CHAPTER 23

PRECISION ANALYSIS FOR FIXED-POINT COMPUTATION

George A. Constantinides
Department of Electrical and Electronic Engineering
Imperial College, London

Many values in a computation are naturally represented by integers, which have very efficient hardware implementations; basic operations are relatively cheap, and they map well to an FPGA's underlying hardware. However, some computations naturally result in fractional values, that is, numbers where part or all of the value are less than 1—for example, 0.25, 3.25, and π—or that are so large that representation as integers is too costly—for example, 10^{120}. Handling these values is a significant concern because the hardware necessary to compute on *scaled* values can be significant in speed, power consumption, and area.

In arithmetic for reconfigurable computing designs, it is common to employ fixed point instead of floating point to represent scaled values. This chapter explores the reason for this design decision and the associated analysis that must be performed in order to choose an appropriate fixed-point representation for a particular design. Since designs for reconfigurable logic can be customized for particular applications, it is appropriate to fit the number system to the underlying application properties.

23.1 FIXED-POINT NUMBER SYSTEM

In general-purpose computing, floating-point representations are most commonly used for the representation of numbers containing fractional components. The floating-point representations standardized by the IEEE [22] have several advantages, the foremost being portability across different computational platforms.

In general, we may consider a floating-point number $X[t]$ at time t as made up of two components: a signed mantissa $M[t]$ and a signed exponent $E[t]$ (see equation 23.1). Within this representation, the ratio of the largest positive value of X to the smallest positive value of X varies exponentially with the exponent $E[t]$ and hence doubly exponentially with the number of bits used to store the exponent. As a result, it is possible to store a wide dynamic range with only a few bits of exponent, while the mantissa maintains the precision of the

representation across that range by dividing the corresponding interval for each exponent into equally spaced representable values.

$$X[t] = M[t] \cdot 2^{E[t]} \tag{23.1}$$

However, the flexibility of the floating-point number system comes at a price. Addition or subtraction of two floating-point numbers requires the alignment of radix ("decimal") points, typically resulting in a large, slow, and power-hungry barrel shifter. In a general-purpose computer, this is a minor concern compared to the need to easily support a wide range of applications. This is why processors designed for general-purpose computing typically have a built-in floating-point unit.

In embedded applications, where power consumption and silicon area are of significant concern, the fixed-point alternative is more often used [24]. We can consider fixed point as a degenerate case of floating point, where the exponent is fixed and cannot vary with time (i.e., $E[t] = E$). The fixing of the exponent eliminates the need for a variable alignment and thus the need for a barrel shifter in addition and subtraction. In fact, basic mathematical operations on fixed-point values are essentially identical to those on integer values. However, compared to floating point, the dynamic range of the representation is reduced because the range of representable values varies only singly exponentially with the number of bits used to represent the mantissa.

When implementing arithmetic in reconfigurable logic, the fixed-point number system becomes even more attractive. If a low-area fixed-point implementation can be achieved, space on the device can be freed for other logic. Moreover, the absence of hardware support for barrel shifters in current-generation reconfigurable logic devices results in an even higher area and power overhead compared to that in fully custom or ASIC technologies.

23.1.1 Multiple-wordlength Paradigm

For simplicity we will restrict ourselves to 2's complement representations, although the techniques presented in this chapter apply similarly to most other common representations. Also, we will use dataflow graphs, also known as signal flow graphs in the digital signal processing (DSP) community, as a simple underlying model of computation [12]. In a dataflow graph, each atomic computation is represented by a vertex $v \in V$, and dataflow between these nodes is represented by a set of directed edges $S \subseteq V \times V$. To be consistent with the terminology used in the signal-processing community, we will refer to an element of S as a *signal*; the terms *signal* and *variable* are used interchangeably.

The multiple-wordlength paradigm is a design approach that tries to fit the precision of each part of a datapath to the precision requirements of the algorithm [8]. It can be best introduced by comparison to more traditional fixed-point and floating-point implementations. Each 2's complement signal $j \in S$ in a multiple-wordlength implementation of a dataflow graph (V, S) has two parameters n_j and p_j, as illustrated in Figure 23.1(a). The parameter n_j represents the

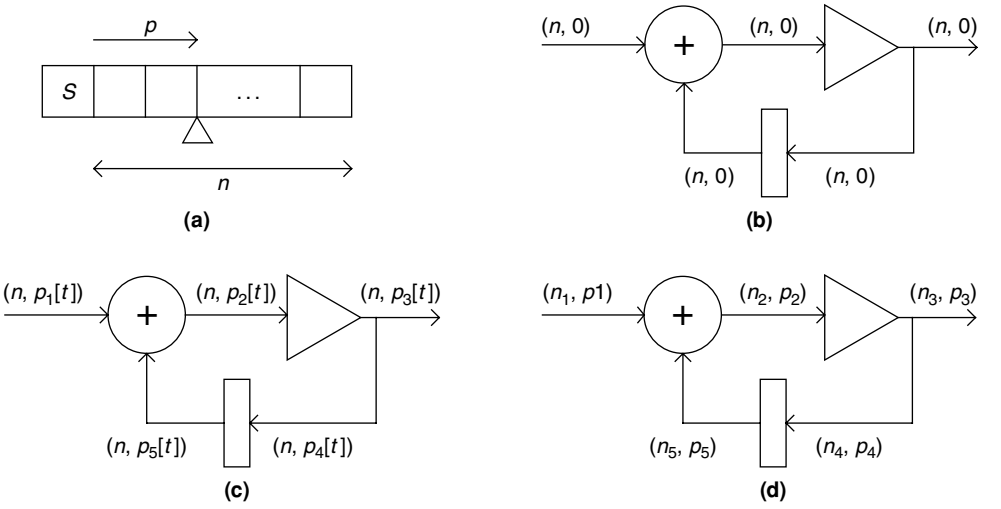

FIGURE 23.1 ■ The multiple-wordlength paradigm: (a) signal parameters ("s" indicates a sign bit); (b) fixed point; (c) floating point; (d) multiple wordlength. The triangle represents a constant coefficient multiplication, or "gain"; the rectangle represents a register, or unit sample delay.

number of bits in the representation of the signal (excluding the sign bit, by convention), and the parameter p_j represents the displacement of the binary point from the least significant bit (LSB) side of the sign bit toward the LSB. Note that there are no restrictions on p_j; the binary point could lie outside the number representation (i.e., $p_j < 0$ or $p_j > n_j$).

A simple fixed-point implementation is illustrated in Figure 23.1(b). Each signal j in this dataflow graph representing a recursive DSP algorithm is annotated with a tuple (n_j, p_j) representing the wordlength scaling of the signal. In this implementation, all signals have the same wordlength and scaling, although shift operations are often incorporated in fixed-point designs in order to provide an element of scaling control [25]. Figure 23.1(c) shows a standard floating-point implementation, where the scaling of each signal is a function of time.

A single systemwide wordlength is common to both fixed and floating point. This is a result of historical implementation on single, or multiple, predesigned arithmetic units. In FPGAs the situation is quite different. Different operations are generally computed in different hardware resources, and each of these computations can be built to any size desired. Such freedom points to an alternative implementation style, shown in Figure 23.1(d). This multiple-wordlength implementation style inherits the speed, area, and power advantages of traditional fixed-point implementations, since the computation is fixed point with respect to each individual computational unit. However, by potentially allowing each signal in the original specification to be encoded by binary words with different scaling and wordlength, the degrees of freedom in design are significantly increased.

23.1.2 Optimization for Multiple Wordlength

Now that we have established the possibility of using multiple scalings and wordlengths for different variables, two questions arise: How can we optimize the scalings and wordlengths in a design to match the computation being performed, and what are the potential benefits from doing so? For FPGA-based implementation, the benefits have been shown to be significant: Area savings of up to 45 percent [8] and 80 percent [15] have been reported compared to the use of a single wordlength across the entire circuit. The main substance of this chapter is to describe suitable scaling and wordlength optimization procedures to achieve such savings.

Section 23.2 shows that we can determine the appropriate scaling for a signal from an estimation of its peak value over time. One of two main techniques—simulation based and analytical—is then introduced to perform this peak estimation. While an analytical approach provides a tight bound on the peak signal value, it is limited to computations exhibiting certain mathematical properties. For computations outside this class, an analytical technique tends to be pessimistic, and so simulation-based methods are commonly used.

Section 23.3 focuses on determining the wordlength for each signal in the computation. The fundamental issue is that, because of roundoff or truncation, the wordlength of different signals in the system can have different impacts on both the implementation area and the error observed at the computation output. Thus, any wordlength optimization system needs to perform a balancing act between these two factors when allocating wordlength to signals. The goal of the work presented in this section is to allocate wordlength so as to minimize the area of the resulting circuit while maintaining an acceptable computational accuracy at the output of the circuit.

23.2 PEAK VALUE ESTIMATION

The physical representation of an intermediate result in a bit-parallel implementation of an algorithm consists of a finite set of bits, usually encoded using 2's complement representation. To make efficient use of the resources, it is essential to select an appropriate *scaling* for each signal. Such a scaling should ensure that the representation is not overly wasteful in catering to rare or impossibly large values and that overflow errors, which lead to low arithmetic quality, do not occur often.

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

For some algorithms, it is possible to estimate the peak value that each signal could reach using analytic means. In the next section, such techniques for two different classes of system are discussed. The alternative, to use simulation to determine the peak signal value, is described in the following section.

Also discussed are some hybrid techniques that aim to combine the advantages of both approaches.

23.2.1 Analytic Peak Estimation

If the DSP algorithm under consideration is a linear time-invariant system, it is possible to find a tight analytic bound on the peak value reachable by every signal in it. This is the problem addressed in the section immediately following. If, on the other hand, the system is nonlinear or time varying, such an approach cannot be used. If the algorithm is nonrecursive—that is, the dataflow graph does not contain any feedback loops—data range propagation may be used to determine an analytic bound on the peak value of each signal. However, this approach, described in the next section, cannot be guaranteed to produce a tight bound.

Linear time-invariant systems

A linear time-invariant (LTI) system is one that obeys the distinct properties of linearity and time invariance. A *linear system* is one that obeys superposition—that is, if its output is the sequence $y_1[t]$ in response to input $x_1[t]$, and is $y_2[t]$ in response to input $x_2[t]$, then it will be $\alpha y_1[t] + \beta y_2[t]$ in response to input $\alpha x_1[t] + \beta x_2[t]$. A *time-invariant* system is one that, given the input $x[t]$ and the corresponding output $y[t]$, will provide output $y[t - t_0]$ a given input $x[t - t_0]$. In other words, shifting the input sequence in time merely shifts the output sequence by the same amount.

From a practical perspective, any computation made entirely of addition, constant coefficient multiplication, and delay operations is guaranteed to be LTI. This class of algorithms, while restricted, is extremely important; it contains all the fundamental building blocks of DSP, such as finite impulse response (FIR) and infinite impulse response (IIR) filters, together with transformations such as the discrete cosine transform (DCT), the fast Fourier transform (FFT), and many color–space conversions.

The remainder of this section assumes a basic knowledge of digital signal processing, in particular the z-transform and transfer functions. For the unfamiliar reader, Mitra [32] provides an excellent introduction. Readers unconcerned with the mechanics of peak estimation for LTI systems may simply take it as read that for such systems it is possible to obtain tight analytic bounds on peak signal values.

Transfer function calculation The analytical scaling rules derived in this section rely on a knowledge of system transfer functions. A transfer function of a discrete-time LTI system between any given I/O pair is defined to be the z-transform of the sequence produced at that output, in response to a unit impulse at that input [32]; these transfer functions may be expressed as the ratio of two polynomials in z^{-1}. The transfer function from each primary input to each signal must be calculated for signal-scaling purposes. This section considers the practical problem of transfer function calculation from a dataflow graph.

Given a dataflow graph $G(V, S)$, let $V_I \subseteq V$ be the set of input nodes, $V_O \subseteq V$ be the set of output nodes, and $V_D \subseteq V$ be the set of unit sample delay nodes. For signal scaling, a matrix of transfer functions $\boldsymbol{H}(z)$ is required, with elements $h_{iv}(z)$ for $i \in V_I$ and $v \in V$ representing the transfer function from the primary input i to the output of node v.

Calculation of transfer functions for nonrecursive systems is a simple task, leading to a matrix of polynomials in z^{-1}; a straightforward algorithm is presented by Constantinides et al. [12]. For recursive systems, it is necessary to identify a subset $V_c \subseteq V$ of nodes whose outputs correspond to a system *state*. In this context, a state set consists of a set of nodes that, if removed from the dataflow graph, would break all feedback loops. Once such a state set has been identified, transfer functions can easily be expressed in terms of the outputs of these nodes using algorithms suitable for nonrecursive computations.

Let $\boldsymbol{S}(z)$ be a z-domain matrix representing the transfer function from each input signal to the output of each of these state nodes. The transfer functions from each input to each state node output may be expressed as in equation 23.2, where $\boldsymbol{A}$ and $\boldsymbol{B}$ are matrices of polynomials in z^{-1}. Each of these matrices represents a z-domain relationship once the feedback has been broken at the outputs of state nodes. $\boldsymbol{A}(z)$ represents the transfer functions between state nodes and state nodes, and $\boldsymbol{B}(z)$ represents the transfer functions between primary inputs and state nodes.

$$\boldsymbol{S}(z) = \boldsymbol{A}\boldsymbol{S}(z) + \boldsymbol{B}(z) \tag{23.2}$$

$$\boldsymbol{H}(z) = \boldsymbol{C}\boldsymbol{S}(z) + \boldsymbol{D}(z) \tag{23.3}$$

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

It is clear that $\boldsymbol{S}(z)$ may be expressed as a matrix of rational functions (equation 23.4), where $\boldsymbol{I}$ is the identity matrix of appropriate size. This allows the transfer function matrix $\boldsymbol{H}(z)$ to be calculated directly from equation 23.3.

$$\boldsymbol{S}(z) = (\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B} \tag{23.4}$$

Example Consider the simple dataflow graph from Section 23.1.1, shown in Figure 23.1. Clearly, removal of any one of the four internal nodes (adder, gain, delay, or the signal branch) from it will break the feedback loop. Let us arbitrarily choose the adder node as a state node and choose the gain coefficient to be 0.1. The polynomial matrices $\boldsymbol{A}(z)$ to $\boldsymbol{D}(z)$ may then be calculated (equation 23.5).

$$\begin{aligned}
\boldsymbol{A}(z) &= 0.1z^{-1} \\
\boldsymbol{B}(z) &= 1 \\
\boldsymbol{C}(z) &= [0\ 1\ 0.1\ 0.1\ 0.1\ 0.1z^{-1}]^T \\
\boldsymbol{D}(z) &= [1\ 0\ 0\ 0\ 0\ 0]^T
\end{aligned} \tag{23.5}$$

Calculation of $\boldsymbol{S}(z)$ may then proceed following equation 23.4, yielding equation 23.6. Finally, the matrix $\boldsymbol{H}(z)$ can be constructed following equation 23.3, giving equation 23.7.

$$\boldsymbol{S}(z) = 1/(1-0.1z^{-1}) \tag{23.6}$$

$$\begin{aligned}\boldsymbol{H}(z) = [1\ 1/(1-0.1z^{-1})\ 0.1/(1-0.1z^{-1})\ 0.1/(1-0.1z^{-1})\ 0.1/(1-0.1z^{-1})\\ 0.1z^{-1}/(1-0.1z^{-1})]^T\end{aligned} \tag{23.7}$$

The runtime of this algorithm grows significantly with the number of *state* signals $|V_c|$, and so selecting a small set of state signals is important. A simple approach is to select all of the delay elements in a circuit, assuming that it has no combinational cycles. Alternatively, techniques such as Levy and Low's [30] can be employed.

Scaling with transfer functions To produce the smallest fixed-point implementation, it is desirable to utilize as much as possible of the full dynamic range provided by each internal signal representation. The first step of the optimization process is therefore to choose the smallest possible value of p_j for each signal $j \in S$ in order to guarantee no overflow.

Consider a dataflow graph $G(V, S)$, annotated with wordlengths $\boldsymbol{n}$ and scalings $\boldsymbol{p}$. Recall that $V_I \subseteq V$ denotes the set of input nodes, and let us say that each such node reaches peak signal values of $\pm M_i (M_i > 0)$ for $i \in V_I$. Let $\boldsymbol{H}(z)$ be the scaling transfer function matrix defined before, with the associated impulse response matrix $h[t]$ related to the transfer function matrix through the component-wise inverse z-transform. Then the worst-case peak value P_j reached by any signal $j \in S$ is given by maximizing the well-known convolution sum (equation 23.8) [32], where $x_i[t]$ is the value of the input $i \in V_I$ at time index t.

Solving this maximization problem provides the input sequence given in equation 23.9, and allowing $N_{ij} \to \infty$ leads to the peak response at signal j given in equation 23.10. Here sgn( ) is the signum function (equation 23.11).

$$P_j = \pm \sum_{i \in V_I} \max_{x_i[t']} \left(\sum_{t=0}^{N_{ij}-1} x_i\left[t'-t\right] h_{ij}[t] \right) \tag{23.8}$$

$$x_i[t] = M_i \operatorname{sgn}\left(h_{ij}\left\lfloor N_{ij}-t-1\right\rfloor\right) \tag{23.9}$$

$$P_j = \sum_{i \in V_I} M_i \sum_{t=0}^{\infty} \left|h_{ij}[t]\right| \tag{23.10}$$

$$\operatorname{sgn}(x) = \begin{cases} 1, & x \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{23.11}$$

This worst-case approach leads to the concept of l_1 scaling, defined in the following paragraphs.

The l_1-norm of a transfer function $H(z)$ is given by equation 23.12, where $Z^{-1}\{\ \}$ denotes the inverse z-transform.

$$l_1\{H(z)\} = \sum_{t=0}^{\infty} Z^{-1}\{H(z)\}[t] \tag{23.12}$$

A dataflow graph $G(V, S)$ annotated with wordlengths $\boldsymbol{n}$ and scalings $\boldsymbol{p}$ is said to be l_1-scaled} if equation 23.13 holds for all signals $j \in S$.

$$p_j = \left\lfloor \log_2 \left(\sum_{i \in V_I} M_i l_1\{h_{ij}(z)\} \right) \right\rfloor + 1 \tag{23.13}$$

The important point about an l_1-scaled algorithm is that the scalings used are *optimal* in the following sense. If any scaling is reduced lower than its value from equation 23.13, it is possible for overflow to result on that variable. If any scaling is increased beyond its value from equation 23.13, the area of the resulting implementation increases or stays the same without any matching improvement in arithmetic quality observable at the algorithm outputs.

Data range propagation

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

Forward propagation A naive way of approaching this problem is to examine the binary-point position that "naturally" results from each hardware operator. Such an approach, illustrated here, is an option in the Xilinx System Generator tool [20].

In the dataflow graph shown in Figure 23.2, if we consider that each input has a range $(-1, 1)$, then we require a binary-point location of $p = 0$ at each input. Let us consider each of the adders in turn. Adder `a1` adds two inputs with $p = 0$ and therefore produces an output with $p = \max(0, 0) + 1 = 1$. Adder `a2` adds one input with $p = 0$ and one with $p = 1$, and therefore produces an output with $p = \max(0, 1) + 1 = 2$. Similarly, the output of `a3` has $p = 3$, and the output of `a4` has $p = 4$. While we have successfully determined a binary-point location for each signal that will not lead to overflow, the disadvantage of this approach

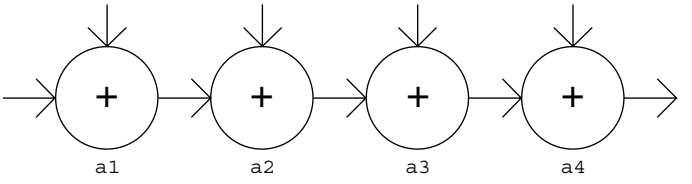

FIGURE 23.2 ■ A dataflow graph representing a string of additions.

should be clear. The range of values reachable by the system output is actually $5^*(-1, 1) = (-5, 5)$, so $p = 3$ is sufficient; $p = 4$ is an overkill of one MSB.

A solution to this problem that has been used in practice is to propagate data ranges rather than binary-point locations [4, 40]. To understand this approach in practice, let us apply the technique to the example of Figure 23.2. The output of adder `a1` is a subset of $(-2, 2)$ and thus is assigned $p = 1$; the output of adder `a2` is a subset of $(-3, 3)$ and is thus assigned $p = 2$; the output of adder `a3` is a subset of $(-4, 4)$ and is thus assigned $p = 3$; and the output of adder `a4` is a subset of $(-5, 5)$ and is thus also assigned $p = 3$. For this simple example, the problem of peak value detection has been solved to optimality.

However, such a tight solution is not always possible with data range propagation. Under circumstances where the dataflow graph contains one or more branches (fork nodes), which later reconverge, such a "local" approach to range propagation can be overly pessimistic. As an example, consider the computation graph representing a constant coefficient multiplication on complex numbers shown in Figure 23.3.

In the figure, each signal has been labeled with a propagated range, assuming that the primary inputs have range $(-0.6, 0.6)$. Under this approach, both outputs require $p = 2$. However, such ranges are overly pessimistic. The upper output in Figure 23.3 has the value $y_1 = 2.1x_1 - 1.8(x_1 + x_2) = 0.3x_1 - 1.8x_2$. Thus, its range can also be calculated as $0.3(-0.6, 0.6) - 1.8(-0.6, 0.6) = (-1.26, 1.26)$. A similar calculation for the lower output provides a range of $(-1.2, 1.2)$. By examining the global system behavior, we can therefore see that in reality $p = 1$ is sufficient for both outputs.

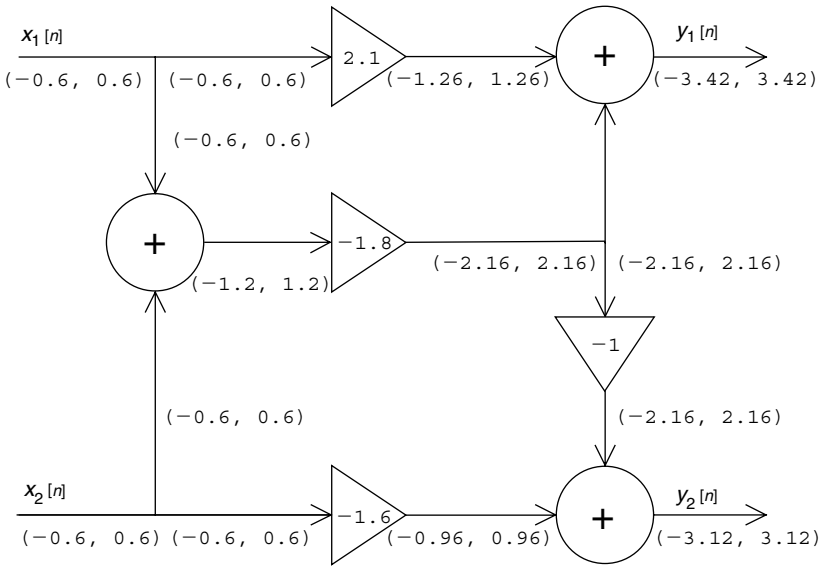

FIGURE 23.3 ■ Range propagation through a complex constant coefficient multiplier. Triangles represent (real) constant coefficient multiplication.

Note that the analytic scheme described previously for linear time-invariant systems would calculate the tighter bound in this case.

In summary, range propagation techniques may provide larger bounds on signal values than are absolutely necessary. This problem is seen *in extremis* with recursive computation graphs. In these cases, it is generally impossible to use range propagation to place a finite bound on signal values, even in cases when such a finite bound can analytically be shown to exist. Under these circumstances, it is standard practice to use some form of simulation to estimate the peak value of signals.

23.2.2 Simulation-based Peak Estimation

A completely different approach to peak estimation is to use simulation—that is, to actually run the algorithm with one or more provided input datasets and measure the peak values reached by each signal.

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

Kim et al. [25] extend the simulation approach by considering more complex forms of the safety factor. In particular, it is possible to extract information from the simulation relating to the class of probability density function followed by each signal. A histogram of the data values for each signal is built, and from it the distribution is classified as unimodal or multimodal, symmetric or nonsymmetric, and zero mean or nonzero mean. Different forms of safety factor are applied in each case.

Simulation approaches are appropriate for nonlinear or time-varying systems, for which data range propagation, described in Section 23.1.2, provides overly pessimistic results (such as for recursive systems). The main drawback of simulation-based approaches is the significant dependence on the input dataset used for simulation; moreover, usually no general guidelines can be given for how to select an appropriate input. These approaches can, of course, be combined with the analytical techniques of Section 23.2.1 [13].

There has been some recent work [34] aiming to put the derivation of safety factors on a sound theoretical footing by using the statistical theory of extreme value distributions [26]. It is known that the distribution of the sum of a large number of statistically independent identically distributed (i.i.d.) random variables approaches the Gaussian distribution (the Central Limit Theorem). What is less well known is that the (scaled) maximum value of a large number of i.i.d. variables also approaches one of three possible distributions, no matter the distribution of the variables themselves. These are the Gumbel, Fréchet, and Weibull distributions [26]. Using this property, and making an assumption on the type of distribution converged to (Özer and colleagues [34] assume Gumbel), provides a statistically sound way of estimating the safety factor required for a given arbitrarily small probability of overflow.

23.2.3 Summary of Peak Estimation

The optimization of a bit-parallel fixed-point datapath can be split into the two problems of determining an appropriate scaling and determining an appropriate wordlength for each signal. We have discussed the first of these two problems in detail. It has been shown that in the case of LTI systems, tight analytic bounds can be placed on the scaling required. Analytic scaling is also possible for non-LTI systems, at the cost of tightness in the bound—disastrously so in the case of recursive systems. The alternative to the analytical approach is the use of simulation on trusted input datasets; some progress has recently been made on the issue of statistically sound simulation-based peak determination.

23.3 WORDLENGTH OPTIMIZATION

Once a scaling has been determined, it is necessary to find an appropriate wordlength for each signal. While optimizing the scaling usually improves circuit quality without changing circuit functionality (assuming no overflows occur), wordlength optimization trades circuit quality (area, delay, power) for result accuracy. The major problem in wordlength optimization is to determine the error at system outputs for a given set of wordlengths and scalings of all internal variables. We will call this problem *error estimation*. Once a technique for error estimation has been selected, the wordlength selection problem reduces to utilizing the known area and error models within a constrained optimization setting: Find the minimum area implementation satisfying certain constraints on arithmetic error at each system output.

The majority of this section is taken up with the problem of error estimation (Section 23.3.1). Following on from this discussion, the problem of area modeling is addressed. Optimization techniques suitable for solving the wordlength determination problem are introduced (Section 23.3.2), with some discussion of the problem's inherent computational complexity.

23.3.1 Error Estimation and Area Models

Traditionally, much of the research on estimating the effects of truncation and roundoff noise in fixed-point systems has focused on DSP uniprocessors. This leads to certain constraints and assumptions on quantization errors—for example, that the wordlength of all signals is the same, that quantization is performed after multiplication, and that the wordlength before quantization is much greater than that following it [36]. The multiple-wordlength paradigm allows a more general design space to be explored, free from these constraints.

The effect of using finite register length in fixed-point systems has been studied for some time. Oppenheim and Weinstein [36] and Liu [29] lay down standard models for quantization errors and error propagation through LTI systems based on a linearization of signal truncation or rounding. Error signals, assumed to be uniformly distributed, uncorrelated with each other and

with themselves over time, are added whenever a truncation occurs. This approximate model has served very well because quantization error power is dramatically affected by wordlength in a uniform wordlength structure, decreasing at approximately 6 dB per bit. This means that it is not necessary to have highly accurate models of quantization error power in order to predict the required signal width [35]. In a multiple-wordlength circuit, the implementation error power may be adjusted much more finely, and so the resulting implementation tends to be more sensitive to errors in estimation. This has led to a simple refinement of the model, which will be discussed soon.

The most generally applicable method for error estimation is simulation: Simulate the system with a given "representative" input and measure the deviation at the system outputs when compared to an accurate simulation (usually "accurate" means IEEE double-precision floating point [22]). Indeed, this is the approach taken by several systems [6, 27]. Unfortunately, simulation suffers from several drawbacks, some of which correspond to the equivalent simulation drawbacks discussed in Section 23.2, and some of which are peculiar to the error estimation problem.

First, there is the problem of dependence on the chosen "representative" input dataset. Second, there is the problem of speed: Simulation runs can take a significant amount of time, and during an optimization procedure a large number of simulation runs may be needed. Third, even the "accurate" simulation will have errors induced by finite wordlength effects that, depending on the system, may not be negligible.

We will be using signal-to-noise ratio (SNR), sometimes referred to as signal-to-quantization-noise ratio (SQNR), as a generally accepted metric for measuring the quality of a fixed-point algorithm implementation [32] (although other measures, such as maximum instantaneous error, exist). Conceptually, the output sequence at each system output resulting from a particular finite-precision implementation can be subtracted from the equivalent sequence resulting from an infinite-precision implementation. The difference is known as the *fixed-point error*.

The ratio of the output power (i.e., the sum of squared signal values) resulting from an infinite precision implementation to the fixed-point error power of a specific implementation defines the SNR. For the purposes of this chapter, the signal power at each output is fixed because it is determined by a combination of the input signal statistics and the dataflow graph $G(V, S)$. To explore different implementations of the dataflow graph, it is therefore sufficient to concentrate on noise estimation, which is the subject of this section.

The approach taken to wordlength optimization should depend on the mathematical properties of the system under investigation. After briefly considering simulation-based estimation, we will examine analytic or semi-analytic techniques that may be applied to certain classes of system. Next we will describe one such method, which may be used to obtain high-quality results for linear time-invariant algorithms. Then we will generalize this approach to nonlinear systems containing only differentiable nonlinear components.

Simulation-based methods

Simulation-based methods for wordlength optimization were first established at Seoul National University, and some of them have been integrated into the Signal Processing Worksystem of Cadence.

In Kim et al. [25] and Kum and Sung [27], the search space is reduced by grouping together all variables involved in a multiply–add operation and optimizing them as a single-wordlength "block." Within each block, the Oppenheim model of quantization noise is applied [35].

Although simulation is almost certainly the most widespread mechanism for estimating the impact of a given choice of wordlength, it suffers from the drawbacks discussed earlier. Indeed, the dependence of the result on the input dataset, while widely acknowledged, is rarely considered in depth. The class of algorithm for which simulation forms a suitable mechanism has also remained unclear. Recently, Alippi [1] proposed an analytical framework within which the question of simulation input dependence can be addressed. A mechanism for understanding the perturbation of Lebesgue-measurable functions, an extremely wide class of algorithmic behavior, has been proposed that uses the theory of randomized algorithms. The essential contribution of this work, for the purposes of fixed-point analysis, has been to demonstrate that simulation is an appropriate mechanism for analyzing fixed-point error. Moreover, Alippi [1] provides a theoretically sound guideline on the number of simulations required in order to be confident, to within a certain probability, that the SNR is within a given limit (alternative signal quality metrics are also Lebesgue measurable and hence can be used as well).

An analytic technique for linear time-invariant systems

We will first address error estimation for LTI systems. An appropriate noise model for truncation of LSBs is described in the subsection that follows. It is then shown that the noise injected through truncation can be analytically propagated through the system in order to measure the effect of such noise on system outputs.

Noise model A common assumption in DSP design is that signal quantization (rounding or truncation) occurs only after a multiplication or multiply–accumulate operation. This corresponds to a uniprocessor viewpoint, where the result of an n-bit signal multiplied by an n-bit coefficient needs to be stored in an n-bit register. The result of such a multiplication is an $n' = 2n$-bit word, which must therefore be quantized down to n bits. Considering signal truncation, the least area-expensive method of quantization [18], the lowest value of the truncation error in 2's complement with $p = 0$, is $2^{-n'} - 2^{-n} \approx -2^{-n}$, and the highest value is 0 (2's complement truncation error is always nonpositive).

It has been observed that values between these values tend to be equally likely to occur in practice, so long as the $2n$-bit signal has sufficient dynamic range [29, 36]. This observation leads to the formulation of a uniform distribution model [36] for the noise of variance $\sigma^2 = 2^{-2n}/12$ for the standard normalization of $p = 0$. It has also been observed that, under the same conditions, the

spectrum of such errors tends to be white because there is little correlation between low-order bits over time even if there is a correlation between high-order bits. Similarly, different truncations occurring at different points within the implementation structure tend to be uncorrelated.

When considering a multiple-wordlength implementation, or truncation at different points within the datapath, some researchers have opted to carry the uniform distribution model over to the new implementation style [25]. However, there are associated inaccuracies involved in such an approach [7]. First, quantizations from n' bits to n bits, where $n' \approx n$, will suffer in accuracy because of the discretization of the error probability density function; for example, if $p = 0$, $n' = 2$, $n = 1$, then the only possible error values are 0 and $-1/4$. Second, in such cases the lower bound on error can no longer be simplified in the preceding manner because $2^{-n'} - 2^{-n} \approx -2^{-n}$ no longer holds.

These two issues may be resolved by considering a discrete probability distribution for the injected error signal. For 2's complement arithmetic, the truncation error injection signal $e[t]$ caused by truncation from (n', p) to (n, p) is bounded by equation 23.14.

$$-2^p\left(2^{-n} - 2^{-n'}\right) \le e[t] \le 0 \tag{23.14}$$

It is assumed that each possible value of $e[t]$ has equal probability, as discussed earlier. For 2's complement truncation, there is nonzero mean $E\{e[t]\}$ (equation 23.15) and variance σ_e^2 (equation 23.16).

$$E\{e[t]\} = -\frac{1}{2^{n'-n}}\sum_{i=0}^{2^{n'-n}-1} i \cdot 2^{p-n} = -2^{p-1}\left(2^{-n} - 2^{-n'}\right) \tag{23.15}$$

$$\sigma_e^2 = \frac{1}{2^{n'-n}}\sum_{i=0}^{2^{n'-n}-1}\left(i \cdot 2^{p-n'}\right)^2 - E^2\{e[t]\} = \frac{1}{12}2^{2p}\left(2^{-2n} - 2^{-2n'}\right) \tag{23.16}$$

Note that for $n_1 \gg n_2$ and $p = 0$, equation 23.16 simplifies to $\sigma_e^2 \approx 1/12\ \ 2^{-2n}$, which is the well-known predicted error variance of Oppenheim and Schafer [35] for a model with continuous probability density function.

Noise propagation and power estimation If it is our aim to optimize the wordlengths used in a design, then it is important to be able to predict the arithmetic quality observable at the design outputs. Given a set of wordlengths and scalings, it is possible to use the truncation model described in the previous section to predict the variance of each injection input. For each signal $j \in S$, a straightforward application of equation 23.16 may be used, with n_1 equal to the "natural" full-precision wordlength produced by the source component, $n_2 = n_j$, and $p = p_j$.

By constructing noise sources in this manner for the entire dataflow graph, a set $F = \{(\sigma_p^2, \boldsymbol{R}_p)\}$ of injection input variances σ_p^2, and their associated transfer function to each primary output $\boldsymbol{R}_p(z)$, can be constructed. From this set it is possible to predict the nature of the noise appearing at the system primary

outputs, which is the quality metric of importance to the user. Since the noise sources have a white spectrum and are uncorrelated with each other, it is possible to use L_2 scaling to predict the noise power at the system outputs. The L_2 norm of a transfer function $H(z)$ is defined in equation 23.17, where Z^{-1} denotes the inverse z-transform. It can be shown that the noise variance E_k at output k is given by equation 23.18.

$$L_2\{H(z)\} = \left(\sum_{n=0}^{\infty} \left| Z^{-1}\{H(z)\}[n] \right|^2 \right)^{1/2} \tag{23.17}$$

$$E_k = \sum_{(\sigma^2,\, R) \in F} \sigma^2 {L_2}^2\{R_k\} \tag{23.18}$$

A hybrid approach for nonlinear differentiable systems

With some modification, some of the results from the preceding section can be carried over to the more general class of nonlinear time-varying systems containing only differentiable nonlinearities. In this section we address one possible approach to this problem, deriving from the type of small-signal analysis typically used in analogue electronics [12, 38].

Perturbation analysis To make some of the analytical results on error sensitivity for LTI systems applicable to nonlinear systems, the first step is to linearize these systems. The assumption is made that the quantization errors induced by rounding or truncation are sufficiently small not to affect the system's macroscopic behavior. Under such circumstances, each system component can be locally linearized or replaced by its "small-signal equivalent" [38] in order to determine the output behavior under a given rounding scheme.

We will consider one such n-input component, the differentiable function $Y[t] = f(X_1[t], X_2[t], \ldots, X_n[t])$, where t is a time index. If we denote by $x_i[t]$ a small perturbation on variable $X_i[t]$, then a first-order Taylor approximation for the induced perturbation $y[t]$ on $Y[t]$ is given by equation 23.19.

$$y[t] \approx x_1[t] \left. \frac{\partial f}{\partial X_1} \right|_t + \ldots + x_n[t] \left. \frac{\partial f}{\partial X_n} \right|_t \tag{23.19}$$

Note that this approximation is linear in each x_i but that the coefficients may vary with time index t because, in general, $\partial f/\partial X_1$ is a function of $X_1, X_2, \ldots, X_n$. Thus, by applying such an approximation, we have produced a linear time-varying small-signal model for a nonlinear time-invariant component. Such an analysis is readily extended to a time-varying component by expressing $Y[t] = f(t, X_1[t], X_2[t], \ldots, X_n[t])$.

The linearity of the resulting model allows us to predict the error at system outputs due to *any* linear scaling of a small perturbation of signal $j \in S$ analytically, given the simulation-obtained error from a *single* such perturbation instance at j, which can be obtained by a single simulation run. Thus, this method can be considered to be a hybrid analytic/simulation error analysis [15].

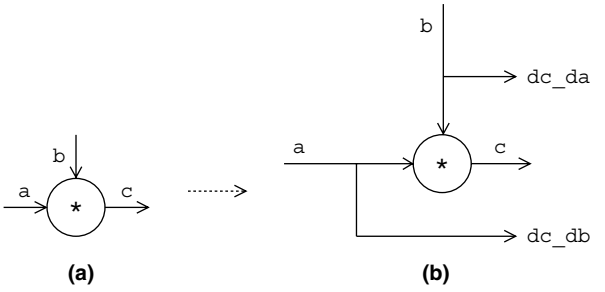

FIGURE 23.4 ■ A local graph transformation to insert derivative monitors: (a) multiplier node; (b) with derivative monitors.

Derivative monitors To construct the small-signal model, we must first evaluate the differential coefficients of the Taylor series model for nonlinear components.

In general, methods must be introduced to calculate the differential of each nonlinear node type. This is performed by applying a graph transformation to the dataflow graph, introducing the necessary extra nodes and outputs to do this calculation.

The general multiplier is the only nonlinear component considered explicitly in this section, although the approach is general; the graph transformation for multipliers is illustrated in Figure 23.4. Since $f(X_1, X_2) = X_1X_2$, $\partial f/\partial X_1 = X_2$ and $\partial f/\partial X_2 = X_1$.

After insertion of the monitors (`dc_da` and `dc_db`, which capture the derivatives of `c` with respect to `a` and `b`, respectively), a simulation may be performed to write the derivatives to appropriate data files to be used by the linearization process, which is described next.

Linearization Our aim is to construct a small-signal model, which can be simulated to determine the sensitivity to rounding errors. Once we have obtained the derivative monitors, the construction of the small-signal model may proceed, again through graph transformation. All linear components (adder, constant coefficient multiplier, fork, delay, primary input, primary output) remain unchanged as a result of the linearization process. Each nonlinear component is replaced by its first-order Taylor model. Additional primary inputs are added to the dataflow graph to read the Taylor coefficients from the derivative monitor files created by the previous large-signal simulation.

As an example, the Taylor expansion transformation for the multiplier node is illustrated in Figure 23.5. The inputs `dc_da` and `dc_db` are themselves time-varying sequences, derived from the previous step of the procedure. Note that the graph portion of Figure 23.5(b) still contains multiplier "nonlinear" components, although one input of each multiplier node is now external to the model. This absence of feedback ensures linearity, although not time invariance.

Noise injection In Section 23.3.1, L_2 scaling was used to analytically estimate the noise variance at a system output through scaling of the (analytically

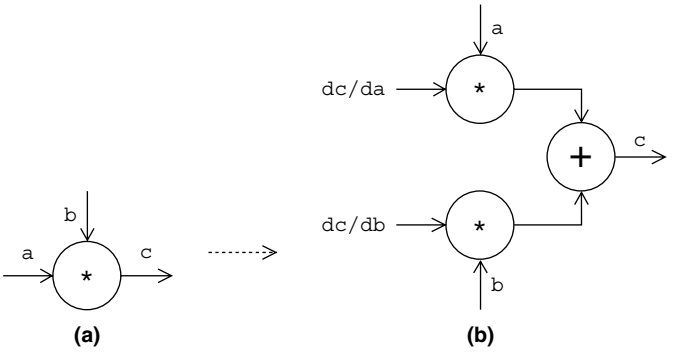

FIGURE 23.5 ■ A local graph transformation to produce a small-signal model: (a) multiplier node; (b) first-order Taylor model.

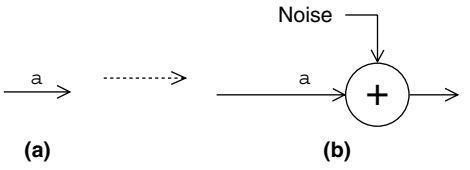

FIGURE 23.6 ■ A local graph transformation to inject perturbations: (a) original signal; (b) with noise injection.

derived) noise variance injected at each point of quantization. Such a purely analytic technique can be used only for LTI systems. In this section we discuss an extension of the approach for nonlinear systems.

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

Thus, the next step of the procedure is to transform the graph through the introduction of an additional adder node, and associated signals, and then simulate the graph with a known noise. In our case, to simulate truncation of a 2's complement signal, the noise is independent and identically distributed with a uniform distribution over the range $[-2\sqrt{3}, 0]$, chosen to have unit variance $(1/12(2\sqrt{3})^2 = 1)$, in this way making the measured output response an unscaled "sensitivity" measure. The graph transformation of inserting a noise injection is shown in Figure 23.6. One of these transformations is applied to a distinct copy of the linearized graph for each signal in the dataflow graph,

after which zeros are propagated from the *original* primary inputs, to finalize the small-signal model. This is a special case of constant propagation [2] that leads to significantly faster simulation results for nontrivial dataflow graphs.

The entire process is illustrated for a simple dataflow graph in Figure 23.7. The original graph is shown in (a). The perturbation analysis will be performed for the signals marked (*) and (**). After inserting derivative monitors

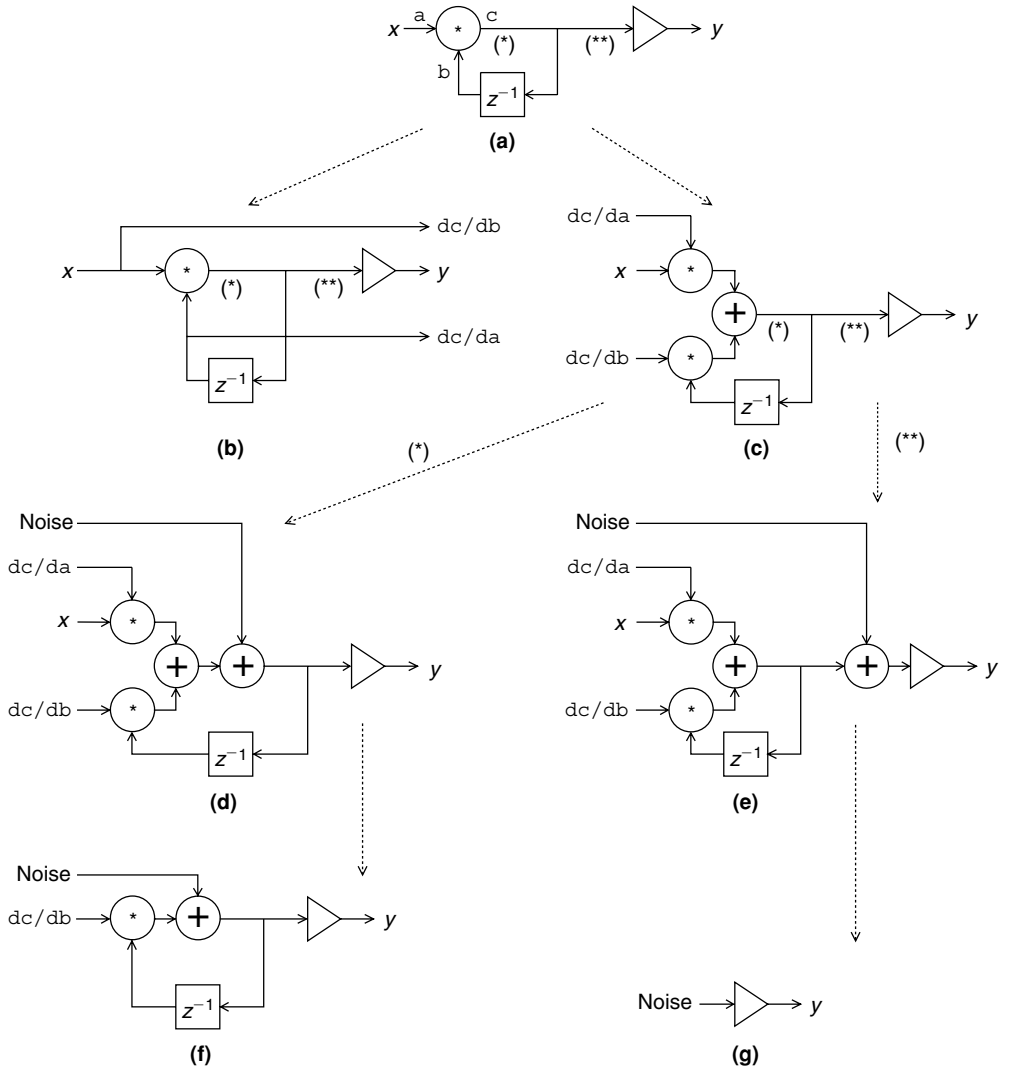

FIGURE 23.7 ■ An example of perturbation analysis: (a) original dataflow graph; (b) transformed dataflow graph; (c) linearized dataflow graph; (d) variant for (*) signal; (e) variant for (**) signal; (f) simplified graph for (*) signal; (g) simplified graph for (**) signal.

for nonlinear components, the transformed DFG is shown in (b). The linearized DFG is shown in (c), and its two variants for the signals (*) and (**) are illustrated in (d) and (e), respectively. Finally, the corresponding simplified DFGs after zero propagation are shown in (f) and (g), respectively.

High-level area models

To implement a multiple-wordlength system, component libraries must be available to support multiple-wordlength arithmetic. These libraries can then be instantiated by the synthesis system and must be modeled in terms of area consumption to provide the wordlength optimization procedure with a cost metric.

Integer arithmetic libraries are available from FPGA vendors (e.g., Xilinx Coregen or Altera LPM macros). Parameterizable macros for standard arithmetic functions operating on integer arithmetic form the basis of the multiple-wordlength libraries synthesized to by wordlength optimization tools such as Right-Size [15] and Synoptix [8]. Blocks from each of these vendors may have slightly different cost parameters, but the general approach described in this section is applicable across all of them. Example external interfaces of multiple-wordlength library blocks for constant coefficient multipliers (`gain`) and adders (`add`) written in VHDL are shown in Listing 23.1 [23].

Listing 23.1 ■ Constant coefficient multipliers (`gain`) and adders (`add`) written in VHDL.

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

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

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

skewed manner, as shown in Figure 23.8. Note, too, that `dataa` and `datab` are permuted as necessary to ensure that `BSHL` is always nonnegative.

In the figure, (a) shows that the MSB of input `b` protrudes beyond that of input `a` and that all the output bits are drawn from the core integer addition of the overlap. Figure 23.8(b) shows that the MSB of input `a` protrudes beyond that of input `b` and that all output bits are drawn from the core integer addition of the overlap. Figure 23.8(c) shows that the MSB of input `b` protrudes beyond that of input `a` but that some of the output bits are drawn from the LSB overhang of input `a` and are thus produced "free." Figure 23.8(d) shows that the MSB of input `a` protrudes beyond that of input `b` but that some of the output bits

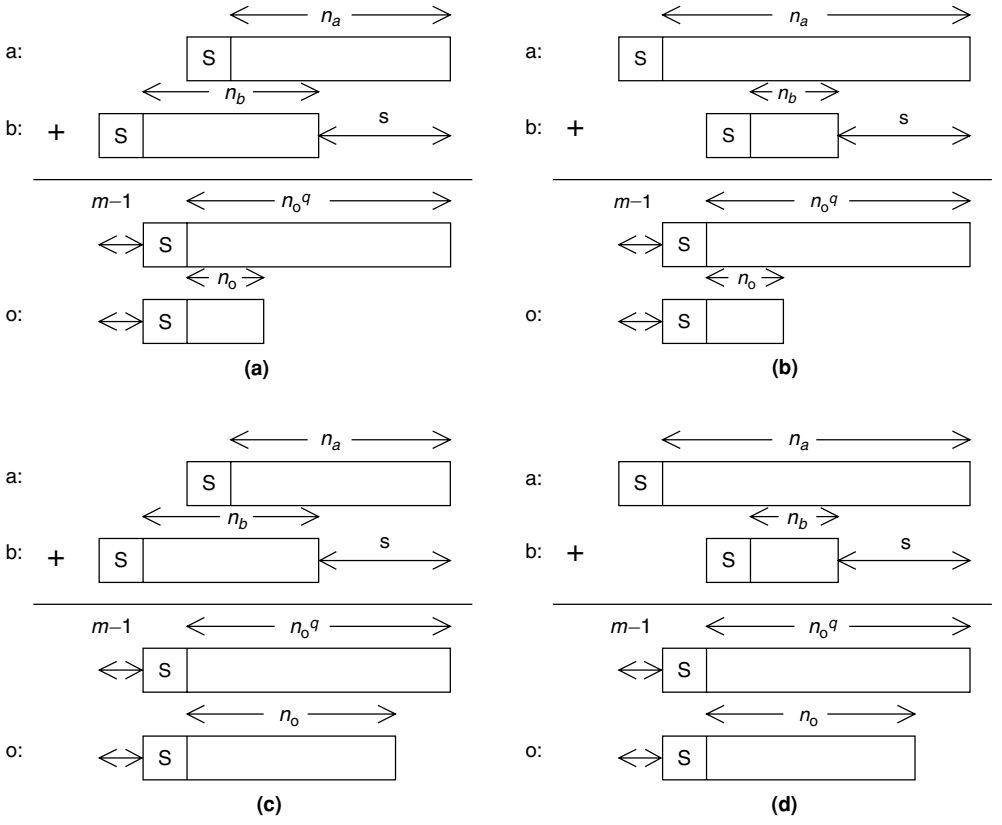

FIGURE 23.8 ■ Four multiple-wordlength adder formats arising in practice: (a) MSB of input `b` protruding beyond MSB of input `a`; (b) MSB of input `a` protruding beyond MSB of input `b`; (c) MSB of input `b` protruding beyond MSB of input `a`, with "free" output bits; (d) MSB input `a` protruding beyond MSB of input `b`, with "free" output bits. (`s` denotes the value of the BSHL generic; `m` denotes the value of the `NULLMSBS` generic.)

are drawn from the LSB overhang of input a and are thus produced "free." In each case, the upper result shows the "error-free" wordlength n_o^q without further truncation, whereas the lower result shows the wordlength n_o after potential further truncation.

Each of the library block parameters has an impact on the area resources consumed by the overall system implementation. It is generally assumed when constructing a cost model that each operator in the dataflow graph will map to a separate hardware resource and that the area cost of wiring is negligible [17]. These assumptions (relaxed by Constantinides et al. [12]) simplify the construction of an area cost model. It is sufficient to estimate separately the area consumed by each computation node and then sum the resulting estimates. In reality, of course, logic synthesis, performed after wordlength optimization, is likely to result in some logic optimization between the boundaries of two connected library elements. This may result in lower area than estimated, but experience shows that these deviations from the area model are small.

The area model for a multiple-wordlength adder is reasonably straightforward. A ripple-carry architecture is used [21] since FPGAs provide good support for fast ripple-carry implementations. The only area-consuming component is the core (integer) adder constructed from the vendor library. This adder has a width of max(AWIDTH − BSHL, BWIDTH) − NULLMSBS + 2 bits. Depending on the FPGA architecture in question, each bit may not consume the same area; however, because some bits are required for the result port whereas others may be needed only for carry propagation, their sum outputs remain unconnected and therefore the sum circuitry is optimized away by logic synthesis. The cost model thus has two parameters k_1 and k_2, corresponding to the area cost of a sum-and-carry full adder and to the area cost of a carry-only full adder, respectively. The area of an adder is expressed in equation 23.20.

$$\begin{aligned} &A_{\text{add}}(\texttt{AWIDTH, BWIDTH, BSHL, NULLMSBS, OUTWIDTH}) \\ &\quad = k_1(\texttt{OUTWIDTH} + 1) + k_2(\max(\texttt{AWIDTHBSHL, BWIDTH}) \\ &\quad\quad - \texttt{NULLMSBS} - \texttt{OUTWIDTH} + 1) \end{aligned} \tag{23.20}$$

Area estimation for general multipliers can proceed in a similarly straightforward way. However, the equivalent problem for constant coefficient multipliers is significantly more problematic. A constant coefficient multiplier is typically implemented as a series of additions through a recoding scheme such as the classic Booth technique [3]. This implementation style causes the area consumption to be highly dependent on the coefficient value. In addition, the exact implementation scheme used by the vendor integer arithmetic libraries is known only to the vendor.

A simple area model has been proposed (equation 23.21) and the coefficient values k_3 and k_4 have been determined through the synthesis of several hundred multipliers of different coefficient values and widths [12]. The model has then been fitted to this data using a least-squares approach. Note that the model does not account for NULLMSBS because, for a properly scaled coefficient,

NULLMSBS ≤ 1 for a constant coefficient multiplier and therefore has little impact on area consumption.

$$A_{\text{gain}}(\texttt{INWIDTH, OUTWIDTH, COEFWIDTH}) = k_3\,\texttt{COEFWIDTH}(\texttt{INWIDTH} + 1) + k_4(\texttt{INWIDTH} + \texttt{COEFWIDTH} - \texttt{OUTWIDTH}) \quad (23.21)$$

More detailed area models for components are discussed by Chang and Hauck [14].

23.3.2 Search Techniques

A heuristic search procedure

Because the wordlength optimization problem is NP-hard [16], several heuristic approaches have been developed to find feasible wordlength vectors having small, though not necessarily optimal, area consumption. An example heuristic is shown in Listing 23.2. After performing binary-point estimation using the techniques of Section 23.2, the algorithm determines the minimum uniform wordlength satisfying all error constraints. The design at this stage corresponds to a standard uniform wordlength design with implicit power-of-two scaling, such as may be used for an optimized uniprocessor-based implementation. Each wordlength is then scaled up by a factor $k > 1$, which represents a bound on the largest value that any wordlength in the final design may reach (in the Synoptix implementation of this algorithm [8], $k = 2$ has been used).

The resulting structure forms a starting point from which one signal wordlength is reduced by one bit on each iteration. The signal wordlength to reduce is decided in each iteration by reducing each wordlength in turn until it violates an output noise constraint (Listing 23.2). At this point there is likely to have been some pay-off in reduced area, and the signal whose wordlength reduction provided the largest pay-off is chosen. Each signal's wordlength is explored using a binary search.

Listing 23.2 ■ Algorithm wordlength falling.

```
Input: A Dataflow Graph G(V,S) and binary-point vector p.
Output: An optimized wordlength vector n.
begin
  Let the elements of S be denoted as S = { j1, j2, ..., j|S| }
  Determine u, the minimum uniform wordlength satisfying error
    criteria
  Set n ← 1ku
  do
     currentcost ←  AREA(n)
     foreach ji ∈ S do
       bestmin ← currentcost
       Set w to the smallest positive value where the error criteria
        are satisfied for wordlength [n1 ... ni-1 w ni+1 ... n|S|]
       Set minval ← AREA([n1 ... ni-1 w ni+1 ... n|S|])
       if minval < bestmin, set bestsig ← i and bestmin ← minval
     end foreach
```

```
    if bestmin < currentcost
      n_bestsig ← n_bestsig − 1
  while bestmin < currentcost
end
```

Alternative search procedures

The algorithm described in Section 23.3.1 is a heuristic; it does not guarantee to produce the optimum area cost for a given set of error constraints. A technique to discover the true optimum-wordlength vectors has also been proposed [10] that uses integer linear programming (ILP) to model the constraint space and objective functions. This technique was able to demonstrate that the heuristic from Section 23.1.1 provides good-quality results for the small benchmark problems addressed by both approaches. Like all NP-hard problems [16], however, finding the optimum solution becomes computationally infeasible for large problem sizes. The methodology of Constantinides et al. [10] is applicable only for very small practical problems and is thus more of a theoretical than practical interest.

Several other heuristic search procedures have been proposed in the literature, and we will review some of the more interesting ones (further comparisons are made in the brief survey by Cantin et al. [6]).

An approach used by Kum and Sung [27] is based on the intuition that the error observable at a system output reduces monotonically with each wordlength in that system. This is a plausible conjecture, but is not always the case. Indeed, it was shown independently by Constantinides [9] and Lehtinen and Renfors [31] that this conjecture may be violated in practical situations. Nevertheless, if we accept it for the moment, a natural search procedure becomes apparent. We may divide the search into two phases. In the first phase, the system is simulated with all but one variable having a very large precision (e.g., double precision floating point). In this way, we can find the point at which the output constraints are violated because of quantization on this variable alone. Repeating this for all variables provides, under the conjecture, a lower bound on each element of the wordlength vector. The second phase of the algorithm is invoked if the constraints are violated when these lower bounds are used as the wordlength vector. In this case, the precision of all variables is increased by an equal number of bits until the constraints are satisfied. A variation on the second phase is to exhaustively explore all possibilities above this lower bound, until the constraints are satisfied [27].

The common meta-heuristics of simulated annealing and genetic algorithms have been used for this problem—for example, by Chang and Hauck [14]—(using a linear combination of area and error as an objective function [28, 40]). While there are practical advantages to using tried-and-tested meta-heuristics for combinatorial problems, the smooth nature of the constraints and objectives, as outlined previously, means that it is likely that better results can be obtained within a fixed computation time budget by using application-specific heuristic techniques.

23.4 SUMMARY

This chapter introduced the fundamental problems of designing optimized fixed-point arithmetic circuits in custom hardware, including FPGA devices. The fixed-point number system is of widespread interest in the FPGA community because of the highly efficient arithmetic implementations possible when compared to what can be achieved with floating-point arithmetic. However, much more than with floating point, working with fixed point requires designers to have a good grasp of the numerical robustness issues involved with their designs. Performing such design by hand is tedious and error prone, which has motivated the development of automatic procedures, some of which have been described in this chapter.

The freedom in custom hardware to use multiple wordlengths in a design creates the possibility of shaping the circuit datapath to the requirements of the algorithm, leading to low-area, high-speed, and low-power implementations. This emerging paradigm throws up a new challenge, however: wordlength optimization.

This chapter demonstrated that wordlength determination can be considered as a constrained optimization, and suitable models were presented for FPGA-based bit-parallel implementations, together with signal-to-noise ratio of linear time-invariant and differentiable nonlinear time-varying systems. In each case, we described at least one error estimation procedure in depth and discussed related procedures and their advantages and disadvantages.

We will now consider some fruitful avenues for further research in this field, broken down into MSB-side optimization, error modeling, and search procedures.

The work discussed in Section 23.2 either avoids overflow completely (e.g., l_1-scaling) or reduces the probability of overflow to an arbitrary level (e.g., extreme value theory) without considering the effect of overflow on signal-to-noise ratio or other accuracy metrics. In algorithms where the worst-case variable range is much larger than the average-case range, it may make sense to save area by allowing rare overflow and its consequent reduction in arithmetic accuracy. This problem was discussed by Constantinides et al. [11] using a simple model of the error induced by overflow, based on approximating all signals by Gaussian random variables. The results achieved were weakened, however, by an inability of the proposed method to accurately estimate the correlations between overflow errors at different points within the algorithm. Further work could provide much stronger bounds.

The analytical error-modeling approaches discussed in Section 23.3.1 can adequately deal with linear time-invariant systems or with time-varying systems containing only differentiable nonlinearities. This still leaves open the problem of adequately modeling systems containing nondifferentiable nonlinearities. This is a serious omission, as it includes any algorithm containing conditionally executed statements, where the condition is a logical expression containing variables generated by the algorithm itself (in the case where the variables

are external inputs, this can be viewed as a time-varying differentiable system). Further work incorporating the results from the analysis of nonlinear dynamical systems is likely to shed new light here.

Both heuristic and optimal search procedures were discussed in Section 23.3.2. One of the limitations of the optimal approach from Constantinides et al. [10] is that is has relied on coercing inherently nonlinear constraints into a linear form, resulting in a large ILP problem. Branch-and-bound, or other combinatorial search procedures, on top of bounding procedures from the more general field of nonlinear mathematical programming may be able to provide optimal results for significantly larger problems. Further effort is also called for in the development of heuristic search procedures. None of the heuristics presented thus far can guarantee a bounded distance to optimality, although under certain error metrics the wordlength optimization problem is approximatible in this sense. It would be useful to concentrate efforts on heuristics that do provide these guarantees.

It is my belief that, apart from a practical design problem, the problem of wordlength optimization has much to offer in terms of understanding the numerical properties of algorithms. The earliest contributions to this subject can be traced back to two giants of computing, Alan Turing [39] and John von Neumann [33]. At the time, IEEE standard floating point was nonexistent, and it was necessary to carefully design the architecture around the algorithm. FPGA-based computing has reopened this method of design by giving an unprecedented degree of freedom in the implementation of numerical algorithms.

References

[1] C. Alippi. Randomized algorithms: A system-level poly-time analysis of robust computation. *IEEE Transactions on Computers* 51(7), 2002.

[2] A. V. Aho, R. Sethi, J. D. Ullman. *Compilers: Principles, Techniques and Tools,* Addison-Wesley, 1986.

[3] A. D. Booth. A signed binary multiplication technique. *Quarterly Journal Mechanical Applications of Mathematics* 4(2), 1951.

[4] A. Benedetti, P. Perona. Bit-width optimization for configurable DSPs by multi-interval analysis. *Proceedings of the 34th Asilomar Conference on Signals, Systems and Computers*, 2000.

[5] M.-A. Cantin, Y. Savaria, P. Lavoie. An automatic word length determination method. *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2001.

[6] M.-A. Cantin, Y. Savaria, P. Lavoie. A comparison of automatic word length optimization procedures. *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2002.

[7] G. A. Constantinides, P. Y. K. Cheung, W. Luk. Truncation noise in fixed-point SFGs. *IEE Electronics Letters* 35(23), November 1999.

[8] G. A. Constantinides, P. Y. K. Cheung, W. Luk. The multiple wordlength paradigm. *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April–May 2001.

[9] G. A. Constantinides. *High-level Synthesis and Wordlength Optimization for Digital Signal Processing Systems*, Ph.D. thesis, University of London, 2001.
[10] G. A. Constantinides, P. Y. K. Cheung, W. Luk. Optimum wordlength allocation. *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2002.
[11] G. A. Constantinides, P. Y. K. Cheung, W. Luk. Synthesis of saturation arithmetic architectures. *ACM Transactions on Design Automation of Electronic Systems* 8(3), 2003.
[12] G. A. Constantinides, P. Y. K. Cheung, W. Luk. *Synthesis and Optimization of DSP Algorithms*, Kluwer Academic, 2004.
[13] M. Chang, S. Hauck. Precis: A design-time precision analysis tool. *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002.
[14] M. Chang, S. Hauck. Automated least-significant bit datapath optimization for FPGAs. *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2004.
[15] G. A. Constantinides. wordlength optimization for differentiable nonlinear systems. *ACM Transactions on Design Automation for Electronic Systems*, January 2006.
[16] G. A. Constantinides, G. J. Woeginger. The complexity of multiple wordlength assignment. *Applied Mathematics Letters* 15, 2002.
[17] G. DeMicheli. *Synthesis and Optimization of Digital Circuits,* McGraw-Hill, 1994.
[18] P. D. Fiore. Lazy rounding. *Proceedings of the IEEE Workshop on Signal Processing Systems*, 1998.
[19] C. Fang, T. Chen, R. Rutenbar. Floating-point error analysis based on affine arithmetic. *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2003.
[20] J. Hwang, B. Milne, N. Shirazi, J. Stroomer. System level tools for DSP in FPGAs. In R. Woods and G. Brebner, eds., *Processing Field Programmable Logic*, Springer-Verlag, 2001.
[21] K. Hwang. *Computer Arithmetic: Principles, Architecture and Design*, Wiley, 1979.
[22] *IEEE Standard for Binary Floating-point Arithmetic* (ANSI/IEEE Standard 991), 1986.
[23] *IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis* (IEEE Standard 1076.6), 1999.
[24] C. Inacio, D. Ombres. The DSP decision: Fixed point or floating? *IEEE Spectrum* 33(9), September 1996.
[25] S. Kim, K. Kum, W. Sung. Fixed-point optimization utility for C and C++ based digital signal processing programs. *IEEE Transactions on Circuits and Systems II* 45(11), November 1998.
[26] S. Kotz, S. Nadarajah. *Extreme Value Distributions: Theory and Applications*, Imperial College Press, 2000.
[27] K.-I. Kum, W. Sung. Combined wordlength optimization and high-level synthesis of digital signal processing systems. *IEEE Transactions on Computer-Aided Design* 20(8), August 2001.
[28] D.-U. Lee, A. Gaffar, R. Cheung, O. Mencer, W. Luk, G. A. Constantinides. Accuracy guaranteed bit-width optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2006.
[29] B. Liu. Effect of finite word length on the accuracy of digital filters—A review. *IEEE Transactions on Circuit Theory* 18(6), 1971.
[30] H. Levy, D. W. Low. A contraction algorithm for finding small cycle cutsets. *Journal of Algorithms* 9, 1988.

[31] V. Lehtinen, M. Renfors. Truncation noise analysis of noise shaping DSP systems with application to CIC decimators. *Proceedings of the European Signal Processing Conference*, 2002.
[32] S. K. Mitra. *Digital Signal Processing*, McGraw-Hill, 1998.
[33] J. von Neumann, H. H. Goldstine. Numerical inverting of matrices of high order. *Bulletin of the American Mathematics Society* 53, 1947.
[34] E. Özer, A. Nisbet, D. Gregg. Stochastic bit-width approximation using extreme value theory for customizable processors. *Proceedings of the International Conference on Compiler Construction*, 2004.
[35] A. V. Oppenheim, R. W. Schafer. *Digital Signal Processing,* Prentice-Hall, 1975.
[36] A. V. Oppenheim, C. J. Weinstein. Effects of finite register length in digital filtering and the fast fourier transform. *IEEE Proceedings* 60(8), 1972.
[37] W. Sung, K. Kum. Simulation-based wordlength optimization method for fixed-point digital signal processing systems. *IEEE Transactions on Signal Processing* 43(12), December 1995.
[38] A. S. Sedra, K. C. Smith. *Microelectronic Circuits,* Saunders, 1991.
[39] A. Turing. Rounding-off errors in matrix processes. *Quarterly Journal of Mechanics* 1, 1948.
[40] S. A. Wadekar, A. C. Parker. Accuracy sensitive wordlength selection for algorithm optimization. *Proceedings of the International Conference on Computer Design*, October 1998.