power of node n.
p_{sys}	power of the system.
$p_{\max}$	maximum power in set $\Lambda \cup Q$.
q	a module in library Q.
Q	module(i.e., arithmetic unit) library.
r	a replica in replica set R.
R	replica set.
R_c	replica set for c'th coefficient such that $\cup_{c \in C} R_c = R$.
$R_{c,j}$	replica set for j'th non-zero entry in c'th coefficent such that $\cup_{j \in c \cap j \neq 0} R_{c,j} = R_c$.
R_t	replica set for t'th two-term such that $\cup_{t \in T} R_t = R$.
s_a, s_d, s_p	optimization scales for a_{sys}, d_{sys}, and p_{sys} respectively.
S	a universal set for $C \cup B$.
S_ψ	the set of coefficent and bus quantization for path $\psi \in \Psi$ such that $\cup_{\psi \in \Psi} S_\psi = S$.
t	a two-term in set T.
T	the set of two-terms.
u, v	Lagrange multipliers.
x	input or source.
y	output or sink.
z	binary variable.
α	ancestor process start time.

β	scaling of different bands.
γ_b, γ_c	bus and coeffiecnet quantization scales for optimization.
δ	delay in DFG (i.e., edge weight).
Δ_a, Δ_b	stepsize for coefficient quantization.
Δ_{bus}	stepsize for fractional data bus quantization.
Δc	error signal on coefficient c to produce c_q.
Δy	error signal on output y to produce y_q.
ϵ	mean-squared error (i.e. e^2).
ζ	coefficient stepsize scaling (i.e. $\frac{\Delta_a}{\Delta_b}$).
η_ψ	path error for path $\psi \in \Psi$.
θ	operation start time.
Θ	set of non-zero entries for coefficents with odd number of non-zero entries.
Θ_c	set of non-zero entries for coefficent c such that $\Theta_c \in \Theta$, $\forall c \in C_\Theta$.
ϑ	a non-zero entry in Θ or Θ_c.
κ	number of arithmetic units for realizing a node of n_{l_τ} length.
λ	a latch in latch library Λ.
Λ	latch library.
ξ	scheduling table entry.
π_τ	period of scheduling table τ.
ϖ	probability density function.
ρ	rate.
σ^2	variance.
ς	scaling constant for distinction of an adder or negator during module selection.
τ	scheduling table.
Υ	set of scheduling tables.
φ	variable for life time analysis.
ψ	a path in set Ψ.
Ψ	set of all possible paths from inputs to outputs.
Ω	scale for the determination of stepsize for coefficient quantization.
$\aleph$	enhanced node set of N.

ABBREVIATIONS

ALU	Arithmetic Logic Unit
CSD	Canonic Signed Digit.
DFG	Data Flow Graph.
FIR	Finite Impulse Response.
HLS	High Level Synthesis.

1. INTRODUCTION

The tremendous progress of semiconductor technology in recent decades has led to the continuous increase in the integration level of electron devices on a single substrate. As a result, extremely complex circuits have been fabricated at higher integration rates.

From the designer's point of view, it will be harder to cope with the design errors as the complexity of the circuit increases. Also, as feature size decreases, the chip must be designed more carefully, because the factors neglected at a larger feature size may become effective. These factors will cause the design time to be longer, hence design cost will increase. On the other hand, integration of several circuits on the same chip will perform better as a whole, because it will be free of problems due to external bus-width limitations, packaging constraints, interconnection delays, and device parasitics.

High integration rate is desirable by the manufacturer, because the same circuit can be fabricated on a smaller chip area, reducing the production cost. However, this is valid only if the volume of sales of a chip is large enough to recapture the design and manufacturing costs. In reality, only a few circuit applications, such as general purpose microprocessors, can enjoy high volume of sales and a long life. Improvement in the circuit technology makes circuits obsolete in a very short time. The manufacturer must sell the products of the related technology as soon as possible for an appreciable profit. This means that a chip must be available on market before its technology is out of date. Therefore, design time must decrease. Consequently, error-prone designs must be eliminated at the very beginning to minimize design and verification time. Employing more design engineers to cope with this problem is an extra burden on the production cost.

At this stage, silicon compilers come into the picture. They exploit Computer-Aided Design (CAD) techniques for optimizing circuit quality and reducing design time. Synthesis techniques speed up design cycle and human effort. Optimization techniques enhance design quality. Engineers spend time only on designing basic building blocks, such as multipliers, of the required technology. Once the circuit specification is given,

the silicon compiler itself generates the optimal layout using those pre-designed building blocks residing in the libraries. As a result, design time of a circuit within given specifications reduces appreciably. It must be kept in mind that the whole design may not be optimal. Nevertheless, the manufacturer will be happy to earn much more money than before because he has been selling the acceptable product at the right time.

The power of synthesis and optimization techniques is not fully exploited in the available silicon compilers. Therefore, intensive studies on silicon compilers continue at universities and commercial institutions. Silicon compilers handling digital circuits are studied much more than the ones dealing with analog circuits. From here on, digital silicon compilers are meant by the phrase 'silicon compilers'.

The circuits must be modeled before silicon compilers are used to synthesize and optimize them. Models can be classified according to their levels of abstraction and views. The famous Y- Chart set forth by Gajski and Kuhn in [1] is used to show the relation between views and levels of abstraction. Its enhanced version using [2] is shown in FIGURE 1.1. The segments of Y stand for three views: structural, behavioral and geometrical. Behavioral views describe the function of the circuit. Structural views model the circuit in terms of block interconnections. Geometrical views deal with the real physical elements of the circuit. The dashed boxes in the Y- Chart stand for levels of abstraction of the corresponding views: architectural, logic and physical. At the architectural level, the circuit is the set of operations. At the logic level, it is a set of logic functions. At the physical level, it is a set of geometrical entities.

Synthesis is a set of transformations between two axial views at the same level of abstraction. So there are three synthesis tasks: architectural, logic and physical. In architectural synthesis (also named as high-level synthesis), structural view of the circuit is obtained from behavioral view at the architectural level of abstraction. This is the assignment of operations to circuit components including their execution timings and component interconnections. In logic synthesis, structural view of the circuit is obtained from behavioral view at the logic level of abstraction. It is the generation of logic models as an interconnection of logic primitives stored in libraries using logic specifications of the circuit. Since generation of geometrical view is done at the physical level, usually the name physical design is used in place of physical synthesis. Positions

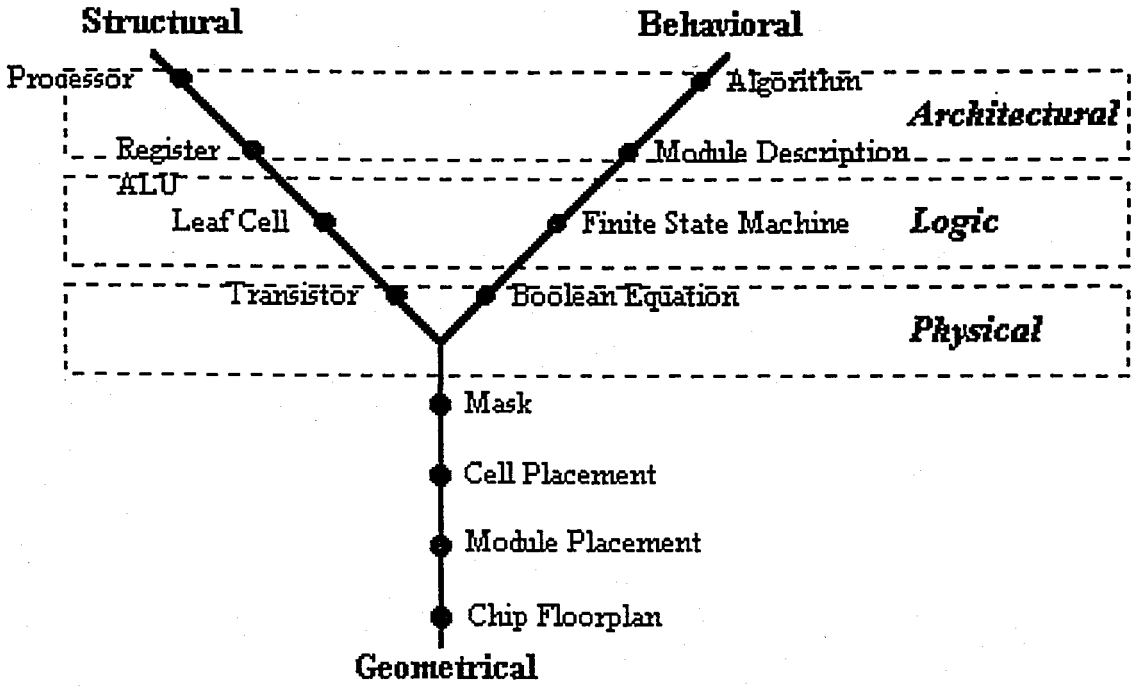


FIGURE 1.1. Enhanced Y-chart.

of the geometric entities in the layout and layout generation is carried out using this synthesis tool. The organization of these synthesis tools during silicon compilation is shown in FIGURE 1.2.

Optimization tools are used in conjunction with synthesis tools to generate testable circuits at the marginal constraints determined by area, performance and recently, power. Combined optimization is carried out using all of the constraints which are given as upper bounds on area and power, lower bounds on performance. These constraints determine the design evaluation space, on which the optimization process is carried.

During design of a circuit, the modified Y-Chart in [3] is used as shown in FIGURE 1.3 to include the test stage. The transition between the traditional axial views are straightforward and explained roughly above. The additional dashed axis for testability is not the view of the circuit but a stage during the design process. This means that at all stages, the physical circuit is tested and at the occurrence of fault, a transition will be done to the structural view to redesign the circuit. This is called as the structured design, and some of the arrows coming to and going from test axis can

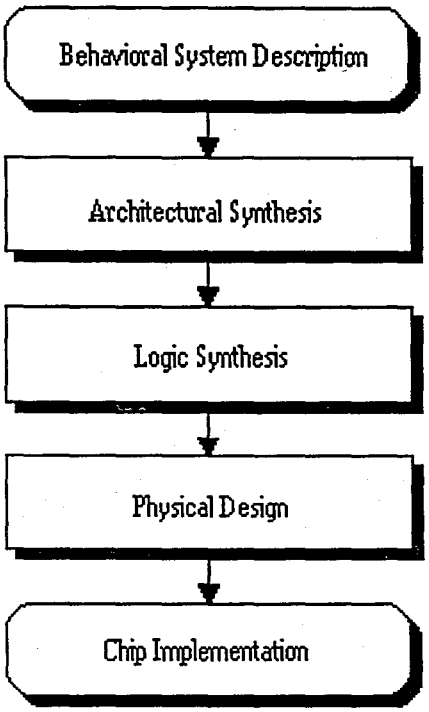


FIGURE 1.2. Organization of synthesis tools during silicon compilation.

be removed depending on the designers' choice.

Synthesis tools used in silicon compilers are not mature at the same level. There has been an intense study on physical design automation since 1965. At first, known techniques for PCB design were adapted. Later on, these were combined with optimization techniques and modified to meet the specifications of multilayer chip designs. Nowadays, it is the most well-established part of silicon compilation as stated in [4].

Logic synthesis tools have evolved similar to physical design tools. Classical ones are based on combinatorial optimization algorithms and find a minimum implementation of the specified functions. Today's logic synthesis tools are capable of binding the combinational components to a given library [2], [5].

High-level synthesis (HLS) is the hottest research area in silicon compilation since early 1980s. However, apart from a few exceptions, a smooth transition from technology to industry has failed. The main reasons are related to lack of integrated methodology, including design entry, simulation and synthesis rather than the quality of synthesis algorithms as stated in [6]. Most of the HLS tools are optimal with respect to minimum number of resources used or minimum cycle time required to

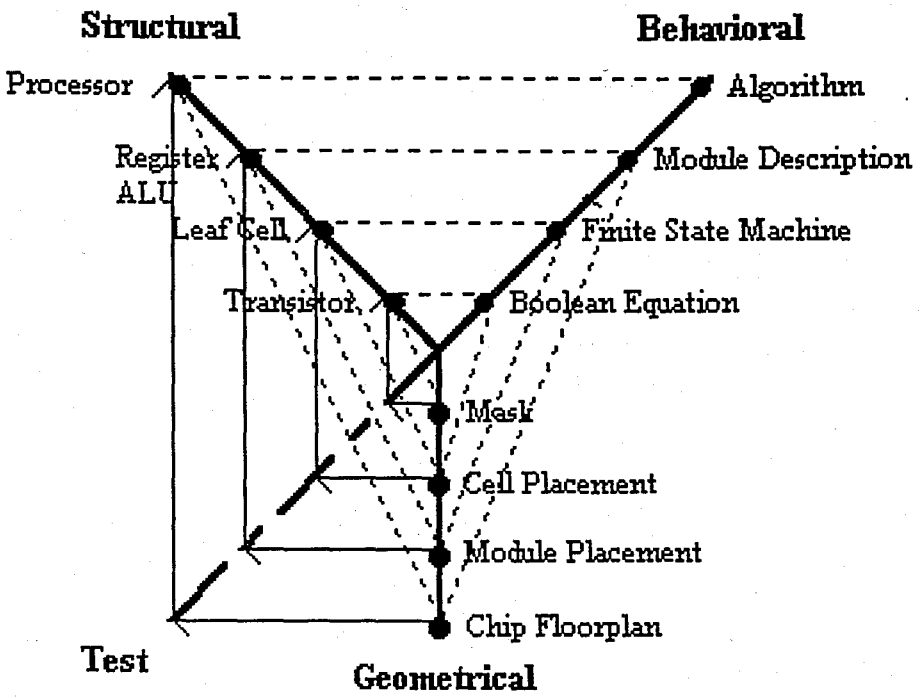


FIGURE 1.3. Modified Y-chart.

carry out a certain job. On the other hand, this optimization does not necessarily lead to smaller or faster final implementation, because size and delay of control logic and all interconnections in the circuit can be significant and they are not usually taken into account. Also, most of the HLS tools are application oriented. For example, Cathedral system is developed for digital signal processing (DSP) applications [7]. The hardware design language (HDL) to the HLS tool is also very important because its semantics and grammar determines the freedom of user interaction on the design. This is necessary for the designer to produce a chip with the desired requirements. Therefore as this freedom increases, the tool becomes more flexible and powerful. The possible organization of jobs in architectural synthesis is given in FIGURE 1.4 to understand TABLE 1.1, which is a brief survey on HLS tools of several silicon compilers. There also exists some scheduling and module selection tools that can be integrated to computer-aided-design tools like Mentor Graphics EDA. Some of them use integer programming techniques for scheduling like [8] and [9]. A newly developed tool uses DFG transformation for multirate systems [10]. The last one also selects modules simultaneously. In another scheduler is designed for scheduling cyclic DFGs using loop pipelining under resource

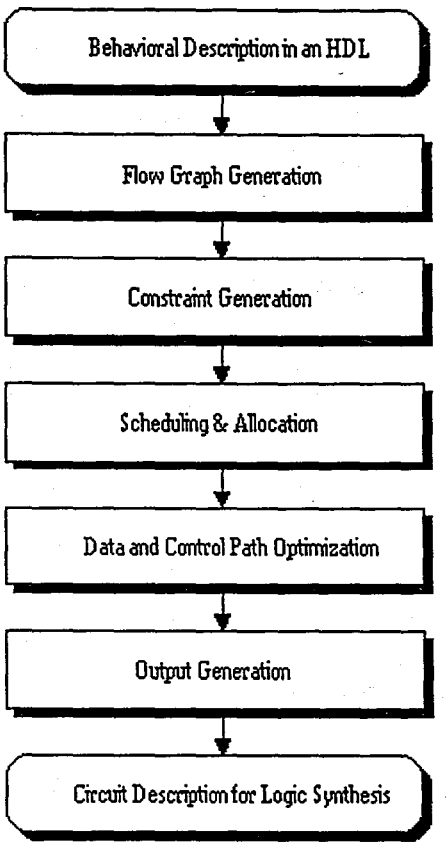


FIGURE 1.4. Organization of jobs in architectural synthesis.

constraints [11]. A cost-optimized algorithm for selecting slow components on non-critical paths and fast components on critical paths has also been developed for high-performance pipelines [12], [13].

HLS tools described above exploit some computer architecture techniques with the conventional optimization ones for optimal final architectures. One of them is the well-known pipelining for synchronous circuits to increase throughput of a machine without changing basic cycle time. If different algorithm instances are executed in an overlapping fashion on a single data path, it is *functional pipelining.* If the operation instances are executed in an overlapping fashion on a single data path, it is *structural pipelining* [14], [15]. Superscalar and superpipelined machine pipelines are modified versions for better performance [16]. Another type of pipelining is *wave pipelining* which is a method of high performance circuit design implementing pipelining in logic without the use of intermediate latches or registers. Even though wave pipelining

has been well-known since 1969, its popularity has begun with the development of sophisticated CAD tools in 1990s [17].

One other technique is *folding* which is the process of executing many algorithm operations in one hardware operator [18], [19]. This technique is useful when matching two systems with different sample rates and comparable number of processors. Folding of systolic arrays and irregular data flow graphs with a given folding set are studied in [16] and [19].

Good old lifetime analysis of [20] is the fundamental of most scheduling and allocation algorithms in HLS tools. It determines the number of storage units. Each storage unit is used to preserve a variable during its lifetime which is the time spanned from its generation to its final reference by any operator [2].

DSP is the most studied area in design automation, because it is one of the most well- established branches of electrical engineering for several years. In the last few years, it is stimulated by the progression of multirate techniques. The rapid development on multirate digital signal progressing is complemented by the emergence of new applications. These include subband coding of speech, audio and video signals, multicarrier data transmission, fast transforms using digital filter banks and discrete wavelet transform of all types of signals. The key property of multirate algorithms is their computational efficiency. As a result, they can be implemented economically using modern digital signal processors. At the same time, typical structures of multirate systems are models for the design of highly efficient architectures for microelectronic devices [21].

Typical structures of multirate systems are used for the design of highly efficient architectures of microelectronic devices. They are based on elliptic, IIR or FIR type digital filters and decimation and/or interpolation units. The IIR filter produces the best result among all filters whereas an elliptic filter is less efficient. The FIR filter is better than the elliptic one if the transition band is wide and is the worst one for a narrow transition band and a single stage multirate system. The comparison of single stage filters taken from [22] is shown in FIGURE 1.5. However, FIR filter is the only one that realizes exact linear phase. Also it can be realized in very efficient forms, both recursive and nonrecursive. In nonrecursive FIR filter-based realizations, stability is guaranteed while implemented on a finite wordlength digital system. Besides, sensi-

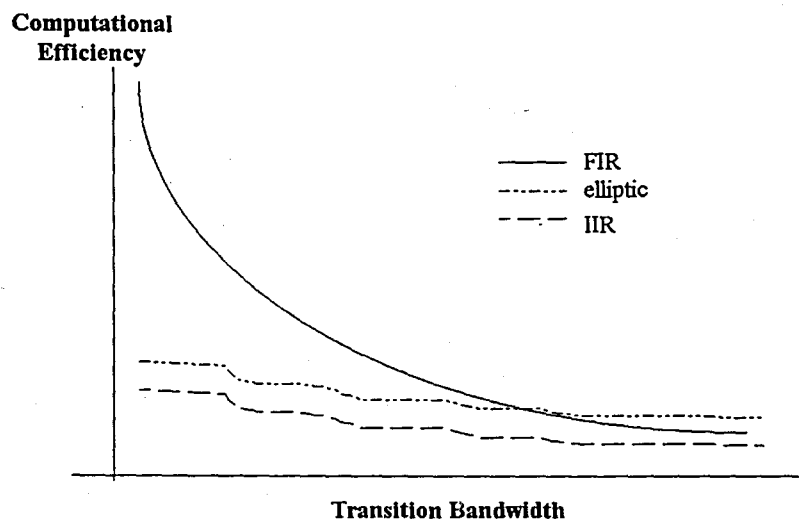


FIGURE 1.5. The comparison of single stage filters in multirate systems.

tivity to filter coefficients is very low. It is very easy to design an FIR filter with any arbitrary specification [23]. The most important feature of an FIR-based cascaded multirate system is that it is nearly as efficient as that of an IIR-based cascaded system. As a result, in the hardware realization of multirate systems, FIR-based structures are used. In the literature, there are several FIR-based hardware implementations of discrete wavelet transform and M-channel filter banks [24]-[32].

1.1. Our HLS Tool

Our HLS tool whose flow diagram is shown in FIGURE 1.6 and FIGURE 1.7, accepts inputs as an enhanced form of ordinary directed data flow graph (DFG). The format and commands that must be used for correct operation of the system can be found in Appendix A. The tool basically incorporates structural pipelining and folding schemes to produce bit-parallel outputs. The inputs must be digit-serial-bit-parallel [33].

The FIR-based blocks (multirate or not) can be entered as single nodes in an

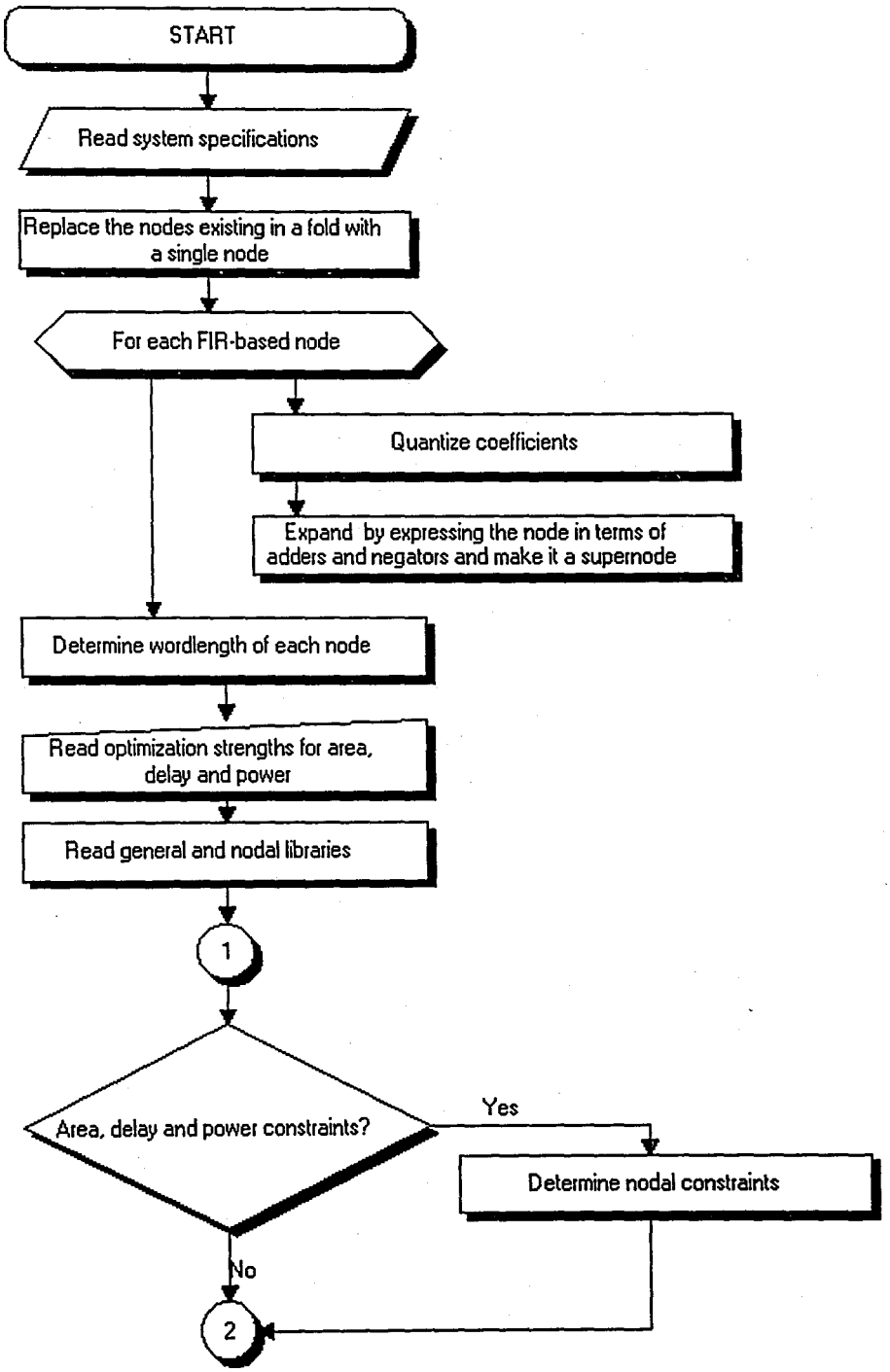


FIGURE 1.6. Organization of our HLS tool.

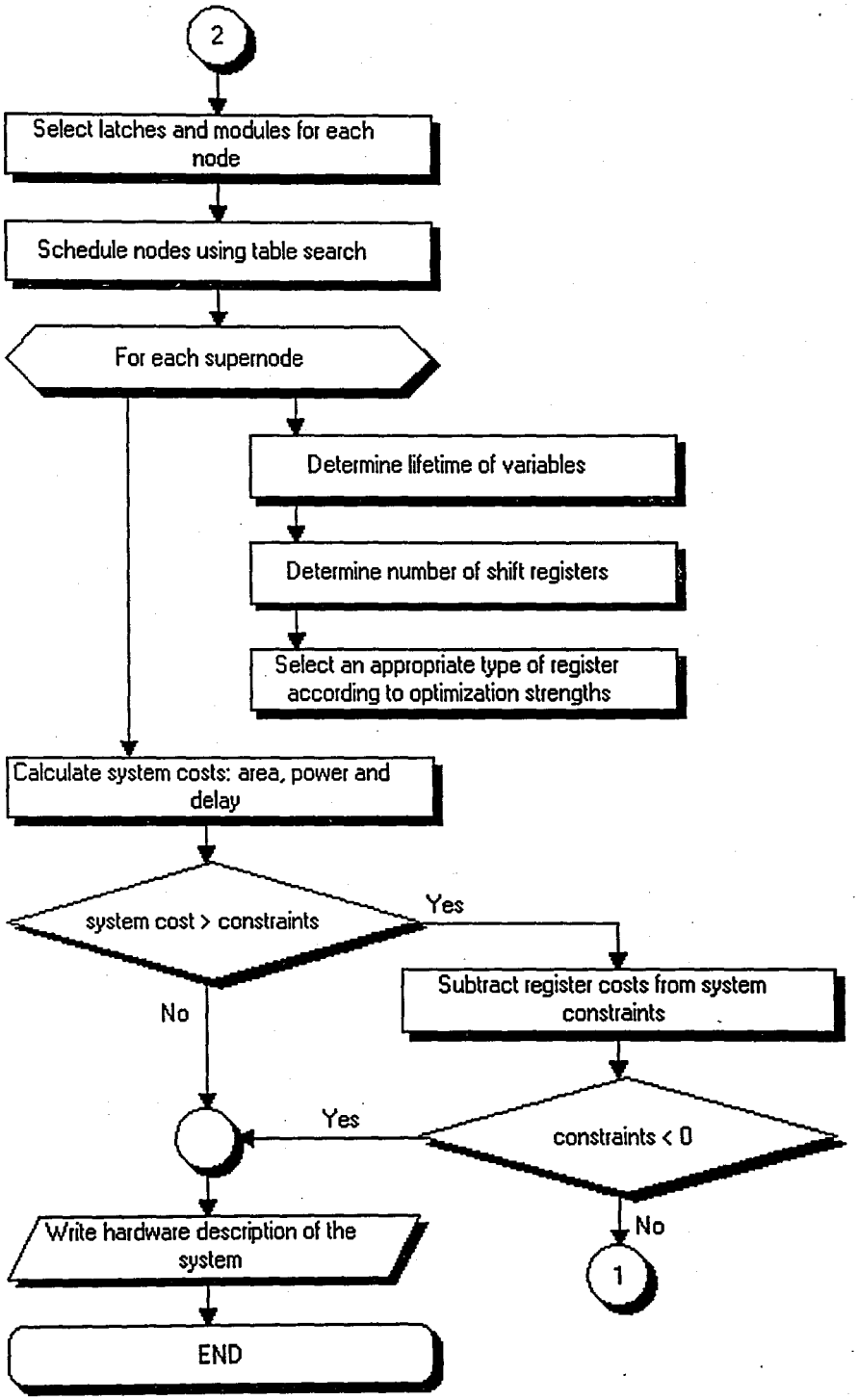


FIGURE 1.7. Organization of our HLS tool (continued).

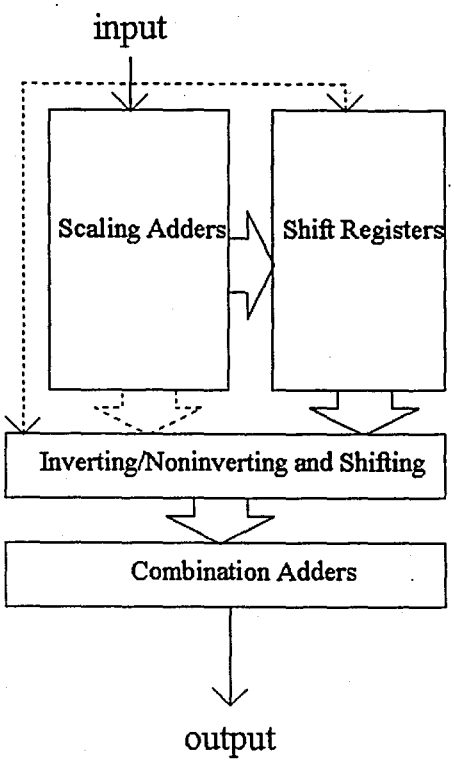


FIGURE 1.8. Target architecture for FIR-based systems.

input file. After quantizing the coefficients of these blocks (Chapter 2.), a seed of quantized coefficients is formed which contains only the numbers that can be used to generate the coefficients by shifting and/or inverting (Chapter 3.). The seed numbers are generated by using Scaling Adders block in FIGURE 1.8 which is the target architecture for all FIR-based nodes. As it can be observed here, after the input is processed by Scaling Adders, they are stored in Shift Registers block. The output is processed by Combination Adders preceeding the Inverting/Noninverting and Shifting block which generates the final version of the processed input. This architecture helps us fold all kinds of FIR-based blocks in a system provided that the tap-length of each FIR-based filter in the system are the same and clock frequency of the system allows such a folding. This operation helps us reducing operators without sacrificing from system quality.

After decomposing all FIR-based nodes into adders, the system is now composed of only adders and multipliers if the multipliers exist explicitly in the input file. After determining wordlengths of all operators, the module selection process is carried out

by using library search for a given technology according to the optimization criterion specified by the user. There are three objectives in module selection: Area, delay and power. The user has to specify the strength for each objective and can put upper bounds on some or all of the objectives. Instead of multiobjective optimization, we preferred additive optimization after normalizing all modules in the library (Chapter 4.). The module selection algorithm is developed to select slow components on non-critical paths and fast components on critical paths for high-performance pipelines. If the upper bounds are infeasible due to poor library elements, the bounds are dropped. Then, scheduling is carried out by table search method to determine the required number of registers (including Shift Registers block in FIGURE 1.8) and delay elements in the systems. Using this number, the area, delay and power cost due to these elements is determined and it is checked against the upper bounds if there are any. In case of insatisfaction, their costs are subtracted from upper bounds and the module selection process is carried out again. This process is carried until the best result satisfying the bounds is found. A design example is given in Chapter 5.

This HLS tool can be improved by using the future research topics that are explained in Chapter 6.

TABLE 1.1 HLS tools of some silicon compilers with their notable properties.

Silicon Compiler or the HLS tool (Application Area)	*Scheduling and Allocation Schemes*
Sehwa (ASIC design) [14]	Resource allocation table generation based on resource sharing and followed by pipelined scheduling
HAL (ASIC design) [15]	Force-directed scheduling under fixed resource constraints
V-Compiler (ASIC design) [20]	Percolation scheduling followed by lifetime analysis for resource allocation
Cathedral-II (DSP) [34]	As-Soon-As-Possible (ASAP) scheduling under fixed resource constraints
PHIDEO (video applications) [35]	Data-path synthesis using retiming followed by an improved force-directed scheduling under fixed resource constraints
LAGER (DSP) [18]	Probabilistic resource allocation followed by a scheduling using discrete relaxation
HIS (synchronous systems) [6]	As-Fast-As- Possible scheduling and resource allocation using graph coloring techniques
CALLAS (synchronous systems) [36]	Maximally parallel and chained resource allocation followed by unconstrained ASAP scheduling
FAMOS (DSP) [37]	Kernighan and Lin's graph partitioning algorithm [4] for scheduling under fixed resource constraints
JOSHUA (DSP) [38]	Simultaneous scheduling, allocation and binding using integer linear programming techniques
MARS (DSP) [39]	Concurrent scheduling and resource allocation based on iterative loops exploiting inter-iteration and intra-iteration constraints
MATISSE (digital systems) [40]	Static, dynamic and pipelined scheduling with different clocking schemes and synchronization under fixed resources
WAVESCHED (synchronous systems) [41]	Wave-like scheduling exploiting inherent parallelism such that independent loops share the same resources

2. COEFFICIENT QUANTIZATION IN FIR-BASED NODES

On implementation of multirate systems in software applications, filter coefficients are assumed to be of infinite or very high precision and this is usually the case when implementing several FIR-based algorithms. However, the picture is not quite the same for the hardware realization of these algorithms because of finite silicon area. Therefore, filter coefficients must be quantized; i.e., limited to a predetermined wordlength, causing output signal degradation. This can be minimized by excessively increasing the wordlength, but circuit complexity, area, and operation power increases beyond the constraints set by the hardware system design engineers. The common practice is to find an optimum wordlength for an acceptable error at the output. Several studies have been made on finite wordlength effects [42]-[49].

In this study, statistical methods are used to estimate the wordlength of a fixed-point FIR-based multirate filter. The general equation for a K-tap FIR filter is

$$y[m] = \sum_{k=0}^{K-1} x[m-k]\, c[k]. \tag{2.1}$$

As it is also evident from this equation, the input signal x is convolved with the coefficient array c of the K-tap FIR to produce the m'th instance of the output signal, y. The output of a D-decimating and I-interpolating multirate filter can be written as

$$y[Dm] = \sum_{k=0}^{K-1} x[Im-k]\, c[k]. \tag{2.2}$$

Interpolation and decimation processes should not harm the nature of the signal for perfect reconstruction. The filter coefficients are rounded and they still meet the system specifications with some additional error. However, the choice of quantization stepsize is important to decrease the error. In this analysis, all filter coefficients are limited to the same wordlength; i.e., bit assignment process is not used. This helps us to use folding that help us shrink the effective chip area [19]. The analysis can be extended to handle all basic hardware structures explained above. In J-level cascaded designs,

data bus limitations have to be taken into account, because common trend is the usage of a fixed-size data bus for both inter and intra chip operations. Statistical methods can be used to estimate error due to limited bus size; i.e., bus quantization. Then, a model can be easily formed to find the optimal chip area for a given tolerable error at the output.

In the following section, a statistical method which estimates the error with the given wordlength of the coefficients and extends the analysis to cascaded systems will be introduced. Also an analysis will be carried to determine the stepsize for coefficient quantization. Section 2.2 discusses the quantization of fractional part of the data bus. In Section 2.3 a model is formed to solve the problem which can be expressed as "Given the tolerable error at the output, what is the optimal chip area?". A solution methodology for optimality is also given. Section 2.4 presents examples and experiment results.

2.1. Effects of Coefficient Quantization

The analysis will first be carried out for a single multirate filter. Then, it will be generalized to cover the analysis of J-level and L-band multirate systems. Throughout the analysis, the input signal is assumed to be scaled so that no overflow occurs for the integer part of the data bus. We have to make such an assumption due to the dynamic limitation of fixed point arithmetic. The method of scaling is given in detail in [50]. Therefore, from here on, the terms "input" and "output" are used for scaled input and output.

2.1.1. Effects of Coefficient Quantization in a Multirate Filter

Upon quantization of filter coefficients, error, Δc, is introduced to original coefficients in order to obtain quantized values, c_q. Therefore, the degraded signal at the

output, y_q, and the error term of the output, Δy, can be given as follows:

$$\begin{aligned} c_q[k] &= c[k] + \Delta c[k] \\ y_q[Dm] &= y[Dm] + \Delta y[Dm] = \sum_{k=0}^{K-1} x[Im-k]\,(c[k] + \Delta c[k]) \end{aligned} \tag{2.3}$$

$$\Delta y[Dm] = \sum_{k=0}^{K-1} x[Im-k]\,\Delta c[k] \tag{2.4}$$

We define error due to coefficient quantization, e_c, at the output as

$$e_c = \sqrt{\frac{E\left[(\Delta y)^2\right]}{E\left[(y)^2\right]}} = \sqrt{\frac{\sigma_{\Delta y}^2}{\sigma_y^2}} \tag{2.5}$$

where $E\left[(.)^2\right]$ is the power of the function inside and it is equivalent to signal variance σ^2 if it is a zero-mean random process.

For most applications, coefficients are fractional numbers between -1 and 1. Therefore, fixed-point quantization of coefficients by rounding with sufficiently large number of bits is enough for their realization. Uniformity of this process is essential to preserve system linearity. Rounding can be viewed as a zero-mean random process uniformly distributed between $-\Delta/2$ and $+\Delta/2$ where Δ is the quantization step size. Then, the variance of quantization error will be given as indicated in [50]:

$$E\left[\Delta c^2\right] = \sigma_{\Delta c}^2 = \frac{\Delta^2}{12}. \tag{2.6}$$

We can model coefficient distribution as a zero-mean random process which is uniform between $-c_{\max}$ and $+c_{\max}$ where $c_{\max}$ is the absolute maximum of all coefficients. Even though they are not always symmetric around the origin, modelling the distribution this way does not effect the analysis in an adverse manner. The probability density function (p.d.f.) of c, ϖ_c, is simply $1/2c_{\max}$ in the occurrence region of c. The power of coefficient distribution is

$$E\left[c^2\right] = \sigma_c^2 = \int_{-c_{\max}}^{c_{\max}} c^2 \varpi_c dc = \frac{c_{\max}^2}{3}. \tag{2.7}$$

The input x, the coefficient c, and the quantization Δc are assumed to be

statistically independent. Keeping this in mind, the power of Δy can be derived as follows:

$$\begin{aligned} E\left[\Delta y^2\right] &= \int \Delta y^2 \varpi_{\Delta y} d\Delta y \\ &= \int_{-\Delta/2}^{\Delta/2} \int_{x_{\min}}^{x_{\max}} \left(\sum_{k=0}^{K-1} x\left[Im-k\right] \Delta c\left[k\right] \right)^2 \varpi_x \varpi_{\Delta c} dx d\Delta c, \quad \forall Dm. \end{aligned} \tag{2.8}$$

If it is also assumed that the samples of x, c and Δc are uncorrelated, then the cross terms of (2.8) will vanish to zero; only squared terms will survive:

$$\begin{aligned} E\left[\Delta y^2\right] &= \int_{-\Delta/2}^{\Delta/2} \int_{x_{\min}}^{x_{\max}} \sum_{k=0}^{K-1} \left(x\left[Im-k\right] \Delta c\left[k\right] \right)^2 \varpi_x \varpi_{\Delta c} dx d\Delta c \\ &= \sum_{k=0}^{K-1} \left(\int_{-\Delta/2}^{\Delta/2} \Delta c\left[k\right]^2 \varpi_{\Delta c} d\Delta c \right) \left(\int_{x_{\min}}^{x_{\max}} x\left[Im-k\right]^2 \varpi_x dx \right) \\ &= \sum_{k=0}^{K-1} \sigma_{\Delta c[k]}^2 E\left[x\left[Im-k\right]^2 \right] \end{aligned} \tag{2.9}$$

Since the power of the input signal within the convolution window is less than or equal to the overall input signal power, $E\left[x\left[Im-k\right]^2\right] = E\left[x^2\right]$ for the extreme case. Also, $\sigma_{\Delta c[k]}^2 = \sigma_{\Delta c}^2$. Therefore, the output error signal power is simply

$$E\left[\Delta y^2\right] = K \sigma_{\Delta c}^2 E\left[x^2\right] = K \frac{\Delta^2}{12} E\left[x^2\right]. \tag{2.10}$$

We can compute output signal power, $E\left[y^2\right]$, in a similar manner to obtain the following result:

$$E\left[y^2\right] = K \sigma_c^2 E\left[x^2\right] = K \frac{c_{\max}^2}{3} E\left[x^2\right] \tag{2.11}$$

Then by using (2.5), (2.10) and (2.11), e_c can be derived as

$$e_c = \frac{\Delta}{2c_{\max}}. \tag{2.12}$$

However, a slowly varying signal has correlated samples, so the cross terms of (2.8) will survive to some extent. Assuming that all cross terms exist, (2.8) can be rewritten as:

$$
\begin{aligned}
E\left[\Delta y^2\right] &= \int_{-\Delta/2}^{\Delta/2}\int_{x_{\min}}^{x_{\max}}\left(\sum_{k=0}^{K-1}\left(x\left[Im-k\right]\Delta c\left[k\right]\right)^2\right)w_x w_{\Delta c}dxd\Delta c \\
&+\int_{-\Delta/2}^{\Delta/2}\int_{x_{\min}}^{x_{\max}}\left(\sum_{k=0}^{K-1}\left(x\left[Im-k\right]\Delta c\left[k\right]\sum_{j\neq k}^{K-1}x\left[Im-j\right]\Delta c\left[j\right]\right)\right)w_x w_{\Delta c}dxd\Delta c \\
&\leq \int_{-\Delta/2}^{\Delta/2}\int_{x_{\min}}^{x_{\max}}\left(\sum_{k=0}^{K-1}\left(x\left[Im-k\right]\Delta c\left[k\right]\right)^2\right)w_x w_{\Delta c}dxd\Delta c \\
&+(K-1)\int_{-\Delta/2}^{\Delta/2}\int_{x_{\min}}^{x_{\max}}\left(\sum_{k=0}^{K-1}\left(x\left[Im-k\right]\Delta c\left[k\right]\right)^2\right)w_x w_{\Delta c}dxd\Delta c \\
&\leq K\int_{-\Delta/2}^{\Delta/2}\int_{x_{\min}}^{x_{\max}}\sum_{k=0}^{K-1}\left(x\left[Im-k\right]\Delta c\left[k\right]\right)w_x w_{\Delta c}dxd\Delta c \\
&\leq K\sum_{k=0}^{K-1}\left(\int_{-\Delta/2}^{\Delta/2}\Delta c\left[k\right]^2 w_{\Delta c}d\Delta c\right)\left(\int_{x_{\min}}^{x_{\max}}x\left[Im-k\right]^2 w_x dx\right) \\
&\leq K\sum_{k=0}^{K-1}\sigma_{\Delta c[k]}^2 E\left[x\left[Im-k\right]^2\right]
\end{aligned}
\tag{2.13}
$$

$$
E\left[\Delta y^2\right] = K^2\sigma_{\Delta c}^2 E\left[x^2\right] = K^2\frac{\Delta^2}{12}E\left[x^2\right]. \tag{2.14}
$$

Note that the final equation shows that worst case output error signal power for correlated input samples is K times (2.10). Then by using (2.5), (2.11) and (2.14), worst case error due to coefficient quantization, $e_{c,\max}$, can be found as:

$$
e_{c,\max} = \frac{\sqrt{K}\Delta}{2c_{\max}}. \tag{2.15}
$$

On the other hand, it must be kept in mind that $e_{c,\max}$ is an unrealizable upper bound because the input with correlated samples will not only effect output error signal, but also output signal as well. However, $e_{c,\max}$ solidly determines the feasibility region of e_c; i.e.,

$$
e_c < e_{c,\max} \tag{2.16}
$$

2.1.2. Effects of Coefficient Quantization in a J-Level Multirate System

Assume a cascaded multirate system as shown in FIGURE 2.1. We can write the signal at the output of the j'th level as follows:

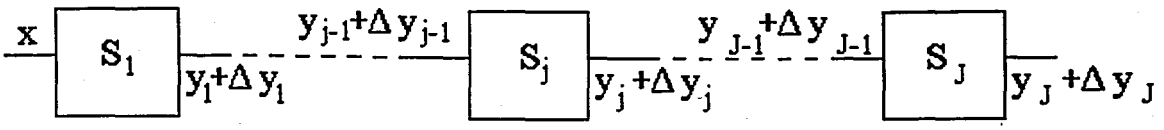


FIGURE 2.1. A typical cascaded multirate system. S_i stands for an FIR filter with a decimator and/or an interpolator.

$$y_j[Dm] + \Delta y_j[Dm] = \sum_{k=0}^{K_j-1} (y_{j-1}[Im-k] + \Delta y_{j-1}[Im-k])(c_j[k] + \Delta c_j[k]) \quad (2.17)$$

Here each stage can have a different coefficient set, c_j, and evidently different number of taps, K_j. The output signal and the output error signal of j'th level can be written using above the equation as follows:

$$y_j[Dm] = \sum_{k=0}^{K_j-1} y_{j-1}[Im-k]\, c_j[k] \quad (2.18)$$

$$\Delta y_j[Dm] \cong \sum_{k=0}^{K_j-1} (y_{j-1}[Im-k]\, \Delta c_j[k] + \Delta y_{j-1}[Im-k]\, c_j[k]) \quad (2.19)$$

The output signal power for j'th level can be found using the above methodology to obtain the result

$$E\left[y_j^2\right] = K_j \sigma_{c_j}^2 E\left[y_{j-1}^2\right] \quad (2.20)$$

Since the system is cascaded, (2.20) can be used iteratively to find output signal power for a J-level multirate system as

$$E\left[y_J^2\right] = E\left[x^2\right] \prod_{k=1}^{J} K_k \sigma_{c_k}^2. \quad (2.21)$$

The output error signal power can be found similarly as,

$$E\left[\Delta y_j^2\right] = K_j \sigma_{\Delta c_j}^2 E\left[y_{j-1}^2\right] + K_j \sigma_{c_j}^2 E\left[\Delta y_{j-1}^2\right] \quad (2.22)$$

Because of the system's defined structure, (2.22) can be used iteratively with (2.20):

$$\begin{aligned} E\left[\Delta y_j^2\right] &= K_j\sigma_{\Delta c_j}^2 K_{j-1}\sigma_{c_{j-1}}^2 E\left[y_{j-2}^2\right] \\ &+K_j\sigma_{c_j}^2\left(K_{j-1}\sigma_{\Delta c_{j-1}}^2 E\left[y_{j-2}^2\right]+K_{j-1}\sigma_{c_{j-1}}^2 E\left[\Delta y_{j-2}^2\right]\right) \\ &= K_jK_{j-1}\left(\sigma_{\Delta c_j}^2\sigma_{c_{j-1}}^2 E\left[y_{j-2}^2\right]+\sigma_{c_j}^2\sigma_{\Delta c_{j-1}}^2 E\left[y_{j-2}^2\right]+\sigma_{c_j}^2\sigma_{c_{j-1}}^2 E\left[\Delta y_{j-2}^2\right]\right) \end{aligned} \tag{2.23}$$

As it can be seen from the above result, the intermediate stages affect the output only by their coefficients and their respective quantization errors. Note that the Δ symbol appears only once in each summation term, and it does not occur more than once for every variable throughout the summation operation. This will yield the following result for a J-level system from output to the input:

$$\begin{aligned} E\left[\Delta y_J^2\right] = \left(\prod_{j=1}^{J} K_j\right) \; & (\sigma_{\Delta c_J}^2\sigma_{c_{J-1}}^2\sigma_{c_{J-2}}^2\ldots\sigma_{c_1}^2 E\left[x^2\right] \\ & +\sigma_{c_J}^2\sigma_{\Delta c_{J-1}}^2\sigma_{c_{J-2}}^2\ldots\sigma_{c_1}^2 E\left[x^2\right] \\ & +\sigma_{c_J}^2\sigma_{c_{J-1}}^2\sigma_{\Delta c_{J-2}}^2\ldots\sigma_{c_1}^2 E\left[x^2\right] \\ & +\ldots+\sigma_{c_J}^2\sigma_{c_{J-1}}^2\sigma_{c_{J-2}}^2\ldots\sigma_{\Delta c_1}^2 E\left[x^2\right] \\ & +\sigma_{c_J}^2\sigma_{c_{J-1}}^2\sigma_{c_{J-2}}^2\ldots\sigma_{c_1}^2 E\left[\Delta x^2\right]) \end{aligned} \tag{2.24}$$

Then, error of a J-level multirate system, $e_{c,J}$, can be found by using (2.5), (2.21), (2.24)

$$e_{c,J} = \sqrt{\frac{E\left[\Delta x^2\right]}{E\left[x^2\right]}+\sum_{j=1}^{J}\frac{\sigma_{\Delta c_j}^2}{\sigma_{c_j}^2}} = \sqrt{\frac{E\left[\Delta x^2\right]}{E\left[x^2\right]}+\sum_{j=1}^{J}\frac{\Delta_j^2}{4c_{j,\max}^2}}. \tag{2.25}$$

If the input signal has no error term, i.e. $\Delta x = 0$, and $J = 1$, then (2.25) reduces to (2.12) as expected. If $J > 1$ with similar stages as in wavelet transform, then a result which can be easily estimated will be obtained:

$$e_{c,J} = \frac{\sqrt{J}\Delta}{2c_{\max}} \tag{2.26}$$

We can find maximum error for a J-level system, $e_{c,J,\max}$, by following the strategy for the single level case.

$$e_{c,J,\max} = \sqrt{\left(\prod_{j=1}^{J} K_j\right)\left(\frac{E\left[\Delta x^2\right]}{E\left[x^2\right]}+\sum_{j=1}^{J}\frac{\Delta_j^2}{4c_{j,\max}^2}\right)} \tag{2.27}$$

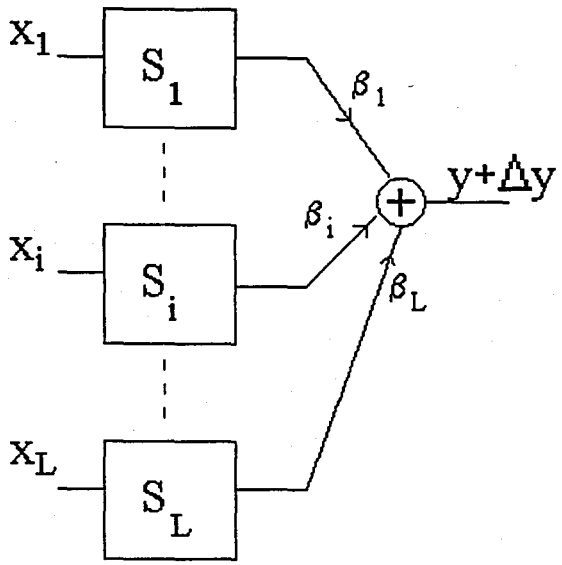


FIGURE 2.2. A typical multiband multirate filter. S_i stands for a cascaded system.

Equation (2.25) usually becomes sufficient to estimate error in cascaded multirate filters for realistic data, because correlation between subsequent samples decreases as a result of decimation and interpolation. This is obvious in decimation case. In interpolation case, zeroes are padded between subsequent samples before processing in the filter. Therefore, the correlation between the samples of the new data array is very close to zero in the overall array.

A multirate system can be formed by also adding the outputs of several cascaded multirate filters in a signed manner. Such a system is depicted in FIGURE 2.2. Here, the outputs of all filters can be multiplied by signed numbers, β_i, and added to each other. Then, the output signal and the output error signal for an L-band system can be written as follows:

$$\begin{aligned} y &= \textstyle\sum_{i=1}^{L} \beta_i y_i \\ \Delta y &= \textstyle\sum_{i=1}^{L} \beta_i \Delta y_i \end{aligned} \tag{2.28}$$

Then, the error at the output can be derived using (2.5) together with the above equations. It can be easily observed that the overall error will be at most the maximum on a branch;

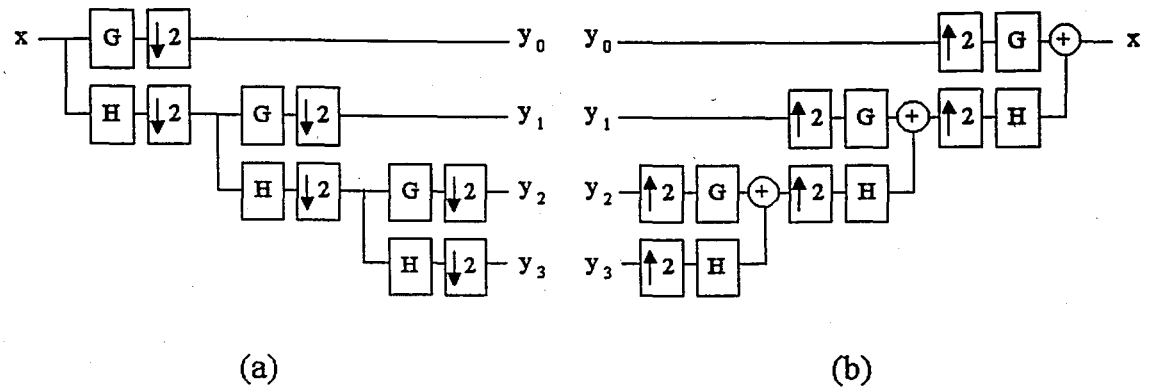


FIGURE 2.3. A two-band-three-level wavelet transform pair: (a)analysis part, (b)synthesis part.

$$e_{c,L}^2 \leq \max_{1\leq i\leq L} \left(\beta_i^2 e_{c,i}^2\right) \tag{2.29}$$

and equality can be used for worst case conditions. Here, $e_{c,i,\max}$ can also be used in place of $e_{c,i}$ to find maximum error for worst case conditions.

2.1.3. Determination of Coefficient Quantization Step Size

It is known that there is a relation between the step size Δ and number of bits used for quantization, c_{bit}, for signed numbers as follows:

$$\Delta = \Omega 2^{-(c_{bit}-1)} \tag{2.30}$$

where Ω is a scaling constant and one bit is reserved for sign. Therefore, Equation (2.25) can be rewritten as

$$e_{c,J} = \sqrt{\frac{E\left[\Delta x^2\right]}{E\left[x^2\right]} + \sum_{j=1}^{J} \frac{\Omega_j^2 2^{-2c_{bit_j}}}{c_{j,\max}^2}}. \tag{2.31}$$

Now, if similar stages are used with $J > 1$ as in wavelet transform shown in FIGURE 2.3, then c_{bit} can be found using (2.26) in conjunction with (2.30) as follows:

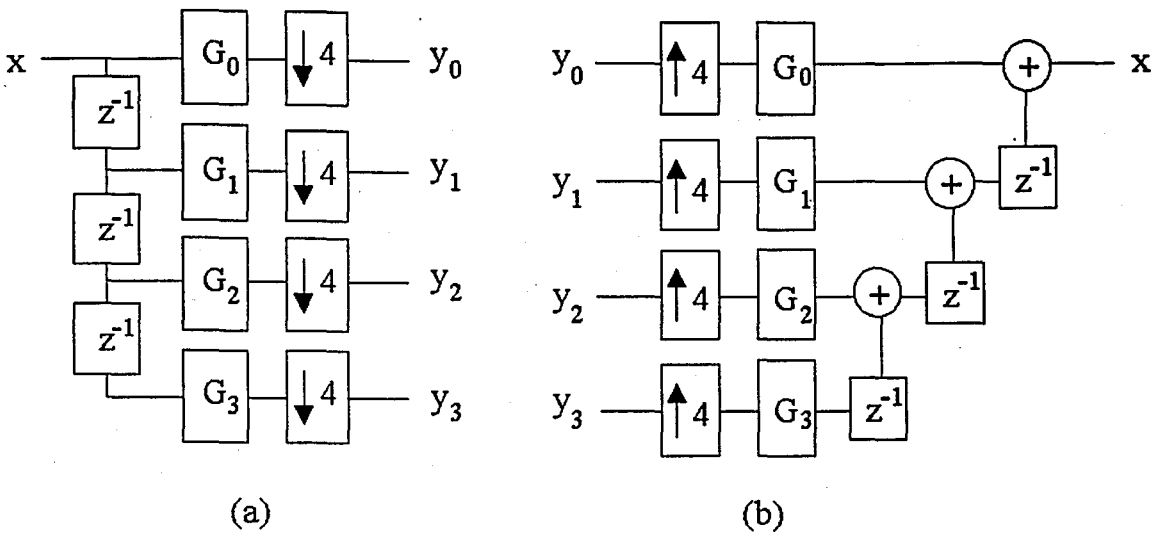


FIGURE 2.4. A four-band wavelet transform: (a)analysis part, (b)synthesis part.

$$c_{bit} = -\log_2 \frac{c_{\max} e_J}{\Omega \sqrt{J}} \tag{2.32}$$

An L-band wavelet transform is made up of one level branches as shown in FIGURE 2.4, but the longest branch of an L-band J-level wavelet transform is at the J'th level. In the light of Equation (2.32), it can be claimed that as the level of the wavelet transform increases, more bits are required to satisfy a given error criterion. It must be noted that the final equation is not related to the band number L of the filter.

In the above analysis, the significance of step size constant Ω was not mentioned. Usually it is taken as 1:

$$\Delta = \Delta_a = 2^{-(c_{bit}-1)} \tag{2.33}$$

So Equation (2.31) can be rewritten as

$$e_{c,J} = \sqrt{\frac{E\left[\Delta x^2\right]}{E\left[x^2\right]} + \sum_{j=1}^{J} \frac{2^{-2c_{bit_j}}}{c_{j,\max}^2}}. \tag{2.34}$$

However, this process does not guarantee the utilization of all bits efficiently. Also while transmitting the data through the channel, it is more likely that it will be contaminated with noise. Therefore, it will be better to decrease the step size to alleviate the effect of channel noise on the signal and to express the coefficients with more accuracy. This

can be achieved by taking Ω any number smaller than one. On the other hand, the maximum number must be expressed within the given wordlength. Therefore,

$$|c_{\max}| \leq \Omega \leq 1. \tag{2.35}$$

Taking Ω to be the smallest value which is equal to $|c_{\max}|$, stepsize can be redefined as follows:

$$\Delta = \Delta_b = c_{\max} \tag{2.36}$$

then the error of the output signal, $e_{c,J}$, using Equation (2.31), can be written as,

$$e_{c,J} = \sqrt{\frac{E\left[\Delta x^2\right]}{E\left[x^2\right]} + \sum_{j=1}^{J} 2^{-2c_{bit_j}}}. \tag{2.37}$$

which is absolutely less than the value defined by Equation (2.33), since $|c_{\max}| < 1$.

Another drawback of designs using Δ_a as the step size is the mismatch between effective wordlength and the desired wordlength. This phenomenon for Çağlar's eight tap wavelet transform coefficient set [51], is shown in FIGURE 2.5. This is completely dependent on the coefficient set. However, such a case does not occur when using Δ_b as the step size, because of full exploitation of the entire wordlength.

As a result, usage of Δ_b instead of Δ_a is preferable for coefficient quantization. However, scaling process is necessary at the output as $\Omega_j \neq 1$. This is not problematic for systems having all paths at the same level. An example can be an L-band single level wavelet transform pair as shown in FIGURE 2.4 where $L = 4$. Here, since both analysis and synthesis parts are symmetric, scaling of output at the synthesis part by ζ^2, where ζ is given as

$$\zeta = \frac{\Delta_b}{\Delta_a}, \tag{2.38}$$

is sufficient as shown in FIGURE 2.6. Scaling problem must be handled more seriously for a tree with branches at different levels. An example can be devised to understand this more easily. Think of a two-band three-level wavelet design with its analysis and synthesis parts as shown in FIGURE 2.3. Both analysis and synthesis parts are

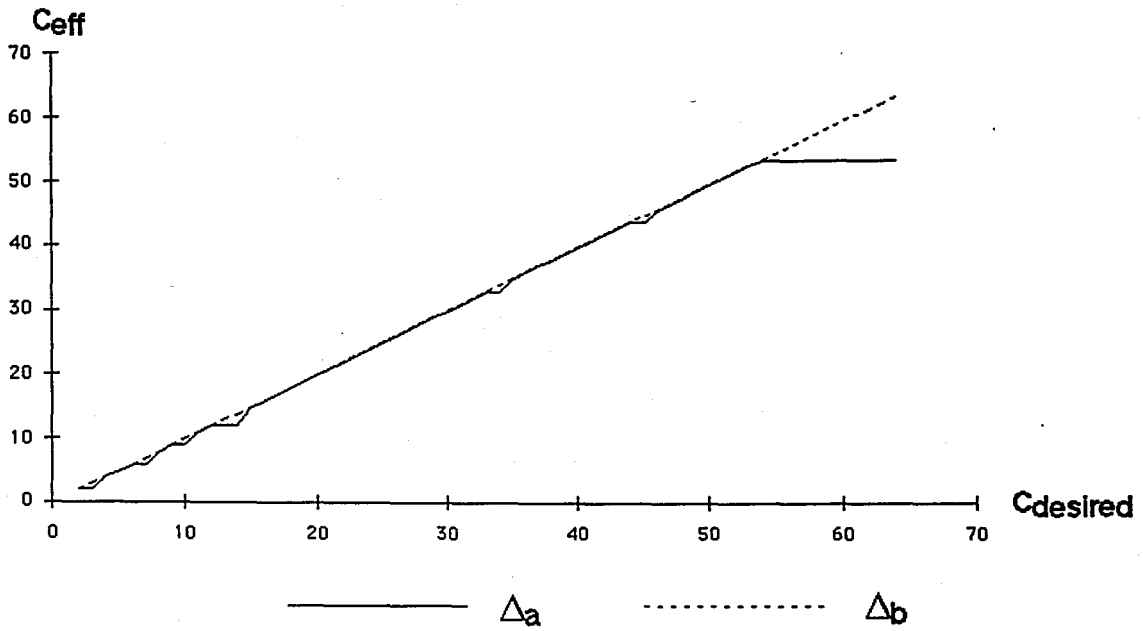


FIGURE 2.5. The effective wordlength vs. the desired wordlength due to the quantization stepsize. B stands for Δ_b and A stands for Δ_a.

connected via a channel to obtain the paths shown in FIGURE 2.7. As it can be seen, there are four paths with different levels: 2, 4, 6, and 6. This means that the output cannot be scaled as it is done in the previous case. Then, throughout the design process, if Δ_b is used in the analysis part, then it must be switched to Δ_a in the synthesis part as shown in FIGURE 2.8. In this case, it is obvious that the silicon area is larger for the synthesis part. Therefore, one can prefer to use Δ_a throughout the design process. On the other hand, it must be kept in mind that the previously described hybrid design is less sensitive to channel noise.

2.2. Effects of Bus Quantization

In the above analysis, it is assumed that the data bus is varying with the coefficient quantization wordlength, meaning that the width of the output data bus is different than that of the input data bus at each stage. For example, if the input

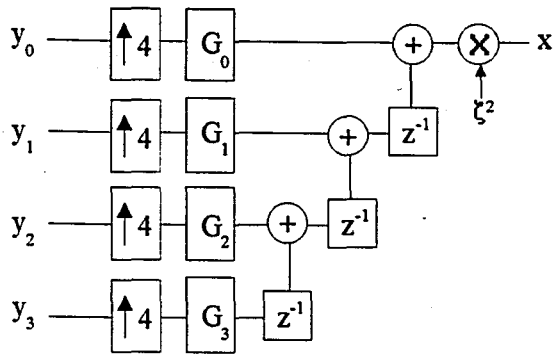


FIGURE 2.6. Realization of FIGURE 2.4.b. when the analysis part is realized using Δ_b.

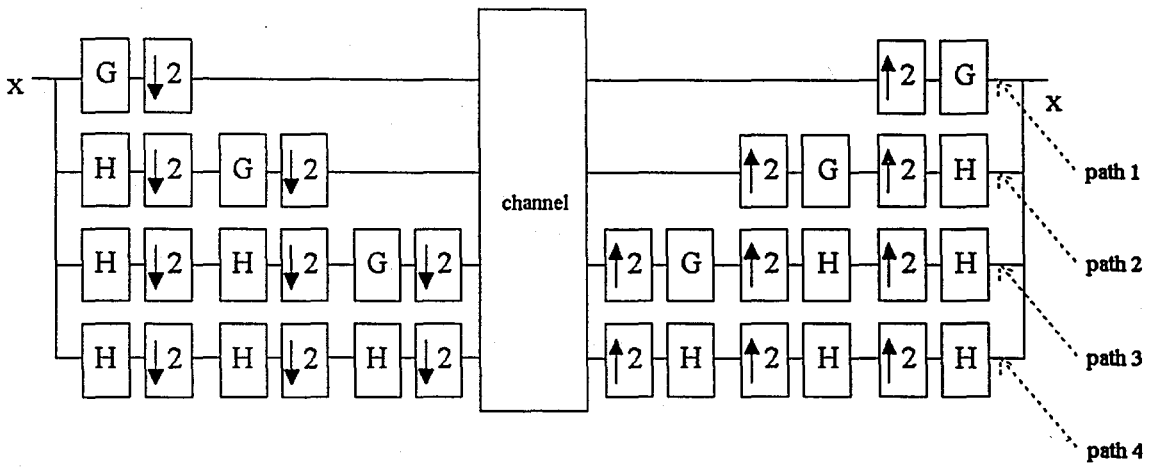


FIGURE 2.7. Path decomposition of the system formed by connecting analysis and synthesis parts of FIGURE 2.3. via a channel.

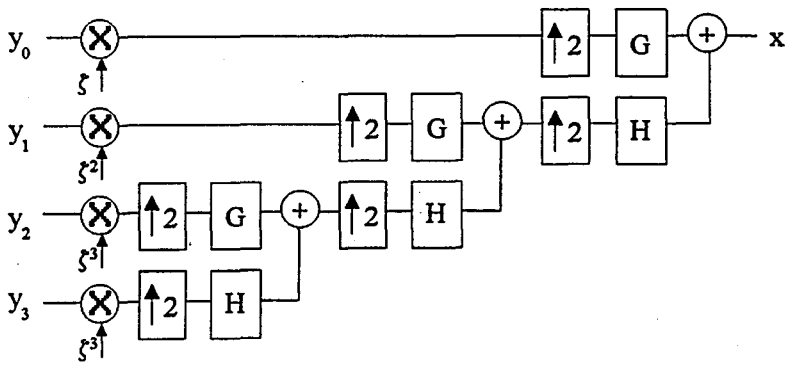


FIGURE 2.8. Realization of FIGURE 2.3.b. when the analysis part is realized using Δ_b.

data bus is 8 bits wide and the coefficient precision of the filter is limited to 8 bits again, then the bus width at the output will be 16 bits. If a similar filter follows this one, then the output bus-width will be 24 bits. This situation can be allowable in the chip, but at the output ports, they must be limited because of the maximum allowable output pins for the chip. Besides, the data bus may get large enough to consume more area than the main processing and controlling units. Therefore, in the chip design, bus wordlength is usually fixed. Data bus carries both integer and fractional parts of the data. Bus wordlength limitation is carried out on fractional data bus. From here on, the term 'bus' will be used for output bus of the multirate module.

2.2.1. Effects of Bus Quantization at the Output of a Single Level Multirate Filter

Bus quantization can be done either by truncation or rounding. In both cases, some error is added to the output signal as follows:

$$y + \Delta y \rightarrow (y + \Delta y)_q = y + \Delta y_q = y + \Delta y + \Delta b \tag{2.39}$$

Truncation is simpler than the rounding process but produces more error. If Δ_{bus} is the bus quantization stepsize, power of error signal for a truncated bus is given in [50] as follows:

$$E\left[\Delta b^2\right] = \frac{\Delta_{bus}^2}{3} \tag{2.40}$$

(2.6) can be reused for rounding case:

$$E\left[\Delta b^2\right] = \frac{\Delta_{bus}^2}{12} \tag{2.41}$$

Bus quantization error, e_b, can be defined in a similar way as (2.5), and using (2.11), it can be found as follows:

$$e_b = \sqrt{\frac{E\left[\Delta b^2\right]}{E\left[y^2\right]}} = \sqrt{\frac{E\left[\Delta b^2\right]}{K\frac{c_{max}^2}{3}E\left[x^2\right]}} \tag{2.42}$$

Then, total error for a single level multirate system with bus quantization can be given

as follows:

$$e = \sqrt{e_c^2 + e_b^2} \tag{2.43}$$

In this equation, either e_c or $e_{c,\max}$ can be used depending on the desired design quality.

2.2.2. Effects of Bus Quantization at the Output of a J-Level Multirate System

For a J-level cascaded system as shown in FIGURE 2.1 with bus quantization, an analysis is done similar to that of the previous section. The error signal at the output of the j'th level before bus quantization can be written as:

$$y_j + \Delta y_j = \sum_{i=1}^{j} \left(y_{j-1} + \underbrace{\Delta y_{j-1} + \Delta b_{j-1}}_{(\Delta y_{j-1})_q} \right) (c_j + \Delta c_j) \tag{2.44}$$

The output signal at the j'th level is the same as (2.18). Therefore, the output signal power for a J-level cascaded system is the same as (2.21). However, the output error signal is modified because it contains previous stages' error terms for bus quantization, and this modified error term at the j'th level can be labeled as $(\Delta y_j)_q$. Then, the output error signal can be written as follows:

$$\Delta y_j \cong \sum_{i=1}^{j} y_{j-1} \Delta c_j + \sum_{i=1}^{j} \Delta y_{j-1} c_j + \sum_{i=1}^{j} \Delta b_{j-1} c_j \tag{2.45}$$

Using the above equation, the output error power is

$$\begin{aligned} E\left[\Delta y_j^2\right] &= K_j \sigma_{\Delta c_j}^2 E\left[y_{j-1}^2\right] + K_j \sigma_{c_j}^2 E\left[\Delta y_{j-1}^2\right] + K_j \sigma_{c_j}^2 E\left[\Delta b_{j-1}^2\right] \\ &= K_j \sigma_{\Delta c_j}^2 K_{j-1} \sigma_{c_{j-1}}^2 E\left[y_{j-2}^2\right] + K_j \sigma_{c_j}^2 K_{j-1} \sigma_{\Delta c_{j-1}}^2 E\left[y_{j-2}^2\right] \\ &+ K_j \sigma_{c_j}^2 K_{j-1} \sigma_{c_{j-1}}^2 E\left[\Delta y_{j-2}^2\right] + K_j \sigma_{c_j}^2 K_{j-1} \sigma_{c_{j-1}}^2 E\left[\Delta b_{j-2}^2\right] \\ &+ K_j \sigma_{c_j}^2 E\left[\Delta b_{j-1}^2\right] \end{aligned} \tag{2.46}$$

If this process is iterated to the input, the output error power for an J-level cascaded multirate system will be given as follows

$$E\left[(\Delta y_J)_q^2\right] = E\left[\Delta y_J^2\right] + \left(\prod_{j=2}^{J} K_j \sigma_{c_j}^2\right) E\left[\Delta b_1^2\right] + \left(\prod_{j=3}^{J} K_j \sigma_{c_j}^2\right) E\left[\Delta b_2^2\right] + \ldots + K_J \sigma_{c_J}^2 E\left[\Delta b_{J-1}^2\right] + E\left[\Delta b_J^2\right] \tag{2.47}$$

Using the above equation in conjunction with (2.5) and (2.21), then total error for J-level cascaded system, e_J is

$$e_J = \sqrt{e_{c,J}^2 + \sum_{j=1}^{J} \frac{E\left[\Delta b_j^2\right]}{E\left[x^2\right]\left(\prod_{i=1}^{j} K_i \sigma_{c_i}^2\right)}} \tag{2.48}$$

If $J > 1$ with identical stages as in wavelet transform and fixed data bus at the output of each stage, i.e. $\Delta b_j = \Delta b$, $\forall j \leq J$, then a much simpler result is obtained:

$$e_J = \sqrt{e_{c,J}^2 + \frac{E\left[\Delta b_j^2\right]}{K\sigma_c^2 E\left[x^2\right]} \sum_{j=0}^{J-1} (K\sigma_c^2)^{-j}} = \sqrt{e_{c,J}^2 + \frac{E\left[\Delta b_j^2\right]}{E\left[x^2\right]} \frac{1-(K\sigma_c^2)^{-J}}{(K\sigma_c^2)-1}} \tag{2.49}$$

For an L-band system as shown in FIGURE 2.2, the system error can be defined as

$$e_L^2 \leq \max_{1 \leq i \leq L} \left(\beta_i^2 e_{J_i}^2\right) \tag{2.50}$$

where e_{J_i}'s are given as in (2.43) or (2.48).

The bus stepsize is given as follows:

$$\Delta_{bus} = 2^{-(b-1)} \tag{2.51}$$

This stepsize is unique due to the nature of the bus.

2.3. Model of a Multirate System

Using the above derivations, the output system error can be easily determined if the wordlength of each stage in a multirate system is given.

The above derivations can also be used to formulate a problem which can be stated as follows: "Given a tolerable error at the output, what will be the estimated chip area?". Now, a model will be set up for solving this question.

Let us define a variable, ϵ_i, as the square of a unit error term. For handling equations (2.29) and (2.50), there are two methods. The well-known method is using L constraints for each maximization operation as follows:

$$\begin{aligned} \epsilon_{c,L} - \beta_i^2 \epsilon_i \geq 0 \quad \forall i \leq L \\ \epsilon_L - \beta_i^2 \epsilon_{J_i} \geq 0 \quad \forall i \leq L \end{aligned} \tag{2.52}$$

The other method is derived by using the structure of the multirate system by forming paths from inputs to outputs as shown in FIGURE 2.7. At first, adders on each path are removed. Then, for each output, the given tolerable error is reflected to each path's output with predefined scales. As a result, these constraints and related variables can be removed. If the circuit is to be realized by using folding, then all modules that belong to a fold on a path are counted and a folding variable which is scaled by this count appears in the model. This process will be illustrated in an example in Section 2.4.

After the definition of variables, each path can be viewed as a cascaded multirate system. Then, it is evident that the number of constraints will be equal to the number of paths from inputs to the outputs. Let us define S as the set of indices of all multirate filters and bus quantization operations in the system, and Ψ_N as the set of all possible paths from inputs to outputs for N which is the node set given in the DFG file. Covering subsets S_ψ of set S for each ψ'th path can be formed such that $\psi \in \Psi_N$. In each S_ψ, only the variables that stand for error terms of multirate filters and bus quantization operations on the ψ'th path, ϵ_s, exist for $s \in S_\psi$ and $\psi \in \Psi_N$. If square of the given tolerable error for ψ'th path is η_ψ, then the constraints can be rewritten for all paths as:

$$\begin{aligned} CT1: &\ \textstyle\sum_{s\in S_\psi} \epsilon_s \le \eta_\psi \quad \forall \psi \in \Psi_N \\ CT2: &\ 0 < \epsilon_s \le \underline{\epsilon}_s \quad \forall s \in S \end{aligned} \tag{2.53}$$

These constraints form a bounded feasible region because there are upper and lower bounds on all variables. The convexity of the region is defined by type 1 constraints. However, it is an open convex set because ϵ_s can never reach its lower limit because of its definition. The upper bound $\underline{\epsilon}_s$ is the error due to the smallest number of bits for quantizing. This is two bits for coefficient quantization (a sign bit and a value bit) and one bit for bus quantization (only a sign bit). Therefore, the feasibility region is a bounded open convex set.

In the design process, some modules cannot be larger than some predefined area due to design restrictions. Therefore, those modules' coefficients cannot be realized using more bits than allowed. Similarly, due to restricted chip area and pin amount, bus-width must also be limited. For this case, note that the minimum error, $\bar{\epsilon}_s$, is obtained when the quantizing bits are equal to the maximum permissible bits. Thus this class of constraints can be rewritten as

$$CT3: \ \epsilon_s \ge \bar{\epsilon}_s \quad \exists s \in S. \tag{2.54}$$

The last constraint type comes from again design process restrictions. Some modules can be previously designed with fixed errors or some modules can be designed using prefixed operators. The bus-width can be fixed at some locations, for example, at the output of the chip. All these types of errors are called as *fixed errors* and are denoted by ϵ_{f_s}. Then, the final class of constraints is

$$CT4: \ \epsilon_s = \epsilon_{f_s} \quad \exists s \in S. \tag{2.55}$$

Consequently, any ϵ_s satisfying $CT1$, $CT2$, $CT3$, and $CT4$ gives the solution for the above question. However, it does not guarantee the minimal chip area.

We have to define an objective function to obtain minimum chip area using the above constraints given the tolerable error at the output. Note that ϵ_s is exponentially proportional to number of quantization bits as it is seen from stepsize definitions (2.33), (2.36) and (2.51). Let us partition the set S into two convex sets: C, the set of

indices for coefficient wordlengths in multirate filters, and B, the set of indices for bus quantization operations. Both the coefficients and the data bus have to be minimized to obtain minimum chip area. Also scaling of both coefficient quantization and bus quantization with some nonnegative constants, γ_c and γ_b respectively, can be done for emphasizing their relative importance. Then, the optimization model can be rewritten as

$$\begin{aligned} &\min F_S: \quad \min\left(-\textstyle\sum_{c\in C}\gamma_c \log_2 \epsilon_c - \sum_{b\in B}\gamma_b \log_2 \epsilon_b\right) \quad C \cup B = S, \\ &\text{subject to} \\ &f_1: \qquad \textstyle\sum_{s\in S_\psi} \epsilon_s \le \eta_\psi \qquad \forall \psi \in \Psi_N, \\ &f_2: \qquad \epsilon_s \le \underline{\epsilon}_s \qquad \forall s \in S, \\ &f_3: \qquad \epsilon_s \ge \overline{\epsilon}_s \qquad \exists s \in S, \\ &f_4: \qquad \epsilon_s = \epsilon_{f_s} \qquad \exists s \in S. \end{aligned} \tag{2.56}$$

keeping in mind that ϵ_s is always positive!

2.3.1. Optimal Solution of the Model

The objective function, F_S, of (2.56) is a convex function since $-\log_2 \epsilon_c$ and $-\log_2 \epsilon_b$ are convex, and summation of scaled convex functions is convex. Since the constraints form a convex feasibility region, then $\min F_S$ is a convex programming problem. It is known that any local optimal solution of a convex program is the global optimal solution. Thus, finding a local optimum solution is sufficient for finding a global optimal solution for the model given in (2.56). So Kuhn-Tucker (KT) necessary and sufficient conditions can be to solve the problem. These general theorems can be found in any standard text such as [52], [53].

In this model's solution, at first the fixed errors are removed from the model by substituting them to their corresponding places in $CT1$ and $CT2$ before generating the KT conditions. In case of infeasibility, the designer must either increase the tolerable error or use modules and bus quantization schemes with smaller errors. If the problem

is feasible, the new path errors, $\overline{\eta}_\psi$, are evaluated by subtracting fixed errors from original path errors for all paths that contain them. After this operation, fourth type constraints are removed.

Next, the feasibility of $CT3$ constraints are checked. If they violate feasibility, it is up to the designer to make the problem feasible as explained above. In case of a feasible problem, the model given by (2.56) before generating KT conditions will be

$$\begin{array}{lll} \min F_S: \quad \min -\sum_{c\in C}\gamma_c \log_2 \epsilon_c - \sum_{b\in B}\gamma_b \log_2 \epsilon_b & C\cup B = S, & \\ \text{subject to} & & \\ f_1: & \sum_{s\in S_\psi}\epsilon_s \le \overline{\eta}_\psi & \forall \psi \in \Psi_N, \\ f_2: & \epsilon_s \le \underline{\epsilon}_s & \forall s \in S, \\ f_3: & \epsilon_s \ge \overline{\epsilon}_s & \exists s \in S, \end{array} \tag{2.57}$$

The optimality conditions of the problem will be given by the KT conditions. They can be listed for this problem if $\epsilon_s^* = (\epsilon_c^*, \epsilon_b^*)$ is a local minimizer as follows:

$$\begin{array}{lll} KT1: & \left[\frac{\partial F_S}{\partial \epsilon_c} + \sum_{\psi\in\Psi_N} u_\psi \frac{\partial f_1}{\partial \epsilon_c} + w_c \frac{\partial f_{2_c}}{\partial \epsilon_c} - v_c \frac{\partial f_{3_c}}{\partial \epsilon_c}\right]_{\epsilon_c^*} = 0 & \forall c \in C, \\ KT2: & \left[\frac{\partial F_S}{\partial \epsilon_b} + \sum_{\psi\in\Psi_N} u_\psi \frac{\partial f_1}{\partial \epsilon_b} + w_b \frac{\partial f_{2_b}}{\partial \epsilon_b} - v_b \frac{\partial f_{3_b}}{\partial \epsilon_b}\right]_{\epsilon_b^*} = 0 & \forall b \in B, \\ KT3: & u_\psi\left(\sum_{s\in S_\psi}\epsilon_s^* - \overline{\eta}_\psi\right) = 0 & \forall \psi \in \Psi_N, \\ KT4: & w_s(\epsilon_s^* - \underline{\epsilon}_s) = 0 & \exists s \in S, \\ KT5: & v_s(\epsilon_s^* - \overline{\epsilon}_s) = 0 & \exists s \in S, \\ KT6: & u_\psi \ge 0 & \forall \psi \in \Psi_N, \\ KT7: & w_s \ge 0, v_s \ge 0 & \exists s \in S. \end{array} \tag{2.58}$$

Note that the KT conditions can be written as the multiplication of a 0-1 matrix multiplied by a vector of Lagrange multipliers. Using (2.57) and (2.58) the problem can be solved uniquely for optimality. Because of the nice structure of the problem, it converges to the optimum value quite fast.

This model can also handle folding operations quite efficiently. Folding effects only $CT1$ constraints, hence $KT1$ and $KT2$ conditions. This is shown in the following section.

2.4. Examples and Experiments

2.4.1. Example 1.

Let us form the model for the two-band three-level wavelet analysis and synthesis chips with no noise in the channel as shown in FIGURE 2.7. Assume that each path has different errors, and there is bus quantization only before the channel. Also assume that $\gamma_c = \gamma_b = 1$ for all quantization terms. The variable ϵ_{G1a} is the name of the variable of the G filter at the first level at the analysis part. All other variable names are written in a similar manner. Then the optimization problem can be written as

$$\begin{aligned}
&\min -(\log_2 \epsilon_{G1a} + \log_2 \epsilon_{G2a} + \log_2 \epsilon_{G3a} + \log_2 \epsilon_{G1s} + \log_2 \epsilon_{G2s} + \log_2 \epsilon_{G3s}) \\
&\quad -(\log_2 \epsilon_{H1a} + \log_2 \epsilon_{H2a} + \log_2 \epsilon_{H3a} + \log_2 \epsilon_{H1s} + \log_2 \epsilon_{H2s} + \log_2 \epsilon_{H3s}) \\
&\quad -(\log_2 \epsilon_{bus_{G1a}} + \log_2 \epsilon_{bus_{G2a}} + \log_2 \epsilon_{bus_{G3a}} + \log_2 \epsilon_{bus_{H3a}}) \\
s.t. & \\
u_1 \quad & \epsilon_{G1a} + \epsilon_{G1s} + \epsilon_{bus_{G1a}} \leq \eta_1 \\
u_2 \quad & \epsilon_{H1a} + \epsilon_{G2a} + \epsilon_{H1s} + \epsilon_{G2s} + \epsilon_{bus_{G2a}} \leq \eta_2 \\
u_3 \quad & \epsilon_{H1a} + \epsilon_{H2a} + \epsilon_{G3a} + \epsilon_{H1s} + \epsilon_{H2s} + \epsilon_{G3s} + \epsilon_{bus_{G3a}} \leq \eta_3 \\
u_4 \quad & \epsilon_{H1a} + \epsilon_{H2a} + \epsilon_{H3a} + \epsilon_{H1s} + \epsilon_{H2s} + \epsilon_{H3s} + \epsilon_{bus_{H3a}} \leq \eta_4
\end{aligned} \tag{2.59}$$

and all variables satisfy $CT2$ constraints. KT conditions can be written as

$$\begin{aligned}
KT1, KT2: \quad & \frac{1}{\epsilon_{G1a}} = \frac{1}{\epsilon_{G1s}} = \frac{1}{\epsilon_{bus_{G1a}}} = u_1 \\
& \frac{1}{\epsilon_{G2a}} = \frac{1}{\epsilon_{G2s}} = \frac{1}{\epsilon_{bus_{G2a}}} = u_2 \\
& \frac{1}{\epsilon_{G3a}} = \frac{1}{\epsilon_{G3s}} = \frac{1}{\epsilon_{bus_{G3a}}} = u_3 \\
& \frac{1}{\epsilon_{H1a}} = \frac{1}{\epsilon_{H1s}} = u_2 + u_3 + u_4 \\
& \frac{1}{\epsilon_{H2a}} = \frac{1}{\epsilon_{H2s}} = u_3 + u_4 \\
& \frac{1}{\epsilon_{H3a}} = \frac{1}{\epsilon_{H3s}} = \frac{1}{\epsilon_{bus_{H3a}}} = u_4
\end{aligned} \tag{2.60}$$

Note that $KT3$ conditions are clearly redundant. This means that the constraints given in the model must be tight; i.e., they must be equal to η_ψ for ψ'th path. Therefore, the problem can be solved uniquely using the above equations.

2.4.2. Example 2.

Assume now folding is used to realize the above system. Let us say that our folding sets are $F_1 = \{G1a, G2a, G3a\}$, $F_2 = \{H1a, H2a, H3a\}$, $F_3 = \{G1s, G2s, G3s\}$, $F_4 = \{H1s, H2s, H3s\}$. Then, the model is

$$\begin{aligned}
&\min -(\log_2 \epsilon_{F1} + \log_2 \epsilon_{F2} + \log_2 \epsilon_{F3} + \log_2 \epsilon_{F4}) \\
&\quad -(\log_2 \epsilon_{bus_1} + \log_2 \epsilon_{bus_2} + \log_2 \epsilon_{bus_3} + \log_2 \epsilon_{bus_4}) \\
&s.t. \\
u_1 &\qquad \epsilon_{F1} + \epsilon_{F3} + \epsilon_{bus_1} \leq \eta_1 \\
u_2 &\qquad \epsilon_{F1} + \epsilon_{F2} + \epsilon_{F3} + \epsilon_{F4} + \epsilon_{bus_2} \leq \eta_2 \\
u_3 &\qquad \epsilon_{F1} + 2\epsilon_{F2} + \epsilon_{F3} + 2\epsilon_{F4} + \epsilon_{bus_3} \leq \eta_3 \\
u_4 &\qquad 3\epsilon_{F2} + 3\epsilon_{H4} + \epsilon_{bus_4} \leq \eta_4
\end{aligned} \tag{2.61}$$

and all variables satisfy $CT2$ constraints. KT conditions can be written as

$$\begin{aligned}
&\frac{1}{\epsilon_{F1}} = \frac{1}{\epsilon_{F3}} = u_1 + u_2 + u_3 \\
&\frac{1}{\epsilon_{F2}} = \frac{1}{\epsilon_{F4}} = u_2 + 2u_3 + 3u_4 \\
KT1, KT2: \quad &\frac{1}{\epsilon_{bus_1}} = u_1 \\
&\frac{1}{\epsilon_{bus_2}} = u_2 \\
&\frac{1}{\epsilon_{bus_3}} = u_3 \\
&\frac{1}{\epsilon_{bus_4}} = u_4
\end{aligned} \tag{2.62}$$

2.4.3. Example 3.

Now let us give a numerical example for the analysis part of the system defined in Example 2. As a result, only folds $F1$ and $F2$ exist. Both of these folds use

Daubechies' six tap coefficient set, but $F1$ uses the original one and $F2$ uses the high-pass one. Assume that there is no specified error for the output of path 1. The other errors are $e_2 = 4.47\%$, $e_3 = 7.07\%$ and $e_4 = 8.37\%$. Then $\eta_2 = (4.47\%)^2 = 0.02$, $\eta_3 = 0.05$ and $\eta_4 = 0.07$. Assume that $F2$ is previously designed with 7 bits and $F1$ has an upper bound of 6 bits on its coefficient quantization. Also, the bus quantization bits for all paths are variable except the ones before channel: The output of path 2 has a truncated fixed bus quantization of 5 bits and the output of path 4 has a rounded fixed bus quantization of 6 bits. The output of path 3 has a truncated variable bus with an upper bound of 6 bits and its scale is $\gamma_b = 0.01$. If the stepsize of the coefficient quantization is taken as (2.33), then the primal model is formed as follows:

$$\begin{array}{ll} & \min -\log_2 \epsilon_{F1} - 0.01 \log_2 \epsilon_{bus_3} \\ s.t. & \\ u_1 & \epsilon_{F1} \leq 0.00183 \\ u_2 & \epsilon_{F1} + \epsilon_{bus_3} \leq 0.00466 \\ w_1 & \epsilon_{F1} \geq 0.00067 \\ w_2 & \epsilon_{bus_3} \geq 1.93379 * 10^{-7} \end{array} \tag{2.63}$$

both variables satisfy $CT2$ constraints. KT conditions can be written as

$$\begin{array}{ll} KT1: & \frac{1}{\epsilon_{F1}} = u_1 + u_2 - w_1 \\ KT2: & \frac{0.01}{\epsilon_{bus_3}} = u_2 - w_2 \\ KT3: & u_1 (\epsilon_{F1} - 0.00183) = 0 \\ & u_2 (\epsilon_{F1} + \epsilon_{bus_3} - 0.00466) = 0 \\ KT4: & w_1 (0.00067 - \epsilon_{F1}) = 0 \\ & w_2 (1.93379 * 10^{-7} - \epsilon_{bus_3}) = 0 \\ KT5: & u_1 \geq 0 \quad u_2 \geq 0 \\ KT6: & w_1 \geq 0 \quad w_2 \geq 0 \end{array} \tag{2.64}$$

The solution of this system yields truncated output bus quantization of path 3 with only 1 bit, which is actually the sign bit. $F1$ is must be realized with 6 bits to satisfy both folding and path error requirements. Final path errors are found as $e_1 = 2.59\%$, $e_2 = 2.90\%$, $e_3 = 3.17\%$ and $e_4 = 2.24\%$.

Now if the same problem is solved using (2.36), then the model and KT conditions will be exactly the same except for constant values. The solution of this system yields truncated output bus quantization of path 3 with only 1 bit, which is actually the sign bit. $F1$ must be realized with 5 bits to satisfy both folding and path error requirements. Final path errors are found as $e_1 = 3.33\%$, $e_2 = 3.43\%$, $e_3 = 3.52\%$ and $e_4 = 1.37\%$.

As this example shows, fewer bits are sufficient to realize the same system when (2.36) is used instead of (2.33) for system stepsize.

2.4.4. Experiments

One dimensional linear, triangular, square, random inputs and 256x256 gray-scale Lenna image were utilized in our experiments. The two-dimensional image is converted into a one-dimensional signal by connecting each row at the end of the previous one. These signals are applied to G filter of FIGURE 2.3 with Daubechies' coefficients for six-tap multirate filter coefficients given in [54] to obtain FIGURE 2.9 and FIGURE 2.10. Then they are applied to the systems formed by connecting analysis and synthesis parts of FIGURE 2.3 and FIGURE 2.4 via noiseless channels. The coefficients used were Daubechies' and Çağlar's. The results are shown in figures from FIGURE 2.11 to FIGURE 2.14 respectively. These experiments were first carried out with no bus quantization. Then all the experiments are repeated using bus quantization. The bus quantization operation is carried out for all bits between 0 and 16. The corresponding figures are from FIGURE 2.15 to FIGURE 2.26. The graphs support the above results.

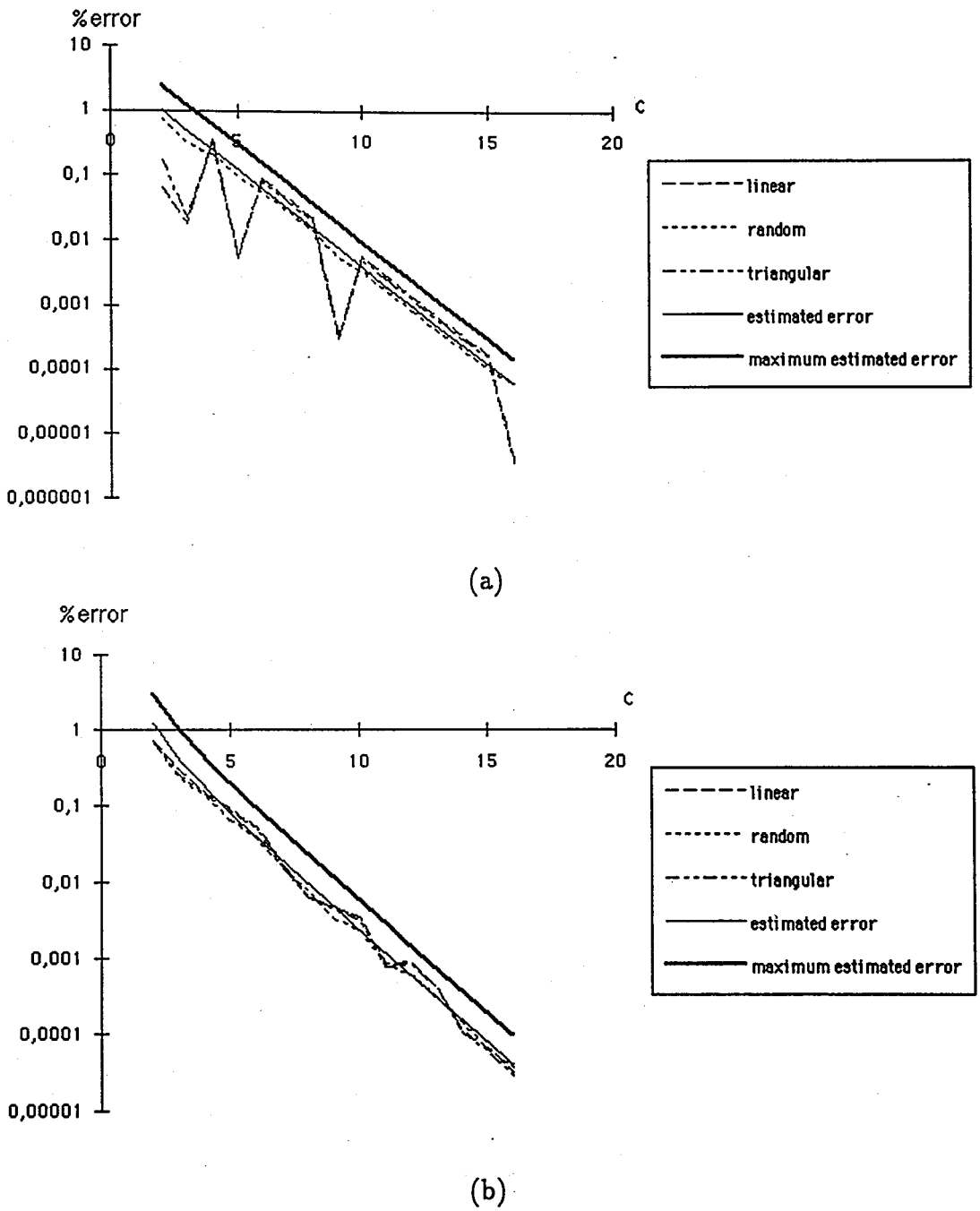


FIGURE 2.9. Experimental and estimated errors on a single level multirate filter. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)

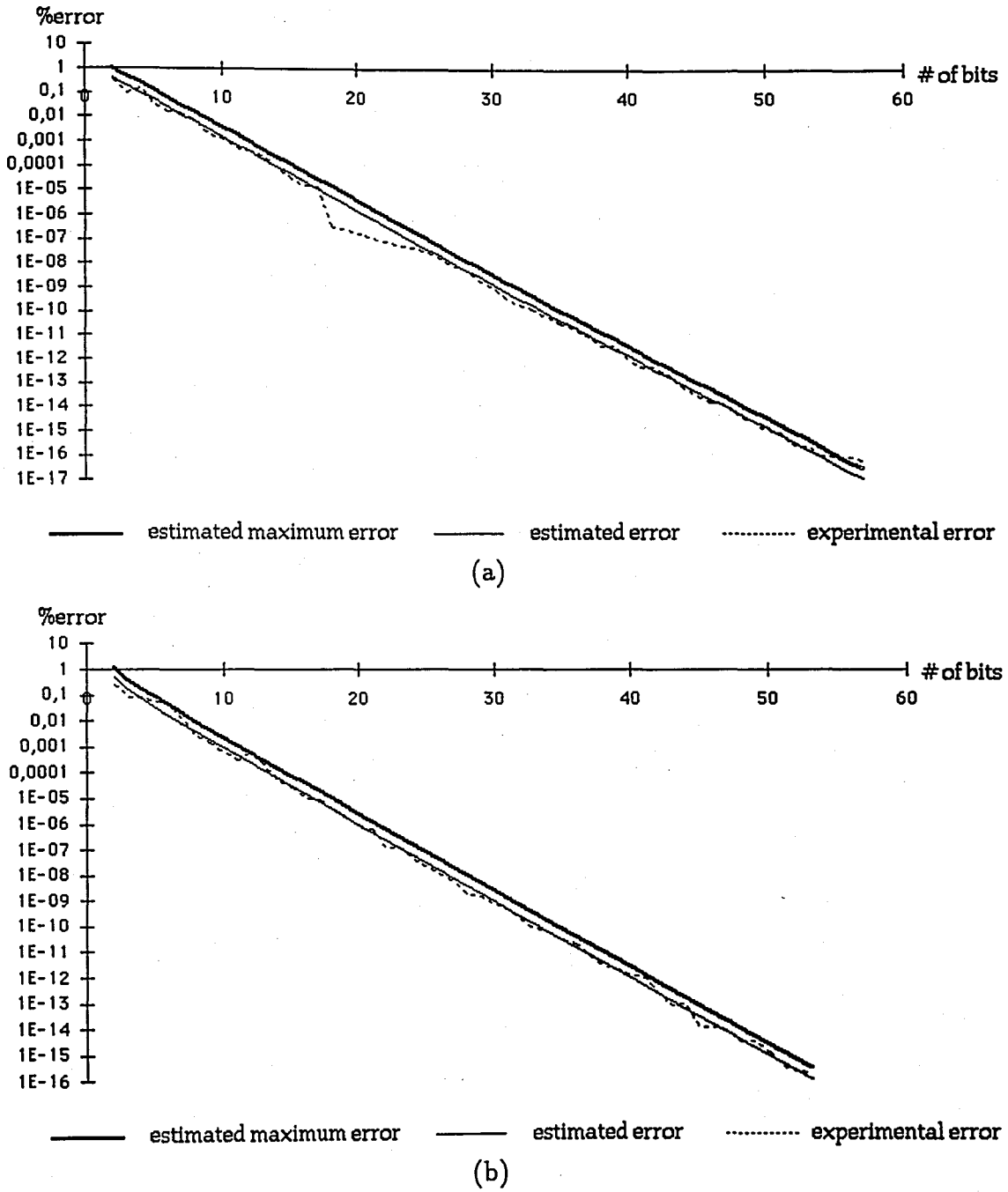
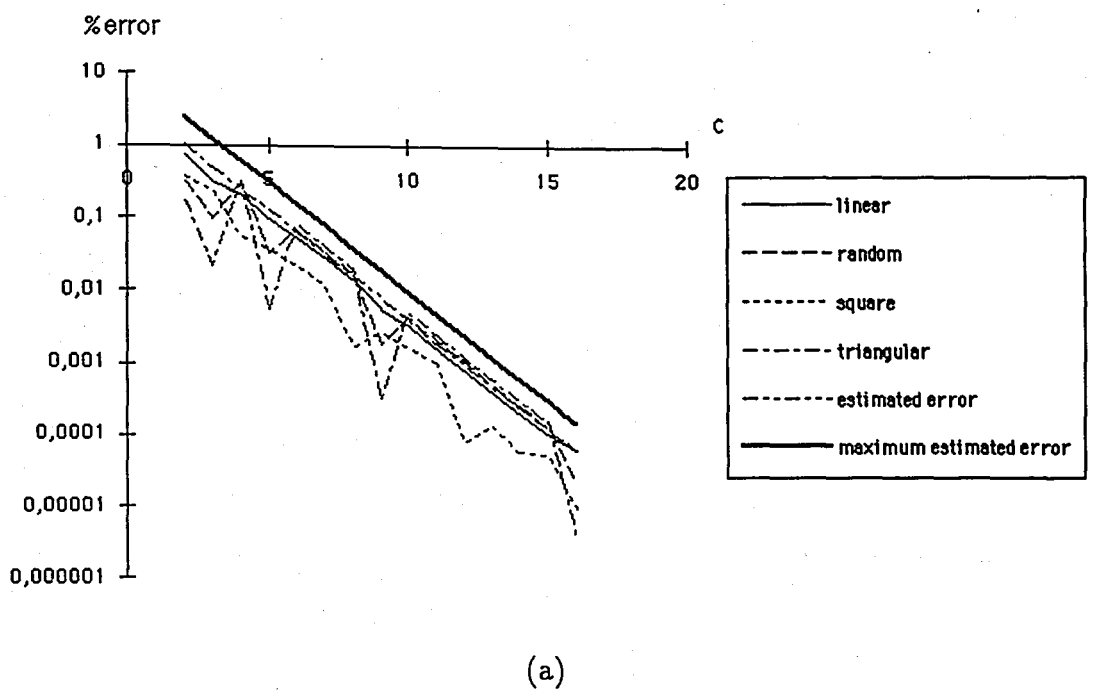
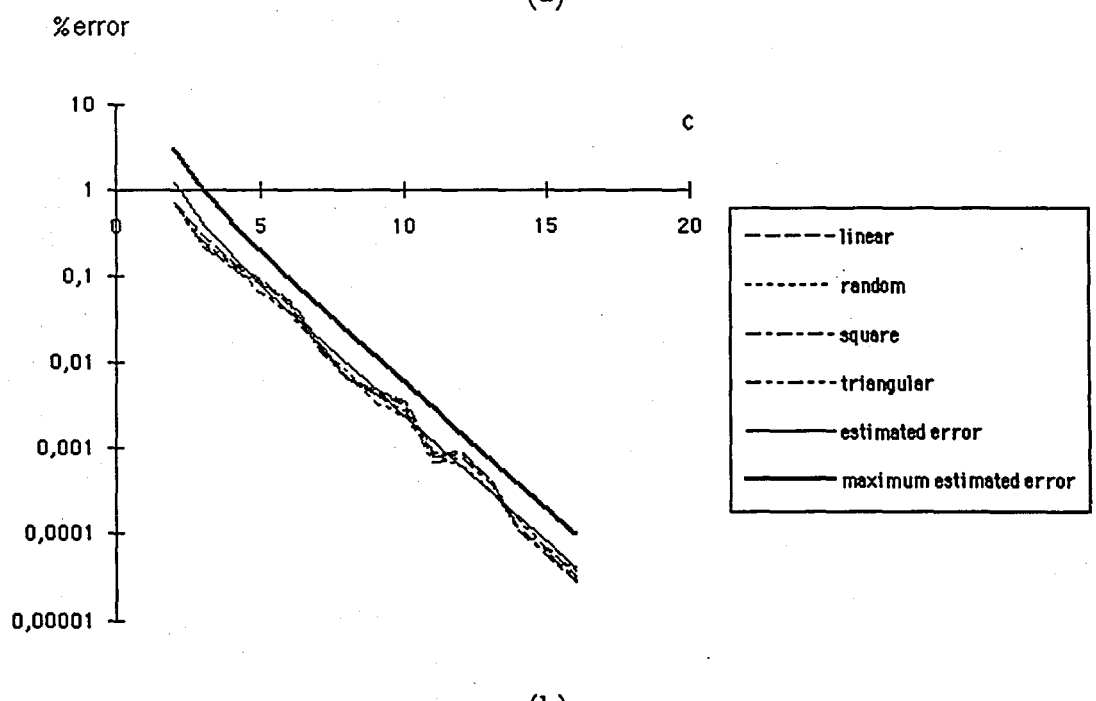


FIGURE 2.10. Experimental and estimated errors on a single level multirate filter for the Lenna image. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)



(a)



(b)

FIGURE 2.11. Experimental and estimated errors on a two-band-three-level multirate system. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)

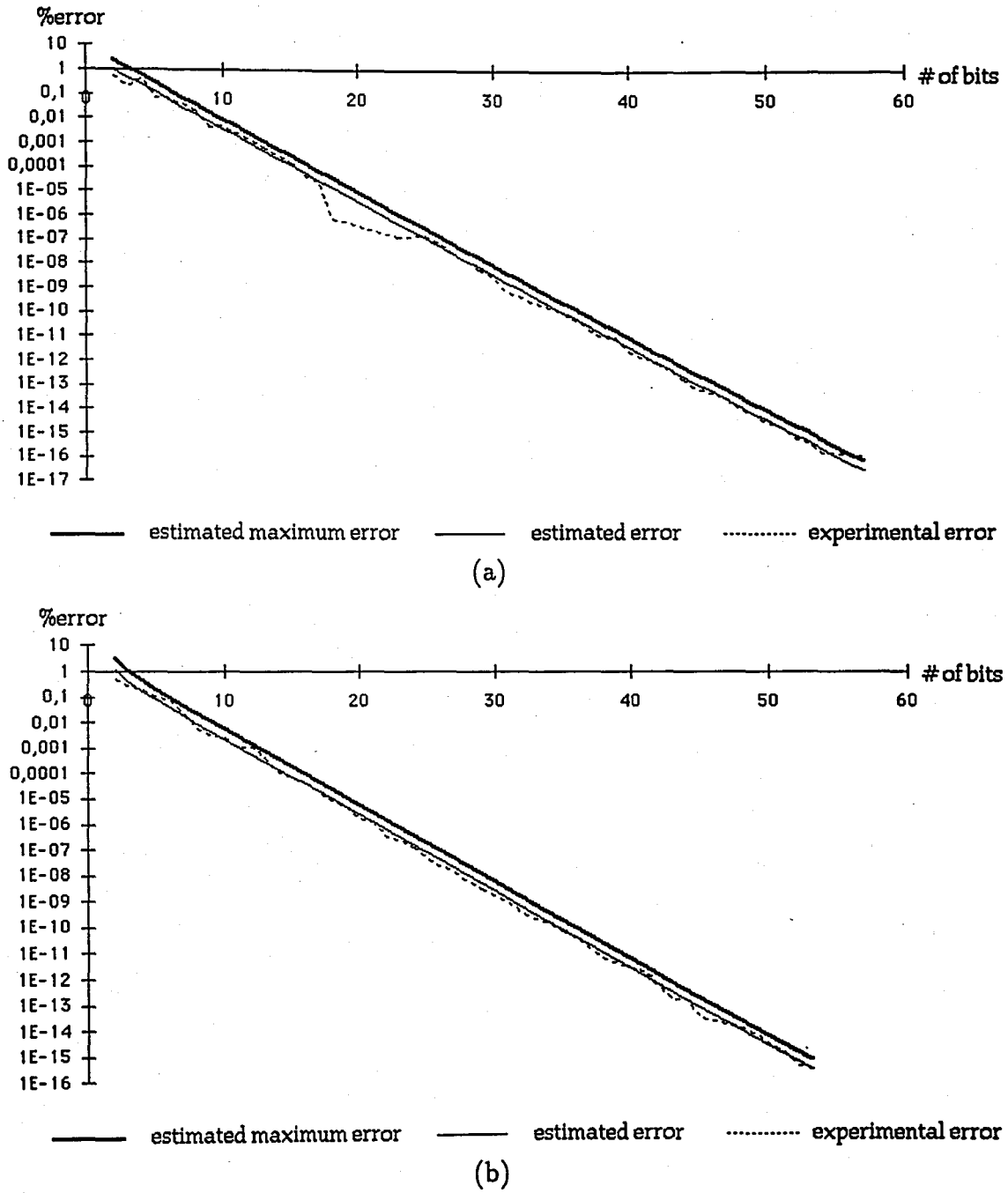


FIGURE 2.12. Experimental and estimated errors on a two-band-three-level multirate system for the Lenna image. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)

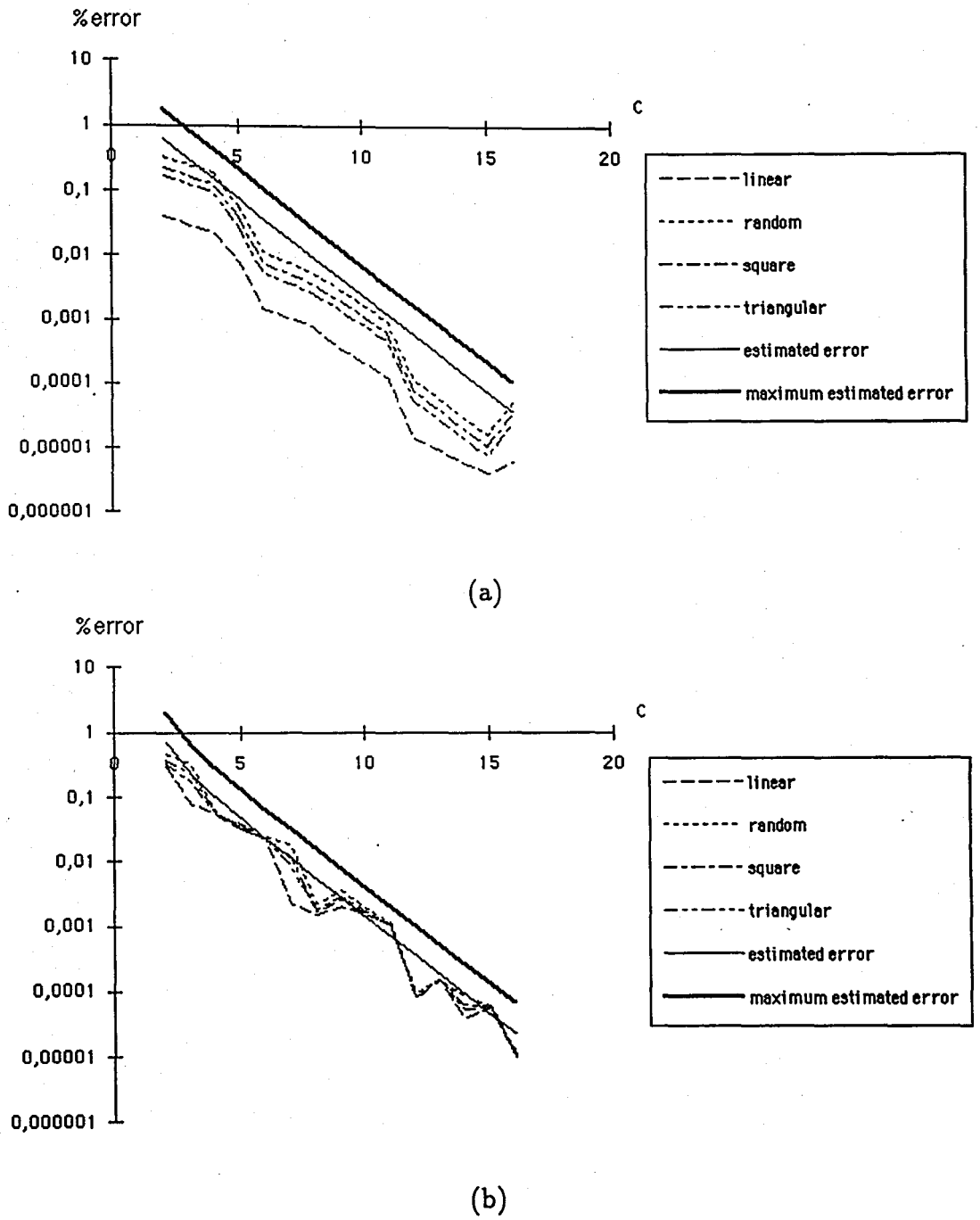


FIGURE 2.13. Experimental and estimated errors on a four-band multirate system. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)

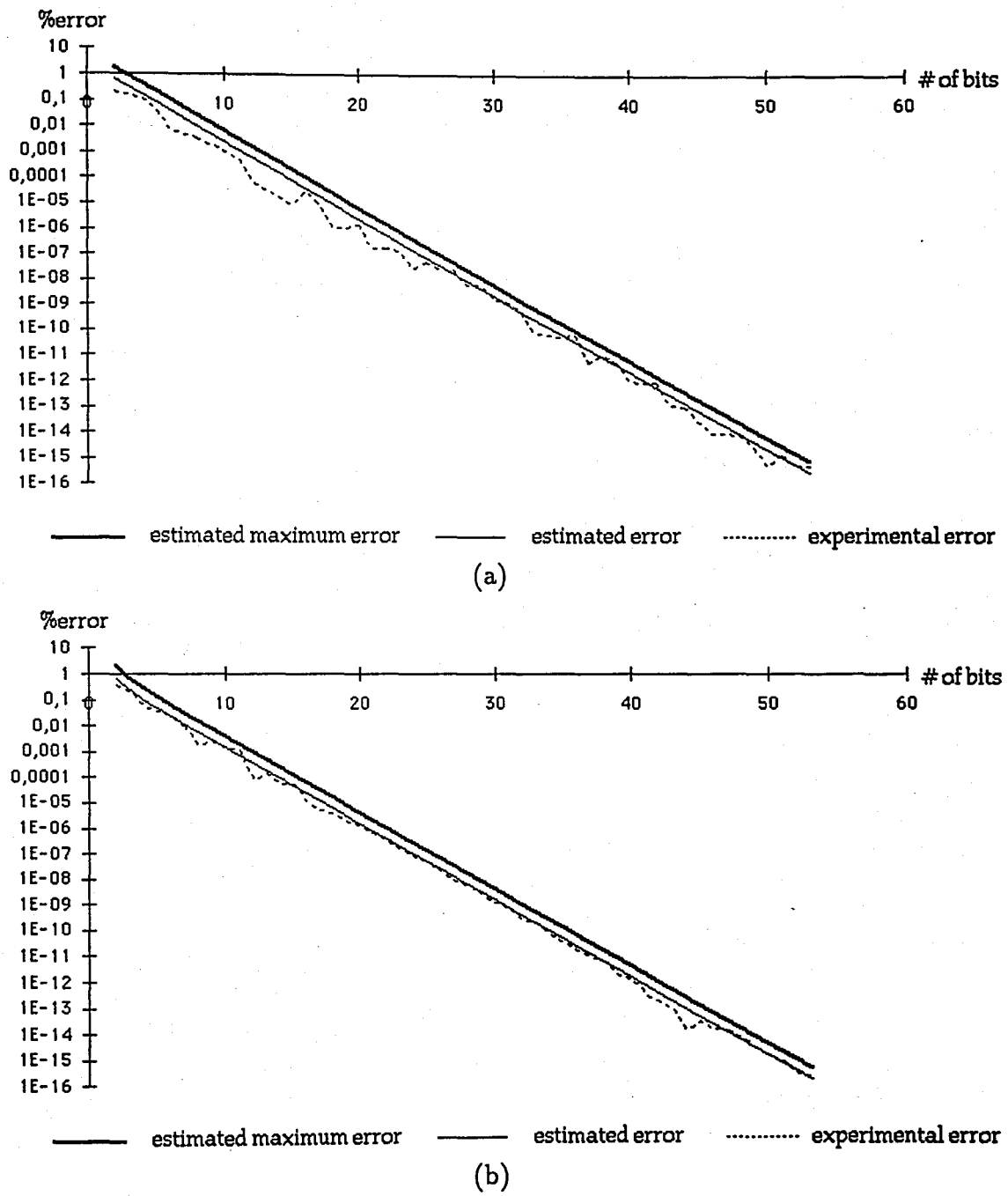


FIGURE 2.14. Experimental and estimated errors on a four-band multirate system for the Lenna image. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)

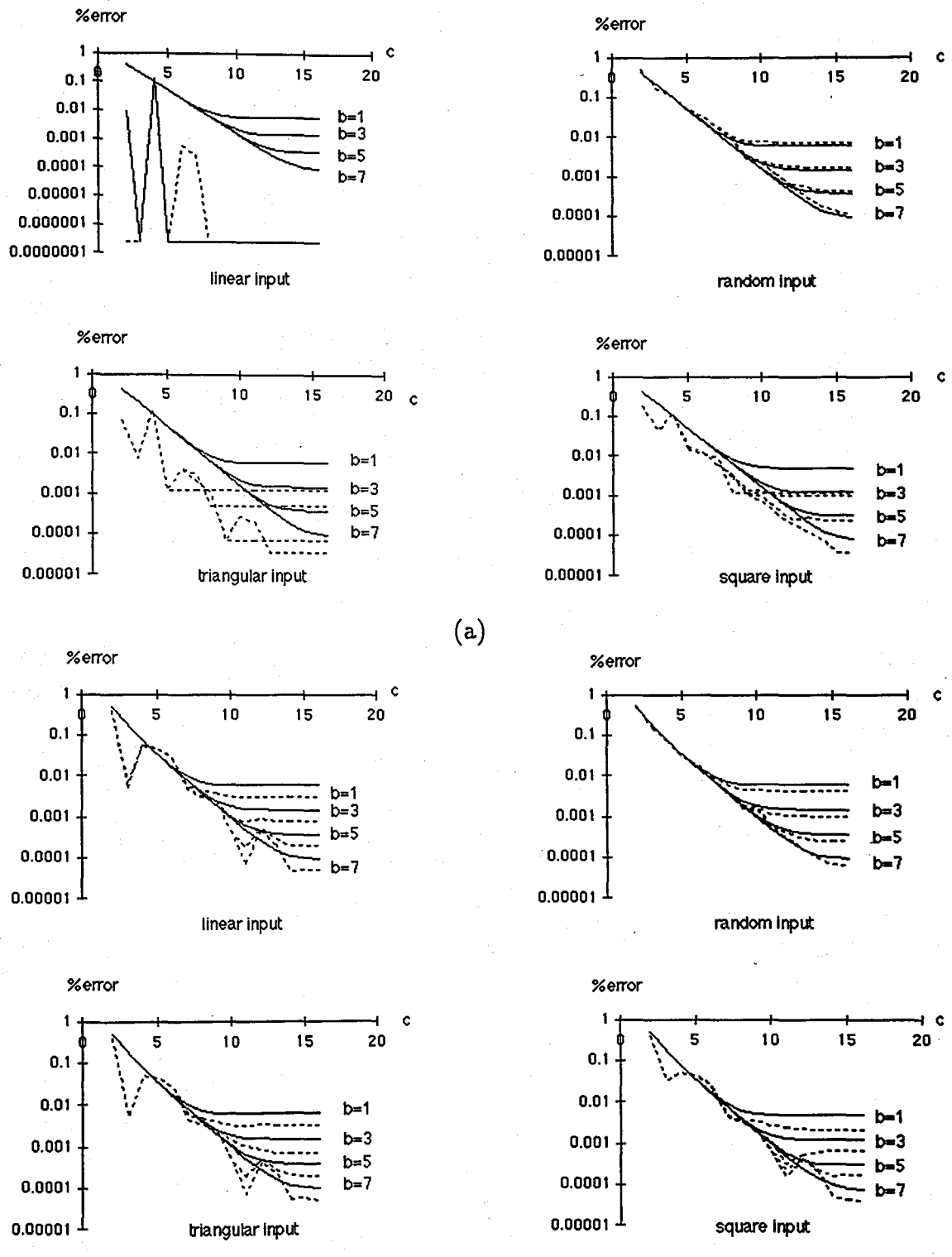
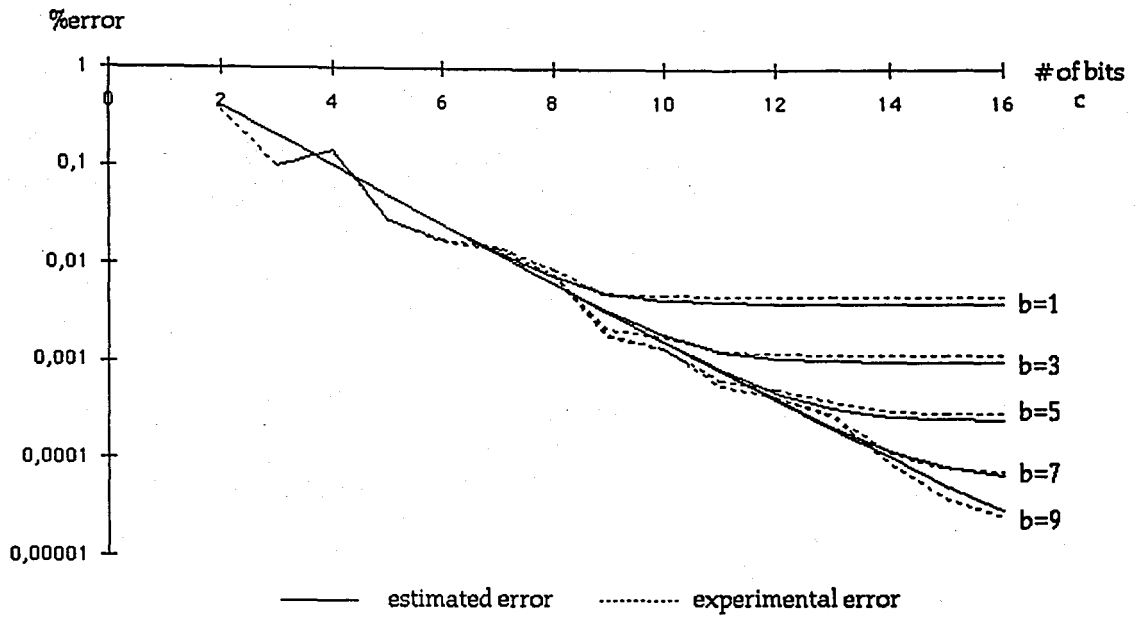
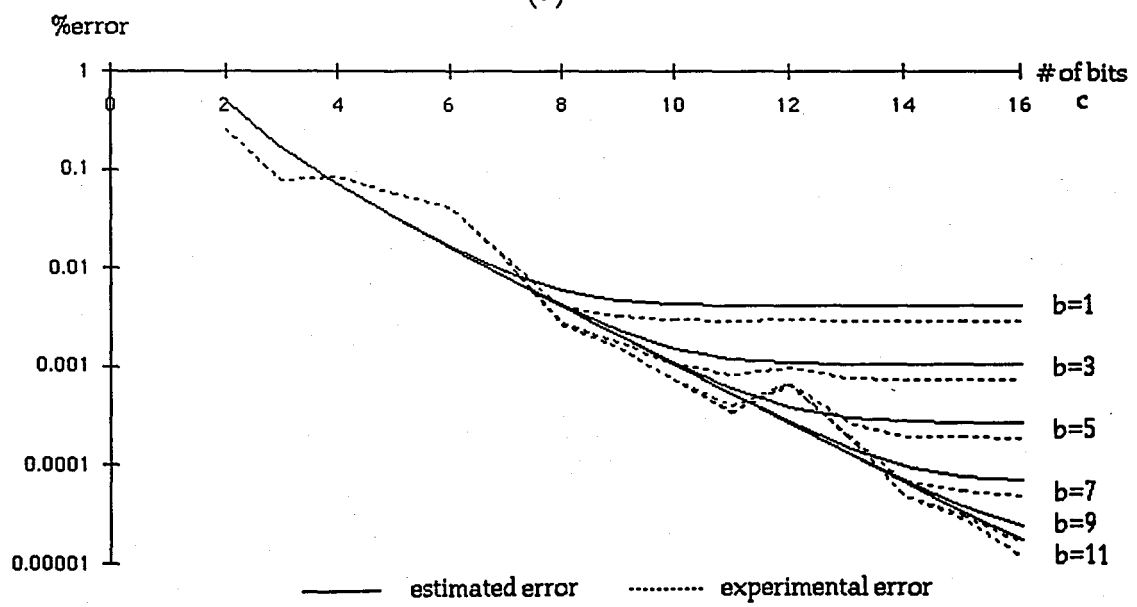


FIGURE 2.15. Experimental and estimated errors on a single level multirate filter with rounded fractional bus quantization at the output. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)



(a)



(b)

FIGURE 2.16. Experimental and estimated errors on a single level multirate filter with rounded fractional bus quantization at the output for the Lenna image. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)

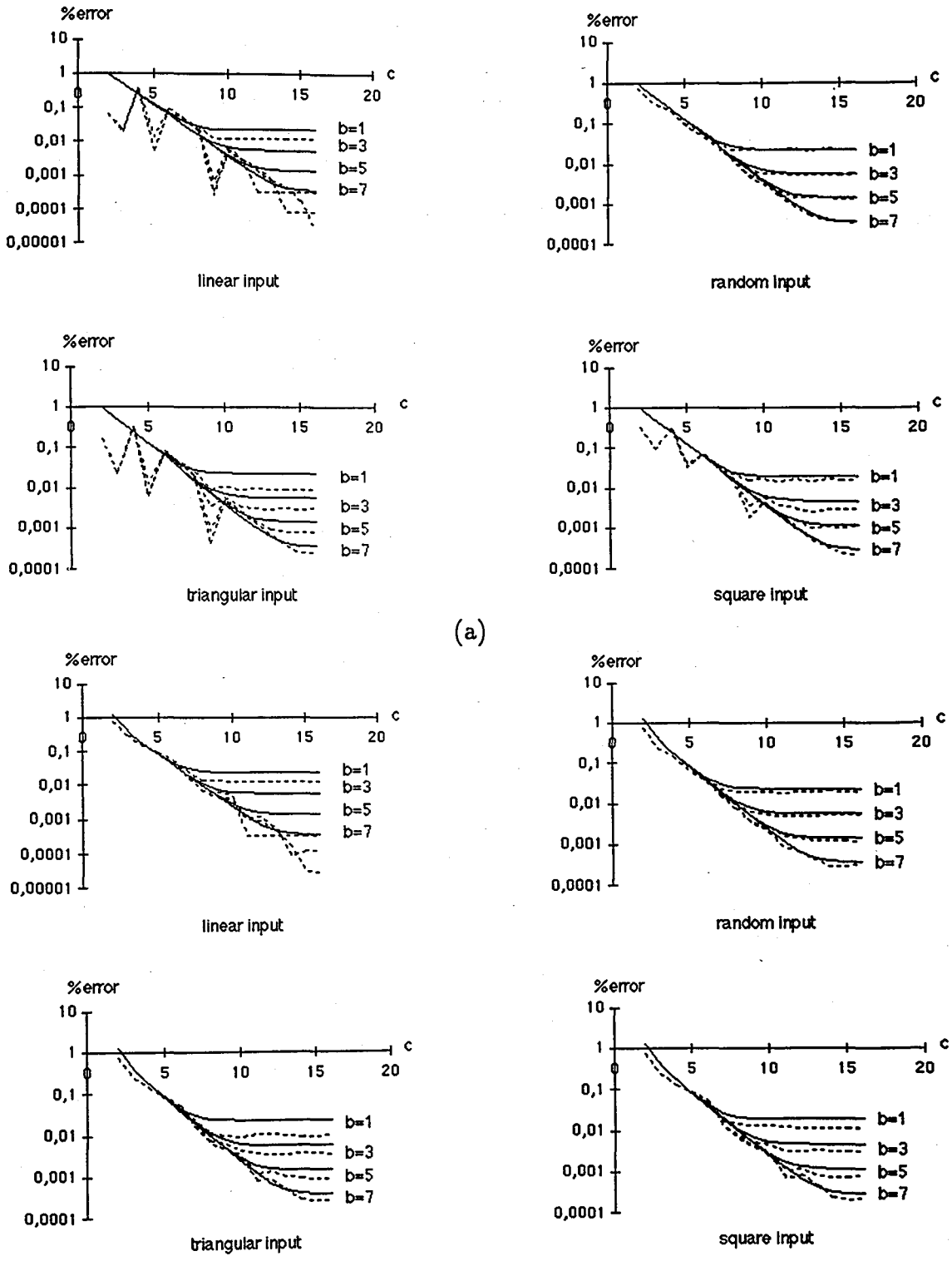
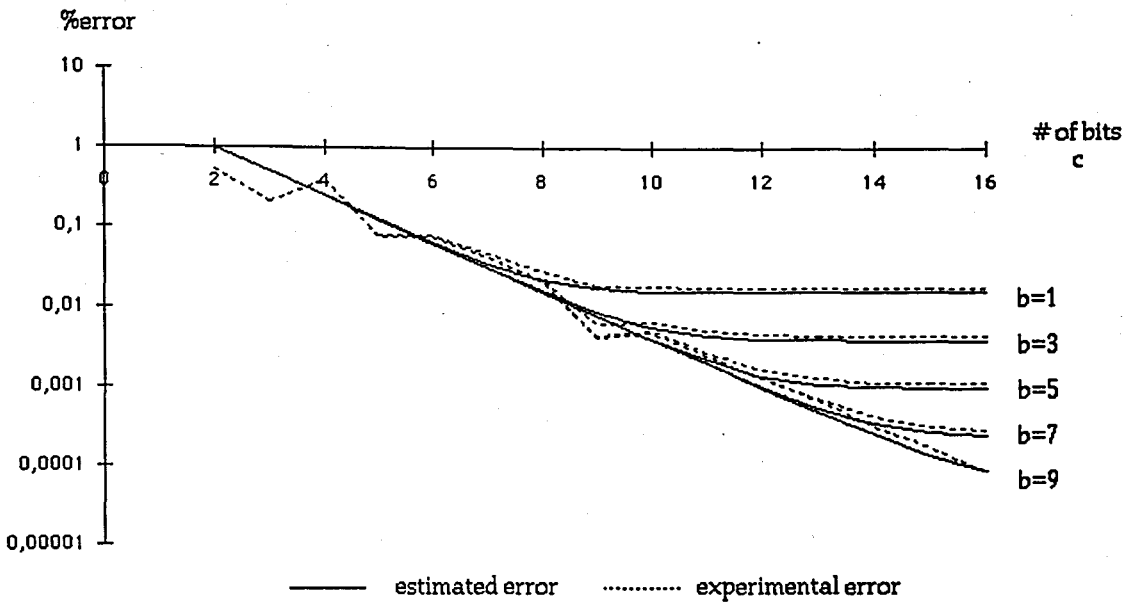
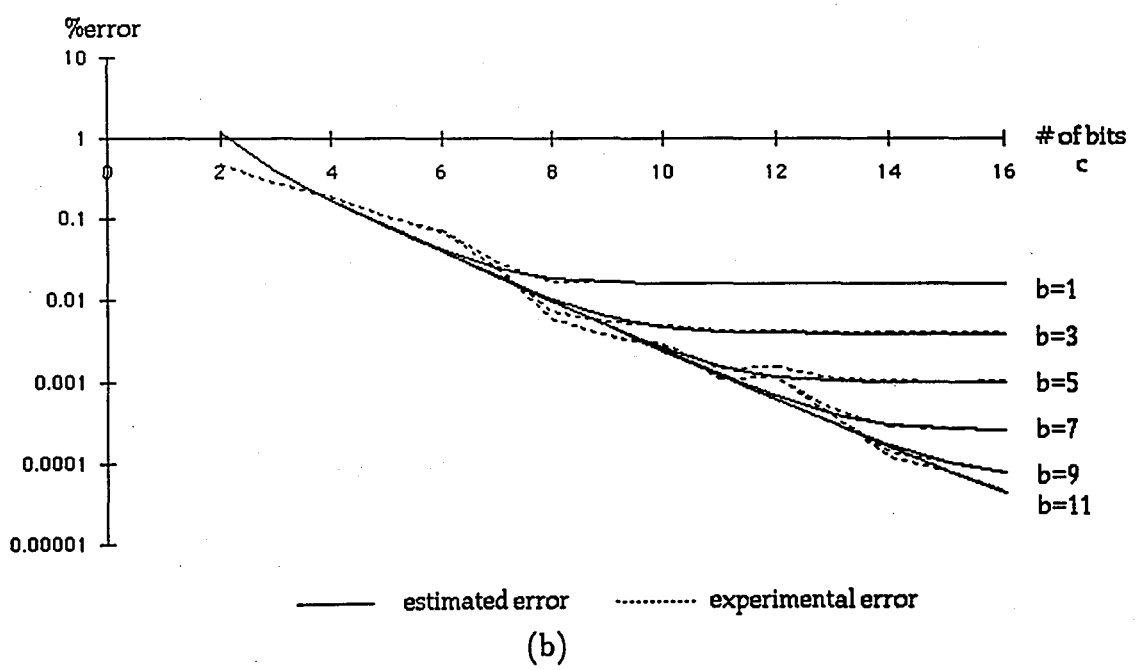


FIGURE 2.17. Experimental and estimated errors on a two-band-three-level multirate system with rounded fractional bus quantization at the output. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)



(a)



(b)

FIGURE 2.18. Experimental and estimated errors on a two-band-three-level multirate system with rounded fractional bus quantization at the output for the Lenna image. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)

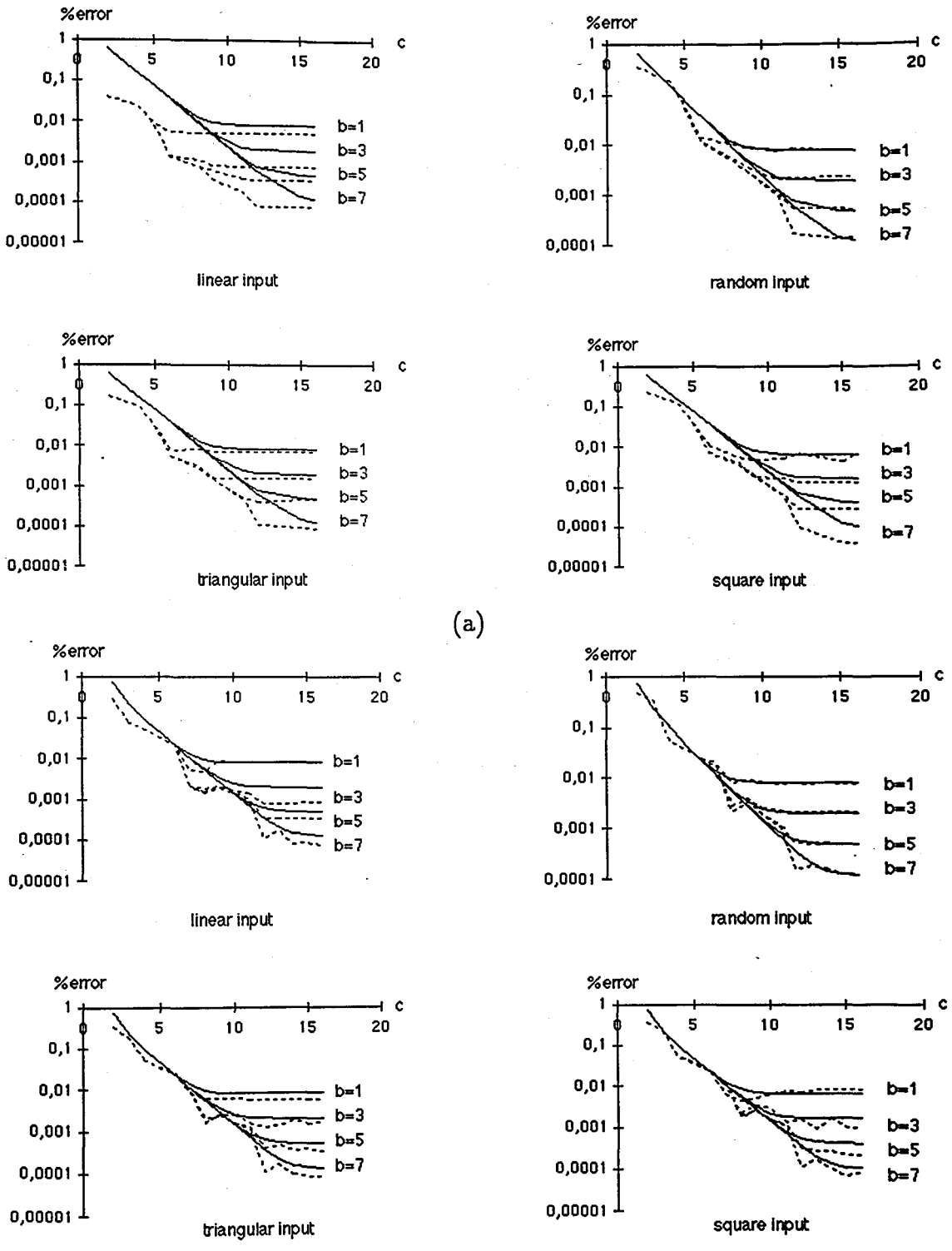
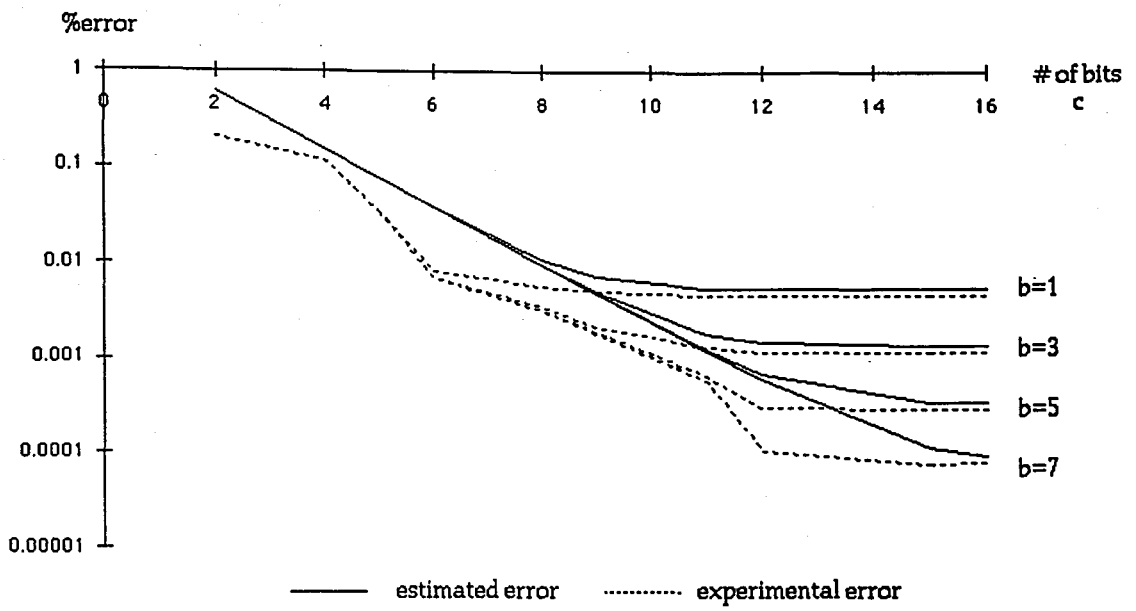
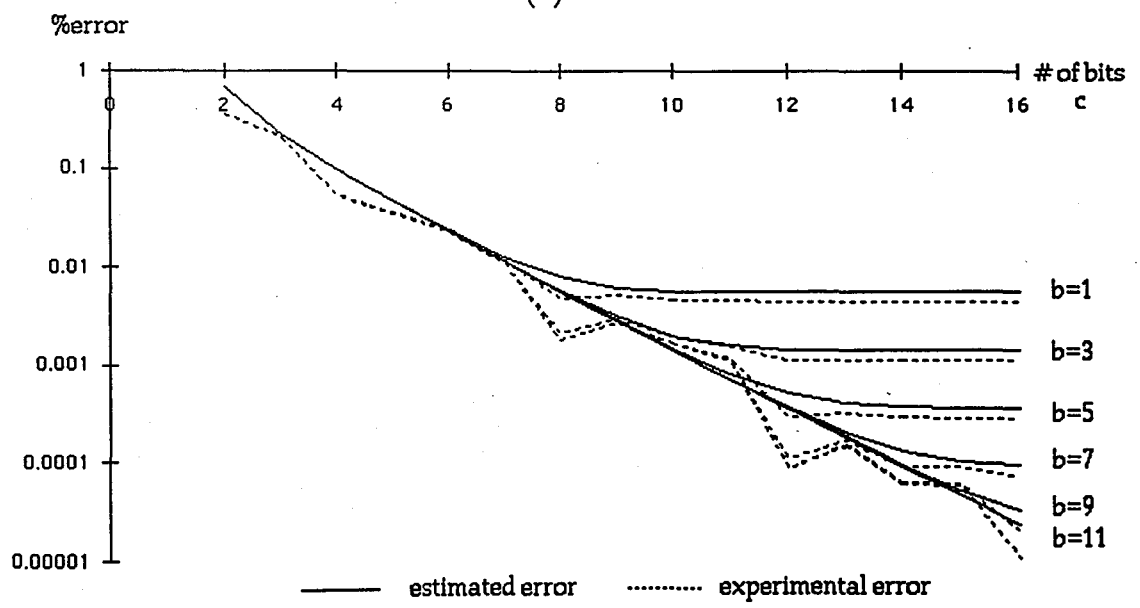


FIGURE 2.19. Experimental and estimated errors on a four-band multirate system with rounded fractional bus quantization at the output. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)

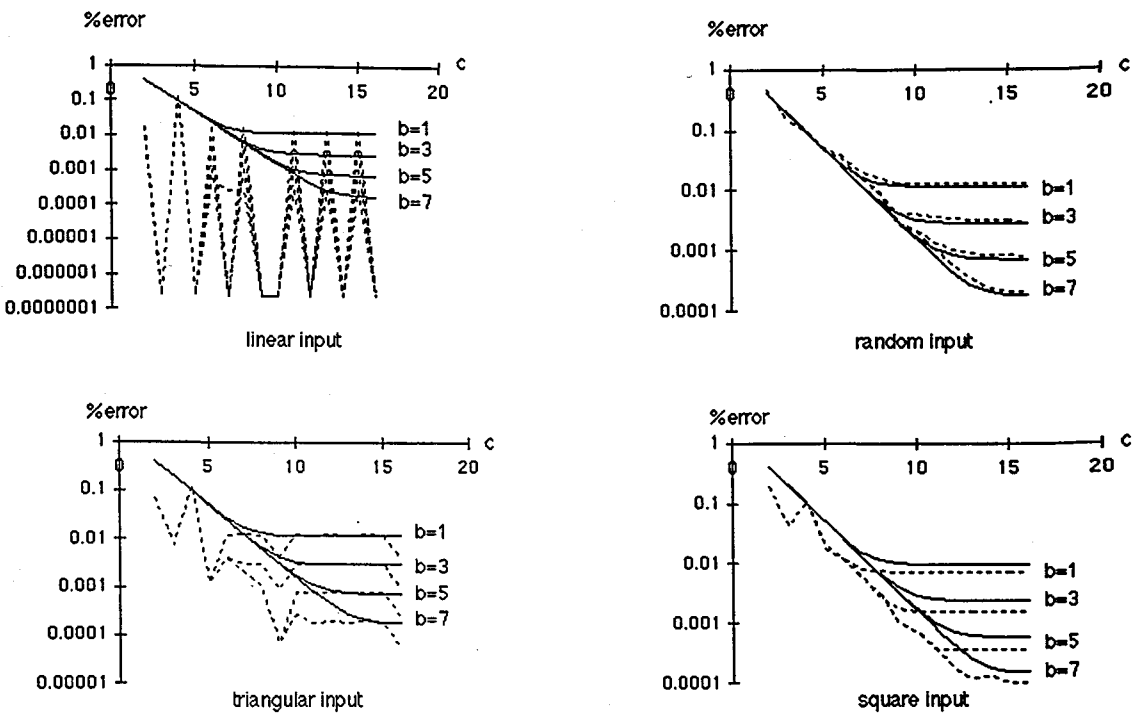


(a)

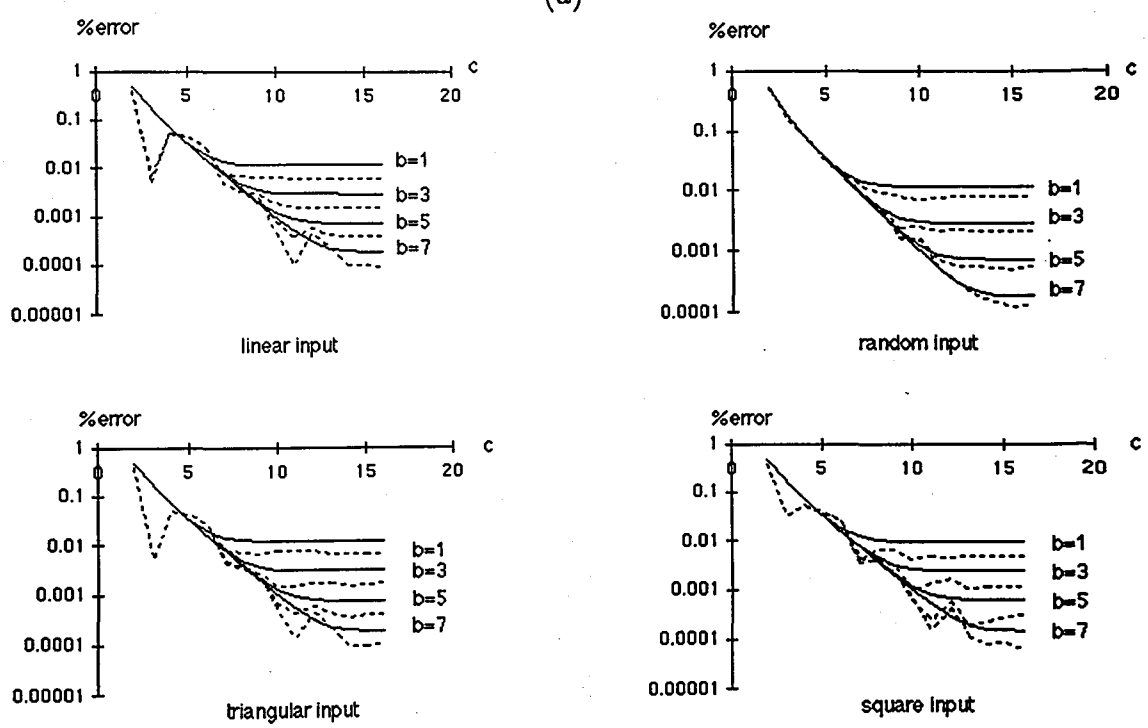


(b)

FIGURE 2.20. Experimental and estimated errors on a four-band multirate system with rounded fractional bus quantization at the output for the Lenna image. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)



(a)



(b)

FIGURE 2.21. Experimental and estimated errors on a single level multirate filter with truncated fractional bus quantization at the output. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)

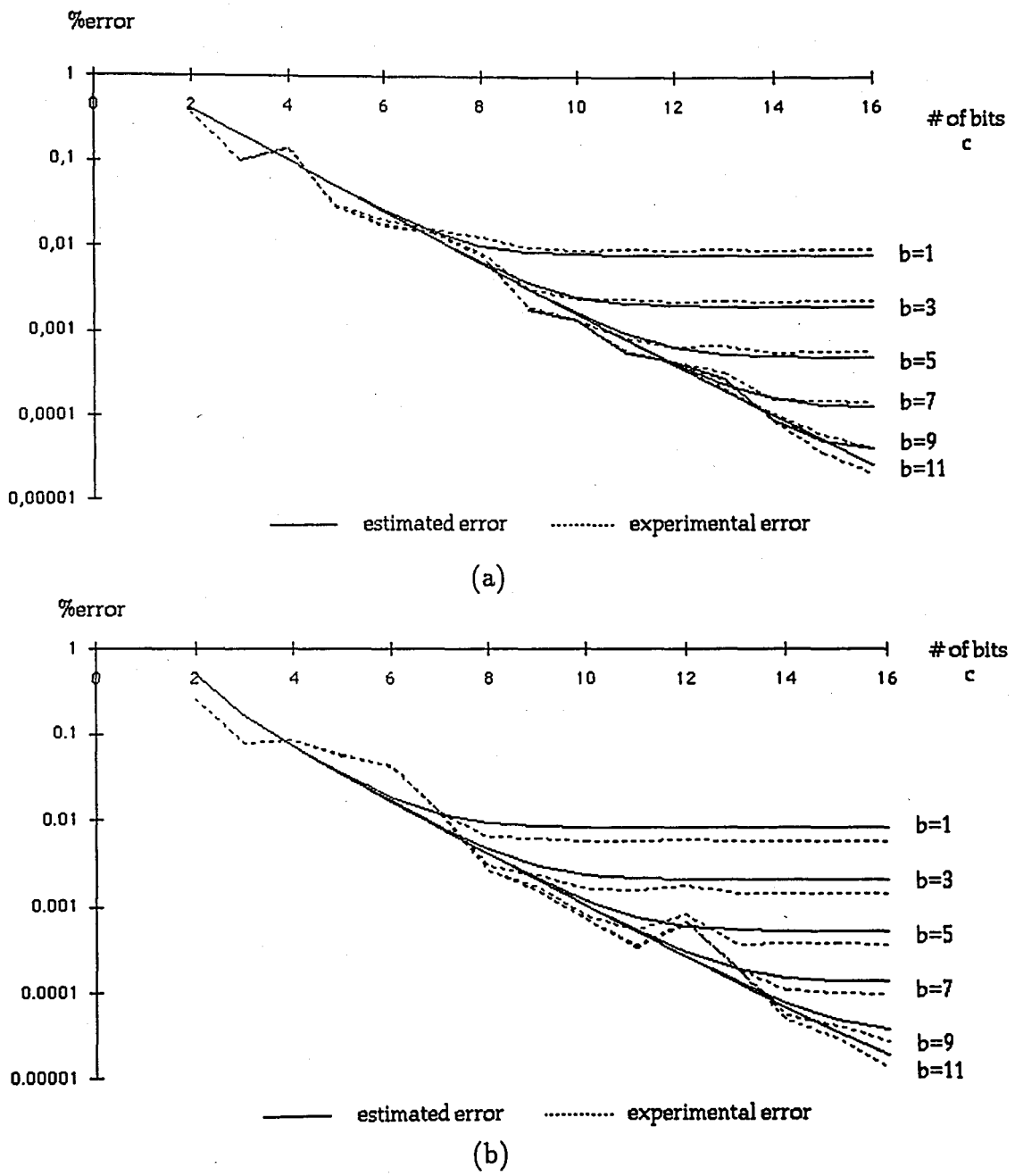
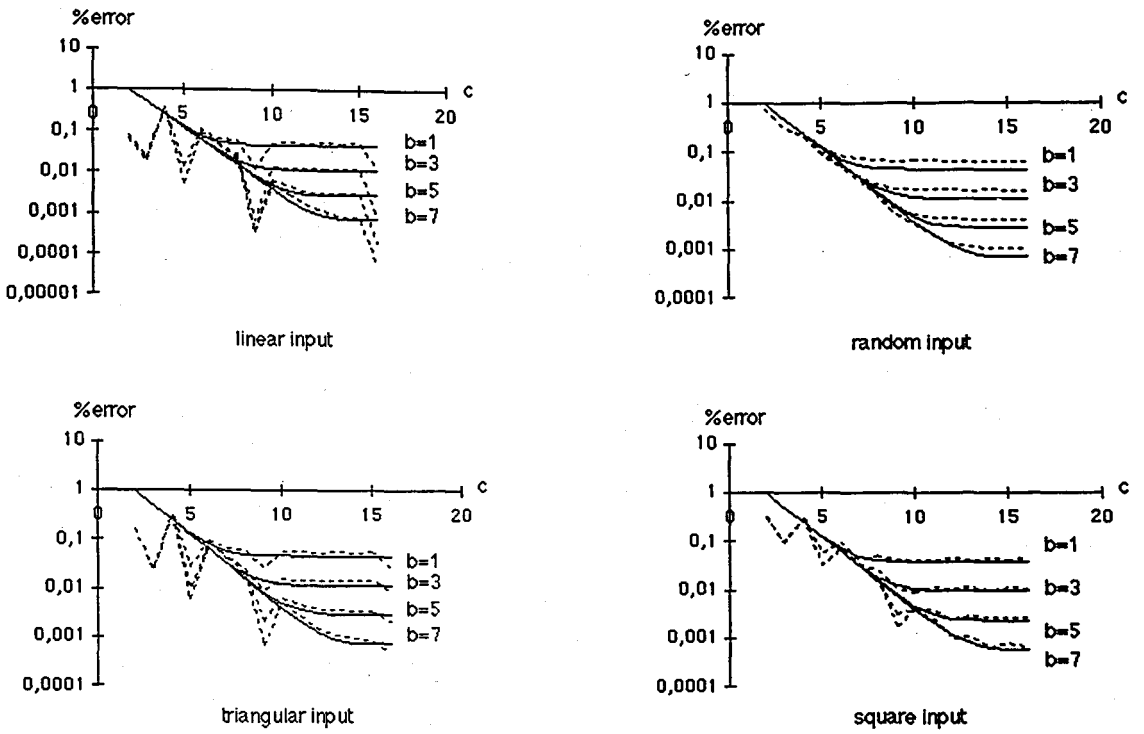
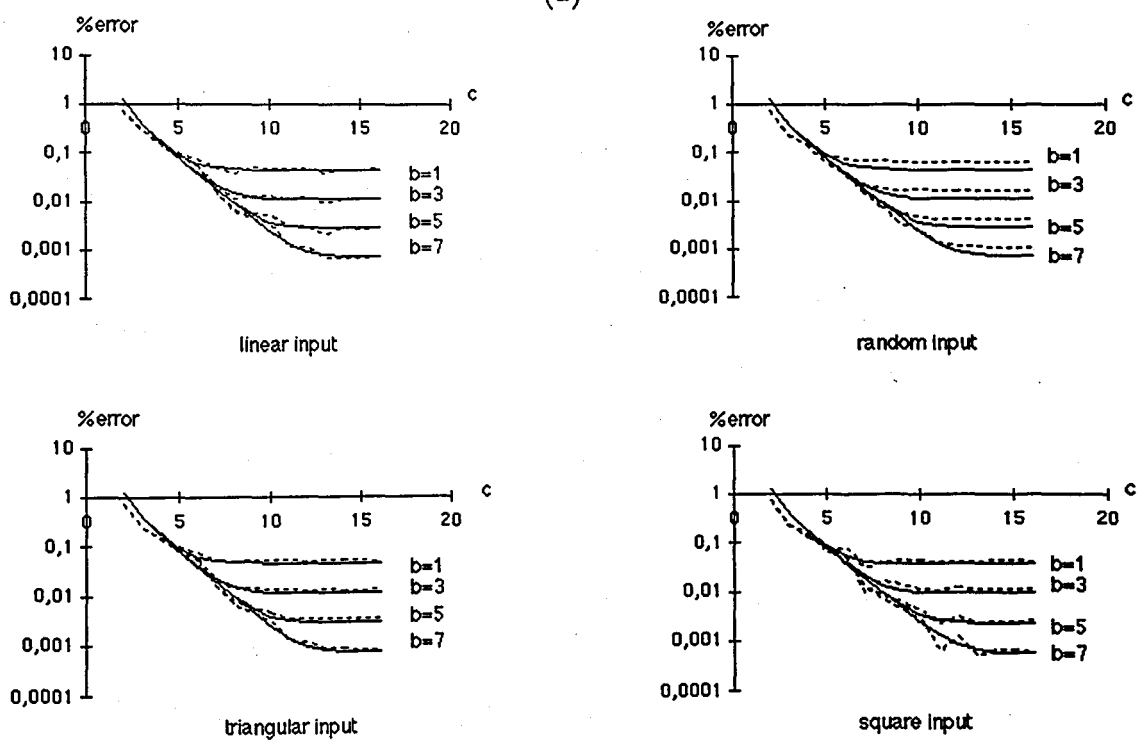


FIGURE 2.22. Experimental and estimated errors on a single level multirate filter with truncated fractional bus quantization at the output for the Lenna image. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)



(a)



(b)

FIGURE 2.23. Experimental and estimated errors on a two-band-three-level multirate system with truncated fractional bus quantization at the output. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)

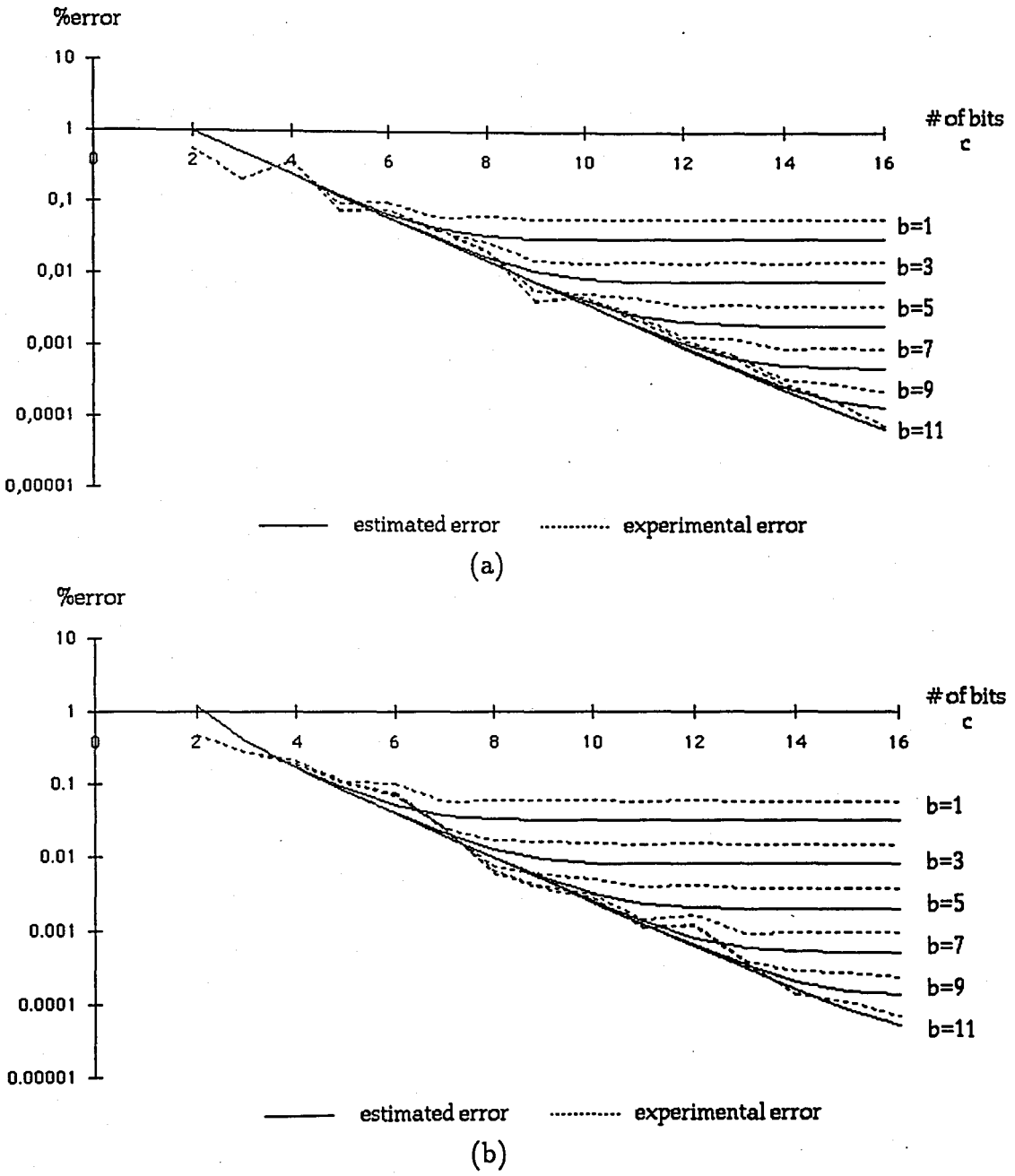


FIGURE 2.24. Experimental and estimated errors on a two-band-three-level multirate system with truncated fractional bus quantization at the output for the Lenna image. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)

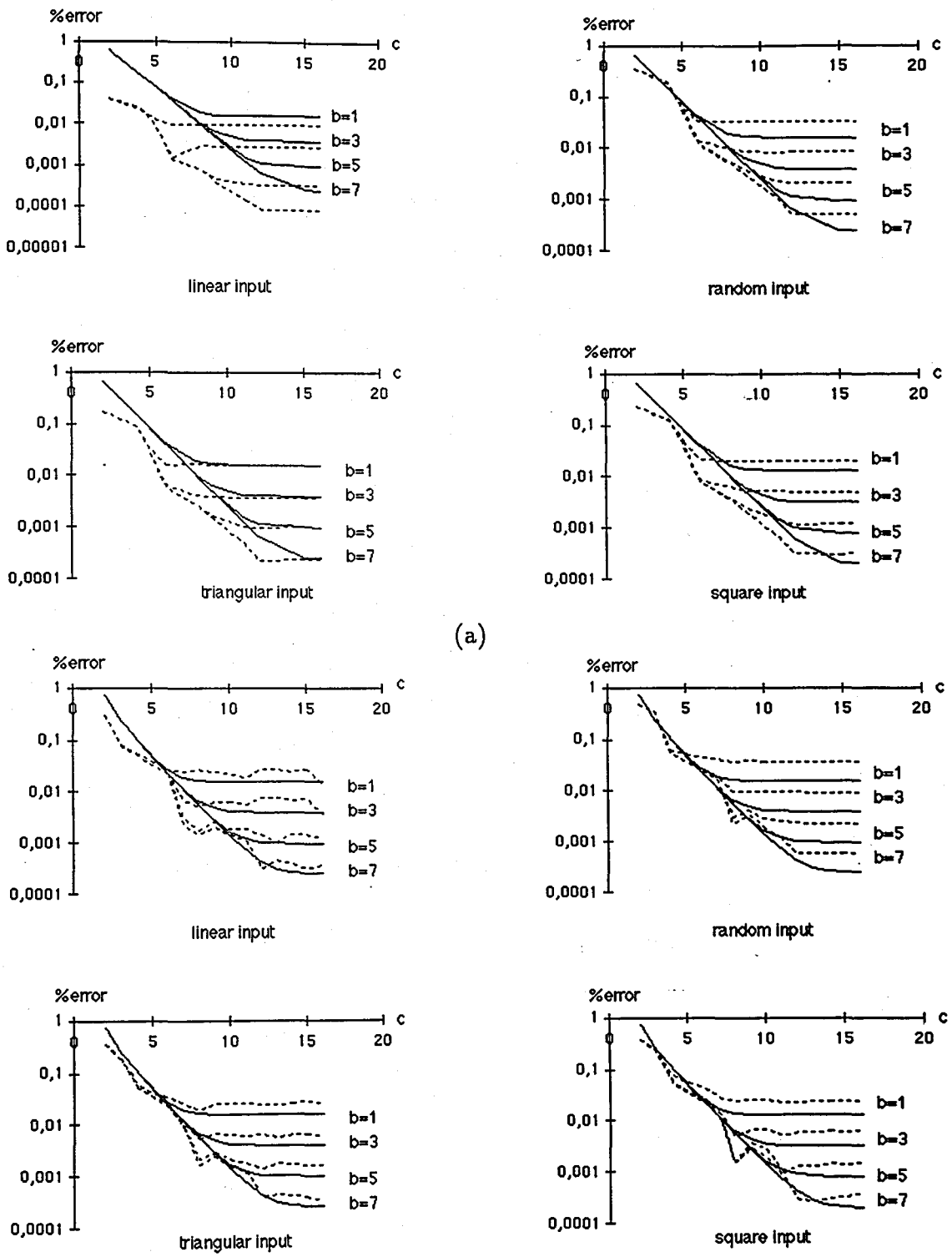


FIGURE 2.25. Experimental and estimated errors on a four-band multirate system with truncated fractional bus quantization at the output. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)

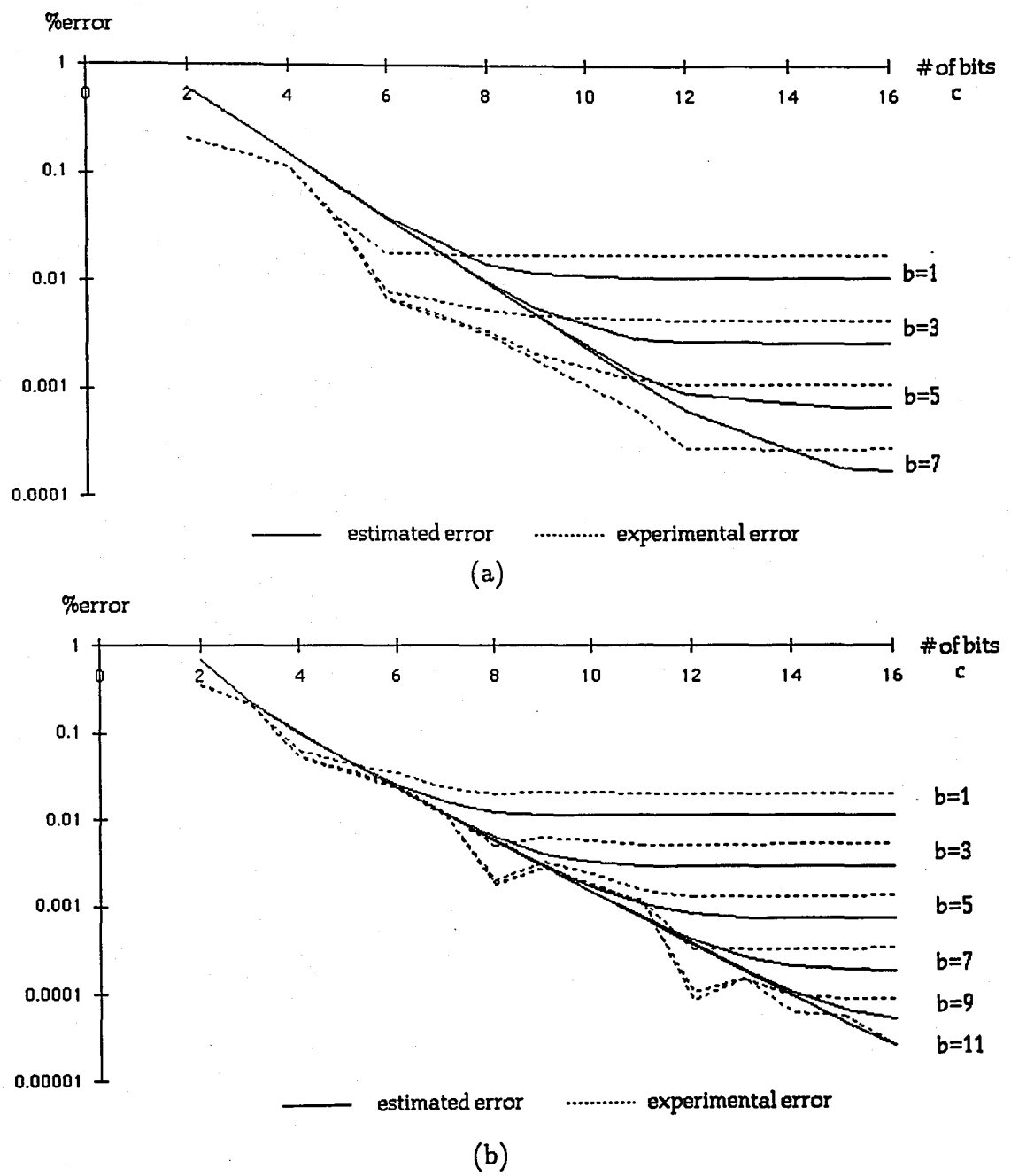


FIGURE 2.26. Experimental and estimated errors on a four-band multirate system with truncated fractional bus quantization at the output for the Lenna image. Coefficient quantization stepsize is Δ_a for (a) and Δ_b for (b)

3. MULTIPLIERLESS REALIZATION OF FIR-BASED SYSTEMS

In Digital Signal Processing (DSP) algorithms, most of the operations are arithmetic operations. While implementing these algorithms, multiplication and division require more complex architectures than others. As general multiplication and division architectures occupy huge areas compared to additions and subtractions, using these general architectures are avoided as much as possible. This can be done when a set of signals is multiplied by some constant coefficients. In this case, multiplications are represented by a series of adders and/or subtractors and shifters. This kind of multiplier representation is very effective in terms of area, delay and power compared to general multipliers. The hardware systems that use this representation for multipliers are called as *multiplierless* systems. As adders and subtractors have similar structures, both of them will be referred to as "adders" unless their difference is explicitly mentioned.

Even though multiplierless systems are more cost effective than general ones, there is a trend to make these systems even more effective. The representation style of the constants determines the number of adders required. It is proved by Garner that Canonic Signed Digit (CSD) representation requires, on average, 33% fewer adders than standard binary representation [55]. CSD representation has the nice property of standard binary representation such that each number is uniquely represented [56]. CSD is known as a special case of signed digit (SD) representation, and is the representation which requires the fewest adders. However, the number of adders can be minimized by using intermediate results. It is proved that the designs using SD representation and intermediate results can obtain an average improvement of 16% for 12-bit wordlength and 26.6% for 32-bit wordlength over CSD [57]. However it must be noted that a number can have many different SD representations and this increases exponentially as the wordlength increases linearly. Therefore, one must find the best SD representation for each constant, but this is a time-inefficient process because finding the best representation is an $\mathcal{NP}$-complete problem. Bull and Horrocks tried to solve this problem by heuristic methods resulting in suboptimal solutions [58]. Dempster

was able to solve this problem at optimality for all numbers with wordlengths up to 12-bits by using graphs and topology, and stored all results as the entries of a lookup table. Going beyond this wordlength exhaustively is not only time-inefficient but also memory-inefficient, as stated in [57].

In DSP systems, there are several constant coefficients. Therefore, instead of optimizing each coefficient separately, one can go through optimization of all coefficients at once. There are basically two groups of methods in Finite Impulse Response (FIR)-based DSP systems. In the first group of methods, after deciding on the characteristics of the FIR-filter and the wordlength, each tap's coefficient is chosen such that it is represented by using either CSD or a single adder, [59]-[64]. In the second group, the coefficients are chosen to reflect the system characteristics. After they are quantized with a predetermined wordlength, a set of partial sums that reduces the number of adders and/or shifters in the system is formed using groups of coefficients or each coefficient separately, [58], [65]-[76]. In other words, the filter is designed with implementation constraints in mind for the first group, whereas the best implementation is chosen after the design is done for the second group.

It is observed that the systems produced using the methods in the second group have better performance than the first one if same number of adders are used. This is not surprising because the quantized response of the system is known and accepted in the second group before the choice of partial sums which exactly reflect the quantized coefficients. However, coefficients are chosen in the first group such that it will resemble the quantized system response.

The method explained in this paper falls into the second category. All of the methods described in second group are capable of handling only FIR-based systems except one [71], which handles all kinds of systems. Our algorithm is able to handle all systems which can be written as the addition of some scaled inputs, i.e. some inputs are multiplied by some constants. This is the usual case for most of the recent DSP systems using multirate signal processing techniques or transforms for reducing computational complexity without compromising from the system quality. In these techniques, as some constants appear more than once, realizing each of them separately is a redundant process and increases area and power of the system firmware. FIR-based systems constitute a subset of this kind of systems. Some of the FIR-based algorithms

can produce better results than ours in single FIR filters in the system. However, when multiple FIR filters are used with similar tap coefficients with decimators and interpolators like wavelet transform, our algorithm outperforms all algorithms in the number of adders, shifters and delay elements. The main reason is the fact that our algorithm uses CSD representation of all coefficients in the system and handles all of them *at the same time.*

In the next section, the problem will be stated after giving some definitions that are used throughout the paper. Then, in Section 3.2 the theoretical foundations of the two-term representation of a system is explained. Due to the nature of the problem, either a mathematical model which is explained in Section 3.3 can be formed and solved optimally by branch-and-bound or the problem can be solved near-optimally by a greedy algorithm explained in Section 3.4. In the worst case, it converges much faster than [68] and can work easily for all wordlengths without suffering from memory requirements. The final impact is done by using the greedy method iteratively so that the number of adders can be reduced further. This is explained in Section 3.5. The results can be improved by using refinements explained in Section 3.6. Experimental results are presented in Section 3.7. The final section mentions about the determination of wordlengths of all nodes after the expanding the FIR-based nodes as adders and negators.

3.1. Problem Statement and Definitions

While defining terms, the four-tap FIR filter of [71] will be used. Although the coefficients used in this example are integers, it does not affect the generality of the problem because all fixed point fractional numbers can be expressed as integer numbers. The coefficients used in this example and their CSD representations are tabulated in TABLE 3.1.

A *two-term* is an odd number formed by combining only two nonzero entries in a CSD number. For instance, in the CSD representation of a, there are 5 two-

TABLE 3.1 CSD representation of FIR filter coefficients. $\underline{1}$ stands for -1.

a	815	$10\underline{1}0\underline{1}0\underline{1}000\underline{1}$
b	621	$0\underline{1}0\underline{1}00\underline{1}0\underline{1}01$
c	831	$10\underline{1}0\underline{1}000000\underline{1}$
d	105	$000\underline{1}0\underline{1}0\underline{1}001$

terms: $17(\equiv 10001)$, $-63(\equiv \underline{1}000001)$, $257(\equiv 100000001)$, $-1023(\equiv \underline{1}0000000001)$, $-3(\equiv \underline{1}01)$. Each two-term can have *replicas* formed by either shifting or negating or applying both on the two-term. For example, the two-term 17 has three replicas in the number: negated, shifted by 4 and negated, and shifted by 6. A two-term can appear in more than one constant. If TABLE 3.1 is examined, it can be easily observed that the two-term 17 appears in the CSD representations of constants a, c and d. The constants in the system can be expressed using two-terms. Some constants may have odd number of nonzero entries as a and b does. In this case, additional *single-terms* must be used to express this type of constants exactly. Each bit used to represent a constant is a single-term.

Our first aim is the minimization of the number of adders. This can be achieved by using the two-terms and their replicas. The key idea is the usage of minimum number of two-terms which have the best-fitting replicas for number representation. After finding the two-terms, *combination additions* can be used to add up the replicas in the expression of the constants. After the decision on two-terms and replicas for additions, the second aim is the minimization of shifting operations.

For the FIR example of TABLE 3.1, only three two-terms are sufficient as it can be seen in FIGURE 3.1. There are six combination additions for constant formations using replicas. There are seven shifters. Note that additionally, three adders are needed for intra-tap additions of the FIR filter.

This approach can be applied safely for systems which are made up of linear subsystems, without separating subsystems if timing and switching is adjusted correctly. So the problem can be stated as the minimization of the number of adders, shifters and registers of a given system or a subsystem.

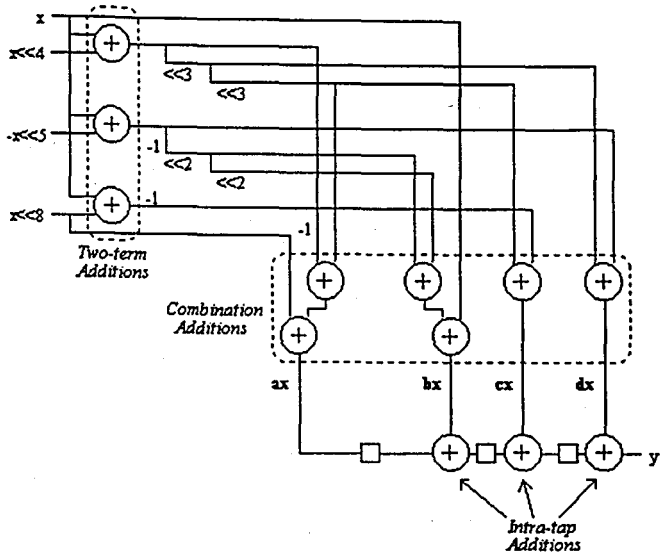


FIGURE 3.1. Realization of the FIR filter.

3.2. Theoretical Base of the Two-Term Method

At this point, some mathematical sets which will be used in the discussion below will be defined. Absolute value of a set stands for the cardinality of the set, e.g., $|S|$ is cardinality of S.

The set of constants which will undergo adder minimization process is defined as C and all of the constants are represented using CSD. The set of two-terms for the constants in C is T. The set which contains replicas of the two-term $t \in T$ is defined as R_t. The union of these sets form the general replica set, R (i.e., $R_t \subset R$ for all $t \in T$). This set can be partitioned such that the replicas that appear in a constant c form the set R_c. The replica sets can also be united to form the set $R_{c,j}$ for each j'th nonzero entry in a constant $c \in C$. Obviously, each $R_{c,j}$ is a subset of R. The constants that have an odd number of nonzero entries constitute the set C_Θ such that $C_\Theta \subseteq C$. For each constant $c \in C_\Theta$, there is a set which is formed by expressing each nonzero entry as an element of the set. This set is defined as Θ_c. The union of these sets form the general set for the nonzero entries of the constants with odd number of nonzero entries, Θ (i.e., $\Theta_c \subset \Theta$ for all $c \in C_\Theta$).

Theorem 1 *If there are $|\Theta_c|$ nonzero entries in the CSD representation of a number,*

then there are $|\Theta_c| - 1$ *additions to realize the number [55].*

Corollary 1 *If there are* $|\Theta_c|$ *nonzero entries for each* $c \in C$*, then there are* $\sum_{c \in C} |\Theta_c| - |C|$ *additions to realize the system.*

The set of all possible adders for realizing the constants is formed by uniting R and Θ. Note that $|R \cup \Theta|$ is greater than the adder quantity determined by Corollary 1. Therefore it is a redundant set.

Proposition 1 *If there are* $|\Theta_c|$ *nonzero entries in a constant* $c \in C$*, then there are* $\left\lfloor \frac{|\Theta_c|}{2} \right\rfloor$ *replicas in* R *to express* c*.*

Proof: There are $\frac{|\Theta_c|}{2}$ replicas in c which uses $|\Theta_c|$ nonzero entries. All of them cannot be used for expression because each entry of c appears $|\Theta_c| - 1$ times in R. Therefore, replicas for number expression must be selected in such a way that each entry appears only once. If $|\Theta_c|$ is an even number, it is obvious that $\frac{|\Theta_c|}{2}$ replicas are needed to express c. This formula needs a slight modification if $c \in C_\Theta$: It is obvious that at least an entry will not appear in the expression. This slack can be compensated by using the elements in Θ.

Corollary 2 *If* C_Θ *is not empty, then for each constant* $c \in C_\Theta$*, one and only one element (the single-term)* ϑ*, such that* $\vartheta \in \Theta_c$ *is necessary to express the number exactly.*

Proof: In Proposition 1, it is claimed that $\left\lfloor \frac{|\Theta_c|}{2} \right\rfloor$ replicas are necessary to express $c \in C_\Theta$. A replica stands for two entries in c. Since there are $|\Theta_c|$ nonzero entries in c, then one nonzero entry ($|\Theta_c| - 2\left\lfloor \frac{|\Theta_c|}{2} \right\rfloor = 1$) is needed from Θ_c.

Corollary 3 *If there are* $|\Theta_c|$ *nonzero entries for each* $c \in C$*, then there are* $\sum_{c \in C} \left\lfloor \frac{|\Theta_c|}{2} \right\rfloor$ *replicas in* R *to express the system.*

Corollary 4 *If there are* $|\Theta_c|$ *nonzero entries for each* $c \in C$*, then there are* $\sum_{c \in C} \left\lceil \frac{|\Theta_c|}{2} \right\rceil - |C|$ *combination additions to express the system. If* C_Θ *is not empty, then* $|C_\Theta|$ *of those additions is reserved for the additions of the single-term.*

Proof: Corollaries 1, 2 and 3 can be used for the proof.

On using two-term expressions, it is obvious that the number of combination additions cannot be reduced. Then, replicas in R must be used in such a careful way that the adder number given in Corollary 3 is reduced. Note that the number of replicas will not change. The important thing is the number of two-terms in T that are used to implement the system. Then our target set $R \cup \Theta$ reduces to $T \bigcup \Theta$ because R implies T.

Due to the structure of the problem, there is a strong possibility that selection of two-terms problem is NP-complete, because guessing subsets is in NP and the membership in NPC can be shown by polynomially reducing set packing feasibility problem to the selection of two-terms problem.

We can be solve this problem optimally by using well-known integer programming methods. algorithms like branch-and-bound. Also some heuristics can be developed to solve it suboptimally. In this study, both a mathematical model that can be solved at optimality using branch-and-bound and a greedy heuristic method is developed for solving the problem near-optimally.

After solving the problem, the solution set $T^* \bigcup \Theta^*$ is formed. Note that it is no more a redundant set for system representation. Also it should not be forgotten that $|\Theta^*| = |C_\Theta|$ and its value is always fixed before solving the problem since it is a combination addition. Therefore, using Corollary 4, the total number of additions in C is determined by

$$|T^*| - |C| + \sum_{c \in C} \left\lceil \frac{|\Theta_c|}{2} \right\rceil \tag{3.1}$$

3.3. Mathematical Model for Optimum Solution

Let us define binary variables z_t and z_{tr} such that

$$z_t = \begin{cases} 1, & \text{if } t \text{ is selected} \\ 0, & \text{otherwise} \end{cases}, \forall t \in T, \tag{3.2}$$

$$z_{tr} = \begin{cases} 1, & \text{if replica } r \text{ of } t \text{ is selected} \\ 0, & \text{otherwise} \end{cases}, \forall t \in T, \forall r \in R, \tag{3.3}$$

If at least one replica is selected, then its corresponding two-term will also be selected. This can be modelled as:

$$\begin{matrix} \sum_{r \in R_t} z_{tr} - z_t \geq 0 \\ \sum_{r \in R_t} z_{tr} - |R_t| \, z_t \leq 0 \end{matrix}, \forall t \in T. \tag{3.4}$$

According to Proposition 1 and Corollary 2, even though there are $|\Theta_c| - 1$ replicas for each j'th nonzero entry in each constant $c \in C$, at most one of the replicas can be used to represent each j'th nonzero entry:

$$\sum_{r \in R_{c,j}} z_{tr} \leq 1, \quad \begin{matrix} 1 \leq j \leq |\Theta_c|, \forall c \in C, \\ \exists t \in T. \end{matrix} \tag{3.5}$$

We can express Proposition 1 for each constant $c \in C$ as follows:

$$\sum_{r \in R_c} z_{tr} = \left\lfloor \frac{|\Theta_c|}{2} \right\rfloor, \quad \forall c \in C, \quad \exists t \in T. \tag{3.6}$$

Our aim is to minimize the number of two-terms in T. Therefore, the objective function will be

$$\min f_T = \min \sum_{t \in T} z_t \tag{3.7}$$

The model will be formed by combining the objective defined above with constraints (3.4), (3.5) and (3.6):

$$
\begin{aligned}
&\min f_T = \quad \min \textstyle\sum_{t\in T} z_t \\
&\text{subject to} \\
&\qquad \textstyle\sum_{r\in R_c} z_{tr} = \left\lfloor \frac{|\Theta_c|}{2} \right\rfloor \qquad \forall c \in C, \exists t \in T, \\
&\qquad \textstyle\sum_{r\in R_t} z_{tr} - z_t \geq 0 \qquad \forall t \in T, \\
&\qquad \textstyle\sum_{r\in R_t} z_{tr} - |R_t|\, z_t \leq 0 \qquad \forall t \in T, \\
&\qquad \textstyle\sum_{r\in R_{c,j}} z_{tr} \leq 1 \qquad 1 \leq j \leq |\Theta_c|, \forall c \in C, \exists t \in T, \\
&\qquad z_t \in B^1 \qquad \forall t \in T, \\
&\qquad z_{tr} \in B^1 \qquad \forall t \in T, \forall r \in \overline{R},
\end{aligned}
\tag{3.8}
$$

This is a binary integer programming problem and can be solved by one of the known techniques depending on its size [80]. The optimum value obtained, f_T^* stands for $|T^*|$. Then total number of adders can be obtained using Equation (3.1). The single-terms $\vartheta \in \Theta$ can be determined after eliminating selected two-terms and replicas from the representation of the constants $c \in C$.

Proposition 2 *The optimal results are produced in* $O\left(2^{|R|}\right)$ *time in the worst case.*

Proof: There are $|R|$ replicas to be checked whether any of them exist in the final result.

3.3.1. Algorithm for Optimal Solution

The algorithm for optimal solution can be summarized as follows:

1. Quantize constants under consideration with the given wordlength or predetermined output system error [78], [79].
2. Represent all quantized coefficients using CSD.
3. Obtain integer forms of all numbers and scale them so that they are odd numbers to form C and C_Θ.
4. Form the sets T, Θ, R.

TABLE 3.2 Quantized filter coefficients of Çağlar's four-band system.

taps	G_0	G_1	G_2	G_3
0	−.0664062	−.09375	−.09375	−.0664062
1	.09375	.0664062	−.0664062	−.09375
2	.40625	.566406	.566406	.40625
3	.566406	.40625	−.40625	−.566406
4	.566406	−.40625	−.40625	.566406
5	.40625	−.566406	.566406	−.40625
6	.09375	−.0664062	−.0664062	.09375
7	−.0664062	.09375	−.09375	.0664062

5. Form the model for the problem.

6. Solve the problem optimally by using branch-and-bound.

7. Reduce the number of shifts.

3.3.2. An Example

We will use the four-band wavelet transform pair suggested in [51] for our illustrations. The structures are presented in FIGURE 2.4. In this figure, each G is an FIR filter. Assume that both analysis and synthesis parts are connected via a noiseless channel and the reconstructed u must have an error of 0.7%. All coefficients are unscaled. After applying the method in [78], it is found that each G filter in the system must be quantized using 9 bits, including the sign bit, and the output error drops to 0.5%. When CSD representation is used instead of ordinary binary representation, the quantizing wordlength drops to 8 bits. The quantized system coefficients are tabulated in TABLE 3.2.

As it can be observed from this table, there are absolutely four coefficients: .0664062, .09375, .40625, .566406. Their integer equivalents are 17, 24, 104, 145 respectively. Note that the second and third integers are not odd. They can be made odd numbers after dividing them by eight. These four odd numbers constitute our set C and they are tabulated with their CSD representations in TABLE 3.3. Note that c_2

TABLE 3.3 Common terms of Çağlar's quantized four-band system.

c_0	17	00010001
c_1	3	00000101
c_2	13	00010101
c_3	145	10010001

and c_3 also constitute C_Θ. All formed sets are tabulated in TABLE 3.4. In this table, the term $t_1c_2s_0$ represents that first two-term is shifted 0 times to left and appears in second element of C.

The mathematical model of this system can be easily formed using Equation (3.8) and TABLE 3.4 as follows:

$$
\begin{aligned}
&\min f_T = && \min \textstyle\sum_{t=0}^{3} z_t \\
&\text{subject to} && \\
& && z_{00} = 1, \\
& && z_{10} = 1, \\
& && z_{01} + z_{11} + z_{12} = 1, \\
& && z_{02} + z_2 + z_3 = 1, \\
& && \textstyle\sum_{r=0}^{2} z_{0r} - z_0 \geq 0, \quad \sum_{r=0}^{2} z_{1r} - z_1 \geq 0, \\
& && \textstyle\sum_{r=0}^{2} z_{0r} - 3z_0 \leq 0, \quad \sum_{r=0}^{2} z_{1r} - 3z_1 \leq 0, \\
& && z_{01} + z_{11} \leq 1, \quad z_{11} + z_{12} \leq 1, \quad z_{01} + z_{12} \leq 1, \\
& && z_{02} + z_2 \leq 1, \quad z_{02} + z_3 \leq 1, \quad z_2 + z_3 \leq 1, \\
& && z_t \in B^1, \qquad \forall t \in T, \\
& && z_{tr} \in B^1, \quad \forall t \in T, \forall r \in R.
\end{aligned}
\tag{3.9}
$$

After solving this problem, the two-terms 17 and 3 are selected. Their used replicas are r_{0_0}, r_{0_2}, r_{1_0} and r_{1_1}. Single terms are Θ_{2_1} and Θ_{3_2}. In the replicas, there are no shifts. However, upscaling the constants requires 5 shifts. The number of total additions is 4 which is found by Equation (3.1). This means that part of FIGURE 2.4(a) can be realized using only 4 adders and 5 shifters for 8-bit quantization wordlength if the scheduling and clocking is adjusted correctly. The analysis part is shown in FIGURE 3.2 for illustration of typical implementation. Here, each adder is assumed to have a unit delay and the data bus carrying the fractional part is eight-bit wide.

TABLE 3.4 The sets formed using common terms of Çağlar's quantized four-band system. The term $t_1 c_2 s_0$ represents that first term is shifted 0 times to left and appears in second element of C.

C	$=\{c_0, c_1, c_2, c_3\}$	$=\{17, 3, 13, 145\}$
C_Θ	$=\{c_2, c_3\}$	$=\{13, 145\}$
T	$=\{t_0, t_1, t_2, t_3\}$	$=\{17, -3, 129, 9\}$
R_{t_0}	$=\{r_{0_0}, r_{0_1}, r_{0_2}\}$	$=\{t_0c_0s_0, t_0c_2s_0, t_0c_3s_0\}$
R_{t_1}	$=\{r_{1_0}, r_{1_1}, r_{1_2}\}$	$=\{-t_1c_1s_0, t_1c_2s_0, -t_1c_2s_2\}$
R_{t_2}	$=\{r_{2_0}\}$	$=\{t_2c_3s_0\}$
R_{t_3}	$=\{r_{3_0}\}$	$=\{t_3c_3s_4\}$
$R_{c_0} = R_{c_0,0} = R_{c_0,4}$	$=\{r_{0_0}\}$	$=\{t_0c_0s_0\}$
$R_{c_1} = R_{c_1,0} = R_{c_1,2}$	$=\{r_{1_0}\}$	$=\{-t_1c_1s_0\}$
R_{c_2}	$=\{r_{0_1}, r_{1_1}, r_{1_2}\}$	$=\{t_0c_2s_0, t_1c_2s_0, -t_1c_2s_2\}$
$R_{c_2,0}$	$=\{r_{0_1}, r_{1_1}\}$	$=\{t_0c_2s_0, t_1c_2s_0\}$
$R_{c_2,2}$	$=\{r_{1_1}, r_{1_2}\}$	$=\{t_1c_2s_0, -t_1c_2s_2\}$
$R_{c_2,4}$	$=\{r_{0_1}, r_{1_2}\}$	$=\{t_0c_2s_0, -t_1c_2s_2\}$
R_{c_3}	$=\{r_{0_2}, r_{2_0}, r_{3_0}\}$	$=\{t_0c_3s_0, t_2c_3s_0, t_3c_3s_4\}$
$R_{c_3,0}$	$=\{r_{0_2}, r_{2_0}\}$	$=\{t_0c_3s_0, t_2c_3s_0\}$
$R_{c_3,4}$	$=\{r_{0_2}, r_{3_0}\}$	$=\{t_0c_3s_0, t_3c_3s_4\}$
$R_{c_3,7}$	$=\{r_{2_0}, r_{3_0}\}$	$=\{t_2c_3s_0, t_3c_3s_4\}$
Θ_{c_2}	$=\{q_{2_0}, q_{2_1}, q_{2_2}\}$	$=\{0, 2, 4\}$
Θ_{c_3}	$=\{q_{3_0}, q_{3_1}, q_{3_2}\}$	$=\{0, 4, 7\}$

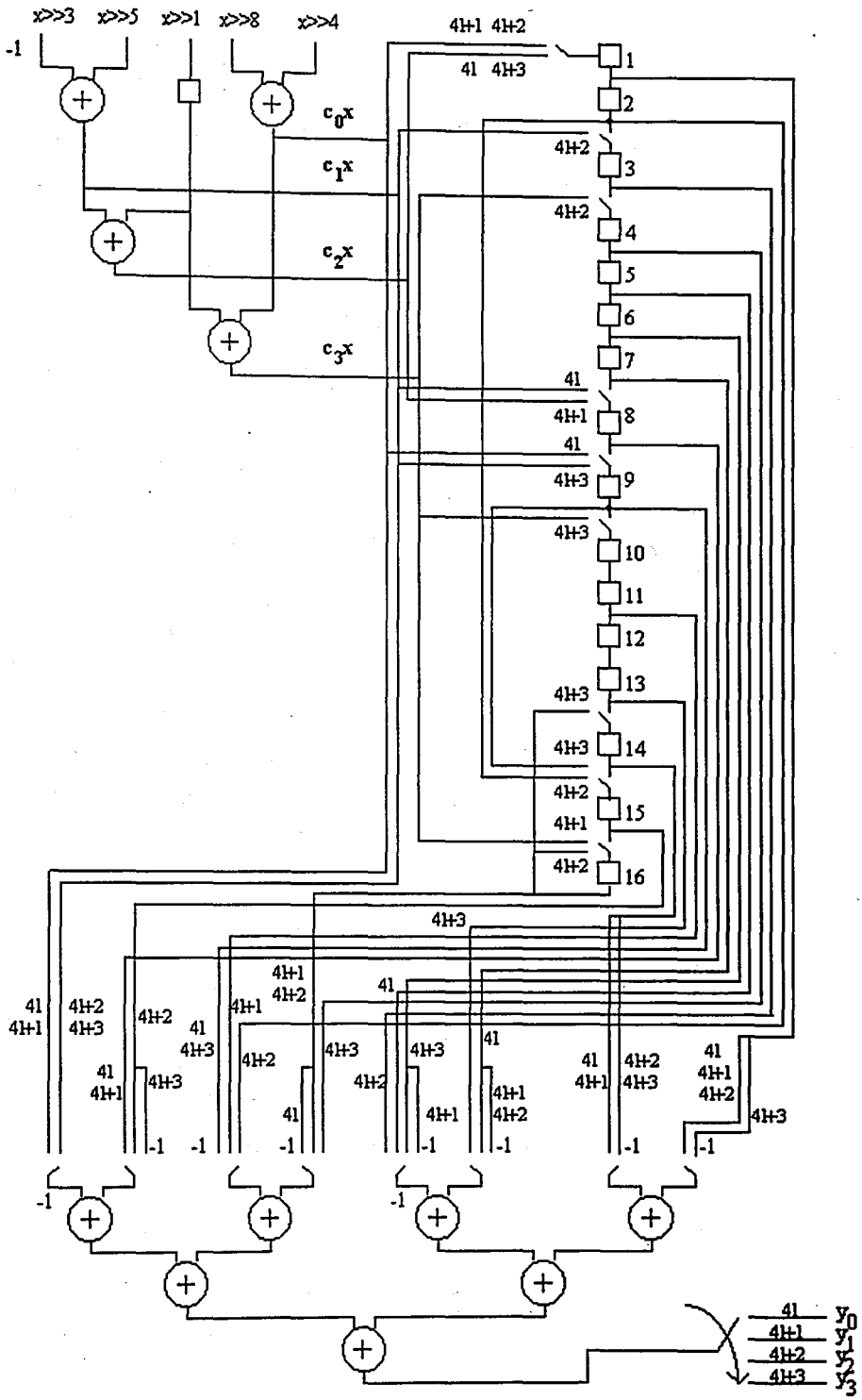


FIGURE 3.2. Realization of the analysis part of four-band wavelet transform pair.

3.4. Greedy Method

Remembering that our aim is minimizing the number of adders and shifters, a greedy method is developed to solve the problem. It, at least, produces an upper bound on the system representation in a very short time. The main point of the method is selecting the two-term which has the maximum number of replicas. The algorithm can be described as follows:

1. Quantize constants under consideration with the given wordlength or predetermined output system error [78], [79].
2. Represent all quantized coefficients using CSD.
3. Obtain integer forms of all numbers and scale them so that they are odd numbers to form C and C_Θ.
4. Initialize $T_F = \emptyset$, $R_F = \emptyset$ and $\Theta_F = \emptyset$ as the solution sets.
5. Form the sets T, Θ and R.
6. Calculate $g = \max_{t \in T} |R_t|$.
7. Choose t such that $|R_t| = g$. If there exists more than one t satisfying this condition then choose the one which is formed by combining two nonzero terms of different sign. If there is no or more than one t satisfying this condition, then choose t which is formed by combining the closest nonzero terms. These additional conditions help minimization of the final chip area.
8. $\{t\} \cup T_F \rightarrow T_F$.
9. $R_t \cup R_F \rightarrow R_F$.
10. Erase all $r \in R_t$ from the CSD representation of all $c \in C$, i.e., $c - r \rightarrow c$, $\forall r \in R_t$, $\forall c \in C$.
11. If there exists some c in C such that $c = 0$ or 2^l where l is a non-negative integer, then $C \setminus \{c\} \rightarrow C$. If $c = 2^l$ (i.e., $c \in C_\Theta$), then l determines ϑ, and $\{\vartheta\} \cup \Theta_F \rightarrow \Theta_F$.

12. If $C = \emptyset$, stop with T_F, R_F and Θ_F as the solution sets. Else, jump to step 5.

13. Reduce the number of shifts in the final result.

When compare the solution sets with the optimum ones, it is obvious that $|T_F| \geq |T^*|$, $|R_F| = |R^*|$ and $|\Theta_F| = |\Theta^*|$.

Proposition 3 *Greedy algorithm developed for finding minimum number of two-terms is a polynomial-time algorithm which produces the results in* $O(|\Theta_c|)$ *time.*

Proof. The algorithm runs $|T_F|$ times to produce the results. There can be at most $\sum_{c \in C} \left\lfloor \frac{|\Theta_c|}{2} \right\rfloor$ two-terms if each of them has no replicas according to Corollary 3. Then

$$|T_F| \leq \sum_{c \in C} \left\lfloor \frac{|\Theta_c|}{2} \right\rfloor \leq \frac{|C|}{2} \max_{c \in C} |\Theta_c| = \frac{|C|}{2} |\Theta_c| \tag{3.10}$$

where $|\Theta_c|$ is $\max_{c \in C} |\Theta_c|$.

After running the algorithm on the given example in the previous section, the same two-terms are selected. Their used replicas are r_{0_0}, r_{0_2}, r_{1_0} and r_{1_1}. Single terms are the same. In the replicas, there are no shifts. However, 5 shifts have to be used for upscaling the constants. The number of total additions and total shifts are again 4 and 5.

3.5. Iterative Method

Everything seems fine up to this point: A model is set up which can be used to solve the problem optimally but in exponential time, and developed an algorithm which solves the problem near-optimally but in linear time. In addition, both of them run safely for all wordlengths. Yet, there is still an opportunity of reducing adders furthermore. This is done by finding out higher order two-terms which are formed by combining either a two-term and a single-term, or a pair of two-terms. The theoretical base set in Section 3.2. is still valid.

Now let's illustrate the situation with a small example: Assume that there are two numbers, a (= 1001010001) and b (= 0101010001). As it can be seen easily, after running the two-term algorithm, only one two-term will be selected, i.e. 101, and there will be five additions two realize the system. However the total number of additions will be reduced if the numbers are rewritten using the found two-term as $a = 10000f0001$ and $b = 01000f0001$ where f stands for 101. If the algorithm is rerun again, then a second order two-term will be formed as $f0001$ and the system will be realized by using four additions. There is no need to run the algorithm again because there is no common two-term in both numbers. So iterative usage of the algorithm can reduce the number of adders in a system further.

It is evident that both single-run near-optimal and optimal methods can be used iteratively until there is no common two-term in all numbers. The iterative near-optimal one runs again in $O(|\Theta_c|)$ time and the optimal one in exponential time. However it should not be forgotten that iterative usage of the optimal method may not yield an optimum solution for the overall system.

3.6. Refinements

The two-terms are formed by either adding two subterms, or subtracting two subterms, or adding two negative subterms. The subtractions cover much more area than adders. The effective adder count is different than the adders/subtractors count. In effective adder count, the cost of a subtraction or a negative addition is usually twice than that of a negator or an adder and the cost of a negator is equivalent to that of an adder. Therefore effective chip area will be less if the refinements are done:

1. Negative additions can be replaced by regular adders and inverting the sign of the two-term in subsequent two-terms as shown in FIGURE 3.3.

2. If a two-term is a subtraction and it appears as a subtrahend more than it does as a minuend in subsequent two-terms, then the inputs are switched and the sign of the two-term in subsequent two-terms is inverted. This is shown in FIGURE

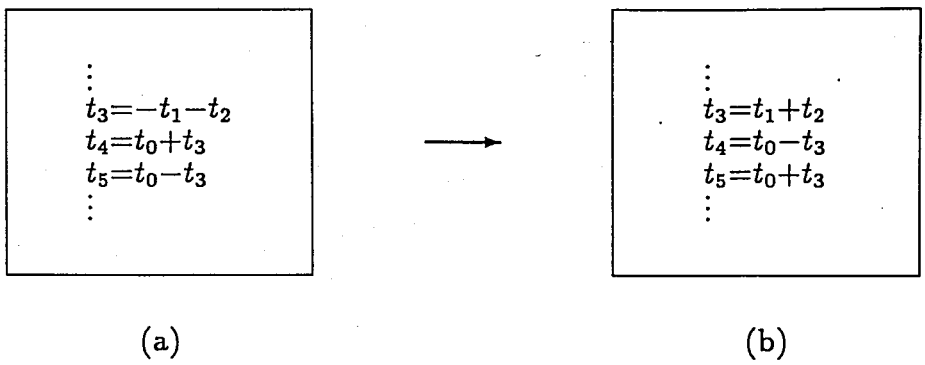


FIGURE 3.3. Refinement 1: (a)Original form: cost=5, (b)after the removal of negative additions: cost=4. Each t_i stands for a two-term or an input

3.4. This refinement process produces better results if the conversion starts from the highest order two-terms.

3. Subtractions can be replaced by putting a negator at the output of the subtrahend and converting the subtractor to an adder. This change produces very efficient results if the subtrahend appears in more than one operation. This is shown in FIGURE 3.5.

3.7. Experiments

The codes that generate the model and solve the problem near-optimally are written in standard C. The programs are applied on the experiments shown in TABLE 3.5. In this table, *org* stands for original number of adders/subtractors and shifters, *opt* stands for the optimal solution of the model which can be solved either by a standard 0-1 integer programming solver, *nopt* stands for the near-optimal solution which is found by using the greedy algorithm described above, and *itnopt* stands for the near-optimal solution which is found by using the greedy algorithm iteratively. The terms DCT, B4L0 and B2L3 stand for the one-dimensional eight-point DCT, the four-band wavelet

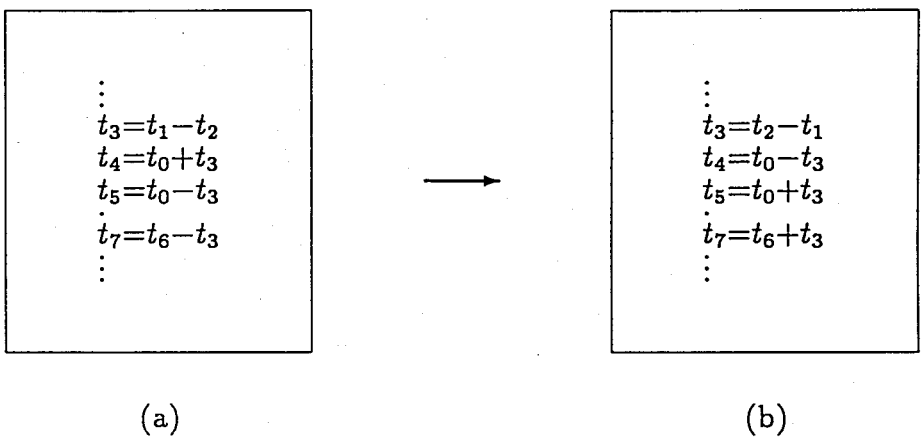


FIGURE 3.4. Refinement 2: (a)Original form: cost=7, (b)after switching inputs: cost=6. Each t_i stands for a two-term or an input

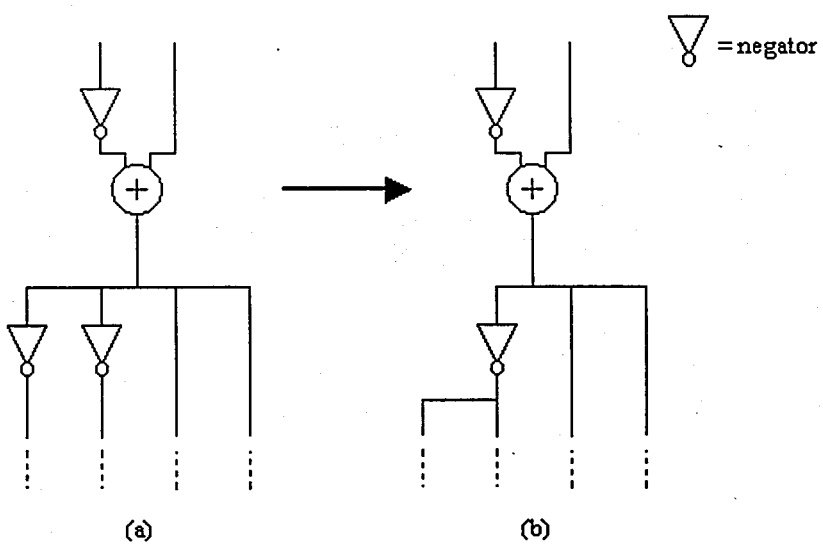


FIGURE 3.5. Refinement 3: (a)Original form: cost=4, (b)after refinement: cost=3.

TABLE 3.5 Experimental Results: DCT-Discrete Cosine Transform, B4L0-Four Band Wavelet Transform, B2L3-Two Band Three Level Wavelet Transform

Example	*# of bits*	*# of shifts*				*# of adders*			
		org	*nopt*	*opt*	*itnopt*	*org*	*nopt*	*opt*	*itnopt*
DCT	8	208	11	10	10	144	10	9	10
	12	272	15	13	12	208	16	14	13
	16	352	17	19	19	288	19	18	19
	24	544	26	30	28	480	31	31	29
B4L0	8	80	6	6	5	48	4	4	4
	12	112	9	8	7	80	7	7	6
	16	160	8	9	8	128	11	10	8
	24	256	13	15	13	224	18	17	12
B2L3	8	108	7	7	7	72	6	6	5
	12	180	7	7	7	144	11	11	8
	16	234	15	18	13	198	19	19	12
	24	270	19	22	12	234	22	22	12

transform whose coefficients are given in [51] and the two-band three-level wavelet transform whose coefficients are given in [54], respectively. The number of adders and shifters obtained by these algorithms is less than that of the previous methods. As it can be observed in some of the examples, the results produced by the greedy algorithm are exactly the same as the optimal results and iterative near-optimal method can produce better results than both.

The effective cost without refinements appears as *noref* column in TABLE 3.6. The effective cost after applying Refinement 1 is the *ref*1 column, after applying refinements 1 and 3 is the *ref13* column, after applying all refinements is the *refall* column and in the above-mentioned table. In this table, the combination addition counts are also added. As it can be observed, refinement process produces better results.

TABLE 3.6 Effective adder counts for experiments: DCT-Discrete Cosine Transform, B4L0-Four Band Wavelet Transform, B2L3-Two Band Three Level Wavelet Transform

Example	*# of bits*	*noref*	*ref1*	*ref13*	*refall*
DCT	8	31	30	28	28
	12	35	35	31	31
	16	42	41	38	38
	24	54	54	52	52
B4L0	8	20	20	20	20
	12	24	24	24	22
	16	29	29	28	26
	24	34	34	31	29
B2L3	8	18	18	18	18
	12	24	24	23	22
	16	26	26	26	26
	24	27	27	27	27

3.8. Determination of Nodal Wordlengths

After decomposing all FIR-based nodes into adders and negators, the system is now composed of adders, negators and multipliers and node set given in DFG file, N, is transformed to the enhanced node set $\aleph$. Now an FIR-based node is a supernode consisting of ordinary nodes. Since this HLS tool is designed to handle two-input operators, the nodal wordlengths can be determined as shown in TABLE 3.7. In this table, the operator definition of ADD has the carry information at the input inherently. The colon (:), stands for the fractional point in the operator. Therefore, nodal wordlength, n_l, for node n can be calculated as

$$n_l = n_{l_i} + n_{l_f} \tag{3.11}$$

where subscript l_i and l_f are integer and fractional lengths, respectively. Note that ADD operation requires an extra addition by 1which is necessary to adjust the value of the carry bit for the correct calculation of signed numbers.

TABLE 3.7 Operator wordlength determination. l_i stands for wordlength of the integer part and l_f stands for that of the fractional part.

Type	*Definition*	*Operational Wordlength*	*Carry*
ADD	$(i_1 : f_1) + (i_2 : f_2)$	$\max(l_{i_1} + l_{i_2}) : \max(l_{f_1} + l_{f_2})$	YES
NEG	$\overline{(i : f)}$	$l_i : l_f$	NO
MUL	$(i_1 : f_1) \times (i_2 : f_2)$	$(l_{i_1} + l_{i_2}) : (l_{f_1} + l_{f_2})$	NO

4. MODULE SELECTION AND SCHEDULING

Module selection requires an optimization process because of user-defined design objectives and constraints. This process selects and allocates both operators and storage elements like registers and latches as described in Section 4.1. Then scheduling is done by lifetime analysis and table search methods (Section 4.2.). Using lifetime analysis, minimum number of registers in FIR-based nodes is determined. If the user-defined constraints are not satisfied, new constraints are set and new modules are selected. This process is repeated until the best result is found, whether feasible or not.

4.1. Module Selection

Module selection is an $\mathcal{NP}$-complete problem [2]. Our aim is to realize the system with the given libraries and system clock frequency so as to find at least a solution. Therefore, exploitation of structural pipelining ([14], [15]) is unavoidable. As a result, for each operation, the algorithm should handle selection, allocation and scheduling of latches as well as those of arithmetic operation units. Therefore, modelling the system is nearly impossible because all possible combinations of candidate arithmetic and storage units must take place as variables in the model. This is why the development of a heuristic method is preferred.

There can be several methods for optimal selection of modules. In this developed HLS tool, module selection is done optimally for each node, instead of making selection for all nodes altogether. It is assumed that locally optimal solutions will yield a general optimal solution. Since the modules have to be selected according to the user defined strengths for final chip area, delay and power, i.e. $\{s_a, s_d, s_p\}$, multiobjective optimization seems to be the best method. However a method utilizing a single objective with normalized variables and library values can also be used. This is explained

in Subsection 4.1.1. If the user sets upper bounds on area, delay and power[1], module selection must be done in such a way that each node will fit in those bounds. This requires the determination of nodal constraints as explained in Subsection 4.1.2. The selection process of latches and arithmetic units (AU) is explained in Subsection 4.1.3.

4.1.1. Library Generation

The input file has to contain the technology, system clock frequency and library information. The nodal AU libraries are formed as shown in TABLE A2. As it can be seen in this table, the user can use AU libraries under specified technology both for some and for all nodes. However, latch libraries are set only for all nodes. The format of library elements is given in Appendix B. Two candidate libraries, Λ and Q, are formed separately for latches and AUs respectively for each node in such a way that they only contain the modules operating at the system clock frequency or higher. For example, if the node is a multiplier and system clock frequency is $300MHz$, then the AU library must consist of the multipliers operating at $300MHz$ or higher.

Then, maximum module area, power and delay of both libraries, $\{a_{\max}, d_{\max}, p_{\max}\}$, is searched:

$$\begin{aligned} a_{\max} &= \max_{i \in \Lambda \cup Q} a_i \\ d_{\max} &= \max_{i \in \Lambda \cup Q} d_i \\ p_{\max} &= \max_{i \in \Lambda \cup Q} p_i \end{aligned} \tag{4.1}$$

All of the modules are scaled by these maximum values yielding a maximum value of 1 for each feature. Note that the normalized features are unitless. For example, if there are two adders and a latch of areas $10\mu m^2$, $15\mu m^2$ and $20\mu m^2$, then their values will be 0.5, 0.75 and 1 respectively after normalization with $a_{\max} = 20\mu m^2$.

It must be noted that since the HLS tool works on bit-parallel-digit-serial architectures, the modules in the libraries must be of this type. Otherwise, their format convertor schemes must exist inherently in the libraries (Appendix B).

[1]The tool does not take routing and interconnect area, power and delay into consideration.

4.1.2. Nodal Constraint Generation

As explained above, nodal constraints are generated if and only if there exist system constraints specified by the user. Otherwise they are set to infinity, meaning that they are unbounded:

$$\begin{aligned} \overline{a}_n &= \infty \\ \overline{d}_n &= \infty \\ \overline{p}_n &= \infty \end{aligned} \tag{4.2}$$

where $\left\{\overline{a}_n, \overline{d}_n, \overline{p}_n\right\}$ are maximum allowable area, delay and power for node n.

Nodal area and power constraints are generated according to the wordlength of the operators. Nodal delay constraints are determined according to the longest path.

The area of an operator is proportional with its wordlength in bit-parallel architectures. Even though there are several factors for the determination of power of an operator, a rough but not incorrect assumption can be the same amount of power dissipation for each bit processor. If there are $|\aleph|$ nodes in the system, then nodal constraints can be written as

$$\begin{aligned} \overline{a}_n &= \overline{a}_{sys} \frac{n_l}{\sum_{n \in \aleph} n_l} \quad , \forall n \in \aleph, \\ \overline{p}_n &= \overline{p}_{sys} \frac{n_l}{\sum_{n \in \aleph} n_l} \quad , \forall n \in \aleph, \end{aligned} \tag{4.3}$$

where $\overline{a}_{sys}$ and $\overline{p}_{sys}$ stand for maximum allowable area and power for the system, n_l is the wordlength of the n'th node.

For calculating nodal delay constraints, an iterative method is preferred as shown in FIGURE 4.1. In this figure, $|\psi_i|$ is the number of nodes on path ψ_i, $\overline{d}_i$ is the path delay for ψ_i, and $\overline{d}_{sys}$ is the delay constraint for the system. The run-time of this algorithm is $O\left(|\Psi_\aleph|\right)$ where $\Psi_\aleph$ is the set of all possible paths from inputs to outputs for the enhanced node set $\aleph$.

```
DELAY_DISTRIBUTION (linked nodal list ℵ) {
    form paths from ℵ such that ψ ∈ Ψ_ℵ, ∀n ∈ ℵ ;
    for each ψ ∈ Ψ_ℵ
        d̄_ψ = d̄_sys;
    while ψ ≠ ∅ {
        for each ψ ∈ Ψ_ℵ if (|ψ*| = max_{ψ∈Ψ_ℵ} |ψ|) {
            for each n ∈ ψ* {
                d̄_n = d̄_ψ* / |ψ*|;
                for each ψ ∈ Ψ_ℵ and ψ ≠ ψ*
                    if (n ∈ ψ) {
                        remove n'th node from ψ;
                        d̄ψ = d̄ψ − d̄_n;
                        }
                }
            Ψ_ℵ = Ψ_ℵ \ {ψ*};
            }
        }
    }
```

FIGURE 4.1. Algorithm for nodal delay constraint generation

4.1.3. Latch and Arithmetic Unit Selection

If a node n is a multiplier then a multiplier of at least n_l bits is selected as shown in FIGURE 4.2. Note that n_l is the nodal wordlength as calculated in Section 3.8. No latch selection is done for this type of nodes because multipliers usually have built-in latches at their inputs.

In the selection of adder-based nodes, i.e. adders and negators, latches and arithmetic units are selected simultaneously. Note that a negator is a bank of inverters followed by an adder. Since the cost of an inverter is negligible compared to that of an adder, the cost due to inverters can be ignored. Also the negator has a single input, but an adder has two inputs. This means that the number of latches for the inputs of the adder is much more than that of an inverter. For adders and negators, the schemes of FIGURE 4.3 are proposed. The boxes stand for latches. Note that latch for the carry bit, C, at the input is dashed, meaning that the latch does not appear if the module is placed in the same time slot with its carry generator. Also the carry input is 1 if and only if the module is the least significant adder of the negator. For pipelined

SELECT_MULTIPLIER $(n, \{s_a, s_d, s_p\}, \{a_{\max}, d_{\max}, p_{\max}\}, \{\bar{a}_n, \bar{d}_n, \bar{p}_n\}, Q)$ {
for each $q \in Q$;
if$(q_l \geq n_l)$ {
$cost_q = s_a \frac{a_q}{a_{\max}} + s_d \frac{d_q}{d_{\max}} + s_p \frac{p_q}{p_{\max}}$;
$excess_q = level\left(\frac{\bar{a}_n - a_q}{a_{\max}}\right) + level\left(\frac{\bar{d}_n - d_q}{d_{\max}}\right) + level\left(\frac{\bar{p}_n - p_q}{p_{\max}}\right)$;
}
$minexcess = \min_{q \in Q} excess_q$;
$mincost = \min_{q \in Q} cost_q$ **such that** $excess_q = minexcess$;
select q^* **such that** $cost_{q^*} = mincost$ **and** $excess_{q^*} = minexcess$;
}

FIGURE 4.2. Algorithm for selection of multipliers.

modules, none of the initial latches exist for adders, and an additional bank of latches appear for inverters before adders for negators.

The algorithm for selection of adder based modules for each $n \in \aleph$ is shown in FIGURE 4.4 which calls the subroutine in FIGURE 4.8 for the selection of a non-pipelined module and the subroutine in FIGURE 4.9 for the selection of a pipelined module. This algorithm can be explained briefly for each $n \in \aleph$ as follows: Current wordlength is set to remaining wordlength which is initially n_l. The node of length current wordlength is realized using each element in both module and latch libraries. Then the cost of each realization is calculated. If there exists no realization satisfying nodal constraints, the realization with minimum constraint violation is picked. Otherwise, the realization with minimum cost is picked. Even though the selected realization may contain more than one module, one and only one module and latch combination is picked and placed. The related cost is subtracted from the constraints. The remaining wordlength is calculated by subtracting the wordlength of the module from the current wordlength. The procedure continues until the remaining wordlength is negative or zero.

In the algorithm *SELECT_LATCH_AND_ADD*, a data structure, M is formed to allocate j'th selected AU and latch in time slot i. The functions *level* and *check* are described as:

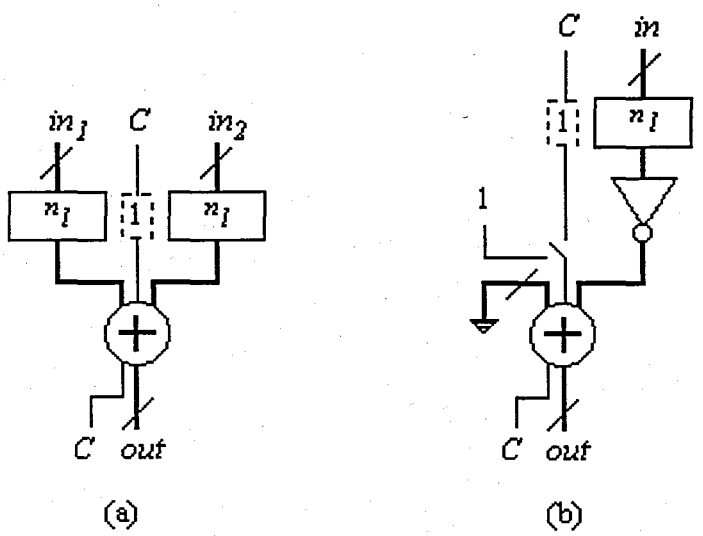


FIGURE 4.3. Proposed connection schemes for not-pipelined adder-based modules: (a) An adder, (b) a negator.

$$level(u) = \begin{cases} u & \text{, if } u > 0 \\ 0 & \text{, else} \end{cases} \tag{4.4}$$

$$check(u) = \begin{cases} u & \text{, if } u > 0 \\ \infty & \text{, else} \end{cases} \tag{4.5}$$

An important statement of this algorithm is the realization of the system using a $\lambda \in \Lambda$ and a $q \in Q$ for $n_{l_r} \leq n_l$ for the (i,j)'th entry of M where n_l is the nodal wordlength of node n. Assume that $n_{l_r} \leq m.q_l$ where m is an integer. There are two cases for this operation:

4.1.3.1. Case 1: Module q is a non-pipelined adder. If $m.d_q + d_\lambda \leq clock$, then $\varsigma.n_{l_r}$ bits of λ type latches and m number of q adders are needed to realize n_{l_r} bits of node n in one clock period as seen in FIGURE 4.5. Note that ς is a scalar as defined in FIGURE 4.4. The delay of this architecture, $d_{\lambda q}$ is, of course, *clock*.

If $m.d_q + d_\lambda > clock$, the node n must be realized in more than one clock cycle. Then there exists an integer $\kappa < m$ such that $\kappa.d_q + d_\lambda \leq clock$. Then n_{l_r} bits of node n can be realized in $\left\lceil \frac{m}{\kappa} \right\rceil$ clock cycles. The number of latches used can be determined

SELECT_LATCH_AND_ADD $(n, \{s_a, s_d, s_p\}, \{a_{\max}, d_{\max}, p_{\max}\}, clock, \{\bar{a}_n, \bar{d}_n, \bar{p}_n\}, \Lambda, Q)$ {
save $\bar{d}_n$ to t and set $\bar{d}_n = clock \left\lfloor \frac{\bar{d}_n}{clock} \right\rfloor$;
$n_l = n_{l_i} + n_{l_f}$;
form M ;
$state = 0$;
if n is an adder then $\varsigma = 2$;
else $\varsigma = 1$;
repeat{
$i = j = 0$;
$n_{l_r} = n_l$;
repeat{
for each $\lambda \in \Lambda$
for each $q \in Q$ {
realize an h-level pipelined architecture of n_{l_r} length
using m units of q and $m \times n_{l_r}$ units of λ ;
calculate $a_{\lambda q}$, $d_{\lambda q}$, $p_{\lambda q}$;
$cost_{\lambda q} = s_a \frac{a_{\lambda q}}{a_{\max}} + s_d \frac{d_{\lambda q}}{d_{\max}} + s_p \frac{p_{\lambda q}}{p_{\max}}$;
$excess_{\lambda q} = level\left(\frac{\bar{a}_n - a_{\lambda q}}{a_{\max}}\right) + level\left(\frac{\bar{d}_n - d_{\lambda q}}{d_{\max}}\right) + level\left(\frac{\bar{p}_n - p_{\lambda q}}{p_{\max}}\right)$;
}
$minexcess = \min_{\lambda \in \Lambda, q \in Q} excess_{\lambda q}$;
$mincost = \min_{\lambda \in \Lambda, q \in Q} cost_{\lambda q}$ such that $excess_{\lambda q} = minexcess$;
select (λ^*, q^*) such that $cost_{\lambda^* q^*} = mincost$ and $excess_{\lambda^* q^*} = minexcess$;
if $\left(q^*_{pipeline} = 0\right)$ *NOT_PIPED* $(M, i, j, \lambda^*, q^*, \varsigma, n_{l_r})$;
else *PIPED* $(M, i, j, \lambda^*, q^*, \varsigma, n_{l_r})$;
remove allocated latches' and AU's area and power costs from upper limits;
$\bar{a}_n = check(\bar{a}_n)$;
$\bar{d}_n = check(\bar{d}_n)$;
$\bar{p}_n = check(\bar{p}_n)$;
$n_{l_r} = n_{l_r} - q^*_l$;
}until $(n_{l_r} \leq 0)$;
if $(state = 0)$
if $(t \neq \infty$ and $\bar{d}_n = \infty)$ {
$\bar{d}_n = clock \left\lceil \frac{\bar{d}_n}{clock} \right\rceil$;
$state = state + 1$;
}
else $state = 0$;
}until $(state \neq 0)$;
}

FIGURE 4.4. Algorithm for selection of adder-based modules.

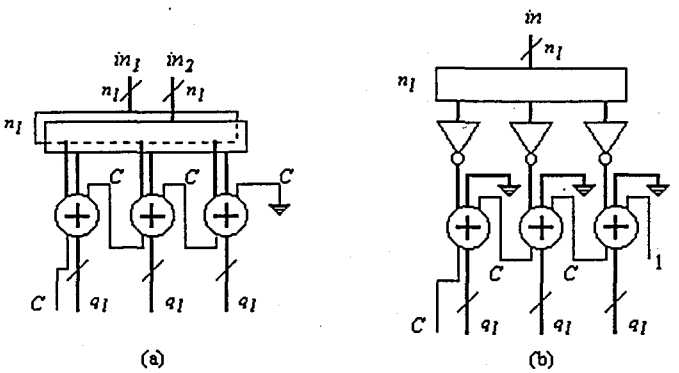


FIGURE 4.5. Realization of node n using q_l-length non-pipelined adders for n is (a) an adder, (b) a negator if $3d_q + d_r < clock$. Note that $i = j = 0$ in this realization, i.e. no selection is done up to that module.

like this:

1. There exist $n_{l_r}.\varsigma$ bits of latches and κ number of q at the first cycle.

2. For all other cycles, there exists κ number of q, and $\varsigma.(n_{l_r} - i\kappa q_l)$ bits of latches for the inputs, and $n_l + 1 - (n_{l_r} - i\kappa q_l)$ bits of latches for the outputs. Since there exist $\left\lceil \frac{m}{\kappa} \right\rceil - 1$ such cycles, then the total number of latches for these cycles are

$$\begin{aligned} &\textstyle\sum_{i=1}^{\left\lceil \frac{m}{\kappa} \right\rceil - 2} \varsigma.(n_{l_r} - i\kappa q_l) + n_l + 1 - (n_{l_r} - i\kappa q_l) \\ &= \left(\left\lceil \frac{m}{\kappa} \right\rceil - 1\right)\left[\frac{(\varsigma-1)}{2}\left(2n_{l_r} - \left\lceil \frac{m}{\kappa} \right\rceil \kappa q_l\right) + n_l + 1\right]. \end{aligned} \tag{4.6}$$

This phenomenon for an adder is illustrated in FIGURE 4.6. In this figure, $m = 5$ and $\kappa = 2$. There exists a previously allocated adder of q_s length. The shaded blocks stand for previously allocated blocks. A negator can also be implemented in the similar way.

The delay of this architecture is calculated as

$$d_{\lambda q} = \left\lceil \frac{m}{\kappa} \right\rceil clock \tag{4.7}$$

The area (or power) of both states, $a_{\lambda q}$ (or $p_{\lambda q}$) can be calculated by multiplying the number of latches in the architecture by a_λ (or p_λ) and adding a_q (or p_q).

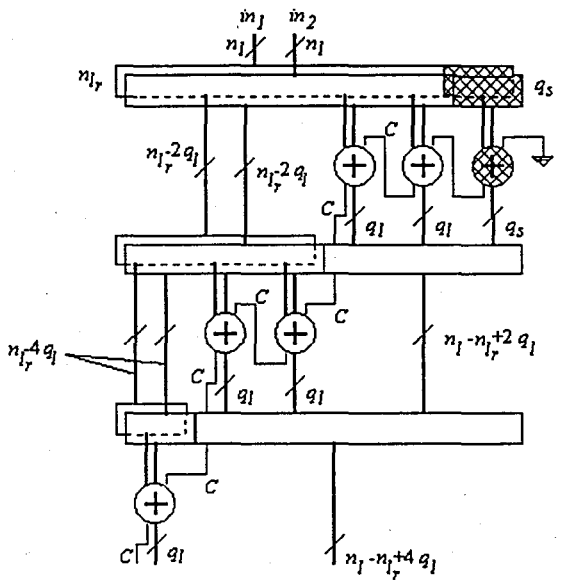


FIGURE 4.6. Realization of node n using q_l-length non-pipelined adders for n is (a) an adder, (b) a negator if $5d_q + d_r > clock$. Note that $i = 0$ and $j = 1$ in this realization, i.e. a module of length q_s has already been selected. Previous allocations are identified by shaded blocks.

4.1.3.2. Case 2: Module q is a pipelined adder. Since q is pipelined, then the system can be realized using $m.q_{pipeline}$ clock periods yielding a delay of

$$d_{\lambda q} = m.q_{pipeline} clock. \tag{4.8}$$

The number of latches in this system is calculated as follows:

1. For all time slots during the computation of κ'th q, there exist $q_{pipeline}$ times $\{n_l - [n_{l_r} - (\kappa - 1)\, q_l]\}$ bits of latches for the outputs.

2. For all time slots during the computation of κ'th q, if $n_{l_r} < \kappa q_l$ there exist no latches for the inputs. Otherwise there exist $[\varsigma.(n_{l_r} - \kappa q_l)]\, q_{pipeline}$ bits of latches for the inputs.

This phenomenon is shown in FIGURE 4.7. In this figure, $m = 3$. The shaded block stands for a previous allocation. A negator can also be implemented in a similar way. However, if $i = 0$, the inverters must exist at a separate time slot, implying that

$$d_{\lambda q} = (m.q_{pipeline} + 1)\, clock. \tag{4.9}$$

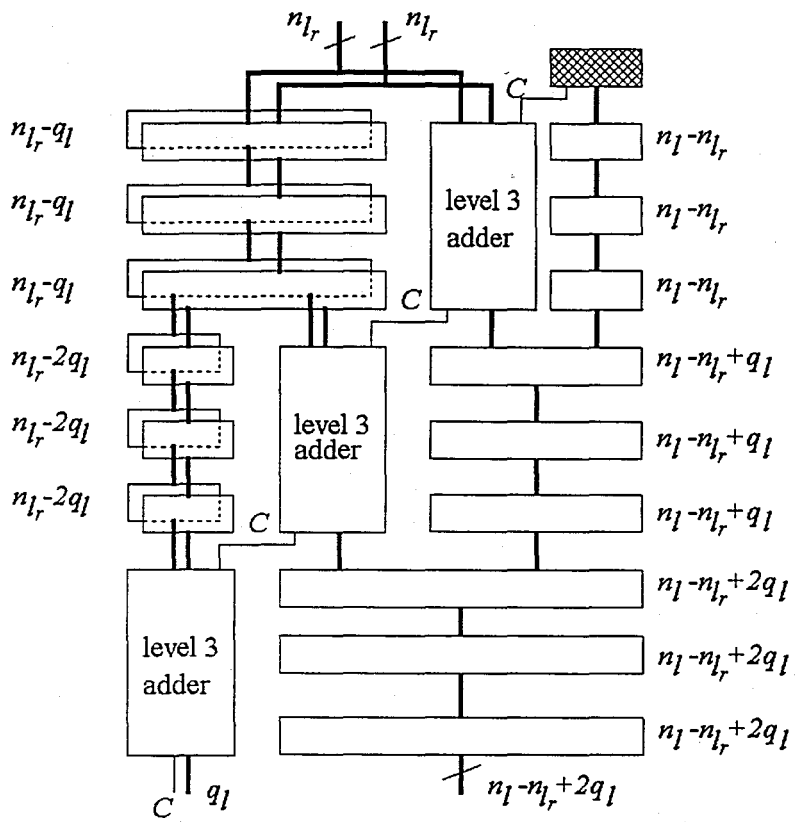


FIGURE 4.7. Realization of node n using q_l-length three-level pipelined adders for n is an adder. Note that $i = j \neq 0$ in this realization, i.e. modules of $n - b$ length q_s have already been selected in previous cycles. Previous allocations are identified by shaded blocks.

The area (or power) of both states, $a_{\lambda q}$ (or $p_{\lambda q}$) can be calculated by multiplying the number of latches in the architecture by a_λ (or p_λ) and adding a_q (or p_q).

After mentioning about the realization part for two cases, the final comment on the algorithm shown in FIGURE 4.4 is that the *check* function removes the upper bounds if the picked modules cause infeasibility so as to find *at least* a solution with the given libraries, because the nodal upper bounds are not strict. The run-time of this algorithm is determined by the dimension of libraries and the wordlength of node n, i.e. $O(|\Lambda||Q|n_l)$.

The subroutines in FIGURE 4.8 and FIGURE 4.9 are quite straightforward. The only thing that needs explanation is the word $NULL$, meaning that the required parameter is not implemented. For example $M_{ij}(\lambda, q, l) = (\lambda^*, NULL, n_{l_r})$ means that no AU is placed in (i, j)'th slot.

```
NOT_PIPED (M, i, j, λ*, q*, ς, n_{l_r}) {
    M_ij (λ, q, l) = (λ*ς, q*, q_l*);
    if (j ≠ 0) {
        calculate delay, d in i'th pipeline level;
        if (d > clock){
            M_ij (λ, q, l) = (λ*ς, NULL, n_{l_r});
            d̄_n = d̄_n − clock;
            i = i + 1;
            j = 0;
            M_ij (λ, q, l) = (λ*, NULL, n_l − n_{l_r} + 1);
            j = j + 1;
            M_ij (λ, q, l) = (λ*ς, q*, q_l*);
            }
        j = j + 1 ;
        }
    }
```

FIGURE 4.8. Algorithm for placement of a non-pipelined adder-based module.

```
PIPED(M, i, j, λ*, q*, ς, n_{l_r}) {
    if (there exists a q in i'th slot or (i = 0 and ς = 1)) {
        M_ij (λ, q, l) = (λ*ς, NULL, n_{l_r});
        d̄_n = d̄_n − clock;
        i = i + 1;
        }
    if (i > 0)
        if (j = 0)
            for (m = [i, q*_pipeline + i]) M_mj (λ, q, l) = (λ*, NULL, n_l − n_{l_r});
        else
            for (m = [i + 1, q*_pipeline + i]) M_mj (λ, q, l) = (λ*, NULL, n_l − n_{l_r});
    j = j + 1;
    M_ij (λ, q, l) = (NULL, q*, q_l*);
    d̄_n = d̄_n − q*_pipeline clock;
    if (n_{l_r} > q_l*) {
        j = j + 1 ;
        for (m = [i, q*_pipeline + i]) M_mj (λ, q, l) = (λ*ς, NULL, n_{l_r} − q_l*);
        i = i + q*_pipeline;
        j = 0;
        M_ij (λ, q, l) = (λ*, NULL, n_l − n_{l_r} + q_l*);
        j = 1;
        }
    }
```

FIGURE 4.9. Algorithm for placement of pipelined adder-based modules

4.2. Scheduling

Scheduling is done by combining as-soon-as-possible (ASAP), as-late-as-possible (ALAP) and table search methods. Before starting scheduling operation, output and input rates of each node and operation order of each node in a fold (Subsection 4.2.1.) is determined. Then scheduling tables are formed for all folds and nodes which do not belong to a fold (Subsection 4.2.2.). For the scheduling of FIR based nodes, the target architecture of FIGURE 1.8 is used as explained in Subsection 4.2.3. Scheduling of other nodes are rather simple as explained in Subsection 4.2.4. The system scheduling is completed by linking the tables (Subsection 4.2.5.).

4.2.1. Input-Output Rate Determination

The DFG file itself is used to determine the rates. Input and output nodes are designated as *sources* and *sinks* respectively. A node's input nodes are called *ancestors* and output nodes are called *descendants*. For example, in the DFG given in FIGURE 4.10, x is the source, y is the sink and the ancestor of node $wvd1$ is x and its descendant is y. Note that this figure is the DFG form of FIGURE 2.4 with edge weights standing for delay elements. Also the nodes are folded and the number of outputs is reduced to one by using a switch of period 4.

Rate of a source, ρ_{source}, is determined as

$$\rho_{source} = \frac{\text{clock frequency of the system}}{\text{frequency of the source}}. \tag{4.10}$$

The output rate of a node n in node set N, ρ_{oum_n}, is dependent on the output rates and switching rates of its ancestors, $\rho_{switch_{an}}$, and the interpolation, I_n, and decimation, D_n, rates of the node n as follows:

$$\rho_{oum_n} = \frac{\rho_{oum_a}}{\rho_{switch_{an}}} \frac{D_n}{I_n} \tag{4.11}$$

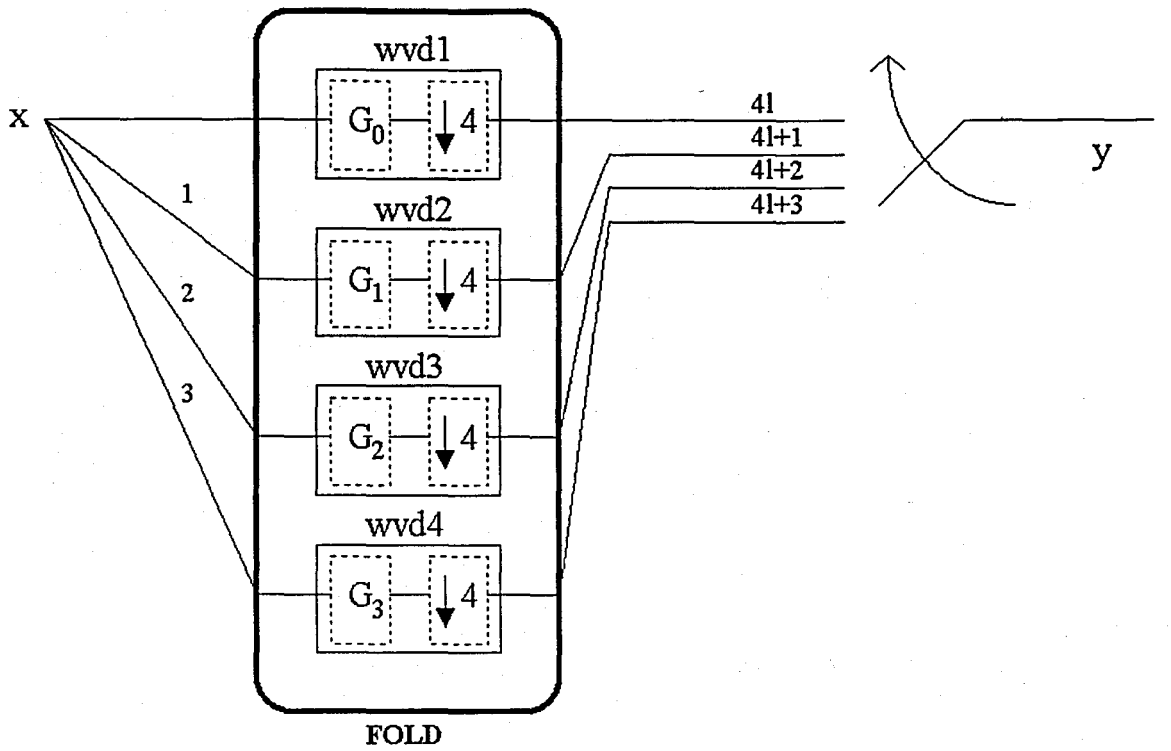


FIGURE 4.10. DFG of a four-band wavelet analysis structure using folding.

where subscript a stands for a node in ancestor node set of node n, i.e. A_n. Note that there can be more than one ancestor in A_n. Then the ratio $\frac{\rho_{oum_a}}{\rho_{switch_a}}$ must be the same for all a in A_n to produce a unique result for ρ_{oum_n}.

After calculating output rates, the nodes existing in a fold must undergo a start time analysis process to guarantee picking at most one and only one node at each clock as shown in FIGURE 4.11. In this process, the symbol N_f is the node set of fold f in fold set F, and the symbol θ_n is the start time of n'th node. Note that there are two undesired exits, i.e. ERROR!!!. This means that nodes in fold f have rates such that they cannot exist in a fold, i.e. there is no time slot to start the operation of a node in N_f.

Another important thing is the determination of start time for handling each ancestor. This is extremely important if the nodes with different ancestors exist in the same fold as shown in FIGURE 4.12. Note that this figure is the DFG form of FIGURE 2.3. Also the G and H nodes are separately folded. The algorithm is presented in FIGURE 4.13. The symbol A_{N_f} is the ancestor node set of fold f in fold

```
OPERATION_START_TIME_ANALYSIS (N_f){
        sort nodes in N_f according to their ρ_out_n;
        form an integer array M of length |M|;
        for each n ∈ N_f
            if there exists m ∈ M such that m is empty and m < ρ_out_n{
                m is full;
                θ_n = m;
                t = m + ρ_out_n;
                while (t < |M|) if (t is empty){
                        t is full;
                        t = t + ρ_out_n;
                        }
                else ERROR!!!
                }
            else ERROR!!!
        }
```

FIGURE 4.11. Algorithm for operation start time analysis in a fold.

set F, the symbol δ_{an} is the edge weight between a'th ancestor and n'th node, and the symbol α_a is the start time of a'th ancestor. Note that there are two undesired exits, i.e. ERROR!!!. This means that nodes in fold f have rates such that they cannot exist in a fold, i.e. there is no time slot to handle an ancestor in A_{N_f}. The same algorithm can be applied to handle ancestors at sinks. In this case, each sink will behave as a fold.

Both of these algorithms are linear-time algorithms.

Let us illustrate what is explained in this subsection with two examples: The first one is FIGURE 4.10: Assume that input frequency of x is $30MHz$ and clock frequency of the system is $30MHz$. Then $\rho_{source} = 1$, $\rho_{out_{wvd1}} = \rho_{out_{wvd2}} = \rho_{out_{wvd3}} = \rho_{out_{wvd4}} = 4$ and $\rho_{out_y} = 1$. After running the algorithms, the following results are obtained: $\theta_{wvd1} = \alpha_{wvd1} = 0$, $\theta_{wvd2} = \alpha_{wvd2} = 1$, $\theta_{wvd3} = \alpha_{wvd3} = 2$, $\theta_{wvd4} = \alpha_{wvd4} = 3$. The second example is FIGURE 4.12: Assume that input frequency of x is $15MHz$ and clock frequency of the system is $30MHz$. Then $\rho_{source} = 2$, $\rho_{out_{wvdl1}} = \rho_{out_{wvdh1}} = 4$, $\rho_{out_{wvdl2}} = \rho_{out_{wvdh2}} = 8$, $\rho_{out_{wvdl3}} = \rho_{out_{wvdh3}} = 16$, and $\rho_{out_y} = 2$. After running the algorithms, the following results are obtained: $\theta_{wvdl1} = \alpha_{wvdl1} = \theta_{wvdh1} = \alpha_{wvdh1} = 0$, $\theta_{wvdl2} = \alpha_{wvdl2} = \theta_{wvdh2} = \alpha_{wvdh2} = 1$, $\theta_{wvdl3} = \theta_{wvdh3} = 2$ and $\alpha_{wvdl3} = \alpha_{wvdh3} = 3$.

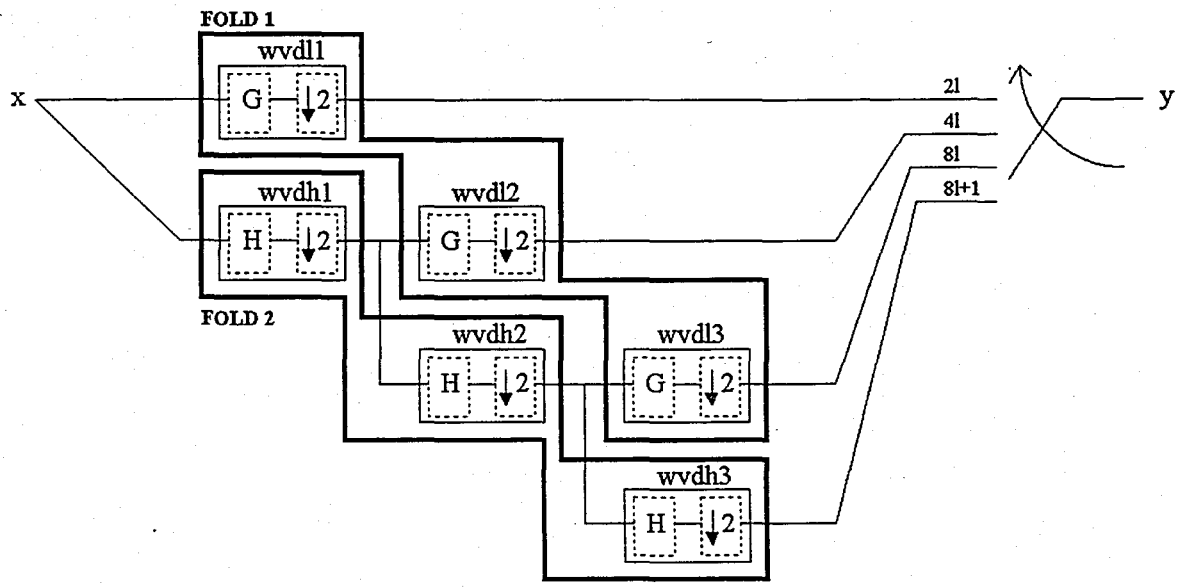


FIGURE 4.12. DFG of a two-band three-level wavelet analysis structure using folding.

ANCESTOR_PROCESS_START_TIME_ANALYSIS (N_f)**{**

sort all ancestors a in A_{N_f} according to their ρ_{out_a};
form an integer array M of length $|M|$;
for each $a \in A_{N_f}$
if there exists $m \in M$ **such that**
m is empty **or** m is preoccupied by the same node as a **{**
m is full;
$\alpha_a = m$;
$t = m + \rho_{out_n}$;
while $(t < |M|)$ **if** (t is empty **or** m is preoccupied by the same node as a)**{**
t is full;
$t = t + \rho_{out_n}$;
}
else ERROR!!!
}
else ERROR!!!
}

FIGURE 4.13. Algorithm for start time determination of ancestors in a fold.

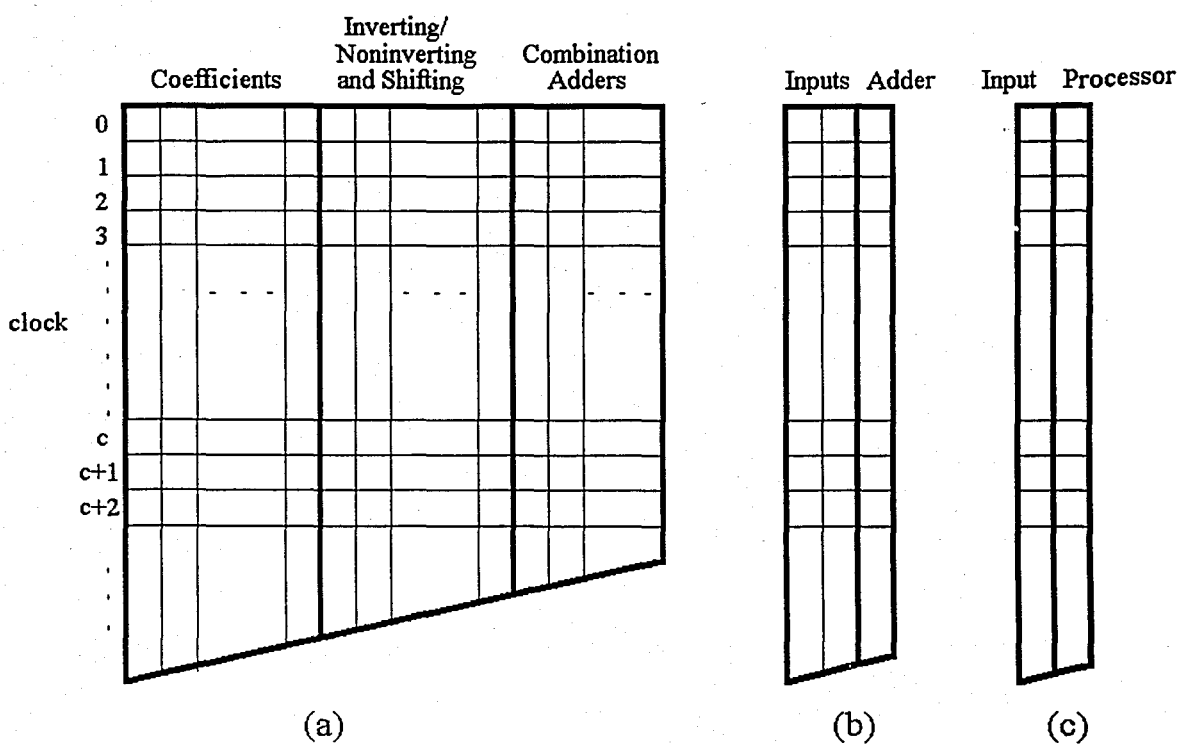


FIGURE 4.14. Scheduling table types for (a) FIR-based nodes, (b) two-input nodes like adders, (c) single-input nodes like negators.

4.2.2. Scheduling Table Formation

The set of scheduling tables is Υ. Scheduling tables are formed only for folded nodes and the nodes that do not belong to a fold, i.e. the number of scheduling tables, $|\Upsilon|$, is

$$|\Upsilon| = |F| + \left| N \setminus \left\{ \bigcup_{f \in F} N_f \right\} \right| \tag{4.12}$$

Each row of the scheduling table is the clock time. The column number is determined according to the type of node as shown in FIGURE 4.14. As a result their scheduling process is different. Note that FIR-based nodes or of type single-input-single-output, their scheduling table is quite different than the other nodes' of the same type, because they are super-nodes, i.e. contain more than one operation. Each table element ξ_{ij} is a data structure which is composed of fields shown in TABLE 4.1. Here, subscripts i and j stand for the row and column information of the entry, respectively.

TABLE 4.1 Fields of an entry ξ in the scheduling table of an FIR-based node or fold.

Field	*Explanation*
soa	Start of computation
eoa	End of computation
eol	End of life

For an FIR-based node or fold, each block shown in FIGURE 1.8 requires a different type of scheduling. Basically, scheduling of Scaling Adders and Combination Adders blocks are done using ASAP scheduling [2]. Scheduling of Inverting/Noninverting and Shifting block is done by using table search and a modified version of ALAP scheduling. There are three types of super-columns for coefficients, Inverting/Noninverting and Shifting block and Combination Adders block as shown in FIGURE 4.14. Note that the column number separated for the coefficients is not necessarily equal to the number of adders in Scaling Adders block. The ASAP scheduling of Scaling Adder block determines the clock time for generating multiplications with coefficients. Therefore, the number of columns in the Coefficients super-column is the number of taps in the FIR-based node or fold, i.e. K. There must be as many columns in the Inverting/Noninverting and Shifting super-column as the number of taps in the FIR-based node or fold, i.e. K again, because that block is used for obtaining the real values for FIR-based computations. Finally the number of columns in the Combination Adders super-column is equal to $K-1$. Therefore, total number of columns in the table of FIGURE 4.14(a) is $3K-1$.

For a node or fold with two inputs, the table shown in FIGURE 4.14(b) is used. There is one super-column for the inputs. The other column is only for the adder. Therefore, total number of columns in the table of FIGURE 4.14(b) is 3.

For a node or fold with a single inputs, the table shown in FIGURE 4.14(c) is used. There are two columns, one for the adder, and one for the process delay. Therefore, total number of columns in the table of FIGURE 4.14(c) is 2.

4.2.3. Scheduling of FIR-based Nodes

For each FIR-based node or fold the following steps are taken:

Scheduling of Scaling Adders in FIGURE 1.8 is done using ASAP scheduling. Therefore the required time, ς_k, for coefficient multiplication of k'th tap can be easily known after this process.

Using the rates and start times that are calculated in Subsection 4.2.1., the Coefficients and Inverting/Noninverting and Shifting super-columns are filled for each FIR-based node n and each of its ancestor a as follows: At first, the location of the entry to be filled is searched. The row and column, i and j, of an entry in the Coefficients super-column are calculated as

$$i = \lceil \theta_n + m.\rho_{out_n} \rceil \tag{4.13}$$

$$j = k, \text{ if and only if } \frac{m.D_n - k - \delta_{an}}{I_n} \text{ is an integer.} \tag{4.14}$$

where m is a natural number, k is the tap order in the FIR-based node and δ_{an} is the edge weight, i.e. the delay element between the ancestor a and node n. Then, each entry ξ_{ij} for k'th tap in Coefficients super-column is filled as follows:

$$\begin{aligned} \left(\xi_{ij_{soa_k}}\right)_C &= \left\lceil \frac{(m.D_n - k - \delta_{an}) \bmod K}{I_n} \frac{\rho_{out_a}}{\rho_{switch_{an}}} \right\rceil + \alpha_n \\ \left(\xi_{ij_{eoa_k}}\right)_C &= \left(\xi_{ij_{soa_k}}\right)_C + d_k \end{aligned} \tag{4.15}$$

where subscript C stands for Coefficients super-column, and d_k is the delay required to realize the constant multiplication at the k'th tap. Each entry ξ_{ij} for k'th tap in Inverting/Noninverting and Shifting super-column is filled as follows:

$$\begin{aligned} \left(\xi_{ij_{soa_k}}\right)_{I\_NI} &= \left(\xi_{ij_{eoa_k}}\right)_C \\ \left(\xi_{ij_{eoa_k}}\right)_{I\_NI} &= \left(\xi_{ij_{soa_k}}\right)_{I\_NI} + d_k \end{aligned} \tag{4.16}$$

where d_k is the delay of k'th negator if the input scaled by the Scaling Adders must be negated. Otherwise, d_k is 0. For example, the coefficient .0664062 appears as it is at the 7'th tap and its negative form appears at the 0'th tap of node G_0 in TABLE 3.2.

This means that d_0 is the delay of zeroth negator as calculated in Section 4.1., and d_7 is 0. The subscript I_NI stands for Inverting/Noninverting and Shifting super-column.

An important procedure starts after filling out Coefficient and Inverting/Noninverting and Shifting super-columns of the table τ for all nodes in the fold: The aim is maximizing the utilization of Shift Registers block in FIGURE 1.8. Since the Combination Additions block is ASAP scheduled, the Inverting/Noninverting and Shifting block must be scheduled ALAP to realize this aim. Therefore, $(\xi_{i_{eoa}})_{I\_NI}$ which is the end of computation of Inverting/Noninverting and Shifting block is calculated as follows:

$$(\xi_{i_{eoa}})_{I\_NI} = \max_{0\leq k<K} \left(\frac{m.D_n - k - \delta_{an}}{I_n} \frac{\rho_{out_a}}{\rho_{switch_{an}}} + \left(\xi_{ij_{eoa_k}}\right)_{I\_NI} \right) \bmod \left(\frac{\rho_{out_a}}{\rho_{switch_{an}}} \frac{K}{I_n} \right) \tag{4.17}$$

However, this is not usually sufficient because $(\xi_{i_{eoa}})_{I\_NI}$ for all i must be unique to inhibit bus conflict. Therefore the algorithm shown in FIGURE 4.15 is proposed. The function LCM is the well-known least-common-multiplication algorithm. After determining $(\xi_{i_{eoa}})_{I\_NI}$, the following computations are carried out as follows:

$$\begin{aligned} \left(\xi_{ij_{soa_k}}\right)_{I\_NI} &= \left(\xi_{ij_{soa_k}}\right)_{I\_NI} + (\xi_{i_{eoa}})_{I\_NI} - \left(\xi_{ij_{eoa_k}}\right)_{I\_NI} && ,\forall i,j,k \\ \left(\xi_{ij_{eoa_k}}\right)_{I\_NI} &= \left(\xi_{ij_{eol_k}}\right)_{I\_NI} = (\xi_{i_{eoa}})_{I\_NI} && ,\forall i,j,k \\ \left(\xi_{ij_{eol_k}}\right)_{C} &= \left(\xi_{ij_{soa_k}}\right)_{I\_NI} && ,\forall i,j,k \end{aligned} \tag{4.18}$$

Now that the computations in Inverting/Noninverting and Shifting block finish at the same clock cycle, then Combination Adders block can be scheduled using ASAP.

To finish scheduling of FIR-based tables, the number and switching of Shift Registers in FIGURE 1.8 block must be done. The number of registers is calculated using the algorithm in [81]. This algorithm requires the period and the variables with their birth times and life times.

The period of the table τ, π_τ, can be determined by finding a group of successive rows in the Coefficient super-column: π_τ is initially set to 1, i.e. the second row of τ. Starting from π_τ, each row is searched whether each entry on the row is generated by the same ancestor and coefficient as the initial row. The search stops if a matching row is found. Then the rows found between the initial and the $(\pi_\tau - 1)$'th row are checked in the intervals $[\pi_\tau, 2\pi_\tau - 1]$, $[\pi_\tau, 2\pi_\tau - 1]$, ... whether they really match. In case of

SCHEDULE_INVERTING_NONINVERTING (τ) {
$|M| = LCM\left(\frac{\rho_{out_a}}{\rho_{switch_{an}}}, \forall a \in A_{N_f}\right)$;
form an integer array M of length $|M|$;
for all$i \in M$, $m_i = 0$;
for all $i \in M$ if($m_i = (\xi_{i_{soa}})_{I\_NI} \bmod |M|$)
$m_i = m_i + 1$;
for all $i \in M$ if $(m_i > 1)$\{
find a $(\xi_{i_{soa}})_{I\_NI}$ such that $m_i = (\xi_{i_{soa}})_{I\_NI} \bmod |M|$;
find a location t in M such that $m_t = 0$;
if $(t > i)$ $(\xi_{i_{soa}})_{I\_NI} = (\xi_{i_{soa}})_{I\_NI} + t - i$;
else $(\xi_{i_{soa}})_{I\_NI} = (\xi_{i_{soa}})_{I\_NI} + t - i + |M|$;
$m_i = m_i - 1$;
$m_t = 1$;
}

FIGURE 4.15. Algorithm for scheduling of Inverting/Noninverting and Shifting block.

a mismatch , the whole process starts again.

Throughout the table within the period π_τ and Coefficients super-column, all entries using the same ancestor and coefficient with the equivalent $\left(\xi_{ij_{soa_k}}\right)_C$ are grouped to form the variable set Ψ and each group is treated as a variable, φ. Life time and birth time of each variable, i.e. φ_{life} and φ_{birth} respectively, are determined using all elements $g \in \varphi$ as

$$\begin{aligned} \varphi_{life} &= \max_{g\in\varphi}\left[\left(\xi_{ij_{eol_k}}\right)_C\right]_g - \left[\left(\xi_{ij_{soa_k}}\right)_C\right]_g \quad ,\forall\varphi \in \Psi, \\ \varphi_{birth} &= \left[\left(\xi_{ij_{soa_k}}\right)_C\right]_g \bmod \pi_\tau \qquad\qquad ,\forall\varphi \in \Psi. \end{aligned} \tag{4.19}$$

However, the algorithm for register number calculation of [81] is modified in our HLS tool: The original algorithm assumes that each variable must occupy at least a register. In fact, this is not necessary because if a variable's life time is 0, i.e. $\varphi_{life} = 0$, then it need not occupy a register. Therefore, during register number calculation, the variables with 0 life time are omitted. This modification reduces the number of registers. The algorithm forms a circular life time chart, i.e. a segmented wheel, of period π_τ and each variable is assumed as a piece of wire of φ_{life} length to be wound around the wheel starting from φ_{birth}. After winding all wires, the number of wires on each wheel segment is counted and the maximum number is taken to be the register

number.

After calculating the number of registers, the scheduling of variables in registers are done by using a modified version of the forward-backward register allocation algorithm [82]. The original algorithm suffers from the fact explained in the preceding paragraph. In addition it assumes that each variable lives within a period, i.e. $\varphi_{life} + \varphi_{birth} \leq \pi_\tau$. Finally, the algorithm forward allocates each variable until it is dead, otherwise the variable must reach the last register. This produces incomplete scheduling for variables with $\varphi_{life} + \varphi_{birth} \geq \pi_\tau$. Therefore, the algorithm is modified as follows: The allocation of each variable is done such that it can cover maximum number of consecutive forward registers to minimize the number of switches meaning that allocation of a variable does not necessarily end at the last register even if it is not dead. It can handle variables with $\varphi_{life} + \varphi_{birth} \geq \pi_\tau$. The algorithm forward allocates all variables with non zero life times within a period π_τ in a cyclic manner. If there exists variables with incomplete allocation, it searches for available registers starting from the last register (i.e. backward search) and forward allocation takes place. This process is recursive and finishes after all variables are allocated.

After obtaining the register numbers in Shift Registers block, the type of register, r_τ is determined by library search in register array R:

$$r_\tau = r_S = r_{CA} = \min_{r \in R} [(s_a a_r + s_p p_r) \text{ such that } d_r < clock] \tag{4.20}$$

The wordlength of r_τ is determined by the maximum length adder and negator in the Scaling Adder block. Note that there are also registers in Scaling Adders and Combination Adders blocks due to ASAP scheduling, r_S and r_{CA} respectively. Their wordlengths can be determined by the wordlength of the ancestor adder or negator to an adder or negator.

4.2.4. Scheduling of Other Nodes

Scheduling of other nodes is similar to FIR-based nodes. The super-columns, Inputs and Input in FIGURE 4.14(b) and (c) are treated as Coefficients super-column

in FIGURE 4.14(a) with one exception which is that the coefficient of each entry is only 1. The super-columns, Adder and Processor in FIGURE 4.14(b) and (c) are treated as Inverting/Noninverting and Shift super-column in FIGURE 4.14(a). Then register count calculation and register allocation is done as explained on page 95.

4.2.5. Scheduling Between Tables

The scheduling of the system is completed after linking the tables of all nodes. This is done by using the periods of the ancestor nodes' table, π_{τ_a}, and the node's table, π_{τ_n}, and the input and output entries of tables, i.e. the first super-columns and the last columns. The aim is generating period convertors which consist of registers and switches only. The algorithms explained on page 95 are used to realized these data convertors.

4.3. System Area, Delay and Power Calculation

After picking modules, scheduling the nodes by using tables, the delay, area and power of each node is calculated. If they do not satisfy the constraints, the register costs of tables are subtracted from the system constraints and module selection and scheduling processes are repeated until no further recovery is obtained or constraints are satisfied. This means that a result is obtained even though it may be infeasible.

5. A DESIGN EXAMPLE

Let us design the system shown in FIGURE 4.10 with the coefficient set given in [51]. In this figure, each G is an FIR filter. The output error for all paths is 0.45%. All coefficients undergo an unscaled quantization. Also assume that the input signal x consists of five bits for the integer part where the most significant bit indicates the sign of the number. Assume that maximum area, latency and power are $50mm^2$, $1\mu s$, and $30mW$. The clock frequency $100MHz$ and the system will be realized using ES2-0.8μm technology. The input file of this system is shown in FIGURE A. 1.

After running the quantizer explained in [78], it is found that each node in the fold must be quantized using 9 bits, including the sign bit, and the output error is to 0.34%. The quantized system coefficients are tabulated in TABLE 3.2. Then the DFG is transformed as shown in FIGURE 5.1 without edge delay information.

When CSD representation is used instead of ordinary binary representation, the quantizing wordlength drops to 8 bits. Now, the node shown by **F** can undergo the process as described in [77] for multiplierless realization of coefficient multiplications. Note that the two-terms and replicas obtained by this process constitute the Scaling Adders of FIGURE 1.8 as shown in FIGURE 5.2.

The Inverting/Noninverting and Shifting, and Combination Adders blocks are the other important parts in enhanced node set $\aleph$ and they are shown in FIGURE 5.3.

After the enhanced node set $\aleph$ is obtained, the wordlength of operators and constraints on each node are determined. Using the library sets given in FIGURE B. 2 and FIGURE B. 1, the selected modules are shown in FIGURE 5.4 and FIGURE

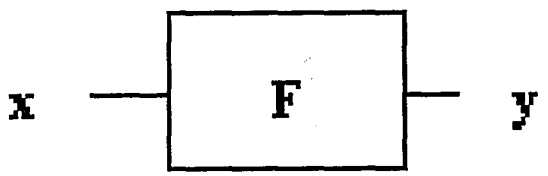


FIGURE 5.1. New DFG formed after replacing fold **F** in place of **FOLD** in FIGURE 4.10.

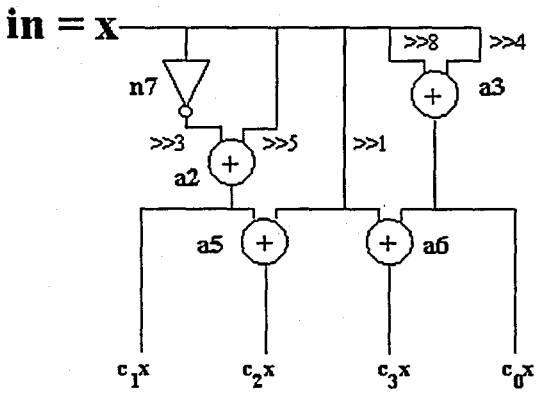


FIGURE 5.2. Scaling Adders block of node F.

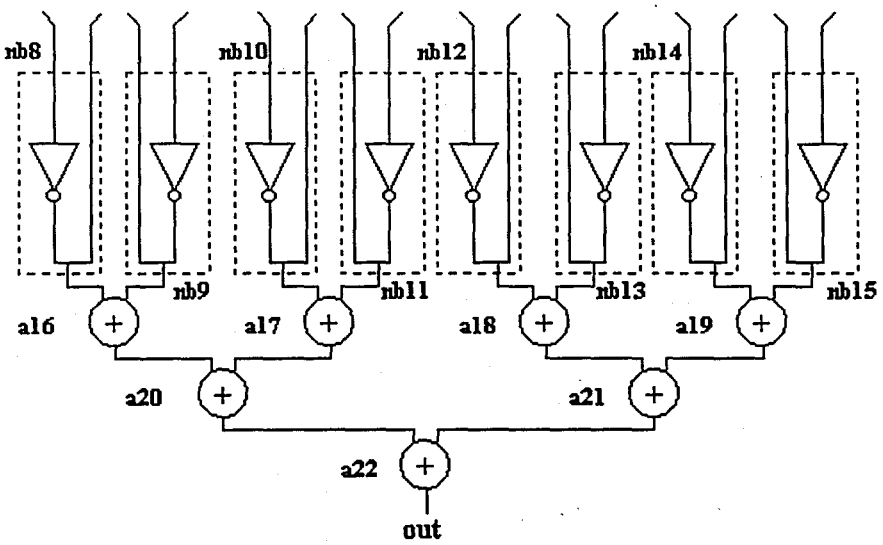


FIGURE 5.3. Inverting/Noninverting and Shifting, and Combination Adders blocks of node F.

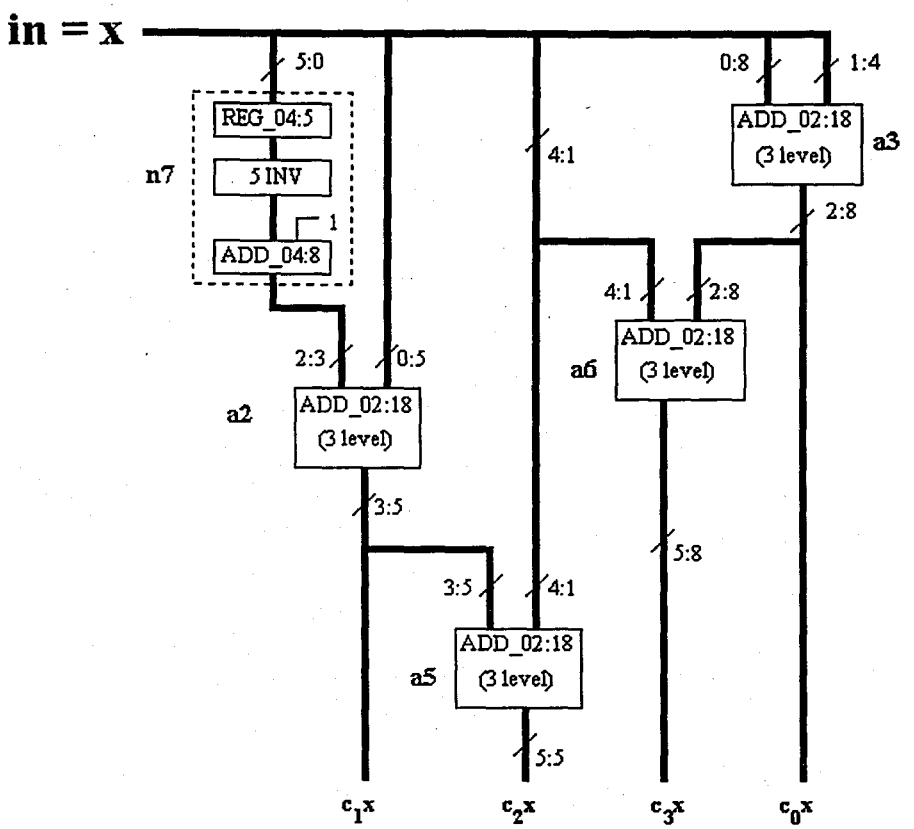


FIGURE 5.4. Scaling Adders block of node F after module selection.

5.5 with their appropriate wordlengths. After running the scheduler, we finally obtain the layout shown in FIGURE 5.6. Its structural and behavioral output files are shown in FIGURE C. 4 and FIGURE C. 1 respectively. The timing of the circuit is given in TABLE 5.1, TABLE 5.2, TABLE 5.3, TABLE 5.4 and TABLE 5.5. In all these tables, C stands for the absolute clock time. The **delay** column in the last table represents the propagation delay in terms of clock cycle. The **Exp.** column in the same table represents the instance at the output of the node. For example, $wvd1_1$ is the first instance at the output of the node wvd1.

Upon investigation of scheduling tables, it is evident that at each clock cycle, different instances of the input stream are processed at different levels of the operators. Therefore the initial claim that the tool produces fully pipelined architectures is proved. Additionally, note that each clock cycle is fully utilized. This means that even though there are redundant cycles in the original DFG due to decimation, this redundancy is removed by our tool by reorganizing the scaling process and appropriately scheduling

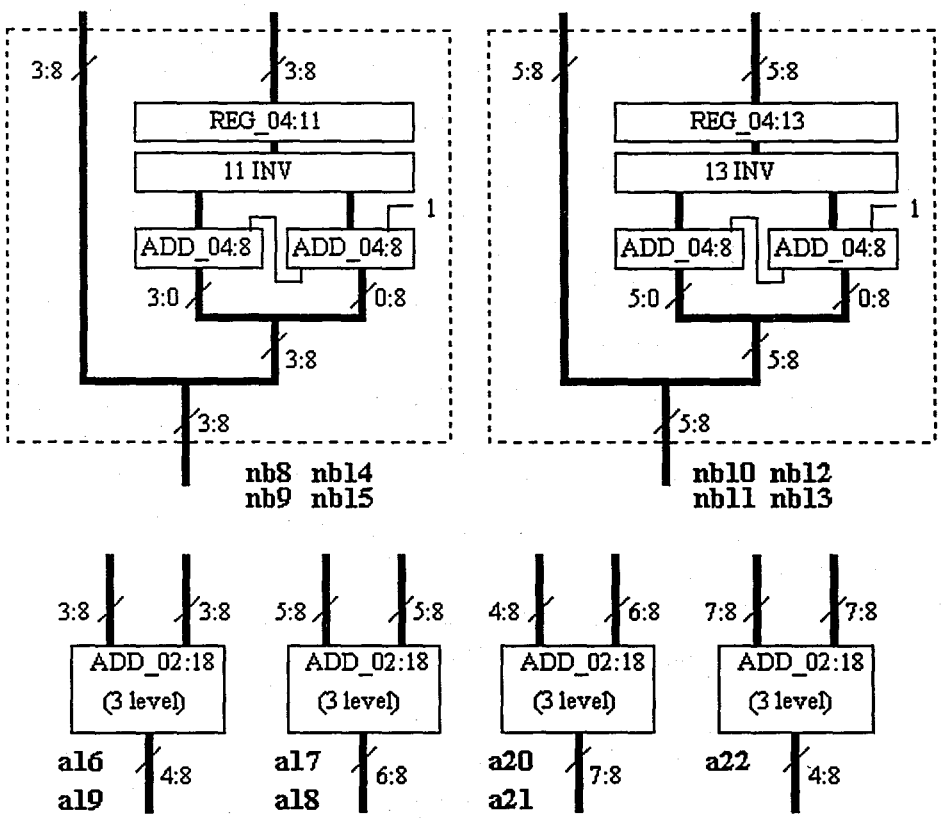


FIGURE 5.5. Inverting/Noninverting and Shifting, and Combination Adders blocks of node **F** after module selection.

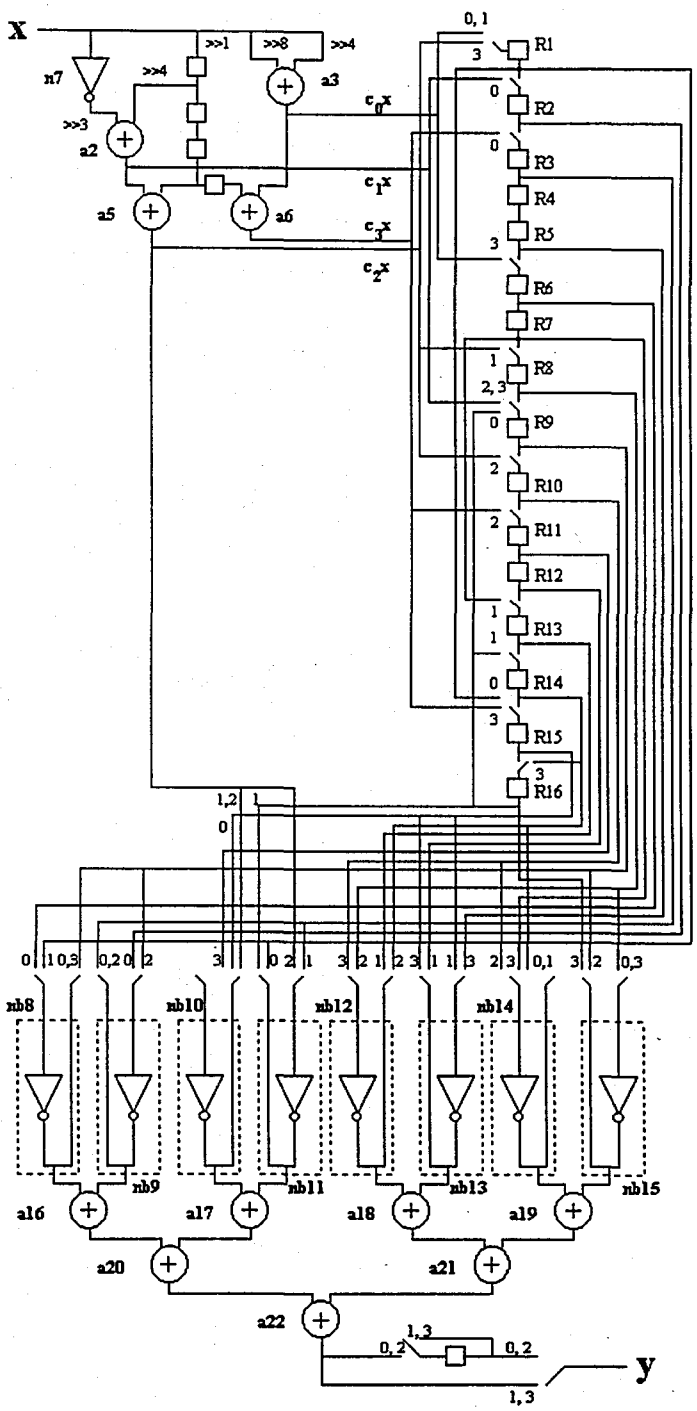


FIGURE 5.6. Complete layout of analysis part of the four-band wavelet transform.

TABLE 5.1 Timing table of the design example (Table 1 of 5)

C	*Scaling Adders*									
	x	**a2** (3)	**a3** (3)	**a5** (3)	**a6** (3)	**n7** (1)	**Rs1**	**Rs2**	**Rs3**	**Rs4**
0	8									
1	ff					f8	8			
2	f					1	ff	8		
3	3		8.8			f1	f	ff	8	
4	fc	ff.4	ff.ef			fd	3	f	ff	8
5	f5	0.18	0.ff			4	fc	3	f	ff
6	fe	ff.98	0.33		4.88	b	f5	fc	3	f
7	ff	ff.b8	ff.bc	3.4	ff.6f	2	fe	f5	fc	3
8	f7	0.6	ff.45	ff.98	8.7f	1	ff	fe	f5	fc
9	f	1.08	ff.de	6.18	1.b3	9	f7	ff	fe	f5
10	6	0.3	ff.ef	1.38	fd.bc	f1	f	f7	ff	fe
11	f2	0.18	ff.67	fe.6	f9.c5	fa	6	f	f7	ff
12	7	0.d8	0.ff	fb.88	fe.de	e	f2	6	f	f7
13	fb	fe.98	0.66	ff.3	ff.6f	f9	7	f2	6	f
14		ff.7	ff.12	ff.98	fa.e7	5	fb	7	f2	6
15		1.5	0.77	fc.58	8.7f			fb	7	f2
16		ff.58	ff.ab	6.18	3.66				fb	7
17		0.78		2.7	f8.12					fb
18				fa.5	3.f7					
19				2.d8	fd.2b					
20				fd.f8						

TABLE 5.2 Timing table of the design example (Table 2 of 5)

C	*Registers*									
	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**
4						8.8				
5	ff.ef	ff.4					8.8			
6	0.ff	ff.ef	ff.4							
7		0.ff	ff.ef	ff.4					fe.98	
8	3.4		0.ff	ff.ef	ff.4	ff.bc			ff.b8	fe.98
9	ff.45	0.6	8.7f	0.ff	ff.ef	ff.4	ff.bc		8.8	ff.b8
10	ff.de	ff.45	0.6	8.7f	0.ff	ff.ef	ff.4	6.18		8.8
11		ff.de	ff.45	0.6	8.7f	0.ff	ff.ef	ff.4	0.3	1.38
12	fe.6		ff.de	ff.45	0.6	ff.67	0.ff	ff.ef	0.18	0.3
13	0.ff	0.d8	fe.de	ff.de	ff.45	0.6	ff.67	0.ff	ff.bc	0.18
14	0.66	0.ff	0.d8	fe.de	ff.de	ff.45	0.6	ff.3	0.ff	ff.bc
15		0.66	0.ff	0.d8	fe.de	ff.de	ff.45	0.6	ff.7	ff.98
16	fc.58		0.66	0.ff	0.d8	0.77	ff.de	ff.45	1.5	ff.7
17	ff.ab	ff.58	3.66	0.66	0.ff	0.d8	0.77	ff.de	ff.67	1.5
18		ff.ab	ff.58	3.66	0.66	0.ff	0.d8	2.7	ff.de	ff.67
19			ff.ab	ff.58	3.66	0.66	0.ff	0.d8		fa.5
20	2.d8			ff.ab	ff.58		0.66	0.ff		
21					ff.ab	ff.58		0.66	0.77	
22						ff.ab	ff.58		0.66	0.77
23							ff.ab	ff.58		
24								ff.ab		

TABLE 5.3 Timing table of the design example (Table 3 of 5)

C	*Registers*					
	R11	**R12**	**R13**	**R14**	**R15**	**R16**
6			8.8			
7	4.88			8.8		
8		4.88			ff.6f	8.8
9	fe.98		4.88		3.4	ff.6f
10	ff.b8	fe.98	ff.bc	ff.6f		
11	fd.bc	ff.b8	fe.98	ff.bc	ff.6f	
12	1.38	fd.bc	ff.b8	fe.98	f9.c5	ff.bc
13	0.3	1.38	fd.bc	ff.b8	fe.6	f9.c5
14	0.18	0.3	ff.67	f9.c5	ff.b8	
15	fa.e7	0.18	0.3	ff.67	f9.c5	ff.b8
16	ff.98	fa.e7	0.18	0.3	8.7f	ff.67
17	ff.7	ff.98	fa.e7	0.18	fc.58	8.7f
18	1.5	ff.7	0.77	8.7f	0.18	
19	3.f7	1.5	ff.7	0.77	8.7f	0.18
20	fa.5	3.f7	1.5	ff.7	fd.2b	0.77
21		fa.5	3.f7	1.5	2.d8	fd.2b
22				fd.2b	1.5	
23					fd.2b	1.5

TABLE 5.4 Timing table of the design example (Table 4 of 5)

C	*Inverting/Noninverting and Shifting*							
	nb8 (0/1)	**nb9** (0/1)	**nb10** (0/1)	**nb11** (0/1)	**nb12** (0/1)	**nb13** (0/1)	**nb14** (0/1)	**nb15** (0/1)
5	f7.8							
6	0.11	ff.4						
7	fe.98	0.11	4.88					
8	ff.b8	0.ff	ff.6f	3.4				
9	0.44	0.48	6.18	ff.6f	4.88			
10	0.bb	0.6	1.38	f7.81	ff.6f	fc.c		
11	0.3	0.bb	fd.bc	fe.c8	f9.e8	ff.6f	7.8	
12	0.18	ff.de	f9.c5	fe.6	fe.c8	f7.81	0.11	0.c
13	0.99	ff.e8	ff.3	f9.c5	fd.bc	1.38	1.68	0.11
14	ff.01	0.d8	ff.98	1.22	f9.c5	1.a	0.48	0.ff
15	ff.7	ff.01	fa.e7	0.68	0.d	f9.c5	0.44	ff.b8
16	1.5	0.66	8.7f	fc.58	0.68	1.22	0.bb	ff.a
17	ff.89	fe.b	2.7	8.7f	fa.e7	ff.98	ff.d	0.bb
18	0.55	ff.58	fa.5	fc.9a	8.7f	3.a8	ff.e8	ff.de
19		0.59	3.f7	5.b	fd.9	8.7f	0.99	0.18
20			fd.2b	2.d8	5.b	fc.9a	ff.01	0.28
21				fd.2b	3.f7	fa.5	f8.9	ff.01
22					fd.2b	d.28	fe.a	0.66
23						fd.2b	ff.89	1.5
24							0.55	0.a8
25								0.55

TABLE 5.5 Timing table of the design example (Table 5 of 5)

C	*Combination Adders*							Rout	y	delay ~	Exp.
	a16 (3)	a17 (3)	a18 (3)	a19 (3)	a20 (3)	a21 (3)	a22 (3)				
15	ff.f6	f8.25	f6.49	0.d1							
16	0.81	f8.f5	fe.f4	1.79							
17	ff.d9	0.ba	fb.65	1.47							
18	fe.71	fb.4f	fa.95	ff.fc	f8.1b	f7.1a					
19	1.b6	4.d7	1.8a	0.5b	f9.76	0.6d					
20	fe.39	a.ef	fa.7f	0.8b	0.93	fc.ac					
21	ff.ad	f6.ea	c.27	ff.c6	f9.c0	fa.91	ef.35		ef.35	21	$wvd2\_1$
22					6.8d	1.e5	f9.e3				
23					9.28	fb.0a	fd.3f	f9.e3	fd.3f	21	$wvd4_0$
24					f6.97	b.ed	f4.51	f9.e3	f9.e3	23	$wvd1\_1$
25							8.72	f4.51	8.72	21	$wvd2_0$
26							4.32	f4.51	f4.51	23	$wvd3_0$
27							2.84	4.32	2.84	21	$wvd4_1$
28								4.32	4.32	23	$wvd1_0$

them at the input of the Inverting/Noninverting and Shifting block. Note that the original architecture requires two separate clocks: one for capturing the input and processing in the FIR filters and one for decimation process. On the other hand, the architecture produced by the new HLS tool requires a single clock to realize both of the processes simultaneously. This is easily done by folding FIR-based nodes, which is a brand-new feature of this newly developed tool.

The **Exp.** column in TABLE 5.5 requires additional explanation. It should be noted that an output is produced at each clock cycle as it is ordered by the user. However, a careful reader will easily see that there is a frame-shift at the output for this design. This is completely dependent on the available library elements, optimization criteria and the output switching instances determined by the user.

A newly-developed similar tool [10] uses folding in multirate systems like we do, but they fold simple operations like multipliers and adders, but we fold composite nodes like FIR-based nodes. Therefore a direct comparison cannot be done. However it is felt that their tool also minimizes redundant clock cycles. They also use a single clock to realize multirate systems. Finally the core chip area and power are found to be $2.91mm^2$ and $28.82mW$ respectively. The propagation delay is $230ns$.

6. CONCLUSION

In this thesis, a silicon compiler is developed to reduce design time for the hardware realization of FIR-based multirate DSP algorithms for bit-parallel-digit-serial architectures. This tool is developed on FIR-based multirate systems because it is proved that an FIR-based cascaded multirate system is nearly as efficient as an IIR-based cascaded system FIGURE 1.5. This is a completely novel study, because there does not exist a silicon compiler of this type at our knowledge.

Specifications of many HLS tools are investigated as shown in TABLE 1.1. All of these tools can handle multirate digital systems as well as ours, but efficiency of our system is much higher than that of the other systems as our compiler reduces the idle cycles in multirate systems by allowing folding [18], [19]. As a result a multirate system can be realized using *a single clock signal* at a much smaller area, power consumption and latency.

There is a recent study which synthesizes multirate systems by Denk and Parhi [10] which also uses folding. However, this study does not handle FIR filters as a single node which is the real improvement in our compiler. They use multipliers for constant multiplications at the taps of the FIR filters, but our compiler decomposes FIR-based nodes into multiplierless nodes, reducing the nodes' areas significantly without sacrificing from system quality. In short, their synthesis tool folds simple nodes, but our compiler folds composite nodes like FIR filters, FIR filters followed by a decimator or FIR filters following an interpolator.

Our tool has a capability of quantizing filter tap coefficients given the output error of the system. Data bus-width can be controlled by the user throughout the system between nodes. The tool also calculates the path errors. This feature reduces the design time of a design engineer, because this service is given by the tool itself.

The module selection and scheduling is a rather important process in our silicon compiler as usual. The module selection process is done by library search in a such a way that the throughput is maximized. Structural pipelining is used to realize this aim. The formation of folding sets by the user really effects the throughput of the system. A cost-optimized algorithm for selecting slow components on non-critical paths and fast

TABLE 6.1 Some experimental results for 8-bit coefficient quantization

Experiment	*Registers*	*Adders*
FIGURE 2.4 (a)	24	37
FIGURE 2.3 (a)	69	84

components on critical paths has also been developed for high-performance pipelines. A similar study has been carried out by [13]. The main difference between their and our algorithms is that their algorithm carries out selection for one pipe throughout all nodes at each stage but our algorithm makes selection for each node separately under some user defined constraints and optimization scales. Scheduling is done by the table search method to pick idle cycles and place an operation at that cycle if its rate allows. The compiler uses a two-level scheduler, one for scheduling the operations in a node or a fold, and one for scheduling the data flow between nodes or folds.

The run-time of the compiler is basically dependent on the libraries, because every sub algorithm runs in linear time except the wordlength estimation in FIR-based filters. Depending on the nodes and folding sets in the DFG, the estimation process may take longer than library search. For small systems like FIGURE 2.4(a) and FIGURE 2.3(a), the results are shown in TABLE 6.1.

The basic flow of the algorithm is presented in FIGURE 1.6 and FIGURE 1.7. This tool accepts an enhanced version of data flow graph (DFG) as the input file. Previously designed modules can be easily used if they appear as user-defined modules in the input file. The arithmetic unit modules can be set not only for all nodes but also for a group of nodes as well.

6.1. Future Research

The tool handles decimation process more successfully than the interpolation process due to the single output of the target architecture as shown in FIGURE 1.8. This architecture is poor at handling missing cycles of the interpolation because it

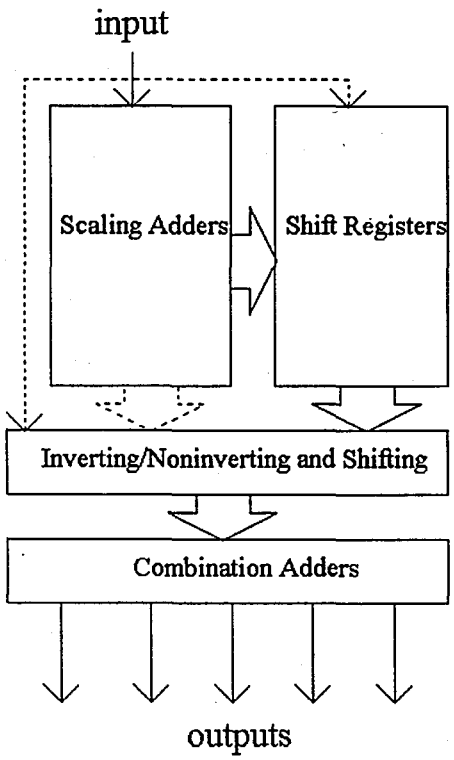


FIGURE 6.1. Target architecture for utilizing missing cycles in interpolation.

allows a single output at a time. However if there are multiple outputs, then the missing cycles will be filled with the concurrent operations. So the proposed architecture for fully utilizing missing cycles of interpolation process is shown in FIGURE 6.1.

The tool also allows folding process to reduce effective chip area and power. However, the folds are determined by the user. This is a good point for user-interaction, but the user cannot be sure which folding will produce a more effective result. For example if a comparison is made between FIGURE 4.12 and FIGURE 6.2, it will be seen that the first one requires a clock at least two times faster than the input u for correct operation while the second one can operate at the same frequency as the input u. Therefore, an automatic fold estimation strategy must be developed so as to find the best result which is FIGURE 6.2 in our example.

Finally, the tool produces an output file of switching bit-parallel-digit-serial architectures. However, this file cannot be processed by layout generators. The output format of the file must be changed to synthesizable VHDL for a more general use of application.

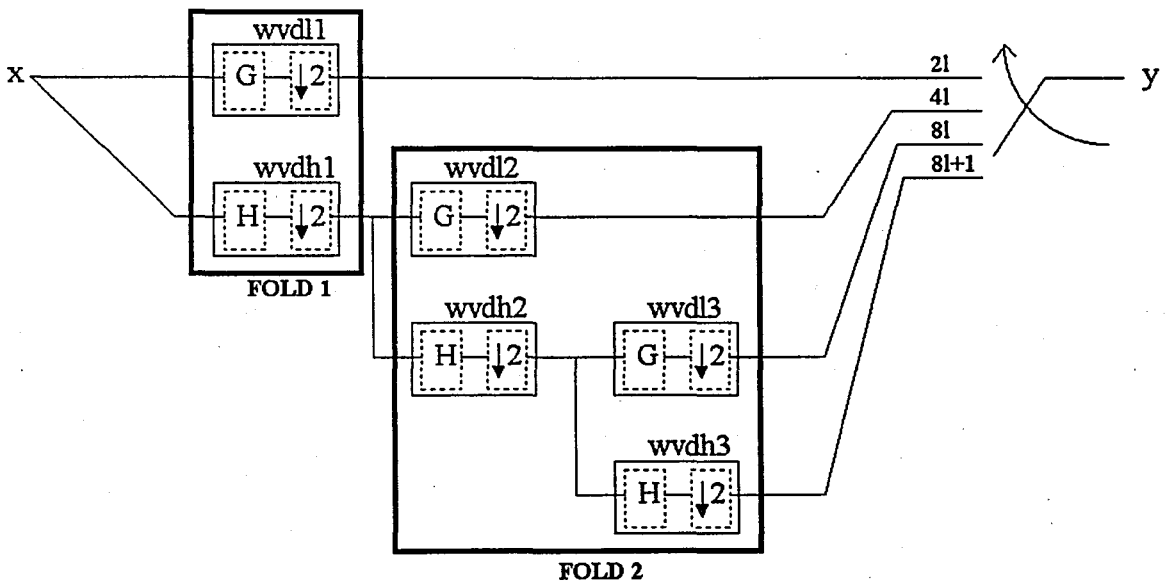


FIGURE 6.2. A different folding scheme for the analysis part of the two-band-three-level wavelet transform.

APPENDIX A.

USER MANUAL

A. 1. Usage

To use this program, type

word *filename*

The filename must be the dfg file in the format explained in Section A. 2. Then the program asks the user to enter the input file(s) if the system described in dfg file has one(s). Note that if the input data is an image file, it must be in raw format with an extension of ".raw". If it is a data file, then it must consist of only integer numbers and it must have an ".dat" extension.

It generates "out.dat" where the wordlength of modules and buses are determined.

A. 2. Input file format

The input file consists of a DFG part, and **GenSpecs** part as the minimum requirement. It can also have **Folds** and **NodeSpecs** parts. An input file must begin with the DFG part with the following lines:

TABLE A. 1 Types of nodes that must exist in DFG part.

a	:Adder
d	:Decimator
fir	:FIR
i	:Interpolator
m	:Multiplier
n	:Negator
u	:A user defined module
wvd	:Decimating FIR (FIR followed by an decimator)
wvi	:Interpolating FIR (FIR following an interpolator)
x	:Input
y	:Output

Number of Edges=N :Number of all edges that exist in a dfg. N is a positive integer.

Number of Vertices=N :Number of all vertices that exist in a dfg. N is a positive integer.

Then as many lines as determined by the Number of Edges must be entered in the following format by using all vertices whose number is given by the the Number of Vertices:

$source_{name}$ $source_{terminal}$ $target_{name}$ $target_{terminal}$ $delay\_switch_{period}$ $switch_{time}$

Node names can start with the strings shown in TABLE A. 1 for the description of their operational behaviors. Note that one can implement any multirate system by using all these node definitions. After the nodal system definitions, the following cards exists:

Folding Sets : This **optional** card is followed by lines that are in the format M $nodename_1$ ..... $nodename_M$ where M is the number of nodes in that fold, $nodename_M$ is the name of M'th node in the fold. Note that the folds must not overlap, i.e. no node can exist in more than one fold. This card must finish with a 0.

GenSpecs : Specifications are valid all over the system. This is a **necessary** card that must exist. Some of the commands must exist for the operation of the system. These are

clockfreq : Clock frequency in MHz.

tech : Technology of the system.

Optional commands binding all nodes are listed as follows

maxarea : Maximum allowable chip area in μm^2.

maxdelay : Maximum allowable chip delay in ns.

maxpower: Maximum allowable chip power in mW.

reglib: N $libname_1$ $libname_2$ ..... $libname_N$

: Use N number of latch libraries that are specified. Default is ALL, meaning the usage of all libraries of the specified technology. $libname_1$ is the name of the first library file. See Appendix B. 1 for details.

uselib: N $libname_1$ $libname_2$ ..... $libname_N$

: Use N number of module libraries that are specified. Default is ALL, meaning the usage of all libraries of the specified technology. $libname_1$ is the name of the first library file. See Appendix B. 1 for details.

genlib: N $libname_1$ $libname_2$ ..... $libname_N$

: Generate N number of module libraries that are specified. Default is ALL, meaning the generation of all libraries of the specified technology. $libname_1$ is the name of the first library file. See Appendix B. 2 for details.

bvall : Variable data bus. This is the default option, but if it is specifically written as a command then it cannot be overridden by single-node commands.

bvtall: L : Truncated data bus which can be at most L bits wide. It cannot be overridden by single-node commands.

bvrall: L : Rounded data bus which can be at most L bits wide. It cannot be overridden by single-node commands.

bftall: L : Truncated data bus which is L bits wide. It cannot be overridden by single-node commands.

bfrall: L : Rounded data bus which is L bits wide. It cannot be overridden by single-node commands.

Optional commands binding some nodes (i.e. group nodes) are listed below. Their difference from the commands for all nodes is that they can be overridden by single-node commands. Therefore all nodes except the ones that are singularly described by single-node commands are subject to the constraints defined by these commands.

bvn : Variable data bus.

bvtn: L : Truncated data bus which can be at most L bits wide.

bvrn: L : Rounded data bus which can be at most L bits wide.

bftn: L : Truncated data bus which is L bits wide.

bfrn: L : Rounded data bus which is L bits wide.

NodeSpecs : This card is **necessary** if FIR-based and user- defined nodes are used. It is followed by lines that give detailed information about each node. The specifications for all nodes are single-line commands. Its format is
nodename $spec_1 : exp_1 ... spec_K : exp_K$
where $spec_K$ is the K'th specification and exp_K is the explanation for K'th specification. Each command field is separated either by a comma or by a space. These are listed as follows:

e : A floating number that gives the error of the output node. It is the error at that output for the worst-case path.

end : End of input file.

ib : Number of integer bits for inputs. (Sign bit must be included).

fb : Number of fractional bits for inputs.

freq : Input(s) and output(s) frequencies in MHz. Cannot be higher than clockfreq.

cf: L : The coefficient quantization is done by L bits. This command is used for FIR-based nodes. If cf and cv are both used, then the last written overrides the previous one.

cv: L : The coefficient quantization is done by at most L bits. This command is used for FIR-based nodes. If cf and cv are both used, then the last written overrides the previous one.

cs: F : A positive-definite scaling coefficient for the node. It is 1.0 by default. The higher it is, the bigger the importance of the node in the optimization process.

bv : Variable bus at the output of the node.

bvt: L : Truncated bus which can be at most L bits wide at the output of the node.

bvr: L : Rounded bus which can be at most L bits wide at the output of the node.

bft: L : Truncated bus which is L bits wide at the output of the node.

bfr: L : Rounded bus which is L bits wide at the output of the node.

bs: F : A positive scaling coefficient for the output bus of the node. It is 0.0 by default. The higher it is, the bigger the importance of the bus in the optimization process.

c_file: *filename*

: The data file full with constants of the node, e.g. tap coefficients of an FIR filter.

q_type: C : If C is U, then it means unit quantization which is the direct conversion of the constants to their binary forms. If it is S, then it means scaled conversion which exploits the whole binary range for increasing noise immunity. Default is U.

su: L : Interpolated by L, an integer. Default is 1.

sd: L : Downsampled by L, an integer. Default is 1.

uselib: N $libname_1$ $libname_2$ ..... $libname_N$

: Overrides the general uselib specified by GenSpecs. N is 0 by default. For detailed information, see the Table TABLE A. 2.

genlib: N $libname_1$ $libname_2$ ..... $libname_N$

: Overrides the general genlib specified by GenSpecs. N is 0 by default. For detailed information, see the Table TABLE A. 2.

adduselib: N $libname_1$ $libname_2$ ..... $libname_N$

: Appends the general uselib specified by GenSpecs. N is 0 by default. For detailed information, see the Table TABLE A. 2.

addgenlib: N $libname_1$ $libname_2$ ..... $libname_N$

: Appends the general genlib specified by GenSpecs. N is 0 by default. For detailed information, see the Table TABLE A. 2.

u_file: *filename*

: The file which covers the information about the user defined nodes. Each user defined nodes must have such files for an errorprone operation. The commands that must be found in this file are

area : the area of the node in μm^2.

clockfreq : clock frequency in MHz.

power : the power of the node in mW.

in_ibit : Input integer bits (inc. sign bit).

in_fbit : Input fractional bits.

out_ibit : Output integer bits (inc. sign bit).

out_fbit : Output fractional bits.

.endmod : End of specifications.

Optional commands are:

e : Mean-squared error that will be added to system error by the inclusion of this node. It is 0 by default.

TABLE A. 2 Nodal effects when *uselib* in GenSpecs is used with *uselib* and *adduselib* in NodeSpecs. The same table applies to genlib also . SPEC means anything other than ALL.

GenSpecs	*NodeSpec*		*Nodal Results*
uselib	*uselib*	*adduselib*	
ALL	$N = 0$	$N = 0$	$uselib_{GenSpecs}$
ALL	$N = 0$	$N > 0$	$uselib_{GenSpecs}$
ALL	$N > 0$	$N = 0$	$uselib_{NodeSpecs}$
ALL	$N > 0$	$N > 0$	error !!
SPEC $(N > 0)$	$N = 0$	$N = 0$	$uselib_{GenSpecs}$
SPEC $(N = 0)$	$N = 0$	$N = 0$	error !!
SPEC $(N > 0)$	$N = 0$	$N > 0$	$uselib_{GenSpecs} + adduselib$
SPEC $(N = 0)$	$N = 0$	$N > 0$	error !!
SPEC	$N > 0$	$N = 0$	$uselib_{NodeSpecs}$
SPEC	$N > 0$	$N > 0$	error !!

pipeline : The level of pipelining. Default is 0, meaning no pipeline.

su: L : Interpolated by L, an integer. Default is 1.

sd: L : Downsampled by L, an integer. Default is 1.

.end : End of user input file

A typical input file is presented in FIGURE A. 1. It implements FIGURE 2.4(a) with ES2 technology at the clock frequency of $10MHz$.

```
Number of Edges= 8
Number of Vertices= 6
x 0 wvd1 0 0 0
x 0 wvd2 0 1 0
x 0 wvd3 0 2 0
x 0 wvd4 0 3 0
wvd1 0 y 0 0 4l
wvd2 0 y 0 0 4l+1
wvd3 0 y 0 0 4l+2
wvd4 0 y 0 0 4l+3
Folds=
4 wvd1 wvd2 wvd3 wvd4
0
GenSpecs=
clockfreq:100
tech:es2
maxarea:50000000
maxdelay:1
maxpower:30
uselib:1
lib1
reglib:1
reglib
NodeSpecs=
x ib:5 freq:100
wvd1 c_file:caglar_1.dat q_type:UNIT, sd:4
wvd2 c_file:caglar_2.dat q_type:UNIT, sd:4
wvd3 c_file:caglar_3.dat q_type:UNIT, sd:4
wvd4 c_file:caglar_4.dat q_type:UNIT, sd:4
y e:0.0045
end
```

FIGURE A. 1. A typical input file for a four-band analysis structure

APPENDIX B.

LIBRARY FORMAT

B. 1. For **reglib**, **uselib** and **adduselib** Commands

The HLS tool can use the modules that exist in files under LIB*tech*\ under work directory. For example, in FIGURE A. 1, the *tech* is specified as es2. This means that the library files must exist under LIB\es2\. For each module in the library, the necessary specifications can be listed as follows:

clockfreq:Clockfrequency in MHz.
area :The area of the module in μm^2.
power :The power of the module in mW.
name :Name of the module. Must be unique for correct operation
type :Type of the module. (see Table TABLE B. 1)
length :Wordlength of the module. It must be set to 1 for a register.
.endmod:End of specifications.

There exists only one optional card:
pipeline:Pipelining level of the operator. Default is 0, meaning that there is no pipelining.

Each library file must end with an **.endlib** card. A sample library is shown in Figure FIGURE B. 1. If **clockfreq** of a module is lower than the operating system frequency, than it is automatically dropped during library generation (see Section 4.1.1.).

TABLE B. 1 Types of nodes that must in a user defined library.

ADD	:Adder
MUL	:Multiplier
REG	:Register, latch

```
name:ADD_1 type:ADD length:4 area:28340 clockfreq:120 power:0.6 .endmod
name:ADD_2 type:ADD length:18 area:90900 clockfreq:400 power:0.297 pipeline:3 .endmod
name:ADD_3 type:ADD length:6 area:45630 clockfreq:400 power:0.75 .endmod
name:ADD_4 type:ADD length:8 area:54440 clockfreq:350 power:0.36 .endmod
name:ADD_5 type:ADD length:4 area:28000 clockfreq:7.5 power:0.04 .endmod
name:MUL_3 type:MUL length:4 area:10 clockfreq:450 power:5.6 .endmod
.endlib
```

FIGURE B. 1. A typical library used by **uselib** and **adduselib** commands.

For example, if this library is *lib*1 of Figure FIGURE A. 1, then ADD_5 will not be selected during library generation due to the operating system frequency, $100MHz$. Note that MUL_3 will not be selected in spite of satisfying the frequency criterion, because a multiplying node does not exist in DFG of the same figure.

A typical register library is presented in Figure FIGURE B. 2. Note that each **length** is set to 1.

B. 2. For **genlib** and **addgenlib** Commands

The HLS tool can generate the libraries of TABLE B. 3. These names are reserved for library generation. The rules of library generation is shown in TABLE

```
name:REG_01 type:REG length:1 area:2824 clockfreq:120 power:0.06 .endmod
name:REG_02 type:REG length:1 area:2020 clockfreq:400 power:0.066 .endmod
name:REG_03 type:REG length:1 area:4563 clockfreq:450 power:0.075 .endmod
name:REG_04 type:REG length:1 area:4896 clockfreq:350 power:0.033 .endmod
.endlib
```

FIGURE B. 2. A typical library used by the **reglib** command.

TABLE B. 2 Types of adders that can be generated using **genlib** and **addgenlib** comments.

Name	*Type*
RCA	Ripple Carry Adder
CSA	Carry Select Adder
CSkA	Carry Skip Adder
CLA	Carry Look-ahead Adder
MCA	Manchester Chain Adder

TABLE B. 3 Style of adder generation.

Name	*Type*
ALL	Generate all adders of all bits
ALL_*b*	Generate all adders of *b* bits
CSA	Generate Carry Skip Adders of all bits
CSA_*b*	Generate a Carry Skip Adder of *b* bits

B. 2 for CSA. The same rules are valid for all adders of TABLE B. 3. For example to generate a CLA of 5 bits, it is sufficient to write CLA_5. The details of library generation can be picked from [83] and [84].

APPENDIX C.

FORMAT FOR THE OUTPUT FILES

There are two output files: BEHAVE.DAT and STRUCT.DAT. These two files gives the behavioral and structural information about the newly formed system by the compiler. In both files, the nodes in FIR-based nodes and folds have the literals as shown in TABLE C. 1.

Both files consist of the new DFG and the information about the nodes in the new DFG except inputs and outputs.

Each line in the new DFG is underlined. If there exists a plain line in the new DFG (i.e., it is not underlined), then this means that it is the explanation line for the nearest underlined line before the plain line.

The nodes in m'th fold in the input file are represented by the new node FOLD_m.

The symbols $\longleftarrow$ or $\longrightarrow$ stand for the data-flow. The head of the arrow points at the target and the tail of the arrow is at the source.

The information within square brackets stands for the wordlength of the module. Here, the ":" character is the indicator for the place of the fractional point. For example

TABLE C. 1 Types of nodes existing in FIR-based nodes and folds

Name	*Type*
a	adder
b	buffer
n	negator
nb	negator or buffer in composite form
R	register

a2[2:5] means that a2 is a seven bit adder which has two bits for the integer and five bits for the fractonal parts.

C. 1. Format of BEHAVE.DAT

This file contains the behavioral information about the final system.

The information within square brackets stands for the wordlength of the module. Here, the ":" character is the indicator for the place of the fractional point. For example a2[2:5] means that a2 is a seven bit adder which has two bits for the integer and five bits for the fractional parts.

The information within paranthesis stands for equivalent nodes of a node. Here, the ":" character is the separator of different nodes. For example nb8=(n8:b8) means that nb8 is either n8 or b8.

The information within curly brackets stands for the latched data whose clocking instances are determined by the inital value given before the brackets. If there exists an "@" symbol, this means that the data is fetched at the clock instance and period following the symbol. For example, "a6 @ 4l+2 $\longrightarrow$ {R17 R18 R19}" means that data at the output of a6 is latched to R17 at 4l+2 where 4 is the clocking period and 2 is the clocking instance. Then the output of R17 is latched to R18 at 4l+3 and the output of R18 is latched to R19 at 4l and so on.

A typical BEHAVE.DAT is shown in FIGURE C. 1, FIGURE C. 2 and FIGURE C. 3. It is the behavioral description of the circuit given by FIGURE A. 1.

C. 2. Format of STRUCT.DAT

The STRUCT.DAT contains the structural information about the final system. It begins with the technical information like clock frequency, technology of the system,

```
New DFG
========

x -> FOLD_0
-----------------

F OLD_0-> y
-------------------
Registers:1 of wordlength [8:8]
FOLD_0 @ 41+0 -> {Rout Rout}
FOLD_0 @ 41+2 -> {Rout Rout}
y <- Rout @ 41+0
y <- FOLD_0 @ 41+1
y <- Rout @ 41+2
y <- FOLD_0 @ 41+3

FOLD_0( in[5:0], out[8:8] )
===================================

Scaling Adders
---------------------
a2[3:5] <- (in -> 1R[5:0])>>5 + n7>>3
a3[2:8] <- in>>8 + in>>4
a5[3:5] <- a2 + (in -> 4R[5:0])>>1
a6[5:8] <- a3 + (in -> 3R[5:0])>>1
n7[5:0] <- in
Registers:16 of wordlength [5:8]
a3 @ 41+3 -> { R6 R7 R13 R14 R16 R9 R10 }
a2 @ 41+3 -> { R9 R10 R11 R12 R13 R14 R15 R16 }
a5 @ 41+1 -> { R8 }
a6 @ 41+3 -> { R15 R16 R14 R15 }
a6 @ 41+2 -> { R11 R12 R13 }
a5 @ 41+2 -> { R10 R11 R12 }
a2 @ 41+2 -> { R9 R10 R11 R12 R13 R14 }
a3 @ 41+0 -> { R1 R2 R3 R4 R5 R6 R7 R8 }
a3 @ 41+1 -> { R1 R2 R3 R4 R5 R6 R7 R8 R9 }
a5 @ 41+3 -> { R1 R15 }
a6 @ 41+0 -> { R3 R4 R5 }
a2 @ 41+0 -> { R2 R3 R4 R5 R6 R7 R8 }
```

FIGURE C. 1. A typical BEHAVE.DAT (Figure 1 of 3)

```
Inverting, Noninverting and Shifting
-----------------------------------------------------
nb8[3:8] = (n8:b8)
nb9[3:8] = (n9:b9)
nb10[5:8] = (n10:b10)
nb11[5:8] = (n11:b11)
nb12[5:8] = (n12:b12)
nb13[5:8] = (n13:b13)
nb14[3:8] = (n14:b14)
nb15[3:8] = (n15:b15)
n8 <- R6 @ 4l+0
n9 <- R9 @ 4l+0
b10 <- a5 @ 4l+1
b11 <- R16 @ 4l+1
b12 <- R13 @ 4l+1
b13 <- R12 @ 4l+1
n14 <- R14 @ 4l+0
n15 <- R8 @ 4l+0
b8 <- R9 @ 4l+0
b9 <- R3 @ 4l+0
b10 <- R15 @ 4l+0
b11 <- R1 @ 4l+0
n12 <- R10 @ 4l+3
n13 <- R5 @ 4l+3
n14 <- R7 @ 4l+3
n15 <- R8 @ 4l+3
b8 <- R9 @ 4l+3
n9 <- R2 @ 4l+2
b10 <- R11 @ 4l+3
n11 <- a5 @ 4l+2
n12 <- R8 @ 4l+2
b13 <- R15 @ 4l+3
n14 <- R10 @ 4l+2
b15 <- R16 @ 4l+3
n8 <- R1 @ 4l+1
b9 <- R3 @ 4l+2
b10 <- a5 @ 4l+2
n11 <- R3 @ 4l+1
b12 <- R14 @ 4l+2
n13 <- R15 @ 4l+1
n14 <- R14 @ 4l+1
b15 <- R9 @ 4l+2
```

FIGURE C. 2. A typical BEHAVE.DAT (Figure 2 of 3)

```
Combination Adders
------------------------------
a16[4:8] <- nb8 + nb9
a17[6:8] <- nb10 + nb11
a18[6:8] <- nb12 + nb13
a19[4:8] <- nb14 + nb15
a20[7:8] <- a16 + a17
a21[7:8] <- a18 + a19
a22[8:8] <- a20 + a21
out = a22
```

FIGURE C. 3. A typical BEHAVE.DAT (Figure 3 of 3)

the register libraries and final system specifications like area, delay and power.

For each node and fold, the arithmetic unit libraries appear after the name of the node and fold.

The information within curly brackets stands for the structural explanation of a node. Each clock instance is described by "pipe" word. For example "pipe1" is the first clock cycle in the adder. The modules latched at a clock instance are separated via "|" symbols as follows :

$$m * register : n_r \quad module : n_m \tag{C.1}$$

Here, m is the number of registers of type *register* of n_r bits, and there exists only one arithmetic unit of type *module* of n_m bits. If "EMPTY" appears instead of module or register types, this means that it is not implemented. For example if "EMPTY" appears in place of *register*, then there does not exist a register for this entry.

A typical STRUCT.DAT is shown in FIGURE C. 4 and FIGURE C. 5. It is the structural description of the circuit given by FIGURE A. 1.

```
Technology: ES2
Clock Frequency: 100MHz
Area: 2.91e6 um2
Delay: 0.23 us
Power: 28.82 mW
Register Libraries: reglib

New DFG
========
x -> FOLD_0
-----------------
F OLD_0-> y
------------------
Registers:1 REG_02:16

FOLD_0( in[5:0], out[8:8] )
===================================

Aritmetic Module Libraries: lib1
Scaling Adders
---------------------
Registers:4 REG_02:5
a2[3:5] = {
pipe1: 0*EMPTY:0 ADD_02:18
pipe2: 0*EMPTY:0 EMPTY:0
pipe3: 0*EMPTY:0 EMPTY:0  }
a3[2:8] = {
pipe1: 0*EMPTY:0 ADD_02:18
pipe2: 0*EMPTY:0 EMPTY:0
pipe3: 0*EMPTY:0 EMPTY:0  }
a5[5:5] = {
pipe1: 0*EMPTY:0 ADD_02:18
pipe2: 0*EMPTY:0 EMPTY:0
pipe3: 0*EMPTY:0 EMPTY:0  }
a6[5:8] = {
pipe1: 0*EMPTY:0 ADD_02:18
pipe2: 0*EMPTY:0 EMPTY:0
pipe3: 0*EMPTY:0 EMPTY:0  }
n7[5:0] = {
pipe1: 1*REG_04:5 ADD_04:8  }

Registers:16 REG_02:13
---------------------------------
```

FIGURE C. 4. A typical STRUCT.DAT (Figure 1 of 2)

```
Inverting, Noninverting and Shifting
------------------------------------------------
nb8[3:8] = {
pipe1: 1*REG_04:8 ADD_04:8  | 1*REG_04:3 ADD_04:8  }
nb9[3:8] = {
pipe1: 1*REG_04:8 ADD_04:8  | 1*REG_04:3 ADD_04:8  }
nb10[5:8] = {
pipe1: 1*REG_04:8 ADD_04:8  | 1*REG_04:5 ADD_04:8  }
nb11[5:8] = {
pipe1: 1*REG_04:8 ADD_04:8  | 1*REG_04:5 ADD_04:8  }
nb12[5:8] = {
pipe1: 1*REG_04:8 ADD_04:8  | 1*REG_04:5 ADD_04:8  }
nb13[5:8] = {
pipe1: 1*REG_04:8 ADD_04:8  | 1*REG_04:5 ADD_04:8  }
nb14[3:8] = {
pipe1: 1*REG_04:8 ADD_04:8  | 1*REG_04:3 ADD_04:8  }
nb15[3:8] = {
pipe1: 1*REG_04:8 ADD_04:8  | 1*REG_04:3 ADD_04:8  }

Combination Adders
------------------------------
a16[4:8] = {
pipe1: 0*EMPTY:0 ADD_02:18
pipe2: 0*EMPTY:0 EMPTY:0
pipe3: 0*EMPTY:0 EMPTY:0  }
a17[6:8] = {
pipe1: 0*EMPTY:0 ADD_02:18
pipe2: 0*EMPTY:0 EMPTY:0
pipe3: 0*EMPTY:0 EMPTY:0  }
a18[6:8] = {
pipe1: 0*EMPTY:0 ADD_02:18
pipe2: 0*EMPTY:0 EMPTY:0
pipe3: 0*EMPTY:0 EMPTY:0  }
a19[4:8] = {
pipe1: 0*EMPTY:0 ADD_02:18
pipe2: 0*EMPTY:0 EMPTY:0
pipe3: 0*EMPTY:0 EMPTY:0  }
a20[7:8] = {
pipe1: 0*EMPTY:0 ADD_02:18
pipe2: 0*EMPTY:0 EMPTY:0
pipe3: 0*EMPTY:0 EMPTY:0  }
a21[7:8] = {
pipe1: 0*EMPTY:0 ADD_02:18
pipe2: 0*EMPTY:0 EMPTY:0
pipe3: 0*EMPTY:0 EMPTY:0  }
a22[8:8] = {
pipe1: 0*EMPTY:0 ADD_02:18
pipe2: 0*EMPTY:0 EMPTY:0
pipe3: 0*EMPTY:0 EMPTY:0  }
```

FIGURE C. 5. A typical STRUCT.DAT (Figure 2 of 2)

APPENDIX D.

ERROR CODES

The HLS tool stops operating at the conditions which cause the following error codes:

ERROR: Cannot open *filename.*

Trouble: A file named *filename* does not exist in the working directory.

Solution: Be sure that the file named *filename* really exists in the working directory.

ERROR: Invalid format in *filename.*

Trouble: Two different bus specifications exist for the system.

Solution: See bus specification comments of **GenSpecs** card in Appendix A.

Trouble: A node that does not exist in DFG part appears in one of the folds or under **NodeSpecs**card.

Solutions:

1. Check names of all nodes used in folds and under **NodeSpecs** card appear in DFG part (see Appendix A.).

2. Check that the number of edges and vertices are correct.

ERROR: Invalid node in *filename.*

Trouble: The type of a node in *filename* cannot be identified.

Solutions:

1. Check that all nodes in *filename* begin with one of the strings specified in Table TABLE A. 1.

2. Check that the number of edges and vertices are correct.

ERROR: Invalid, unmatching or unspecified features for *nodename*.

Troubles:

1. **e** command exists at *nodename* which is not an output.
2. **u_file**command exists at *nodename* which is not a user-defined module.
3. Some or all of **ib, fb, freq** commands exist at *nodename* which is not an input.
4. If *nodename* is an input, then its frequency is higher than the system clock frequency.
5. Some or all of **cf, cv, cs, c_file, q_type** commands exist for *nodename* which is not an FIR-based node.
6. Bus specifications of *nodename* does not match with the system's.
7. Some or all of **adduselib, uselib, addgenlib, genlib** commands exist at *nodename* which is an input, or an output, or a user-defined module.
8. If *nodename* is an input, it does not have both **ib** and **fb** commands.

Solution: Correct specifications of *nodename* under **NodeSpecs** card.

Trouble: Libraries of *nodename* does not match libraries of **GenSpecs** part.

Solution: See Table TABLE A. 2 to correct errors.

Troubles: If *nodename* is a user-defined module, then it

1. does not have a specific name for layout generation.
2. has an undefined specification.
3. does not have area and power specifications.
4. does not have input bits or output bits.
5. either does not have an operation frequency or has an operation frequency lower than the system clock frequency.

Solution: See **u_file** command in Appendix A to correct errors.

Troubles: If *nodename* is a module in a latch or arithmetic unit library, then it

1. does not have a specific name for layout generation
2. has an undefined specification.
3. does not have area and power specifications.
4. does not have a wordlength.
5. does not have an operation frequency.
6. either does not have a type or has an undefined type.

Solution: See Appendix B to correct errors.

ERROR: Clock frequency is unspecified.

Trouble: System clock frequency is not specified.

Solution: Enter the system clock frequency as described in **GenSpecs** part in Appendix A.

ERROR: Technology is unspecified.

Trouble: System technology is not specified.

Solution: Enter the system technology as described in **GenSpecs** part in Appendix A.

ERROR: Insufficient memory!

Trouble: Memory is not sufficient to open an internal array.

Solution: Increase the effective memory in the system.

ERROR: m and n cannot exist in the same fold.

Troubles:

1. m and n have different libraries.
2. m and n have different values of fixed bus (coefficient) quantization.
3. One of them has a fixed value of bus (coefficient) quantization and the other one has an upper bound which is lower than the fixed value on variable bus (coefficient) quantization.
4. One of them has a fixed bus, the other one has a rounded bus.
5. m and n have different values of fixed bus or coefficient quantization.
6. Cannot schedule m and n in the same fold with the given switching periods, or input and system clock frequency.
7. m is a user-defined node and
   (a) n has a **c_file** command.
   (b) n has library commands.
   (c) *su* and/or *sd* values are not equal.
   (d) both nodes have different **e** values.

Solution: Either separate the nodes or change the specifications.

ERROR: Infeasible solution space.

Trouble: Fixed errors or upper bounds on quantization bits cause infeasibility.

Solution: See Section 2.3..

ERROR: Wrong inputs or outputs for *nodename.*

Troubles: 1. *nodename* is an input and appears as a target.

2. *nodename* is an output and appears as a source.

3. *nodename* is an adder or a multiplier but its number of sources is less than two or it has no target.

4. *nodename* is an FIR-based node, or a negator, or a decimator or an interpolator but it has no source or no target.

Solution: Correct the DFG part.

ERROR: No latches are available to satisfy the given system.

Trouble: The latches given in the latch libraries are too slow to realize the system with the given clock frequency (see **clockfreq** in Appendix A.).

Solutions: 1. Change the used latch libraries so that there exist latches that can operate at the given clock frequency.

2. Decrease the clock frequency.

ERROR: No library elements are available to satisfy the given system.

Trouble: The elements given in the arithmetic unit libraries are too slow to realize the system with the given clock frequency (see **clockfreq** in Appendix A.).

Solutions: 1. Change the used arithmetic unit libraries so that there exist modules that can operate at the given clock frequency.

2. Decrease the clock frequency.

ERROR: Rate mismatch at the input of node *nodename*.

Troubles: 1. The sources of *nodename* appear at the input of *nodename* at different rates.

2. *nodename* is an output and sources to the output cannot be scheduled.

Solution: Check rate of each source.

REFERENCES

[1] Gajski, D. D. and R. H. Kuhn, "Guest Editors Introduction: New VLSI Tools," *IEEE Computer Magazine*, vol. 16, no. 12, pp. 11-14, December 1983.

[2] De Micheli, G., *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, 1994.

[3] Gebotys, C. H. and M. I. Elmasry, *Optimal VLSI Architectural Synthesis: Area, Performance and Testability*, Kluwer Academic Publishers, Boston, 1992.

[4] Sherwani, N. A., *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, Boston, 1993.

[5] Fox, J. R., "A Higher Level of Synthesis", *IEEE Spectrum*, vol. 30, no. 3, pp. 43-47, March 1993.

[6] Bergamaschi, R. A. and A. Kuehlman, "A System for Production Use of High-Level Synthesis," *IEEE Transactions on VLSI Systems*, vol. 1, no. 3, pp. 233-243, September 1993.

[7] Geuskens, B., CATHEDRAL: Silicon Compilers for Digital Signal Processing, Advanced VLSI Design, Rensselaer Polytechnique Institute, April 1993.

[8] Hwang, C. T., J. H. Lee, and Y. C. Hsu, "A Formal Approach to the Scheduling Problem in High Level Synthesis," *IEEE Transactions on CAD*, vol. 10, pp. 464-475, April 1991.

[9] Ito, K., L. E. Lucke, and K. K. Parhi, "ILP-Based Cost Optimal DSP Synthesis with Module Selection and Data Format Conversion," *IEEE Transactions on VLSI Systems*, vol. 6, pp. 582-594, December 1998.

[10] Denk, T. C. and K. K. Parhi, "Synthesis of Folded Pipelined Architectures for Multirate DSP Algorithms," *IEEE Transactions on VLSI Systems*, vol. 6, pp. 595-607, December 1998.

[11] Chao, L. F., A. S. LaPaugh and E. H. M. Sha, "Rotation Scheduling: A Loop Pipelining Algorithm," *IEEE Transactions on CAD*, vol. 18, pp. 229-239, March 1997.

[12] Hwang, C. T., Y. C. Hsu, and Y. L. Lin, "PLS-A Scheduler for Pipeline Synthesis," *IEEE Transactions on CAD*, vol. 12, pp. 1279-1286, September 1993.

[13] Bakshi, S. and D. D. Gajski, "Component Selection for High-Performance Pipelines," *IEEE Transactions on VLSI Systems*, vol. 4, pp. 181-194, June 1996.

[14] Park, N. and A. C. Parker, "Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications," *IEEE Transactions on CAD*, vol. 7, no. 3, pp. 356-370, March 1988.

[15] Paulin, P. G. and J. P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASICs," *IEEE Transactions on CAD*, vol. 8, no. 6, pp. 661-679, June 1989.

[16] Ranganathan, N., *VLSI Algorithms and Architectures*, IEEE Computer Society Press, California, 1993.

[17] Burleson, W. P., M. Ciesielski, F. Klass, and W. Liu, "Wave-Pipelining: A Tutorial and Research Survey," *IEEE Transactions on VLSI Systems*, vol. 6, no. 3, pp. 464-474, September 1998.

[18] Brodersen, R. W., *Anatomy of a Silicon Compiler*, Kluwer Academic Publishers, Boston, 1992.

[19] Parhi, K. K., C. Wang and A. P. Brown, "Synthesis of Control Circuits in Folded Pipelined DSP Architectures," *IEEE Journal of Solid State Circuits*, vol. 27, pp. 29-43, January 1992.

[20] Berstis, V., "The V Compiler: Automating Hardware Design," *IEEE Design and Test*, vol. 6, no. 1, pp. 8-17, April 1989.

[21] Fliege, N. J., *Multirate Digital Signal Processing*, John Wiley & Sons, Inc., Chichester, England, 1994.

[22] Crochiere, R. E. and L. R. Rabiner, *Multirate Digital Signal Processing,* Prentice-Hall Inc., New Jersey, USA, 1983.

[23] Mitra, S. K. and J. F. Kaiser, *Handbook for Digital Signal Processing,* John Wiley & Sons, Inc., New York, USA, 1993.

[24] Parhi, K. K. and T. Nishithani, "VLSI Architectures for Discrete Wavelet Transform," *IEEE Transactions on VLSI Systems*, vol. 1, pp. 191-202, June 1993.

[25] Knowles, G., "VLSI Architecture for the Discrete Wavelet Transform," *Electronics Letters*, vol. 26, pp. 1184-1185, July 1990.

[26] Lewis, A. S. and G. Knowles, "VLSI Architecture for 2-D Daubechie Wavelet Transform without Multipliers," *Electronics Letters*, vol. 27, pp. 171-173, January 1991.

[27] Wu, P. C., L. G. Chen, and T. D. Chiueh, "Scalable Implementation Scheme for Multirate FIR Filters and Its Application in Efficient Design of Subband Filter Banks," *IEEE Transactions on Video Technology*, vol. 6, pp. 407-410, August 1996.

[28] Grzeszczak, A., M. M. Mandal, S. Panchanatan and T. Yeap, "VLSI Implementation of Discrete Wavelet Transform," *IEEE Transactions on VLSI Systems*, vol. 4, pp. 421-433, December 1996.

[29] Denk, T. C. and K. K. Parhi, "VLSI Architectures for Lattice Structure Based Orthonormal Discrete Wavelet Transforms," *IEEE Transactions on CAS II*, vol. 44, pp. 129-132, February 1997.

[30] Fridman, J. and E. S. Manolakos, "Discrete Wavelet Transform: Data Dependence Analysis and Synthesis of Distributed Memory and Control Array Architectures," *IEEE Transactions on Signal Processing*, vol. 45, pp. 1291-1308, May 1997.

[31] Kim, J. T., Y. H. Lee, T. Isshiki, and H. Kunieda, "Scalable VLSI Architectures for Lattice Structure-Based Discrete Wavelet Transform," *IEEE Transactions on CAS-II*, vol. 45, pp. 1031-1043, August 1998.

[32] Wu, A. Y., and K. J. R. Liu, "Algorithm-Based Low-Power Transform Coding Architectures: The Multirate Approach," *IEEE Transactions on VLSI Systems*, vol. 6, pp. 707-718, December 1998.

[33] Parhi, K. K., "A Systematic Approach for Design of Digit-Serial Signal Processing Architectures," *IEEE Transactions on CAS*, vol. 38, no. 4, pp. 358-375, April 1991.

[34] De Man, H., J. Rabaey, P. Six, and L. Claesen, "Cathedral-II: A Silicon Compiler for Digital Signal Processing," *IEEE Design and Test*, vol. 3, no. 4, pp. 13-25, December 1986.

[35] Lippens, P. E. R., J. L. van Meerbergen, A. van der Werf, W. F. J. Verhaegh, B. T. McSweeney, J. O. Huisken, and O. P. McArdle, "PHIDEO: A Silicon Compiler for High Speed Algortihms," *Proceedings of European Conference on Design Automation (EDAC-91)*, pp.436-441, Amsterdam, Netherlands, February 1991.

[36] Biesenack, J., M. Koster, A. Langmaeir, S. Ledeux, S. März, M. Payer, M. Pilsl, S. Rumler, H. Soukup, N. Wehn, and P. Duzy, "The Siemens High-Level Synthesis System CALLAS," *IEEE Transactions on VLSI Systems*, vol. 1, no. 3, pp. 233-243, September 1993.

[37] Park, I. C. and C. M. Kyong, "FAMOS: An Efficient Scheduling Architecture for High-Level Synthesis," *IEEE Transactions on CAS*, vol. 12, pp. 1437-1447, October 1993.

[38] Wilson, T. C., G. W. Grewal, and D. K. Banerji, "An ILP Solution for Simultaneous Scheduling, Allocation and Binding in Multiple Block Synthesis," *Proceedings of IEEE International Conference on Computer Design*, pp. 581-586, Cambridge, Massachusets, 1994.

[39] Wang, C. Y. and K. K. Parhi, "High-level DSP Synthesis Using Concurrent Transformations, Scheduling and Allocation," *IEEE Transactions on CAD*, vol. 14, pp. 274-295, March 1995.

[40] Chen, C. T. and K. Küçükçakar, "High Level Scheduling Model and Control Synthesis for a Broad Range of Design Applications," *Proceedings of IEEE/ACM International Conference on CAD-97*, pp. 236-243, San Jose, California, 1997.

[41] Lakshminarayana, G., K. S. Khouri, and N. K. Jha, "Wavesched: A Novel Scheduling Technique for Control-flow Intensive Behavioral Descriptions," *Proceedings of IEEE/ACM International Conference on CAD-97*, pp. 244-250, San Jose, California, 1997.

[42] Elbek, G., S. Balkır and G. Dündar, "Design and Simulation of a Two-Band Three-Level Wavelet Decomposition Architecture Using Folding Algorithm," *Proc. of ICT'96*, pp.178-181, İstanbul, September1996.

[43] Atabek, Y., G. Dündar, S. Balkır, E. Anarım, and H. Çağlar, "Design of M-Band Wavelet Filter with Perfect Reconstruction Architecture", *Proc. of ICT'96*, pp.225-228, İstanbul, September 1996.

[44] Dempster, A. G. and M. D. Macleod, "Variable Statistical Wordlength in Digital Filters," *IEE Proceedings-Vision Image Signal Processing*, vol. 143, pp. 62-66, February 1996.

[45] Parhi, K. K., "Finite Word Effects in Pipelined Recursive Filters," *IEEE Transactions on Signal Processing*, vol. 39, pp. 1450-1454, June 1991.

[46] Dündar, G. and K. Rose, "Effects of Quantization on Multilayer Neural Networks," *IEEE Transactions on Neural Networks*, vol. 6, pp. 1446-1451, November 1995.

[47] Choy, J. and K. Rose, "The Impact of Reduced Quantization on DSP Design for Wavelet Transformation," in press.

[48] Bomar, B. W., L. M. Smith, and R. D. Joseph, "Roundoff Noise Analysis of State-Space Digital Filters Implemented on Floating Point Digital Signal Processors," *IEEE Transactions on CAS-II*, vol. 44, pp. 952-955, November 1997.

[49] Cho, N. I. and S. U. Lee, "Optimal Design of Finite Precision FIR Filters Using Linear Programming with Reduced Constraints," *IEEE Transactions on Signal Processing.*, vol. 46, pp. 195-199, January 1998.

[50] Oppenheim, A. V. and R. W. Schafer, *Digital Signal Processing,* Prentice-Hall Inc., New York, USA, 1975.

[51] Alkın, O. and H. Çağlar, "Design of Efficient M-band Coders with Linear-Phase and Perfect Reconstruction Properties," *IEEE Transactions on Acoustics, Speech, Signal Processing*, vol. 43, pp. 1579-1590, July 1995.

[52] Bazaraa, M. S., J. J. Jarvis and H. D. Sherali, *Linear Programming and Network Flows*, John Wiley & Sons, Inc., Singapore, 1990.

[53] Fiacco, A. V. and G. P. McCormick, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, John Wiley & Sons, Inc., USA, 1968.

[54] Daubechies, I., "The Wavelet Transform, Time-Frequency Localization and Signal Analysis," *IEEE Transactions on Information Theory*, vol. 36, pp. 961-1005, July 1990.

[55] Garner, H. L., "Number Systems and Arithmetic," *Advanced Computers*, vol. 6, pp. 131-194, 1965.

[56] Hwang, K., "Computer Arithmetic: Principles, Architectures and Design," John Wiley & Sons, USA, 1979.

[57] Dempster, A. G. and M. D. Macleod, "Constant Integer Multiplication Using Minimum Adders," *IEE Proceedings - Circuits Devices Systems*, vol. 141, pp. 407-413, Oct. 1994.

[58] Bull, D. R. and D. H. Horrocks, "Primitive Operator Digital Filters," *IEE Proceedings G*, vol. 138, pp. 401-412, 1991.

[59] Lim, Y. C. and S. R. Parker, "FIR Filter Design over a Discrete Powers-of-Two Coefficient Space," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-31, pp. 583-590, June 1993.

[60] Samueli, H., "An Improved Search Algorithm for the Design of Multiplierless FIR Filters with Powers-of-Two Coefficients," *IEEE Transactions on CAS*, vol. 36, pp. 1044-1047, July 1989.

[61] Shaffeu, H., M. M. Jones, H. D. Griffiths, and J. T. Taylor, "Improved Design Procedure for Multiplierless FIR Digital Filters," *Electronics Letters*, vol. 27, pp. 1142-1144, June 1991.

[62] Horng, B., H. Samueli and A. N. Willson, "The Design of Low Complexity Linear-Phase FIR Filter Banks Using Powers-of-Two Coefficients with an Application to Subband Image Coding," *IEEE Transactions on CAS for Video Technology*, vol. 1, pp 318-324, December 1991.

[63] Cemes, R. and D. Ait-Boudaoud, "Genetic Approach to Design of Multiplierless FIR Filters," *Electronics Letters*, vol. 29, pp. 2090-2091, November 1993.

[64] Oh, W. J. and Y. H. Lee, "Implementation of Programmable Multiplierless FIR Filters with Powers-of-Two Coefficients," *IEEE Transactions on CAS II*, vol. 42, pp. 553-555, August 1995.

[65] Jain, R., P. T. Yang, and T. Yoshino, "FIRGEN: A Computer-Aided Design System for High Performance FIR Filter Integrated Circuits," *IEEE Transactions on Signal Processing*, vol. 39, pp. 1655-1668, July 1991.

[66] Mehendale, M., S. D. Sherlekar and G. Venkatesh, "Synthesis of Multiplier-less FIR Filters with Minimum Number of Additions," *Proceedings of IEEE/ACM International Conference on Computer-Aided-Design,* 1995.

[67] Li, D., "Minimum Number of Adders for Implementing a Multiplier and Its Application to the Design of Multiplierless Digital Filters," *IEEE Transactions on CAS II*, vol. 42, pp. 453-460, July 1995.

[68] Dempster, A. G. and M. D. Macleod, "Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters,". *IEEE Transactions on CAS II*, vol. 42, pp. 569-577, September 1995.

[69] Dempster, A. G. and M. D. Macleod, "General Algorithms for Reduced-Adder Integer Multiplier Design," *Electronics Letters*, vol. 31, pp. 1800-1802, October 1995.

[70] Hartley, R. I., "Subexpression sharing in filters using Canonic signed digit multipliers," *IEEE Transactions on CAS II*, vol. 43, pp. 677-688, October 1996.

[71] Potkonjak, M., M. B. Srivastava, and A. P. Chandrakasan, "Multiple constant multiplications: Efficient and versatile framework and algorithms exploring com-

mon subexpression elimination," *IEEE Transactions on CAD*, vol. 15, pp. 151-165, February 1996.

[72] Pasko, R., P. Schaumont, V. Derudder and D. Durackova, "Optimization method for broadband modem FIR filter design using common subexpression elimination," *Proceedings of ISSS'97*, 1997.

[73] Li, J. and S. Tantarana, "Multiplierfree realizations for FIR multirate converters based on mixed-radix number representation," *IEEE Transactions on Signal Processing*, vol. 45, pp. 880-890, April 1997.

[74] Dempster, A. G. and M. D. Macleod, "Comments on "Minimum number of adders for implementing a multiplier and its application to the design of multiplierless digital filters,". *IEEE Transactions on CAS II*, vol. 45, pp. 242-243, February 1998.

[75] Chang, T. S. and C. W. Jen, "Hardware efficient transform designs with cyclic formulation and subexpression sharing," *Proceedings of ISCAS'98*, Monterey CA, May 1998.

[76] Pasko, R., P. Schaumont, V. Derudder, S. Vernalde and D. Durackova, "A new algorithm for elimination of common subexpressions," *IEEE Transactions on CAD*, vol. 18, pp. 58-68, January 1999.

[77] Yurdakul, A. and G. Dündar, "Multiplierless realization of linear DSP transforms by using common two-term expressions," *Journal of VLSI Signal Processing*, Kluwer Academic Publishers, in press.

[78] Yurdakul, A. and G. Dündar, "Statistical Methods for the Estimation of Quantization Effects and Determination of Optimal Quantization Stepsize in FIR-Based Multirate Systems," *IEEE Transactions on Signal Processing*, in press.

[79] Yurdakul, A. and G. Dündar, *Statistical Methods for the Estimation of Quantization Effects in FIR-Based Multirate Systems*, Technical Report, FBE(EE-1/97-9), Boğaziçi University, Turkey, 1997.

[80] Nemhauser, G. L. and L. A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, USA, 1988.

[81] Parhi, K. K., "Calculation of Minimum Number of Registers in Arbitrary Life Time Chart," *IEEE Transactions on CAS-II*, vol. 41, no. 6, pp. 434-436, June 1994.

[82] Parhi, K. K., "Systematic Synthesis of DSP Data Format Converters Using Lifetime Analysis and Forward-Backward Register Allocation," *IEEE Transactions on CAS-II*, vol. 39, no. 7, pp. 423-440, July 1992.

[83] Diker, A., "An Optimization Method for the Estimation of Power Dissipation in Adders as CMOS Arithmetic Building Blocks" Ms. Thesis, Bogazici University, February 1999.

[84] Karakuş, G., "A Fast and Accurate Delay Estimation Method for Adders as CMOS Arithmetic Building Blocks in VLSI Design" Ms. Thesis, Bogazici University, February 1999.