Floating-point to Fixed-point Conversion

By

Changchun Shi

B.S. (California Institute of Technology) 1998
M.S. (University of California, Berkeley) 2002

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

In

Engineering-Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor Robert W. Brodersen, Chair
Professor Laurent El Ghaoui
Professor Deborah Nolan
Spring 2004

The dissertation of Changchun Shi is approved:

_____

Chair Date

_____

Date

_____

Date

University of California, Berkeley

Spring 2004

Floating-point to Fixed-point Conversion

by

Changchun Shi

ABSTRACT

Statistical Quantization Effects and Floating-point to Fixed-point Conversion

Changchun Shi
Professor Robert Brodersen, Advisor
Department of Electrical Engineering and Computer Science
University of California, Berkeley

The digital signal processing (DSP) algorithms used by communication systems are typically specified as floating-point or, ideally, infinite precision operations. On the other hand, digital VLSI implementations of these algorithms rely on fixed-point approximations to reduce cost of hardware while increasing throughput rates. One essential step of a top-down design flow is to determine the fixed-point data type of each signal node, namely the word-length, truncation mode and overflow mode. This is commonly referred as floating-point to fixed-point conversion (FFC) problem. Conventional approaches are typically both time-consuming and error-prone since ad-hoc assignments of fixed-point data type are performed manually and iteratively.

We first formulate FFC problem into an optimization framework. The optimization variables are defined by the fixed-point data-types to be determined; the objective function is hardware cost, and the constraint functions are system specifications. In this unified point of view the past techniques are compared. A primary goal is to make the optimization automatic and fast which requires an understanding of the relationships between these functions and the variables. One critical step is the identification of the

right metric that judges the quality of an FFC and is sufficiently general. This metric is directly related to quantization effects and will serve as the constraint functions. We first categorize functional blocks in a system according to their quantization behavior; then, a novel statistical perturbation theory provides the guideline of using simulations to obtain constraint functions in their semi-analytical form. The theoretical work reduces the otherwise exponential complexity of characterizing quantization effects to a polynomial one. The other critical step to achieve automated FFC is the automatic acquisition of hardware-cost function. This has been done using a high level resource estimation tool and function-fitting method.

Based on the preceding methodology, an FFC tool in Matlab and Simulink environment has been built for Xilinx FPGA designs as a demonstration. The FFC tool has been successfully tested on several complicated digital designs—namely a binary phase shift keying (BPSK) transceiver, a U-Sigma block of singular value decomposition (SVD) system and an Ultra-wide band (UWB) system. The conversions normally take from minutes to hours, varying according to system complexity. These are orders of magnitudes faster than existing tools, which are projected to take weeks to do the conversions. Without reducing system performance, the FFC can reduce their hardware-costs by 1.5 to 50 times.

The hardware resource estimation part of our FFC utility is based on my summer intern project in Xilinx, Inc. Unlike existing resource estimations that rely on post-netlisting information or post-placement-and-routing map report, this pre-netlisting estimator (now part of System Generator 3.1) in Matlab environment speeds up estimations by 2-3 orders of magnitudes.

The proposed FFC methodology can also be applied to ASIC design when hardware cost is chip area, power consumption, and so on. One necessary pre-requisite is a similar hardware estimation tool and hardware cost function model.

_____

Professor Robert W. Brodersen

To My Wife Chao

Table of Contents

List of Figures

List of Tables

Acknowledgements

Without the following people, the research described in this thesis would be impossible.

I'd like to express my sincere gratitude to my advisor Professor Bob Brodersen. Bob is one of the few persons who I felt lucky to meet in my life. Besides his truly insightful technical advices that guide me through my PhD and M.S. projects, he is always patient, trustful and encouraging to support me becoming a better scholar and a better person. With his inspiration of "practicing to become strong", I became physically active and I truly enjoyed it. His understanding of my "telecommuting" research style made my graduate study so much easier. His selflessness, rightness and enthusiasm in research set him a perfect role-model for both my academic career and personal life.

I am also especially grateful to have Professor Laurent El Ghaoui and Professor Deborah Nolan in my thesis committee. They taught me two of the most enjoying and intellectually stimulating courses at Berkeley—Convex Optimization and Theoretical Statistics—from which I benefited so much. Their supports and cares contribute as inseparable factor that makes this thesis possible. I also want to thank them to spend time on reading this thesis.

I'd also like to thank some other colleagues. Many great researchers have been working on related subjects over the last few decades. Without them, the current research

could have never gone so far. I am especially grateful to some of the best professors who taught me related material or give me useful advice: Professor Laurent El Ghaoui, Professor David Tse, Professor Deborah Nolan (in Statistics department), Professor Jan Rabaey, Professor Borivoje Nikolic, Professor Edwards Lee, and many others. I also want to thank my group mates: Haiyun Tang, Kathy Lu, Ning Zhang, Mike Chen, Peimin Chi, Ada Poon, Dejan Markovic, Danejila, Tufan Karalar, Kevin Camera, Engling Yeo and many others. I had learned a lot from many of the useful discussion with them.

My thanks also go to the researchers at Xilinx, Inc., where I did my summer Intern. They are Jim Hwang, Nabeel, Vinay, Scott McMillan, David Square, and many others in the System Generator and DSP groups. Their genuinely nice personality shaped my attitude to life. Their active healthy life-style and advices also made that summer the most influential and enjoying time.

I also want to thank Dr. Jim Hwang (again) at Xilinx Inc., Professor Rajeev Jain at University of California of Los Angeles (now also with Conexant Inc.), Dr. Etan Cohen at Conexant Inc., Dr. David Shaw at DEShaw research group, Dr. Benoit Lemonnier at Synplicity Inc., and Dr. Robert Hewes and Dr. Manish Goel at Texas Instrument. Their help, time and understandings during the last few months of my graduate study, together with Bob's and Chao's constant supports, shaped my research attitude and set off the next stage of my research career.

Besides the scientific advisors, other professional and friendly staff members made my life at Berkeley Wireless Research Center (BWRC) and EECS department an easy and happy experience. In particular, I want to thank Tom Boot and Ruth Gjerde for

their professional and personal cares and inspirations. They set the warm tone for us students to enjoy and concentrate on research. I also want to thank Elise Mills and Brian Richard for their numerous helps on administrative supports.

Also, Tom Boot, Tufan Karalar, Ruth Gjerde, and Mike Chen have helped me greatly on taking care of the submission of this dissertation while I was away working in New York. Many thanks to them!

I like to thank some other great scholars I am lucky to meet earlier in my life, they are Professor Yousheng Shu at Peking University and Professor Peter Asbeck at University of California at San Diego. Their trusts and selfless supports built up my confidence and shaped my love of science, technology and life in general.

Last but certainly not least, I want to express my heartfelt thanks to my wife Chao. Her beauty and honesty have transformed me into a better person. Because of Chao, my love of every beautiful thing has become purer. Without her, I wouldn't be able to have the persistency to carry this research to the current depth. Without her, I, as a person, can hardly be so happy inside. I also want to thank my parents who always provide me the freedom to do what I feel the right thing. Finally, I wish to thank my best friend, Chengzhou Wang, whose true friendship since middle school has made my life more joyful and complete. My four cats, Bob, QQ, Xiaobao and π, accompany me through my Ph.D. years, especially many of the nights when I need to stay up late. It is such a privilege and luck to have them and to take care of these four little "spirits"!

Chapter 1
Introduction

This thesis analyzes some difficulties associated with floating-point to fixed-point conversion (FFC) problem. Several analytical results that we derived from a perturbation theory simplify the problem and lead to an automated FFC methodology for digital VLSI communication systems. As a demonstration, the methodology is implemented into a FFC tool in Matlab and Simulink environment that is used to convert floating-point FPGA systems.

In this introduction chapter, the FFC problem is defined and motivated. A global optimization point of view is used to abstract the problem into mathematical level; at this level, various aspects of achieving both efficient and reliable FFC become easier to identify. Then, the organization of the thesis is described. Instead of full explanation of all the topics with detailed references, only an outline with a few references that are necessary to illustrate the general picture of FFC are given here.

A great deal of understanding of FFC problem has been previously conducted in my master thesis “statistical methodology for floating-point to fixed-point conversion”

[3]. By studying, summarizing, and extending the results of a large number of references, such as [72-86], it serves as a good source of getting you familiar with the nature of FFC problem as well as a practical methodology for linear-time-invariant (LTI) system in particular. I believe it would be more beneficial if my master thesis [3] can be read briefly before this thesis as the latter is based on the knowledge learned there. In fact, this thesis, in my opinion, can be viewed as an extension of [3]. Overlaps between these two are tried to be minimized whenever possible. Many unsolved issues by the time that my master thesis was completed have found their answers here. It should be a fun experience to read both of them.

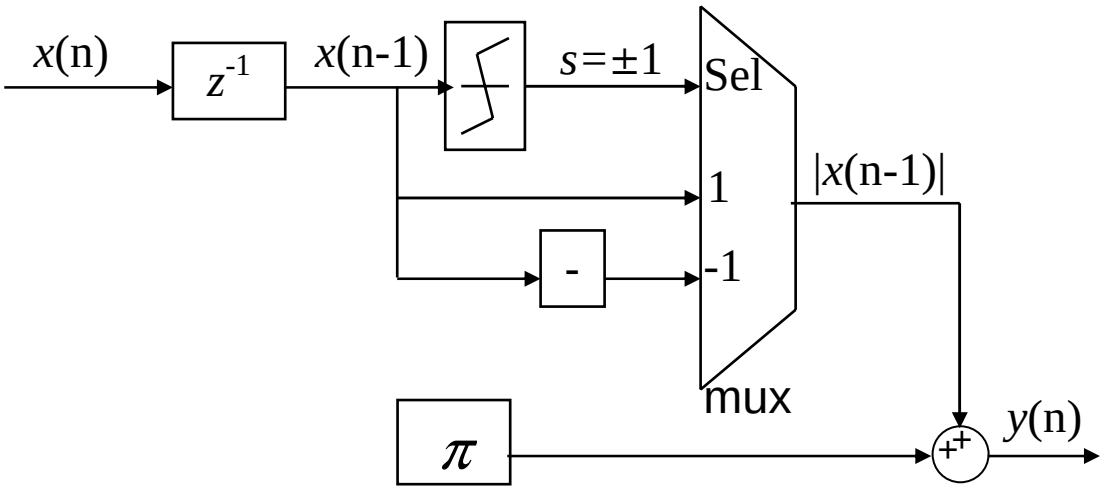
1.1 Definition of Floating-point to Fixed-point Conversion

The algorithms used in communication systems are typically first proposed as algebraic operations. We usually numerically verify them, especially complicated ones, in digital computer under some carefully designed test vectors. The underlying data types in this digital computing are either single precision floating-point or double precision floating-point [55]. Either one of these floating-point representations has limited number of bits used for mantissa and exponent. This inevitably causes numerical errors, either due to roundoff at least-significant-bit (LSB) side of the mantissa, or due to the saturation of the exponent. Our goal is to implement the algorithm in application specific integrated circuits (ASIC). So, the realization of the algorithm in digital computer is only to simulate its performance; thus, the designers need to assure these numerical errors are negligible to validate these verifications. Topics related to this kind of numerical error are beyond this work. Fortunately, they are almost always negligible for most communication and digital signal processing (DSP) systems because of the following two

reasons. First, in these systems, input signals are of relatively small variation in dynamic range. Second, only a few significant bits of an output value are meaningful, whereas others are corrupted by unknown physical noise anyway. Therefore, the single and double floating data types are practically treated as infinite-precision as mentioned in my master thesis [3]. As a result, unless otherwise specified, the rest of the thesis will refer floating-point data type and infinite-precision date type interchangeably.

$$y(n) = \pi + |x(n-1)|$$

(a)

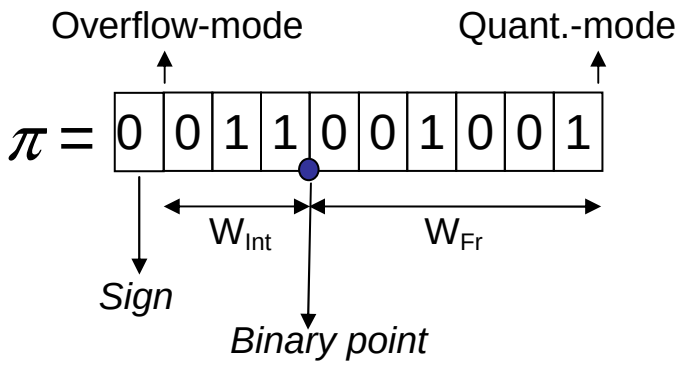


(b)

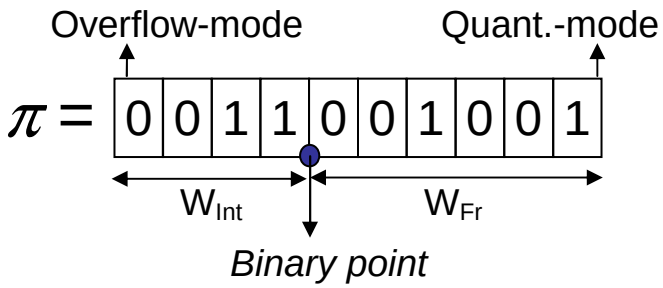
Fig. 1-1 (a) A conceived algorithm in algebraic form, and (b) architectural form

In a top-down design flow to implement these algorithms, the next step is to determine the system architecture, such as the amount of parallelism and pipelining scheme [56]. For example, in an Orthogonal-frequency-division-modulation (OFDM)

communication system, it is necessary to choose the architecture for its FFT unit: column based, CORDIC based, or fully parallel butterfly-based, and so on [57-58]. The details of this architectural description should reach the level of arithmetic operators, such as adders, multiplexers (MUX's), and delays. In this thesis, we assume this description is already given, possibly by system architecture designers. Similar to algorithms, in case of lacking information to choose one out of several promising architectures at this moment, all of them should be implemented to certain level and compared. As an example, Fig. 1-1 shows part of a conceived DSP algorithm described in both its algebraic form, and its architectural form. The architecture designers have to verify that the inclusion of structural refinements does not modify the algorithm functionality. This is done again by floating-point simulations under the same test vectors that are used previously.



(a)



(b)

Fig. 1-2 Fixed-point representations of π:
(a) 2's complement, and (b) unsigned magnitude.

Recently some design environments that catch architectural information and allow high speed simulations have been developed. These tools include graphical platforms such as Simulink from MathWorks [59] and System Generator from Xilinx, Inc. [60] that is built on top of Simulink environment. Other non-graphical high level design environments such as SystemC based on C [61] and AccelChip based on Matlab [62] allows even faster behavioral simulation, though they are not as intuitive as the graphical ones. Once the structural floating-point system has been tested, the immediate task is to determine the data types that are feasible in a final implementation. A large number of digital ASIC implementations rely on purely fixed-point approximations to reduce hardware costs while increasing throughput rates. Other approximation methods, such as light-weight floating-point design [9], are beyond the scope of this thesis. Fig. 2-2 shows two most commonly used fixed-point data types that are considered, representing the irrational number π. Therefore, we need to determine the fixed-point data type of each signal node, namely the number type (either 2's complement signed or unsigned), word-length (both integer word length and fractional word length), truncation mode (either roundoff or truncation) and overflow mode (either saturation or wrap-around).

A negative integer word length is valid; it represents that the first few bits in fractional part are unnecessary to be specified since their values do not vary. Similarly fractional word lengths can be negative as well. Fixed-point data type has this flexibility to save hardware. For example, let's use the architecture in Fig. 1-1 to perform $|x|$ for an even integer value x between -7 to 7; that is, x is -6, -4, -2, 0, 2, 4 or 6. In 2's complement format, x is 1010, 1100, 1110, 0, 0010, 0100, or 0110. At the first glance, both the

Negate block and the Mux block in Fig. 1-1 need to support at least 4-bit fixed-point operation. In fact, only 3 bits are needed in hardware because the last integer bit is always 0. In this case, one can specify the fractional word length as -1 to save hardware.

In the previous example, three bits are needed for two reasons. First, the range of x is known, and the integer word length can be 4 without any overflow. Second, we know the last integer bit is not needed. In information theory, the bits to the left of the 4th integer bit carry no information since the entropy of the i^{th} bit given the 4th bit is identically 0, that is,

$$\begin{aligned} &P(b_i = 1 \mid b_4) \cdot \log P(b_i = 1 \mid b_4) + P(b_i = 0 \mid b_4) \cdot \log P(b_i = 0 \mid b_4) \\ &= 0, \forall i > 4, \end{aligned} \tag{1-1}$$

where P denotes probability and b_i is the i^{th} bit to the left of binary point. Similarly, b_1 in our example has 0-entropy because this bit is always 0 in even integers.

On the other hand, even with none-zero information, some bits can be eliminated as long as the information is not useful in particular application. For example, this information may be dominated by physical noises or architecture defects, or it may simply be ignored by the rest of the system that processes it. This thesis is largely to identify those bits with valuable information. Conventional approaches rely on manual and try-and-error assignments of fixed-point data types. These methods are both time-consuming and error-prone [13]. As communication systems and digital signal processing units become increasingly complex, more intelligent methods are called for. Here, we assume the system is already implemented in architecture level, and the goal is to quickly and reliably produce the fixed-point correspondence. We define this part of design flow as the Floating-point to Fixed-point Conversion (FFC) problem. Finite-word-length

effects can drastically vary depending on the system architecture. For example, a finite impulse filter (FIR) implemented in different architectures, such as direct form I, direct form II, transposed forms, and cascaded form, all have different quantization behaviors [4][5][56]. Therefore a system description lacking of its structural information provides is not ready for FFC problem. Once the fixed-point data types are resolved, a design flow continues to circuit level and physical level [56], which is again beyond the scope of this work.

In Section 1.2, the FFC problem is formulated into an optimization problem. Under this unified framework, our methodology is briefly explained in Section 1.3, which also points out the chapters that give detailed studies.

1.2 FFC formulation in optimization framework

An FFC problem often happens in one of the following two situations. First, under certain test vectors that the algorithm designers have carefully chosen, the floating-point system with architectural information passing to FFC stage already satisfies some behavioral system specifications. A specification is usually based on some statistics of output data, such as output bit-error rate (BER) and signal-to-noise ratio (SNR). Since fixed-point data types are unknown yet, no reliable hardware-cost information can be introduced at the floating-point design stage. Therefore, FFC is to decide the fixed-point data types throughout the system, such that the system still satisfies the same specifications. The goal here is to minimize hardware cost. Equation (1-2) shows the optimization frame work described above

$$
\begin{aligned}
&\text{minimize hardware - cost} \\
&f_{\mathrm{HW}}(\mathrm{W}_{\mathrm{Int},1}, \mathrm{W}_{\mathrm{Fr},1}, \mathrm{W}_{\mathrm{Int},2}, \mathrm{W}_{\mathrm{Fr},2}, \ldots; o_1, o_2, ..; q_1, q_2, ...; n_1, n_2, ...) \\
&\text{subject to specifications} \\
&S_j(\mathrm{W}_{\mathrm{Int},1}, \mathrm{W}_{\mathrm{Fr},1}, \mathrm{W}_{\mathrm{Int},2}, \mathrm{W}_{\mathrm{Fr},2}, \ldots; o_1, o_2, ..; q_1, q_2, ...; n_1, n_2, ...) \\
&\quad < 0, \forall j,
\end{aligned}
\tag{1-2}
$$

where

f_{HW}- hardware-cost function,

S_j- the j^{th} system behavioral constraint function,

and the following variables are associated with the fixed-point data types of the i^{th} signal node,

$\mathrm{W}_{\mathrm{Int},i}$- integer word length (always integer),

$\mathrm{W}_{\mathrm{Fr},i}$- fractional word-length (always integer),

o_i- overflow mode (0 for wrap-around, or 1 for saturation),

q_i - quantization modes (0 for truncation, or 1 for round-off to the nearest),

n_i - number systems (0 for 2's complement, or 1 for unsigned magnitude).

Here we have limited the search space formed by these variables to what is commonly used in circuit design, as shown in parentheses.

Second, sometimes designer have constraints in hardware-cost, such as area and power, together with some system specifications. And the objective is to minimize another system performance. Without losing generality, we let the objective function be S_1. Then this situation can be modeled as

$$
\begin{aligned}
&\text{minimize a system performance} \\
&S_1(\text{fixed - point data types}) \\
&\text{subject to} \\
&f_{\text{HW}}(\text{fixed - point data types}) < f_0, \\
&S_j(\text{fixed - point data types}) \quad < 0, \forall j > 1,
\end{aligned}
\tag{1-3}
$$

where the fixed-pint data types are same specified as in (1-2) and f_0 is a hardware-cost specification value. Interestingly, this problem is essentially equivalent to (1-2) [10]. In fact, if we modify the first constraint in (1-2) and obtain the following variation

$$
\begin{aligned}
&\text{minimize hardware cost} \\
&f_{\text{HW},\, s_1}(\text{fixed - point data types}) \\
&\text{subject to specifications} \\
&S_1(\text{fixed - point data types}) \quad < s_1, \\
&S_j(\text{fixed - point data types}) \quad < 0, \forall j > 1,
\end{aligned}
\tag{1-2'}
$$

where a slack variable s_1 is introduced in the first constraint. Problem (1-2') and (1-2) are only slightly different, and can be solved with the same method and complexity. In fact, solving (1-2') repeatedly for different s_1, we can get a tradeoff curves between f_{HW} and s_1, as shown in Fig. 1-3. The curve must be non-increasing due to the construction of (1'). Fig. 1-3 also shows the feasible regions for (1-2) and (1-3). Basically, if we can solve any problem in format (1-2) and (1-2') efficiently, problem (1-3) is solved directly from the tradeoff curve.

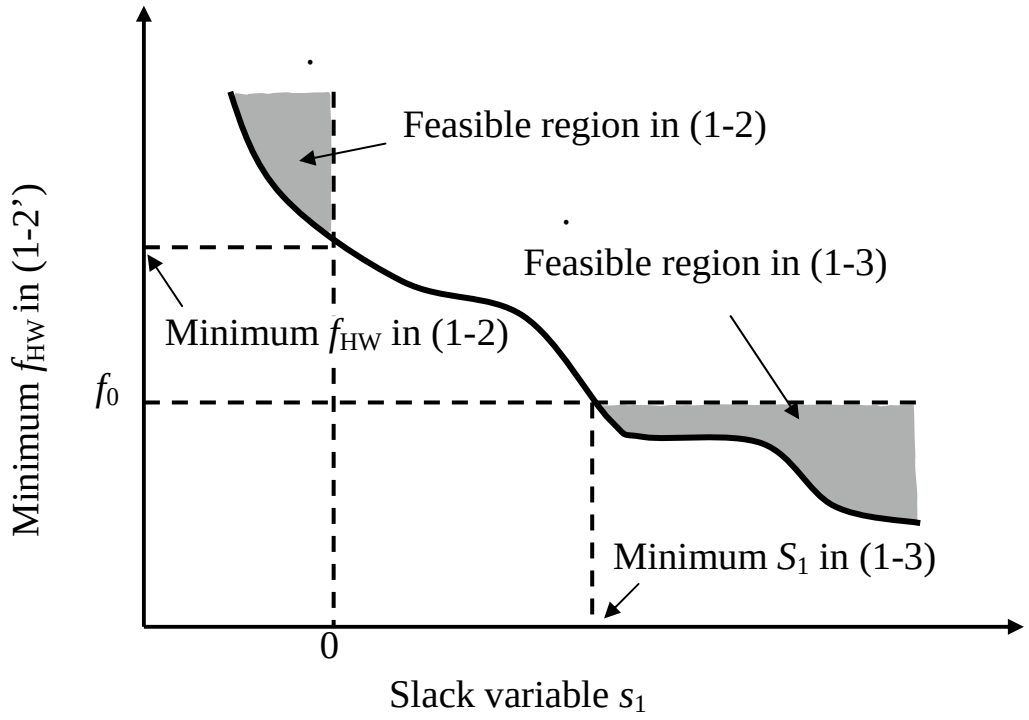


Fig. 1-3 Tradeoff curve between minimum hardware and specification level. The curve shows a possible minimum hardware cost in problem (1-2') as a function of slack variable s_1. Both optimization problems defined in (1-2) and (1-3) become trivial with the curve.

Therefore, we consider (1-2) as our basic optimization formulation of FFC. The constraints in (1-2) should be defined in such a way that they are satisfied at least by the floating-point system. If we choose very large word lengths and 2's complement number systems for all signal nodes in the fixed-point system, the system becomes arbitrarily close to infinite-precision. Consequently, the optimization problem (1-2) must be feasible. On the contrary, problem (1-3) is not guaranteed feasible if, for example, f_0 is set negative or too small for a hardware-area cost function.

1.3 Our methodology and thesis organization

In this section, we will briefly describe some past FFC techniques and our strategies. This serves the purpose of getting you familiar with many of the related

problems; it also point you to the particular chapter that you are most interested in, including the hardware-cost estimation, the quantization effects, the optimization algorithm, as well as how to automate the methodology into usable a FFC tool.

1.3.1 Brief review of the past techniques

Recently a few strategies have been proposed to automate FFC for communication systems described in C/C++ [13-15]. While the integer-part word-length and overflow modes of the fixed-point operands are commonly determined by avoiding signal overflow, the determination of the fractional word-length relies on different methods. In both [13] and [14], there is no gross hardware-cost function given; neither is the problem treated as an optimization. However, the implicit goal is to minimize all the word-lengths at the same time. The task of minimizing multiple objective functions is unrealistic unless the constraints are special so that they all can be minimized simultaneously. This is done in [14] by having one constrain function for each word-length: the input word lengths are pre-assigned; for the rest of the word lengths the integer part should be sufficiently large to cover the signal ranges that it governs (same for both [13] and [15]), and the fractional part should be sufficiently large so the local quantization noise power is much smaller than the one caused by quantizations of the inputs. These strongly decoupled constraint functions are always feasible and can minimize all word lengths at the same time. However, the gross quantization effects from these locally justified quantization sources altogether can still be much greater than the one induced by input quantizations. Therefore it is still necessary to have a final constraint on system performance (e.g. SNR or bit-error rate) as a function of all the word

lengths done by simulations. This becomes the reason for unbounded number of iterations.

In [13], the unjustified pre-assignments of date-types on some signal nodes provide some constraint equations. The deterministic propagation methodology yields inequalities among the fractional word lengths, e.g. the fractional word length at the output of a multiplier should be no less than the sum of those of the two inputs, while the output fraction word length of a delay component should be no less than the input one. Besides the overly pessimistic consideration of quantization effects, feedback loops such as the one in an accumulator can yield contradictive inequalities. This is solved in [13] by possible user interaction using engineering decisions, which also yield undetermined design time.

The work in [15] implies a similar problem formulation to ours, and again has the same treatment on integer word lengths as most other methods. The constraints are chosen to be the system specification functions. However the lack of investigation of the closed form specification function limits their optimization algorithm to be purely heuristic and time-consuming search. In addition, the Monte Carlo simulations among iterations can be inconsistent which adds further complications.

To probe further of the vast literature, refer to Section 2.1, 2.2 and 4.1.

1.3.2 Practical, Reliable and Cost-efficient FFC

In this thesis, the same constraint functions as [13-15] on integer word lengths are adopted in the current work. The assumption that overflow noise hurt the quality of the

design greatly is widely reasonable. However, in some special situations occasional overflows in saturation mode are acceptable and even expected, which won't be discussed further. For each input statistics, a single estimation is needed for the ranges of all signal nodes. With all W_{Int} and *o*-modes determined separately, these variables are dropped out from the optimization problem in (1-2). In the following subsections the hardware-cost function and constraint functions will be studied.

1.3.3 Analytical form of hardware-cost function

A single hardware cost function is to be minimized in eq. (1-2). This could be area, power consumption, and so on. High-level estimations of hardware resources such as area, energy and delay have been studied extensively. For system level optimization, it often suffices to adopt the parameterized library based approach. The area of each block of the library can be modeled as a function of parameters related to fixed-point data types as well as other important technology factors such as feature size and voltage. Provided the architecture choice with all other parameters fixed, the area cost of a library block is uniquely characterized as a function of the fixed-point data-type parameters. The total area of the system can then be estimated as a sum of all the required blocks plus a certain routing overhead. This usually yields a hardware-cost that is a quadratic function of W_{Fr}'s and q's. More detailed discussion can be found in part of Chapter 4 and Chapter 5.

1.3.4 Choices and analytical forms of constraint functions

Unlike in [14] where a number of additional constraint functions are created to solve the multiple-objective-function situation, only the system specifications (such as

bit-error rate and SNR) that are eventually used to judge the quality of the design are initially considered as the constrains. Furthermore, instead of employing the system specifications directly as the optimization constrains as used in [15], a more robust specification scheme is proposed based on the statement that the fixed-point system is expected to deviate only little from the floating-point origin. One natural alternative specification replacing (NOT in addition to) the system specifications is their relative changes between floating and fixed point systems. An innovative perturbation theory is developed, and shows that the change to the first order is a linear combination of all the first and second-order statistics of the quantization noise sources. With the widely used theoretical models of the means and correlations of the quantization noise sources and a couple more assumptions, Chapter 2 and 3 (the latter one concentrates on the situation when one of the assumption is not satisfied) tells us this alternative specification function can be written into closed form

$$\begin{aligned} &|\text{sys. spec. (fxpt)} - \text{sys.spec.(flpt)}| \\ &= \overline{\mathrm{M}}^{\mathrm{T}}\,\bar{u} + \sum_{i\in\{\text{Data Path}\}} c_i 2^{-2\mathrm{W}_{\mathrm{Fr},i}} + \ldots, \end{aligned}$$

where c_i's are constants and $\overline{\mathrm{M}}$ is a constant column vector, the i^{th} element of $\bar{\mu}$ is

$$u_i = \begin{cases} -\frac{1}{2} q_i 2^{-\mathrm{W}_{\mathrm{Fr},i}} & \text{for datapath} \\ \text{fxpt}(a_i, 2^{-\mathrm{W}_{\mathrm{Fr},i}}) - a_i & \text{for const } a_i \end{cases}, \quad q_i = \begin{cases} 0, \text{if round-off} \\ 1, \text{truncation} \end{cases}. \tag{1-4}$$

Function fxpt(a, d) means the value among the set {integer $\times$ d} that is the closest to a, and a_i's are the constants (e.g. filter coefficients) that appear in the floating-point design.

The linear coefficients $\overline{\mathrm{M}}$ and c_i's can be data-fitted by running polynomial numbers of Monte-Carlo simulations. However unacceptably large sample sizes may be

needed to conduct an accurate Monte-Carlo bit-true simulation to detect the small perturbation on top of a large value, whose own estimation error can easily hide the small perturbation. This important issue is resolved by choosing the mean square error (MSE) as the specification function. The MSE error is the output difference between the floating-point system and the fixed-point system.

$$\mathrm{MSE} = \bar{u}^T \mathrm{B} \bar{u} + \sum_{i \in \{\text{Data Path}\}} \mathrm{C}_i 2^{-2\mathrm{W}_{\mathrm{Fr},i}},$$

where B is positive semi-definite, denoted as $\mathrm{B} \succeq 0$, $\mathrm{C} \geq 0$. Again, more details can be found at Chapter 2.

A totally independent study on general multiple-input-multiple-out linear-time-invariant (MIMO LTI) systems based on transfer function method has been conducted in my master thesis [3] which confirms the validation of the MSE formula. The present results are much more general since they apply to non-LTI systems with non-stationary input.

1.3.5 FFC Design Automation

Now, the FFC problem is safely reduced to

minimize

$$\text{Quadratic } f(\mathrm{W}_{\mathrm{Fr},1}, \mathrm{W}_{\mathrm{Fr},2}, \ldots; q_1, q_2, \ldots)$$

subject to

$$\bar{u}^T \mathrm{B}_k \bar{u} + \sum_{i \in \{\text{Data Path}\}} \mathrm{C}_{i,k} 2^{-2\mathrm{W}_{\mathrm{Fr},i}} - A_k < 0,$$

$$\text{with } \mathrm{B}_k \succeq 0, \mathrm{C}_{i,k} \geq 0, \text{ and } A_k > 0.$$

Here vector $\bar{u}$ is defined in the same way as before, and A_k is the tolerance of the k^{th} MSE error. The problem is feasible because as all W_{Fr} increase, the left sides of the constraint

functions asymptotically converge to $-A_k$'s which are always less than 0. Physically that means the fixed-point system becomes infinite precision.

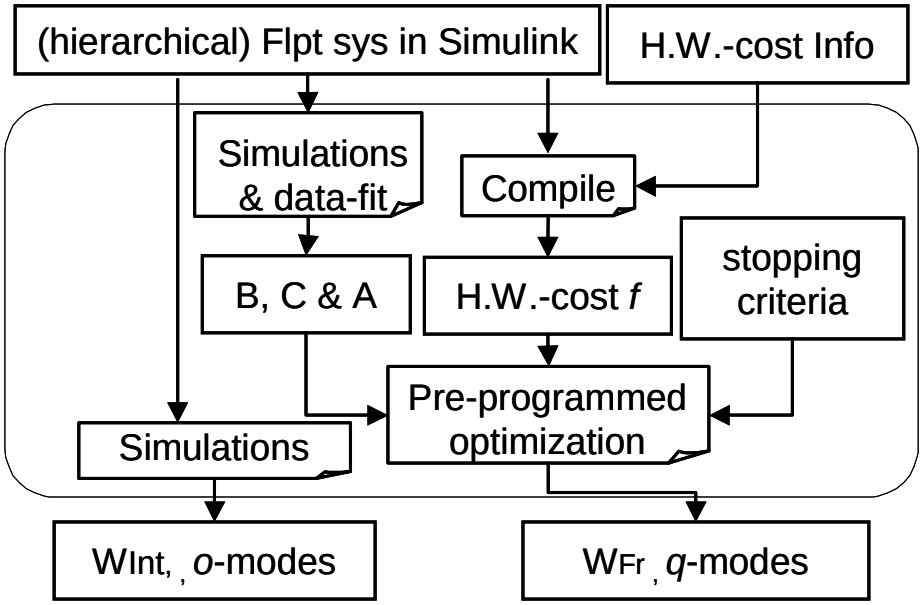


Fig. 1-4 FFC design flow graph.

An essential part of a practical FFC is to automate the process of obtaining the analytical hardware cost function and the analytical specification function. This is achieved following the design flow in Fig. 1-4. First the signal ranges need to be estimated automatically by running one simulation. This simulation also provides us the MSE tolerance vector, A. Secondly, a number of simulations can be conducted following the MSE formula to find out matrix B and C. The analytical hardware-cost function can be achieved by automatically reading the system parameters, provided the hardware-cost formula for each block. Our current design environment is Xilinx System Generator that is based on Mathwork Simulink and Matlab. Chapter 4 and 5 shows that all the tasks above can be automated. The number of Monte-Carlo simulations need to be done is proportional to $[\dim(B)^2 + \dim(C) + 1]$. Chapter 4 further discusses some simplifications

to reduce the number of simulations. Finally, the optimization algorithm specifically suitable for this problem can be preprogrammed.

1.3.6 Scalability

A partition of the system MSE specification into block-wise MSE specifications can factorize the problem into several smaller optimization problems. Moreover, many of the word lengths along forward-directional data path can be pre-related to reduce the number of optimization variables. These two strategies ensure the applicability of the proposed methodology on large communication systems.

1.4 Summary

This chapter serves several purposes.

First, it emphasizes my master thesis [3] and suggests it to be read first. Though its title is similar to this thesis, their contents are deliberately made almost non-overlap to efficiently present the information. One can view this thesis an extension, though my master thesis contains many useful results itself, such as its study of quantization effects of LTI systems.

Second, a brief description of the thesis is provided to prepare you a big picture of the FFC problem. With pointers to specific chapters, you can go directly to the specific chapters for certain topics.

Chapter 2
A Statistical Perturbation Theory on the Quantization Effects

As mentioned in Chapter 1, any general understanding of quantization effects can benefits the FFC process. Most existing studies on fixed-point quantization effects rely on either pure simulations or pure analyses. Pure simulations require extensive computing power, and provide limited insights. The associated complexity is exponential function of the number of fixed-point parameters. Whereas pure analyses aim to find explicit relationships between quantization effects and all system parameters which often becomes too difficult to accomplish even with extensive assumptions and case-by-case efforts to get a result. General theory only exists for linear-time-invariant (LTI) systems.

Based on an innovative perturbation theory and three assumptions—independent and white quantization noises, small noises, and no decision-error-propagation, this chapter derives an analytical relationship between statistical quantization effects and FP parameters and lump all other system parameters into some coefficients. The theory does not require stationary inputs. When a system description and input statistics are given, the theory provides a procedure to understand its statistical quantization effects both

analytically and numerically. In the numerical approach, only a polynomial number of simulations are needed, whereas in the analytical approach, existing theory in LTI systems with stationary inputs becomes a special case. Finally, several examples are given to verify and clarify the theory.

2.1 Brief introduction

As described in the previous chapter, the algorithms used in communication systems and digital signal processing are typically specified with infinite precision (IP) operations at the beginning. In literature, especially for circuit designers, these operations are sometimes referred as floating-point because floating-point computations in digital computers possess high precisions [1-3]. On the other hand, digital implementations of these algorithms rely on finite precision (FP) approximations, sometimes also called limited precision. Finite precision number systems are often represented by fixed-point realizations, among which the most popular one used in hardware implementation systems is binary number, such as 2's complement and binary unsigned number (see, for example [4-9]). When a number cannot be represented by a given fixed-point data type with finite word-length, overflow on the most-significant-bit (MSB) or quantization on the least-significant-bit (LSB) or both take place. These finite-word-length effects cause the FP system behaves differently from the IP system, and this may result in implementation failure of an otherwise functioning IP algorithm.

Motivated to understand this important difference, theorists study finite-word-length effects using mathematical analysis, whereas most hardware designers do so by simulation methods. On the MSB side, overflow noise may cause recursive filters to

oscillate at 0-input, referred as limit cycle effects (see, for example, [4] and [10-12]). However, the effects of overflow noise are often understood purely based on simulations due to their possibly large magnitudes, low occurring probabilities, and strong correlations with signals [1-3][13-17]. We are not going further here on this largely open problem. Some analytical works related to MSB overflow effects take one step back and focus on mathematically predicting the signal statistics, such as variance and higher statistical moments, in an IP system, based on which overflow noises can be prevented [3][5].

In contrast, the finite-word-length effects on LSB side, also referred as quantization effects, are understood better largely benefited by their smaller magnitude, high occurring probabilities and weak correlations with signals. Since deterministic analysis of quantization noise is as difficult as studying overflow noise, theorists approach the problem almost purely statistically over the last few decades (see, for example, [18] for deterministic analysis). Given an IP algorithm, mathematical methods are applied to determine the statistical quantization effects at system output (or internal nodes) in relation to system parameters [3-5][19-51]. System parameters include architectural information such as the number of taps in a filter, system inputs such as constant filter coefficients and the signals to be processed, and FP parameters such as word-lengths and quantization modes. Despite some elegant successes addressing linear-time-invariant (LTI) systems and simple nonlinear systems, doing all these tasks simultaneously is often difficult that requires special care for each individual case, and still yields results that are too complicated to comprehend, such as infinite sums. We characterize quantization effects on FP parameters for a general DSP system based on a

small-signal perturbation theory and three assumptions—independent and white quantization noises, small noises, and no decision-error-propagation. This result covers both stationary and non-stationary inputs. Some large quantization effects are not studied, which is similar to the majority existing analytical work. Statistical quantization effects are described as functions of FP parameters only, together with some other unknown system parameters as simple coefficients that can be determined numerically. Even without determining the coefficients, the expression itself provides valuable insights on understanding quantization effects in general. Procedures to determine these coefficients analytically are also suggested.

On the other hand, pure pure-simulation-based approach to study quantization effects led by designers faces difficulties [13-17] as explained in [1]. Without any theoretical guidance, numerically analyzing quantization effects of FP parameters becomes a combinatorial problem. Its complexity is exponentially related to the number of FP parameters [1]. Moreover, simulation results often yield limited insight on how quantization effects depend on FP parameters systematically. With our results in this Chapter, the complexity of this numerical problem is reduced to a polynomial function of the number of FP parameters.

In Section 2-2, we prepare the basic terminologies and relationships for the rest of the chapter while briefly reviewing the vast existing work in the literature. The first assumption is introduced here. Section 2-3 categorizes the functional blocks and signals in a system according to their different quantization effects, and explains about the randomness of a system. Section 2-4 introduces two more assumptions that lead to our perturbation theory. Section 2-5 explains the numerical concerns of the theory. Examples

are given in Section 2-6, and we summarize in Section 2-7. Again, overflow noise is not studied, and only quantization effects due to roundoff quantization and truncation quantization are given explicitly, which are defined in previous chapter and depicted in next section.

2.2 Quantization basics and literature review

Quantization effects have been actively studied for more fifty years [4][7]. Transferring analog signals using digital-communication techniques and source-coding theory requires quantizations, and the objective is to find a quantization scheme that represents the source information efficiently and reliably [7]. A different quantization effect emerged in the early 1970's due to the raise of digital signal processing. The objective here is to understand how finite-precision (FP) signal processing differs from IP signal processing for a digital discrete-time signal processing system. It is important to discern these two topics when studying the literature. We concentrate on the second topic—quantization effects in DSP systems.

Though different number systems and quantization schemes are proposed for DSP, most implementations rely on binary fixed-point number system—either 2's complement or unsigned binary number—and quantization modes of either roundoff and truncation [4]. The rest of this chapter only explicitly studies the quantization effects of these two modes, which is relevant to current circuit implementation.

A quantizer in our discussion is uniquely characterized by two FP parameters—its binary point position relative to its least-significant-bit and its quantization mode being either roundoff or truncation. The binary point position is equivalent to the fractional

word-length, W_{Fr}, of a binary number. That is, the output can be represented as integer multiples of $2^{-W_{Fr}}$. $W_{Fr} > 0$ implies the quantized signal indeed has a fractional part. $W_{Fr} = 0$ implies the quantized signal can be any integer. And $W_{Fr} < 0$ means the quantized signal has its last $|W_{Fr}|$ integer bits always being 0's—thus not necessarily to be represented in hardware. A quantizer changes its input signal x into a quantized signal, denoted as Q[x]. Fig. 2-1 shows the two quantization scheme normalized by quantization step Δ, where, again,

$$\Delta = 2^{-W_{Fr}}. \tag{2-1}$$

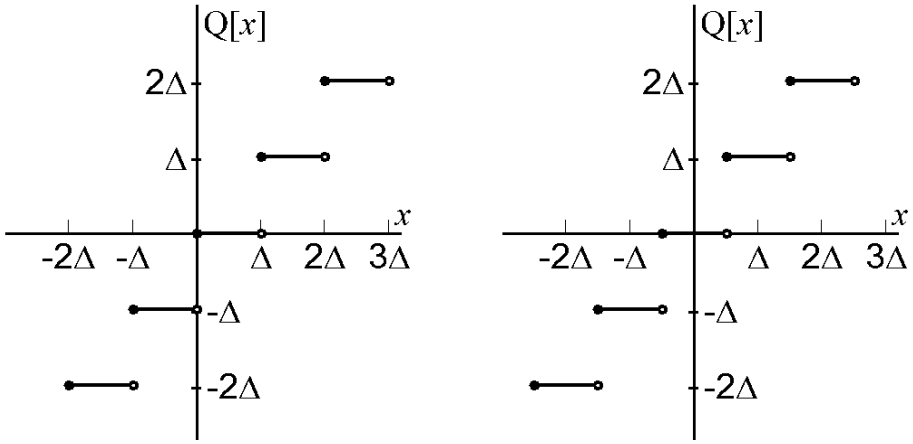


Fig. 2-1 Quantization function for (a) truncation, and (b) roundoff.
Here, $\Delta = 2^{-W_{Fr}}$.

Quantization noise, often referred as quantization error as well, is defined as the difference between quantizer output and input

$$e = \mathrm{Q}[x] - x. \tag{2-2}$$

The exact expression of ε after a quantizer with roundoff mode is

$$e_{\text{roundoff}} = \mathrm{fxpt}(x, \Delta) - x, \tag{2-3}$$

where $\mathrm{fxpt}(x,\Delta)$ is defined as $\mathrm{round}(x/\Delta)\cdot\Delta$, and round(.) is the function that maps its argument to its nearest integer. Similarly with truncation mode,

$$e_{\text{truncation}} = \mathrm{floor}(x/\Delta)\cdot\Delta - x, \tag{2-4}$$

where floor(.) is the floor function, or called greatest integer function. The error is related to W_{Fr} exponentially, that is, where e is on the order of Δ. So, as its name suggests, the noise is usually small when W_{Fr} is large. Fig. 2-2 depicts that a quantizer, as a nonlinear operator, is replaced by an adder with one input connected to the quantization noise, which clearly depends on its input.

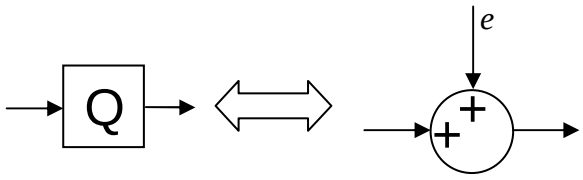


Fig. 2-2 Quantization error model

Because both roundoff and truncation modes partition the real axis into segments of equal distance, the associated quantization functions are periodic. Thus, a Fourier series can write them as an infinite sum of basic analytical functions—such as sums, divisions, and exponentials—of input signal x, as shown in [19] (see [20-24] for related discussions), that is

$$e_{\text{roundoff}} = \sum_{k\neq 0}\frac{(-1)^k\Delta}{j2\pi k}\exp(-2\pi\cdot j\frac{kx}{\Delta}) + \frac{\Delta}{2}\sum_{k=-\infty}^{\infty}\mathbf{1}(x=(2k+1)\frac{\Delta}{2}), \tag{2-5}$$

where 1(.) is the indicator function, which equals to 1 when its argument is true and 0 otherwise. Similar expression can be given for truncation error. Exact analyses of quantization error starting from (5) face serious challenges when the quantization error from one quantizer starts to feed into another one. One way to alleviate these vastly nonlinear effects is to study the quantization effects statistically. For example, taking expected value of a continuous random variable x on both sides of (5) gives the mean of roundoff error [19],

$$E[e_{\text{roundoff}}] = \Delta \sum_{k \neq 0} \frac{(-1)^k}{j2\pi k} \phi_x(\frac{2\pi k}{\Delta}), \tag{2-6}$$

where $\phi_x(.)$ is the characteristic function of x. Based on this approach, [19] studied how the mean and variance of the quantization noise at the output of a finite-impulse-filter (FIR) are related to its FP parameters. Though in this model an FIR only possesses at most two quantizers in any data path, the analyses have already been fairly complicated. The results are again expressed in infinite sums and can only be evaluated numerically, thus it reveals limited insights. This kind of exact analysis seems surreal and inadequate for systems containing long data paths and feedback loops.

Bershad and Bermudez extended the exact analysis when they studied the simplest adaptive filter—least-mean-square (LMS) algorithm [25-26]. To reduce the complexity, they assumed and numerically justified that, in many applications, a one-quantizer model in this nonlinear recursive system is sufficiently accurate. In addition to a popular Gaussian distribution assumption, some critical independence assumptions— or "constant assumptions", depending on how they are viewed—are used. As the result, the first two moments of the differences between actual filter weights and ideal Wiener filter

weights are obtained as iterations of infinite summations. Numerical computations of these iterations agrees well with simulations given the one quantizer model and independence assumption, but this analysis is of limited practical use and it is no longer "exact".

Almost in parallel to the exact theory, the majority of the community has been concentrated on purely statistical approach based on statistical assumptions of the noises. By giving up being "exact", these analyses proceed much further and provide many useful and often inspiringly accurate results. One of the most used assumptions is

> *Assumption 1*: A quantization noise is uniformly distributed in its possible range and independent (in some analysis, a weaker version is used assuming only uncorrelation instead of independence) with other signals, other quantization noises, and itself overtime (therefore it is white). Exceptions are constant signals whose quantization noises can be modeled as constants as well.

Examples of constant signals are filter coefficients of an FIR, step size parameter in an adaptive filter, and so on. These signals are deterministic and deserve special treatment. With Assumption 1, the quantization noise in Fig. 2-2 becomes independent to its signal and can be treated as additive noise statistically. Except for constant inputs, the noise is uniformly distributed between $[-\Delta,0)$ in truncation mode, whereas in roundoff mode the noise is uniformly distributed in $[-\Delta/2, \Delta/2)$. The mean value, u, of the noise can be summarized as

$$u \equiv E[e] = \begin{cases} e, \text{for constant } e \\ -\frac{1}{2} q \cdot 2^{-W_{Fr}} \quad \text{otherwise} \end{cases}, \tag{2-7}$$

where e is a constant when the signal being quantized is constant, and

$$q = \begin{cases} 0, \text{in roundoff mode} \\ 1, \text{in truncation mode} \end{cases}.$$

This is what we stated in eq. (1-4). When a signal or a system parameter, x, is known to be constant, its roundoff quantization value fxpt(x, Δ) as defined in eq. (2-3) is often used in an FP system, that is,

$$e_{\text{constant}} = \text{fxpt}(x, \Delta) - x. \tag{2-8}$$

The variance of the noise, on the other hand, is

$$\text{Var} = E[(e-\mu)^2] = \begin{cases} 0, \text{ for constant } e \\ \frac{1}{12} 2^{-2W_{Fr}}, \text{otherwise} \end{cases}. \tag{2-9}$$

The standard deviation s is $\sqrt{\text{Var}}$.

More accurate models on the mean and variance base on the weaker version of Assumption 1 and take into account that the signal coming from a previous DSP block is already in FP and thus not continuous [3][10][27]. Nevertheless, Assumption 1 is widely used because it models quantization noise efficiently. Many researchers have studied the mathematical condition for this assumption using either simulation or exact analysis [19-24]. For example, certain conditions on the characteristic function of the signal to be quantized make the quantization noise and signals exactly uncorrelated. However, experience shows that it is empirically sufficient to have the input random signals of much greater variance than the quantization noise and of reasonably wide spreads in both

value and frequency spectrum [5]. The first part can be easily satisfied when W_{Fr} becomes sufficiently large and the quantization error decreases exponentially according to (2-3) and (2-4). For most signals to be processed in communication systems and multimedia signal processing systems, signals are corrupted with various noises due to physical environment, referred as physical noises. Thus, the second part of the condition is also satisfied. Our favorite explanation of this condition is the following. From (2-6), when a random input has a much wider spread than Δ, it's characteristic function as its Fourier transfer should have values concentrated between $\pm\pi/\Delta$. So $\phi_x(\frac{2\pi k}{\Delta})$ becomes small except for k=0. This makes roundoff mean in (2-6) close to 0. Similarly, the variance becomes $(1/12)\Delta^2$ strictly. In the following analyses, Assumption 1 is considered strictly true.

Based on Assumption 1, quantization effects of linear-time-invariant (LTI) systems have been solved [3-5], mostly focusing on statistical quantities such as the first moment and the second moment of a signal. In these analyses, only the uncorrelation version of Assumption 1 is needed instead of the more restrict full independence condition. The quantization effects of an LTI system with stationary stochastic input can be summarized together into compact results [3-5][16]. This includes FIR, IIR (with feedback loops), FFT, and many other commonly used functional blocks in non-LTI systems. With Assumption 1, quantization noise occurred along signal data path in a linear system can be separated from input signals without any nonlinear interference, whereas quantization of constant gain coefficients is treated completely differently as deterministic modification of the system transfer functions.

Nonlinear systems such as adaptive filters, including LMS, Block LMS, RLS (recursive least square), and leaky LMS, are also studied extensively [6][28-41]. Because of the nonlinear feedback loop, the output and internal signals contain contributions from multiplications of input signals from various time instances. Analysis cannot proceed unless further assumptions are made on quantization noise and also on the statistical property of the input signals themselves. Efforts in [40] results in a statistical "energy preserving equation", from which calculations about quantization effects (and analysis for IP algorithms in general) of a time-domain LMS system are eased. Some other nonlinear systems are also studied assuming the nonlinear part of the system is not apparent [42-45], such as for CORDIC (coordinate rotation digital computer) system [42]. Yet a general theory on quantization effects on nonlinear systems is not available. Furthermore, it is known that small modifications in system architecture alters quantization effects [1][4] and requires new derivation from the beginning despite all the existing results for similar systems. The situation gets worse as the system is implemented with a top-down design flow, because a complete analysis on quantization effects requires complete understandings of both algorithm and architecture. But algorithm, IP architecture, and FP architecture are often designed by separate designers, causing a communication problem.

The quantization effects associated with the non-stationary input signals become more complicated to analyze, requiring simplified statistical models, such as Markov chain [40-41]. However, the accuracy of the results suffers due to more aggressive simplifications.

As mentioned in Section 2.1, the difficulties met in nonlinear systems are caused by the inefficiency to separate the quantization effects from other complicated ones existing in IP systems. As a result, it is widely admitted that analyses on quantization effects are more advanced and complicated than those for pure IP systems. However, with a couple more assumptions that could be strictly satisfied, we will show that general understanding of quantization effects is indeed possible.

Aside from the analytical approach, advancement in computing promotes simulation-based approach [13-17]. However, lack of understanding, the number of numerical estimations to completely characterize the quantization effects is exponential with respect to the number of quantizers. Suppose a system has L quantizers, each of which has its fractional word-lengths chosen from l possible values and 2 quantization mode, then $(2l)^L$ estimations are required in the characterization task—usually too high to realize. For example, in the process of the classical floating-point to fixed-point conversion, an intuition, stating that higher fractional word-lengths always result to "better" systems, may reduce the estimation complexity greatly [15][17]. We will validate this by giving a sufficient condition in Section 2.7 as an example of our theory. Furthermore, based on our results in this chapter, we propose that only orders of L^2 number of estimations are needed to completely characterize the statistical quantization effects numerically. The complexity is independent to l. Our floating-point to fixed-point conversion tool is largely benefited by this conclusion [1].

2.3 Preparation for perturbation theory

2.3.1 Categorization of signals and blocks

Quantization effects depend on the architecture of the system implementation [1][4]. For example, it is well-known that finite-impulse filters (FIR) implemented in direct form I and direct form II produce the same result in infinite precision arithmetic, yet they have different quantization effects [4]. In fact, the number of quantizers is usually different in various finite-precision (FP) implementations, and they may locate at different places. The most natural way to include all the system description necessary for understanding quantization effects is starting from the structural description, as suggested in [1]. In a structural description, a large system is constructed by smaller functional units such as adders, multipliers, multiplexers, and so on; infinite-precision (IP) or FP systems with architectural information look like a block diagram. Fig. 1-1 in previous chapter depicts a simple algorithm in its architectural form.

In order to identify the different types of functional modules, we first differentiate the types of signals in a DSP system. This is done by comparing the FP system from its IP version. Both IP and FP descriptions of a system can be viewed as different levels of abstractions of a physical design. A FP system can be represented by the same block diagram as the IP system, with some quantizers inserted in the signal paths or after some constants. After each quantizer, the signal is changed from infinite-precision (or higher precision) to finite-precision (or lower precision). We name this kind of signals *arithmetic signals*—new quantizations happen right after them in the FP version of the system.

Some other signals are already in their desired finite-precision even in the IP description. This is possible because of algorithm designers' understanding of Boolean algebra, coding theory, abstract algebra, and other mathematical theory about FP arithmetic. Hence, it is unnecessary to place additional quantizers for these signals in the FP version. Doing so, the IP algorithm designers' vision would be ignored which results in different systems. As an example, one may combine 8 1-bit signals after source coding into an 8-bit number with four of the bits being fractional. This can be done by a serial-to-parallel converter and by thinking that the binary point being in the middle. Then, it would be completely wrong to treat this signal arithmetic and quantize it because the information associated with the bits truncated away is permanently lost. We call these signals that have predetermined FP data-type *logical signals*. In fact, most of them are naturally described in Boolean or binary integer format, and IP system designers can identify them easily.

Some other constraints can also cause predetermined FP data-type, for example, when having limited hardware resources such as fixed precision analog-to-digital converter, multipliers, and so on. In order to develop an algorithm to be implemented using these hardware, it is practical to treat signals after these blocks also of fixed finite-precision. We will not distinguish these *fixed data-type signals* from logical signals.

With the preceding signal-type description, operators (or functional blocks) and its inputs and output in IP system can be summarized into different types as well. Here, we assume one operator always have one output because an operator with multiple outputs can be separated into multiple copies with single output individually.

1 The output is an arithmetic signal. We name the operator *arithmetic operator*. Examples include adder, multiplier, and delay elements as appeared in FIR and LMS filter.

2 All of the inputs are logical, and the output is also logical. The operator is called *logical operator*. These operators often appear in control logic circuits. Examples are AND/NAND/OR gates.

3 Some of the inputs are arithmetic, and the output is logical. The operator is called *decision-making operator*. Examples include the final slicer in a communication system that estimates the transmitted bit, a comparator in a time-synchronization unit to select the right time-offset, and a comparator in a stage-based CORDIC unit to decide in which direction the angle needs to be shifted at a particular stage.

Let us explain this classification. It may first look non-trivial to discern signal types and thus operator types. For example, an adder with arithmetic output (with potentially infinite-precision) is by definition an arithmetic operator. Yet an adder included in control logics such as finite-state machine description that produces strict logic signal output is a logical operator. On the other hand, an adder as an arithmetic operator can also be decomposed into logical operators such as NAND gates operating on logical signals. In this way, architecture description has been driven into too much detail for analysis of quantization effects. This is because modifying the FP parameters now means cutting blocks from (or adding blocks to) the architectural description. Therefore, fixing the architectural description rules out this kind of confusions. As another example, an absolute function denoted as |.| operating on an arithmetic signal is naturally treated as an arithmetic operator. However, as shown in Fig.1-1, it may compose a multiplexer selecting the input signal or input signal's negation based on its sign, and the sign equivalent to a slicer—a decision-making block.

From another point of view, because infinite-precision operations do not exist in implementation, why don't designers only design finite-precision system from the very beginning? The answer is that this is too "abstract" (in the same sense as it appears in Abstract Algebra) for the designers, and the large combination of FP parameters make the task easily forbidding. Therefore, IP description normally starts as the first level of abstraction, which contains mostly infinite-precision operations that are easier for analysis. Then, architectural information is added, and then FP parameters are considered. Yet from the beginning, IP designers might know some signals' final FP types, either inferred by algorithm itself (such as for the slicer in communication systems, and for the source-coding), external reasons (such as component availability in hardware), or even design experience. The last "knowledge"-based approach is not recommended: more scientific reasoning and more advanced design tools can do better [1].) Usually, logical signals are those that become wrong when shortening its word-length and become awkward and unnecessary to increase; in contrast, the FP parameters of arithmetic signals affect the behavior of a system much more incrementally.

In summary, signals types are often confined by and apparent from algorithmic and architectural descriptions of a system. If not, the IP system designers know who they are.

2.3.2 Some definitions

The inputs of a digital system can be considered as discrete-time random process [1][3][6][46]. The operators in a system can usually be treated as deterministic operations that produce random process at their output under this stochastic signal environment. At a

given sample time t, each input signal, internal signal, or output signal is a random variable—with values from different ensemble realizations will follow a probability density function at t. The function could be different at different t, that is, signals throughout the system may be non-stationary random processes.

In reality, an operator, denoted by bold letter **F**, has a finite number of inputs, denoted as K, and they together form a random vector point $(x_1(t), x_2(t), \ldots, x_K(t))$ or, in a simpler notation, $(x_1, x_2, \ldots, x_K)_t$, at time t. This random vector may be non-stationary. An operator could be as small as an adder or as large as a complete communication system. Inputs include normal data-path inputs such as the inputs of an adder, or constant inputs such as the constant coefficients of an LTI filter or adaptive update parameter. At t, any ensemble realization of the random vector $(x_1, x_2, \ldots, x_K)_t$ becomes a regular vector and belongs to a domain Ω_t. Here, Ω_t consists of all the possible realizations at time t and is a subset of the K-dimensional Euclidean space R^K if we treat Boolean signals True and False as real numbers, such as 1 or 0, respectively. For convenience, the random vector $(x_1, x_2, \ldots, x_K)_t$ is considered to have domain Ω_t, that is,

$$(x_1, x_2, \ldots, x_K)_t \in \Omega_t. \tag{2-10}$$

Let $t > t_1 > t_2 > \ldots > 0$, and suppose the system starts to run at $t = 0$, then at a later time t, the output of a causal operator F depends on all its previous and current inputs, $\{(x_1, x_2, \ldots, x_K)_t, (x_1, x_2, \ldots, x_K)_{t_1}, \ldots, (x_1, x_2, \ldots, x_K)_{t_N}, (x_1, x_2, \ldots, x_K)_0\}$, and possibly a random initial state. In a sample based system, t_i is simply t-i. Variables in the initial state are treated as additional inputs to the system which are brought into the system by adders at time 0. So, for conciseness, let's ignore the initial state in subsequent discussion. Now,

the output of **F** at time t as $f_{\mathbf{F}}(x_1, x_2, \ldots, x_K, t)$, called the transfer function of **F**, can be written as

$$f_{\mathbf{F}}(x_1, x_2, \cdots, x_K, t) = \phi_{\mathbf{F},t}\left((x_1, x_2, ..., x_K)_t, (x_1, x_2, ..., x_K)_{t_1}, ..., (x_1, x_2, ..., x_K)_0\right). \quad (2\text{-}11)$$

The functionality of **F** at time t is uniquely expressed by function $\phi_{\mathbf{F},t}$, named as the instantaneous transfer function of **F** at t. It is convenient to rename the input K×(N+2)-dimensional random vector $\{(x_1, x_2, \ldots, x_K)_t, (x_1, x_2, \ldots, x_K)_{t_1}, \ldots, (x_1, x_2, \ldots, x_K)_{t_N}, (x_1, x_2, \ldots, x_K)_0\}$ to $\{\xi_1, \cdots, \xi_M\}$ as

$$\xi_1 = x_1(t), \xi_2 = x_2(t), \ldots, \xi_K = x_K(t),$$

$$\xi_{K+1} = x_1(t_1), \xi_{K+2} = x_2(t_1), \ldots, \xi_{2K} = x_K(t_1),$$

$$\ldots$$

$$\xi_{M-K+1} = x_1(0), \xi_{M-K+2} = x_2(0), \ldots, \xi_M = x_K(0), \quad (2\text{-}12)$$

where M= K×(N+2). $(\xi_1, \cdots, \xi_M)$ are called the expanded variables of $(x_1, x_2, \cdots, x_K)$ at time t with respect to operator **F**. Now, (2-11) reduces to

$$f_{\mathbf{F}}(x_1, x_2, \cdots, x_K, t) = \phi_{\mathbf{F},t}(\xi_1, \xi_2, ..., \xi_M). \quad (2\text{-}13)$$

Only with an ensemble realization of random vector $(\xi_1, \cdots, \xi_M)$, a numerical value of the output at t becomes available using (13). By the definition in (10) and (12), the expanded variables belong to a expanded domain, that is $(\xi_1, \cdots, \xi_M) \in \Omega_t \times \Omega_{t_1} \times \ldots \times \Omega_0$, where "×" means direct product.

It is important to notice that f_F as a function is deterministic. When a system is physically designed, it is often to fulfill a deterministic functionality to process some random (or, sometimes, deterministic) inputs. This is generally accepted in modeling all

DSP systems; therefore, this is not made as a separated assumption in addition to Assumptions 1-3. In $\phi_{\mathbf{F},t}$, the subscript t stresses that this deterministic function itself may vary over time. The technical difficulty of explicitly expressing this deterministic function is not crucial to understand the rest of the chapter.

The examples below explain the definitions so far.

Example 1. An adder operator A at time t has output transfer function

$$f_{\mathbf{A}}(x_1, x_2, t) = x_1(t) + x_2(t),$$

so,

$$f_{\mathbf{A}}(x_1, x_2, t) = \phi_{\mathbf{A},t}(\xi_1, \xi_2, ..., \xi_{\mathrm{M}}) = \xi_1 + \xi_2. \tag{2-14}$$

A multiplexer, denoted by operator M, selects either of the two inputs, x_1 or x_2, to its output depending on the value of the third input—x_{sel}; so its transfer function is

$$f_{\mathbf{M}}(x_1, x_2, x_{\text{sel}}, t) = \mathbf{1}(x_{\text{sel}}(t) = 0) \cdot x_1(t) + \mathbf{1}(x_{\text{sel}}(t) = 1) \cdot x_2(t),$$

That is,

$$\begin{aligned} f_{\mathbf{M}}(x_1, x_2, x_{\text{sel}}, t) &= \phi_{\mathbf{M},t}(\xi_1, \xi_2, ..., \xi_{\mathrm{M}}) \\ &= \mathbf{1}(\xi_3 = 0) \cdot \xi_1 + \mathbf{1}(\xi_3 = 1) \cdot \xi_2. \end{aligned} \tag{2-15}$$

where x_{sel} is a logical signal, and 1(.) is the indicator function as defined in (2-5).

■

Example 2. A timing operator **G** has its output at *t* equal to its single input at another time *g*(*t*), where $g(t) \le t$ for a causal G. That is,

$$f_{\mathbf{G}}(x, t) = x\big(g(t)\big),$$

so,

$$f_{\mathbf{G}}(x,t) = \phi_{\mathbf{G},t}(\xi_1,...,\xi_{\mathrm{M}}) = \xi_{t-g(t)+1}\,. \tag{2-16}$$

More specifically, a unit delay z^{-1} has $g(t) = t-1$, and its instantaneous transfer function is,

$$\phi_{\mathbf{z}^{-1},t}(\xi_1,...,\xi_{\mathrm{M}}) = \xi_2\,.$$

On the other hand, a 2-times down-sampler samples the input at every even time event (here 2 is chosen for clarity and without loss of generality) has

$$g(t) = t\cdot\mathbf{1}(\mathrm{mod}(t,2)=0) + (t-1)\cdot\mathbf{1}(\mathrm{mod}(t,2)=1),$$

where mod(*a*, *b*) gives the remainder of integer a divided by integer *b* and the underlying discrete clock is the one before the down-sampler (because it is of higher frequency) . So

$$\begin{aligned} &f_{\downarrow\mathbf{2}}(x,t) = \phi_{\downarrow\mathbf{2},t}(\xi_1,...,\xi_{\mathrm{M}}) \\ &= \xi_{t-\{\mathbf{1}(\mathrm{mod}(t,2)=0)\cdot t-\mathbf{1}(\mathrm{mod}(t,2)=1)\cdot(t-1)\}+1} \\ &= \xi_{\mathbf{1}(\mathrm{mod}(t,2)=1)+1}, \end{aligned}$$

where the last step is because $\mathbf{1}(\mathrm{mod}(t,2)=1) + \mathbf{1}(\mathrm{mod}(t,2)=0)$ is just 1. Alternatively, we can understand the last equation as

If mod(t,2)=0,

$$f_{\downarrow\mathbf{2}}(x,t) = \phi_{\downarrow\mathbf{2},t}(\xi_1,...,\xi_{\mathrm{M}}) = \xi_1,$$

otherwise, that is, if mod(t,2)=1,

$$f_{\downarrow\mathbf{2}}(x,t) = \phi_{\downarrow\mathbf{2},t}(\xi_1,...,\xi_{\mathrm{M}}) = \xi_2\,.$$

■

In summary, all the random factors of the output of an operator are introduced by its random inputs, whereas the transfer function $\phi_{\mathbf{F},t}$ is deterministically known (as a function of time *t*).

2.3.3 Smooth operators

The concept of "smoothness" of an operator provides the foundation of the perturbation theory to be introduced later. As defined in calculus, deterministic function $\phi_{\mathbf{F},t}$ is said to be smooth on arithmetic signals if it is continuous and differentiable to any desired degree over an open set in which the arithmetic signals belong to, regardless of the realizations of the logical signals. Then, operation **F** is called smooth over its arithmetic inputs, or briefly as **F** is smooth. Discussions later show differentiability to the third degree is usually sufficient in consideration of quantization effects.

It is meaningless to say the operator smooth or not over its logical signal inputs as they have discrete values. Basic arithmetic operators, such as an adder, a multiplier, and so on, clearly are smooth. In Example 1, for a multiplexer whose functionality is defined by (2-15), $\phi_{\mathbf{M},t}$ is clearly smooth on its arithmetic signals (ξ_1, ξ_2) over all their possible values. So, multiplexer is indeed smooth. In Example 2, timing operator is evidently smooth according to (2-16) if the input is arithmetic. The following simple example explains the domain involved in the definition.

Example 3. A reciprocal operator R is given by its transfer function as

$$f_{\mathbf{R}}(x,t) = \phi_{\mathbf{R},t}(\xi_1,...,\xi_{\mathrm{M}}) = 1/\xi_1, \tag{2-17}$$

where ξ_1 is arithmetic signal. Clearly, $\phi_{\mathbf{M},t}$ is smooth if its input belongs to $(-\infty,0)\cup(0, +\infty)$. Similarly an absolute-value operator is also smooth in the same domain. In fact, most mathematical operations, such as sine, cosine, logarithm, exponential and power, are all smooth with properly defined input domain.

■

In our definition, decision-making operators with arithmetic input and logical output can also be treated as smooth operators over its arithmetic inputs. The unsmooth region contains those arithmetic input vector points that, when adding some infinitesimal perturbation at different direction, produce different decisions at the output. For example, a slicer that determines the sign of an input arithmetic signal has unsmooth region containing one point, 0.

Two consequential smooth operators form a combined operator that is also smooth. This is simply because smooth function acting on smooth function provides a joint smooth function. The smooth domain for this combined operator is usually those inputs that cause neither of the two operators to operator at unsmooth region.

2.4 Perturbation theory

Section 2.2 motivates us to understand quantization effects of a general system, similar to what has been successfully done for LTI systems. When the systems and signals satisfy two additional assumptions that are given in the first part of this section, a theory based on perturbation fulfills this task.

2.4.1 Limit large quantization effects

First, in addition to Assumption 1 in Section 2.2 that treats quantization noises as separate and independent system inputs, further regulations on the noise magnitude helps. This gives

> *Assumption 2*. Only the effects caused by small quantization noises in comparison with the magnitudes of their corresponding IP signals are considered.

Since quantization noise introduced by a quantizer is strictly bounded by quantization step Δ in (2-3) and (2-4), Assumption 2 is normally well-satisfied when fractional wordlength of the quantizers in the system are large since Δ decrease exponentially as the fractional word-length increase.

One consequence of Assumption 2 is that large quantization noises due to aggressive quantizers are ignored. Those quantizers may cause the FP system behave significantly differently from the IP system. Such a system would most likely violate the IP designers' original visions on the algorithm, so that it would be more properly considered as a new algorithm, rather than an approximation of the IP system. For example, an adaptive sign-algorithm (SA) is almost identical to LMS, except that it takes the sign of the error signal (a 1-bit quantizer) rather than the full error signal to feedback and update the filter tap weights [47][40]. In literature, they are indeed treated as two algorithms. So, Assumption 2 basically confines us to understand on FP system who behaves slightly different from its IP version. However, this analysis provides insights on explaining some phenomena in aggressively designed FP systems.

2.4.2 Limit quantization effects caused by altered decisions

By definition, only arithmetic signals are modified directly by FP quantizers. Nevertheless, the value of a logical signal in a FP system can indirectly vary from its counterpart in IP system. Quantization effects from quantizers that modify only arithmetic signals may accumulate in front of a decision-making block. These arithmetic signals influenced so much that the decision of a decision-making block may be altered. This altered decision can further propagate through control logic operators and alter the

value of any logic signal following them. They can also bring large magnitude errors to arithmetic signals. This important mechanism is called decision-error-propagation. It brings more subtle quantization effects, under which the FP system may still act as an approximation of the IP one with occasional large deviations. These effects are much more difficult to study. We rule out them by having Assumption 3,

> *Assumption 3*. In a causal discrete-time system, assume each of the arithmetic operators and decision-making operators has its arithmetic inputs sitting in smooth regions of the operator, that is, the probability that its arithmetic inputs occur in the "unsmooth" region is zero.

In Example 3, this simply translate to probability $P(\xi_1 = 0) = 0$. In practice, a safer version might be $P(-\delta < \xi_1 < \delta) = 0$, where δ is an arbitrarily small positive number.

One immediate inference of Assumption 3, together with Assumption 2, is that, as long as quantization noises are sufficiently small, logical signals can not alter due to the aforementioned decision-error-propagation mechanism in a FP system. Under arbitrarily small quantization effects, arithmetic inputs of an arithmetic operator can only change in an infinitesimally small neighbor around its IP point, which is still included in its open smooth region. So its output also changes infinitesimally small amount around its IP value. (Here need the operator function continuous.) Similarly, a decision-making block keeps its output value same as IP one because its inputs only change little from their IP value, sitting in the open smooth region. The whole system, even with recursive loop, will operate nicely similar to its IP version. In this way, only arithmetic operators propagate quantization effects for any finite time t. Therefore, the values of logical

signals in FP system are always the same as in IP system at any time instance before and including the current time.

Most existing analytical studies on quantization effects focus on systems that satisfy Assumption 3. A large number of these systems do not even contain logical signals, thus no decision-making blocks and logical blocks either. For example, linear-time-invariant (LTI) systems consist of constant gains, adders and delays, whereas least-mean-square (LMS) and recursive-least-square (RLS) systems contain multipliers in addition, yet only arithmetic operators are involved. Furthermore, all these arithmetic operators are basic and smooth. On the other hand, a CORDIC system does include decision-making and logical operators, but its quantization effects has been studied implicitly assuming no logical signals is different between the FP and IP systems [42].

2.4.3 View FP the same as IP system, but with different noise inputs

As stated at the end of Section III, a system can be treated as a combined operator. Furthermore, Assumptions 2 and 3 infer that the operator is smooth on its arithmetic signals. Since all the internal operators operate in their smooth region, the combined operator also does the same thing.

Denote the original IP system as S_{IP} and the FP system as S_{FP}. Assumption 1 says that a quantizer can be treated as an adder with the quantizer input and independent error noise as two of its inputs. In this way, one additional input per quantizer is brought into the system with an additional arithmetic adder. By replacing all quantizers with adders in the FP system, we get a new system, called $\mathcal{S}$. This system $\mathcal{S}$ can represent both the IP system, S_{IP}, by setting the noise inputs of these adders constantly 0, whereas $\mathcal{S}$ can

represent S_{FP} as well by having the noise inputs the corresponding noise sources. Fig. 4 depicts these relationships. Assumption 1 allows us to treat IP and FP versions of a system simply as the same system $\mathcal{S}$, but with different inputs. Here $\mathcal{S}$ differs from S_{IP} since the former contains additional adders.

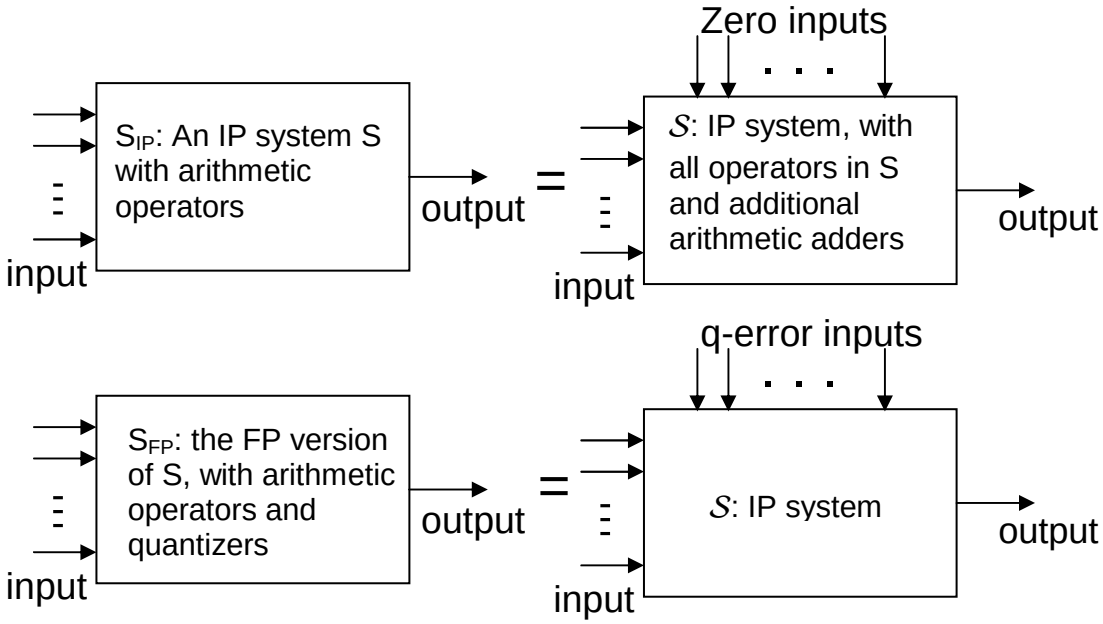


Fig. 2-3 FP system S_{FP} is treated as an IP system with changes on some error input signals.

Let bold letters $\boldsymbol{s}$, S_{IP} and S_{FP} be the operators associated with $\mathcal{S}$, S_{IP} and S_{FP}, respectively. Then $\boldsymbol{\mathcal{S}}$ is a combined operator of all those in S_{IP} and the smooth adder-operators replacing quantizers. When all quantization noise inputs are 0, the internal signals of $\mathcal{S}$ are identical to those in S_{IP}. Because quantization noises only modify arithmetic signals under Assumptions 2 and 3, $\mathcal{S}$ is smooth on both the original arithmetic inputs of S_{IP} and on the quantization noise inputs.

Denote the transfer function of $\mathcal{S}$ as $f_{\mathcal{S}}$, its signal inputs as (x_1, x_2, ⋯, x_K), and the error inputs as (e_1, e_2, ⋯, e_L), where L is the number of additional quantizers in S_{FP} comparing with S_{IP}. The expanded variables of (x_1, x_2, ⋯, x_K) and (e_1, e_2, ⋯, e_L) over $\mathcal{S}$ are denoted as (ξ_1, ⋯, ξ_M) and (ε_1, ⋯, ε_N), respectively. Since (ε_1, ⋯, ε_N) is simply a rearrangement of (e_1, e_2, ⋯, e_L) at different discrete time instances, (ε_1, ⋯, ε_N) in the IP system should be constant 0's.

Now, the transfer function of S_{FP} and S_{IP} can be stated in terms of the transfer function of $\mathcal{S}$,

$$\begin{aligned} & f_{\mathbf{S_{FP}}}(x_1, x_2, \cdots, x_K, t) \\ &= f_{\mathcal{S}}(x_1, x_2, \cdots, x_K, e_1, e_2, \cdots, e_L, t) \\ &= \phi_{\mathcal{S},t}(\xi_1, \cdots, \xi_M, \varepsilon_1, \cdots, \varepsilon_N), \end{aligned} \tag{2-18}$$

and

$$\begin{aligned} & f_{\mathbf{S_{IP}}}(x_1, x_2, \cdots, x_K, t) \\ &= f_{\mathcal{S}}(x_1, x_2, \cdots, x_K, 0, 0, \cdots, 0, t) \\ &= \phi_{\mathcal{S},t}(\xi_1, \cdots, \xi_M, 0, \cdots, 0). \end{aligned} \tag{2-19}$$

The next Subsection will use the differentiability part in Assumption 3 to study the difference between (2-18) and (2-19).

2.4.3 Taylor expansion

All the discussion in previous two subsections could still hold if "continuous" was to substitute "smooth" in Assumption 3. A continuous operator's output only change little if its inputs change little. The differentiability in Assumption 3, however, enables

quantitative study on infinitesimal quantization effects. This is done by using Taylor expansion of smooth function $\phi_{\boldsymbol{S},t}(\xi_1,\cdots,\xi_{\mathrm{M}},\varepsilon_1,\cdots,\varepsilon_{\mathrm{N}})$ over its arithmetic inputs signals (ε_1, $\cdots$, ε_{N}) around their IP values that are all zeros. The expansion up to its 2nd-order terms gives

$$\begin{aligned}&\phi_{\boldsymbol{S},t}(\xi_1,\cdots,\xi_{\mathrm{M}},\varepsilon_1,\cdots,\varepsilon_{\mathrm{N}})\\&=\phi_{\boldsymbol{S},t}(\xi_1,\cdots,\xi_{\mathrm{M}},0,\cdots,0)+\sum_{i=1}^{\mathrm{N}}\left(\partial\phi_{\boldsymbol{S},t}\big/\partial\varepsilon_i\right)_{\varepsilon_1=0,\cdots,\varepsilon_{\mathrm{N}}=0}\cdot\varepsilon_i\\&\quad+\sum_{i,j=1}^{\mathrm{N}}\left(\partial^2\phi_{\boldsymbol{S},t}\big/\partial\varepsilon_i\partial\varepsilon_j\right)_{\varepsilon_1=0,\cdots,\varepsilon_{\mathrm{N}}=0}\cdot\varepsilon_i\varepsilon_j+....\end{aligned}\tag{2-20}$$

Using (2-18) and (2-19) on both side of the equation, (20) becomes

$$\begin{aligned}&f_{\mathbf{S_{FP}}}(x_1,x_2,\cdots,x_{\mathrm{K}},t)\\&=f_{\mathbf{S_{IP}}}(x_1,x_2,\cdots,x_{\mathrm{K}},t)+\sum_{i=1}^{\mathrm{N}}\left(\partial\phi_{\boldsymbol{S},t}\big/\partial\varepsilon_i\right)_{\varepsilon_1=0,\cdots,\varepsilon_N=0}\cdot\varepsilon_i\\&\quad+\sum_{i,j=1}^{\mathrm{N}}\left(\partial^2\phi_{\boldsymbol{S},t}\big/\partial\varepsilon_i\partial\varepsilon_j\right)_{\varepsilon_1=0,\cdots,\varepsilon_N=0}\cdot\varepsilon_i\varepsilon_j+....\end{aligned}\tag{2-21}$$

Thus the output of the FP system at time t has been expressed as the IP output with additional small perturbations due to all the quantization noises in a power series format. The coefficients of the power series are no longer functions of the noises themselves.

With Assumption 2 and the smooth operator assumption, the higher order terms are normally negligible comparing with the first two orders. Only the first two orders of terms are kept for relatively easy analysis.

2.4.4 Statistical quantization effects

Both the coefficients and error noises are stochastic signals. Thus, deterministic studies can only be conducted in ways such as the absolute bounds of the difference between IP and FP system.

Commonly, it is the statistics of an output that is most informative. Studies such like direct analysis of the probability distribution functions may provide insights. Though useful, the analysis is very hard to get useful results. Yet study on statistical expectations provides a good tradeoff between the complexity and usefulness.

With Assumption 1, entries of (ε_1, $\cdots$, ε_N) are mutually independent and are independent to (ξ_1, $\cdots$, ξ_M); so, in (2-21), the power terms of (ε_1, $\cdots$, ε_N) and coefficients terms that only are functions of (ξ_1, $\cdots$, ξ_M) are statistically independent. So, doing expectation of (2-21) on both sides and using identity $E[a\cdot b]=E[a]\cdot E[b]$ for independent random variables a and b, it gives

$$\begin{aligned} & E[\, f_{\mathbf{S_{FP}}}(x_1, x_2, \cdots, x_K, t)] \\ &= E[\, f_{\mathbf{S_{IP}}}(x_1, x_2, \cdots, x_K, t)] + E[\sum_{i=1}^{N} \left(\partial \phi_{\mathbf{S},t} \big/ \partial \varepsilon_i \right)_{\varepsilon_1=0,\cdots,\varepsilon_N=0}] \cdot E[\varepsilon_i] \\ &+ E[\sum_{i,j=1}^{N} \left(\partial^2 \phi_{\mathbf{S},t} \big/ \partial \varepsilon_i \partial \varepsilon_j \right)_{\varepsilon_1=0,\cdots,\varepsilon_N=0}] \cdot E[\varepsilon_i \varepsilon_j], \end{aligned} \tag{2-22}$$

where only the first two terms are kept. Note that

$$\begin{aligned} & E[\varepsilon_i \varepsilon_j] \\ &= E[\varepsilon_i] E[\varepsilon_j] + E\left[\varepsilon_i - E[\varepsilon_i]\right] \cdot E\left[\varepsilon_j - E[\varepsilon_j]\right], \\ &= \mu_i \mu_j + \gamma_{i,j} \cdot \sigma_i \sigma_j \end{aligned} \tag{2-23}$$

where $\mu_i = E[\varepsilon_i]$ is the mean of ε_i,

$$\sigma_i = \sqrt{E[(\varepsilon_i - E[\varepsilon_i])^2]} \tag{2-24}$$

is the standard deviation of ε_i, and

$$\gamma_{i,j} = \frac{E[\varepsilon_i - \mu_i] \cdot E[\varepsilon_j - \mu_j]}{\sigma_i \sigma_j} \tag{2-25}$$

is the auto-correlation coefficient between ε_i and ε_j. Due to Cauchy-Schwartz inequality [48], $|r_{i,j}| \le 1$ always holds. Furthermore, (2-23) can be simplified by noticing that (ε_1, $\cdots$, ε_N) are mutually independent as stated in Assumption 1. That is, $\gamma_{i,j} = 0$ if $i \neq j$, and $r_{i,j} = 1$ if $i = j$. Thus, (2-22) becomes

$$\begin{aligned} & E[f_{\mathbf{S_{FP}}}(x_1, x_2, \cdots, x_K, t)] \\ &= E[f_{\mathbf{S_{IP}}}(x_1, x_2, \cdots, x_K, t)] + \sum_{i=1}^{N} E\left[\left(\partial \phi_{\mathbf{S},t} \Big/ \partial \varepsilon_i\right)_{\varepsilon_1 = 0, \cdots, \varepsilon_N = 0}\right] \cdot \mu_i \\ &+ \sum_{i}^{N} E\left[\left(\partial^2 \phi_{\mathbf{S},t} \Big/ \partial \varepsilon_i^{\,2}\right)_{\varepsilon_1 = 0, \cdots, \varepsilon_N = 0}\right] \cdot \sigma_i^{\,2} \\ &+ \sum_{i,j=1}^{N} E\left[\left(\partial^2 \phi_{\mathbf{S},t} \Big/ \partial \varepsilon_i \partial \varepsilon_j\right)_{\varepsilon_1 = 0, \cdots, \varepsilon_N = 0}\right] \cdot \mu_i \mu_j . \end{aligned} \tag{2-26}$$

It is more straightforward to represent the expression in terms of the statistics of quantization noises (e_1, e_2, $\cdots$, e_K) directly. To do this, define the mean and standard deviation for e_i according to (2-7) and (2-9) as

$$u_i = E[e_i], \text{ and } s_i = \sqrt{E[(e_i - u_i)^2]}. \tag{2-27}$$

These statistics can be related to the quantization mode and fractional word-length using (2-7)-(2-9). Now, replacing all μ_i's and σ_i's in (2-26) with corresponding u and s

according to the definition of expanded variables in (2-12), possibly repeatedly. Collecting all the coefficients in front of the same u_i, s_i and $u_i u_j$, denoting the final coefficients to m_i, h_i and $n_{i,j}$, respectively, and moving the first term of the right side of (2-27) to the left side, we get

$$\boxed{\begin{aligned} &E[f_{\mathbf{S_{FP}}}(x_1, x_2, \cdots, x_K, t) - f_{\mathbf{S_{IP}}}(x_1, x_2, \cdots, x_K, t)] \\ &= \sum_{i=1}^{L} m_i(t) \cdot u_i + \sum_{i}^{L} h_i(t) \cdot s_i^{\,2} + \sum_{i,j=1}^{L} n_{i,j}(t) \cdot u_i u_j . \end{aligned}} \tag{2-28}$$

Here the summation is from 1 to L, rather than 1 to N used for expanded variables in (2-26). Furthermore, (2-28) is a deterministic relationship where every term on the right side is no longer random. This central result (2-28) is worth interpreting below. Examples in Section 2-6 provide further clarifications.

First, if none of m_i's degenerates to 0, Assumption 2 infers that $u_i << 1$ and the last summation of (2-28) is always negligible comparing with the first summation. However, the summation containing s_i^2 cannot be neglected since $u_i = 0$ strictly in roundoff quantization mode, in which case those terms dominate the perturbation. So,

$$\begin{aligned} &E[f_{\mathbf{S_{FP}}}(x_1, x_2, \cdots, x_K, t) - f_{\mathbf{S_{IP}}}(x_1, x_2, \cdots, x_K, t)] \\ &\approx \sum_{i=1}^{L} m_i(t) \cdot u_i + \sum_{i=1}^{L} h_i(t) \cdot s_i^{\,2} . \end{aligned} \tag{2-29}$$

This relation was first reported in [2] without giving the proof there.

Second, the detailed expressions of the coefficients are not given. These expressions, together with the value of L, summarize the statistics of the input signals, and the algorithmic and architectural information of the system. All the FP parameters, such as fractional word-lengths and quantization modes, on the other hand, are exhibited

in the mean and variances given by (2-27). That is, (2-28) separates quantization effects from IP characteristics. Furthermore, our derivation actually gives a procedure to find out the coefficients expression explicitly. This includes getting system $\mathcal{S}$ as discussed in Fig. 2-3, changing (x_1, x_2, ⋯, x_K) and (e_1, e_2, ⋯, e_L) to their expanded variables (ξ_1, ⋯, ξ_M) and (ε_1,⋯, ε_N), figuring out the analytical instantaneous transfer function $\phi_{\mathcal{S},t}(\xi_1,\cdots,\xi_M,\varepsilon_1,\cdots,\varepsilon_N)$, conducting the derivatives as shown in (2-20), doing expectations for the coefficient functions in (2-22), and finally switching the variables back to (x_1, x_2, ⋯, x_K) and (e_1, e_2, ⋯, e_L) and simplifying the result. The most difficult tasks in this procedure are often finding the transfer function explicitly and simplifying the expectations of those coefficients. Not surprisingly, a closed form expression often cannot be acquired. Yet, however hard it is, the difficulties are only technical. Of course, despite the insights provided by (2-28), the result becomes quantitatively useful in practice only when the coefficients are evaluated. Some of the examples in Section 2-6 try to execute the complete calculation following the preceding procedure. On the other hand, Section 2-5 introduces the method to achieve the numerical values of these coefficients computationally.

Third, only Assumptions 1-3 are used in our derivation. Therefore, the result and the procedure work on non-stationary inputs with general statistical distributions, as well as transient analysis of a system under stationary inputs.

2.4.5 Some useful variations of (2-28)

Though the formula given in previous subsection can be very useful in analysis, various difficulties may occur in practice. For example, the mean of the output might not

convey enough statistical information. Therefore, it is generally necessary to find the statistical difference between a smooth function, $g(\cdot)$, of the outputs of the IP and FP systems. Let g be the operator whose transfer function is g, g is a smooth operator; its acting on the output of a system gives a new system. The new system still satisfies Assumptions 1-3 and is of great interest. Then, the result (2-28) directly applies to give

$$\begin{aligned} & E[g(f_{\mathbf{S_{FP}}}(x_1, x_2, \cdots, x_{\mathrm{K}}, t)) - g(f_{\mathbf{S_{IP}}}(x_1, x_2, \cdots, x_{\mathrm{K}}, t))] \\ & = \sum_{i=1}^{\mathrm{L}} m_i^g(t) \cdot u_i + \sum_{i}^{\mathrm{L}} h_i^g(t) \cdot s_i{}^2 + \sum_{i,j=1}^{\mathrm{L}} n_{i,j}^g(t) \cdot u_i u_j . \end{aligned} \tag{2-30}$$

Here L is the same as in (2-29) because no new quantizers are introduced by including g. The coefficients, superscripted by g, are different from those in (2-29) unless $g(\cdot)$ is an identity function. Either the procedure mentioned in previous Subsection or the simulation method of Section 2-5 may find out the coefficients.

This result may still lack the ability to reveal quantization effects in practice. In fact, (2-30) suggests the difference interested is often dominated by the means of quantization noises; thus, the random nature of the noises, summarized by their variances, does not show up in (2-30). So it is valuable to study the statistics of a function of the output difference between S_{IP} and S_{FP}. First, suppose g is a smooth memoryless function, that is, the function value at time t only depends on its input at time t, then,

$$\begin{aligned} & E[g\big(f_{\mathbf{S_{FP}}}(x_1, \cdots, x_{\mathrm{K}}, t) - f_{\mathbf{S_{IP}}}(x_1, \cdots, x_{\mathrm{K}}, t)\big)] \\ & \approx g(0) + g'(0) \cdot E[(f_{\mathbf{S_{FP}}} - f_{\mathbf{S_{IP}}})] \ + \tfrac{1}{2} g''(0) \cdot E[(f_{\mathbf{S_{FP}}} - f_{\mathbf{S_{IP}}})^2], \end{aligned} \tag{2-31}$$

where a Taylor expansion is given on g to the second order. This relationship indicates that $E[(f_{\mathbf{S_{FP}}} - f_{\mathbf{S_{IP}}})]$ and $E[(f_{\mathbf{S_{FP}}} - f_{\mathbf{S_{IP}}})^2]$ are the keys to understand the expectation of any memoryless function of the output difference between the FP and IP systems. While

the former has been studied in previous subsection, it is straightforward to find out that the mean-squared error (MSE) of $(f_{\mathbf{S_{FP}}} - f_{\mathbf{S_{IP}}})$ can be represented as

$$E[(f_{\mathbf{S_{FP}}}(x_1, x_2, \cdots, x_{\mathrm{K}}, t) - f_{\mathbf{S_{IP}}}(x_1, x_2, \cdots, x_{\mathrm{K}}, t))^2]$$
$$= \sum_{i,j=1}^{\mathrm{L}} b_{i,j}(t) \cdot u_i u_j + \sum_{i}^{\mathrm{L}} c_i(t) \cdot s_i^{\,2}, \tag{2-32}$$

where $b_{i,j}(t)$ is defined to equal $b_{j,i}(t)$ for simplicity (it is the sum $b_{i,j}(t) + b_{j,i}(t)$ that is uniquely determined). We use coefficient $b_{i,j}(t)$ in (2-32) as the i^{th}-row and j^{th}-column to form a K-by-K matrix B(t) and use $\bar{\mu}$ to represent the column vector formed by $(u_1,\ldots,u_{\mathrm{K}})^T$, where superscript T means vector transpose. Then, (2-32) becomes

$$\boxed{\begin{aligned} &E[(f_{\mathbf{S_{FP}}}(x_1, x_2, \cdots, x_{\mathrm{K}}, t) - f_{\mathbf{S_{IP}}}(x_1, x_2, \cdots, x_{\mathrm{K}}, t))^2] \\ &= \bar{u}^T \mathbf{B}(t) \bar{u} + \sum_{i}^{\mathrm{L}} c_i(t) \cdot s_i^{\,2}. \end{aligned}} \tag{2-33}$$

Since the MSE quantity has to be non-negative regardless of each μ_{i} and σ_{i}, matrix B(t), denoted by the bold letter, must be symmetric and positive semi-definite and $c_i(t)$ has to be positive, denoted as $\mathrm{B}(t) \succeq 0$, and $c_i(t) \geq 0$.

Finally, the function g defined in (2-31) may have memory to study the correlation characteristics of the output quantization noise. Let g_m be one such function where the subscript m indicates that it has memory. Then, the statistic of interest is

$$E[g_m(f_{\mathbf{S_{FP}}}(x_1, \cdots, x_{\mathrm{K}}, t) - f_{\mathbf{S_{IP}}}(x_1, \cdots, x_{\mathrm{K}}, t), t)]. \tag{2-34}$$

The following method outlines one solution to this problem. First, the difference between two systems $\mathrm{S_{IP}}$ and $\mathrm{S_{FP}}$ creates a FP system ($\mathrm{S_{IP}}$-$\mathrm{S_{FP}}$). Followed by the smooth system that is based on function g_m, it becomes a new FP system. All the quantizers in this new system are the same as those in $\mathrm{S_{FP}}$. The IP version of this compound system is ($\mathrm{S_{IP}}$-$\mathrm{S_{IP}}$),

which is constant zero, followed by the system using function g_m. The compound system satisfies Assumptions 1-3 if S_{IP} does. So, the result (2-28) applies to

$$E[g_m\left(f_{\mathbf{S_{FP}}}(x_1,\cdots,x_K,t)-f_{\mathbf{S_{IP}}}(x_1,\cdots,x_K,t),t\right)]-E[g_m(0,t)]$$
$$=\sum_{i=1}^{L} m_i^{g_m}(t)\cdot u_i+\sum_{i}^{L} h_i^{g_m}(t)\cdot s_i^{\,2}+\sum_{i,j=1}^{L} n_{i,j}^{g_m}(t)\cdot u_i u_j. \qquad (2\text{-}35)$$

Here $g_m(0,t)$, by our definition, indicates the output of g_m at time t given all its previous and current inputs are 0. This is a deterministic value and often zero if g_m is unbiased; so, the expectation of the last term in (2-35) can be removed, and (2-35) essentially provides the statistic in (2-34). Of course, MSE of the output after g_m becomes similar to (2-33), where the first order term disappears. That is,

$$E[\{g_m\left(f_{\mathbf{S_{FP}}}(x_1,\cdots,t)-f_{\mathbf{S_{IP}}}(x_1,\cdots,t),t\right)\}^2]-E[\{g_m(0,t)\}^2]$$
$$=\overline{u}^T \mathrm{B}^{g_m}(t)\overline{u}+\sum_{i}^{L} c_i^{\,g_m}(t)\cdot s_i^{\,2}. \qquad (2\text{-}36)$$

This will be used in Example 2 of section VI.

In summary, the output difference between IP and FP systems is a random variable at any time instance, as given in (2-21). Its mean and variance given in (2-28) and (2-33) provide basic understandings of its regularity. In most applications, this information is the sufficient statistics to depict the gross quantization effects. In principal, this provides the statistical quantization effects for all discrete-time DSP systems under stochastic signal environment when Assumptions 1-3 are satisfied. The difficulties left to determine those coefficients are purely technical in both the analytical procedure suggested in previous Subsection and the computational method in the next Section.

2.5 Application in numerical simulations

Advancements of digital computers promote simulation-based approach to understand quantization effects. This can neatly fit into the results in previous Section. Suppose a system has L quantizers, each of which has l possible fractional word-lengths and 2 quantization mode, then, $(2l)^L$ Monte-Carlo estimations are required to understand a statistical quantity that summarizes the quantization effect. Each of these estimations reveals the relationship between the statistical quantity and a specific setup of the FP parameters. The number of estimations needed for a complete understanding of quantization effects is exponentially related to number of quantizers. This exponential number is often too high to be practical. In some applications, such as floating-point to fixed-point conversion (FFC) [13-17], the situation may be alleviated. In simulation based FFC, an assumption based on the intuition—higher fractional word-lengths always result better systems—leads to great savings on the number of estimations needed [13-16]. We will study when this assumption is appropriate in Section 2.6.

However, the analytical results in previous sections indicate that at any time t, only $2L + (L+1)\cdot L/2$ coefficients are unknown and to be determined in (2-28), and $(L+1)\cdot L/2 + L$ independent coefficients are to be determined in (2-33). Together, $(L+4)\cdot L$ unknown coefficients are to be determined, and this is the number (and also the least number) of well-designed estimations needed to numerically determine the quantization effects as given in (2-31). This number, on the order of L^2, is a quadratic function of L—a huge saving comparing with the previous exponential relationship $(2l)^L$. In fact, the number is not a function of l—as long as the word-lengths are not too small to violate Assumptions 1-3. When the mean errors u_i's are all 0, which is the case as all

quantization modes are roundoff and no constant coefficients are involved, then only 2L unknown coefficients are left in (2-29) and (2-33). Then, the number of estimations needed becomes a linear function of L. Our floating-point to fixed-point design tool is largely benefited by these results [1].

Now, a polynomial number of estimations is needed to understand the quantization effects at a time *t*. Fortunately such efforts do not need to repeat at each time instance to understand the quantization effects for all the time before *t*. In fact, they only introduce a small increase to the existing numerical cost. To explain this, let's first look at the components of estimation cost here. The expectations in (2-28), (2-30), and (2-33) can be estimated using different types of estimators, among which ensemble-average estimator is the most intuitive and un-biased one. It estimates the expectation by averaging the corresponding output at a time while running simulations from time 0 to *t* multiple times. All the simulations are based on the same input statistics but with different random ensample realizations. Therefore, to get the numerical output at *t*, all the outputs before the time become byproducts. So, the only overhead-cost to get quantization effects from 0 to *t*-1 is doing *t* more numerical averages that are fast.

The estimation methods may still be too costly because an estimation using a large ensemble average means a large number of digital simulations. This can be eased by using ergodic averages. Whenever the input statistics (or probability density function) do not change rapidly over time, and the mapping from (x_1, x_2, ⋯, x_K, e_1, e_2, ⋯, e_L) to (ξ_1, ⋯, ξ_M, ε_1, ⋯, ε_N) does not change over time rapidly either, over a short period of time, the output signal of the system can be treated as a stationary random process. As a

result, the expectation at a time t can be estimated using their ergodic averages over a short period of time around *t*. In this way, only one digital simulation from time 0 to *t* is needed. This saving becomes useful in many practical applications where simulations to do large ensemble average become too expensive [1].

Another subtle point regarding the way to conduct estimation can also affects the estimation efficiency dramatically. At least two methods can be used to numerically simulate the left side of (2-28). In the first one, $E[f_{\mathbf{S_{FP}}}(x_1, x_2, \cdots, x_K, t)]$ and $E[f_{\mathbf{S_{IP}}}(x_1, x_2, \cdots, x_K, t)]$ are estimated using separate simulations, whereas alternatively, $E[f_{\mathbf{S_{FP}}}(x_1, x_2, \cdots, x_K, t)] - E[f_{\mathbf{S_{IP}}}(x_1, x_2, \cdots, x_K, t)]$ is estimated directly using $E[f_{\mathbf{S_{FP}}}(x_1, x_2, \cdots, x_K, t) - f_{\mathbf{S_{IP}}}(x_1, x_2, \cdots, x_K, t)]$. Both methods can be done in run-time (then the second method requires the simulation of the two systems simultaneously to get their differences) or by post-processing of the saved system outputs. However, errors associated with any estimation often make the first method much less efficient. In fact, the estimation error of each of the two terms could be much higher than the small difference between them. Thus, an accurate simulation of the difference requires a large sample size in each estimation. The second method avoids this numerical difficulty by testing the difference directly. In this way, either the sample size is greatly reduced to achieve the same accuracy, or the accuracy improves comparing with the first method using the same sample size. That is, the second estimation scheme is often more efficient.

2.6 Examples

Example 1. As mentioned in previous section, in simulation based FFC, it is often assumed that higher fractional word-lengths always give better systems [13-16]. In fact, this assumption is not always true, unless some additional conditions are satisfied. This can be illustrated in (2-28) as it depends on the signs of the first order coefficient m_i's. Only when m_i and u_i have the same signs for all i, it is true that higher fractional word-length cause smaller u_i and therefore smaller quantization effects.

Furthermore, the intuition is not true even for the noise power $E[(f_{\mathbf{S_{FP}}} - f_{\mathbf{S_{IP}}})^2]$ in (2-33) that does not have the first order term. For example, a simple high precision subtractor with two quantized signal inputs gives the output quantization noise power as $(u_1 - u_2)^2 + (s_1^2 + s_2^2)$. Though the standard deviation terms indeed decrease as any of the two W_{Fr}'s increases, the first term $(u_1 - u_2)^2$ can have a complicated behavior.

In general, one sufficient condition for $E[(f_{\mathbf{S_{FP}}} - f_{\mathbf{S_{IP}}})^2]$ to satisfy the intuition is that all quantizers use roundoff modes exclusively because then (2-33) reduces to $\sum_{i=1}^{L} c_i(t) \cdot s_i^2$ with $c_i(t) \geq 0$.

■

Example 2. Assuming the input signals are 0-mean stationary random processes, a transfer function method derives that, for a multiple-input-multiple-output linear-time-invariant (MIMO -LTI) system [2-3],

$$\mathbf{R}_{\Delta\bar{y}\Delta\bar{y}}(e^{j\omega}) = \Delta\mathbf{H}_{\bar{y}\bar{x}}(e^{j\omega}) \cdot \mathbf{R}_{\bar{x}\bar{x}}(e^{j\omega}) \cdot \Delta\mathbf{H}^{H}_{\bar{y}\bar{x}}(e^{j\omega}) + \mathbf{H}_{\bar{y}\bar{q}}(e^{j\omega}) \cdot \mathbf{R}_{\bar{q}\bar{q}}(e^{j\omega}) \cdot \mathbf{H}^{H}_{\bar{y}\bar{q}}(e^{j\omega}), \tag{2-37}$$

where the notation and every term are explained in Appendix A. the notations have been slightly modified from [2-3] to accommodate the present context:

Appendix A shows that (2-33) at steady state can be derived from (2-37), whereas (2-37) can be partially derived from (2-36). So, (2-37) verifies the perturbation theory in previous sections.

Based (2-37), a simple Biquad IIR system and other more complicated systems like FFT are examined [2-3]. Theoretical results of the power spectrum density at the output agree well with the simulation, which verifies the LTI theory and therefore the perturbation theory.

■

Example 3: Although error bounds have been studied extensively [18] for CORDIC algorithm, the straight-forward statistical analysis on quantization error variance was done only recently [42]. A CORDIC algorithm contains decision-making operators as well as arithmetic operators. However, ignoring decision-errors, quantization effects is shown in (2-14) of [42], or restated as

$$E[(flpt - fxpt)^2] = E[|e_{or}|^2] = \frac{2}{K^2}\left|E[e_r(N)] + \sum_{j=1}^{N-1}\overline{P}(j)E[e_r(j)]\right|^2 + \frac{2}{K^2}\left(\sigma^2_{e_r(N)} + \sum_{j=1}^{N-1}\left(\prod_{i=j}^{N-1}k(i)^2\right)\sigma^2_{e_r(j)}\right) + E[|e_s|^2]. \tag{2-37}$$

This clearly fits the pattern in (2-33). In fact, without decision errors, the system becomes linear to propagate all the errors; so, the result is predictable using LTI relationship (2-37).

■

Example 4. Based on the perturbation results, various analyses for least-mean-square algorithms have been done and published. They are listed here briefly. For more detail, please read the corresponding reference.

In [52], the transient analysis of a one-tap LMS algorithm with correlated input is studied following both the analytical procedure mentioned after (2-29) and the computational approach mentioned in Section V. Both methods lead to good agreement with Monte-carlo simulation. This example helps to explain how to use the perturbation results of this chapter in multiple ways.

In [2], by direct simulation, (2-28) and (2-33) are validated for a 12-tap LMS algorithm with stationary inputs.

In fact, quantization effects of an LMS and its related algorithms are probably the most studied other than LTI systems [6][28-41][53]. Those results all support our results from perturbation theory, in one way or another.

■

Example 5. The perturbation theory provides the basis for the understanding of many other phenomena present in digital systems, sometimes in a qualitative way. As an example of this statement, let's study the saturation effect of signal-to-noise ratio (SNR)

with respect to finite-word-length. That is, it is often observed that the SNR improves quickly by increasing any fractional word-length W_{Fr} of an FP system until it suddenly reaches a threshold, after which the SNR practically stays the same.

Let n_q, n_{ph}, and x be the accumulated quantization noise, the accumulated physical noise, and the signal at a certain signal node; then, the SNR, defined as the signal power over total noise power, becomes $\frac{E[x^2]}{E[(n_q + n_{\text{ph}})^2]}$. For simplicity, assuming n_q and n_{ph} are uncorrelated, SNR becomes $\frac{E[x^2]}{E[{n_q}^2] + E[{n_{\text{ph}}}^2]}$. From (2-33), as any W_{Fr} increases, $E[{n_q}^2]$ decreases exponentially, which causes the SNR increase quickly. However, when those terms in (2-33) that are associated with this W_{Fr} are already smaller than $E[{n_{\text{ph}}}^2]$, the SNR can at most change 3dB even when W_{Fr} goes to infinity. This explains the saturation effect.

■

2.7 Summary

After a brief review of existing literature, a perturbation theory has been developed to study quantization effects of digital signal processing systems. The analysis applies to linear or non-linear systems, with stationary or non-stationary inputs. Possible applications are explained, followed by a few examples that support our results.

To extending the analysis, one might study the effects of decision-errors, which has been emphasized and studied partially elsewhere [51]. On the other hand, statistical

effects might not be the most interested for all applications. It could be interesting to extend (2-21) in a deterministic way for those situations.

Finally, the assumptions in this chapter limit our results. However, as Dr. Box [54] said, "Models of course, are never true but fortunately it is only necessary that they be useful." Our automated floating-point to fixed-point conversion tool for communication systems has taken advantage of the present results. Its success, besides the few examples section 2-6, proves the model here to be indeed useful [1]. On the other hand, Chapter 3 is going to talk the situation when Assumption 3 is not true.

Chapter 3
Quantization Effects with the Presence of Decision-errors

Most existing analyses of quantization effects, including those in Chapter 2, are given under the condition (sometimes implicitly) that all decision-making blocks, if exist in a system, produce identical decisions in both fixed-point and infinite-precision (IP) implementations. However, in doing floating-point to fixed-point conversion (FFC), a fixed-point design with occasional decision errors may still be an acceptable approximation of the IP system. In this Chapter, we study the effect of this decision error, and relate its probability to the fixed-point data types. The FFC methodology that is briefly described in Chapter 1 is then extended to include systems with possible decision errors due to quantization. The analytical results here are applied to both CORDIC and BPSK transceiver as examples.

3.1 Introduction

By now, it should be clear that to lower hardware costs, most implementations of digital systems rely on binary fixed-point number systems—either 2's complement or unsigned-magnitude—with roundoff and truncation quantization. Existing work

reviewed and presented in the previous chapter studies the effect of this quantization on systems that have no decision-making blocks, a term that is to be defined in section 2, or based on the assumption that there is no decision error. Therefore, the automated infinite-precision (often also referred as floating-point) to fixed-point conversion (FFC) method as been briefly described in Chapter 1 is uses this assumption. However, in many complicated communication and DSP systems, decision errors in a fixed-point system are acceptable as long as its probability is small; then, the system is still a fair approximation of its IP correspondence. Other FFC methods based on unguided optimization and recursive estimations without understanding of the effects of these decision errors, on the other hand, require a large number of long simulations [17]. This becomes especially time-consuming when each simulation takes minutes to hours in bit-error-rate (BER) type of estimation.

Based on a study of the types of decision making blocks and the probability of decision errors as a function of fixed-point data-types in a system, we extend the FFC method to include possible decision errors. The updated FFC problem formulation looks still similar with additional constraints, such that each requires one BER type of estimation for coefficient fitting—itself a well-defined task. Finally, we show two examples, BPSK transceiver with root-raised-cosine-filter and CORDIC, to support our analytical results.

3.2 System Description

In Section 2.3.1, signals of a digital system have been categorized into arithmetic or logical, whereas and operators have been categorized into arithmetic, logical and

decision-making. Operators have been further defined as either smooth or unsmooth depending on their continuity and differentiability.

Let **SL** be the slicer operator of a slicer. It has one arithmetic input x and one logic signal output y. The slicer function transfer this input x to the output y, given below as

$$y = f_{\mathbf{SL}}(x) = \begin{cases} 1, \text{if } x \geq 0 \\ -1, \text{if } x < 0 \end{cases}. \tag{3-1}$$

It turns out all decision-making operators can be equivalently modeled as a combination of arithmetic operator, slicers (a basic decision-making operator), and logical operators as shown in Fig. 3-1.

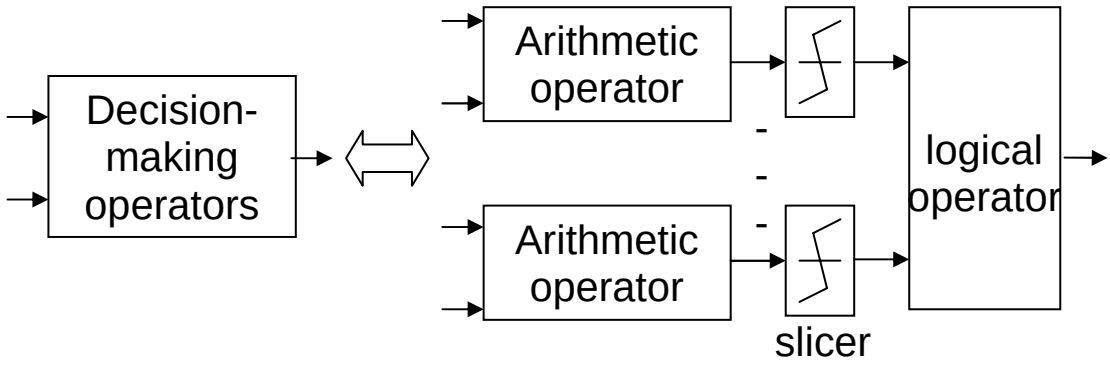


Fig. 3-1 Decomposition of a decision-making block

Example: suppose **A** is a comparator defined as with two input x_1 and x_2, and one output y

$$y = f_{\mathbf{A}}(x_1, x_2) = \begin{cases} 1, \text{if } x_1 > x_2 \\ 0, \text{if } x_1 \leq x_2 \end{cases},$$

where input x_1 and x_2 are arithmetic and y is logical output. It can be easily verify that

$$f_{\mathbf{A}}(x_1, x_2) = \mathbf{NAND}(f_{\mathbf{SL}}(x_2 - x_1), 1),$$

where **NAND** is a logic operator that gives 0 if two inputs are the same, and 1 otherwise. So **A** is equivalent to a combination of a subtractor, a slicer and a NAND operator. Therefore in the subsequent discussion we will concentrate on the quantization effect of a slicer.

■

3.3 Probability of Decision Errors

According to its definition, all arithmetic signals (and only they) can be modified by quantizers. However, logical signals in FP system can also differ from its counterpart in IP, because quantization effects from quantizers for arithmetic signals can alter the decision of those decision-making operators, and therefore the value of logical signal at its output. These changes can further propagate through control logic operators and alter the value of any other logic signal following them. Chapter 2 shows that quantization errors propagating through smooth arithmetic operators give a small perturbation of the output in an IP system. This small perturbation will be transferred as possible different decision through a decision-making operator.

Let x be the IP input of a slicer, θ be the difference between FP and IP version of x, that is,

$$\theta = x_{\mathbf{FP}} - x_{\mathbf{IP}} = x_{\mathbf{FP}} - x. \tag{3-2}$$

For simplicity, assume

Assumption 4: input signal difference θ of a slicer is statistically independent to x at any time instant.

Though this assumption is not strictly true since θ is a deterministic function of all the input signals, in a complicated system both the dependence of θ and x on input signals are so mixed that it often suffices to consider them independent.

Let the probability density of x and θ be p_x and p_θ respectively. Then the FP decision may differ from IP decision according to the following formula

$$P(f_{\mathbf{SL}}(x_{\mathbf{FP}}) = -1, f_{\mathbf{SL}}(x_{\mathbf{IP}}) = 1) = P(x + \theta < 0, x \geq 0)\,. \tag{3-3}$$

With preceding Assumption 4, the probability above can be written as a double integral over x and θ,

$$\begin{aligned} &P(f_{\mathbf{SL}}(x_{\mathbf{FP}}) = -1, f_{\mathbf{SL}}(x_{\mathbf{IP}}) = 1) \\ &= \iint\limits_{\substack{x+\theta<0 \\ x\geq 0}} p_x(x)\,p_\theta(\theta)\,dx\,d\theta = \int\limits_{-\infty}^{0}\left(p_\theta(\theta) \int\limits_{0}^{-\theta} p_x(x)dx \right)d\theta. \end{aligned}$$

Under the assumption that error magnitude is small comparing with signal x, the integral regarding $p_x(x)$ is around $p_x(0)$ in the integral, the probability becomes

$$\begin{aligned} &P(f_{\mathbf{SL}}(x_{\mathbf{FP}}) = -1, f_{\mathbf{SL}}(x_{\mathbf{IP}}) = 1) \\ &\cong p_x(0) \int\limits_{-\infty}^{0} \big(-\theta \cdot p_\theta(\theta)\big)d\theta = -p_x(0) \cdot E_\theta[\theta \cdot \mathbf{1}(\theta < 0)], \end{aligned} \tag{3-4}$$

where the last step follows directly from definition of expectation value.

Similarly, the probability of error from decision of -1 in IP system to decision of 1 in FP system is given by

$$\begin{aligned} &P(f_{\mathbf{SL}}(x_{\mathbf{FP}}) = 1, f_{\mathbf{SL}}(x_{\mathbf{IP}}) = -1) \\ &= p_x(0) \cdot E_\theta[\theta \cdot \mathbf{1}(\theta \geq 0)]. \end{aligned} \tag{3-5}$$

Sum (3-4) and (3-5) together, we get the probability of decision error event between IP and FP system as

$$P(f_{\mathbf{SL}}(x_{\mathbf{FP}}) \neq f_{\mathbf{SL}}(x_{\mathbf{IP}}))$$
$$= p_x(0) \cdot (E_\theta[\theta \cdot \mathbf{1}(\theta \geq 0)] + E_\theta[-\theta \cdot \mathbf{1}(\theta < 0)]).$$

The two expectations in the parenthesis can be combined together as the expectation of $E_\theta[|\theta|]$. Therefore, the proceeding equation can be written as

$$P(f_{\mathbf{SL}}(x_{\mathbf{FP}}) \neq f_{\mathbf{SL}}(x_{\mathbf{IP}})) = p_x(0) \cdot E_\theta[|\theta|]. \tag{3-6}$$

However, due to Cauchy-Swartz inequality,

$$E_\theta[|\theta|] \leq (E_\theta[|\theta|^2])^{1/2}.$$

So (3-6) can finally be written into a form

$$P(f_{\mathbf{SL}}(x_{\mathbf{FP}}) \neq f_{\mathbf{SL}}(x_{\mathbf{IP}})) = \gamma \cdot p_x(0) \cdot \sqrt{E_\theta[\theta^2]}. \tag{3-7}$$

where $\gamma \leq 1$. Furthermore, γ is usually between 0.7 and 1 for practical distribution of θ. For example, $\gamma = \sqrt{2/\pi} \cong 0.8$, $\gamma = \sqrt{3}/2 \cong 0.87$, and $\gamma = 1$ for cases that θ has zero-mean Gaussian distribution, zero-mean uniform distribution, and two point masses symmetric around 0, respectively.

Equation (3-7) shows that the decision difference between IP and FP system is proportional to the square-root of the accumulated quantization error power $E[\theta^2]$, also called mean-squared error (MSE) of θ. The coefficients may vary for different systems and signal environment depending on how well the independence Assumption 4 applies. This quantity has been related directly to the fixed-point data types in Chapter 2. In fact, notice that $E[\theta^2]$ is just the left side of (2-28). That is, $E[\theta^2] = E[|\text{ flpt} - \text{fxpt }|^2]$.

In the following example

Example: When θ has a Gaussian distribution $\mathcal{N}(\mu_\theta, \sigma_\theta^2)$, Appendix B shows a more complicated way to prove (3-7).

■

Therefore the quantization errors brought into the system in arithmetic operators turns into this decision error with a probability proportional to $p_x(0)\sqrt{E[\theta^2]}$. The coefficients might vary in real system depending on how well assumption 4 is in reality.

In reality, especially in communication systems, there is a more subtle system specification—a "desired decision" might exist, named as $D(x)$, associated with x at each time instance. Even in the IP system, the decision-making operators might produce "wrong" decisions simply because of physical noise or system/architecture deficiency. In this case it is how much more often such error happens in a FP system that is most concerned. Without losing any generality, let $D(x)=1$ corresponds to the correct decision, then let the probability density of x in this case be $p_{x|D(x)=1}(x)$. This function is usually different from $p_x(x)$. Then the probability or wrong decision in FP system differs from IP system by

$$\begin{aligned}
&P(f_{\mathbf{SL}}(x_{\mathbf{FP}}) = -1 \,|\, D(x) = 1) - P(f_{\mathbf{SL}}(x_{\mathbf{IP}}) = -1 \,|\, D(x) = 1) \\
&= \iint\limits_{x+\theta<0} p_{x|D(x)=1}(x)\, p_\theta(\theta)\, dx\, d\theta - \int\limits_{-\infty}^{0} p_{x|D(x)=1}(x)\, dx \\
&= \int\limits_{-\infty}^{+\infty} \left(p_\theta(\theta) \left(\int\limits_{-\infty}^{-\theta} p_{x|D(x)=1}(x)\, dx - \int\limits_{-\infty}^{0} p_{x|D(x)=1}(x)\, dx \right) \right) d\theta \\
&= \int\limits_{-\infty}^{+\infty} \left(p_\theta(\theta) \int\limits_{0}^{-\theta} p_{x|D(x)=1}(x)\, dx \right) d\theta
\end{aligned} \tag{3-8}$$

Now again notice θ is concentrated around 0, so

$$
\begin{aligned}
&\int_0^{-\theta} p_{x|D(x)=1}(x)dx \\
&\approx \int_0^{-\theta} (p_{x|D(x)=1}(0) + p'_{x|D(x)=1}(0)x)dx \\
&= -\theta \cdot p_{x|D(x)=1}(0) + \tfrac{1}{2}\theta^2 \cdot p'_{x|D(x)=1}(0)
\end{aligned}
\tag{3-9}
$$

Therefore (3-8) becomes

$$
\begin{aligned}
&P(f_{\mathbf{SL}}(x_{\mathbf{FP}}) = -1 \,|\, D(x) = 1) - P(f_{\mathbf{SL}}(x_{\mathbf{IP}}) = -1 \,|\, D(x) = 1) \\
&= -\mu_{\theta|D(x)=1} \cdot p_{x|D(x)=1}(0) + \tfrac{1}{2} E_{\theta|D(x)=1}[\theta^2] \cdot p'_{x|D(x)=1}(0)
\end{aligned}
\tag{3-10}
$$

Here both error moments are conditioned on $D(x)$=1. The second order terms are included since μ_θ might be 0 in some cases. When μ_θ>0 the probability of wrong error under $D(x)$=1 actually gets less in FP system. However the probability of wrong error under $D(x)$=-1 gets larger. Net wrong decision probability is the sum of these two. In case θ and $D(x)$ are independent the conditional moments are the same as moments. With this simplification the result becomes

$$
\begin{aligned}
&P(f_{\mathbf{SL}}(x_{\mathbf{FP}}) \neq D(x)) - P(f_{\mathbf{SL}}(x_{\mathbf{IP}}) \neq D(x)) \\
&= \mu_\theta \cdot (p_{x|D(x)=-1}(0)P(D(x) = -1) \\
&\qquad - p_{x|D(x)=1}(0)P(D(x) = 1)) \\
&\quad + \tfrac{1}{2} E[\theta^2] \cdot (p'_{x|D(x)=1}(0) \cdot P(D(x) = 1) \\
&\qquad\qquad - p'_{x|D(x)=-1}(0) \cdot P(D(x) = -1))
\end{aligned}
\tag{3-11}
$$

Example: In maximum likelihood detection of BPSK signals that are corrupted by IID symmetrical noise, the threshold is set at 0 because

$$p_{x|D(x)=-1}(0)P(D(x) = -1) = p_{x|D(x)=1}(0)P(D(x) = 1).$$

Also note normally, as in Gaussian physical noise,

$$p'_{x|D(x)=-1}(0) > 0, p'_{x|D(x)=1}(0) < 0 .$$

Therefore only the second term is left in (3-11) and it becomes

$$\begin{aligned} & P(f_{\mathbf{SL}}(x_{\mathbf{FP}}) \neq D(x)) - P(f_{\mathbf{SL}}(x_{\mathbf{IP}}) \neq D(x)) \\ = & \tfrac{1}{2} E[\theta^2] \cdot (p'_{x|D(x)=1}(0) P(D(x)=1) \\ & - p'_{x|D(x)=-1}(0) P(D(x)=-1)) \end{aligned} \quad . \qquad (3\text{-}12)$$

That again points out the importance of studying $E[\theta^2]$.

■

3.4 Effects of decision errors

Decision errors are brought into the system via decision-making operators as described in the previous section. These erroneous signals can further affect either logic signals via control operators or arithmetic signal via arithmetic operators. The pattern of this error propagation becomes generally difficult to track.

One overly simplified treatment can be given assuming decision error events at a node happen randomly following a Poisson process, independent to all the signals in IP system. Let's consider a slicer again as this can be treated as the building block for all decision-making blocks. When a random error happens, the error value is either -2 or 2 depending on whether the signal value in IP system is 1 or -1, respectively. So, clearly each error is dependent to the IP signal value. With this Poisson process assumption, one can do simulations to an IP system by randomly altering the decision, and examine the signal variation at the output of the system comparing with IP output.

The problem of the approach above is that decision-error events are closely related to signal values in the IP system. For a slicer, it has been pointed out that an error

usually happens when the magnitude of the input IP signal to a slicer is compatible to the accumulated quantization error. Therefore, the quantization effect of this decision error is associated with special occasions of input signals. The propagated error then can be very different to an error that happens randomly as assumed in previous paragraph.

The following example provides the treatment for an absolute-value function and illustrates the previous statement.

Example: A slicer determines the sign of input. If positive a following multiplexer will select the input; otherwise the negated value is selected, as shown in Fig. 3-2.

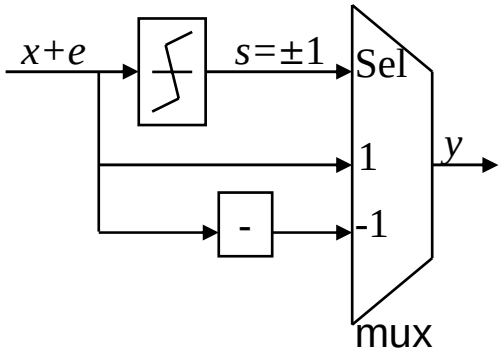


Fig. 3-2 An implementation of absolute value function.

In this example, let's assume both IP signal x and quantization error follow zero-mean independent Gaussian distribution with $\mathcal{N}(0, \sigma_x^2)$ and $\mathcal{N}(0, \sigma_e^2)$, respectively. The actual error at output become

$$
\begin{aligned}
&E[(|x+e|-|x|)^2] = \sigma_e^2 + 2\sigma_x^2 - 2E[|x+e|\cdot|x|] \\
&= \sigma_e^2 + 2\sigma_x^2 - 2E[|(x+e)x|] \\
&= \sigma_e^2 + 2\sigma_x^2 - 2E[(x+e)x\cdot\mathbf{1}((x\ge 0, e\ge -x) \text{ or } (x<0, e<-x))] \\
&-2E[-(x+e)x\cdot\mathbf{1}((x\ge 0, e<-x) \text{ or } (x<0, e\ge -x))] \\
&= \sigma_e^2 + 2\sigma_x^2 - 2E[(x+e)x] \\
&+4E[(x+e)x\cdot\mathbf{1}((x\ge 0, e<-x) \text{ or } (x<0, e\ge -x))] \\
&= \sigma_e^2 + 8E[(x+e)x\cdot\mathbf{1}(x\ge 0, e<-x)] \\
&= \sigma_e^2 + \frac{4}{\pi}\sigma_x^2(-\frac{\sigma_e}{\sigma_x} + \frac{\pi}{2} - \tan^{-1}(\frac{\sigma_x}{\sigma_e})) \\
&\approx \sigma_e^2 - \frac{4\sigma_e^3}{3\pi\sigma_x}
\end{aligned}
\tag{3-13}
$$

Here **1**(x) is again the indicator function which gives 0 when x is false and 1 when x is true. The last step is using Taylor expansion around $\frac{\sigma_e}{\sigma_x}$, considering $\sigma_e << \sigma_x$. A simpler method is using Price's theorem [87]. Let y_1 and y_2 be both $\mathcal{N}$ (0,1) with cross-correlation ρ. Define $R = E[|y_1|\cdot|y_2|]$. Notice the first and second derivatives of an absolute function is

$$
|x|' = \begin{cases} -1, x<0 \\ 1, x>0 \end{cases}, \text{and } |x|'' = 2\delta(x),
$$

that is, the second derivative is a Delta function with a factor 2. Thus, according to Price's theorem one gets

$$\frac{\partial^2 R(\rho)}{\partial \rho^2}$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \frac{|y_1|''\cdot|y_2|'' e^{-(y_1^2+y_2^2-2\rho\cdot y_1 y_2)/2(1-\rho^2)} dy_1 dy_2}{2\pi\sqrt{1-\rho^2}}$$

$$= \frac{2dt}{\pi\sqrt{1-\rho^2}}$$

So let

$$y_1 = x/\sigma_x \,,\; y_2 = x+e/\sqrt{\sigma_x^2+\sigma_e^2}\,,$$

$$\text{and } \rho_0 = E[y_1 y_2] = \sigma_x/\sqrt{\sigma_x^2+\sigma_e^2}\,,$$

we get

$$E[|(x+e)x|] = \sigma_x\sqrt{\sigma_x^2+\sigma_e^2}\cdot R$$

$$= \sigma_x\sqrt{\sigma_x^2+\sigma_e^2}\left(\int_0^{\rho_0}\left[\left.\frac{\partial R}{\partial\rho}\right|_{\rho=0} + \int_0^y \left.\frac{\partial^2 R}{\partial\rho^2}\right|_{\rho=t} dt\right]dy + R\big|_{\rho=0}\right)$$

$$= \sigma_x\sqrt{\sigma_x^2+\sigma_e^2}\left(\int_0^{\rho_0}\left[\left.\frac{\partial R}{\partial\rho}\right|_{\rho=0} + \int_0^y \frac{2dt}{\pi\sqrt{1-t^2}}\right]dy + R\big|_{\rho=0}\right)$$

$$= \sigma_x\sqrt{\sigma_x^2+\sigma_e^2}\left(\int_0^{\rho_0}\left[0 + \int_0^y \frac{2dt}{\pi\sqrt{1-t^2}}\right]dy + \frac{2}{\pi}\right)$$

$$= \frac{\pi}{2}\left(\sigma_x^2 \sin^{-1}(\rho_0) + \sigma_x\sigma_e\right) = \frac{\pi}{2}\left(\sigma_x^2 \tan^{-1}\left(\frac{\sigma_x}{\sigma_e}\right) + \sigma_x\sigma_e\right)$$

This together with the first step of (3-13) gives the same result as (3-13).

On the other hand, if it is assumed that decision errors happen independently according to probability given in previous subsection, the error variance should be roughly

$$
\begin{aligned}
& E[(|x+e|-|x|)^2] \\
\approx\ & E[(-|x|-|x|)^2 \mathbf{1}\left(f_{\mathbf{SL}}(x_{\mathbf{FP}}) \neq f_{\mathbf{SL}}(x_{\mathbf{IP}})\right)] \\
=\ & E[(-|x|-|x|)^2] E[\mathbf{1}\left(f_{\mathbf{SL}}(x_{\mathbf{FP}}) \neq f_{\mathbf{SL}}(x_{\mathbf{IP}})\right)] \\
=\ & 4\sigma_x^2 \cdot P\left(f_{\mathbf{SL}}(x_{\mathbf{FP}}) \neq f_{\mathbf{SL}}(x_{\mathbf{IP}})\right) \\
=\ & \frac{4\sigma_x^2}{\sqrt{2\pi}} \frac{1}{\sqrt{2\pi}\sigma_x} \sigma_e = \frac{2}{\pi}\sigma_e\sigma_x
\end{aligned}
$$

This approximation gives an estimate of quantization error that is one order higher than the actual result in (3-13). Therefore, it is not acceptable to assume independency between decision error events and IP signal.

■

This example clearly indicates the independence assumption described at the beginning of Section 3.4 can oversimplified the problem. On the other hand, assuming the decision are always the same in FP and IP system, MSE($y_{\mathbf{FP}}$ - $y_{\mathbf{IP}}$) would become

$$E[(\text{sgn}(x)(x+e) - \text{sgn}(x)x)^2] = \sigma_e^2 . \tag{3-14}$$

Comparing with the result in (3-13) one can see that decision-error at the slicer only introduced error variance proportional to σ_e^3. This is not surprising since decision error happens at rate $\sim\sigma_e$, when signal-magnitude is around σ_e as well; so the resulted MSE magnitude is expected to be $\sim\sigma_e\sigma_e^2 = \sigma_e^3$. Of course not all decision-errors influence the system in such a manner. Therefore it is beneficial to categorize decision-error into two groups: *soft decision-error* and *hard decision-error*.

A *soft decision-error* is defined to be a decision-error that when happens affect the system output in the same scale as the quantization error noise. From the analysis

above it affects MSE($y_{\mathbf{FP}}$ - $y_{\mathbf{IP}}$) by small magnitude on the order of $\mathrm{Var}(e)^{3/2}$ or even less significantly. The decision-errors in the previous absolute value function are soft errors.

On the other hand, a *hard decision-error* is defined to be a decision-error that when happens the system output may change in a magnitude much greater than quantization error noise. The following example shows a system with hard decision-errors.

Example: A one-tap version of sign algorithm is given in Fig. 3-3. Comparing with LMS algorithm where residue error e is directly used in updating equation it use sgn(e) to speed up the algorithm as a multiplier in the feedback loop is eliminated. Suppose the input signal x_n is 1 or -1. The desired tap weight w_o is 1, and physical noise v has uniform distribution $U(-\frac{1}{2b},\frac{1}{2b})$. Let x_n and v_n be IID sequences that are mutually independent. Assume the only difference between FP and IP system is the presence of a quantizer after the tap weight w_n in FP system. That is, at time instant n a quantization noise q_n is inserted in the system, where q_n follows uniform distribution $U(-\Delta_q/2+\mu_q, \mu_q+\Delta_q/2)$, where $\mu_q=0$ or $-\Delta_q/2$ for round-off and truncation quantizers. To simplify the analysis we consider the case $\Delta_q<2\mu\ll\frac{1}{2b}$, where μ is update step coefficient.

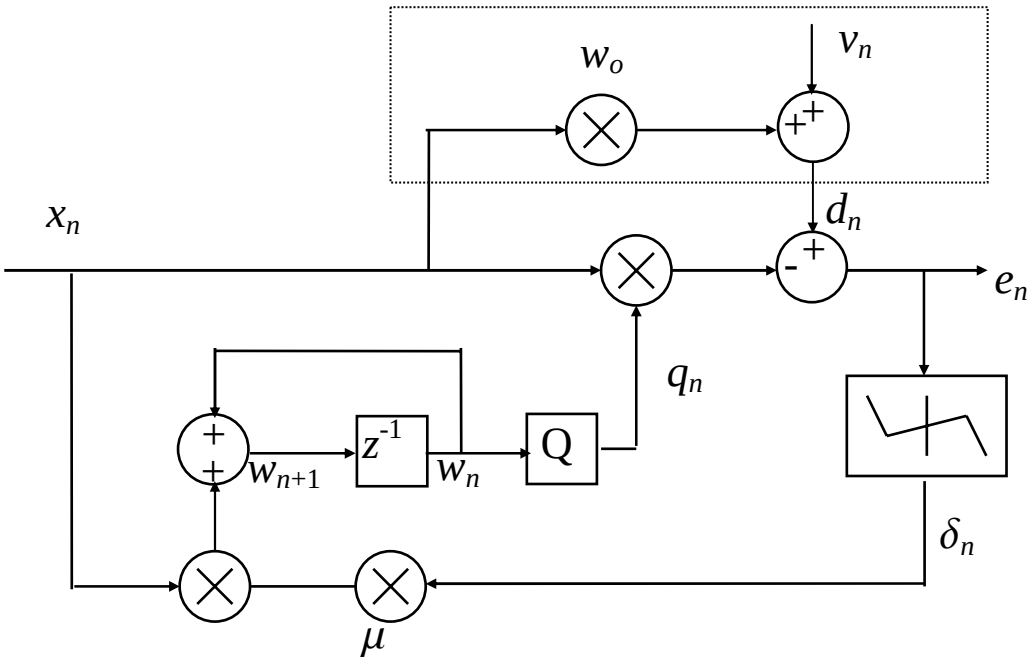


Fig. 3-3 Sign algorithm in Example.
Dashed box shows the desired system with noise *v*.

When the corresponding signals in a FP system and IP system may be different use a superscript FP or IP to discern, and define the difference between them by a prefix Δ; thus

$$\Delta s_n = s_n^{\mathrm{FP}} - s_n^{\mathrm{IP}}. \tag{3-15}$$

Then the updating equations for tap weight are

$$w_{n+1}^{\mathrm{FP}} = w_n^{\mathrm{FP}} + x_n \delta_n^{\mathrm{FP}} \mu \text{ and } w_{n+1}^{\mathrm{IP}} = w_n^{\mathrm{IP}} + x_n \delta_n^{\mathrm{IP}} \mu;$$

So

$$\Delta w_{n+1} = \Delta w_n + x_n \cdot \Delta \delta_n \cdot \mu. \tag{3-16}$$

Since δ_n^{FP}, δ_n^{IP} and x_n are either 1 or -1, both w_n^{FP} and w_n^{IP} can only be integer multiples of μ. Similarly $\Delta\delta_n$ is either 2, 0, or 2; so Δw_n must be integer multiples of 2μ. Now

$$
\begin{aligned}
&\Delta\delta_n = \delta_n^{\mathrm{FP}} - \delta_n^{\mathrm{IP}} \\
&= \mathrm{sgn}(e_n^{\mathrm{FP}}) - \mathrm{sgn}(e_n^{\mathrm{IP}}) \\
&= \mathrm{sgn}(v_n + x_n w_o - x_n(q_n + w_n^{\mathrm{FP}}) - \mathrm{sgn}(v_n + x_n w_o - x_n w_n^{\mathrm{IP}}) \\
&= \begin{cases} 2, \text{ if } v_n + x_n w_o - x_n(q_n + w_n^{\mathrm{FP}}) \geq 0 \text{ and } v_n + x_n w_o - x_n w_n^{\mathrm{IP}} < 0 \\ -2, \text{ if } v_n + x_n w_o - x_n(q_n + w_n^{\mathrm{FP}}) < 0 \text{ and } v_n + x_n w_o - x_n w_n^{\mathrm{IP}} \geq 0 \\ 0, \text{ otherwise} \end{cases} \\
&= \begin{cases} 2, \text{ if } x_n(q_n + \Delta w_n) + x_n(w_n^{\mathrm{IP}} - w_o) \leq v_n < x_n(w_n^{\mathrm{IP}} - w_o) \\ -2, \text{ if } x_n(q_n + \Delta w_n) + x_n(w_n^{\mathrm{IP}} - w_o) > v_n \geq x_n(w_n^{\mathrm{IP}} - w_o) \\ 0, \text{ otherwise} \end{cases}
\end{aligned}
\tag{3-17}
$$

An immediate observation is that if $\Delta w_n > 0$, $q_n + \Delta w_n > 0$ as $q_n < 2\mu \leq \Delta w_n$ which suggests $\Delta\delta_n$ can be 2 only if $x_n < 0$, and -2 only if $x_n > 0$. Together with equation (3-16), it says $\Delta w_{n+1} < \Delta w_n$ strictly. Similarly, if $\Delta w_n < 0$, then $\Delta w_{n+1} > \Delta w_n$ strictly. So, starting from $\Delta w_0 = 0$, the only possible values of Δw_n are 0 and $\pm 2\mu$.

On the other hand one can view $(w_n^{\mathrm{IP}} - w_o)$ as a random walk as well with step size μ. When $|w_n^{\mathrm{IP}} - w_o|$ is away from 0 the adaptive mechanism will drive it to walk towards 0 with a higher probability than to walk further away from 0. Thus at steady state $|w_n^{\mathrm{IP}} - w_o|$ will center around 0 and highly unlikely reach near $\frac{1}{2b}$. So

$$
\begin{aligned}
&\Delta\delta_n \big| x_n \\
&= \begin{cases} 2, \text{ w/ prob } b \cdot E_{q_n}[-x_n(q_n + \Delta w_n) \cdot \mathbf{1}(-x_n(q_n + \Delta w_n) > 0)] \\ -2, \text{ w/ prob } b \cdot E_{q_n}[x_n(q_n + \Delta w_n) \cdot \mathbf{1}(x_n(q_n + \Delta w_n) > 0)] \\ 0, \text{ w/ prob } 1 - b \cdot E_{q_n}[|\, x_n(q_n + \Delta w_n)\,|] \end{cases} ;
\end{aligned} \tag{3-18}
$$

together with (3-16) one gets the transition probability

$$
\begin{aligned}
&P(\Delta w_{n+1} = 0 \mid \Delta w_n = 2\mu) \\
&= P(x_n \cdot \Delta\delta_n = -2 \mid \Delta w_n = 2\mu) \\
&= P(x_n = 1) \cdot P(\Delta\delta_n = -2 \mid x_n = 1, \Delta w_n = 2\mu) \\
&\quad + P(x_n = -1) \cdot P(\Delta\delta_n = 2 \mid x_n = -1, \Delta w_n = 2\mu) \\
&= E_{q_n}[(q_n + 2\mu)b] \\
&= \begin{cases} 2\mu \cdot b, \text{ if round - off} \\ (2\mu - \dfrac{\Delta_q}{2})b, \quad \text{if truncation} \end{cases}
\end{aligned} \tag{3-19}
$$

Where $\Delta_q < 2\mu$ has been applied to secure the $q_n + 2\mu$ always positive. Similarly

$$
P(\Delta w_{n+1} = 0 \mid \Delta w_n = -2\mu) = \begin{cases} 2\mu \cdot b, \text{ if round - off} \\ (2\mu + \dfrac{\Delta_q}{2})b, \quad \text{if truncation} \end{cases}
$$

as well, and

$$
\begin{aligned}
&P(\Delta w_{n+1} = 2\mu \mid \Delta w_n = 0) \\
&= P(x_n \cdot \Delta\delta_n = 2 \mid \Delta w_n = 0) \\
&= P(x_n = 1) \cdot P(\Delta\delta_n = 2 \mid x_n = 1, \Delta w_n = 0) + \\
&\qquad P(x_n = -1) \cdot P(\Delta\delta_n = -2 \mid x_n = -1, \Delta w_n = 0) \\
&= P(x_n = 1) \cdot E_{q_n}[P(\Delta\delta_n = 2 \mid x_n = 1, \Delta w_n = 0)] + \\
&\qquad P(x_n = -1) \cdot E_{q_n}[P(\Delta\delta_n = -2 \mid x_n = -1, \Delta w_n = 0)] \\
&= E_{q_n}[-q_n b \cdot \mathbf{1}(q_n < 0)] = \begin{cases} \dfrac{\Delta_q}{8} b, \text{ if round - off} \\ \dfrac{\Delta_q}{2} b, \quad \text{if truncation} \end{cases}
\end{aligned} \tag{3-20}
$$

Similarly

$$
\begin{aligned}
&P(\Delta w_{n+1} = -2\mu \mid \Delta w_n = 0) \\
&= E_{q_n}[q_n b \cdot \mathbf{1}(q_n > 0)] \\
&= \begin{cases} \dfrac{\Delta_q}{8} b, \text{if round-off} \\ 0, \text{if truncation} \end{cases}
\end{aligned}
\tag{3-21}
$$

In steady state, interstate-transitions are balanced following two equations

$$
\begin{cases}
P(\Delta w_n = 0)P(\Delta w_{n+1} = -2\mu \mid \Delta w_n = 0) \\
\qquad = P(\Delta w_n = -2\mu)P(\Delta w_{n+1} = 0 \mid \Delta w_n = -2\mu), \\
P(\Delta w_n = 0)P(\Delta w_{n+1} = 2\mu \mid \Delta w_n = 0) \\
\qquad = P(\Delta w_n = 2\mu)P(\Delta w_{n+1} = 0 \mid \Delta w_n = 2\mu)
\end{cases}
\tag{3-22}
$$

Solving equation (3-20), (3-21) and (3-22), together with the fact that the total probability equals one, the state probability are shown in Table 3-1.

$P(\Delta w_n)$	$\Delta w_n = -2\mu$	$\Delta w_n = 0$	$\Delta w_n = 2\mu$
Round-off	$\dfrac{\Delta_q}{16\mu}\dfrac{1}{1+\Delta_q/8\mu}$	$\dfrac{1}{1+\Delta_q/8\mu}$	$\dfrac{\Delta_q}{16\mu}\dfrac{1}{1+\Delta_q/8\mu}$
Truncation	0	$1-\dfrac{\Delta_q}{4\mu}$	$\dfrac{\Delta_q}{4\mu}$

Table 3-1 State probability of signed algorithm

A number of simulations from an actual SA algorithm implemented in Matlab are generated to prove the analysis. Plotting both the simulation results and calculation results, Fig. 3-4 shows the good agreement between these two and validates the analyses in this example.

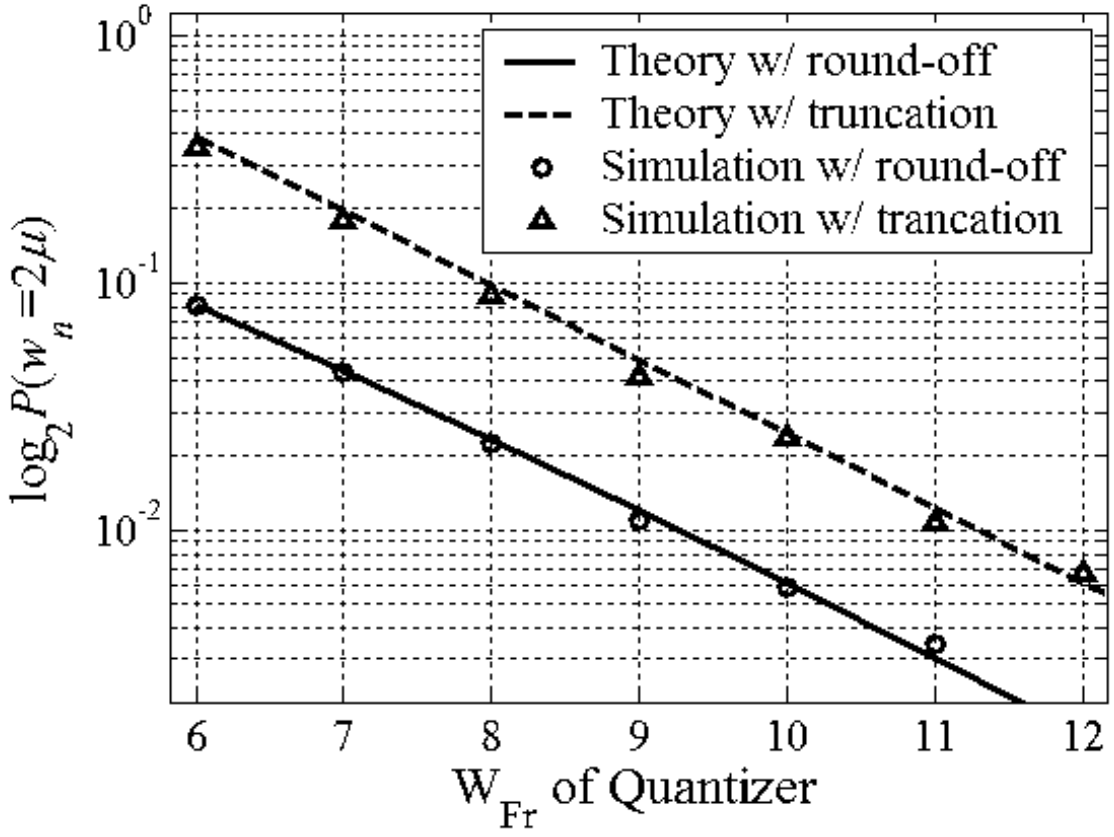


Fig. 3-4 Semi-log plot of $P(\Delta w_n = 2\mu)$ for the signed algorithm.
System parameters are μ=0.01, b=2.5 and $W_{Fr} \le 6$ since $2^{-W_{Fr}} = \Delta_q < 2\mu$. In all simulations for truncation cases, $P(\Delta w_n = -2\mu) = 0$.

Remark: One can use result from Section 3.3 to directly get equation (3-18). For example, $P(\Delta\delta_n = 2|x_n)$ can be calculated as the following. Given x_n, the total quantization noise at the slicer is $-x_n(\Delta w_n + q_n)$ which is small comparing with physical noise v_n; so according to equation (3-4), we get

$$\begin{aligned} & P(\Delta\delta_n = 2|x_n) = P(\delta_n^{\mathrm{IP}} = -1, \delta_n^{\mathrm{FP}} = 1) \\ &= p_{e_n^{\mathrm{IP}}}(0) \cdot E_{q_n}[-x_n(q_n + \Delta w_n) \cdot \mathbf{1}(-x_n(q_n + \Delta w_n) > 0)] \end{aligned} \tag{3-23}$$

where $e_n^{\mathrm{IP}} = v_n + x_n(w_o - w_n^{\mathrm{IP}})$. As the coefficient is not important in our calculation of steady state probability since they cancel out, this is equivalent to the corresponding part of (3-18).

In fact, if $|x_n(w_o - w_n^{\mathrm{IP}})|$ is always less than $\frac{1}{2b}$, one can prove $p_{e_n^{\mathrm{IP}}}(0) = b$ regardless of the distribution of $x_n(w_o - w_n^{\mathrm{IP}})$ under the assumption v_n is uniformly distributed. Then (3-23) and the corresponding case of (3-18) are exactly the same.

■

Because now the MSE(Δw_n) is proportional to the first order of quantization step, the decision errors in the preceding example are indeed hard decision-errors.

Unfortunately, a general study of the gross effects of decision errors is not available and is certainly one of the most interesting topics for future research. Nevertheless, FFC can still be done with the presence of decision errors. The basic idea is to suppress their probability below some tolerance, which is very small itself. Section 3.5 continues on this topic.

3.5 Application in Floating-point to fixed-point conversion

3.5.1 General Analyses

The accumulated quantization noise power MSE(θ) is related to fixed-point data-types, namely fractional word-lengths $W_{Fr,1}, W_{Fr,2}, \ldots$ and quantization modes $q_1, q_2, \ldots$. It has been stated in (2-28) and restated here for convenience,

$$\mathrm{MSE}(\theta) = \bar{u}^T \mathrm{B} \bar{u} + \sum_{i \in \{\text{Data Path}\}} \mathrm{C}_i 2^{-2W_{Fr,i}}, \tag{3-24}$$

where, again, coefficients B is a positive semi-definite matrix, denoted as $\mathrm{B} \succeq 0$, and $\mathrm{C}_i \geq 0$. Vector $\bar{u}$ is related to fixed-point data-types deterministically as shown in (1-4). These coefficients in (3-24) can be found using simulations in an FFC problem with careful setups of fixed-point data types. From (3-7) and the discussion in previous section, using large word lengths for the setups avoids strong decision errors in the simulations and leads to the coefficients in the same as described in Chapter 2. That is, when doing simulations to get those coefficients in (3-24), decision-errors can be ignored when word-lengths are chosen to be large.

Furthermore, since weak-decision-errors can be neglected in both simulation and analysis, we just need to regulate the chance of strong decision-errors. The quantization effects further caused by a strong decision-error do not affect the system performance in an avalanche effect, because the IP system is tested to be robust under physical noise. So the probability of decision error at a strong decision-making block needs to be smaller than those caused by physical noise, which usually corresponds to BER specification, that is,

$$P(f_{\mathbf{SL}}(x_{\mathbf{FP}}) \neq f_{\mathbf{SL}}(x_{\mathbf{IP}})) < \alpha \cdot BER,$$

where design parameter α is a positive guard fractional number. Substituting (3-7) into this inequality, we get

$$\gamma \cdot p_x(0) \cdot \sqrt{E_\theta[\theta^2]} < \alpha \cdot BER.$$

Here $E_\theta[\theta^2]$ is the same as MSE(θ) in (3-24) since the effects of previous strong decision errors, which happen at some sample-time long before, have faded away. Rewriting this equation, we get

$$\text{MSE} < \left(\alpha \cdot BER \big/ \gamma \cdot p_x(0)\right)^2. \tag{3-25}$$

A stronger version of (3-25) is by substituting the fractional number γ by 1. Furthermore, $p_x(0)$ can be directly obtained by estimating the probability of decision difference between the IP system and an otherwise identical system, but with an additive noise n of power MSE_n added at the input of the decision-making block. Denote this probability as $P(f_{\mathbf{SL}}(x_{\mathbf{IP}\text{ with }n}) \neq f_{\mathbf{SL}}(x_{\mathbf{IP}}))$, from (3-7), we get

$$p_x(0) = \frac{P(f_{\mathbf{SL}}(x_{\mathbf{IP}\text{ with }n}) \neq f_{\mathbf{SL}}(x_{\mathbf{IP}}))}{\gamma_n \cdot \sqrt{\text{MSE}_n}}, \tag{3-26}$$

where γ_n depends only on the noise shape of n, as explained shortly after (3-7). With (3-26), the right side of (3-25) is completely determined, denoted as A; therefore, (3-25) reduces to $\text{MSE} - A < 0$. This condition, associated with (3-24), again gives a constraint function on FFC problem in exactly the same form of those showed in previous two chapters, where no decision-making blocks have been considered. Thus, with a condition for each strong decision-making block, the FFC problem is re-formulated in the

same form as stated in Section 1.3.5. The only change is some additional constraint functions. One BER type estimation is needed for each of this strong decision-making blocks—a very well-defined task.

3.5.2 BPSK and CORDIC examples

The first example of weak decision errors whose quantization effects can be neglected are those happened in a CORDIC system with large number of rotation stages [42][18]. In fact, the errors at CORDIC output caused by decision errors can be essentially bounded by the residue error caused by finite rotation stages—one type of architecture imperfection that vanishes as the number of stages becomes large [18]. Furthermore, these errors, as shown in Section 3.3, happen with very small probability. These two reasons ensure that the noise power at CORDIC output can be accurately predicted regardless of the possible internal decision errors [42].

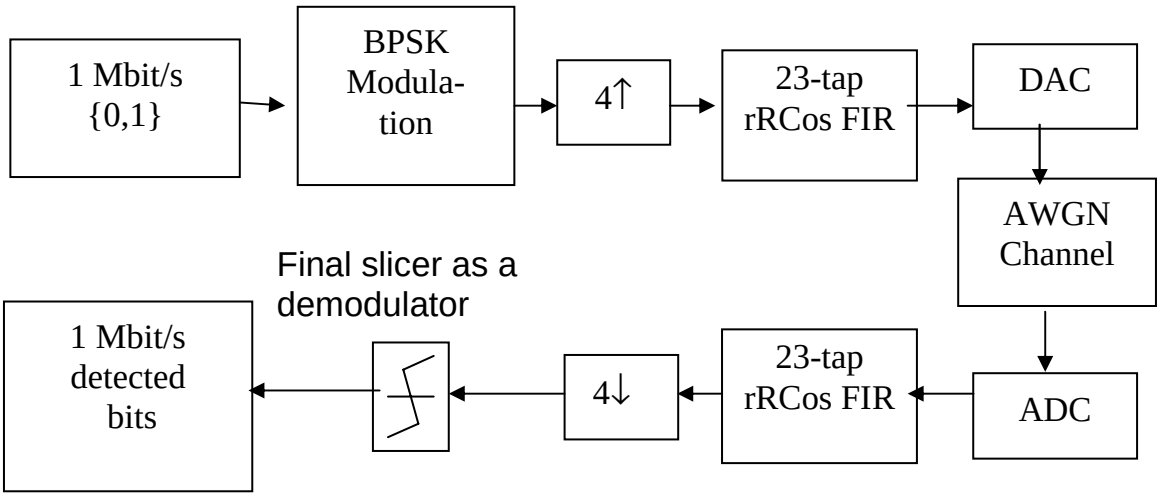


Fig. 3-5 A BPSK system.
The adders, filter coefficients and gain output of the root-raised-cosine filters, as well as ADC, suffer quantization noises.

Second, we validate our central result (3-7) and (3-26) of this Chapter using the binary-phase-shift-keying (BPSK) base-band transceiver in Fig. 3-5. Two root-raised-

cosine FIR filters, each with 23 taps, act as band limiter and matched filter, respectively [46]. The slicer, as a demodulator, makes decisions on transmitted data based on the signal polarity of its input. Fig. 4 shows that the probability of decision errors in FP system, calculated as a function of MSE of quantization noise using the (3-7) and (3-26), indeed agrees well with simulation results with various word length realizations of all the fixed-point operators in the system.

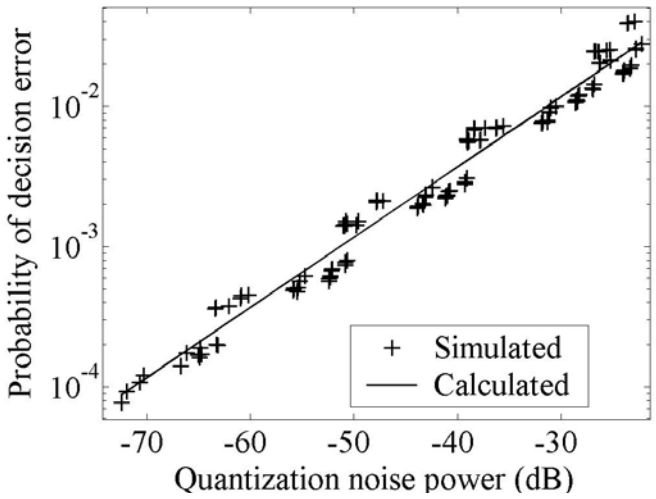


Fig. 3-6 Calculated and simulated probability of decision errors for BPSK.
BPSK Calculated curve is from (3-7), where γ=1 and $p_x(0)$ is obtained from (3-26) with one BER type estimation using an additive i.i.d. sequence {0.1, -0.1} with equal probability.

3.6 Summary

Two examples were given to illustrate and support our analysis of the effect and probability of a decision error. Based on the result, we have extended the FFC methodology to include decision making blocks and decision errors due to quantization. It should be point out that the understanding of the effect of decision errors is far from complete, despite the efforts done here. Nevertheless, the categorization of soft and hard

errors in Section 3.4 and the continuation of FFC methodology in Section 3.5 partly enable us deal with them.

Chapter 4
Automated FFC Tool

This chapter explains a floating-point to fixed-point conversion (FFC) tool that has been implemented for digital signal processing and communication systems. This tool automates the floating-point to fixed-point conversion (FFC) process for digital signal processing systems. The tool automatically optimizes fixed-point data types of arithmetic operators, including overflow modes, integer word lengths, fractional word lengths, and the number systems. The approach is based on statistical modeling explained in Chapter 2 and 3, hardware resource estimation to be explained in Chapter 5 and global optimization based on an initial structural system description. The basic technique exploits the fact that the fixed point realization is a weak perturbation of the floating point realization which allows the development of a system model which can be used in the optimization process.

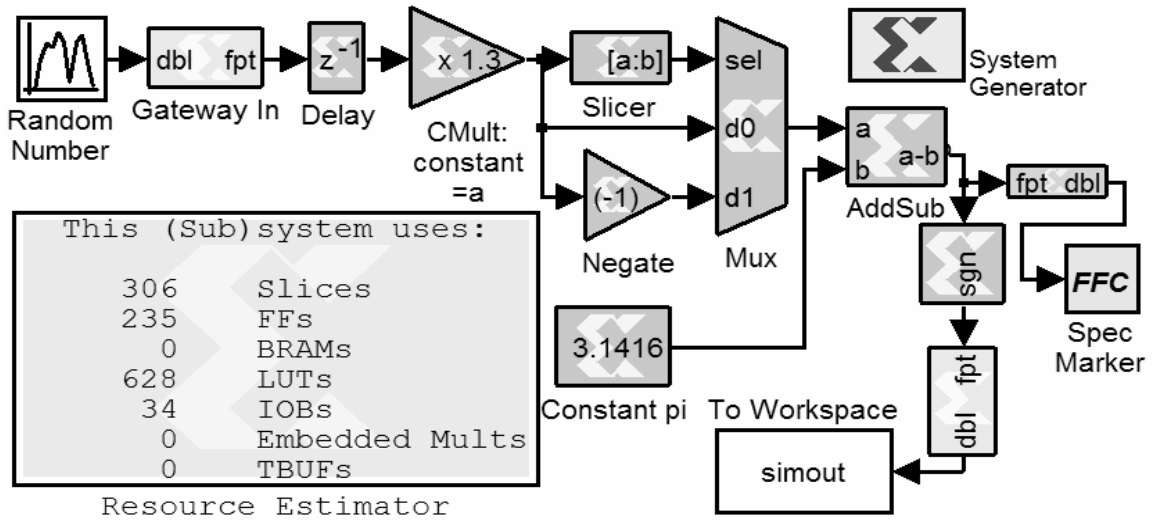
4.1 Introduction

The advances of software and hardware co-design environments enable us to capture architectural information at an early design stage and conduct bit-true and cycle-accurate simulations. The verified design can then be mapped to hardware automatically.

A number of platforms that are based on Simulink have been successful, targeting either FPGA's, such as in System Generator from Xilinx, or ASIC's. Fig. 1 shows such an algorithm description in both algebraic form and structural form that is implemented using System Generator blocks. If 32-bit word-lengths are specified throughout the design, the numerical precision is typically sufficiently high that the structural form is verified to perform the target algebraic functionality without errors from finite wordlength effects, i.e. it is essentially a floating point description. A resource estimation tool based on the Simulink description has been developed that automatically calculates FPGA hardware requirements, such as slices, used in the design. For ASIC designs, a similar tool could be developed which estimates the chip area requirements based on the number of cells used in the block realization. To reduce these hardware-costs, increase the throughput, and reduce power, one essential step is to optimize the fixed-point data-types to use the minimum word lengths possible—the task of FFC.

$$y(n) = \mathrm{sgn}(|\, a \cdot x(n-1)\,| - \pi)$$

(a)



(b)

Fig. 4-1 A simple algorithm in System Generator.
(a). The algorithm in algebraic form; (b). A high-precision architectural version with the output of the hardware "Resource Estimator".

For many application domains and algorithms, high precision in the computation is wasteful and significant hardware reductions are possible. However, determining the minimum requirements through manual optimization is time-consuming and rarely optimal [13].

Chapter 1 gives the motivation and briefly describes our methodology of an automated floating-point to fixed-point conversion (FFC) tool. Chapter 2 and 3 provide some novel analytical results originated at this purpose. This chapter illustrates in more details on how these results can be used collaboratively to an implementable automated FFC. As seen in Fig. 1, the designer inserts an FFC utility block, "Spec Marker", from the FFC library into the system at critical nodes, which is used to enter user specifications

of the system performance and the tool then automatically produces a system with optimized fixed-point data types. Though implemented targeting FPGA's, the same methodology may be applied to other hardware, such as ASIC's, provided a hardware-cost estimator is available.

Section 4.2 gives a more detailed review of existing techniques. Section 4.3 describes our strategy with emphasis on the techniques used in improving efficiency of the automated optimization. Sections 4 and 5 compare the performance of this approach on several typical designs with previously described optimization techniques.

4.2 Further review of the past techniques and our design environment

4.2.1 Past techniques

Section 1.3.1 briefly reviewed existing FFC techniques without exposing too many details. This is extended in this section as you have been more familiar with FFC by now. Those existing FFC tools [13-17] using normal *C* or *C++* for simulation and system description take advantage of the higher simulation speed of these tools over graphical editors. To push this further, [49] developed new compiler to speed up the simulation. However, the disadvantage of sequential language such as *C* or *C++* is their lack of support for architectural information of a system. As explained in previous chapters, architectural information affects the finite word length effects. So, considerable amount of efforts of these past techniques focus on developing the support for architecture information in *C* or *C++*. Yet some aspects of the system architecture are still very implicit. As a result, hardware-cost estimations become difficult. For example, the system in Fig. 1 includes a MUX that would naturally be implemented as an "if-else-end"

statement in *C* or *C++*. Yet, in most of the past techniques, this statement is modeled to take no hardware area; consequently, Mux's are often modeled to take no hardware.

Besides their simulation environments, the other common feature in the past techniques is how to determine the number system, the integer word length, and the overflow mode for each signal node. By extracting the signal statistics using simulations, the signal range can be predicted. Thereafter, the fixed-point data types are set to hold this range with high confidence. The most complete description of this method is given in [15]. Let's explain this method from our optimization point of view stated in Section 1.2. In fact, if overflow happens frequently, these overflow errors that are large relative to quantization errors on the fractional part are likely going to cause system failing the constraints in equation (1-2). So, a simple method that prevents it is to separate this "avoiding overflow" constraint from the original ones. This action results some losses of optimality of the solution to (1-2). Later in robust optimization part of this Chapter, we are going to support this idea by arguing the loss of optimality is small.

On the other hand, the past FFC techniques mainly differ on how the fractional word-lengths and quantization modes are determined. In both [13] and [14], no gross hardware-cost function are given nor mentioned. There, the FFC problem is not treated as an optimization explicitly. However, their implicit goals are to minimize all the word-lengths at the same time. The task of minimizing multiple objective functions simultaneously is unrealistic unless the constraints are separable to constraints that each only depends on one variable.

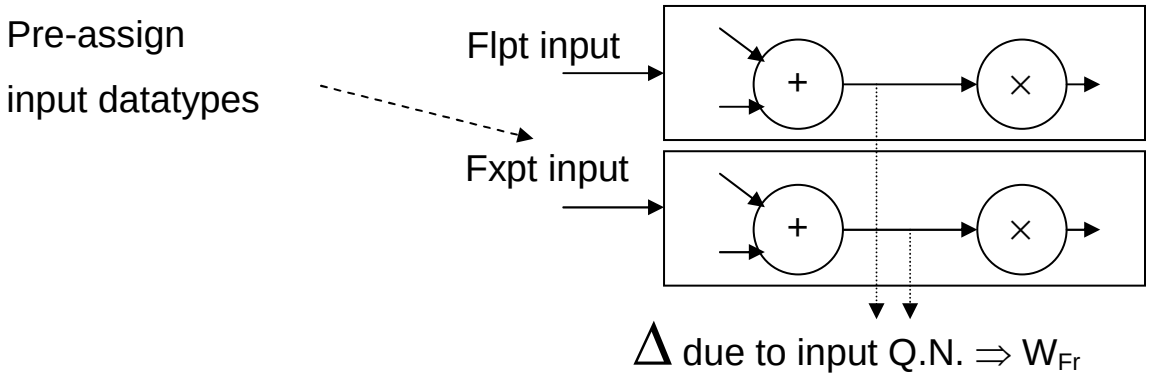


Fig. 4-2 "Guided" FFC methodology [14]

Fig. 4-2 summarizes the FFC methodology of [14]. There, one constraint function is assigned for each word-length in the following way. The input word-lengths are pre-assigned, and the fractional part are set to be sufficiently large so that the local quantization noise power is much smaller than the one caused by quantization of the inputs. These strongly decoupled constraint functions are always feasible and can minimize all word-lengths at the same time. However, the gross quantization effects from these locally justified quantization sources altogether can still be much greater than the one induced by quantizing the input. Therefore, it is still necessary to have a final constraint on system performance, such as SNR or BER, as a function of all the word-lengths. This results a possibly unbounded number of iterations in [14].

In [13], some unjustified pre-assignments of date types on a set of selected signal nodes provide a number of constraint equations. The deterministic propagation methodology yields inequalities among the fractional word-lengths; for example, the fractional word-length at the output of a multiplier should be no less than the sum of those of the two inputs, while the output fraction word-length of a delay component should be no less than the input one. Besides being overly pessimistically considering

quantization effects, this approach may result contradictive inequalities under the presence of feedback loops such as the one in an accumulator. This is prevented in [13] by possible user interaction using engineering decisions, which causes undetermined design time and design efforts.

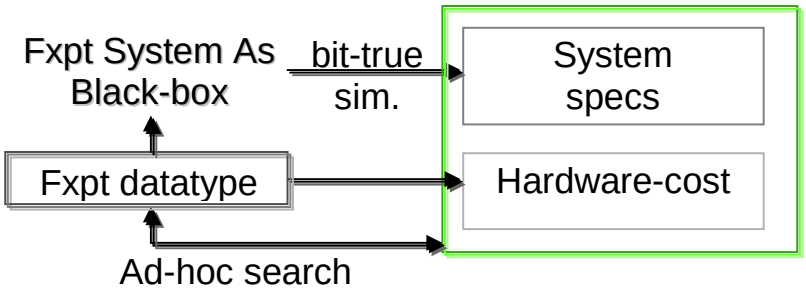


Fig. 4-3 Adhoc search FFC method [15]

Fig. 4-3 summarizes [15] and some other works published by Dr. Sung's group. Similar approach is taken in other works such as [92]. The problem is formulated in an implicitly similar way as ours. However the lack of investigations on the closed form specification function and simulation efficiency limits their optimization algorithm to be heuristic and time-consuming search. In addition, the Monte Carlo simulations among iterations can be inconsistent which adds further complications. Furthermore, their hardware cost function is manually input to the optimization scheme, and is lack of justification.

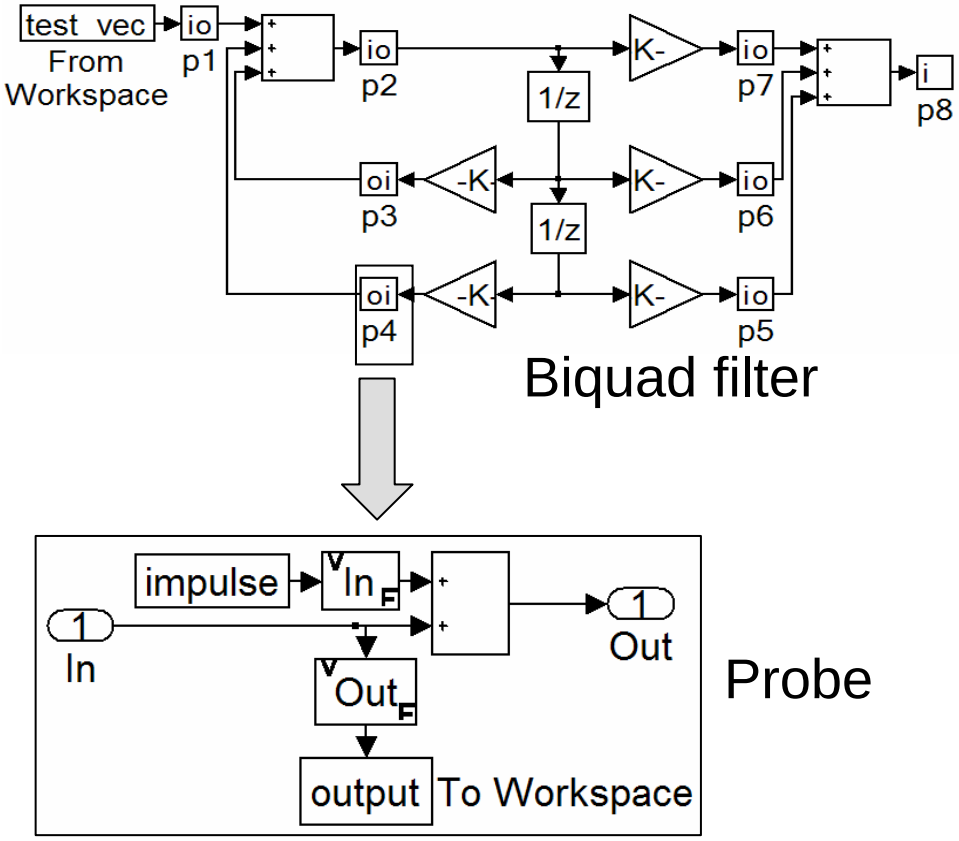


Fig. 4-4 Impulse probing method to get transfer functions [3].

Dr. Jain and his colleagues studied the wordlength optimization of linear filters [93][94]. Using either extensive specialized analytical result or guided simulations, the word lengths of constant coefficients instead of data path are optimized. [94] further extend the topic to integration with synthesis and layout tools to provide a total solution of FIR filter design, which is beyond the scope of this thesis. However, the FFC tools only work for part of LTI systems.

In my master thesis [3], I proposed an automation using the same optimization point of view as (1-2) for linear-time-invariant (LTI) systems. Each quantization noise source on the LSB side is treated white, and uncorrelated to each other. The quantization

noises from all different sources accumulate at the output through linear transfer function in frequency domain. We insert a test impulse input, called probe, to each quantization source and test the output response using an impulse response simulation; therefore, we can obtain the impulse response for each noise source automatically. And these impulse responses provide the transfer function in frequency domain. Being very efficient, this approach requires the system to be LTI.

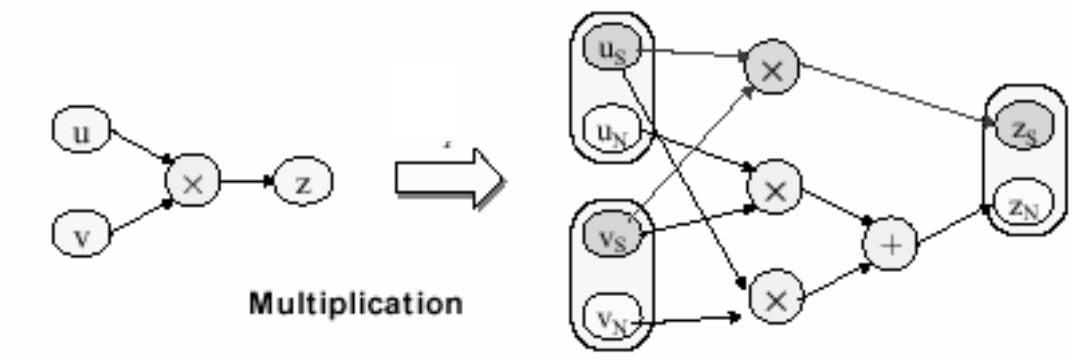


Fig. 4-5 A graphical translation method to find quantization effects [16].
This figure is copied from [16].

Recently, [16] proposes another interesting new method that is heavily analytical. It also assumes a quantization at the LSB side generates a white and uncorrelated noise source. This noise propagates in the system. If part of the system is LTI, the transfer function theory similar to the one in [3] applies. However, in stead of basing on simulations, a graphical translator is used to directly understand the structure of the system to obtain the transfer function. Furthermore, the method extends to non-linear systems that have no feedback loops by modeling that a quantization noise propagates through a system one operator after another without considering the possible correlations. The limitations in this approach are two folds: nonlinear systems with feedbacks are not modeled; second, the inputs of an operator may be indeed correlated to each other since they originate partly from the same quantization source earlier in the system.

Finally many optimization procedures similar to [15] but with different optimization procedures are tested and compared in [17]. It appears from the comparisons that the most promising procedures are those based on simulation with the assumption that increasing word-length improves system performance and increases hardware-cost, but otherwise are unguided. However, as showed in one example 1 of Chapter 2, this statement is not always true, which shakes the foundation of these procedures. Another critical drawback of this method is the simulation time which can be unreasonably large for complex systems and thus sub-optimal solutions are often obtained. Furthermore, these approaches use a system description that doesn't contain architectural information, which compromises their results, since the architecture-free high-level hardware-cost estimations are often very crude.

As pointed out in previous Chapters, we will also use simulations to evaluate various sensitivities, but will use these results to develop a model of the system which is then used in the optimization problem (1-2). This allows us to dramatically reduce the amount of simulation required.

4.2.2 Simulation environment

We choose Xilinx System Generator embedded in Simulink as our system input and simulation environment. First, it is able to capture architectural information much better than any of the environments built in past techniques except for [3] where we used Simulink which is essentially the same as System Generator. This graphical input also fit seamlessly with Berkeley Emulation Engine (BEE) project [89] and Simulink to Silicon Hierarchical Automated Flow Tools (SSHAFT) project [56] in Berkeley Wireless

Research Center. BEE project starts from fixed-point system in System Generator; then it partition and map the system to multiple Xilinx FPGA chips for fast system emulation. At the same time, it can map the same system to application specific integrated circuits (ASIC) design. We believe designs of complicated systems should eventually be done completely by algorithm and system designers with understanding of hardware, whereas the rest of the implementation should be automatically done by design flows. Our choice of Xilinx System Generator is based on popular software Matlab and Simulink from MathWorks. This environment is often familiar to these high-level designers.

Chapter 5 describes a tool integrated in Xilinx System Generator that can estimate high-level hardware-resource fast and accurately [60][64][91]. This estimation tool meets our needs for a hardware-resource estimator to retrieve hardware information automatically.

Furthermore, Xilinx System Generator blocks allows a block to be simulated as either floating-point (double precision) system, full-precision fixed-point system, or user-defined fixed-point system. By applying simple parametric changes, the system can act as any of the three systems. This feature basically accomplishes one of some major tasks in some past FFC techniques. Adopting it allows us to concentrate on other important aspects of FFC.

Our choice of this graphical input environment just allows us to be concrete on the discussion in rest of the chapter, especially regarding the automation part of our methodology. We foresee no important technical difficulties to effectively implement our methodology in other environments, such as DSP Canvas [92] based on C++, SystemC

[61][79] based on C++, AccelChip [62] based on Matlab, Simulink itself, SPW DSP simulation environment [95] that is similar to Simulink, Ptolemy [96] from University of California Berkeley, and probably many others that I do not know. In fact, the methodology promoted in this thesis can almost be directly applied on them. Implementations of a system in some higher level languages among this list make the simulation and initial compilation of the system relatively faster since no graphical information needs to be supported, that is, architectural information needs not to be graphically displayed and maintained. This seems to promote this kind of environment over block-diagram based tools.

However, graphical information shown in platforms such as Simulink helps to reveal the relationship between functional blocks, and becomes invaluable in maintaining the system. Furthermore, Simulink has the feature of using Matlab scripts to generate and modify a model [3]. All these features make Simulink as well as System Generator easy to use and to automate. The naturally hierarchical description of the system also makes a library approach easy as well. In my opinion, this is the best environment to demonstrate different FFC methodologies. The sister EDA tools associated with Simulink, some of which are developed inside Berkeley Wireless Research Center by my colleagues, further increase the value of implementing FFC in this environment.

More detailed descriptions of System Generator and Simulink can be found in Chapter 5, Appendix C, and many of the references such as [3][89].

4.3 Automation and Implementation of FFC

Admitting that numerical simulations become slower using graphical editor for numerical simulation, our strategy concentrates on speeding up FFC by understanding the problem in three important aspects of optimization problem (1-2).

First, we ensure complete design automation by obtaining direct access and control of the fixed-point data types of each block. Furthermore, we implemented the tool in such a way that minimal cares are needed from the floating-point system designers. That is, our FFC tool can read a design that is either completely fresh for optimization or with some parts already optimized. In addition, block grouping is supported and recommended for faster conversion.

Second, we make efforts on understanding the hardware cost as a function of fixed-point data types and how to retrieve this information automatically. This becomes possible since the architectural decisions have been made, either permanently or temporarily, before FFC.

Third, we make efforts on understanding the constraints as functions of fixed-point data types, and how to retrieve this information automatically. This includes not only the relationships themselves, but also the choice of constraint functions. We will show some constraint functions are time-consuming to simulate or difficult to be treated as constraint functions or both, whereas some others are very easy to obtain. Good choices of constraint functions are essential for a fast and accurate FFC tool.

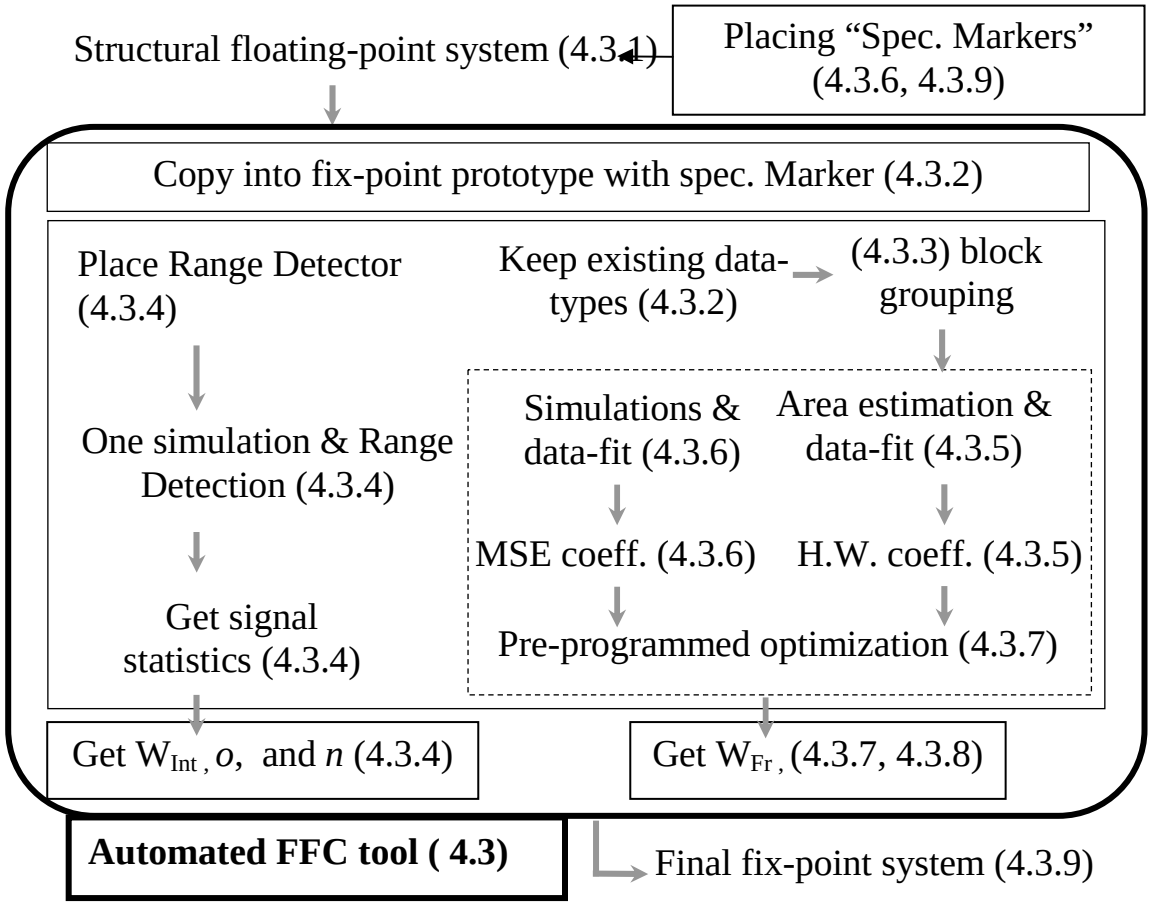


Fig. 4-6 Detailed FFC automation and design flow graph.
The box with bold line shows the details of the tool. The section numbers in parentheses show the sections in which the corresponding topics are covered in detail.

Fig. 4-6 shows the design flow graph of our FFC. In the following subsections, these aspects are address one by one; some other important but probably less novel considerations are also described here.

4.3.1 Tool Infrastructure

The first task of our FFC is to copy the floating-point system, possibly with fixed-point parts, into a temporary prototype system. This ensures the consequent modifications from our FFC tool leave the original system untouched.

Total automation of our FFC tool relies on complete accessibility and controllability of the Simulink model in Matlab environment. Fortunately, this is well supported by the Model Construction Command set, including commands such as find_system, set_param, get_param, add_block, delete_block, add_line, and delete_line [3][59]. With these commands, we can write scripts to sort out all the blocks in a system, to understand how blocks, lines and ports are connected, to set fixed-point data types as variables, and so on. In fact, an extension of using these commands is to modify the model architecture [3], which is beyond the scope of this thesis.

Each block may have input and output ports. System Generator allows the user to specify the fixed-point data types of the output ports of most blocks with some exceptions; for example, a Delay block has its output data type follow its input data always. Some signals can break into multiple branches, each of which feeds into different blocks. It is possible that different branches of the same signal need different word-lengths. This can be done by inserting a Quantizer block at each branch. Nevertheless, we ignore this dimension of optimization because cutting the number of independent quantizers is one key to speed up optimization. Thus, the output data types of all blocks fully characterize the fixed-point implementation of a system. For better controllability, we maintain this output data type information in a Matlab structure, named *block structure* (bs), with fields describing data types. Fig. 4-7 shows the Matlab initialization of one of the block structure. The “o_wf_master” field designates the fractional wordlength group to which the block belongs, and this is explained in Section 4.3.3.

```
bs{i}.bh=blocks{i};    % blocks{i} is the handle of the i-th
                       % block after sorting bs{i}.range=[ ];
                       % block output range statistics. bs{i}.sign= [ ];
                       % block number system. bs{i}.omode='s';
                       % block output overflow mode.
bs{i}.qmode='r';       % block output quantization mode. bs{i}.o_wi=
Inf;                   % block output integer word length.
bs{i}.o_wf= ['my_fxpt_o_wf_'  int2str(i)];
                       % block output fractional
                       % word length, as a string variable

%% variables on the right side of the following expressions
%% have been obtained using get_param( ) command
bs{i}.old_sign= arith; % block number system in
                       % floating-point (flpt) system
bs{i}.old_o_wi= w-wf-(arith=='s');
                       % flpt output integer word length
bs{i}.old_o_wf= wf;    % flpt block fractional word length

%% in word length grouping, the following variable indicates the
%% master word length the current block is to follow
bs{i}.o_wf_master='';% default is empty

%% following variable determines whether the output data type in
%% flpt version is to be used in the fxpt system.
bs{i}.use_old=0;       % default is 0 as not to use flpt.
```

Fig. 4-7 Initialization of the i^{th} block structure in Matlab.
% start Matlab comment for the rest of the line.

4.3.2 Keep useful fixed-point information

A floating-point system in System Generator contains many signals whose fixed-point data types are already specified. As described in Chapter 2 and 3, we call them *logic signals*. As an example, Fig. 4-8 shows that the Select port of a 2-input multiplexer must be 1-bit unsigned, which must be specified in this way even in a floating-point system. No change is needed on these signal data types. Therefore, it is important for the block structure to remember the floating-point data types. Fig. 4-7 shows that the fields

“old_sign”, “old_o_wi” and “old_o_wf” are used for this reason, and the old data type is used when field “use_old” is set 1.

An analysis of the library blocks is therefore initially undertaken to define the obvious logical signals, but even some arithmetic signals, which have been pre-defined (such as those determined by availability of existing hardware) are also considered to have “fixed” data-types and left untouched. These signals can be simply tagged by the system designer, or configured through an additional rule, which for example may limit the signal data-types to be greater than a minimum value and when that minimum level is reached in the optimization are set to fixed status. The remaining signals are all considered “arithmetic” and subject to be changed freely by the FFC tool.

4.3.3 Block grouping

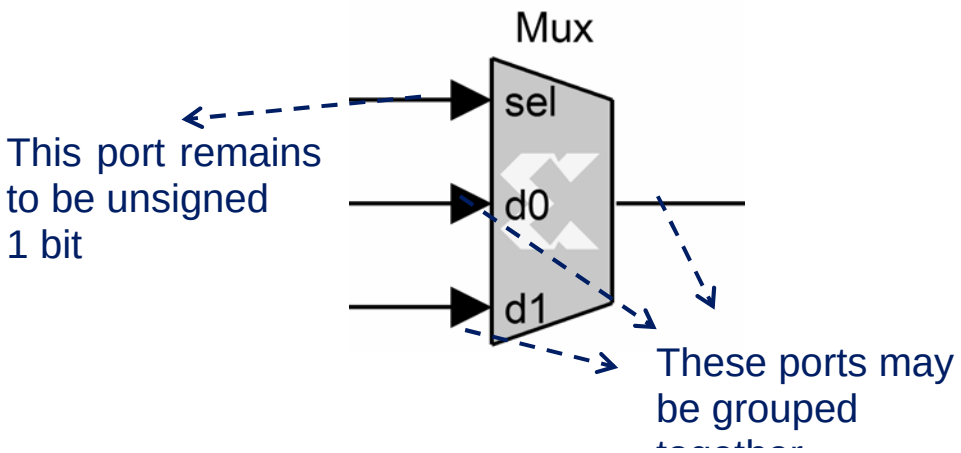


Fig. 4-8 Possible grouping rules for a multiplexer

Section 4.3.4 will discuss how to determine the variables on integer sides, whereas Section 4.3.5, 4.3.6 and 4.3.7 concentrate on the fractional part. It turns out with all the efforts the optimization problem in (1-2) is still too complex, and it is generally

necessary to reduce the optimization space. This is done by grouping some blocks together to have the same fractional word-lengths [15]. Fig. 4-8 shows that some signal ports can be grouped together quite intuitively. For example, we might want to force a Mux's two signal inputs to have the same fixed-point data type, and so might be its output. This means the block in front of the two signal inputs and the Mux itself may be group together to have the same fractional word-length at their outputs, that is, they should have the same "o_wf_master" field. Grouping more blocks together can greatly reduce the optimization space in (1-2), leading to faster optimization. Table 4-1 lists the rules that we developed so far. The more rules are used, the faster FFC becomes, and the less optimal the resulted conversion becomes. Sometimes, we shall not group one block with others because the output feeds into many other blocks, which means the output will largely influence hardware cost and finite word-length effects. This observation is reflected in rule 1, 3, 5, and 6.

Rule number	Rule description
0*	Group inputs and outputs of Delay, Register (except for Reset port), Up sampler, and Down samplers
1	Group inputs of an Add/sub, unless anyone of them are inputs of 5 or more blocks
1.1	Always group two inputs of an Add/sub
2	Group inputs and output of an Add/sub, unless its two inputs are not grouped or its output is not connected to an Add/sub block
2.1	Group inputs and output of an Add/sub, unless its inputs are not grouped
3	Same as rule 1, but for Mux block
3.1	Always group inputs of a Mux
4	Same as rule 2.1, but for Mux block
5	Same as rule 1, but for Relational block
5.1	Always group inputs of a Relational block
6	Same as rule 1, but for Logical block
6.1	Always group inputs of a Logical block
7	Same as rule 2.1, but for Logical block

Table 4-1 Grouping rules
*Rule 0 is always applied.

One example of a rule specification is “rule=[1.1, 5]”, which means rule number 1.1 and number 5 are to be applied. Refer [60] for block descriptions.

Before grouping a system automatically, we have to figure out the connectivity among functional blocks, where we differentiate blocks in a graphical editor as either *functional* or *supportive*. Fig. 4-9 shows a simple system with a Multiplier following an Adder. Both the Adder and Multiplier block are functional since they are essential to accomplish the desired signal processing. On the contrary, the Subsystem block and the associated In and Out blocks in Fig. 4-9 are just supportive and make the hierarchical description possible. In current Simulink environment, the only two other supportive blocks are From and Goto blocks, which are used to eliminate long wires in Simulink. All other blocks are functional. Finding the connectivity among functional blocks is basically to flatten the design hierarchy. This is important since grouping involves frequent cross-references of adjacent functional blocks.

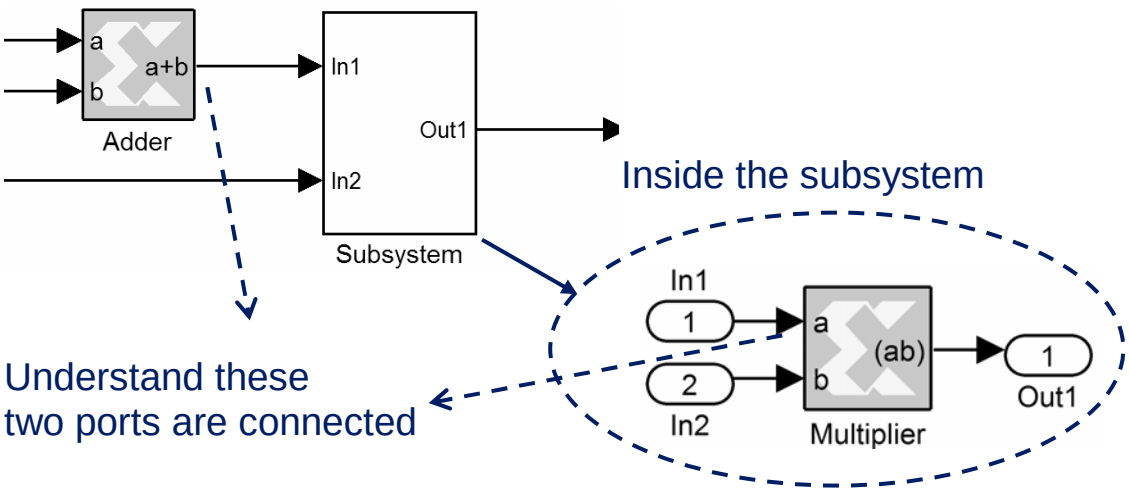


Fig. 4-9 Resolving block connectivity.

Fig. 4-10 shows our iterative strategy on resolving functional block connectivities. Inside the loop, we exchange the connection information between blocks and between ports in supportive blocks. In this way, the supportive blocks are treated to be transparent. The iteration is completed when all ports know their adjacent functional blocks; then the

port information is saved into a subfield of “userdata” cell (cell is a Matlab data type [59])— “userdata.appendix”— associated with each System Generator block. “Userdata” is a standard variable supported for each block in Simulink. To prevent possible erasion of “userdata” created by floating-point system designer, the original data are obtained using get_param(.) command and saved in a subfield “userdata.backup_userdata”.

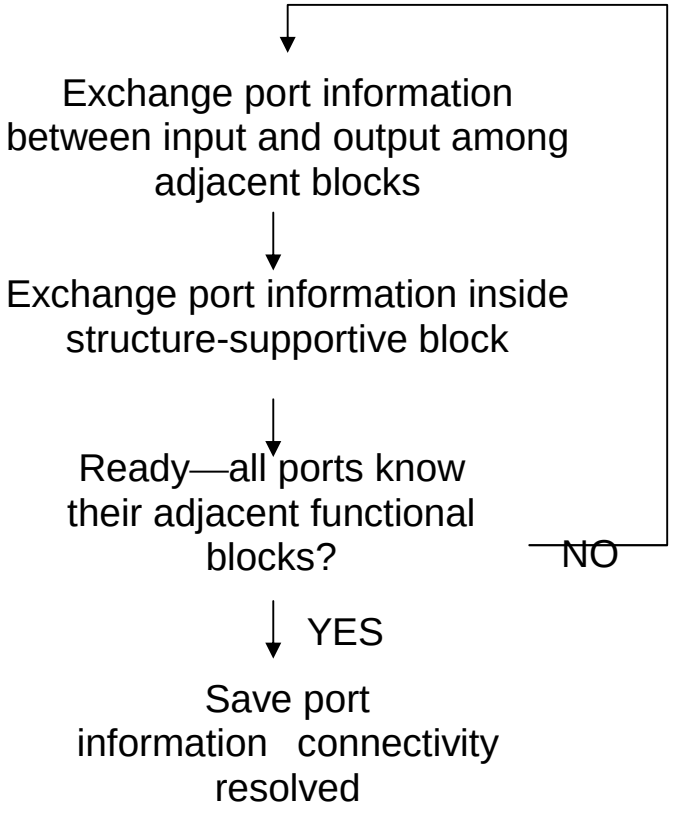


Fig. 4-10 Algorithm to resolve functional-block connectivity.
This is done by considering supportive block transparent.

Once connectivity is resolved, the tool proceeds to grouping according to rules described in Table 4-1. Fig. 4-11 shows our grouping strategy. Each rule causes more blocks placed in the same group list. When all rules are done, the list is simplified to remove any redundancy in a group. Then a master cell array variable, called “wl_master”, remembers the blocks in each list (group), and the index of wl_master is written to

"bs{i}.o_wf_master". Then, we can easily cross-reference the corresponding "wl_master" and "bs". In the mask of System Generator block, we use set_param(.) command to set the fractional word-lengths to be a variable, named "my_fxpt_o_wf_i" and saved in "bs{i}.o_wf". Thereafter, we may easily set the value for these variables in Matlab, and the system automatically uses their new values for simulation or hardware resource estimation. This accomplishes automatic control of system parameters to prepare the system for a large number of Simulations in the following few sections.

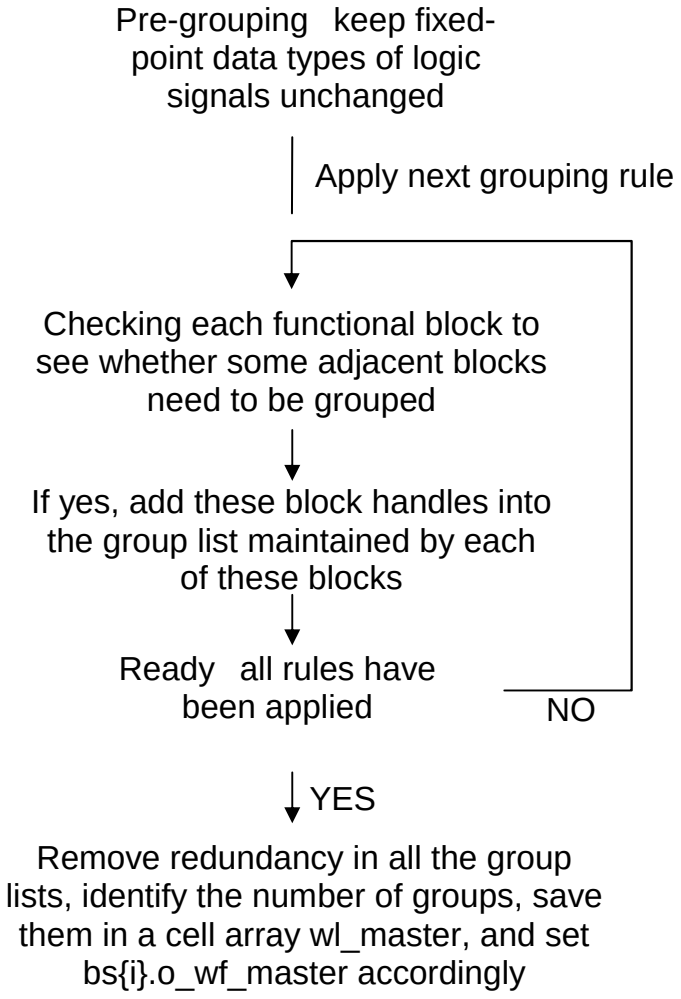


Fig. 4-11 Grouping methodology.

The first step in Fig. 4-11 is called pre-grouping, which figures out who are interpreted as logical signals in a floating-point design. Table 4-2 shows our pre-grouping rules that are currently used. The last pre-grouping rule simply ignores those signals that already are of light weight in fixed-point data type. This is to catch some logical signals that other rules possibly missed. Furthermore, it appreciates floating-point designer's decision on fixed-point data types of some of the signals.

The next few sections explain how to determine the other fields in the block structure in Fig. 4-7. Once all the fields are obtained, we can again use set_param(.) command to substitute the variables, which are previously set in block parameters, to their final values and save the system as the FFC output fixed-point system as shown in the last step of Fig. 4-6.

In summary, this Section discusses our strategy on managing system infrastructure related to FFC problem in System Generator and MathWorks environment. With proper modification, similar infrastructure and grouping rules can apply to other structural system descriptions.

Rule description
All Boolean signals stay to be Boolean
All blocks proceeding Enable ports, Mux Select ports, Memory address ports, and Register Reset ports stay untouched
Outputs of Relational blocks stay untouched.
Blocks connected to Inverter block, Slice block (with some exception), Bus Concatenator block, and Type Reinterpreting block, stay untouched
All signals that are less than 6 bits in floating-point design are considered optimized and thus stay untouched

Table 4-2 Pre-grouping rules to identify logic signals
Refer to [5] for detailed description of the blocks mentioned above.

4.3.4 Integer overflow

As stated in below, we follow a strategy similar to those used in [14-17] to treat the fixed-point data types related to the integer part, comprising W_{Int}, *n* (number systems), and *o* (overflow-mode). We briefly repeat the strategy and describe how to implement it in our environment. Finally we discuss its drawbacks.

We assume overflow noises hurt the behavioral quality of the system so greatly that they should be avoided by all means. Monte Carlo simulations of the floating-point system with the input test vector provide a large set of data for each signal node. Based on these data we can record the maximum and minimum value and estimate the first few of its statistical moments. From these statistics and a few commonly encountered signal probability distribution functions, we can predict the signal range and determine the

integer word-length accordingly by making it just large enough to cover the range. The number system is set to be unsigned if the signal is always positive, and otherwise signed 2's complement. If the maximum value occurred in simulations is within the predicted range, wrap-around mode is to be used for overflow; otherwise, saturation mode is used (as a confidence guard). In this approach, we implicitly assume that hardware-cost monotonically increases as W_{Int} increases, as *n* switches from unsigned to signed for a positive number (because of one more bit needed for the sign), or as *o* switches from wrap-around to saturation (because of the extra logics needed). Consequently, we effectively separate all integer fixed-point data types in the optimization problem in (1) into independent optimization problems, each of which has the parameters related to only the integer fixed-point data types of one signal node.

One way to calculate the statistics of each signal node is by saving all its simulation data to a Matlab workspace variable that is processed after simulation. This approach becomes infeasible because possibly thousands of signal nodes can produce millions to billions floating-point data in a long Monte-carlo simulation—impossible to store. The other approach is to do running average during simulations. We write a Simulink s-function block in *C*, called Range Detector. Once placed in a system and linked a signal node to its input, this block can do running-average estimation of the first four moments of its input during a simulation. Internally only the current averages and sample size—totally five double precision numbers—are saved for each of these Range Detectors during a simulation. Only at the end of a simulation, the final averages are saved to Matlab workspace as our estimations of signal statistics.

Using add_block(.) and add_line(.) commands, one Range Detector is automatically placed into the system after each block corresponding to each entry of the block structure defined in Fig. 4-7. A simulation with all these Range Detectors tells us the signal statistics. Because of the extra computations during a simulation with these Range Detectors, the simulation time is found to increase by up to 100%. However, because no iterations of simulations for range detection are needed in our algorithm showed in Fig. 4-6, this extra simulation time is acceptable.

After the signal statistics are estimated, our FFC tool automatically removes them using Matlab delete_block(.) and delete_line(.) commands. Then the system is ready for connectivity resolving and grouping, as mentioned in Section 4.3.3.

Once determined, these fixed-point data-type variables on the integer side are dropped out from the optimization problem (1-2). Then, (1-2) becomes

$$\begin{aligned} &\text{minimize hardware-cost} \\ &f_{\mathrm{HW}}(\mathrm{W}_{\mathrm{Fr},1}, \mathrm{W}_{\mathrm{Fr},2}, \ldots; q_1, q_2, \ldots) \\ &\text{subject to specifications} \\ &S_j(\mathrm{W}_{\mathrm{Fr},1}, \mathrm{W}_{\mathrm{Fr},2}, \ldots; q_1, q_2, \ldots) < 0, \forall j. \end{aligned} \tag{4-1}$$

Before discussing how to determine the fractional fixed-point data types in next few subsections, we need to explain the drawbacks of the proceeding approach. First of all, overflow-free method is only sufficient, but not necessary, to ensure that the variables on the integer side in (1-2) do not violate the constraint functions. Thus, the proceeding method trades simplicity over design optimality. Second, a more serious problem is the impossibility to avoid overflow completely for many distributions such as a Gaussian whose range is the whole real axis. We may try to model the signal using distributions

that only have finite ranges. This barely transfers the previous difficulty to statistical modeling—we still need to use finite number of data to predict the range in the model. The built in fixed-point data-type propagation in [59] and [60] avoid this difficulty by deterministically propagating integer word-lengths. Unfortunately, this is often overly pessimistic. For example, in a least-mean-square (LMS) algorithm, the residue error signal is usually relatively small after the subtractor between desired signal and filtered input. Then, it is uneconomical to set the integer word-length of the error signal large enough to hold the maximum possible value based purely on integer word-length propagation. The situation becomes worse in systems with feedback loops and long data paths.

Theoretically, the best method is to discover a simple enough relationship between fixed-point data types on the integer side and the constraint functions in (1-2), as what we are doing for fractional quantization noise in next few sections. An unsatisfying attempt in [3] assumes that an overflow noise hurts the system decision with certain average probability. Then we can link the bit-error rate requirement to the maximum probability of overflow incidence. Unfortunately, this "certain probability" is indeed unknown, system dependent, and difficult to simulate, which makes the approach impractical.

In general, the overflow noise is statistically dependent on signals, which causes difficulties on noise modeling. The overflow noise may also be large compare to signal, which causes non-linearity effects. Finally, overflow noise comes with small probability, which makes digital simulation of its effect difficult. Therefore, we think it remains an

open question for both theorists who want to model overflow noise and also for FFC designers who want to avoid long and numerous digital simulations.

Nevertheless, in practice the method adopted in our FFC works well. The robust optimization part of Section 4.3.8 explains some fundamental reasons for this to be true.

4.3.5 Analytical hardware resource estimation

The most immediate task of the optimization problem in (4-1) is to find out the hardware-cost function, specification functions, and their relationships to W_{Fr}, defined as the vector $[W_{Fr,1}, W_{Fr,2}, \ldots]^T$, and the q-modes.

Only one hardware cost function is to be minimized in (4-1). This could be area, power consumption, power delay product, and so on. High-level estimations of hardware resources such as area, energy and delay have been studied extensively, as detailed in Chapter 5. For system level optimization, it often suffices to adopt the approach based on parameterized library. The area of each block in the library can be modeled as a function of parameters related to fixed-point data types as well as other important technology factors such as feature size and voltage. Provided the architecture choice with all other parameters fixed, the area cost of a library block is uniquely characterized as a function of the fixed-point data-type parameters. The total area of the system can then be estimated as a sum of all the required blocks plus a certain routing overhead. This usually yields a hardware-cost that is a quadratic function of $W_{Fr,i}$ with coefficient affected by q_i, that is,

$$f_{\mathrm{HW}}(W) \approx \frac{1}{2} W_{\mathrm{Fr}}{}^{T} \mathrm{H}(q) W_{\mathrm{Fr}} + h(q)^{T} W_{\mathrm{Fr}} + h_0 . \tag{4-2}$$

Chapter 5 describes my summer intern work at Xilinx, Inc. on developing a resource estimation tool in Xilinx System Generator environment. Given a system designed in System Generator environment, the tool can estimate the hardware resource accurately. Each type of block associates with one Matlab function, written based on complete understanding on how the corresponding block hardware is designed in FPGA. Once the block parameters, such as its input and output fixed-point data types, are provided, the function can calculate the hardware resources efficiently. The tool considers both input and output data types because trimming effects can happen at placement and routing stage. This happens when part of the logics are eventually trimmed away since they are dangling. For example, when only the LSB of a multiplier is used, the multiplier is just an AND gate with all other logics removed at placement and routing. (This should happen at least for a good placement and routing tool.) The tool estimates hardware resources normally within seconds, which is several orders of magnitude faster than any previously existing method. The relative error comparing with final implementation is usually within 5%. The tool has been included in version 3.1 of System Generator [60].

We use the proceeding tool to conduct experiments, and to function-fit the coefficients of *f* in (4). Here *f* is an affine function of H, *h*, and h_0, and can be written as

$$f_{\mathrm{HW}}(W_{\mathrm{Fr}}) = W_{\mathrm{long}}^{T}\mathrm{H}_{\mathrm{long}}, \tag{4-3}$$

where the long column vector $\mathrm{H}_{\mathrm{long}}$ captures all the independent entries of symmetric matrix H, vector *h*, and scalar h_0,

$$\begin{aligned}\mathrm{H}_{\mathrm{long}} = (&\mathrm{H}_{11}, \mathrm{H}_{12},..., \mathrm{H}_{1m}, \mathrm{H}_{22}, \mathrm{H}_{23},..., \mathrm{H}_{2m},...., \mathrm{H}_{mm},\\ &h_1, h_2,...h_m, h_0)^{T}.\end{aligned} \tag{4-4}$$

So H_{long} has dimension $[(m\ (m+1))/2 + m+1]\times 1$ where m is the number of word-length groups given in Section 4.3.3. W_{long} is the corresponding column vector formed by the quadratic and linear combinations of entries from W_{Fr}, as well as a constant for h_0. Let $f_{HW,est}(W_{Fr})$ be the estimation using the tool, the relative error can be approximated as

$$\begin{aligned}\frac{f_{HW,est}(W_{Fr}) - f_{HW}(W_{Fr})}{f_{HW}(W_{Fr})} &\approx 1-\frac{f_{HW}(W_{Fr})}{f_{HW,est}(W_{Fr})}\\ &= 1-(W_{long}^{T}\Big/ f_{HW,est})H_{long}\\ &= 1-\hat{W}_{long}^{T}\cdot H_{long}.\end{aligned} \tag{4-5}$$

With different realizations of W_{Fr}, we can obtain a stack of $\hat{W}=(\hat{W}_{long,1},\hat{W}_{long,2},...)$. Suppose the relative error in (4-5) forms a Gaussian random noise, the maximum-likelihood estimation of H_{long} then is given by the following least-square problem [63]:

$$\underset{H_{long}}{\text{minimize}} \left\| 1-\hat{W}^{T} H_{long} \right\|_2 . \tag{4-6}$$

Furthermore we want to keep monotonicity and non-negativity of $f_{HW}(W_{Fr})$, $\forall\ W_{Fr} \geq 0$, which means the minimization in (4-6) is subject to

$$\begin{aligned}&\nabla_{W_{Fr}\backslash} f(W_{Fr}) = W_{Fr}^{T}\cdot H + h \succeq 0, \text{ and } f(W_{Fr}) \geq 0,\ \forall W_{Fr} \succeq 0\\ &\Leftrightarrow \text{each entry of } H, h \text{ and } h_0 \geq 0\\ &\Leftrightarrow H_{long} \succeq 0.\end{aligned} \tag{4-7}$$

So the estimation of H_{long} becomes a Quadratic programming problem [63]. In Matlab, lsqnonneg($\hat{W}^{T}$,1) solves it efficiently.

Once the coefficients are estimated, we obtain an analytical quadratic hardware-cost function $f_{HW}(W_{Fr})$. The quality of the model can by justified by plotting $f_{HW}(W_{Fr})$ versus $f_{est}(W_{Fr})$ for a number of new W_{Fr}. Fig. 10 shows the plot. It is evident quadratic-

fit model works well. On the other hand, if our model of f_{HW} does not include the 2^{nd}-order term H (thus probably under-modeled), it becomes the so-called "linear-modeling" or "linear-fit". Later in Section 4.5, Fig. 4-15 also shows much larger relatively errors using linear-fit. In certain systems, these errors could be too high to validate the model. This justifies the completeness of quadratic hardware-cost model.

One problem associated the hardware-estimation tool of Chapter 5 is the sometime long compilation time for each hardware estimation. When a large number of such estimations are needed, the total compilation time may become quite long. To minimize this problem, after the first compilation, a Matlab script is specially generated using "printf" command, which list all the Matlab functions to be used for hardware-estimation and use variable names as their input arguments. In any subsequent hardware-estimation, the variable values are changed and this script is directly called to estimate the corresponding hardware-cost. This arrangement eliminates most of the compilations and leads orders of magnitudes of simulation time for hardware-cost estimations.

Although we use FPGA resource as the hardware function in (4-1), the approach applies to other functions and to ASIC designs as well. It is therefore only a demonstration of the feasibility of having high level hardware cost as function of fixed-point data types.

4.3.6 Analytical specification functions

To solve the optimization problem (1-2) and its simplification (4-1), we need to repeatedly verify the constraint functions. This is normally done using digital simulations. The cost associated with these verifications can be summarized as

Simulation time = number of optimization iterations ×

simulation time for each optimization iteration. (4-8)

A successful FFC requires total automation; so other simulation cost includes the time to prepare the design automation such as designer's coding time. These additional costs are not considered here. In this section, we find the ways to get less simulation time by reducing each of the two terms on the right side of (4-8).

4.3.6.1 Directly use the difference as specifications

Two most common specifications used in communication systems are signal-to-noise ratio (SNR) and bit-error rate (BER). Unfortunately, blindly adopting these specifications may result intolerably long simulation duration for a reliable conclusion.

Any Monte Carlo simulation only provides finite number of data to estimate a statistical quantity based on a model on probability distribution function (PDF). In statistics, the estimation uncertainty due to finite sample size has been well-studied using two dual methods—confidence interval and hypothesis testing [3][48]. For example, we model bit errors of a communication system as a Poisson process, and denote an estimation of the true BER as $\overline{\mathrm{BER}}$. The estimation uncertainty is denoted as an interval around the estimated value

$$[\overline{\mathrm{BER}} - a(N,\alpha)\cdot\overline{\mathrm{BER}}, \overline{\mathrm{BER}} + a(N,\alpha)\cdot\overline{\mathrm{BER}}], \quad (4\text{-}9)$$

where a is a positive fractional number that is a function of simulation sample size N and confidence level 1-α. Roughly speaking, from an estimation based on N samples, with

probability 1-α, the true BER is within the interval given in (4-9). With large sample approximation,

$$N \cong 4 \cdot k_\alpha^2 \frac{1}{\mathrm{BER} \cdot a^2}\ , \tag{4-10}$$

where k_α is the $1-\frac{1}{2}\alpha$ quantile of a normal distribution [3]. The inverse relationship between N and BER, as well as a^2, results a potentially very large sample size. In the optimization problem (4-1), this kind of simulation needs to be iterated many times, which could be too long to be acceptable [15].

Example 1. If a confidence level=0.95 (so k_α=1.96), BER= 2×10^{-4}, and a= 0.05, (12) gives the sample size about 7×10^6; that is, about 1500 errors need to occur to ensure the small 0.95-confidence interval of [1.9×10^{-4}, 2.1×10^{-4}] given in (11). Hypothesis testing yields similar conclusion [3]. This large sample size could be very slow to simulate in Simulink environment for a relatively complicated system. For example, a binary-pulse-shift-keying (BPSD) system introduced later in Section VIII takes about 8 hours to obtain 2×10^6 samples at the output for each simulation.

■

In an optimization scenario, even more samples for each simulation are necessary to compare the performances of two realizations of the same system. The two realizations can either be floating-point and fixed-point, or be both fixed-point. The comparison is used to judge if a parameter change makes the system perform better or worse. Confidence levels that are too large to reveal the performance difference may result either

wrong decision or wrong suggestions for the subsequent parameter change. Therefore, a very small confidence interval may be necessary.

Example 2. Continuing Example 1, suppose that two fixed-point realizations of the system above have true $BER_F = 2.01\times10^{-4}$ and $BER_F' = 2.02\times10^{-4}$. In order to make sure unprimed fixed-point system performs better than the primed system, one needs a much smaller a as small as 2.5×10^{-3} for each estimation, or $[2.005\times10^{-4}, 2.015\times10^{-4}]$ and $[2.015\times10^{-4}, 2.025\times10^{-4}]$, separately; otherwise, the two confidence intervals overlaps. This corresponds to 2.8×10^{9} samples in (4-10) for both simulations.

■

To alleviate this situation, we directly measure the system degradation caused by fixed-point implementation. This is done by finding the difference at the system output between a fixed-point system and floating-point system under the same input signal. This difference is solely caused by quantization noises occurred in the system. In this way, we avoid the intermediate estimation errors that can easily be so large that the degradation caused by fixed-point implementation is covered. Example 3 below shows the saving by direct measuring the difference.

Example 3: Continuing Example 2, we directly measure the errors due to quantization noises. Assuming these errors are independent to floating-point errors (strictly speaking this is an over-simplification), the BER's due to quantization noises are 0.01×10^{-4} and 0.02×10^{-4}, respectively. To tell the unprimed system is better, we need the accuracy on BER estimation again better than 0.005×10^{-4}. But we only need a about 0.5 and 0.25 for the two simulations. From (4-10), we need 1.4×10^{7} and 2.8×10^{7} samples, separately. This is two orders of magnitude saving from Example 2.

■

The three examples shows that direct measuring the difference due to quantization noises may saves us a lot simulation time. The same argument applies on other statistical specifications such as SNR.

We use mean squared error (MSE) to abstract the difference between floating-point system and fixed-point system, denoted as MSE(flpt-fxpt). This difference is due to quantization noise. Because MSE of a random variable is 0 if and only if the variable is 0 with probability 1 [48], MSE qualifies to reveal the difference. For example, we check the MSE(flpt-fxpt) after the slicer of a BPSK communication system. The slicer produce decisions of either 1 or -1 in both systems; so flpt-fxpt at this node is either ±2 when error happens, or 0 when no error happens. Let p be the probability of an error occur at a time, then

$$\text{MSE (flpt-fxpt)} = (\pm 2)^2 p + 0^2(1- p) = 4p. \quad (4\text{-}11)$$

So MSE(flpt-fxpt) after the slicer is just equivalent to direct measuring the BER. As a specification, we set the difference between floating-point system and fixed-point system much smaller than other degradations at this node that is not controllable at FFC stage,

$$\text{MSE (flpt-fxpt) at a node} << \text{noises due to other sources.} \quad (4\text{-}12)$$

Some sources for these additional degradations are physical noise from channel, physical noise from analog part, and architecture limitations such as circuit approximation for square-root operation. These sources are present for the floating-point system, and cause communication errors even for the floating-point system. In our

approach, we always consider fixed-point implementation an approximation of the floating-point system. This idea is stressed in the formulation in (1-2), where we consider system behavioral performance higher priority, and hardware-cost secondary; therefore, we never want fixed-point non-idealities to be the primary source of the system performance degradation. On the other hand, we do not consider how to design a fixed-point algorithm directly. Examples of this direct approach include designing a finite-state machine and a communication source coder. In these designs, direct abstract (or Boolean) algebra are used even at the algorithm level.

4.3.6.2 Measure the difference at the right places

Example 1 in Section 4.3.6.1 shows that measuring MSE(flpt-fxpt) after a slicer is costly. This is because the probability of noisy events (errors here) is low. Chapter 3 shows that not all signal nodes are suitable for MSE estimation. Furthermore, Chapter 3 links the MSE(flpt-fxpt) after decision-making operators to the MSE(flpt-fxpt) in front of the operator. And the latter MSE become very easy to simulate.

Even more essentially, the perturbation theory of Chapter 2 allows MSE(flpt-fxpt) after arithmetic operators to be written as an explicit function of fixed-point data types in the system, which is further illustrated in Section 4.3.6.3.

4.3.6.3 Perturbation theory provides valuable information

An innovative perturbation theory has been developed in Chapter 2. With the widely used theoretical models of quantization noises, a specification function telling the difference between the floating-point system and fixed-point system can be written into closed form

$$
\begin{aligned}
&|\text{sys.spec.(fxpt)} - \text{sys.spec.(flpt)}| \\
&= \overline{\mathrm{M}}^{\mathrm{T}}\,\bar{\mathrm{u}} + \sum_{i\in\{\text{Data Path}\}} e_i 2^{-2\mathrm{W}_{\mathrm{Fr},i}} + \ldots,
\end{aligned}
\tag{4-13}
$$

where e_i's are constants and $\overline{\mathrm{M}}$ is a constant column vector, the i^{th} entry of vector $\bar{u}$ is given by (1-4). Moreover, this perturbation theory works on general criterions and even non-stationary input, as long as they can be represented as large ensemble averages of functions of the signal outputs.

Following the perturbation theory, we also get

$$
\mathrm{MSE(flpt-fxpt)} = \bar{u}^T \mathrm{B}\bar{u} + \sum_{i\in\{\text{Data Path}\}} \mathrm{C}_i 2^{-2\mathrm{W}_{\mathrm{Fr},i}},
\tag{4-14}
$$

where B is a positive semi-definite matrix, denoted as $\mathrm{B} \succeq 0$, and $\mathrm{C}_i \geq 0$. This has been stated several times in previous text due to its importance.

Now we safely reduce the FFC problem (4-1) to

$$
\begin{aligned}
&\text{minimize} \\
&\quad \text{Quadratic } f_{\mathrm{HW}}(\mathrm{W}_{\mathrm{Fr},1}, \mathrm{W}_{\mathrm{Fr},2}, \ldots; q_1, q_2, \ldots) \\
&\text{subject to} \\
&\quad \bar{u}^T \mathrm{B}_k \bar{u} + \sum_{i\in\{\text{Data Path}\}} \mathrm{C}_{i,k} 2^{-2\mathrm{W}_{\mathrm{Fr},i}} - A_k < 0, \\
&\qquad \text{with } \mathrm{B}_k \succeq 0, \mathrm{C}_{i,k} \geq 0, \text{ and } A_k > 0.
\end{aligned}
\tag{4-15}
$$

Here vector $\bar{u}$ is defined in the same way as before, and A_k is the tolerance of the k^{th} MSE error. The problem is feasible because as all W_{Fr}'s increase, the left sides of the constraint functions asymptotically converge to $-A_k$'s which are always less than 0. Physically, it means the fixed-point system becomes infinite precision. The number of simulations is significantly less than unguided characterization in which the form in (4-14) is not assumed.

4.3.6.4 Use ergodic average rather than large ensemble average

A statistical quantity of an output at time n can be estimated based on large ensemble average. That is, we run the simulation M times with different random seeds and same statistics to generate input data, and use all the M output received at time n to estimate the mean squared error. Though this approach is theoretically accurate to estimate the expectation function, it requires too many repeated simulations. Its simulation time for each optimization step is

Simulation time for M ensembles $=M\times$ (preparation time for each simulation

$+ \ n \times$ time to get one output sample), (4-16)

where the preparation time for each simulation includes time for the simulation environment to compile the system. This compilation alone takes seconds to minutes. Currently, Simulink may take up to minutes to compile a large system, due to its support to process graphical information. Quite often, a random process at the output that is locally stationary; then, we can use ergodic average—an average based on different output samples in one simulation—to estimate the ensemble average [48]. For each Monte Carlo simulation, we have

Simulation time for iteration = preparation time for each simulation +

$(n+M-1) \times$ time to get one output sample, (4-17)

and the samples between time n to $n+M-1$ are used for the estimation. This potentially saves simulation time by additional orders of magnitude over (4-16). Our tool uses this approach to save simulation time.

In summary, in Section 4.3.6, we adopted multiple ways to make the verification of constraint function faster. First, we argued that measuring the performances of

floating-point system and fixed-point system separately is costly, whereas direct measure using MSE(fxpt-flpt) is suggested. Second, simulation error after decision-making operators is too costly to measure, whereas errors after arithmetic operators are faster. Third, based on perturbation theory, we limit the number of Monte Carlo simulations. Finally, we argued large ensemble average is too costly, whereas ergodic average is faster. All the efforts make the verification efficient.

4.3.7 Optimization step

4.3.7.1 Simplifications

The inclusion of binary round-off mode q_i in problem (4-1) makes the problem intractable with the understanding of the problem. First, the hardware cost function in (4-2) may have very complicated relationship with q_i, and it is a combinatorial problem to reveal this relationship. Second, according to (4-14), we need the number of digital simulations quadratically proportional to the word-length group size if q_i are taken into account, rather than linear relationship otherwise. Third, even with all the functions extracted in (4-15), the problem is still combinatorial, and is very challenging to solve. So in the rest of this chapter, it is assumed that roundoff modes are used everywhere.

The preceding simplification implies $\bar{u}_i = 0$ for $i \in \{\text{datapath}\}$. Furthermore, since pre-grouping of operators is done to reduce the number of simulations in Section 4.3.3, many constant inputs are often grouped together. These multiple quantization sources in each of the constant-group altogether behave similar to a random noise with 0-mean and variance proportional to $2^{-2\times WFr}$. Assuming such 0-mean "noises" from

different constant-groups are uncorrelated to each other, the optimization frame work in (4-15) can then include constant-groups similar to data-path quantization noise. That is,

$$\underset{W_{Fr,i}\geq 0}{\text{minimize}} \quad f(W_{Fr}) = \frac{1}{2} W_{Fr}^{T}\, \mathrm{H}\, W_{Fr} + h^{T} \cdot W_{Fr} + h_0$$

subject to

$$\sum_{i\in\{\text{Data Path and constant-groups}\}} \mathrm{C}_{i,k}\, 2^{-2W_{Fr,i}} - A_k < 0, \qquad (4\text{-}18)$$

with

$$\mathrm{C}_{i,k} \geq 0, \text{ and } A_k > 0; i = 1,2,...,m; k = 1,2,....,\mathrm{K}$$

It should be pointed out that all the groupings are done automatically if the structural information of the system is provided, as shown in Section 4.3.3. Therefore, no ambiquity arises in the definition of index *i* in (4-18) as it is just an ordering of all the word-length groups. The dimension of *k*, denoted as K, specifies the number of critical nodes to be examined for MSE specifications, which is usually less than 100.

4.3.7.2 Fractional word-length optimization

The constraints of problem (4-18) represent a joint set of sublevels of exponential-sum convex functions; thus it is a convex set. So if the Hessian H of the quadratic hardware-cost function is positive semi-definite, (4-18) would be a convex optimization problem [63]. Unfortunately, H is usually not positive semi-definite. An easy counter example is a multiplier of input W_1 and W_2, and output $W_1 + W_2$. The hardware-cost is approximately proportional to $W_1 W_2$, or

$$\frac{1}{2}(W_1, W_2)\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} W_1 \\ W_2 \end{pmatrix},$$

which gives a Hessian that has negative determinant and not positive semi-definite. Therefore, in order to proceed using convex optimization techniques, an approximation of the objective using a convex function, such as an affine function, is needed. A local affine approximation of hardware cost function after grouping is used here for this purpose. So, (4-18) is broken into external iterations based on the inner update as convex optimization. Finally, we address the complications caused by integer constraints of W_{Fr}; this can be done by using ceiling(W_{Fr}) as the result, and then individually adjust each component of the result to see whether any of them can be reduced.

Based on the previous arguments, we can the following algorithm

1. Find an initial feasible W_{Fr} by noticing

$$m \cdot \mathrm{C}_{i,k} 2^{-2\mathrm{W}_{\mathrm{Fr},i}} - A_k < 0, i = 1,2,...,m; k = 1,2,....,\mathrm{K}$$

$$\Rightarrow \sum_i \mathrm{C}_{i,k} 2^{-2\mathrm{W}_{\mathrm{Fr},i}} - A_k < 0, k = 1,2,....,\mathrm{K}.$$

So choose

$$\mathrm{W}_{0\mathrm{Fr},i} = \tfrac{1}{2} \max_{\mathrm{k}} [\log_2 (\frac{m \cdot \mathrm{C}_{i,k}}{A_k})], i = 1,2,...,m.$$

2. Obtain the affine approximation of the objective function around W_{0Fr}, which is $\min_{\Delta W_{Fr}} : f(W_{0Fr} + \Delta W_{Fr}) \approx (W_{0Fr}^T \mathrm{H} + h^T)\, \Delta W_{Fr} + f(W_{0Fr})$.

The constraints remains to be exponential sum on the new variable ΔW_{Fr}, with a couple of practical constraints on the size of ΔW_{Fr}:

$$\sum_i \mathrm{C}_{i,k} 2^{-2(\mathrm{W}_{\mathrm{Fr},i} + \Delta \mathrm{W}_{\mathrm{Fr},i})} - A_k < 0, k = 1,2,....,\mathrm{K}; \text{and}$$

$$\min(\mathrm{W}_{\mathrm{maxFr},i} - \mathrm{W}_{0\mathrm{Fr},i}, \mu) > \Delta \mathrm{W}_{\mathrm{Fr},i} > \max(\mathrm{W}_{\mathrm{minFr},i} - \mathrm{W}_{0\mathrm{Fr},i}, -\mu);$$

$$i = 1,...,m.$$

3. Use Mosek function *mskscopt*(.) to solve the convex separable problem [21]. Once the new $W_{0\mathrm{Fr},i} + \Delta W_{0\mathrm{Fr},i}$ is obtained, update it as the new $W_{0\mathrm{Fr},i}$.

4. Repeat step 2 and 3 until the optimal objective value no longer increases by more than 0.1 slice, or until the iteration number exceed 100 (stopping criteria).
5. Use ceiling($W_{0\mathrm{Fr},i}$) as the integer solution. Then try to decrease any of these m values by one bit; choose the one that decreases f the most while still satisfies the constraints.
6. Repeat 5 until any bit-reduction of $W_{0\mathrm{Fr},i}$ makes the constraints infeasible.

Applying this algorithm on several systems, it is found that step size μ being too large (> 5) or too small (< 0.2) results large number of iterations. By choosing μ from 1 to 2, the iteration process in step 4 usually finishes in less than 10 iterations, with each of them done in a few seconds.

On the other hand, we also found that since the number of word length groups is usually less than 50, so that the procedures in [15] or [17] are usually sufficiently fast. That is, the optimization algorithm to solve (4-18) is not essential since the problem has been understood very well using all the results that have been developed so far.

4.3.8 Robust optimization

So far, we have strived to make the FFC fast and, thus, practical. We use functions to represent hardware-cost and constraints, and also limit the optimization space using grouping methods. Meanwhile, we also try to keep the simplifications accurate enough so that our optimization result reflects the true optimal design choice.

However, any simplified modeling inevitably introduces errors. Therefore, it is important to understand these errors and make the optimization robust.

Given the relationship between MSE and W_{Fr} about m experiments are enough to fit the $m\times$K coefficients $C_{i,k}$'s. Either due to estimation error or under-modeling of MSE, such as various simplifications in Section VI, an estimation of $\mathrm{C}_{i,k}$ is given by a range $[\mathrm{C}_{i,k,\mathrm{lower}}, \mathrm{C}_{i,k,\mathrm{upper}}] \subset \mathrm{R}_{+}$, and the interval size depends on the confidence of estimation of $\mathrm{C}_{i,k}$. The design of W_{Fr} should satisfy (4-18) for any $\mathrm{C}_{i,k} \in [\mathrm{C}_{i,k,\mathrm{lower}}, \mathrm{C}_{i,k,\mathrm{upper}}]$. Thus, it quickly reduces to the following robust version

$$\begin{aligned}
&\text{minimize} \\
&\quad f(W_{Fr}) = \frac{1}{2} W_{Fr}^{T}\, \mathrm{H}\, W_{Fr} + h \cdot W_{Fr} + h_0 \\
&\text{subject to} \\
&\quad \sum_{i=1,2,\ldots \mathrm{m}} \mathrm{C}_{i,k,upper}\, 2^{-2\mathrm{W}_{\mathrm{Fr},i}} - A_k < 0, \\
&\quad \text{with } \mathrm{C}_{i,k,upper} \geq 0, \text{ and } A_k > 0; k = 1,2,\ldots,\mathrm{K}.
\end{aligned} \tag{4-19}$$

The algorithm remains almost the same as in (4-18) except for the replacements of $\mathrm{C}_{i,k}$'s. So the procedure in Section 4.3.7 still applies. Because of the exponential relationship, the optimal wordlength design increases only about $1/2\ \log_2(\mathrm{C}_{i,k,\mathrm{upper}}/\ \mathrm{C}_{i,k})$, so that word length optimization is quite insensitive to MSE estimation errors.

4.3.9 User interface

In addition to the Range Detector and Spec Marker blocks that are implemented as Simulink library blocks, the rest of the tool is fully implemented as Matlab functions that realize each proceeding subsection of Section 3 in a pre-programmed flow. The user places Spec Marker(s) into the high-precision system and specifies the performance levels, and then the tool will execute all the functions in Fig. 4-6 and output the optimal fixed-point design.

Occasionally, MSE becomes a poor way to approximate a specification. For example, the regulation on off-band transmitting power of a radio often translates to the requirement that the digital filter in the transmitter needs to satisfy a specification on the maximum side-lobes of its frequency response. Therefore, there is an option of Spec Marker to choose as a specification function using customized Matlab functions. In this case, convex optimization method as described in Section 3.7, often can not be used.

For systems with non-stationary inputs, the program monitors the MSE at each Spec Marker at different times. There will be a specification level corresponding to each time. So, (4-19) retains its format, but with more constraints.

Finally, for comparison purposes the tool can also easily provide the capability of using a pure-simulation-based approach, similar to those in [15]. That is, no analytical hardware-cost function and constraint functions are drawn. However, it still takes advantage of the discussions in 3.6.1 and 3.6.4, as well as 4.3.2 and grouping method in 4.3.3.

4.4 Applications

4.4.1 Simple binary phase shift keying (BPSK) transceiver

The first system that is automatically converted is a BPSK transceiver system mentioned earlier in Chapter 3. By applying robust optimization, one obtains the final conversion result in about 5 minutes, with only 265 FPGA slices, as shown in Fig. 4-12.

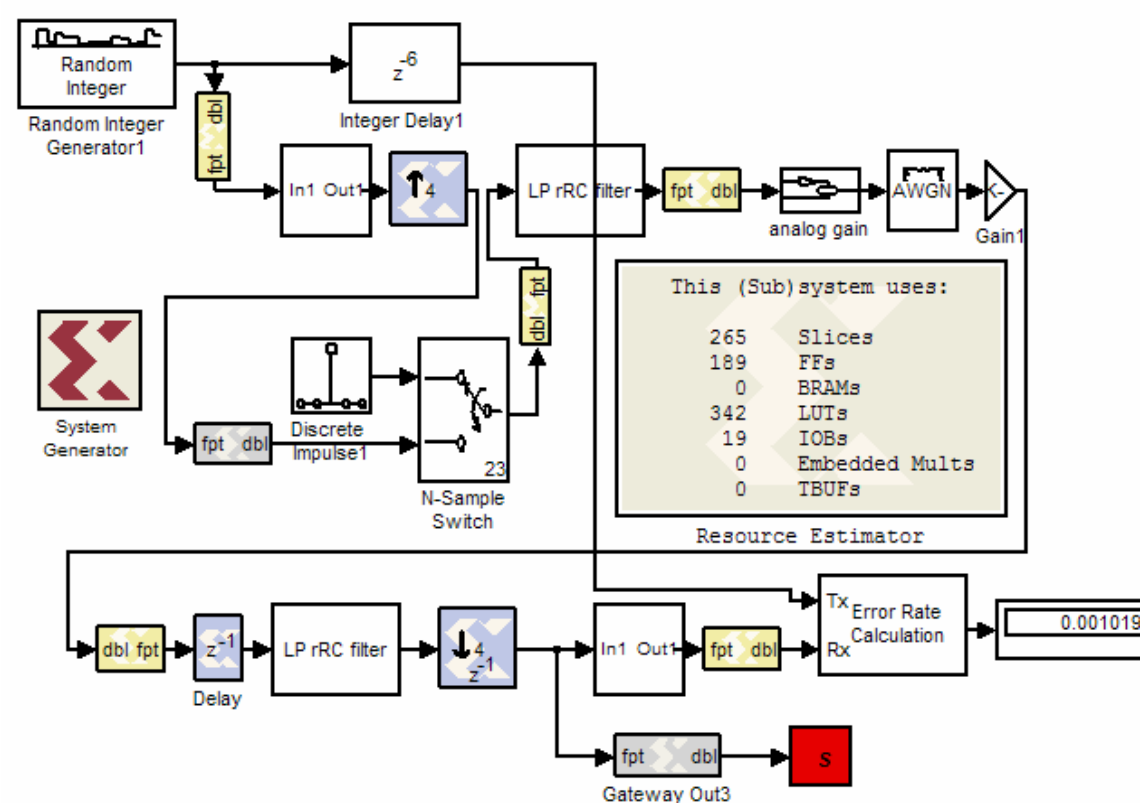
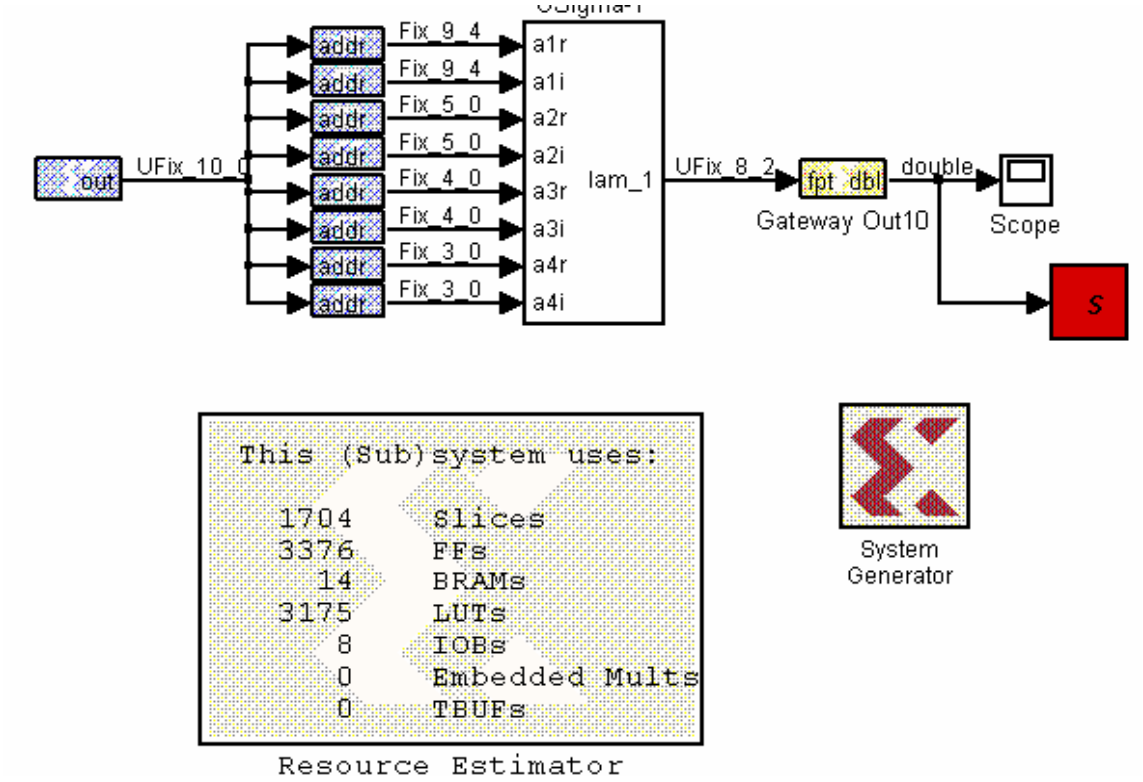


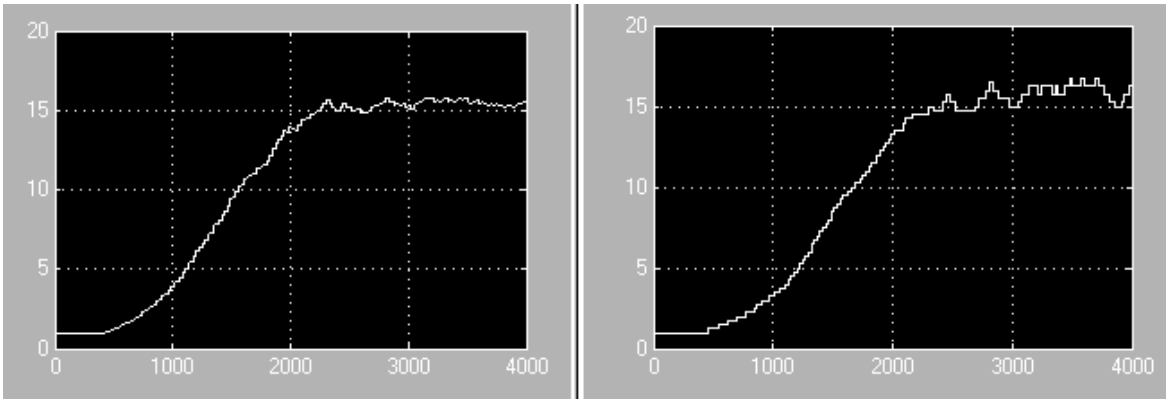
Fig. 4-12 BPSK communication system in System Generator.

4.4.2 U-Sigma block of singular value decomposition (SVD) system

The second system is a SVD-USigma system used in multi-carrier multi-antenna system. This system is nonlinear with feedbacks. Fig. 4-13 shows the system. It takes 40 minutes to FFC, and most time is spent on running the *m* simulations in Section VI. The converted system takes 1704 FPGA slices, which is about 5 times smaller than previous known result of about 8800 slices obtained by floating-point designer with hand-tuning [70]. Fig. 4-13 (b) and (c) show that the convergence behavior of the fixed-point system does not change much from the one in floating-point system.



(a)



(b) (c)

Fig. 4-13 SVD algorithms
(a) SVD U-sigma block with 1704 slices. (b) Eigen-value tracking versus time of the floating-point system to be converted. (c) Eigen-value tracking versus time of the fxpt system.

4.4.3 Ultra-wide band (UWB) baseband implementation

The third system converted is an ultra-wideband system designed by UWB subgroup at Berkeley Wireless Research Center [71]. This is a complicated system contains about 2000 arithmetic and logic units. The system has been hand-tuned to take 6695 slices in fixed-point implementation. The FFC tool works on this partly optimized system and produces a version of 4610 slices. Not much improvements can be done over the original system since on average only 2~3 slices per unit are consumed.

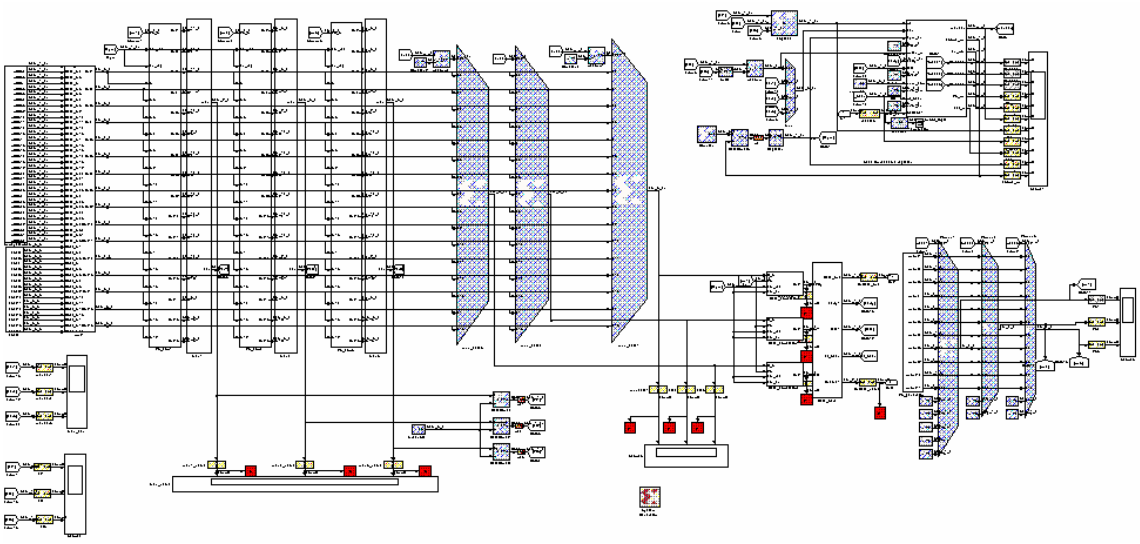


Fig. 4-14 Ultra-wide-band (UWB) baseband
This UWB system contains 16-tap matched filter and 7-entry PN sequence. Right bottom is PN sequence generator; upper right shows control units. 10 specification markers were inserted as shown in red give the constraints.

The conversion takes about 2 hours. Some final tests on the converted fixed-point system show its matching performance to the original system.

In Table 4-3 the results for the three systems are summarized.

Systems:	**SVD-Usigma**	**UWB**	**BPSK Transceiver**
Direct system specifications	Eigenvalue convergence	Detection error and BER	BER
FPGA slices of hand-tuned system	8858	6695	N/A*
Number of MSE Spec. Markers	1	10	1
MSE spec levels	0.1	4×ones(1,10)	0.0005
Grouping rules	[1.1 2.1 3 4 5 6 7]	[1.1 2.1 3 4 5 6 7]	[1 2.1]
FPGA slices of the system after FFC	1704	4610	265
FFC duration	40 minutes	2 hrs	5 minutes

Table 4-3 Summary of the three systems that are FFC'ed

*The fixed-point data types in the system are not hand-tuned before automated FFC; so, no information available.

4.5 Comparison with existing techniques

This section compares the FFC tool with those techniques that are based on pure-simulation. The FFC techniques are superior to other existing tools because of its more general applicability and due to the improved optimality obtained [17].

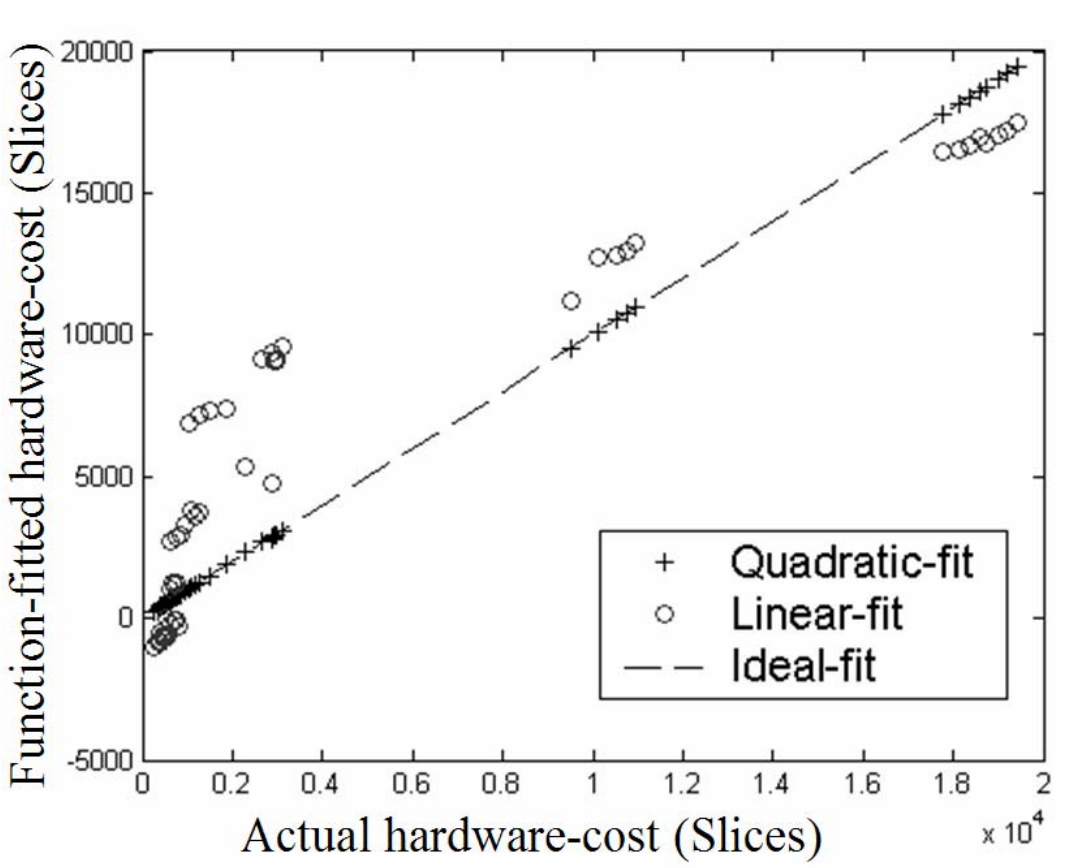


Fig. 4-15 Hardware-cost using various models for the estimate.
The curve shows that a quadratic model is adequate for the BPSK transceiver in Section 4.4.1 (a) using quadratic-fit as proposed in this section, or (b) no 2nd-order term H is modeled—so called linear-fit.

First of all, our tool uses accurate resource estimation instead of handwritten linear hardware model that is used in all previous techniques. Fig. 4-15 shows that a linear model can differs greatly from actual the hardware-cost, while a quadratic model is quite accurate. Furthermore, since existing techniques that do not start from a structural description often do not model the hardware required for blocks such as a MUX, they suffer significant modeling errors.

Second, using simulation based FFC, but without adapting our techniques discussed in Section 4.3.6, the conversion becomes often intolerably long. Even using the

ergodic approach described in Section 4.3.6.4 does not make it acceptable. For example, for the relatively simple BPSK system in Section 4 that has BER=0.00078 in the floating-point design, and a target BER=0.00085 for the fixed-point, each BER simulation take hours to finish. The iterations needed for optimization would further prolong the conversion time. In fact, without using the methods in Section 4.3.6.1, 4.3.6.2 and 4.3.6.3, to finish the conversion listed in Table 4-3 for the BPSK system would take at least 7 days, which is 10^3-10^4 times slower than our proposed method.

Even by further adopting part of our techniques in Section 4.3.6 to avoid BER type of simulation—using MSE(flpt-fxpt) as a direct measure of fix-point system performance, existing techniques are still at least 5 to 6 times longer. This is because the number of simulations in any unguided optimization is at least 5 or 6 times the number of independent wordlength groups, denoted by *m* [15], while the FFC approach described here only requires approximately *m* simulations to characterize all the specification functions.

Finally, since the tool first completely characterizes the hardware functions and specifications, optimizations against different specification levels just repeat the optimization procedure, which is usually performed in seconds. Therefore, it becomes straightforward to produce curves such as shown in Fig. 4-16 in which the hardware cost versus specification can be clearly presented.

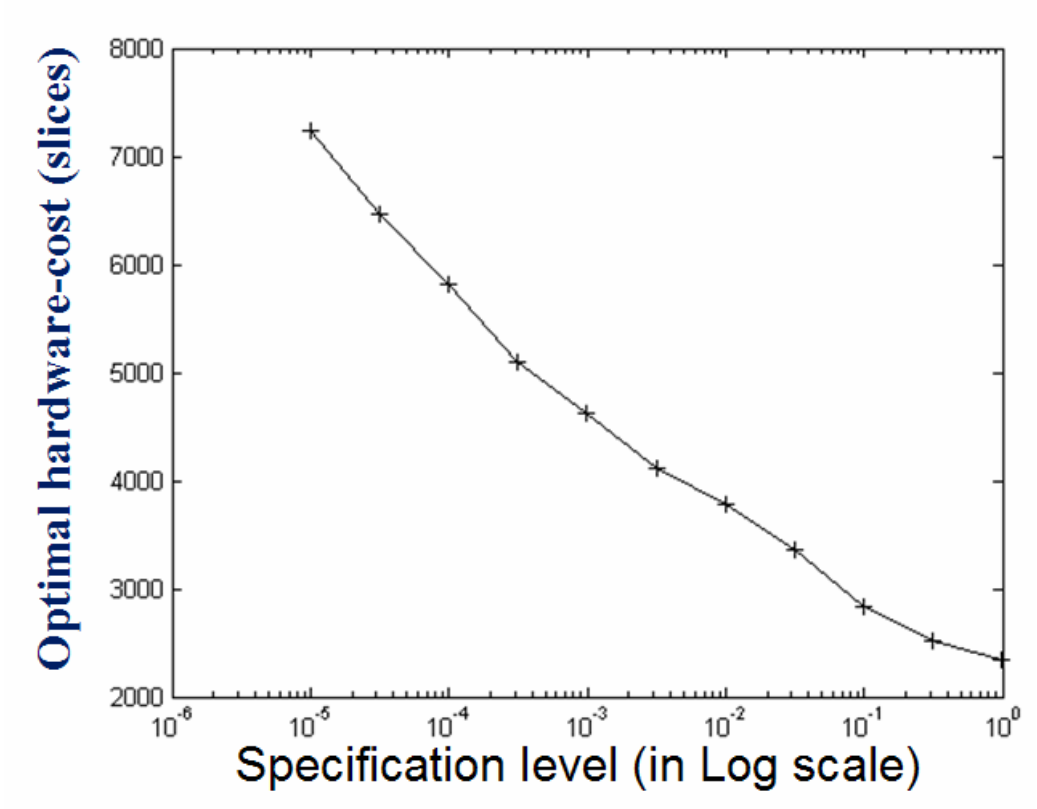


Fig. 4-16 Hardware-cost and specification trade-off for the SVD U-sigma. The system is described in Section 4.4.

4.6 Summary

A comprehensive automated approach for floating-point to fixed-point conversion (FFC) has been presented. With tens of thousands of lines of Matlab codes as the underlying engine, an implementation in a self contained tool in the Xilinx System Generator and Mathworks Simulink environment has been developed and the application of this tool to several real designs has been presented. Hardware-cost information has been modeled and a perturbation approach to determining the specification sensitivity has been implemented and found to give orders of magnitude in speedup over our simulation based techniques. The tool uses FPGA's as the hardware model merely to demonstrate

the feasibility of our methodology, but a similar approach would apply to ASIC designs as well. The essential step is to obtain an accurate model of the hardware-costs.

Our goal is not to find the last few percent improvement of objective function in (1-2), but is to quickly and intuitively determine the data type. Our efforts on understandings of specifications and hardware cost functions shows one promising way to attack this problem, which we hope to inspire similar approach on other design and verification problems, such as architecture and algorithm design. On the other hand, it is important to understand the "loss of optimality" by doing robust optimization.

A few open questions are explained. More strict treatments on the integer side can be valuable. Moreover, the combinatorial optimization problem associated with quantization-modes, which may also be chosen from truncation-mode and round-off modes, also needs further studies. They are emphasized again in Chapter 6.

Chapter 5
FPGA Hardware Resource Estimation

Section 4.3.5 of Chapter 4 mentions the usage of a hardware-cost estimation tool to solve the FFC optimization problem (1-2). This tool is the topic of this Chapter.

When a system is mapped to Xilinx FPGA chip, the consumed slices, LUTs, Block RAMs, Virtex-II embedded multipliers (when applicable), flip-flops, tri-state buffers and IOB counts are referred as its hardware resource information. Existing hardware resource estimations suffer either inaccuracy, slowness or high complexity. A strategy of fast hardware resource estimation in Xilinx System Generator environment is proposed in this Chapter, and is implemented purely in Matlab and Simulink evnironment. Only the pre-netlisting Simulink compilation is required to prepare for the estimation, and each estimation typically takes only seconds or a few minutes. In our verifications, every aspect of the resource estimation agrees within 10% from the map report. The resource information of each System Generator block is characterized into a Matlab function, based on the understanding of IP-core design, as well as the considerations of the trimming effects from the subsequent synthesis tool and mapper. Finally, it is explained how these functions get integrated together with Similink to form

a user-friendly and automated infrastructure. This estimator has been included in System Generator since version 3.1.

5.1 Introduction

Field-programmable gate arrays (FPGAs) have become increasingly important in implementing digital signal processing (DSP) systems such as digital communications, and multimedia. On an FPGA chip, the following basic resources are normally available to realize a system: slices which contains look-up tables (LUTs) and flip-flops (FFs), block memories (BRAM), tri-state buffers (TBUFs), In and Out bonds (IOBs), and dedicated 18x18 multiplies (currently available for Virtex-II chips). A top-down design methodology has been recognized to dramatically speed-up the design process without substantially compromising the performance of the hardware implementation. In fact, together with high level hardware-cost estimation tools, the top-down design flow opens the possibility of global optimization at the system level, which often leads to even more hardware-efficient designs. Our FFC strategy is one excellent example in this trend. FPGA resource usage, as one type of hardware-cost (others are like critical path delay or active-power consumption), is particularly important when the goal is to find the best behavioral system performance (such as signal-to-noise ratio) while fitting into a specific chip, or when the goal is to find the lowest resource meeting a performance specification. As another example, it is sometimes necessary to partition a large system to multiple FPGA chips, which requires estimations of the resources for sub-systems. An optimization process further requires numerous iterations of resource estimations. This motivates a fast resource estimation tool at high level.

Among many available CAD tools, System Generator1 [60][91] for DSP is a successful example for modeling and designing Xilinx FPGA-based signal processing systems in Simulink and Matlab2 [59]. Section 5.2 starts by providing some necessary information of System Generator environment. Then it proceeds to introduce some existing or possible resource estimation methodologies, followed by a proposal of our method. In our method, only the Simulink compilation stage is needed for each estimate. Our method differs from existing ones in that it requires complete understanding of how the IP-cores are designed. Furthermore, it predicts those logics that are trimmed away by synthesis tools and mappers. What is also covered is the topic of how the methodology can be integrated with Simulink GUI and Matlab command line to form a user-friendly infrastructure, which enables estimation for selected parts of a system. Section 5.3 validates the fully implemented resource estimator by studying the estimation results of a couple DSP designs. A few possible future developments are summarized in Section 5.4.

5.2 Resource estimation in System Generator

Our resource estimation is implemented in the System Generator design environment that is described in Section 2.1. Though the methodology is portable to other platforms, the architectural description of a DSP system, as System Generator does naturally, is indeed necessary for accurate estimation.

1 System Generator is a registered trademark of Xilinx Inc.

2 Simulink and Matlab are registered trademarks of Mathworks Inc.

5.2.1 System Generator design environment

At simulation level, System Generator for DSP maintains an abstraction level very much in keeping with the traditional Simulink blocksets, but at the same time automatically translates designs into hardware implementation [60][91]. The system model and hardware implementation are bit-true and cycle-true. Besides some synthesized blocks, the implementation is also made efficient through the instantiation of high-speed and area-efficient intellectual property (IP) cores that provide a range of functionality from arithmetic operations to complex DSP functions. In System Generator, the capabilities of IP cores have been extended transparently and automatically to fit gracefully into a system level framework. For example, although the underlying IP cores operate on unsigned integers, System Generator, through the so called wrapper logics, allows signed and unsigned fixed point numbers to be used, including saturation arithmetic and rounding. While providing functional abstraction of IP cores, the System Generator blocks also provide the FPGA-literate designer access to key features in the underlying silicon, which is often necessary to achieve the highest performance implementation in an area-efficient manner. For example, the System Generator multiplier block has an option to target embedded high-speed 18x18 multipliers in the Virtex-II family of FPGAs.

5.2.2 Resource estimation methodologies

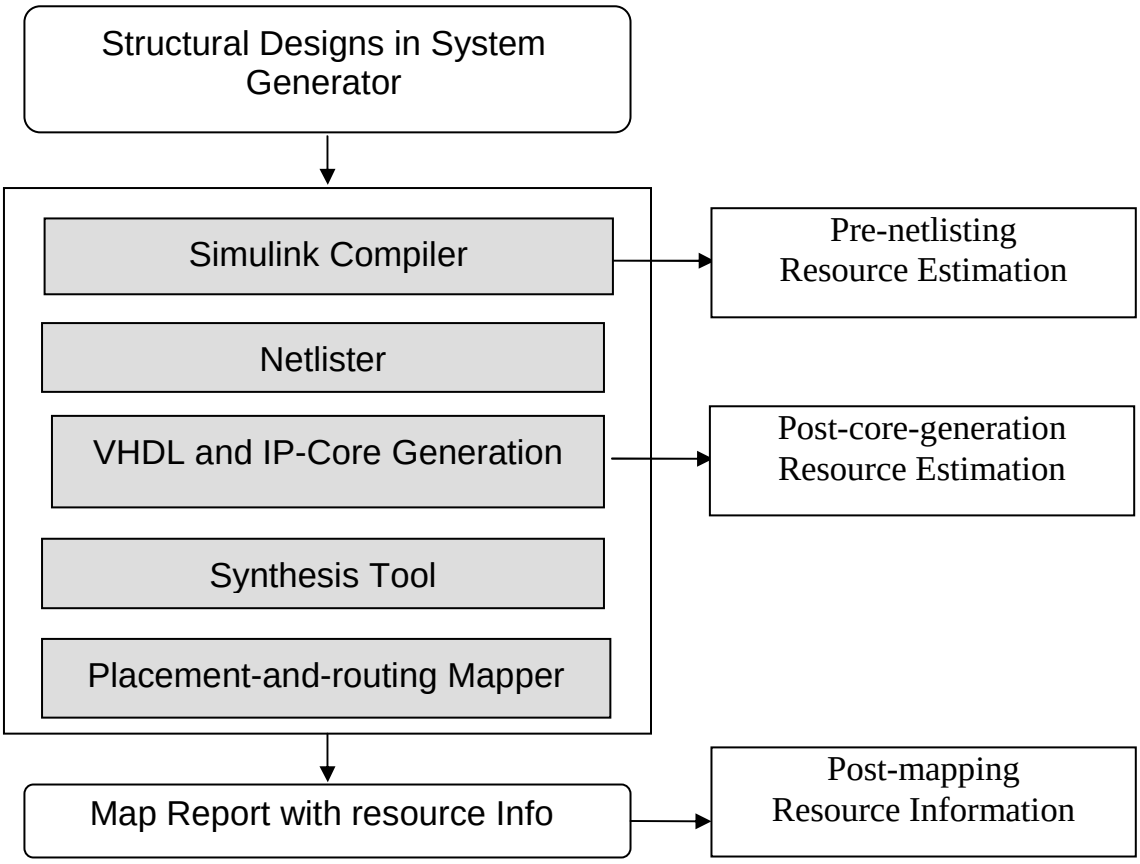


Fig. 5-1 Resource estimation methods

The hardware resource information can be exactly retrieved from the post-placement-and-routing map-report. As shown in Fig. 5-1, the hardware resource information of a model can only become available after Simulink compilation, Netlister, IP-Core generation, synthesis and mapping stages. The whole process can take minutes or even hours, depending on the size of the system.

Fig. 5-1 also shows an estimator can simply sum the resource information available at each core after core-generation. However, the method becomes slow—often

only a couple times faster than to map-report method. Also, all the synthesized logics other than the IP cores are not considered, making the result inaccurate as well.

Several possible ways can be done for the pre-netlisting resource estimation. One of them is to build a database, listing the resources given all the possible combination of a particular block. Each entry of in the database is obtained by a complete design experiment. However, an initial implementation3 of this methodology shows that many blocks involve too many combinations that the tests may easily takes months or more of computing time to complete. Even when this done, the database for some blocks may consume hundreds of Megbytes and is no longer practical. Therefore, only for a few blocks that have less than tens of parameter combinations, this method is useful (and indeed used sparsely in our current estimation tool).

An alternation of this aforementioned method is to build database for IP-cores only, while estimating all other synthesized logics by simple functions [90]. Yet, this method still suffers from the complexity difficulty as complicated cores can have too many parameter combinations. One way to alleviate the preceding complexity difficulties is by ignoring block parameters. As the tradeoff, the estimation become less accurate (e.g. up to 30% or more [89]).

A common problem associated with all these third-part estimation methods is the lack of understanding of IP-core design. The present methodology, however, is based on complete reverse-engineering the IP-core designs. Moreover, trimming effects caused by

3 This was tried by C. Shi under Prof. Robert W. Brodersen's advice at University of California, Berkeley.

synthesis tool and mappers are understood by doing experiments and by collaborative working with the designers of these tools.

5.2.3 Resource estimation at the system level

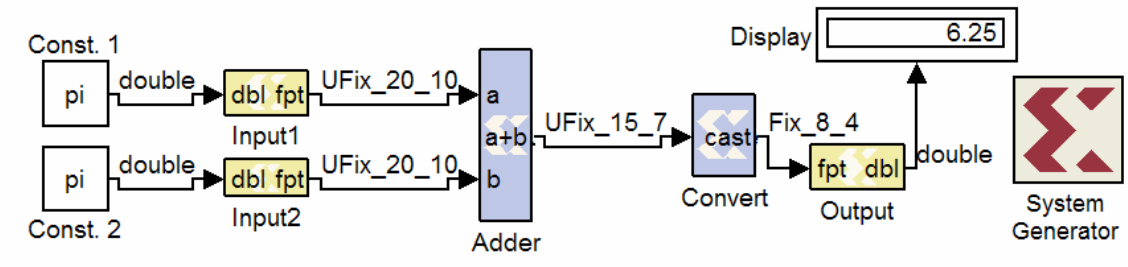


Fig. 5-2 A simple System Generator design with block output data-type displayed. Here UFix_20_10 means an unsigned signal with 20 bits in total and 10 bits of them are fractional. UFix_15_7 is defined accordingly. Fix_8_4 is a 2's-complement signed signal with 8 bits in total and 4 of them are fractional. This design shows the trimming effects.

Fig. 5-2 shows a simple design in System Generator composed by some basic blocks. To estimate the resources of one type of blocks, such as adders, a Matlab function in the following framework is written and called:

```
function tarea=get_BlockType_area(system)
% find out all the blocks of a particular masktype
r = find_system(system, 'masktype',...)
% Initial a Simulink compilation if not yet
% Get area for each of these blocks using a for-loop
for i=1:length(r),
    % Get the data-type of the block inputs and outputs
% as well as all other block parameters
    get_param(r{i},...);
% Use a dedicated function to get the resource
[area(i,:), input_type]=BlockType_area(...);
% update the resource info for block r{i}
end
% End the Simulink compilation if it has not been done
% Get the total area of the particular type
tarea=sum(area,1);
```

The functions find_system(.) and get_param(.) are Simulink model construction commands [3][59]. They allow the control of Simulink system using Matlab scripts, which makes the design automation possible. Simulink compilation is needed to retrieve signal (port) data-types and to compute those formula-based or hierarchically defined mask parameters.

In Fig. 5-2, a full-precision adder would grow its input data-type to UFix_21_10 (with one more integer bit than the input to accommodate overflow). But as the user defines that only 15 bits of the adder output are needed, some of the adder logics will be trimmed away by the synthesis tool or mapper. This trimming effect is referred as block-level trimming and is further studied in Section 5.2.4. Furthermore, if the Convert block uses truncation mode at the LSB side and wrap-around mode on the MSB side, the synthesis tool or the mapper will directly propagate its output wordlengths backward to its input, making the true output of the adder block as UFix_8_4 instead of UFix_15_7. As a result, more logics will be trimmed away from the adder by the synthesis tool. This trimming mechanism is referred as system-level. In the current version of the tool, the system-level (or global) trimming effects are not implemented.

5.2.4 Resource estimation at the block-level

A Matlab function is written for each type of block, initiated as

```
Function [area,input_type] = BlockType_area(block_params).
```

Normally, each BlockType_area(.) function is written in the following steps,

1 case-divide the following steps according to block parameters;
2 understand the data-types for output wrapper and all the sub-cores;

3 calculate the resource for the wrapper and each sub-core with trimming effect;

4 sum different resources together;

5 get input data-types after backward trimming effects .

Whenever applicable, vector signal processing is used to speed-up the calculation. A great amount of efforts are paid to take care the aforementioned block-wise trimming effects. The last step of this procedure prepares the inclusion of the system-level (or global) trimming effects in the future. Extensive map-report tests are done to make sure the estimation function gives either less than a couple units or less than 5% relative error. The following two subsections illustrate these steps using two examples.

5.2.4.1 Resource estimation for an Adder/subtractor block

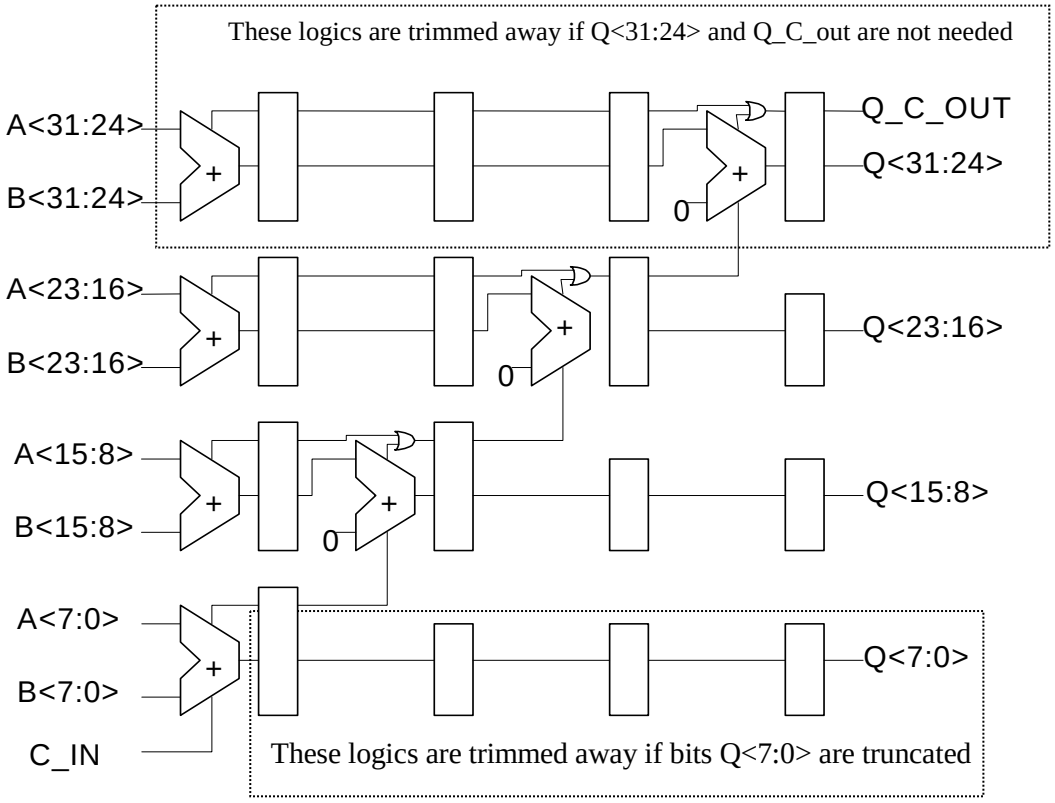


Fig. 5-3 A possible realization of a 32-bit adder.

The blank boxes denote pipeline flip-flops. The logics in the dashed box would be trimmed away if output Q<7:0> is truncated at the output.

The adder/subtractor block, or in short add/sub, is used as the first example. In the get_addsub_area(.) function, the following function is called

```
function [area, input_type]= addsub_area(at_a, wa,wfa, at_b, wb, wfb, at_o,
wo, wfo, prec, q, o, latency, use_core, use_rpm, pipeline, mode)
% This particular function contains about 200 lines of
% Matlab code that are not shown here.
```

Here *at_a* is arithmetic type of input *a*; *wa* is the total wordlength of input a; *wfa* is the fractional wordlength of input *a*; *at_b*, *wb*, and *wfb* are similarly defined for input b; *at_o*, *wo*, and *wfo* are similarly defined for the output; *q* and *o* are the output quantization and overflow modes; *latency* is the extra latency at the output; *use_core* indicates whether the add/sub block is generated using ip-core or freshly synthesized; *use_rpm* indicates whether the block use RPM feature; *pipeline* indicates whether the block is pipelined internally to the greatest extend; *mode* indicates the block is an adder, subtractor or add/sub combination. All these block parameters are obtained in get_addsub_area(.) function before it calls addsub_area(.).

Experiments show that using RPM results slightly higher resources, but usually negligible. The three modes—subtractor, add/sub or adder—usually take similar resources; the difference is negligible except that when two unsigned numbers add each other, some logic LUTs will be replaced by route-through LUTs.

There are three main cases for the add/sub block, or abbreviated as add/sub. They will be described one by one in this section, followed by some major observations that need to be pointed out.

The first case is the pipelined add/sub using IP-core. Then, the adder is divided into pipelined sessions depending on the latency chosen. The main challenge here is to figure out the implementation style of the core, based on different choice of latency and output width. This has been done by reverse-engineering the way the add/sub core is designed.

The second case is the none-pipelined add/sub using IP-core. Here, the challenge is to discern the logic LUTs with the route-though LUTs. One level of latency is absorbed by the slices containing LUTs, and the rest latencies are handled by the SRL17 implementations. All LUTs on the most-significant-bit side are trimmed away when these bits are not needed, whereas only the flip flops and shift-register LUTs are removed in the least significant parts, as shown in Fig. 3.

The third case is the fully synthesized add/sub. This is similar to the none-pipelined core add/sub, with important difference. First, all the latencies are handled by the SRL17s. Secondly, some of the least-significant-bit logics can be trimmed away only one input has none-trivial bits there.

In general, the addsub_area function is designed to understand how trimming affects the add/sub resources, and includes the additional resources needed for the synthesizable wrapper.

Finally, the possible trimming on input bits is described in addsub_area(.) function, which is prepared for handling global trimming effect in the future, when the compiler is able to handle backward data-type propagation.

5.2.4.2 Resource estimation of 18x18 Embedded Multipliers

As another example of writing block level resource estimation function, let's look at the usage of 18x18 embedded multipliers that are currently available in Virtex-II family. When the target multiplier size is less than 18x18, it can be fit into one embedded multiplier. Otherwise, multiple embedded multipliers are needed, each of which generates a partial product, followed by adder logics to sum all the partial products together to form the final output.

By understanding the way the embedded multiplier is used, the usage of these embedded primitives can be written as a simple function of the parameters of the target multiplier, that is,

$$\text{Number of } 18\times18 \text{ Embedded Mults in a Multiplier} = \text{Ceil}\left((N_A + \text{Unsigned}_A - 1)/17\right) \times \text{Ceil}\left((N_B + \text{Unsigned}_B - 1)/17\right) \quad (5\text{-}1)$$

where subscripts A and B denote the two inputs of the multiplier, N_A denote the number of bits of input A, Unsigned_A is either 1 or 0 representing signal A is unsigned or signed, similarly for B. and ceil(.) is the ceiling function as defined in Matlab. The total number of 18x18 multiplier primitives used in a model is simply the sum of the numbers for each parallel multiplier.

5.2.5 User interface and design automation

Fig. 4-1 shows the user interface of the initial resource estimation tool (which has subsequently been slightly changed in the commercially available one [60]). Every Xilinx block that requires FPGA resources has a mask parameter that stores a vector containing

its resource requirements. The Resource Estimator block can invoke underlying functions to populate these vectors (e.g. after parameters or data types have been changed), or aggregate previously computed values that have been stored in the vectors. Each block has a checkbox control "Use Area Above for Estimation" that short-circuits invocation of the estimator function and uses the estimates stored in the vector instead.

In Fig. 4-1, by activating the resource estimator block, a Simulink compilation is initiated. When the complication is done, all the underlying resource estimation functions get_BlockType_area(.)'s can be called, which in turn calls those core functions BlockType_area(.)'s to get the estimated area. The results are then displayed on the estimator box. Furthermore, the resource vector of each individual block is also updated and displayed.

5.3 Experimental results

The Matlab function BlockType_area(.) for each block type has been tested extensively, sometimes exhaustively, against the map-report result under various block configurations. In this section, the complete resource estimation tool is further tested against a number of complicated DSP designs, two of which are reported here. One design is an additive-white-Gaussian-noise (AWGN) simulator that generates pseudo-random AWGN noise. The other one is a stage-based Cordinate rotation digital computer (CORDIC) system. Table 5-1 shows the results on the 7 aspects of the FPGA resources, as well as the time required to get the estimations.

	Slices	FFs	BRAMs	LUTs	IOBs	18x18 Mults	TBUFs	Time (min)
AWGN (Prev. tool)	1571	760	0	1595	27	1	0	15
AWGN (New tool)	1606	760	0	1612	27	1	0	.5

11-stages Cordic (Prev. tool)	453	952	1	773	101	0	0	10
Cordic (New tool)	471	982	1	794	101	0	0	.3

Table 5-1 Comparison of estimation tools.
The proposed resource estimation tool (new tool) with map-report (previous tool) on a couple designs. AWGN is an additive-white-Gaussian-noise simulator.

The results in Table 5-1 are representative to many other tests. Every aspect of the resources obtained from the proposed resource estimation tool agrees with the map-report within 10% (usually within 5%). Yet, the estimation time speeds up by 1-2 orders of magnitude comparing with map-report method. Again, this acceleration is benefited by the elimination of those time-consuming netlisting, synthesis, and placement-and-routing stages in order to get a map-report.

On the other hand, as long as a System Generator design can be compiled by the Simulink compiler, the proposed resource estimation tool is able to estimate. In this way, resource estimation can be obtained for pre-mature designs that cannot even pass the rest of the design flow to reach the map-report stage.

5.4 Acknowledgements

The work described in this Chapter was mostly done at Xilinx Inc. during my summer internship. All the scholars I met there, particularly those who I mentioned in the Acknowledgement at the beginning of this thesis, should basically be the co-authors of this chapter. In addition, I want to thank his Ph.D. advisor, Prof. Robert W. Brodersen, for his constant support and advice on this part of my project. He also wants to thank the rest of the System Generator group, IP-core group, XST synthesis group and Dr. David Square at Xilinx for their useful suggestions and supports.

5.5 Summary

A novel pre-netlisting FPGA resource estimation tool in System Generator has been developed. The estimation is accurate because the architectural information of a design is available in System Generator; it is also because IP-cores designs and trimming effects are understood. Furthermore, total automation of the tool is realized in Matlab functions by taking advantage of the Simulink model construction commands. Verifications on real designs show excellent agreement with map-report. This resource estimation tool is one essential part of our accurate and fast FFC tool as described in Chapter 4.

Further developments can be done in several possible areas. First, a dominant portion of the estimation time is spent on Simulink compilation to obtain the data-types at signal nodes; so, more efficient compiler would speed-up the estimation tool. Secondly, the aforementioned global trimming effects could be important in some designs, which can be taken care of by having a smarter Simulink compiler that can propagate signal date-types both forward and backward. Thirdly, similar estimation tools might be developed for power-consumption and signal path delays.

Chapter 6
Possible Extensions and Future topics

A study on Automated FFC of a discrete-time digital system is proposed in previous chapters. This is derived through the characterizations of signals, design blocks, hardware-cost, and specification functions. A perturbation theory is developed to understand the statistical quantization effects of a fixed-point system, with or without decision-making errors. Based on these results, an automated FFC tool for general communication system is demonstrated. This tool is orders of magnitude faster than existing techniques since it utilizes the analytical results of the perturbation theory as well as many other considerations. For the system chosen, BPSK, UWB and SVD u-sigma, the proposed FFC showed its general applicability and advantages. This chapter emphasizes some related problems that have not been covered in details in this work, which hopefully inspires more researchers and engineers to work on this problem.

First, despite the large efforts in Chapter 3, the quantization effects of decision-making errors are still far from being solved completely. Now, we have a good understanding of the probability for this kind of error to happen. Yet how these errors, with large magnitude, propagate in a non-linear system with feedbacks is difficult to analyze, as being fully demonstrated in the absolute-function example and signed-

algorithm example of Chapter 3. Our only progress made on this topic is the categorization of decision errors to soft ones and hard ones, based on some non-trivial insights. This is certainly one topic to continue for both its theoretical and practical values.

Secondly, similar to the decision errors are the overflow noise, which is probably more difficult to analyze since, unlike decision errors, the magnitude of these errors are also random (more strictly speaking, they are deterministically related to the IP signals). It seems that the only reliable analysis is to trace how the probability density function (PDF) of each signal is going to vary with the presence of overflow events. It has to be emphasized that in non-linear systems with feedback loops the change of a PDF at one node at one time will affects the change of this node and other nodes in the future. I have some detailed analysis on this approach, but so far I don't think a good solution has been nearly found.

Thirdly, power and clock speed, in addition to area, are two other fundamental hardware cost functions that designers want to optimize. It is therefore good if estimation tools and analytical models for these two costs can be done in a similar way to FPGA resources as what we have done. Furthermore, to apply the FFC methodology to ASIC design flows, hardware estimators and the analytical models of the corresponding hardware costs also need to be done.

Fourthly, though our analyses in Chapter 2, 3, 5 and most part of 4 include quantization mode, our final FFC tool does not include them as a design variable as this requires much more simulations to model and raises a combinatorial optimization

problem, as discussed in Chapter 2 and 4. One way to alleviate this problem is just by pre-determine the quantization modes using some procedures. This is certainly an open question.

Finally, it is conceived that hardware emulation engines BEE can help to alleviate the difficulty imposed by long simulation times. However this requires fast mapping from Simulink to FPGA hardware, at least for incremental changes. It also requires the FPGA system large enough to contain a pseudo floating-point system (that is, a fixed-point system with very large word lengths). Currently, these are not yet there.

In the end, I and my research advisor, Bob, wish the methodology can be adopted by anybody else. Our website devoted for our FFC tool [88] is for this purpose. I'd like to provide my personal help whenever you need it.

Appendix A. Perturbation Theory on LTI Systems

A.1 Derivation of (2-33) from (2-37)

First, we explain the notations in (2-37).

Bold letters represent matrices, and barred letters represent vectors;

$(.)^H$ represents the Hermitian transform—the transpose of the conjugate—of a vector or matrix;

The $e^{j\omega}$'s in the parentheses indicate that each term is a function of frequency;

Column vectors $\bar{x}$ and $\bar{y}$ represent the multi-dimensional input and output, respectively, whereas column vector $\bar{q}$ represents the multi-dimensional data-path quantization noise input;

$\Delta\bar{y}$ is the difference vector between the outputs of FP and IP systems, that is, $\bar{y}_{\mathrm{FP}}$-$\bar{y}_{\mathrm{IP}}$;

$\mathbf{R}_{\Delta\bar{y}\Delta\bar{y}}(e^{j\omega})$ is called the power density spectrum matrix of $\Delta\bar{y}$ at a given frequency, that is, its *m*-th row and *n*-th column, $\mathrm{R}_{\Delta\bar{y}_m\Delta\bar{y}_n}(e^{j\omega})$, is

defined to be the cross power density spectrum between stationary random process $\Delta\bar{y}_m$ and $\Delta\bar{y}_n$;

$\mathbf{R}_{\bar{x}\bar{x}}(e^{j\omega})$ and $\mathbf{R}_{\bar{q}\bar{q}}(e^{j\omega})$ are defined similarly to $\mathbf{R}_{\Delta\bar{y}\Delta\bar{y}}(e^{j\omega})$, accordingly;

$\mathbf{H}_{\bar{y}\bar{q}}(e^{j\omega})$ represents the frequency response matrix from $\bar{q}$ to $\bar{y}$, that is, its *m*-th row and *n*-th column, denoted as $\mathrm{H}_{\bar{y}_m\bar{q}_n}(e^{j\omega})$, is defined to be the transfer function from $\bar{q}_n$ to $\bar{y}_m$ in the frequency domain—the Fourier transform of the corresponding impulse response;

$\Delta\mathbf{H}_{\bar{y}\bar{x}}(e^{j\omega}) = \mathbf{H}_{\bar{y}\bar{x}}^{\mathrm{FP}}(e^{j\omega}) - \mathbf{H}_{\bar{y}\bar{x}}^{\mathrm{IP}}(e^{j\omega})$, where $\mathbf{H}_{\bar{y}\bar{x}}^{\mathrm{IP}}(e^{j\omega})$ represents the frequency response matrix from $\bar{x}$ to $\bar{y}$ in the IP system, similarly defined as in $\mathbf{H}_{\bar{y}\bar{q}}(e^{j\omega})$, whereas $\mathbf{H}_{\bar{y}\bar{x}}^{\mathrm{FP}}(e^{j\omega})$ is the frequency response matrix in the FP system where the constant coefficients such as filter tap-gains are quantized.

As stated in Example 2 of Section VI, (2-37) can give (2-33) for LTI systems, whereas (2-36) can give (2-37) partially. Here we will only give the proofs for the 1-input-1-output LTI system. Again, let $\mathrm{H}_{yx}(e^{j\omega})$ be the frequency response from input x to output y. Rewrite the difference between the frequency responses of FP and IP systems as

$$\Delta\mathrm{H}_{YX}(e^{j\omega}) = \bar{u}^T(e^{j\cdot n_1\omega},...,e^{j\cdot n_{\mathrm{L}c}\omega})^T, \tag{A-1}$$

where $\bar{u}$ is the column vector formed by the differences of the L_c constant coefficients between the FP and IP systems, whereas $\mathrm{n_i}$ is the number of unit delays from the *i*-th

constant coefficient to the output. With Assumption 1 and round-off quantization modes, the quantization noise correlation matrix becomes diagonal

$$\mathbf{R}_{\overline{qq}} = \begin{pmatrix} s_1^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & s_{\mathrm{L}_q}^2 \end{pmatrix}. \tag{A-2}$$

With (A-1) and (A-2), (2-37) gives the scalar power spectrum density output difference between FP and IP systems as a function of frequency, that is,

$$\begin{aligned} &\mathrm{R}_{\Delta y \Delta y}(e^{j\omega}) \\ &= \overline{u}^T \left[(e^{j\cdot n_1 \omega}, ..., e^{j\cdot n_{\mathrm{L}c} \omega})^T \cdot \mathrm{R}_{xx}(e^{j\omega}) \cdot (e^{j\cdot n_1 \omega}, ..., e^{j\cdot n_{\mathrm{L}c} \omega}) \right] \overline{u} \\ &+ \sum_i^{\mathrm{L}_q} |\mathrm{H}_{yq_i}(e^{j\omega})|^2 \cdot {s_i}^2, \end{aligned} \tag{A-3}$$

where neither the column vector $\overline{u}$ nor standard deviation s_i, of the white quantization noises, is a function of frequency. Integrating over frequency on both sides and applies Parcevel's theorem (see, e.g. [4] or [5]), (A-3) gives the MSE of the steady-state output,

$$\begin{aligned} &E[(f_{\mathbf{S_{FP}}}(x_1, x_2, \cdots, x_{\mathrm{K}}, t) - f_{\mathbf{S_{IP}}}(x_1, x_2, \cdots, x_{\mathrm{K}}, t))^2] \\ &\overset{t\to\infty}{=} \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathrm{R}_{\Delta y \Delta y}(e^{j\omega}) \cdot d\omega \\ &= \overline{u}^T \left[\frac{1}{2\pi} \int_{-\pi}^{\pi} (e^{j\cdot n_1 \omega}, ..., e^{j\cdot n_{\mathrm{L}c} \omega})^T \mathrm{R}_{xx}(e^{j\cdot n_1 \omega}, ..., e^{j\cdot n_{\mathrm{L}c} \omega}) \cdot d\omega \right] \overline{u} \\ &+ \sum_i^{\mathrm{L}_q} \left[\frac{1}{2\pi} \int_{-\pi}^{\pi} |\mathrm{H}_{yq_i}(e^{j\omega})|^2 \cdot d\omega \right] \cdot {s_i}^2, \end{aligned} \tag{A-4}$$

This proves (2-33) at steady state by identifying the terms in the two brackets as B and c_i, respectively. (A-4) provides the explicit expressions for these coefficients that appear in (2-33). Yet, unlike (2-33), (A-4) no longer applies in transition period.

A.2 Partial derivation of (2-37) from (2-36)

Let g_m in (2-37) be an unbiased LTI system with frequency response $g_m(e^{j\omega})$ with $g_m(0,t)=0$; and again, let $\Delta y(t)$ be $f_{\mathbf{S_{FP}}}(x_1,\cdots,t) - f_{\mathbf{S_{IP}}}(x_1,\cdots,t)$. Then, the term $g_m\left(f_{\mathbf{S_{FP}}}(x_1,\cdots,t) - f_{\mathbf{S_{IP}}}(x_1,\cdots,t),t\right)$ can be written as the convolution product $g_m(t) \otimes \Delta y(t)$, denoted as $o(t)$. If the input is WSS (wide-sense-stationary), at steady state (when t is large), $o(t)$ is also a WSS random process; so, the time index can be removed to get

$$\begin{aligned}&\lim_{t\to\infty} E[\{g_m\left(f_{\mathbf{S_{FP}}}(x_1,\cdots,t) - f_{\mathbf{S_{IP}}}(x_1,\cdots,t),t\right)\}^2] \\ &= \lim_{t\to\infty} E[\{o(t)\}^2] = E[o^2].\end{aligned} \tag{A-5}$$

Let $\mathrm{R}_{oo}(e^{j\omega})$ be the power spectrum density of o, it can be written as the product between power spectrum density of y and the squared frequency response of LTI system $\mathrm{g_m}$ (see, e.g. [3] or [4]), that is,

$$\mathrm{R}_{oo}(e^{j\omega}) = |\,\mathrm{g}_m(e^{j\omega})\,|^2\ \mathrm{R}_{\Delta y\Delta y}(e^{j\omega})\ . \tag{A-6}$$

Integrate over frequency domain and divide both sides by $\frac{1}{2\pi}$ gives,

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} \mathrm{R}_{oo}(e^{j\omega})\cdot d\omega = \frac{1}{2\pi}\int_{-\pi}^{\pi} \mathrm{R}_{yy}(e^{j\omega})\,|\,\mathrm{g}_m(e^{j\omega})\,|^2\cdot d\omega \tag{A-7}$$

On the other hand, (2-36) says that

$$\begin{aligned}&\lim_{t\to\infty} E[\{g_m\left(f_{\mathbf{S_{FP}}}(x_1,\cdots,t) - f_{\mathbf{S_{IP}}}(x_1,\cdots,t),t\right)\}^2] \\ &= \overline{u}^T\mathbf{B}^{g_m}\overline{u} + \sum_i^{\mathrm{L}} {c_i}^{g_m}\cdot {s_i}^2,\end{aligned} \tag{A-8}$$

where the right side is no longer a function of *t*. According to Parceval's theorem, $\frac{1}{2\pi}\int_{-\pi}^{\pi} \mathrm{R}_{oo}(e^{j\omega}) \cdot d\omega = E[o^2]$; so (A-5), (A-7) and (A-8) are exactly the same. Equating the right side of (A-7) and (A-8) and multiplying both sides by 2π which gives

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} \mathrm{R}_{\Delta y \Delta y}(e^{j\omega}) \,|\, \mathrm{g}_m(e^{j\omega}) \,|^2 \cdot d\omega = \overline{u}^T \mathbf{B}^{g_m} \overline{u} + \sum_i^{\mathrm{L}} {c_i}^{g_m} \cdot {s_i}^2. \tag{A-9}$$

Now, let LTI system $g_m^{\omega_0}$ be designed in such a way that the magnitude-square of its frequency response, which must be periodic, approximates a sum of impulse train based on Dirac functions, that is,

$$|\, \mathrm{g}_m^{\omega_0}(e^{j\omega}) \,|^2 = 2\pi \sum_{k \in \text{Integer set}} \delta(\omega - \omega_0 - 2\pi \cdot k). \tag{A-10}$$

Substituting (A-10) in (A-9) and carrying out the integral on the left side gives

$$\mathrm{R}_{\Delta y \Delta y}(e^{j\omega_0}) = \overline{u}^T \mathbf{B}^{\omega_0} \overline{u} + \sum_i^{\mathrm{L}} {c_i}^{\omega_0} \cdot {s_i}^2, \tag{A-11}$$

Replacing ω_0 with ω, we see that (A-11) is almost the same as (A-3), except that (A-3) provides the explicit expressions for $\mathbf{B}^{\omega}$ and ${c_i}^{\omega}$. Remembering that (A-3) is equivalent to (2-37), we asserts that (2-37) has been partially derived from (2-36) with much less effort than being stated in [3].

Appendix B. Another Way to Derive (3-7) for Gaussian θ

When θ is Gaussian distribution, we can derive (3-7) specifically by carrying out the expectations together with some further simplification. Then, starting from (3-5), we get

$$\begin{aligned} & P(f_{\mathbf{SL}}(x_{\mathbf{FP}}) = -1, f_{\mathbf{SL}}(x_{\mathbf{IP}}) = 1) \\ = & \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{\pi}} e^{-\frac{1}{2}\gamma^2} - \gamma \cdot erfc(\gamma)\right) p_x(0) \cdot \sigma_\theta \cdot \end{aligned}$$

Here $\gamma = \frac{\mu_\theta}{\sqrt{2} \cdot \sigma_\theta}$ and *erfc*(·) gives the complimentary error function, that is,

$$erfc(z) = 1 - \frac{2}{\sqrt{\pi}} \cdot \int_0^z e^{-x^2} dx .$$

It is more intuitive to notice that

$$\begin{aligned} & P(f_{\mathbf{SL}}(x_{\mathbf{FP}}) = -1, f_{\mathbf{SL}}(x_{\mathbf{IP}}) = 1) \\ < & \begin{cases} \frac{p_x(0)}{\sqrt{2\pi}} \sigma_\theta, \text{if } \mu_\theta \geq 0 \\ \frac{p_x(0)}{\sqrt{2\pi}} (\sigma_\theta + 2\sqrt{\pi}|\mu_\theta|), \mu_\theta < 0 \end{cases} \end{aligned}$$

Similarly

$$
\begin{aligned}
&P(f_{\mathbf{SL}}(x_{\mathbf{FP}})=1, f_{\mathbf{SL}}(x_{\mathbf{IP}})=-1) \\
&= \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{\pi}}e^{-\frac{1}{2}\gamma^2} + \gamma \cdot erfc(-\gamma)\right) p_x(0)\cdot\sigma_\theta \\
&< \begin{cases} \dfrac{p_x(0)}{\sqrt{2\pi}}\sigma_\theta, \text{if } \mu_\theta < 0 \\ \dfrac{p_x(0)}{\sqrt{2\pi}}(\sigma_\theta + 2\sqrt{\pi}|\mu_\theta|), \mu_\theta \geq 0 \end{cases}
\end{aligned}
$$

It is often the sum of these two probabilities that is most relevant as

$$
\begin{aligned}
&P(f_{\mathbf{SL}}(x_{\mathbf{FP}}) \neq f_{\mathbf{SL}}(x_{\mathbf{IP}})) \\
&< \frac{p_x(0)}{\sqrt{2\pi}}(2\sigma_\theta + 2\sqrt{\pi}|\mu_\theta|) \\
&< p_x(0)\sqrt{E[\theta^2]}.
\end{aligned}
$$

This finishes a more complicated way to prove (3-7) for Gaussian-distributed θ.

Appendix C. FFC Tutorial

Caution: for the up-to-date version of this tutorial, please refer to the version on [88].

For the A simple BPSK communication system using root-raised-cosine filter on both transmitter and receiver is built in this tutorial, which depicts the design processing using Simulink and Xilinx System Generator. Starting from choosing the algorithm to building the floating-point system with architecture information, the system is then converted into fixed-point using our FFC tool.

C.1 Introduction

Can we design a digital chip in a day? Research efforts in Berkeley Wireless Research Center (BWRC) and other places have indicated this is achievable [56]. Built on top of Mathlab, Simulink and Xilinx System Generator, a number of customized Matlab scripts and Simulink libraries automate our FFC design flow. By studying on how our FFC tool can be used in designing a simplified transmitter-receiver system, this tutorial will get you familiarized with it as well as the design environment.

C.2 How to start

Warning: this section of the tutorial may become outdated over time as it is related to some administrative information that changes often. For now, it is assumed that

you have a BWRC account to access all the software resources on our server. Please refer to our website for up-to-date information [88], which also provides the instructions of how to use our source codes at your local machine.

You will need a computer with Matlab, Simulink, Xilinx System Generator installed in order to run through this tutorial. You also need read/execute access to BWRC file server \\hitz.eecs.berkeley.edu\designs to use our Floating-point to Fixed-point Conversion (FFC) Tool. In addition if you want to learn how to map your design to FPGA, you need to refer to other tutorial such as System Generator Tutorial or the tutorial on (Berkeley Emulation Engine) BEE (link available at [88]).

A simple way to solve the problem is to login the MS Windows Remote Desktop Servers available in BWRC, intel2650-2.eecs.berkeley.edu. You will need Remote Desktop Connection Client on your local PC to do that. If you are using Linux, you may use Rdesktop (http://www.rdesktop.org). The server has all the necessary tools installed correctly.

Once you have the software ready, you need to map \\hitz.eecs.berkeley.edu\designs to your network drive, preferably H: disk. The example communication system in this tutorial can be found at H:\ffc\ffc_tutorial.mdl. If you have never used one of the three tools before, you need to read Section C.2.1. Otherwise you can proceed to C.3.

C.2.1 Getting familiar with the environments

The quick way to get started on these tools is to see an existing design. You can do so by type in

```
>>demo
```

in Matlab command line, and start to play around the demo systems there. Notice that Xilinx demos are located at Blocksets→Xilinx directory in the demo window. An example is shown in Figure A1.

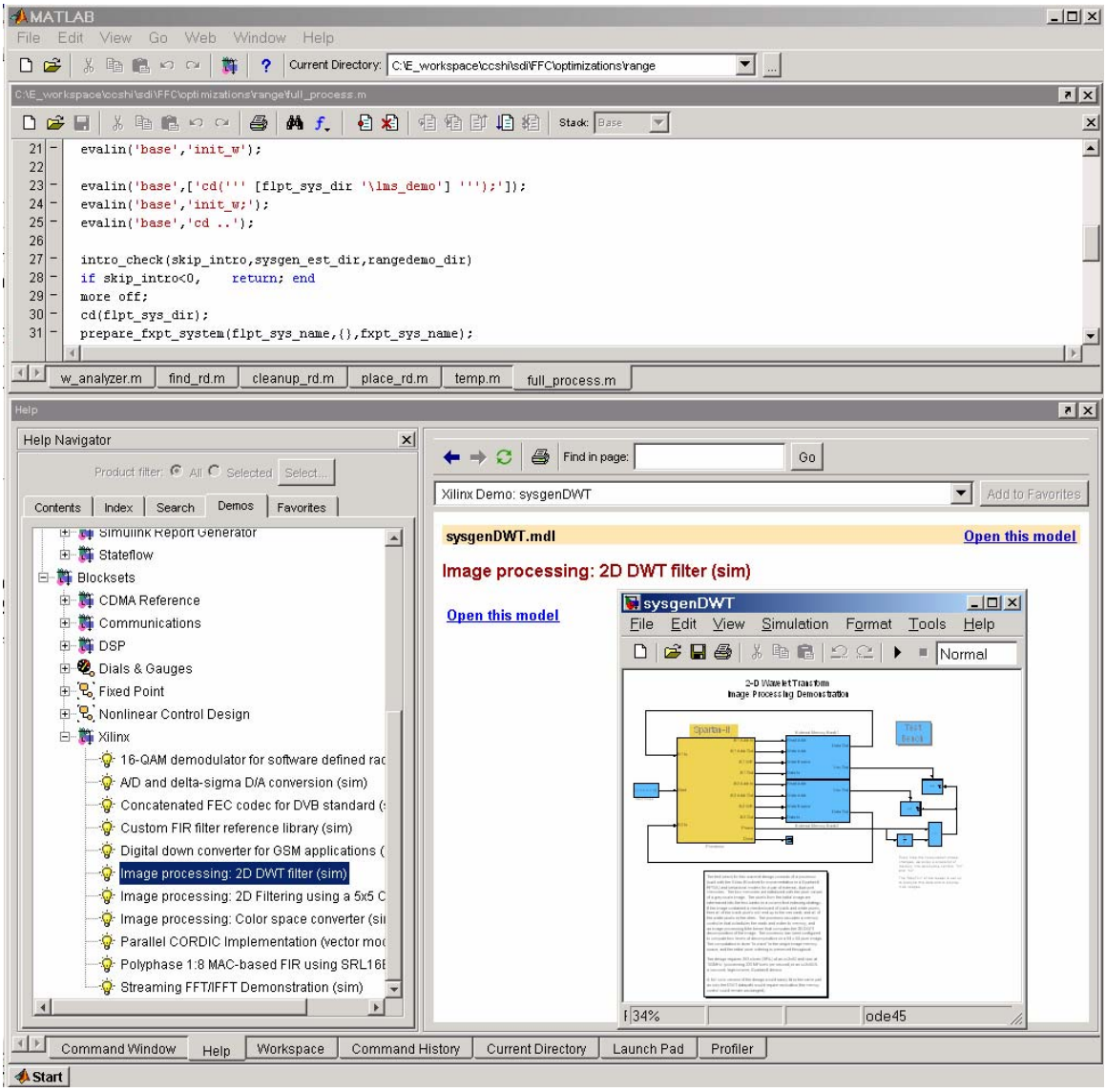


Fig. C-1 Using Matlab demos

If you wish to learn these tools in more a systematic way, please pay more attention on the help file, with a window somewhat like Fig. C-2.

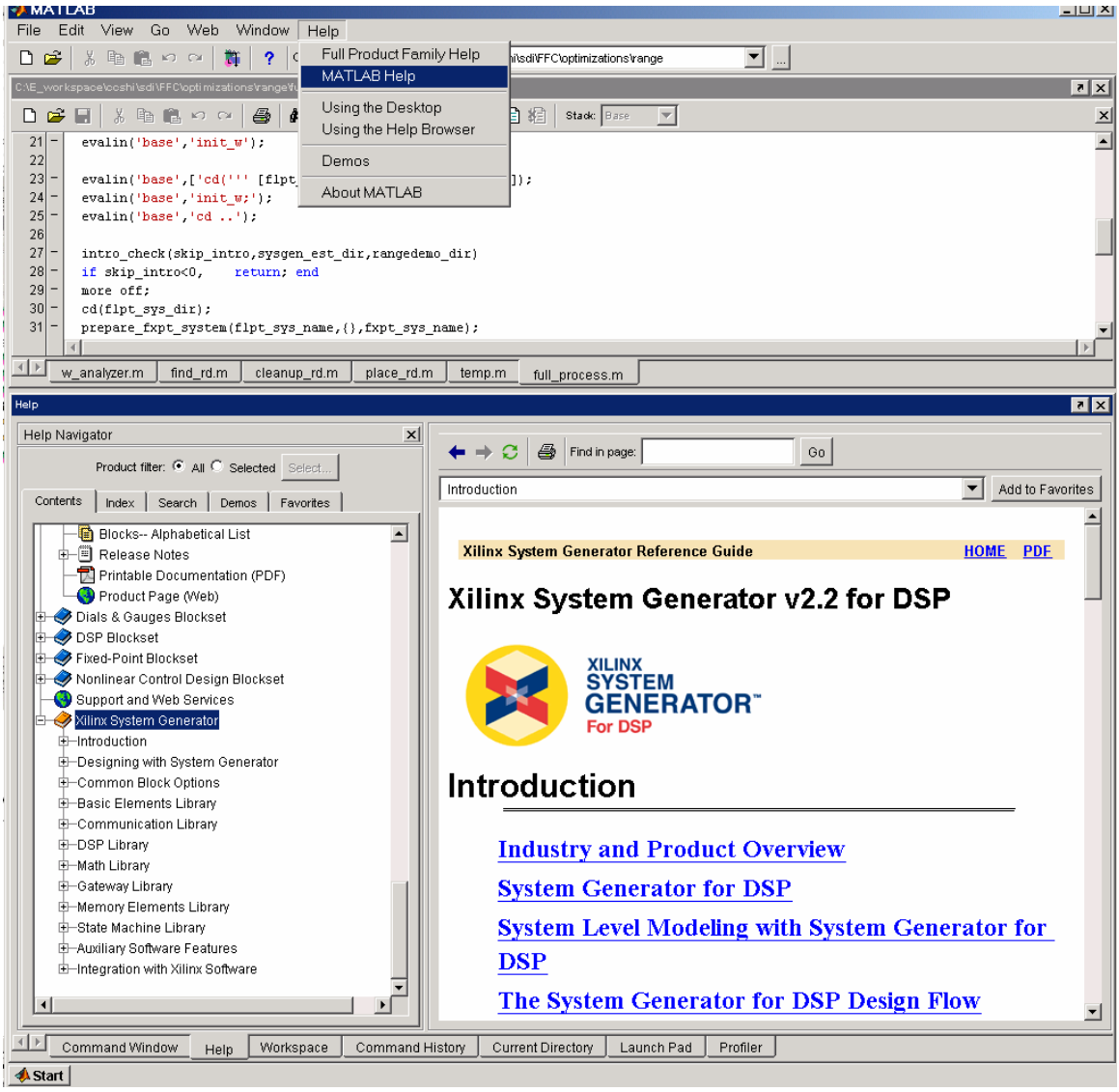


Fig. C-2 Using the Matlab help system

If you still have questions related to Matlab and Simulink, and could not be answered by anybody around you, you might contact help@mathworks.com. They usually respond within the same day.

C.2.2 Using FFC Tool

You should have already mapped \\hitz.eecs.berkeley.edu\designs to H: disk. Now go to H: disk in Matlab:

```
>>cd H:
>>cd ffc
>>ffc_init
```

The last command above swaps the Xilinx library to the version that is prepared for fast hardware resource estimation. You should see some library opened and closed when executing this command. In addition, a few Matlab paths containing FFC scripts are added to the path file. A good way to check that you have successfully done this initialization is to open Xilinx blockset, and see whether you have the resource estimator block in the Basic Elements.

To place the Specification Marker block in your design as mentioned in section 7 the FFC library can be opened by the following command:

```
>>ffc_lib
```

Now you can go to your own directory where your pseudo-floating point system is located, and type in:

```
>>ffc
```

The tool itself will then lead you sequentially through the FFC process. This ffc.m script is located at H:\ffc\ffc_package directory that you have linked to in the initialization step. The definition of many variable names and functions can be found using:

>>help_ffc('keyword').

C.3 Algorithm Study

The system to be built here is a BPSK system as mentioned in Chapter 3 and 4. Suppose we want to do base-band communication with 2-PAM modulation scheme at 1Mbits/sec. Under 2-PAM input symbols (1 symbol/ 1us), such as sequence choosing from binary integer {0,1}, are mapped into a data sequence choosing from {-A, A}. For convenience, we can let A = 1. The receiver needs a 2-PAM demodulator to map received signal into original integer. Suppose the channel impose additive white Gaussian random noise, but otherwise ideal.

Although we have assumed the channel is flat with no fading, in reality it could be band-limited (caused by, for example, RF front-end filtering). Thus rectangular base-band pulse in time domain (sinc(.) shape in frequency domain) through the channel will be clearly distorted. One technique to combating this is to have a low-pass pulse-shaping filter at the transmitter side [46]. To do so, one needs to first over-sample the data sequence at R MHz. This is usually done by an upsampler with integer R. A condition R>2 is necessary to satisfy the Nyquist criteria.

However there are multiple reasons to make R even higher. One of them is to minimize the impairment on the frequency response due to finite-tap implementation of the filter, which causes none-zero stop-band response and hence aliases after the received signal is downsampled. This is what usually called inter-symbol-interference (ISI). Without higher upsampling rate R, this deterioration can be alleviated with the cost of higher filter complexity and signal latency. Another reason of having large R is for time and frequency recovery. When the channel together with RF front-end has a fractional

delay of symbol period, upsampled sequences are needed to identify the right fractional delay "adjustment" the receiver needs to tune [46].

On the other hand, choosing R too high would lead to high clock rate on the digital filter, A/D and D/A converters, which are not desirable. In our case, without much information of other constraints, let's set R = 4.

On the receiver side, it is desirable to have a matched filter that matches the pulse-shaping filter on the transmitter side. With this consideration a commonly used filter shape, called root-raised-cosine filter is used in both transmitter and receiver. After the downsampler on the receiver side, the signal will be perfectly reconstructed if the two root-raised-cosine filters are ideal. Figure 1 shows the algorithms we have conceived so far.

It should be pointed out that if the channel is really as simple as AWGN, one can just feed the 2-PAM modulated signal into the channel. We included more blocks in the design to combat some other channel impairments that are not present here.

C.4 Building the floating-point System – algorithm validation

We can start to write either C or Matlab codes for each of the functional block of Fig. 3-6 and see if the output symbols agree with the input ones by doing simulations. This is what conventionally people would do. This is still a good way to understand your system; however, a more natural way exists to validate our algorithm; that is to use existing Simulink library blocks to draw the diagram in Simulink quickly. A snapshot of the completed system is shown in Figure C-3.

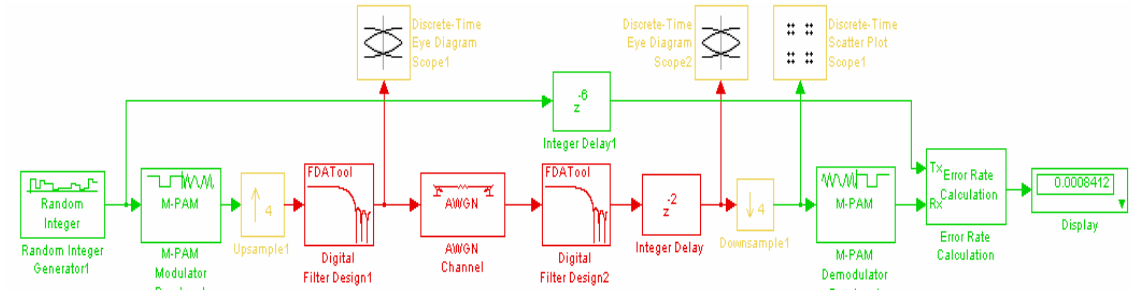


Fig. C-3 Floating-point system in Simulink™ blockset

Notice that there is almost a 1-1 correspondence between above Simulink system with the block diagram in Fig. 3-6. Different colors of the blocks indicate different clock rate. Here let's explain some of them in more detail.

First of all several display blocks are used to help us debug/understand the system. These include the Display block, Deiscrete-time Scatter Plot Scope, and Discrete-Time Eye Diagram Scope. A number of other very useful display blocks can be found in Simulink→Sink library and Communication Blockset→Comm Sink library.

Secondly, the Error Rate Calculation block is used to compare the Tx signal with the Rx ones, and output bit-error-rate (BER).

A couple Integer Delay Blocks are used to synchronize the Tx and Rx signals. In our design both the Tx filter and Rx filter introduce 11 delays (each delay corresponds to 1/(4MHz) = ¼ μs) on their center tap. So another 2 delays of ¼ us are introduced to make the total delay

¼ (11+11+2) = 6 μs,

which is an integer multiple of the symbol period. Without using the integer delay of 2, a large ISI would be seen on Scatter plot; that is, the down-sampler would not sample at the wide-open instance showed in the eye diagram.

Finally two Digital Filter Design blocks are used for the two rRC filters. These two identical blocks are specified using the design mask showed in Fig. C-4.

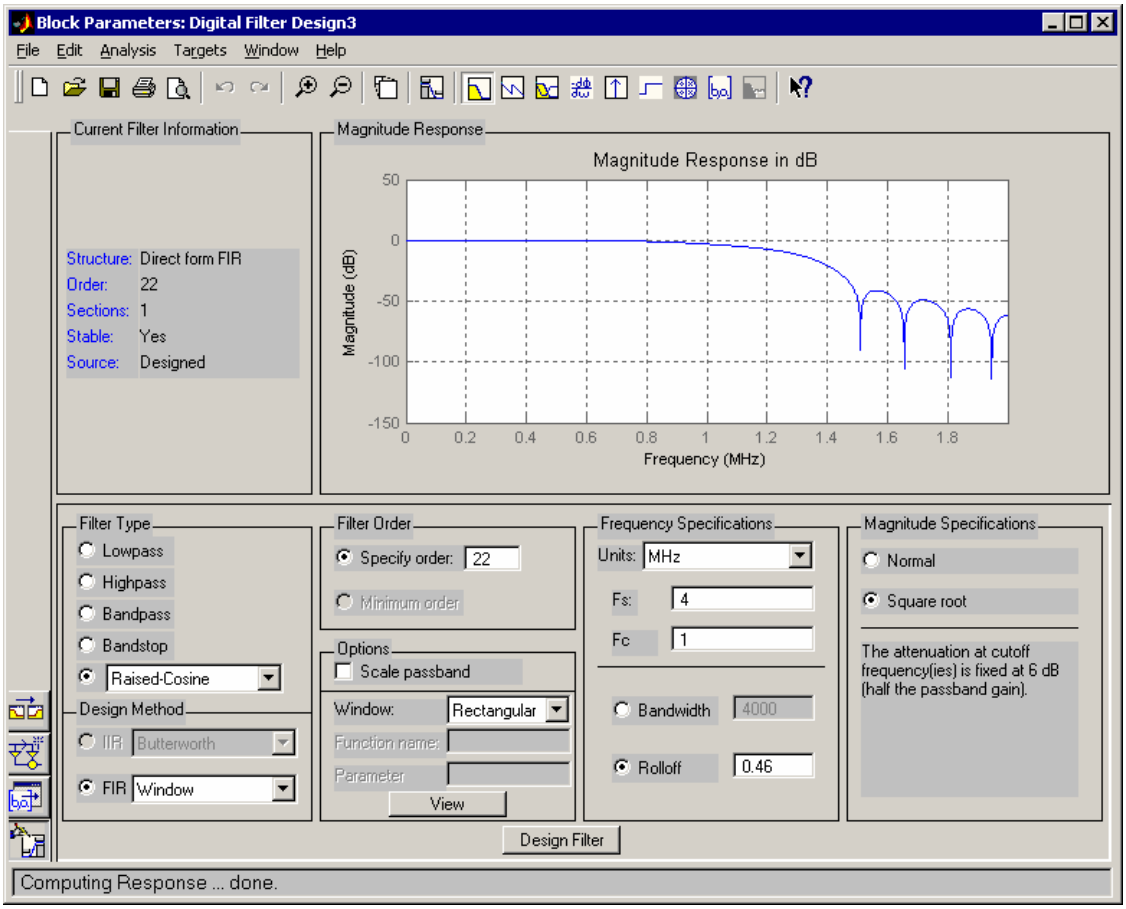


Fig. C-4 Design a root-raised-cosine filter

Here we choose Rectangular window method without trying others. To understand window method, please refer to [4]. The sample frequency is 4MHz since we choose R=4. Rolloff factor is chosen to be 0.46. The higher rolloff factor is, the more relaxed the filter is and the less number of taps will be needed. That would also lead to

more excess bandwidth (total bandwidth needed will be [-(1+rolloff) MHz, (1+rolloff) MHz]). In our system this rolloff factor is another degree of freedom in design; but let's fix it for simplicity. The last parameter that is adjustable is the filter order, we choose the lowest filter order that satisfies the side-band from [1.5MHz, 2MHz] to be 40dB less than the main-lobe, as shown in Fig. C-4. You may try to use Matlab function

```
>>help rcosfir (or firrcos)
```

to do the task. Then a Matlab script can be written to automatically determine the lowest filter order given different choices on Rolloff, windowing method, etc. Here we just try some deferent filter order and found 22 is the minimum one satisfying our specification. Other rolloff factor results to higher or the same filter order. This justifies our choice of rolloff factor of 0.46.

Once the filter coefficients are found one can specify them in a Digital Filter block in Simulink, which basically does the same thing as the Digital Filter Design block. But we won't try that approach here. The filter coefficients can be exported to workspace choosing File→Export in Fig. C-4, as also shown in Fig. C-5.

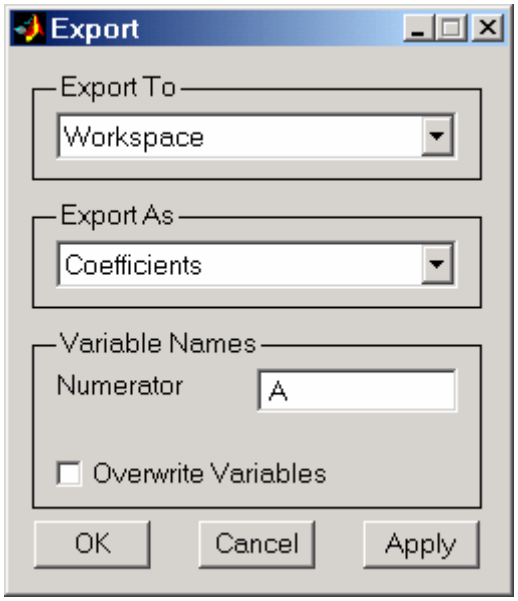
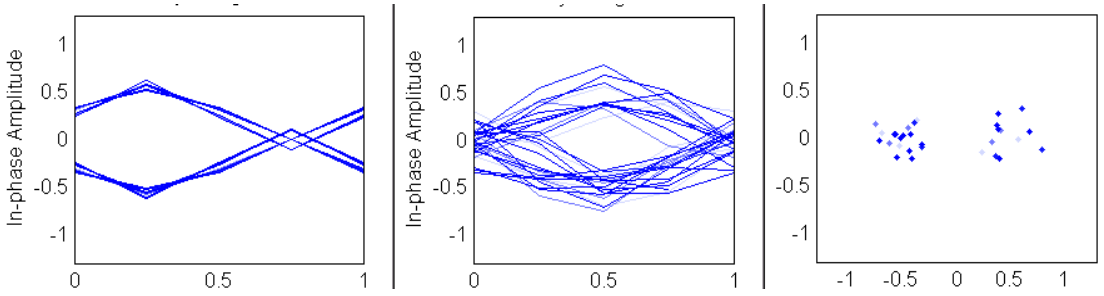


Fig. C-5 Exporting coefficients to workspace vector A

With the two filters designed above, and a channel noise power of 0.1 (i.e. 0.05 for both I-channel and Q-channel), we get the following system performance in Fig. C-6.



a) b) c)

Fig. C-6 Floating-point performance of the BPSK
a) Eye diagram of the transmitted signal, b) eye diagram of the received signal, c) scatter plot before the demodulator

It can be seen that the Tx rRC filter caused some ISI as shown in Fig. C-6-a. The eye is further closed by AWGN noise as shown in Fig. C-6-b. Therefore, the constellation points become blurred in the scatter plot in Fig. C-6-c. As indicated in the

right-most display of Fig. C-3 is the bit error rate display. The error rate calculation block is set in such a way so that 100 bit errors are detected before we stop the simulation. Assuming bit error comes in Poisson process, then the real BER in the following interval with .95-confidence level [3].

$$
\begin{aligned}
&[(100-1.96\sqrt{100})/100,\ (100+1.96\sqrt{100})/100]\times(\hat{BER})\\
&=[0.804,1.196]\times 0.00084\\
&=[0.00067,0.00106].
\end{aligned}
$$

The simulation takes about 30 minutes to finish.

C.5 Building pseudo-floating-point system in System Generator

The floating-point system built in the preceding section can now serve as our system reference. The next step is to impose the architecture information into the system. Xilinx System Generator blocksets (Version 2.3) are used to realize the architecture choice. Historically, we have used the granular blocks of Simulink, such as multiplier, adder etc. for this step. However it turns out it's just easier (for the rest of the BEE or INSECTA flow), though not essential, to build the system directly from System Generator library. Note that the blocks in SysGen library only support fixed-point data-type (but with double over-ride functionality in simulation). This won't cause much difficulties here since we can just choose all the word lengths to be very high whenever possible [3]; when this is done, we call the system pseudo floating-point system with architecture information. This is a good way to validate the architecture choice.

Choosing the architecture correctly is an important task [46]. For example, in our example for the filter structure one can use the built-in FIR block in System Generator DSP library, which is based on distributed arithmetic to save area. But it is often not

power-efficient since the pre-stored partial products need to be frequently loaded from the memory block. Without too much justification, let's use the Delay, Cmult, AddSub, Upsampler, Downsampler, and Gateway In/Out block only to build the system. We want to minimize the number of such blocks in our design. Therefore we explore the linear phase property of the rRC filter. Furthermore the center tap can be normalized to 1 to save another Cmult. The resulting structure is shown in Fig. C-7 and Fig. C-8. A gain of value A(12) (the 12th element of vector A) is used in order to bring the total transmitting power the same (one can think it as analog gain, so it does not consume Cmult).

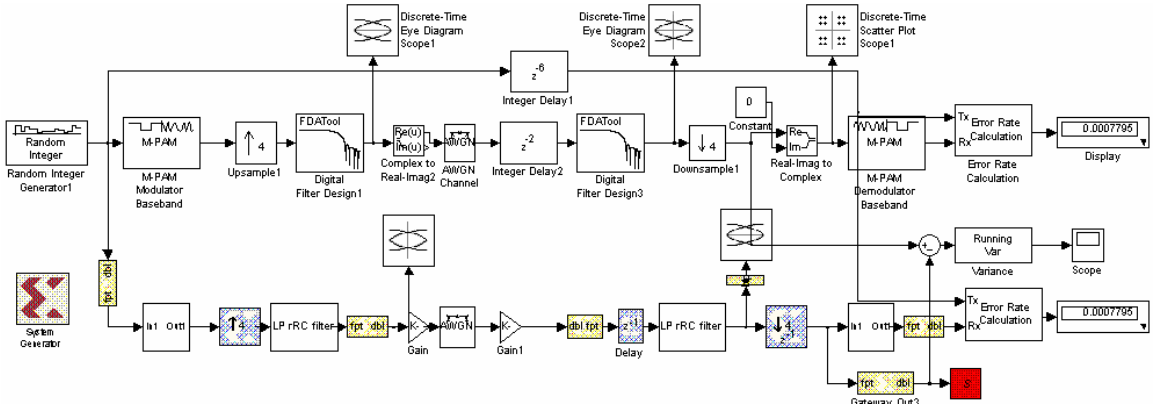


Fig. C-7 Pseudo flpt system in system generator
Simulate time is 0s to 2s (two million output bits are detected)

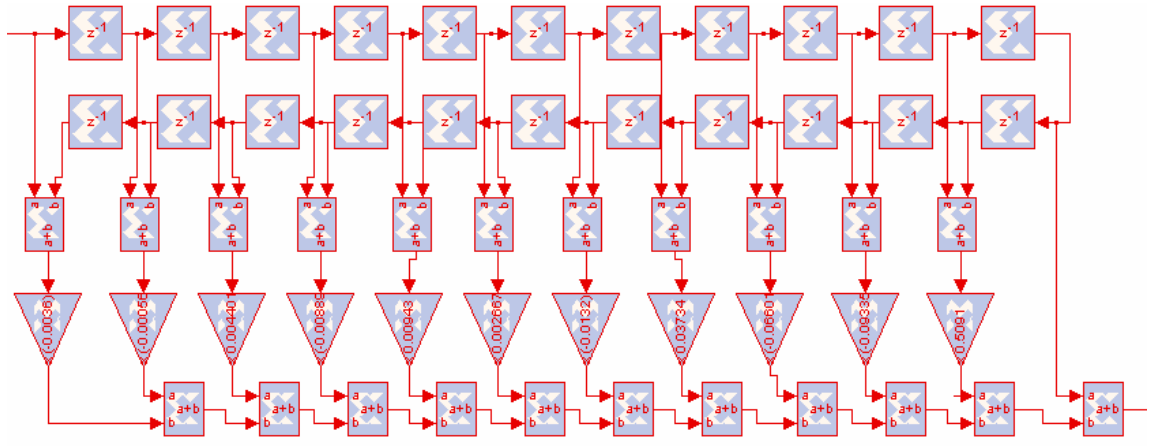


Fig. C-8 LP rRC filter

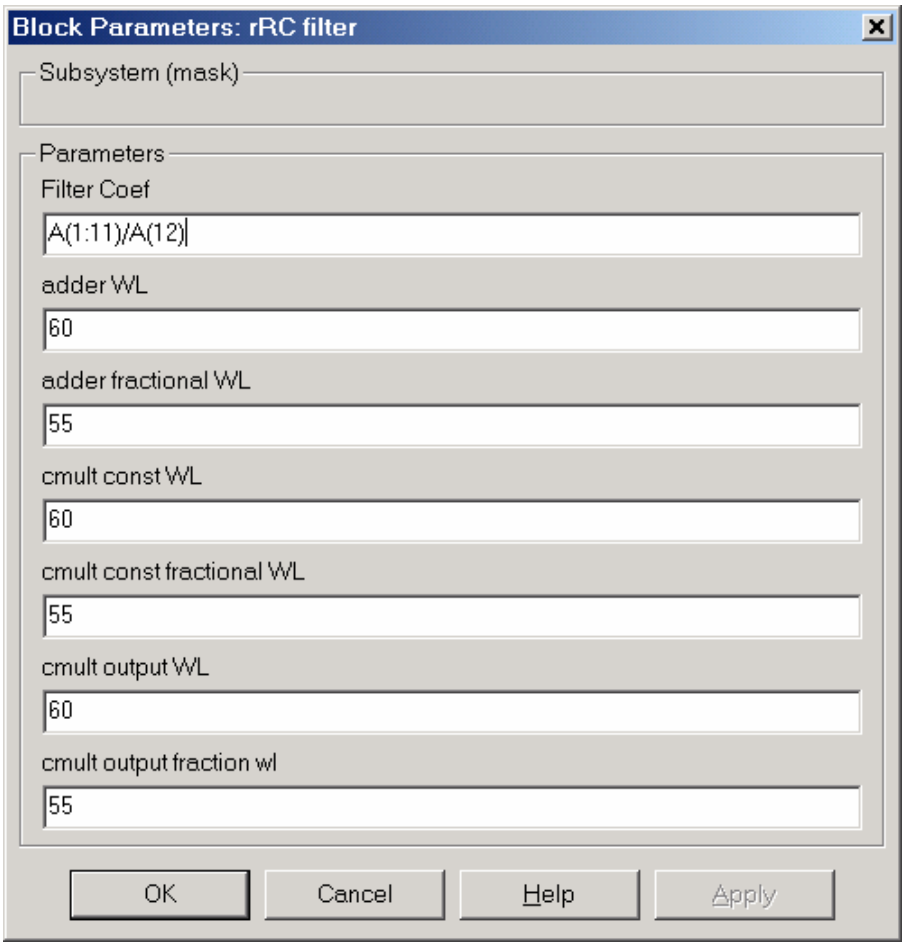


Fig. C-9 Mask parameters of LP rRC filter

Fig. C-8 and C-9 show the detailed structure of the LP rRC filter, and its mask. We have set all the WL to be exceptionally high (60 bits). Simulation indicates the pseudo flpt system and original flpt system performs the essentially the same—the numerical difference is much less than what we care about.

Here be careful that since the original system has both I and Q channels, the noise power indicates the sum of I and Q noises. So, we should choose noise power to be 0.1/2 to get the same BER as previous floating point. The reference system is also modified so contain only the I-channel. With this modification the pseudo-flpt system and flpt system do exactly the same thing up to each cycle—so called “cycle accurate”.

A long duration [0, 2s] is used for the simulation, the BER is found to be 0.0007795, i.e. 2s × 1MHz × 7.7795 ×10^{-4} = 1559 errors. So the 0.95 confidence interval estimate of BER is

$$
\begin{aligned}
&[(1559-1.96\sqrt{1559})/1559,\ (1559+1.96\sqrt{1559})/1559]\times(\hat{BER})\\
&=[0.95,1.05]\times 0.0007795\\
&=[0.00074,0.00082].
\end{aligned}
$$

The simulation takes about 8 hours to finish.

C.6 Building fixed-point System using FFC

Now all the algorithm and architecture decisions have been made in our design. What is left is to decrease the word lengths presented in the previous section, and to determine all the overflow and quantization modes. The goal is to have this done automatically, which results in the floating-point to fixed-point conversion (FFC) tool.

In order to have the conversion, one needs to first identify the node where the difference between fixed-point and floating-point systems will be checked. This is practically done by inserting a Specification Marker block from the FFC library that is also located in H:\ffc\ffc_package\ffc_lib.mdl. From Chapter 3, we know that a natural node to place the marker is the one after gateway-out block of the receiver, which is the only strong decision-making block in the system. At this node, the bit error rate solely caused by quantization noise can be detected. Theoretically, placing the Spec Marker here is a good choice. However, as described in Chapter 4, in practice it is often less attractive due to the long simulation time to fulfill the estimate of a BQER accurately. In

fact since it is normally necessary to have BQER less than BER at least the same number of input samples as the one in previous section are needed to get a high-confidence estimate. That corresponds to long simulation duration for each run, which is too long as many iterations need to be performed. In general, the total BER with both channel noise and quantization noise is not the sum of the flpt BER (without QN) and this BQER, because a slicer (demodulator) block is a nonlinear function of noise power (it is a Q-function of SNR).

From Chapter 3 and 4, a much more robust node to place the Specification Marker block is the one before the 2-PAM demodulator. One reason is that we know the rest of the receiver following this node (the only block left is just a demodulator, i.e. a slicer) does not have word lengths to be determined. Another reason is that the MSE(flpt-fxpt) at this node gives a good indication of the BER performance after the demodulator. In fact, assuming QN and channel noise cause uncorrelated Gaussian noises at this node, it is equivalent to think their sum as a total noise power. So one just needs to make sure the QN power much less than the channel noise power at this node to quantify the statement “fxpt system differs only little from flpt system”.

A system with the marker specified is displayed is Fig. C-10. Compared with the previous design in Fig. C-7, many of the unnecessary blocks have been eliminated here. For example since we already have the pseudo flpt system in System Generator blocks, the first version of the system designed in pure Simulink block set has been deleted. You can leave those blocks there with possibly a slow-down of simulation speed. This newer version is named ffc_tutorial_v2.mdl. One can see that a specification marker has been

placed before the demodulator. In addition you can find some supporting Matlab files in the same directory (H:\ffc\systems\ffc_tutorial1\ ); they are:

System_init.m
ffc_setting.m
and A.mat.

In order to continue the demonstration yourself you need to copy ffc_tutorial_v2.mdl, system_init.m, ffc_setting and A.mat into a working directory of which you have write-access. To prevent possible hazard H:\ffc is secured as read-access only. After the copy you can go to that directory and start the conversion tool yourself by typing in

>>ffc

If you have your own design to FFC you might want to have a directory for that design specifically. In that directory you should have a file named “system_init.m” that initializes your pseudo flpt system, and a file named “ffc_setting” to save FFC design parameters. In our current example, system_init.m basically loads the filter coefficients A.

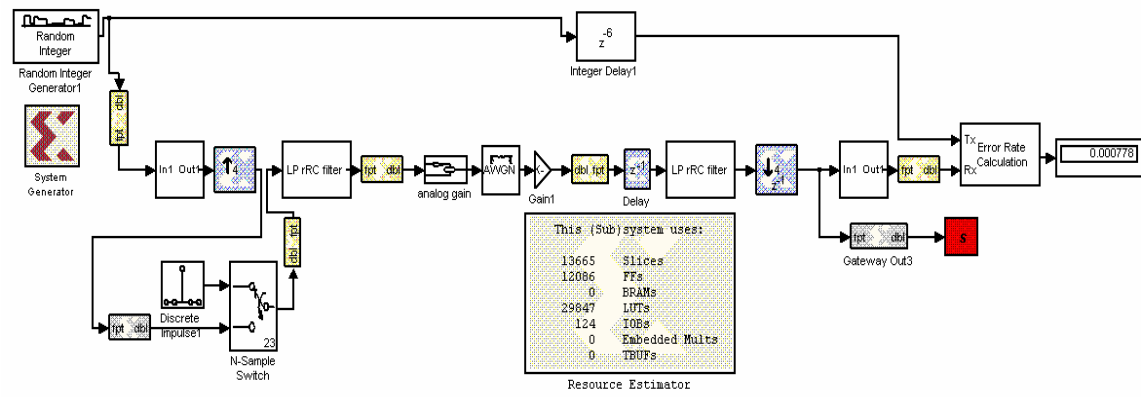


Fig. C-10 ffc_tutorial_v2.mdl file.
Comparing with figure 6 a specification marker and a resource estimator block are included. Furthermore some blocks supporting the floating-point design are eliminated/added to speed up/support simulation.

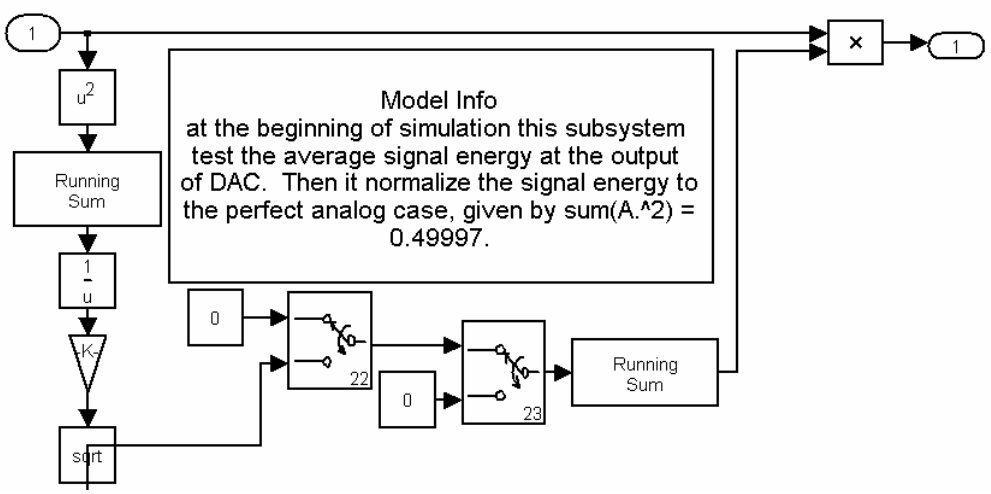


Fig. C-11 Analog gain subsystem in ttc_tutorial_v2.mdl

Another modification is replacing the first A(12) gain to the analog gain block and adding the impulse input in front of the Tx rRC filter. Altogether they make sure the average signal power to be transmitted is the same as the floating-point system. Here the average power of the transmitted signal x(n) is

$$
\begin{aligned}
& E[x(n)^2] \\
&= E[(\sum_m S(n-m)\cdot A_m)^2] \\
&= \sum_m \sum_l E[S(n-m)\cdot S(n-l)]\cdot A_m A_l \\
&= \sum_m \sum_l \delta_{ml}\cdot A_m A_l \\
&= \sum_m A_m{}^2 = \|A\|_2^2
\end{aligned}
$$

We have used the fact that the signal after modulator is a zero mean random process choosing from {1, -1}. Thus the signal power is just the 2-norm square of the filter coefficients. Sometimes the quantization of transmitter filter could increase the

transmitting power. Without normalizing the transmitting power the comparison of performances between floating-point and fixed-point systems is unfair.

To FFC this small system takes about 5 minutes. FFC tool sequentially asks you to input some important information you want to choose, such as design names. On the other hand it might be too lengthy to answer all the questions sequentially. Then you need to create an ffc_setting.m file in the directory and set the parameter “ask_question” to be 0. At one point it also asks you to change the model simulation time. You can input the simulation start and stop time as [0, 1/1e3] (to change simulation time right click your model window, and click Simulation Parameters, where you can see the parameter Start Time and Stop Time). This will make the simulation duration to be 1ms, which results in 1001 output samples at the Spec Marker. That is enough to have a good MSE estimation, assuming the flpt-fxpt difference error is a stationary random process. The assumption is justified since each quantization error is assumed to be stationary. If you use ffc_setting.m, you can see the parameters “sim_start_time” and “sim_stop_time” are set to be 0 and 1/1e3 separately.

Another required important user input is the MSE level you want to choose. You can either manually do a couple tries to understand the relationship between your system performance (e.g. BER) and MSE. You can also do what Section 3.5.2 of Chapter 3 has suggested. On the other hand, in the following we use a different approach, which is mostly analytical. Since the signal power before the demodulator is at about 0.6 (you can estimate it by placing an eye-diagram scope before the demodulator, and see the signal power), the physical noise (PN) power before the slicer of the BPSK system is about

PN power =
(Matlab command line input)
>>fzero('1/2*erfc(1/sqrt(2)*sqrt(.6/x)) – 7.795e-4', 0.1)
= 0.0600.

So suppose the BER deterioration due to quantization noise is 10% of the original BER, i.e. the final BER to be less than $7.995\times10^{-4} \times(1+10\%) = 8.57\times10^{-4}$, we need the quantization noise power (QNP) to be

QNP power =
(Matlab command line input)
>>fzero('1/2*erfc(1/sqrt(2)*sqrt(.6/(x+.06))) – 7.795e-4*(1+.1)', 0.1)
= 0.001.

Thus we need MSE < QNP power =0.001. To be robust, we assume there could be modeling error and estimation error, thus we set

MSE = ½ QNP power = 0.0005.

Next the grouping rules need to be defined. Without grouping, all Xilinx blocks are independently adjustable to find the minimum hardware cost. That would result a problem of too large optimization space (and turns out to be unnecessary in turns of design optimality). You can type

>>help_ffc('rules')

to understand more about the rules. The rules used for the following conversion in this section are [1, 2.1].

With the setting described above we achieve a fxpt system of about 356 slices as shown in Fig. C-12. A final simulation of duration [0, 1s] produces 811 errors; so the BER of the final fxpt system BER is within .95 confidence interval

$$[(811-1.96\sqrt{811})/811,\ (811+1.96\sqrt{811})/811]\times(\hat{BER})$$
$$=[0.93,\ \ 1.07]\times0.000811$$
$$=[0.00076,0.00087].$$

It is therefore of high chance the resulting fxpt system has BER less than the targeted 8.57×10^{-4}.

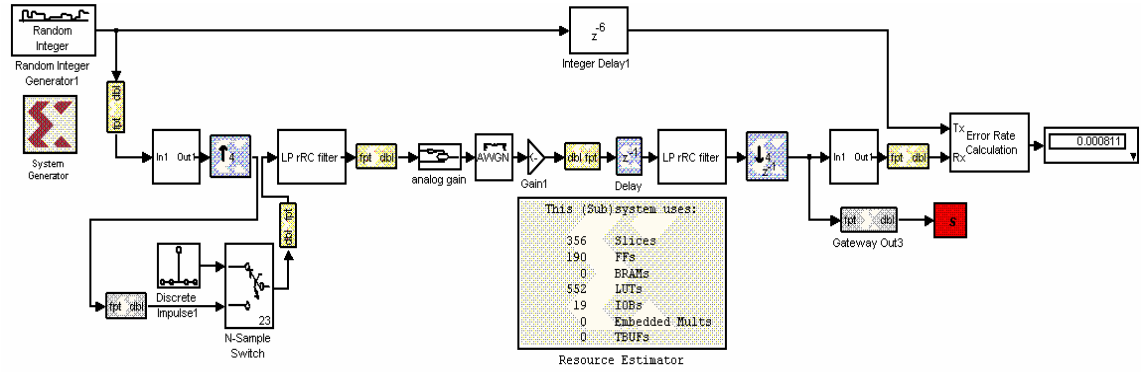


Fig. C-12 The final fxpt system.
With BER ~ 8.11×10^{-4}, and ~ 356 FPGA slices.

You can choose Format→show Port Data Types to see the fxpt data-types used for the final system. In fact, it is probably surprising to find that some of the constant multipliers have coefficients to be zero now (since the constant value is too small to be represented by the small WL fxpt datatypes). These logics will be automatically eliminated in the final placement-and-routing stage.

C.7 Multiple specifications

The FFC conversion in the preceding section is subject to one MSE specification constraint. The FFC tool can handle multiple specifications, some of which can even be non-MSE type. Recall that the transmitter filter frequency response in [1.5MHz, 2MHz] should be less than -40dB, it is natural to set this as the second specification. Thus one more Spec Marker block is inserted in the system, and saved to ffc_tutorial_v3.mdl as shown in Fig. C-13. This Spec Marker chooses "user specified spec. calculation function" as the specification type, and use "simulation_function" as the specification calculation function name. A snap shot of the block mask is shown in Fig. C-14. Thus

there is an associated Matlab function “simulation_function.m”. This function is written to calculate the highest frequency response in interval [1.5MHz, 2MHz]. Figure C-15 shows the frequency response of the resulting system. The simulation to show BER is again about 8 hours for duration [0, 1s].

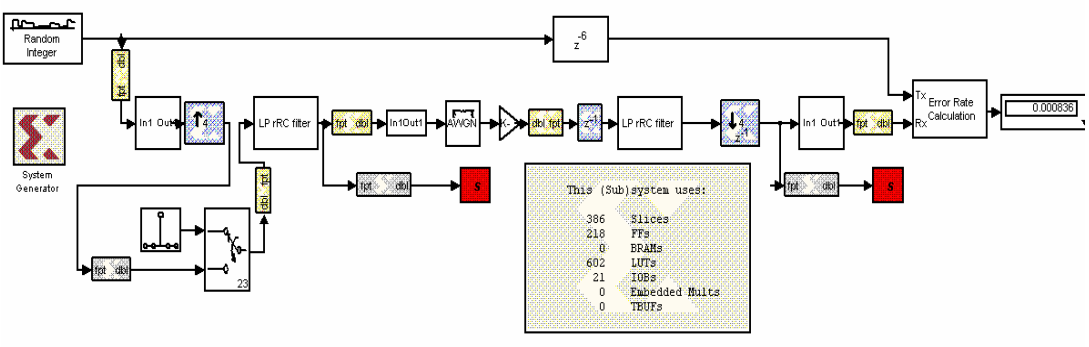


Fig. C-13 386 slices and BER ~ 8.36 $\times 10^{-4}$.
With .95 confidence interval of [7.8 $\times 10^{-4}$, 8.9 $\times 10^{-4}$]; still of good chance within 8.57 $\times 10^{-4}$ spec.

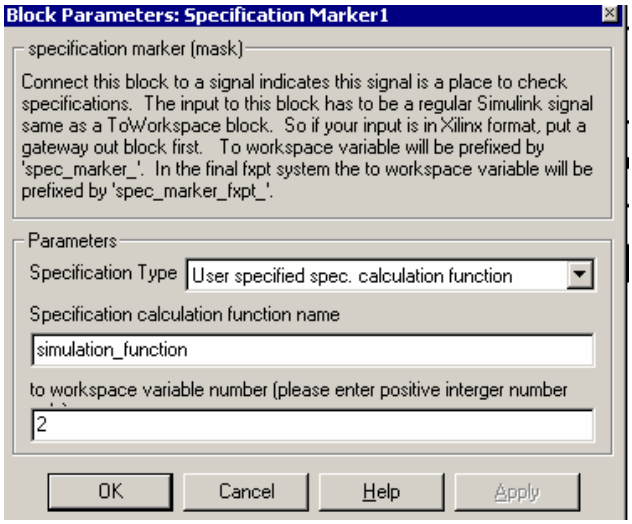


Fig. C-14 New specification Marker parameters

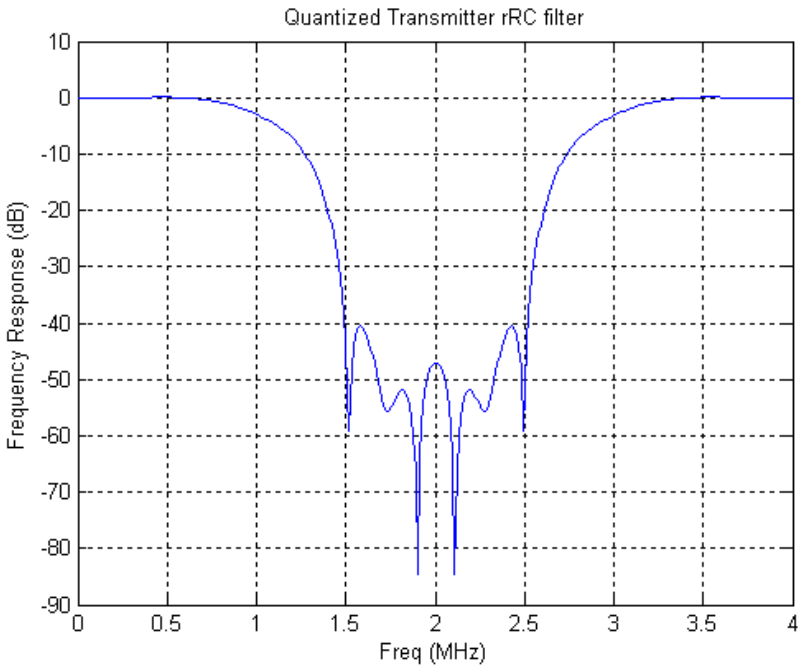


Fig. C-15 Tx rRC filter frequency response satisfies the -40 dB spec.

The conversion takes about 20 minutes, which is considerably more than the case if there is MSE spec only. This is in general the situation. The speed-up for MSE spec results from the careful analysis of the relationship between the MSE and wordlengths in Chapter 2 and 3.

C.8 Some further Analyses

Rules [1 2.1] results eight groups of fractional wordlengths. Let's see what if we apply more or less rules.

If rules [1.1 2.1] are chosen, only 4 groups are resulted (notice from help_fcc that rule 1.1 supercedes rule 1) and the conversion only takes about 2 minutes. The system takes 493 FPGA slices with BER ~ 7.95×10^{-4}, which is about 30% increase over the

design in Section C-7. The performance is compatible. This proves that more rules will speed up the conversion, but introduce loss of optimality.

However that does not mean we should include no rule or grouping in our design. In fact if we chose [1 2] as the rules, 30 groups are resulted, and conversion takes about 20 minutes to finish. The resulting system has 377 slices and BER ~8.36×10^{-4}. This system actually consumes about 6% more resource than the one in section 7! The reason behind it is when groups are so small, the modeling of their analytical hardware-function and MSE-function suffers high error. The error causes uncertainties in optimal decision. Of course it should be emphasized that small 6% difference almost gives the conclusion that having 30 groups won't improve the fxpt conversion much than having 8 groups, at least for this system.

Similar simulation is done for multi-criteria FFC case. Choosing [1.1 2.1] as the rule a system of 650 slices with BER ~ 8.2×10^{-4}. So there is a 60% reduction in hardware cost by applying rules [1 2.1] instead of [1.1 2.1]. The gain here is conversion time, only 5 minutes as opposed to previous 20 minutes.

All the experiments so far use rule 8 that sets "saturation" as overflow mode everywhere. However, since the integer wordlengths are chosen well, it is in general too conservative to use rule 8. An alternative is to use rule 8.1, which sets "wrap-around" as the overflow mode everywhere. The resulting two systems take 256 slices and BER ~ 8.11×10^{-4} (rule [1 2.1 8.1]), and 287 slices and BER ~ 8.36×10^{-4} (rule [1 2.1 8.1]), respectively. Comparing with the previous conversions that uses rule 8, the new results

save about 25% slices! The simulation result are however exactly the same. So these two results should be our final designs.

Another subject mentioned in Section C-7 and Chapter 4 is robust programming. A robust MSE spec was chosen there. Let's see what if MSE is chosen to be 0.001 directly. It gives 338 slices and BER ~ 8.46×10^{-4}—about 7% reduction in hardware resources causes the .95 confidence interval of BER to be [0.00079, 0.00090]. The BER becomes much more likely to be greater than 0.000857 (recall this is 10% more than that of the floating-point system). The small hardware reduction is usually not worth the risk of breaking the specification. The other possible situation is the specification is also flexible in the first place, in which case having a constraint on the spec is in some sense robust programming itself. The message here is to be careful on choosing the MSE spec level: you should understand there could be both under-modeling error and estimation error associated with the specification function; therefore it is usually wise to be a little conservative. After all, this example showed 3dB "relaxation" in MSE only causes a variation on hardware about 7%. This is almost always the situation, due to the function characteristics of objective and constraint functions, being quadratic and exponential , respectively.

C.9 An important remark

The most important remark here is that in our design procedure above, we only did qualitative justification on choosing the algorithms (say data modulation scheme, upsampling rate R, filter type, etc.) and architectures (say filter specification, filter form). To have a good design these "parameters" need to be justified using careful analysis or

simulations. Nevertheless the purpose of this tutorial is to get you familiar with the design process, and mainly on using FFC tool. So these design dimensions have not been explored fully here.

On the other hand higher-level decision (such as algorithm) made without considering the lower-level discrepancies could turn out to be unfavorable when the lower-level design (such as choosing circuit) space is explored. For example, we decided the number of filter taps to be the smallest one satisfying the 40dB attenuation requirement. This was chosen since it saves hardware and results less latency. However it's fairly possible that with fixed-point data types, too high word lengths are needed to maintain the 40dB attenuation because there is not much room left for WL reduction. By relaxing the number of taps to a few more, one might dramatically drop the number of bits needed for each tap; therefore save total hardware cost.

So ideally algorithm, architecture, and fixed-point datatypes should be optimized jointly, maybe with other design variables such as circuit level flexibility, in order to get the truly "best" design. The bad news is a problem like this could easily become too hard to solve. That's exactly the reason design of a large system is almost always divided into different levels, and different blocks. One always tries to reduce the inter-dependency between these levels and blocks to make each of the smaller problems more tractable. Our introduction of MSE specification as a global justification on FFC problem is based on this argument.

Of course one needs to bear in mind that quite often by considering the inter-dependency more carefully one can achieve large improvements. Examples include

Trellis-coding (coding and modulation jointly considered), our approach on FFC problem in some sense (different WLs jointly considered), channel coding (where algorithm is directly done in number theory, which is already fixed-point), etc. But this interesting trade-off is beyond the scope of this tutorial.

C.10 Conclusion

By building a simple base-band digital communication system, we showed a design procedure, starting from algorithm to fixed-point implementation, in our design environment combined with Matlab, Simulink, Xilinx System Generator. One major topic is on how to use our floating-point to fixed-point conversion tool. Table C-1 summarizes the conversions done in this tutorial.

	Grouping Rules	MSE spec level	Max(|H(1.5MHz, 2MHz)|)	Conversion time (minutes)	# of Frac. WL groups	Slices	BER (ML estimate)
Flpt sys	-	-	<-40dB	-	-	~13500 (or -)	~0.00078
	[1 2.1 8]	0.0005	-	5	8	~356	~0.00081
	[1 2 8]	0.0005	-	20	30	~377	~0.000836
	[1.1 2.1 8]	0.0005	-	2	4	~493	~0.000795
*	[1 2.1 8.1]	0.0005	-	5	8	~265	~0.00081
	[1 2.1 8]	0.001	-	5	8	~338	~0.000846
	[1.1 2.1 8]	0.0005	<-40dB	5	4	~650	~0.00082
	[1 2.1 8]	0.0005	<-40dB	20	8	~386	~0.000836
*	[1 2.1 8.1]	0.0005	<-40dB	20	8	~287	~0.000836

Table C-1 Summary of conversions in this tutorial.
Targeted BER for converted systems is 0.000857. "–" means not applicable. "*" means the design is considered to have good performance while relatively less conversion time.

Appendix D. Related Publications

[D.1] C. Shi, and R. W. Brodersen, "Floating-point to fixed-point conversion," To be published, *IEEE Trans. Signal Processing*. 2004.

[D.2] C. Shi, and R. W. Brodersen, "An automated floating-point to fixed-point conversion methodology," *Proc. IEEE Int. Conf. on Acoust., Speech, and Signal Processing*, Vol. 2, pp. 529-532, April 2003.

[D.3] C. Shi, "Statistical method for floating-point to fixed-point conversion," 2002, Master Thesis, Department of EECS, Univ. of California, Berkeley. (Advisor: Robert W. Brodersen).

[D.4] C. Shi, and R. W. Brodersen, "Floating-point to fixed-point conversion with decision errors due to quantization," *Proc. IEEE Int. Conf. on Acoust., Speech, and Signal Processing*, 2004, Canada.

[D.5] C. Shi, and R. W. Brodersen, "A perturbation theory on statistical quantization effects in fixed-point DSP with non-stationary input," *Proc. IEEE Int. Sym. Circs. and Sys.*, 2004, Canada.

[D.6] C. Shi, R. W. Brodersen, "Automated Fixed-point Data-type Optimization Tool for Signal Processing and Communication Systems," *Design Automation Conference*, San Diego, June 2004.

[D.7] C. Shi, *et. al.*, "An Automated Pre-netlisting FPGA-Resource Estimation Tool," Submitted to *International Conference, Field Programmable Logics and Its Applications*, 2004

[D.8] C. Shi, and R. W. Brodersen, "A perturbation theory on quantization effects in digital signal processing," In preparation. *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*.

[D.9] C. Shi, FFC website with all the source codes and related documents. Available[online] http://bwrc.eecs.berkeley.edu/people/grad_student/ccshi/research/

Reference

[1] C. Shi, and R. W. Brodersen, "Floating-point to fixed-point conversion," To be published, *IEEE Trans. Signal Processing*. 2004

[2] C. Shi, and R. W. Brodersen, "An automated floating-point to fixed-point conversion methodology," *Proc. IEEE Int. Conf. on Acoust., Speech, and Signal Processing*, Vol. 2, pp. 529-532, April 2003.

[3] C. Shi, "Statistical method for floating-point to fixed-point conversion," 2002, Master Thesis, Department of EECS, Univ. of California, Berkeley. (Advisor: Robert W. Brodersen).

[4] A. V. Oppenheim, and R. W. Schafer, with J. R. Buck. *Discrete-Time Signal Processing.* 2nd ed., Prentice Hall, 1999, ch. 6.

[5] L. B. Jackson. *Digital filters and signal processing: with MATLAB exercises*, 3rd ed. Boston : Kluwer Academic Publishers, 1996

[6] S. S. Haykin. *Adaptive filter theory*. 3rd Edition. Prentice Hall, 1996.

[7] R. M. Gray, and D. L. Neuhoff, "Quantization," *IEEE Trans. Inform. Theory*, vol. 44, No. 6, pp. 2325-2383, Oct. 1998.

[8] D. A. Patterson, and J. L. Hennessy, *Computer Organization & Design—the Hardware/software interface*, 2nd ed., Morgan Kaufmann, 1998, ch. 4.

[9] C. Fang, T. Chen, and R. A. Rutenbar, “Floating-point error analysis based on affine arithmetic,” *Proc. IEEE Int. Conf. on Acoust., Speech, and Signal Processing.*, vol. 2, pp. 561-564, Apr. 2003.

[10] J. H. McClellan, et al. Computer-based Exercises for Signal Processing using Matlab. Prentice Hall, 1998.

[11] P. H. Bauer, and L. Leclerc, “A computer-aided test for the absence of limit cycles in fixed-point digital filters,” *IEEE Trans. Signal Processing*, vol. 39, pp. 2400-2410, Nov. 1991.

[12] K. Chang, and W. G. Bliss, “Limit cycle behavior of pipelined recursive digital filters,” *IEEE Trans. Circuits Syst.—II: Analog and Digital Signal Processing*, vol. 41, pp. 351-355, May 1994.

[13] H. Keding, M. Willems, M. Coors, and H. Meyr, “FRIDGE: a fixed-point design and simulation environment,” *Proceedings of Design, Automation and Test in Europe*, pp. 429-435, 1998.

[14] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens, “A methodology and design environment for DSP ASIC fixed point refinement”, *Design, Automation and Test in Europe Conference and Exhibition 1999. Proceedings*, pp. 271 –276, 1999.

[15] S. Kim, K. Kum and W. Sung, “Fixed-point optimization utility for *C* and *C++* based digital signal processing programs,” *IEEE Trans. On Circuits Syst. II: Analog and Digital Signal Processing*, vol. 45, pp. 1455-1464, 1998.

[16] D. Menard, and O. Sentieys, “A methodology for evaluating the precision of fixed-point systems,” *IEEE Int. Conf. on Acoust., Speech, and Signal Process.*, vol. 3, pp. 3152-3155, 2002.

[17] M. A. Cantin, Y. Savaria, and P. Lavoie, “A comparison of automatic word length optimization procedures,” *IEEE Int. Sym. Circuits Syst., 2002*, vol. 2, pp. 612 - 615.

[18] X. Hu, S. C. Bass, “A neglected error source in the CORDIC algorithm,” *IEEE Int. Sym. on Circuits Syst.*, vol. 1, pp. 766 -769, May 1993.

[19] P. W. Wong, “Quantization and roundoff noises in fixed-point FIR digital filters,” *IEEE Trans. Signal Processing*, vol. 39, pp. 1552-1563, July 1991.

[20] R. M. Gray, “Quantization noise spectra,” *IEEE Trans. Inform. Theory*, vol. 36, pp. 1220-1244, Nov. 1990.

[21] S. R. Parker, and P. E. Girard, “Correlated noise due to roundoff in fixed point digital filters,” *IEEE Trans. Circuits Syst.*, vol. cas-23, pp. 204-211, Apr. 1976.

[22] I. Tokaji, C. W. Barnes, “Roundoff error statistics for a continuous range of multiplier coefficients,” *IEEE Trans. Circuits Syst.*, vol. cas-34, pp. 52-59, Jan. 1987.

[23] A. B. Sripad and D. L. Snyder, “A necessary and sufficient condition for quantization errors to be uniform and white,” *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-25, pp. 442-448, Oct. 1977.

[24] C. Barnes, B. N. Tran, and S. H. Leung, “On the statistics of fixed-point roundoff error,” *IEEE Trans. Acoust., Speech, and Signal Processing*, vol. asp-33, pp. 595-606, June 1985.

[25] J. C. M. Bermudez, and N. J. Bershad, "A nonlinear analytical model for the quantized LMS algorithm-the arbitrary step size case," *IEEE Trans. Signal Processing*, vol. 44, pp. 1175 -1183, May 1996.

[26] N. J. Bershad, and J. C. M. Bermudez, "A nonlinear analytical model for the quantized LMS algorithm-the power-of-two step size case," *IEEE Trans. Signal Processing*, vol. 44, pp. 2895-2900, Nov. 1996.

[27] M. Leban, and J. Tasic, "A fixed-point quantization model in the statistical analysis of adaptive filters," 1998.

[28] N. J. Bershad, "Nonlinear quantization effects in the LMS and block LMS adaptive algorithms-a comparison," *IEEE Trans. Acoust. Speech, and Signal Processing*, vol. 37, pp. 1540-1512, Oct. 1989.

[29] C. Caraiscos, and B. Liu, "A roundoff error analysis of the LMS adaptive algorithm," *IEEE Trans. Acoust. Speech, and Signal Processing*, vol. 32, pp. 34-41, Feb 1984.

[30] P. S. Chang, and A. N. Willson, Jr., "A roundoff error analysis of the normalized LMS algorithm," *Record 29th Asilomar Conf. Signals, Systems, and Computers*, vol. 2, pp. 1337-1341, 1995.

[31] J. M. Cioffi, "Limited-precision effects in adaptive filtering," *IEEE Trans. Circuits Syst.*, vol. cas-34, pp. 871-883, July 1987.

[32] J. M. Cioffi, "A finite precision analysis of the block-gradient adaptive data-driven echo canceller," *IEEE Trans. Comm.*, vol. 40, May 1992.

[33] M. L. R. de Campos, P. S. R. Diniz, and A. Antoniou, "A finite wordlength analysis of an LMS-Newton adaptive filtering algorithm," *IEEE Int. Sym. Circuits and Syst.*, vol. 1, pp. 870-873, May 1993.

[34] P. S. R. Diniz, M. L. R. de Campos, and A. Antoniou, "Analysis of LMS-Newton adaptive filtering algorithms with variable convergence factor," *IEEE Trans. Signal Processing*, vol. 43, pp. 617-627, Mar. 1995.

[35] S. Gazor, and B. Farhang-Boroujeny, "Quantization effects in transform- domain normalized LMS algorithm," *IEEE Trans. Circuits and Syst. II: Analog and Digital Signal Processing*, vol. 39, pp. 1-7, Jan. 1992

[36] R. Gupta, and A. O. Hero, III, "Power versus performance tradeoffs for reduced resolution LMS adaptive filters," *IEEE Trans. Signal Processing*, vol. 48, pp. 2772-2784, Oct. 2000.

[37] R. Seara, J. C. M. Bermudez, W. P. Carpes, Jr., "An improved quantization model for the finite precision LMS adaptive algorithm," *IEEE Int. Sym. Circuits Syst.*, pp. 858-861, May 1993.

[38] D. Sherwood, and N. Bershad, "Nonlinear quantization effects in the frequency domain complex scalar LMS adaptive algorithm," *IEEE Trans. Acoust., Speech, and Signal Processing*, vol. 34, pp. 140-151, Feb. 1986.

[39] D. Sherwood, and N. Bershad, "Quantization effects in the complex LMS adaptive algorithm: Linearization using dither-theory," *IEEE Trans. Circuits and Syst.*, vol. 34, pp. 848-854, Jul. 1987.

[40] N. R. Yousef, A. H. Sayed, "A unified approach to the steady-state and tracking analyses of adaptive filters," *IEEE Trans. Signal Processing*, vol. 49, pp. 314-324, Feb. 2001.

[41] W. Sethares, D. Lawrence, C. Johnson, Jr., and R. Bitmead, "Parameter drift in LMS adaptive filters," *IEEE Trans. Acoust., Speech, and Signal Processing*, vol. 34, pp. 868-879, Aug. 1986.

[42] S. Y. Park, and N. I. Cho, "Fixed point error analysis of CORDIC processor based on the variance propagation," *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, vol. 2, pp. 565-568, Apr. 2003

[43] B. Zeng, and L. Gu, "Roundoff noise analysis of paraunitary filter banks realized in lattice structure," *Proc. IEEE Digital signal Processing Workshop*, pp. 93-96, Sept. 1996.

[44] C. A. Rabbath, and N. Hori, "On the implementation of filters subjected to quantization of coefficients," *Proc. Int. Conf. Digital Signal Processing*, vol. 2, pp. 665-670. July 1997.

[45] A. Krukowski, R. C. S. Morling, L. Kale, "Quantization effects in the polyphase N-path IIR structure," *IEEE Trans. Instrumentation and Measurement*, vol. 51, pp. 1271-1278, Dec. 2002.

[46] J. Proakis, *Digital Communications*, 4th ed. McGraw-Hill, 2000, ch. 1-3.

[47] V. Mathews, and S. Cho, "Improved convergence analysis of stochastic gradient adaptive filters using the sign algorithm," *IEEE Trans. Acoust., Speech, and Signal Processing*, vol. 35, pp.450-454, Apr. 1987.

[48] P. J. Bickel and K. A. Doksum, *Mathematical statistics: basic ideas and selected topics.* 2nd Edition, Prentice Hall, 2001.

[49] L. De Coster, M. Ade, R. Lauwereins, and J. Peperstraete, "Code generation for compiled bit-true simulation of DSP applications," *Proceedings of 11th Int. Sym. on system synthesis*, pp. 9 –14, 1998.

[50] K. Kuusilinna, *et al*, "Real-time System-on-Chip Emulation," Chapter 10, *Winning the SoC Revolution*, Kluwer Academic Publishers, pp. 229-253, 2003.

[51] C. Shi, and R. W. Brodersen, "Floating-point to fixed-point conversion with decision errors due to quantization," *Proc. IEEE Int. Conf. on Acoust., Speech, and Signal Processing*, 2004, Canada.

[52] C. Shi, and R. W. Brodersen, "A perturbation theory on statistical quantization effects in fixed-point DSP with non-stationary input," *Proc. IEEE Int. Sym. Circs. and Sys.*, 2004, Canada.

[53] C. Shi, R. W. Brodersen, "Automated Fixed-point Data-type Optimization Tool for Signal Processing and Communication Systems," *Design Automation Conference*, San Diego, June 2004.

[54] G. E. P. Box, "Simpling and Bayes inference in scientific modeling and robustness (with discussion)," *J. Royal Statist.* Soc. A 143, pp. 383-430, 1979.

[55] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, 2nd Edition, Cambridge university press, 1996, pp. 28-31.

[56] W. R. Davis, *et al*, "A design environment for high-throughput low-power dedicated signal processing systems," *IEEE Journal of Solid State Circuits*, vol. 37, No. 3, pp. 420-431, Mar. 2002.

[57] Haiyun Tang, “A Unified Approach to Wireless System Design,” PhD thesis, University of California at Berkeley, 2003. Advised by Professor Robert W. Brodersen

[58] Ning Zhang, “Algorithm/Architecture Co-Design for Wireless Communication Systems,” PhD thesis, University of California at Berkeley, 2001. Advised by Professor Robert W. Brodersen

[59] The MathWorks, Inc. Simulink. [Online]. Available: http://www.mathworks.com.

[60] The Xilinx, Inc. System Generator. [Online]. Available: http://www.xilinx.com . Once at above website, search on “System Generator” and “System Generator Resource Estimation” for related information.

[61] SystemC system-level co-design language. [Online] Website available: http://www.systemc.org.

[62] AccelChip Inc., [online] Website available: http://www.accelchip.com/

[63] S. Boyd, and L. Vandenberghe. *Convex optimization*. [Online]. Available: http://www.stanford.edu/~boyd/cvxbook.html.

[64] C. Shi, *et. al.*, “An Automated Pre-netlisting FPGA-Resource Estimation Tool,” Submitted to *International Conference, Field Programmable Logics and Its Applications*, 2004

[65] W. Sung, and K. Kum, “Simulation-based word-length optimization method for fixed-point digital signal processing systems,” *IEEE Trans. Signal Processing*, vol. 43, no. 12, Dec. 1995.

[66] N. Zhang, B. Haller, and R. W. Brodersen, "Systematic architecture exploration for implementing interference suppression techniques in wireless receivers," *Proc. IEEE Workshop on Signal Processing Systems*, LA, October 2000.

[67] M. Nemani, and F. N. Najm, "High-level area and power estimation for VLSI circuits," *IEEE Tran. Computer-Aided Design of Integrated Circuits and Systems*, vol 18, pp. 697-713, June 1999.

[68] Mosek optimization toolbox. [Online]. Available: http://www.mosek.com.

[69] C. Shi, and R. W. Brodersen, "A perturbation theory on quantization effects in digital signal processing," In preparation. *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*.

[70] D. Markovic, R. W. Brodersen, "MIMO SVD Based Algorithm Implementation", 2002 BWRC Winter Retreat, In publication list of MCMA group webpage at http://bwrc.eecs.berkeley.edu/Research/MCMA

[71] Mike Shuo-Wei Chen, "Ultra Wide-band Baseband Design and Implementation", M.S. Thesis, EECS Department, University of California, Berkeley. 2002. (advisor: Robert W. Brodersen).

[72] J. B. Knowles and E.M. Olcayto, "Coefficient accuracy and digital filter response," *IEEE Trans. Circuit Theory*, vol. CT-15, Mar. 1968, pp. 31-41.

[73] Dietrich Schlichthärle, *Digital filters: basics and design*,. Berlin, New York: Springer, 2000.

[74] K. K. Parhi, *VLSI Digital Signal Processing Systems*. John Wiley & Sons, INC. 1999.

[75] K. Chang, W. G. Bliss, "Finite word-length effects of pipelined recursive digital filters," *IEEE Transactions on Signal Processing*, Vol. 42, Aug. 1994 pp 1983 – 1995

[76] J. Ma, K. K. Parhi, E. F. Deprettere, "Pipelined CORDIC-based cascade orthogonal IIR digital filters," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, Vol. 47, Nov. 2000, pp 1238 –1253.

[77] R. K. Brayton, C. H. Tong, "Constructive Stability and asymptotic Stability of Dynamical Systems" *IEEE Trans. CAS-26*, pp 1121-1130, 1980.

[78] K. Premaratne, E. C. Kulasekere, P. H. Bauer, L. J. Leclerc, "An exhaustive search algorithm for checking limit cycle behavior of digital filters**,"** *IEEE International Symposium on Circuits and Systems*, Vol. 3, 1995, pp 2035 -2038.

[79] CoCentric System from Synopsys Inc., [online] website available http://www.synopsys.com, search for CoCentric.

[80] E. Eweda, N. Yousef, S. El-Ramly, "Reducing the effect of finite wordlength on the performance of an LMS adaptive filter**,"** *IEEE International Conference on Communications*, Vol.2, 1998 pp 688 -692.

[81] McLernon, D.C. "Finite wordlength effects in two-dimensional multirate periodically time-varying filters**,"** *IEE Proceedings Circuits, Devices and System*, Vol. 144, 1997, pp 277 –283.

[82] Al-Dhahir, N. **"**On finite word length effects for the FIR MMSE-DFE,**"** *IEEE Communications Letters*, Vol. 2, Aug. 1998, pp 238 –240.

[83] Yasukawa, K.; Milstein, L.B., "Finite word length effects on the performance of MMSE receiver for DS-CDMA systems," *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, Vol. 2, 1997, pp 724 -728.

[84] Song, M.S.; Yang, P.P.N.; Shenoi, K. "Nonlinear compensation for finite word length effects of an LMS echo canceller algorithm suitable for VLSI implementation," *ICASSP, 1988*, pp. 1487 -1490, vol.3.

[85] Sanjit K. Mitra. *Digital signal processing: a computer-based approach.* 2nd ed. Boston : McGraw-Hill/Irwin, 2001

[86] M. Chen, C, Shi. "EE290s Project Report – Adaptive receiver for UWB data Recovery," Fall 2001, EECS, UC. Berkeley.

[87] R. Price, "A Useful Theorem for Nonlinear Devices Using Gaussian Inputs," *IEEE Trans. Information Theory*, 1958, pp 69-72.

[88] C. Shi, FFC website with all the source codes and related documents. Available [online] http://bwrc.eecs.berkeley.edu/people/grad_student/ccshi/research/ .

[89] C. Chang, *et. al.*, "Rapid Design and analysis of communication systems using the BEE hardware emulation environment," *RSP*, 2003.

[90] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee, "Accurate Area and Delay Estimators for FPGAs," *Proc. Design Automation and Test in Europe*, Mar. 2002, Paris, France.

[91] D. B. Parlour, "The reality and promise of reconfigurable computing in digital signal processing," Tutorial of *ISSCC*, Feb 15-19, 2004

[92] R. Jain, C. Chien, E. Cohen, and L. Ho, "Simulation and synthesis of VLSI communication systems," *Proc. Int. Conf. VLSI Design* 1998, pp 336-341.

[93] R. Jain, J. Vandewalle, and H. DeMan, “Efficient and accurate multiparameter analysis of linear digital filters using a multivariable feedback representation,” *IEEE Trans. Circuits and Systems*, Vol. 32, No.3, 1985, pp 225-235.

[94] R. Jain, P. Yang, T. Yoshino, “FIRGEN: a computer-aided design system for high performance FIR filter integrated circuits,” *IEEE Trans. Signal Processing*. Vol. 39, No. 7. 1991, pp 1665-1678.

[95] SPW from CoWare Inc., [online] Website available http://www.coware.com

[96] Ptolemy project, [online] Website available http://ptolemy.eecs.berkeley.edu.