International Scholarly Research Network
ISRN Signal Processing
Volume 2011, Article ID 414293, 17 pages
doi:10.5402/2011/414293

Review Article

An Automated Fixed-Point Optimization Tool in MATLAB XSG/SynDSP Environment

Cheng C. Wang,[1] Changchun Shi,[2] Robert W. Brodersen,[3] and Dejan Marković[1]

[1] *Electrical Engineering Department, University of California, Los Angeles, CA 90095, USA*
[2] *P.O. Box 4004, Incline Village, NV 89450, USA*
[3] *Berkeley Wireless Research Center, Berkeley, CA 94704, USA*

Correspondence should be addressed to Cheng C. Wang, c2wang@ee.ucla.edu

Received 8 December 2010; Accepted 20 January 2011

Academic Editor: B. Yuan

This paper presents an automated tool for floating-point to fixed-point conversion. The tool is based on previous work that was built in MATLAB/Simulink environment and Xilinx System Generator support. The tool is now extended to include Synplify DSP blocksets in a seamless way from the users' view point. In addition to FPGA area estimation, the tool now also includes ASIC area estimation for end-users who choose the ASIC flow. The tool minimizes hardware cost subject to mean-squared quantization error (MSE) constraints. To obtain more accurate ASIC area estimations with synthesized results, 3 performance levels are available to choose from, suitable for high-performance, typical, or low-power applications. The use of the tool is first illustrated on an FIR filter to achieve over 50% area savings for MSE specification of 10^{-6} as compared to all 16-bit realization. More complex optimization results for chip-level designs are also demonstrated.

1. Introduction

Modern DSP systems are usually implemented from infinite-precision algorithms, generally represented in decimal-point numbers. For example, in the equation

$$a = \pi + b, \tag{1}$$

if $b = 5.6$ and $\pi = 3.1416$, we can compute a with relative ease, but most of us would prefer not to compute using binary numbers, where $b = 101.1001100110$ and $\pi = 11.0010010001$. With this abstraction, many algorithms are developed without too much consideration of the binary representation in actual hardware, where something as simple as $0.3 + 0.6$ can never be computed with full accuracy. As a result, the designer may often find the actual hardware performance to be different from expected or that large hardware costs are required for implementation with sufficient precision [1]. The hardware cost depends on the application, but it is generally a combination of performance, energy, or area [2] for most VLSI and DSP designs. Designing hardware with sufficient quantization accuracy and minimal hardware cost is often an iterative process, requiring numerous computer simulations to determine the accuracy and logic synthesis to determine the hardware cost. This greatly impacts both man-hour and time-to-market, as each iteration is a change in the system level and system-level specifications ought to be frozen months before chip fabrication.

To avoid iterative changes in achieving optimal word-lengths, an automated optimization tool is discussed in this paper. This tool operates within the MATLAB/Simulink environment and is publicly available for download. Some knowledge of floating-point to fixed-point conversion (FFC) is useful for efficient usage of this tool and is discussed in Section 2. Section 4 reviews wordlength optimization techniques. Modeling and optimization theory is presented in

Section 3, followed by usage optimization flow in Section 5. Section 6 demonstrates few design examples of different complexity, and Section 7 concludes the paper.

2. Floating-Point to Fixed-Point Conversion

As explained in [3], floating-point arithmetic with large mantissa and exponent often approximates the infinite-precision algebraic algorithm with acceptable accuracy. In a modern computing language, floating-point arithmetic with 32 bits or 64 bits is often employed. The accuracy of floating-point calculation is an important and diverse subject in itself, yet it is common for an algorithm to go through a simulation in floating-point arithmetic with the assumption that the numerical errors caused by floating-point representations are negligible. This assumption is often benign, especially for communication systems and most signal processing systems to be implemented in a hardware solution. In these cases, the algorithm is to overcome more significant effects caused by imperfect modeling, estimations, and subjective interpretations of the physical world. Examples of these kinds are the modeling loss of a communication channel, treating front-end circuit uncertainty as thermal electronic noise, fussiness in judging how similar two objects appear to a human, and so forth. These algorithmic imperfections hide the smaller numerical errors caused by quantization. Therefore, "floating point" in our context really means high-precision representation of a system that can be abstracted as an infinite-precision system. This high-precision design is used as a reference in our optimizations.

In most hardware designs, a high-precision floating-point implementation is often too much of a luxury. Hardware requirements such as area, power consumption, and operating frequency all demand more economical representations of the signal. A lower-cost (and higher-performance) alternative to floating-point arithmetic is fixed point, where the binary point is "fixed" for each data path (Figure 1), and the bit-field is divided into the sign bit (if needed), the integer wordlength (W_{Int}), and the fractional wordlength (W_{Fr}). The maximum representable precision is $2^{-W_{\text{Fr}}}$, and the dynamic range is limited to $2^{W_{\text{Int}}}$. While these limitations may seem unreasonable for a general-purpose computer (unless very large wordlengths are used), they are acceptable for many dedicated hardware designs in specific applications where the input range and precision requirements are well defined. In a communication receiver design, for example, this is often ensured by performing automatic gain control (AGC) up front. Although many well-specified design environments for fixed point exist, we use the Simulink environment because the designer can easily specify wordlength variables, along with overflow mode for saturation or wraparound, and quantization mode for rounding or truncation. It should be noted, however, that selecting rounding and saturation modes usually increases hardware usage.

When sufficiently large wordlengths are chosen in a fixed-point representation of the algorithm, it becomes another high-precision version of the infinite-precision algorithm.

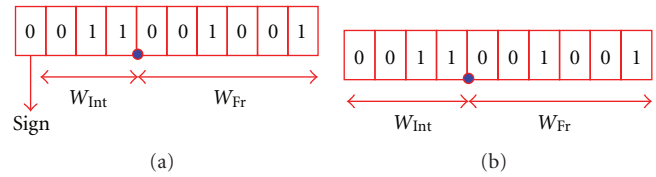


Figure 1: A fixed-point (a) signed number and (b) unsigned number.

From a design perspective, efficiently converting a high-precision design to fixed point requires careful allocation of wordlengths as well as overflow and quantization modes. For example, while excessive wordlength leads to slower performance, larger area and higher power, insufficient wordlength introduces large quantization errors and can heavily degrade the precision of the system. Finding the best tradeoff point in the design space related to fixed-point representations of the original algorithm in an automated way is a topic of great interest. In the literature, this problem is sometimes referred to as wordlength optimization. We, and some others, prefer to call it floating-point to fixed-point conversion, or in short FFC, to emphasize that there should be an initial high-precision reference design and that our tool is to aid in producing its fixed-point version. Besides wordlengths, fixed-point data type also includes other information such as quantization and overflow mode. The high-precision design should already contain all the detailed architectural information. To explore the architectural design space, FFC should be performed on each architectural solution, and the resulting fixed-point design may already show sufficient information to make a design decision; otherwise pushing further into the design flow might be necessary. In fact, one ultimate goal of an efficient and fully automated FFC is to allow such high-level design exploration. More subtle design decisions such as how many taps to include in an adaptive filter are also within this class of problem. It is not hard to imagine that, by introducing more taps in a filter, global system specifications can potentially be satisfied with fewer restrictions on quantization errors. To study all these types of smaller variations to the system, an efficient FFC becomes even more critical.

This section reviews key research results in wordlength optimization. The issues addressed include analytical methods for modeling quantization errors and integrated tool support. The main challenge is to realize a practical tool for an *automated* wordlength optimization that is built on sound theoretical foundations. We first emphasize the advantages and disadvantages of various techniques and then outline our research approach to address the challenges of an automated wordlength optimization.

2.1. Early FFC Tools. In the recent 15 years or so, much attention was given to addressing the FFC problem. Before the investigation of analytical approaches, early efforts focused on building practical FFC tools in specific design environments. Here we review representative approaches.

One notable past technique for determining both W_{Int} and W_{Fr} is Fixed-point pRogrammIng DesiGn Environment (FRIDGE) [4]. In FRIDGE, W_{Fr} is optimized through deterministic propagation, in which the user specifies W_{Fr} at every input node, and every internal node is then assigned a sufficiently large W_{Fr} to avoid any further quantization errors. For example, W_{Fr} of an adder is the maximum of its input W_{Fr} and is the sum of its input W_{Fr} for a multiplier. However, this propagation approach based purely on static structure of the design is often overly conservative and has other drawbacks as well. First, the input W_{Fr} is chosen by the user, which is unjustified by the optimization, so different W_{Fr} at the input leads to completely different results. In addition, not all W_{Fr} can be determined through propagation; some logic blocks (e.g., a feedback multiplier) require user interaction. Due to these issues with the FRIDGE technique, we will only recommend it for W_{Int} optimization, and methods of W_{Fr} optimization are still to be determined.

Another approach for W_{Fr} optimization is through iterative bit-true simulations by Sung et al. [5, 6]. The fixed-point system can be described in software models or Simulink blocks where W_{Fr} for every node is described as a variable. With each simulation, the quantization error of the system (e.g., bit-error rate, signal-to-noise ratio, mean-squared error) is evaluated directly along with a predefined hardware cost (e.g., area, power, or delay) that is computed as a function of wordlengths. Since the relationship between wordlength and quantization is not characterized for the target system, the wordlengths in each iteration are determined in an ad hoc fashion, and numerous iterations are often needed to locate the wordlength-critical blocks [7]. Such iterative search for large systems can be impractical when stringent performance specifications (e.g., a very small bit-error rate in a communication system) are required.

Though impractical for automation, the work by Sung et al. shows the power of bit-true simulations that include architectural descriptions of the system. This simulation environment is not easy to set up. Creating an environment that incorporates the detailed architectural description often becomes the focus of separate research teams, even though the work was originated or targeted as practical FFC tool [8]. We aim to avoid this difficulty by adopting a simulation and design environment in MATLAB Simulink. Companies such as Xilinx and Synopsys, as well as academic research groups, recognize the advantage of this simulation and architectural description environment and invested large resources to support it. The benefit of Simulink is the allowance of bit-true and cycle-true simulations to model actual hardware behavior using functional blocks and third-party blocksets such as Xilinx System Generator and Synopsys Synplify DSP (now called Synphony HLS). This environment also allows direct mapping into hardware description language (HDL), which eliminates the error-prone process of manually converting software language into HDL. Simulink environment practically allows one-to-one mapping between the high-level design and the final low-level logic gates. Each block in the system description is even self-aware of its neighboring connections. This level of structural cognition could be implemented in procedural languages such as C, C++, and MATLAB, but by leveraging this mature Simulink design environment, we could then focus on building an efficient FFC tool.

Another important optimization concept proposed by [5, 6] is "cost efficiency"; where the goal of each iteration is to minimize the hardware cost as a function of wordlengths while meeting the system requirement for quantization error. This implies an optimization framework that we later explicitly proposed [9, 10]. To achieve a wordlength-optimal design, it is necessary to locate the logic blocks that provide the largest hardware cost reduction with minimal increase in quantization error. The formulation for fixed-point data type optimization is founded on this concept, but a nonautomated approach is required to achieve acceptable results within a reasonable timeframe. Simple models for hardware cost were proposed for basic design blocks, but the cost function is created separately by the designer, which could be a tedious work caused by frequent design changes. The importance of grouping various blocks to have the same fractional wordlength to reduce the design complexity was proposed by [6], but grouping was performed manually. Recent similar approaches include [11, 12]. The work in [13] uses simplified noise propagation but otherwise similar approach to address the FFC problem.

2.2. Analytical Work. A large body of past literature studied the quantization effects analytically. The studies range from the fundamental topic of quantization noise of individual quantizer to specialized studies based on individual problems. An extensive survey of such research results prior to 2004 was provided in [3].

Being able to utilize these rich and often mathematically involved research results would certainly provide valuable insights and simplification to the FFC problem. But since these results are done in a system-specific way, it is difficult to utilize them in a general automated tool. Therefore it is important to formulate a rather general theoretical understanding of quantization effects. We pushed the research direction in this front first by generalizing the quantization effect for all linear time-invariant (LTI) systems under stationary stochastic input [14]. While basic signal processing blocks such as FIR, IIR, and FFT are all LTI systems, to have a useful theory for general signal-processing system consisting of a large number of signal-processing blocks, a deeper understanding of quantization effect is necessary. Based on deep understanding of the quantization effects combined with our original "perturbation theory," we were able to abstract quantization noise effects into an elegant formulation which applies to a broad range of designs [15–17]. This will be illustrated in later sections.

In parallel with Shi et al.'s work, Constantinides utilized a "perturbation analysis" to understand quantization effects for nonlinear systems [18]. Both his perturbation *analysis* and our perturbation *theory* start with somewhat similar ideas of linearizing smooth nonlinear systems. However, the perturbation analysis [18] remains as a high level observation of quantization noise behavior. Not much attention was paid to mathematical rigor; the only mathematical relationship

was later shown in [19] without proof and missed the important contribution from the nonzero mean of truncation noises.

In contrast to Constantinide's work, the foundation of Shi's theory is the use of the small-signal nature of quantization noises (thus the validation of system linearization). The theory involves a detailed understanding of global quantization effects of a general system under general input environment. Assumptions such as the noncorrelation between data and quantization noise were clearly stated and studied. The perturbation theory explicitly takes into account the decision-making blocks (which turn a small quantization noise to a large logic error), time-varying input signal, nonzero mean of the quantization noise (such as truncation effects), and the fixed-point effects of constant coefficients in a design. In applying the theory to FFC problem, Shi et al. also emphasized the reason to use the mean-squared quantization errors between the high-precision design and fixed-point design as the specifications. It also showed the relationship of such MSE errors to the original system specification such as signal-to-noise ratio and bit-error rate. Therefore it is our understanding that the perturbation theory and Constantinides' perturbation analysis are largely different research efforts. To date we believe that our perturbation theory gives the most general yet still concrete description of quantization effects. Its effective usage in FFC problem is just one of its many applications.

2.3. Design Automation. The early analytical works spent their main effort on mathematically understanding an algorithm or architecture's gross quantization effects. Although interesting, it is not an automated way to perform FFC. Later, the usage or digital computing allows one to do bit-true simulations to have concrete understanding of the performance of a fixed-point system, which allowed the designer to do iterative simulations to explore FFC problem. The treatment of integer wordlength by FRIDGE can be viewed as an example. Sung et al. noticed the importance of abstracting hardware information and grouping wordlengths together to more efficiently explore the design tradeoffs, but at a manual level. They and other groups also attempted to organize the search for optimal fixed-point data types based on bit-true simulations, so that the number of simulations can be reduced and made suitable for automation. The pitfall there is that with very limited insights into the hardware cost function and the statistical performance of the system as a whole, the depth of such efforts is limited.

A high level of automation and fast conversion are key requirements for an efficient FFC tool. Shi and Brodersen [15–17] formulated and implemented an *automated* FFC framework. First, it explicitly formulates the problem as an optimization: to minimize the hardware cost function with wordlength, overflow, modes and quantization modes as variables. The optimization is subject to the constraints such that the resulting fixed-point system should be close to the given high-precision system under the typical test vectors. Thus the adoption of mean-squared error (MSE) between the two systems becomes a natural specification. It utilizes perturbation theory for an efficient estimation of quantization effects. An automated hardware resource estimation is used to propagate low-level design information to the MATLAB level, as opposed to creating a separate resource estimation file manually. It explores the self-awareness of the Simulink design at block level to automatically group different wordlengths together (deterministically and heuristically) to reduce the design space. It also lays out the detailed treatments of quantization modes in the optimization framework. In general, the implemented FFC tool was applied extensively (and is continuously being used) by the authors and collaborators to optimize complex communication systems or signal processing blocks. The original FFC tool was implemented specifically for Xilinx FPGA design in 2004. With constant modification and improvement, it now applies to different target designs (Synplify DSP and ASIC), and it includes refinements based on user feedback. For example, while the original tool focused on wordlength design and did not include features specifically for quantization mode optimization, we now (again based on the perturbation theory) improve the tool to also optimize the quantization modes in a practical way. The current tool presented in this paper remains the state-of-the-art in many aspects. We also believe that different research teams should examine the source code and consider porting the tool to their design environments.

Constantinides et al. took similar approach to ours in [18–21]. They also start from Simulink environment. It utilizes certain theoretical guidance, called perturbation analysis, for FFC. While the perturbation analysis was fairly limited, the whole FFC approach was a good direction to take. They later extended the work to optimizing the power of a design in [19]. Many of the similarities between these two distinct research groups may be raised from the fact that both teams use Simulink as the design editor. It is not clear to us what level of automation was offered in Constantinides' tool. Nevertheless, we feel it is of readers' interests to have a careful look of his and his colleagues' work [18–21].

In the rest of this paper, we summarize key results of the FFC perturbation theory and then discuss new research results. The emphasis is on the automation, efficiency, and further extension of the FFC tool to cover more design targets. Together with the rest of design flow automation, all the way to the final chip, it demonstrates that design automation for chips is rapidly approaching the level of automation that high-level software language compilation had enjoyed for years. The paper contributes by documenting our updates of the tool and shows its application to different systems and design targets. Interested readers can find extensive explanation of FFC theory and related work in [3, 14]. These two works remain a rich review of FFC as many important results and discussions were not published elsewhere.

3. Automated Wordlength Optimization—Theory

The details of the theory behind our FFC approach are given extensively in [3]. Here we summarize key results used in practice. The framework of the wordlength-optimization

problem is formulated as follows: a hardware cost function is created as a function of every wordlength (actually every *group* of wordlengths, Section 5.3), and such function ought to be minimized subject to meeting all quantization-error specifications (Figure 2) [6, 7]. Such specification may be defined for more than one output, in which case all j requirements need to be met. Since the optimization focuses on wordlength reduction, it is required to start with a design that meets the quantization-error requirements. Since a spec-meeting design is not guaranteed from users, a large number N is initialized for the W_{Fr} of every block, where N is a chosen in such a way as to make the system practically full precision. This leads to the feasibility requirement where a design with wordlength N must meet the quantization-error specification, else a more relaxed specification or a larger N is required. As with most optimization programs, a tolerance a is required for the stopping criteria. A larger a decreases optimization time, but in wordlength optimization, simulation time far outweighs the actual optimization time, so the default value of a is generally used. Since W_{Int} and overflow mode can be determined from onesimulation, similar to FRIDGE and most other tools, the remaining optimization is only required to determine quantization modes and W_{Fr}.

3.1. Modeling Quantization Error. To avoid iterative simulations, which sometimes can be very long themselves when the statistics to be estimated are small error-rates, it is essential to understand that our design problem at the FFC step is to create a fixed-point system that mimics the high-precision system that was already verified extensively, separately. The natural measure of the similarity between these two systems is the mean-squared error of their difference under various input vectors. Based on the original perturbation theory [16], we observe that such MSE follows an elegant formula for the fixed-point data types. The theory, in essence, linearizes a smooth nonlinear time-varying system, and the result is highlighted here;

$$\begin{aligned}
&\text{MSE}\\
&= E\left[\left(\text{Infinite-Precision-Output} - \text{Fixed-Point-Output}\right)^2\right]\\
&= \overline{\mu}^T \mathrm{B}\overline{\mu} + \sum_{i=1}^{p} \mathrm{C}_i 2^{-2W_{\text{Fr},i}}, \quad \mathrm{B} \in \mathbb{S}_+^p,\ \mathrm{C} \in \mathbb{R}_+^p,\\
&\mu_i = \begin{cases} \dfrac{1}{2} q_i 2^{-W_{\text{Fr},i}}, & \text{for data path,} \\ \text{fxpt}\left(c_i, 2^{-W_{\text{Fr},i}}\right) - c_i, & \text{for const } c_i, \end{cases}\\
&q = \begin{cases} 0, & \text{round-off,} \\ 1, & \text{truncation.} \end{cases}
\end{aligned} \tag{2}$$

4. Techniques for Wordlength Optimization

The result states that the MSE error, as defined in (2), can be modeled as a function of all the fractional wordlengths, all the quantization modes, and all the constant coefficients to be quantized in the system. The only unknown variable is the B matrix that captures the interplays of all the means among nonzero-mean quantization-noise sources. The C vector captures how the random nature of the quantization noise sources further contributes to the MSE. The nonnegative nature of MSE implies that B is positive semidefinite and C is nonnegative. Both B and C depend on the system architecture and input statistics in a complicated way—often too complicated to understand theoretically.

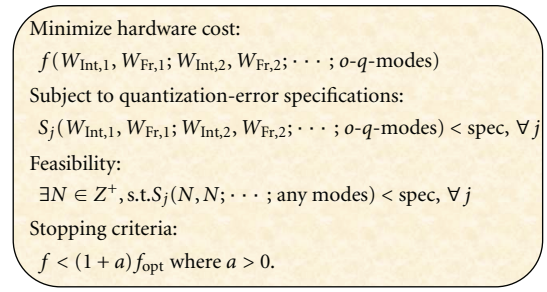


Figure 2: Framework for the automated wordlength optimization tool.

Fortunately, we can estimate the B and C numerically. We can first specify a high number, for example, 50 bits, for all fractional wordlengths and determine the high-precision outputs at the points of interest in the design. Let us first ignore the quantization of constant coefficients, then when only round-off mode is considered, we can just use p number of simulations to identify the C vector, where p is the total number of quantizers along the datapath. Each of the simulations is to decrease one of the fractional wordlengths to a much smaller value (e.g., from 50 to 16). This would expose the contribution to the MSE of the corresponding C_i. To characterize each element of B, we need to introduce truncation mode and use smaller wordlength for only the i and j quantizers.

It is important to notice that while a large design could originally contain hundreds or thousands independent wordlengths to be optimized at the beginning, the design complexity can be drastically reduced by grouping of related blocks to have the same wordlength. In practice, after reducing the number of independent wordlengths, a complex system may only have a few or few tens of independent wordlengths. The reduced wordlength variables form a new matrix B and vector C that are directly related to the original B and C by combining the corresponding terms. The new B and C have considerably less number of entries to estimate, which reduces the number of simulation required.

The locations where the MSE between fixed-point and floating-point models need to be monitored are discussed in detail in [3]. Most of the time, the MSE requirement can be derived from the original system specifications, such as SNR, BER or other types or decision error rates. However, it may require additional nodes where so called hard decision making blocks are present. A hard decision-making block is the one which would amplify the small quantization noise accumulated at its input to a large decision error, and the

decision errors turn out to alter the system MSE error behavior considerably. Not all decision-making blocks are hard. Under soft-decision making blocks, the system may not be directly linearizable using small-signal perturbation theory, yet the MSE can still be formulated [3]. One example of a hard decision block is the frequency and time synchronization units, since a wrong decision there will surely have a hard decision impact to the rest of the data path.

Due to modeling errors and estimation errors, it is often practical to use a more stringent MSE requirement than what appeared necessary in early stages of design. Since MSE is related to wordlength in an exponential way, a moderately more stringent MSE often will not result in a much larger wordlengths. It is still important to test the resulting fixed-point system against the original system specification, if any, as the last verification step. Many details, including the reason for adopting an MSE-based specification, along with justifications for the assumptions used in the perturbation theory are explained in [3].

Once the B and C are estimated, the MSE can be predicted at different combinations of practical wordlengths and quantization modes. This predicted MSE should match closely to the actual MSE as long as the underlying assumptions used in perturbation theory still apply reasonably. The actual MSE is estimated by simulating the system with the corresponding fixed-point data types. Figure 3 demonstrates the validity of the noncorrelation assumption. Shown is a jitter compensation design [29] where simulations are used to fit the coefficients B and C, which in turn are used to directly obtain the "computed" MSE. The actual MSE in x-axis is from simulation of the corresponding fixed-point data types. By varying the fixed-point data types we see that the computed MSE from the once estimated B and C fits well with the actual MSE across the board range of MSEs. This is a special-case verification of the perturbation theory, and [3] explained a number of examples supporting this result from various angles.

4.1. Modeling Hardware Cost. Having an accurate MSE model alone is not sufficient for wordlength optimization. In Figure 2, the optimization goal is to minimize the hardware cost (as a function of wordlength) while meeting the criteria for MSE, therefore hardware cost is evaluated just as frequently as MSE cost, and need to be modeled efficiently. When the design target is an FPGA, hardware cost generally refers to area, but for ASIC designs it is generally power or performance that defines the hardware cost. Traditionally, the only method for area estimation is design mapping or synthesis, but such method is very time consuming. The Simulink design needs to be first compiled and converted to a Verilog or VHDL netlist, then the logic is synthesized as look-up-tables (LUTs) and cores for FPGA, or standard cells for ASIC. The design is then mapped and checked for routability within the area constraint. Area information and hardware usage can then be extracted from the mapped design. This approach is very accurate, for it only estimates the area after the design is routed, but the entire process can take minutes to hours and needs to be re-executed with even the slightest change in the design. These drawbacks hinder it from being used in our optimization, where fast and flexible estimation is the key, as each resource estimation cannot consume more than a fraction of a second. Therefore, a model-based resource estimation is developed to provide area estimations based on cost functions. Each cost function returns an estimated area of a logic block based on its functionality and design parameters such as input and output wordlengths, overflow and quantization modes, and number of inputs. These design parameters are automatically extracted from the Simulink-based design to obtain the area cost, and the cost of each block is accumulated to provide a total area.

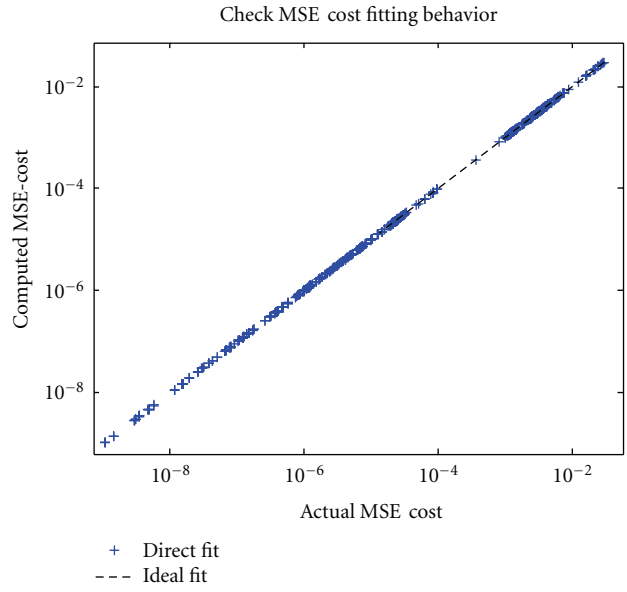


Figure 3: Actual versus computed MSE for as SVD U-Sigma design.

For FPGA applications, area is the primary concern, but for ASIC applications, the cost function can also be changed to model energy or logic delay. The exact FPGA cost functions for Xilinx System Generator blocks are proprietary to Xilinx [23], but the end-user may create similar cost functions for ASIC designs by characterizing synthesis results. The details of ASIC area estimation are covered in the next section.

Since each individual cost function is a quadratic function of W_{Fr}, the total cost function of the design can be modeled as

$$f(W) \approx W^T H_1 W + H_2 W + h_3 \quad \text{where } W = \left(W_{\mathrm{Fr},1}, W_{\mathrm{Fr},2}, \ldots\right)^T. \tag{3}$$

From Figure 4, it is apparent that a quadratic fit provides sufficient accuracy for both FPGA and ASIC area estimations. A linear fit is subpar and is only recommended when the quadratic fit takes too long to complete. It is important to note that the fit function satisfies the property that its derivative to all wordlengths is nonnegative for all

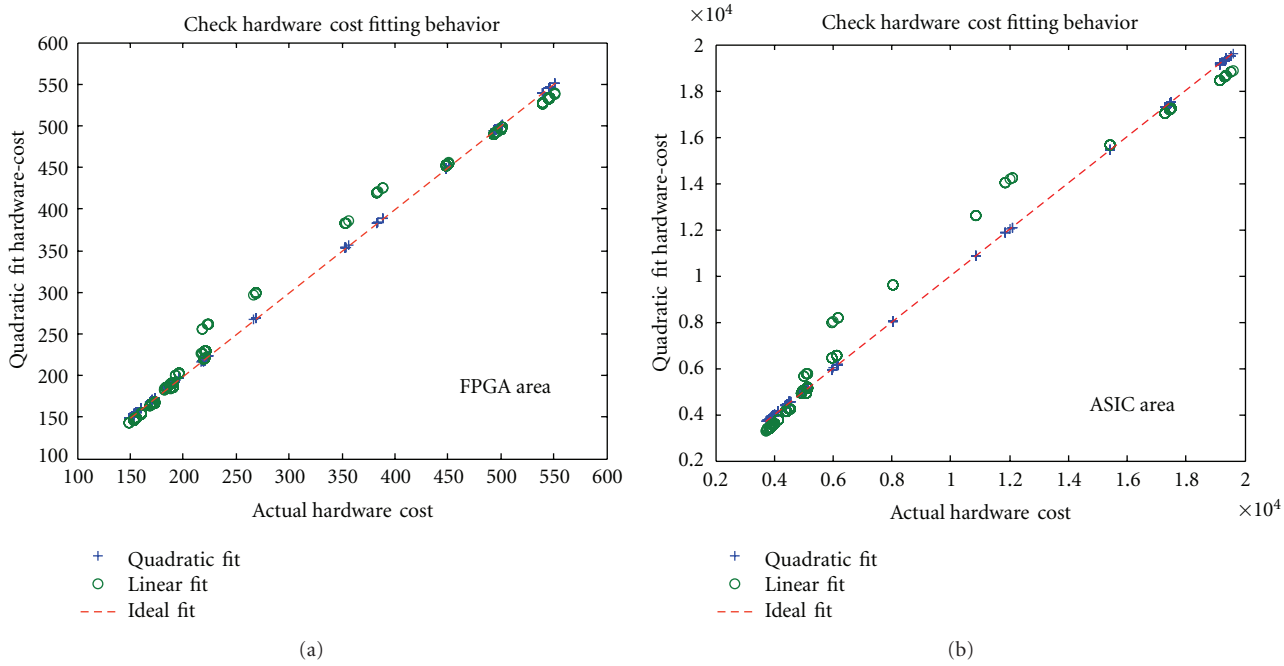


Figure 4: Actual versus computed hardware cost for (a) FPGA and (b) ASIC area-estimation on two designs.

nonnegative wordlengths. This means the cost must monotonically increase in response to any wordlength increase for all nonnegative wordlengths. The area should always be nonnegative as well, meaning all entries of H_1, H_2, and H_3 should be nonnegative. However, this function is not convex in general, as H_1 is often not positive semidefinite.

4.2. Area Estimation for the ASIC Flow. One key advantage of Synopsys Synplify DSP blocksets (now called Synopsys Synphony HLS, but we chose to retain the abbreviation SynDSP in this paper) over XSG is their advantage to create synthesizable Verilog for ASIC synthesis, which greatly expands the scope of this tool beyond FPGA applications. However, the area estimations from [17] are all constructed for the FPGA flow. Due to core usage and LUT structures on the FPGA, the logic area of the FPGA may differ significantly from that of an ASIC, where no cores are used and all logic blocks are synthesized to standard-cell gates. This means an area-optimal design for the FGPA flow is not necessarily optimal for the ASIC flow. It is therefore a key task to provide an accurate area estimation for designs targeting ASIC synthesis.

Most ASIC logic is synthesized given a timing constraint, and the synthesized areas can differ greatly based on the performance criteria. For example, area of a ripple-carry adder is roughly linear to its wordlength, but the area of a carry-look-ahead adder tends to be on the order of $O(N \cdot \log N)$. Therefore for each area characterization, 3 performance criterions will be evaluated. The "high-performance" synthesis reflects the fastest possible synthesized logic, while the "low-power" synthesis reflects the smallest possible synthesized logic. The "typical" synthesis aims to minimize area given a 30–50% performance slack between the high-performance and the low-power designs (roughly representing the minimum area-delay product) [23]. The area function for each performance mode can be fitted into a quadratic function of its design parameters by using a least-squared curve fit in MATLAB.

To accommodate a large variety of DSP blocks, area of adders, multipliers, and registers is characterized first, and many other logic macros can be modeled based on the area information of these low-level primitives. Adder area is a multidimensional function of its input and output wordlength, along with rounding/truncation options. Choosing a signed or unsigned option does not have a significant impact on area (other than the extra bit), but choosing a rounding mode may add, on average, 30% area overhead.

Adder area is more sensitive to the longer of its two input wordlengths, along with its output wordlength—it is approximately linear to these variables, as shown in Figure 5(a). The multiplier area, however, is not as sensitive to its output wordlength; instead, it is sensitive to the shorter of its two input wordlengths. This means a 4-bit by 4-bit multiplier consumes significantly more area than a 2-bit by 6-bit multiplier, even though the outputs for both multipliers are 8 bits. The multiplier area shown in Figure 5(b) indicates that increasing the wordlength of one input only impacts the total area by a small amount, but once the wordlength of the other input is also increased, total area increases quadratically.

Once the area information for each block is collected for a variety of wordlengths, a least-squares fit is used to build an area estimation function for each block (and performance level). The accuracy of the estimated area is then compared

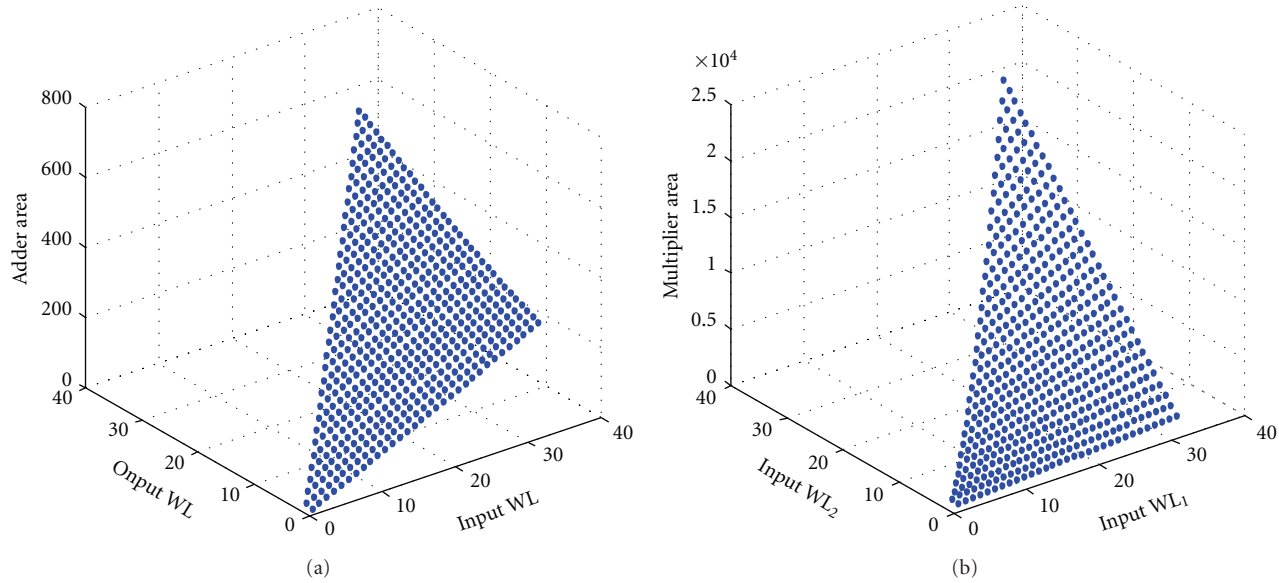


Figure 5: (a) Adder area against its input- and output-WL and (b) multiplier area against its input-WLs.

against actual synthesized area, and the results have shown good fidelity of the model [24].

4.3. Standard-Cell-Based ASIC Area Estimation. Running detailed synthesis for different wordlength combinations is a time-consuming task and is not always feasible for many users. In this case, a simpler alternative is to model area based on standard cell documentation available for chip synthesis. In Figure 6, some snapshots of a standard-cell document are shown, where the dimensions of the gates are marked.

From the area information of each standard-cell, some area estimations can be modeled. For example, an N-bit accumulator can be modeled as the sum of N full-adder cells and N registers, an N-bit, M-input mux can be modeled as $N \cdot M$ 2-input muxes, and an N-bit by M-bit multiplier can be modeled as $N \cdot M$ full-adder cells. The gate sizes can be chosen based on performance requirements. Low-power designs are generally synthesized with gate sizes of $2\times$ (relative to a unit-size gate) or smaller, while high-performance designs typically require gate sizes of $4\times$ or higher. Using these approximations, ASIC area can be modeled very efficiently. Although the estimation accuracy is not as good as the fitting from the synthesis data, it is often sufficient for wordlength optimization purposes.

With adequate models for both MSE cost and hardware cost, we can now proceed with automated wordlength optimization. The next section covers both the optimization flow and usage details of the wordlength optimization tool, which is publicly available for download [25].

5. Automated Wordlength Optimization—Usage Flow

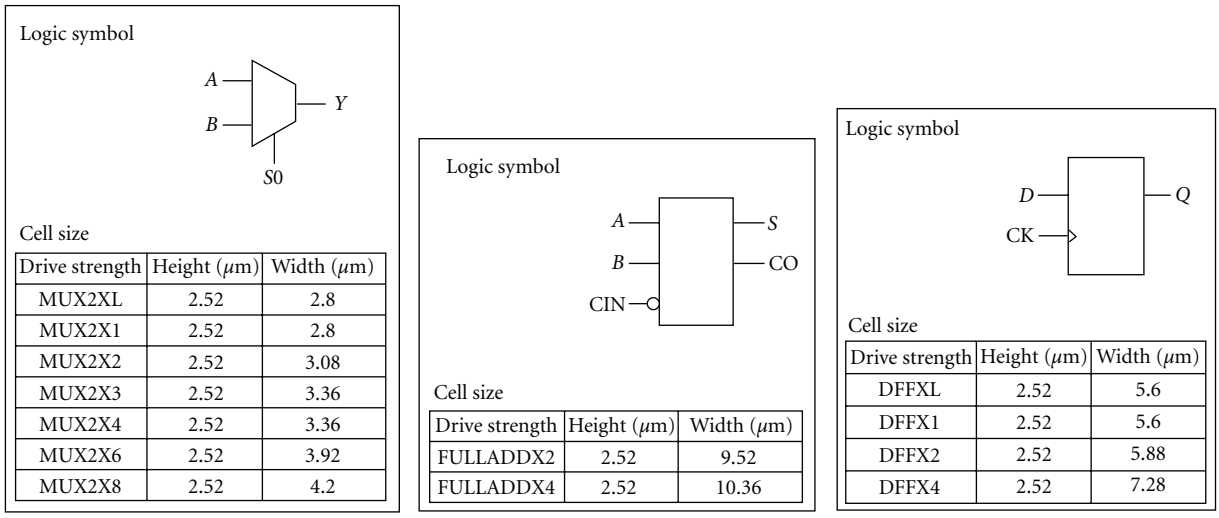
The optimization tool is built in the MATLAB Simulink environment. The original tool from [17] supports only the XSG blockset, but now XSG and SynDSP blocksets are both supported in separate versions of the tool for ASIC support. The user therefore needs to create the design using one of these blocksets. Since generic Simulink blocks cannot be automatically mapped to hardware, it is not supported.

The optimization flow is shown in Figure 7. The bold-faced steps require user interaction. This section describes each of the major steps in the flow, as labeled in the flow graph.

5.1. Initial Setup. Before proceeding to the optimization, an initial setup is required. A setup block (FFC Tool, Figure 8) needs to be added from the optimization library, and the user should open the setup block to specify the parameters. The area target of the design (FPGA, or ASIC of HP, MP, or LP) should be defined. Some designs have an initialization phase in simulation that should not be used for MSE characterization, so the user may specify the portion of the outputs (Output Range) to consider. The optimization rules apply to wordlength grouping and are introduced in Section 5.3. Default rules of [1.1 3.1 4 8.1] are a good start for most users.

The user needs to specify the wordlength range to use for MSE characterization, discussed in Section 3. For example, [8,40] specifies a W_{Fr} of 40 to "full precision," and each MSE iteration will "minimize" one W_{Fr} to 8 to determine its impact on total MSE. Depending on the application, a "full precision," W_{Fr} of 40 is generally sufficient, though smaller values improve simulation time. A "minimum" W_{Fr} of 4 to 8 is generally sufficient, but designs without high sensitivity to noise can even use minimum W_{Fr} of 0. If multiple simulations are required to fully characterize the design, the user needs to specify the input vector for each simulation in the parameter box.

The final important step is the placement of specification markers. The tool characterizes MSE only at the location where Spec Marker is placed, therefore it is generally useful



Drive strength	Height (μm)	Width (μm)
MUX2XL	2.52	2.8
MUX2X1	2.52	2.8
MUX2X2	2.52	3.08
MUX2X3	2.52	3.36
MUX2X4	2.52	3.36
MUX2X6	2.52	3.92
MUX2X8	2.52	4.2

(a)

Drive strength	Height (μm)	Width (μm)
FULLADDX2	2.52	9.52
FULLADDX4	2.52	10.36

(b)

Drive strength	Height (μm)	Width (μm)
DFFXL	2.52	5.6
DFFX1	2.52	5.6
DFFX2	2.52	5.88
DFFX4	2.52	7.28

(c)

Figure 6: Snapshot of a standard-cell documentation for (a) 2-input mux, (b) full-adder cell, and (c) register.

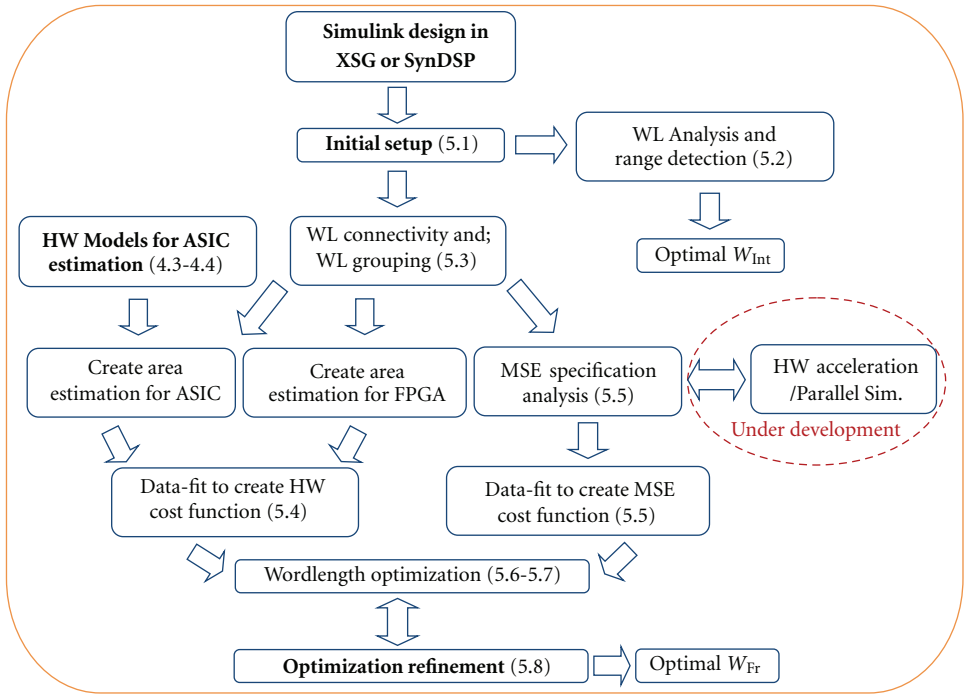


Figure 7: Flow graph of the wordlength optimization tool. Boldfaced steps require user interaction.

to place a Spec Marker at all outputs, and at some important intermediate signals as well (Section 3.1).

5.2. Wordlength Analysis. Based on the FRIDGE algorithm in Section 4, the wordlength analyzer determines W_{Int} from a single iteration of the provided test vector(s), therefore it is important that user provides input test vector(s) that covers the entire range of input, otherwise the unused bits may be mistaken as unnecessary and removed. Here a range detector is a customized block that is used to collect signal statistics. During wordlength analysis, a "Range Detector" is automatically inserted at each active node (Figure 9). Passive nodes such as subsystem input and output ports, along with constant numbers and nondata path signals (e.g., mux selectors, enable/valid signals), are not assigned a range detector.

The range-detector block gathers information such as the mean, variance, and the maximum value at each node. The number of integer bits is determined to be able to cover the mean with ± 4 times the standard deviation (by default) or

its maximum value, whichever requires more bits. With each test vector, if the calculated W_{Int} is greater than the previously determined W_{Int}, the new W_{Int} is used. W_{Int} of constants are determined based on their values directly to ensure no loss of information.

5.3. Wordlength Connectivity and Grouping. With W_{Int} determined by the wordlength analyzer, the remaining effort aims to optimize W_{Fr} in the shortest timeframe possible. Since the number of iterations for optimizing W_{Fr} is proportional to the number of wordlengths, reducing the number of wordlengths is attractive for speeding up the optimization. The first step is to determine the wordlength-passive blocks, which are blocks that do not have physical area, such as input and output ports of submodules in the design, and can be viewed as feedthrough from the wordlength perspective.

The second step is wordlength grouping, which locates wordlength dependencies between blocks and groups the related blocks together under one wordlength variable. Deterministic wordlength grouping includes blocks whose wordlength is fixed, such as mux-select, enable, reset, address bus, and comparator signals, along with constants. These wordlengths are marked as fixed. Some blocks implicitly do not alter its input fixed-point data-type; examples are registers, shift registers, and up- and down-samplers. These wordlengths can be grouped with their source blocks to share the same wordlength information (shown in gray-shaded blocks of Figure 10).

Some wordlengths can be grouped heuristically, such as muxes: allowing each data input of a mux to have its own wordlength group may result in a slightly more optimal design, but it can generally be assumed that all data inputs to a mux have the same wordlength. The same applies to adders/subtractors. Grouping these inputs into the same wordlength can further reduce simulation time, though at a small cost of design optimality. These heuristic wordlength groupings are defined as eight general types of "rules" for the optimization tool, with each type of rules being subdivided to more specific rules. Currently there are rules 1 through 8.1. These rules are defined in the tool's documentation [25] and can be enabled or disabled in the initialization block. We should emphasize that once the rules are chosen, the rest of connectivity and grouping are done automatically by the tool. Users only see the reduced number of independent wordlengths after the groupings have been made.

This level of automation would be much more difficult if Simulink lacked the support of dereferencing among ports, wires, blocks and their properties. If our FFC tool is to be ported to other design environments, considerable attention may be necessary on environment specifics to allow the same level of automation of the connectivity and groupings. We believe grouping is an important level of automation in FFC, for in a complicated system, it is too much burden to the designer to accurately group wordlengths by hand.

5.4. Creating Hardware Cost Function. XSG did not originally have support for high-level resource estimation needed for FFC. One of the authors spent a summer and worked out a version with the XSG team to help implement this feature, which in turn allowed the successful demonstration of FFC methodologies in [23]. The current paper further extends this resource estimation tools to SynDSP and ASIC design.

The hardware cost function is the sum of the hardware costs of individual logic blocks, discussed in Section 4.1. The constructed function is then evaluated iteratively by the hardware cost analyzer. Since each wordlength group defines different logic blocks, they each contribute differently towards the total area. It is therefore necessary to iterate through different wordlength combinations to determine the sensitivity of total hardware cost to each wordlength group. A quadratic number of iterations is usually recommended for a more accurate curve fitting of the cost function (Figure 4). However, if there are too many wordlength groups (e.g., more than 100), then a less accurate linear fit will be used to save time. There are continuous research interests to extend the hardware cost function to include power estimation and speed requirement. Currently these are not fully supported in our FFC tool, but can be implemented without structural change to the optimization flow.

5.5. MSE Specification Analysis. The MSE-specification analysis is based on Section 3, in which the perturbation theory allows the MSE contribution of each block to be examined individually. There we also explained the efficient way to estimate matrix B and vector C. While the full B matrix and C vector are needed to be estimated to fully solve the FFC problem, this would imply an order of $O(N^2)$ number of simulations for each test vector, which sometimes could still be too slow to do. However, it is often possible to drastically reduce the number of simulations needed by exploring design-specific simplifications. One such example is if we are only interested in rounding mode along the data path. Ignoring the quantization of constant coefficients for now, the resulting problem is only related to the C vector, thus only $O(N)$ simulations are needed for each test vector. For truncation modes, a new approach highlighted in Section 5.7 avoids $O(N^2)$ simulations.

For smaller designs and short test vectors, the analysis is completed within minutes, but larger designs may take hours or even days to complete this process, though no intermediate user interaction is required. Fortunately, all simulations are independent of each other, thus many runs can be performed in parallel. Parallel simulation support is currently being implemented. FPGA-based acceleration is a much faster approach, but requires mapping the full-precision design to an FPGA first and masking off some of the fractional bits to 0 to imitate a shorter-wordlength design. The masking process must be performed by programming registers to avoid reperforming synthesis with each change in wordlength. This approach is also under development [26].

5.6. Wordlength Optimization—Rounding Mode. After the MSE-analysis, both MSE and hardware cost functions are available. The user is then prompted to enter an MSE

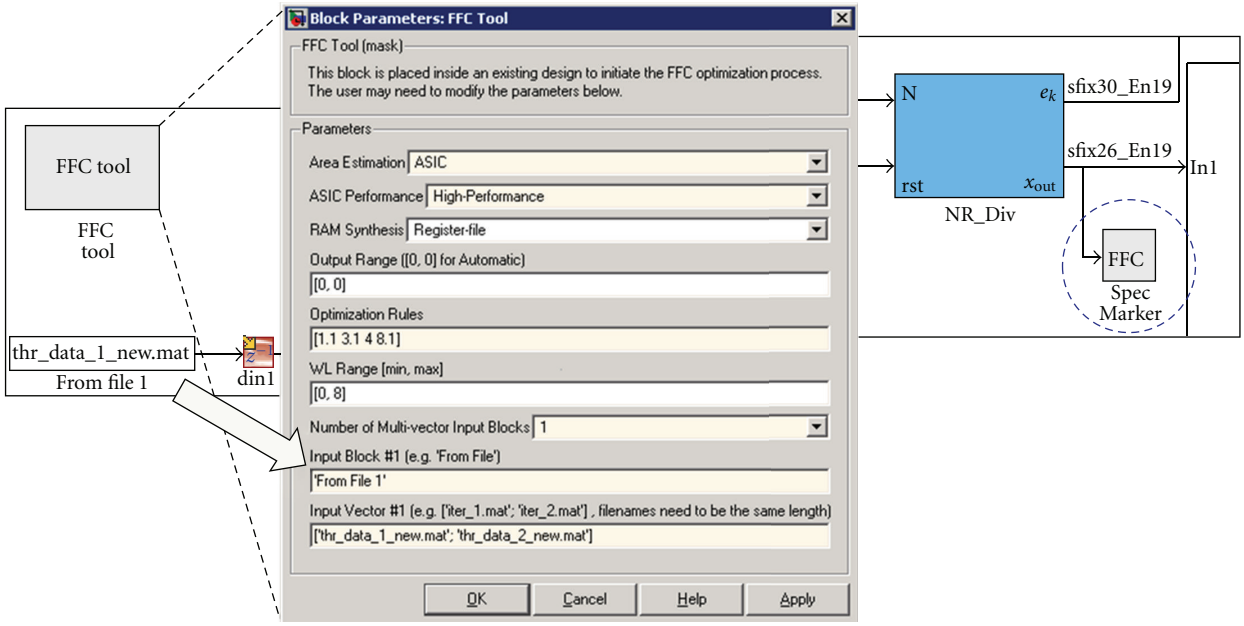


Figure 8: Configuration snapshot for initial setup.

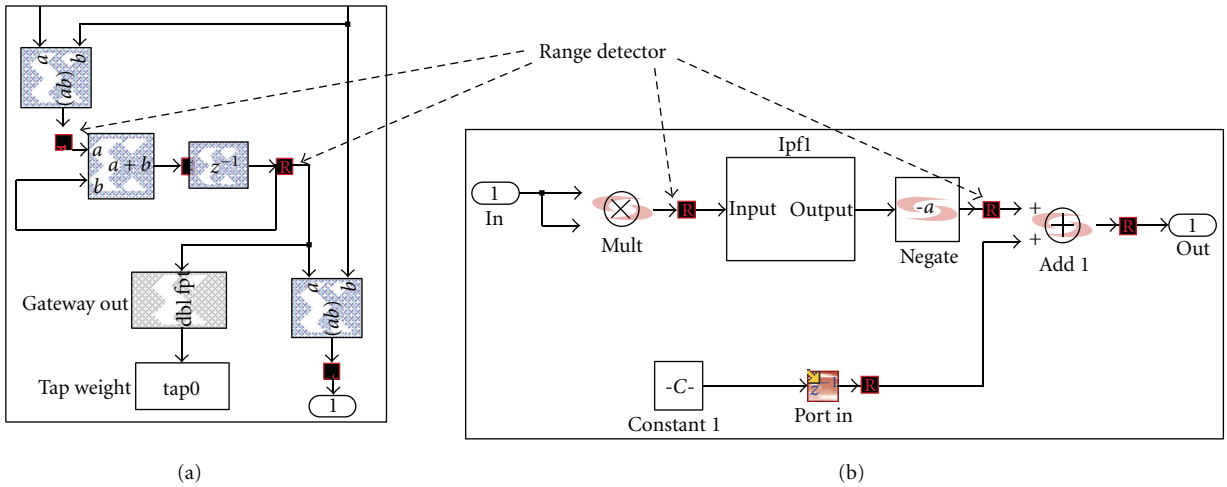


Figure 9: Snapshots of inserted Range Detector for (a) XSG and (b) SynDSP designs.

requirement. If more than one Spec Marker exists in the design, a vector of MSE specification is required: 1 element for each Spec Marker. The MSE requirements are suggested to be obtained from various system specifications [3].

Following Figure 2, the MSE requirement is first examined for feasibility in the "floating-point" system, where every wordlength variable is set to its maximum value. Once the requirement is considered feasible, the tool employs the following algorithm for wordlength reduction.

While keeping all other wordlengths at maximum, each wordlength group is reduced individually to find the minimum possible wordlength while meeting the MSE requirement. Each wordlength group is then assigned its minimum possible wordlength. This is not likely to meet the MSE requirement, so all wordlengths are then increased uniformly until the requirement is met. The wordlength for each group is then reduced temporarily. Since the hardware cost is guaranteed to be nonincreasing with reducing wordlength, the group that results in the largest hardware reduction while meeting the MSE requirement is chosen. This procedure is then iterated until no further hardware reduction is feasible, and a wordlength optimal solution is created.

There are likely other more efficient algorithms to explore the simple objective function and constraint function. For example, a quasiconvex optimization can be used to approach the problem, but we want to emphasize that since we now have the analytical format of the optimization problem, any reasonable optimization procedure will yield the near-optimal point. The important step is the process that allowed us to abstract the original complex design problem to

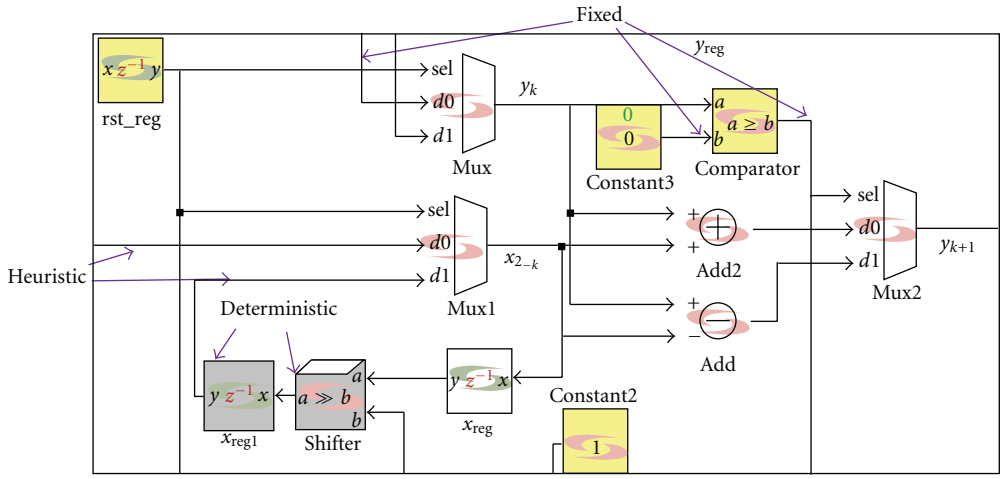


Figure 10: Illustration of WL grouping—dark-shaded blocks belong in the same WL group and light-shaded blocks have fixed WL.

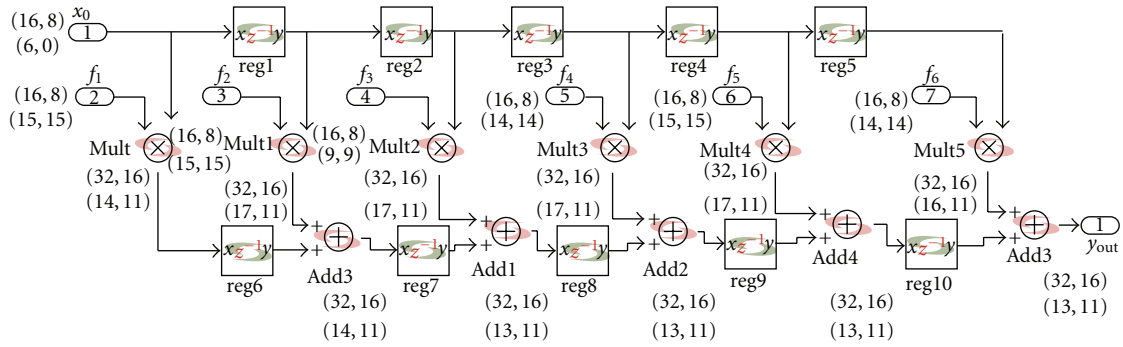


Figure 11: A design example of a 6-tap FIR filter before (top label) and after (bottom label) wordlength optimization for MSE of 10^{-6}, area is 48916 μm^2 and 18356 μm^2, respectively. The wordlength numbers are (total, fractional).

a simple mathematical optimization. Even though a guaranteed globally optimal point of this nonconvex optimization problem is hard to obtain, obtaining reasonable optimality for this concrete optimization problem is welldefined and often fast under a chosen algorithm.

5.7. Wordlength Optimization—Truncation Mode. From Section 3.1, the full matrix B is necessary to estimate MSE under truncation mode, but simulation time on the order of $O(N^2)$ makes this approach difficult for large systems. We now present a methodology that allows the optimization under truncation mode without the need to explicitly estimate the B matrix. This may introduce nonoptimality of the final design, but we suggest that it is practically acceptable.

It is often favorable to use truncation mode along the data path since the inclusion of rounding mode could increase the area significantly [24]. If the user prefers to explore using truncation mode uniformly for parts of the data path, the optimization proceeds as follows.

(1) Find optimal wordlength vector in the rounding mode for a given MSE_{spec}, call it $\text{WL}_r{}^*$. The optimal hardware cost is $\text{HW}(\text{WL}_r{}^*)$, where subscript r stands for rounding mode.

(2) Switch to truncation mode, then with the wordlength being $\text{WL}_r{}^*$, the same MSE criteria will most likely not be satisfied due to the introduction of the B matrix. The truncation MSE is named MSE_t.

(3) Based on (2), we know that

$$\text{WL}_{t,\text{conservative}}{}^* = \text{WL}_r{}^* + \text{ceiling}\left(\frac{1}{2}\cdot\log_2\left(\frac{\text{MSE}_t}{\text{MSE}_{\text{spec}}}\right)\right) \tag{4}$$

would satisfy the MSE in the truncation mode. The sum here is to apply the latter scalar to all entries of the $\text{WL}_r{}^*$ vector that are subject to truncation. This could be a conservative design, $\text{WL}_{t,\text{conservative}}{}^*$, but it will satisfy the original MSE criteria. When there are multiple nodes for MSE specification, then (4) becomes

$$\text{WL}_{t,\text{conservative}}{}^* = \text{WL}_r{}^* + \max\left(\text{ceiling}\left(\frac{1}{2}\cdot\log_2\frac{\text{MSE}_{t,i}}{\text{MSE}_{\text{spec}}}\right)\right), \quad \forall i. \tag{5}$$

This formulation exploits the fact that even with the presence of the B matrix, the total MSE in the truncation mode decreases uniformly with the increase of all wordlengths.

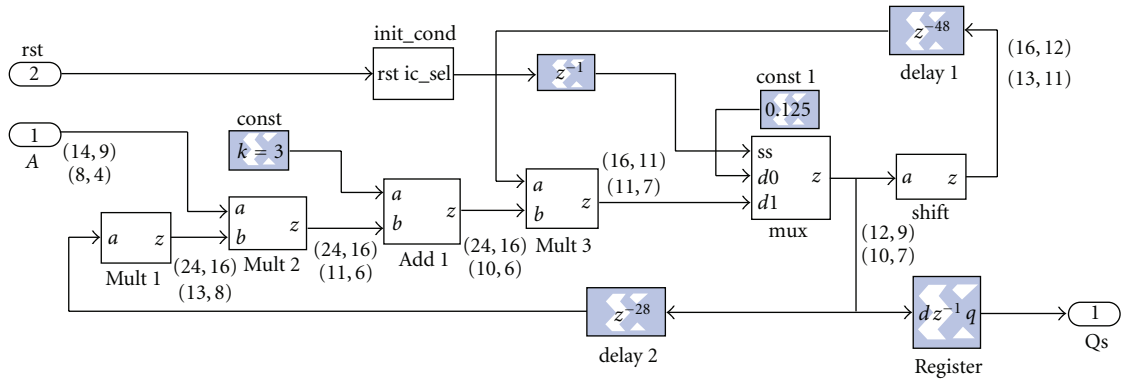


Figure 12: A design example of a 1/sqrt operator before (top label) and after (bottom label) wordlength optimization.

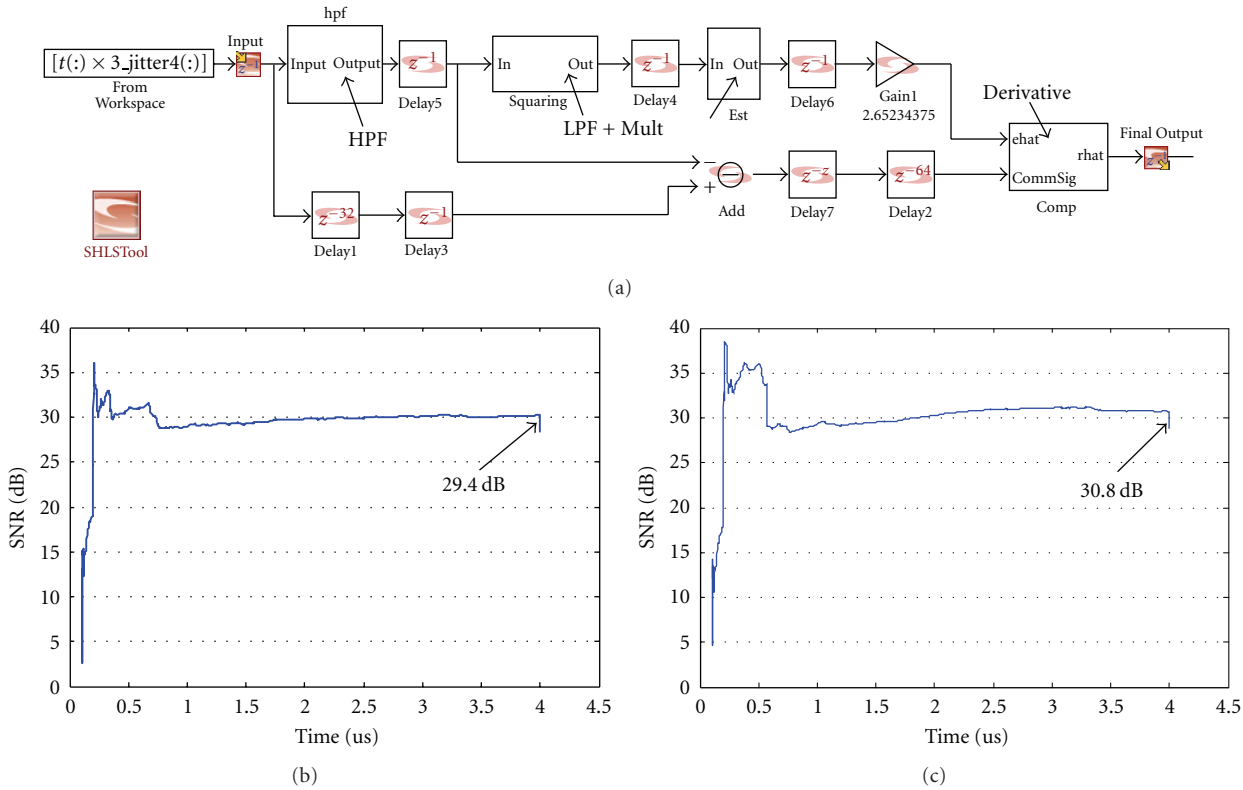


Figure 13: A design example of (a) a jitter compensation unit, (b) shows its original performance, and (c) shows its performance after wordlength optimization for a MSE of 4.5×10^{-9}.

(4) If necessary, $\mathrm{HW}(\mathrm{WL}_{t,\mathrm{conservative}}{}^{*})$ is compared against $\mathrm{HW}(\mathrm{WL}_{r}{}^{*})$. If the savings in hardware from truncation are significant, the former is a more optimal design. But there are cases where the difference between MSE_t and $\mathrm{MSE}_{\mathrm{spec}}$ is so large (e.g., a chain of adders) that make round-off a preferred choice.

The underlying assumption is that the optimal $\mathrm{WL}_t{}^{*}$, the conservative $\mathrm{WL}_{t,\mathrm{conservative}}{}^{*}$, and $\mathrm{WL}_r{}^{*}$ are close to each other in the abstract design space. Most times the loss of optimality of $\mathrm{WL}_{t,\mathrm{conservative}}{}^{*}$ as compared to $\mathrm{WL}_t{}^{*}$ in terms of hardware cost is minimal, since *ceiling* $(1/2 \cdot \log 2(\mathrm{MSE}_t/\mathrm{MSE}_{\mathrm{spec}}))$ is logarithmic to the MSE difference.

Only one additional simulation is introduced in step 2 (for each test vector) during this new procedure. The complete elimination of estimating B matrix means we can only explore the design space suboptimally, but nevertheless the procedure is very time efficient.

An alternative method for the exploration of the truncation mode is to estimate the C vector as if the B matrix does not exist, with each of the N simulations using only truncation mode for the corresponding group. This effectively estimates the sum of the C vector and the B main diagonal entries. This would either overestimate or

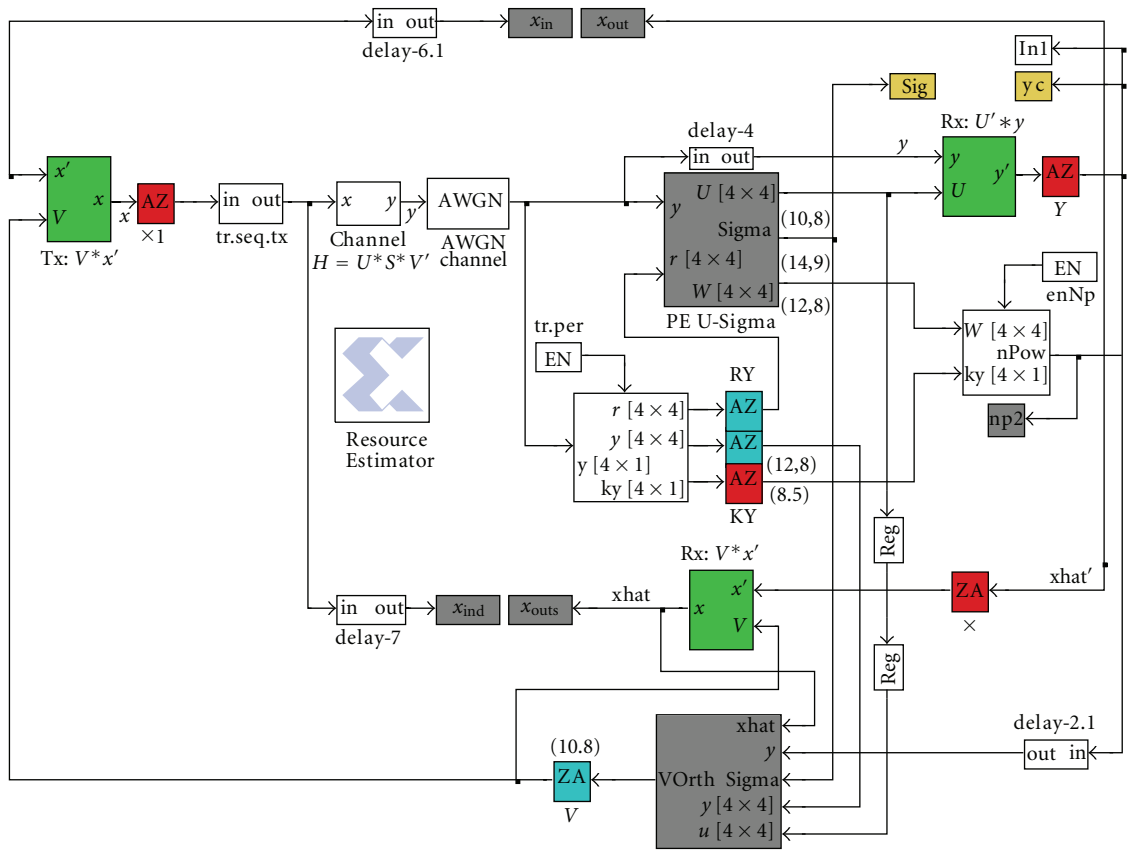


Figure 14: DSP blocks of a MIMO transceiver used to evaluate the SVD algorithm. Key wordlengths are labeled.

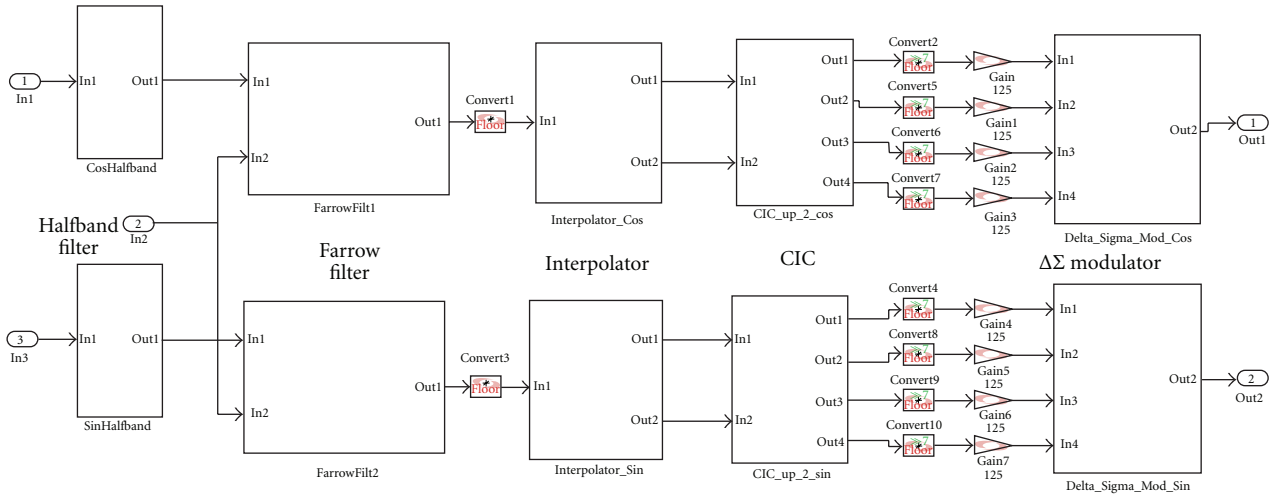


Figure 15: Block-level diagram of a reconfigurable digital front-end (DFE).

underestimate the actual MSE effect, so the tool would need to either uniformly increase or decrease the obtained WL to find $WL_{t,\text{conservative}}{}^{*}$. One additional simulation is also needed here to adjust for the total B matrix from its diagonal.

5.8. Optimization Refinement. The MSE requirement may require a few refinements before arriving at a satisfactory design, but one key advantage of this wordlength optimization tool is its ability to rapidly refine designs without restarting the characterization and simulation process, because both the hardware and MSE cost are modeled as simple functions. In fact, it is now practical to easily explore the tradeoff between hardware cost and MSE performance.

Furthermore, given an optimized design for the specified MSE requirement, the user is then given the opportunity to

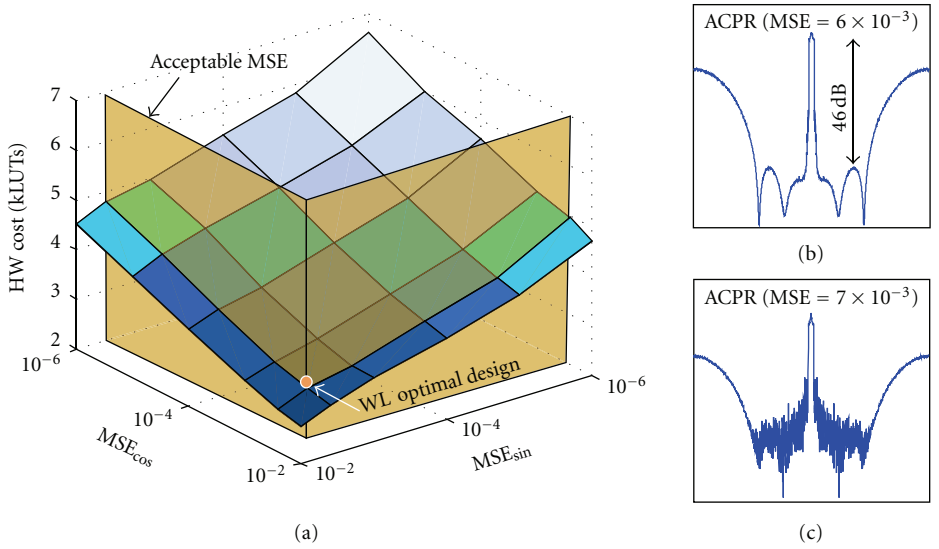


Figure 16: (a) MSE to hardware cost tradeoff for the reconfigurable DFE. (b) ACPR of the wordlength-optimal design at MSE of 6×10^{-3}. (c) ACPR of the design at MSE of 7×10^{-3}.

simulate and examine the design for suitability. If unsatisfied with the result, a new MSE requirement can be entered, and a design optimized for the new MSE is created immediately. This step is still important as the final verification stage of the design to ensure full compliance with all original system specifications.

6. Optimization Results—Examples

A pipelined 6-tap FIR filter in SynDSP is shown in Figure 11 as a simple design example. The wordlength at each logic block before optimization is shown in the top label as (total, fractional), and the wordlength after the optimization is shown in the bottom label. The design is optimized for an MSE of 10^{-6}, and the area savings are greater than 60%. The entire optimization flow for this FIR design is less than 1 minute. For more complex nonlinear systems such as [22, 27, 28], characterization may take overnight, but no intermediate user interaction is required.

Figure 12 shows a design example of a 1/square-root operator in XSG to illustrate optimization of a recursive design. The original design occupies 877 slices, and the optimized design occupies only 409 slices. This demonstrates that the tool is not limited to feedforward designs.

The design of a state-of-the art jitter compensation unit using high-frequency training signal injection [29, 30] is shown in Figure 13(a). Its main blocks include high-pass and low-pass filters, multipliers, and derivative computations. The designer spent many iterations in finding a suitable wordlength, but is still unable to reach a final SNR of 30 dB, as shown in Figure 13(b). This design consumes ~14000 LUTs on a Virtex-5 FPGA. Using the wordlength optimization tool, we finalized on an MSE of 4.5×10^{-9} after a few simple refinements (Section 5.8). Shown in Figure 13(c), the optimized design is able to achieve a final SNR greater than 30 dB while consuming only 9600 LUTs, resulting in 32% savings in area *and* superior performance.

For very complex designs, it is often difficult to perform wordlength optimization on the entire system due to machine and runtime limitations. In this case, the optimization ought to be performed hierarchically. For the MIMO transceiver used to evaluate SVD algorithm (Figure 14) [22], the processing elements for $U\Sigma$ and V are optimized first, and their optimized I/O wordlengths are then propagated to top level to optimize the remaining logic in the top level. This approach made feasible the optimization of a 1 million gate chip. Using automated FPGA mapping from XSG, the designer was able to immediately verify all functional modes of the optimized design in hardware before physical chip synthesis [31], giving designers much higher confidence in the functionality of the fabricated chip. These chips also demonstrate hierarchical extension of the 1/square-root block illustrated in Figure 12, and [31] has more details about the design.

The final detailed example is a high-performance reconfigurable digital front-end for cellular phones (Figure 15). Due to the GHz-range operational frequency required by the transceiver, a high-precision design simply cannot meet the performance requirement. The authors not only explored the possible architectural transformations [32], wordlength optimization was also required to make the performance feasible. Since high-performance designs often synthesize to parallel architectures (e.g., carry look-ahead adder), the wordlength-optimized design results in 40% area savings.

We now explore the tradeoff between MSE and hardware cost, which in this design directly translates to power, area, and timing feasibility. Since this design has two outputs (sine and cosine channels), the MSE at each output can be adjusted independently, shown in Figure 16(a). The adjacent channel power-ratio (ACPR) requirement of 46 dB must be met, which lead to a minimum MSE of 6×10^{-3}. The ACPR of the wordlength optimal design is shown in Figure 16(b). Further wordlength reduction violates ACPR requirement (Figure 16(c)).

Table 1: Summary of wordlength optimization results.

Design	Operation frequency	Gate count	Chip area	Area savings from WL optimization
MIMO SVD [22]	100–512 MHz	420,304	3.5 mm^2 in 90 nm	30%
Sphere decoder [27]	256 MHz	85,000	0.31 mm^2 in 90 nm	20%
Neural DSP [28]	0.4–1.6 MHz	650,000	7.04 mm^2 in 90 nm	15%
Jitter compensation [29]	100 MHz	9,600 LUT	Xilinx Virtex V	32%
Reconfigurable DFE [32]	2.4 GHz	100,000	0.16 mm^2 in 65 nm	40%

We have designed numerous chips across different sizes and operating frequencies using this tool. Due to the length limitation, main results from the selected chips are shown in Table 1.

7. Conclusion and Outlook

This paper discusses the purpose and usage of an automated wordlength optimization tool and its underlying algorithms. Numerous improvements have been made to extend its application to ASIC designs by supporting SynDSP blocksets and ASIC area estimations. With its model-based optimization, it is possible to construct designs for different quantization requirements without manual iteration. The present paper stresses the practicality of the FFC tools and explains the aspects in which it has been improved over its previous version. As before, we encourage readers to download the tool from our public website and try it on their designs. The readers are also encouraged to further refer to [3] for better understanding of the fundamentals of the FFC problem.

FFC research has advanced the field considerably in the past decade or more. Research teams like ours have been enjoying automated FFC on large number of chip designs. However, from what the authors experienced, semiconductor companies who face FFC on a daily basis are still using largely ad hoc and manual methods. This is can be caused by both lack of familiarity with the advanced topics and the resistance to new tools. We are confident that the public availability, and further documentation of our tool will help the industry's adoption of the advanced approach. Tool support groups such as Synopsys, XSG, or even Simulink are the first step toward this realization. Adopting the concepts and techniques in other tool design environments may be less straightforward, but not difficult. Once the tool becomes an integrated and preincluded part of existing tool flow, the semiconductor industry would adopt it more readily.

Due to constant updates in Xilinx and Synopsys blockset, some version compatibility issues may occur, though we aim to provide updates with every major blockset release (support for Synopsys Synphony blockset is recently added).

Disclosure

It is open-source, so feel free to modify it and make suggestions, but please do not use it for commercial purposes without the authors permission.

References

[1] D. A. Patterson and J. L. Hennessy, *Computer Organization & Design: The Hardware/Software Interface*, Morgan Kaufmann, Boston, Mass, USA, 2nd edition, 1997.

[2] D. Marković, V. Stojanović, B. Nikolić, M. A. Horowitz, and R. W. Brodersen, "Methods for true energy-performance optimization," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 8, pp. 1282–1293, 2004.

[3] C. Shi, *Floating-point to fixed-point conversion*, Ph.D. thesis, Department of EECS, University of California, Berkeley, Calif, USA, 2004.

[4] H. Keding, M. Willems, M. Coors et al., "FRIDGE: a fixed-point design and simulation environment," *The Design, Automation, and Test in Europe* , pp. 429–435, 1998.

[5] W. Sung and K. I. Kum, "Simulation-based word-length optimization method for fixed-point digital signal processing systems," *IEEE Transactions on Signal Processing*, vol. 43, no. 12, pp. 3087–3090, 1995.

[6] S. Kim, K. I. I. Kum, and W. Sung, "Fixed-point optimization utility for C and C++ based digital signal processing programs," *IEEE Transactions on Circuits and Systems II*, vol. 45, no. 11, pp. 1455–1464, 1998.

[7] M. Cantin, Y. Savaria, and P. Lavoie, "A comparison of automatic word length optimization procedures," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 612–615, May 2002.

[8] P. Banerjee, "Automatic conversion of floating point MATLAB programs into fixed point FPGA based hardware design," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 263–264, April 2003.

[9] C. Shi and R. W. Brodersen, "An automated floating-point to fixed-point conversion methodology," in *Proceedings of the IEEE International Conference on Accoustics, Speech, and Signal Processing*, pp. 529–532, April 2003.

[10] C. Shi, "Practical, reliable and cost-efficiet floating-point to fixed-point conversion," Qualification Exam, EECS, University of California, Berkeley, Calif, USA, 2002.

[11] S. Roy and P. Banerjee, "Al algorithm for trading off quantization error with hardware resources for MATLAB-based FPGA design," *IEEE Transactions on Computers*, vol. 54, no. 7, pp. 886–896, 2005.

[12] M. L. Chang and S. Hauck, "Precis: a usercentric word-length optimization tool," *IEEE Design & Test of Computers*, vol. 22, no. 4, pp. 349–361, 2005.

[13] L. Zhang, Y. Zhang, and W. Zhou, "Fast trade-off evaluation for digital signal processing systems during wordlength optimization," in *Proceedings of the IEEE/ACM Conference on Computer-Aided Design*, pp. 731–738, November 2009.

[14] C. Shi, *Statistical method for floating-point to fixed-point conversion*, M.S. thesis, Department of EECS, University of California, Berkeley, Calif, USA, 2002.

[15] C. Shi and R. W. Brodersen, "Floating-point to fixed-point conversion with decision errors due to quantization," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 41–44, April 2004.

[16] C. Shi and R. W. Brodersen, "A perturbation theory on statistical quantization effects in fixed-point DSP with non-stationary input," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 373–376, May 2004.

[17] C. Shi and R. W. Brodersen, "Automated fixed-point data-type optimization tool for signal processing and communication systems," in *Proceedings of the Design Automation Conference*, pp. 478–483, San Diego, Calif, USA, June 2004.

[18] G. A. Constantinides, "Perturbation analysis for word-length optimization," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 81–90, April 2003.

[19] J. A. Clarke, G. A. Constantinides, and P. Y. K. Cheung, "Word-length selection for power minimization via non-linear optimization," *ACM Transactions on Design Automation of Electronic Systems*, vol. 14, no. 2, 2009.

[20] G. A. Constantinides, P. Cheung, and W. Luk, "Wordlength optimization for linear digital signal processing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1432–1442, 2003.

[21] G. A. Constantinides, "Word-length optimization for differentiable nonlinear systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 11, no. 1, pp. 26–43, 2006.

[22] D. Markovic, R. W. Brodersen, and B. Nikolić, "A 70GOPS, 34mW multi-carrier MIMO chip in 3.5mm2," in *Proceedings of the The International Symposium on VLSI Circuits, Digest of Technical Papers*, pp. 196–197, June 2006.

[23] C. Shi, J. Hwang, S. McMillan, A. Root, and V. Singh, "A system level resource estimation tool for FPGAs," in *Proceedings of the International Conference on Field Programmable Logics and Its Applications*, pp. 424–433, 2004.

[24] C. C. Wang, "Word-length Optimization for Synplify DSP Blockset with FPGA and ASIC Area-Estimation," EE216B Project with Synopsys University Program, UCLA, 2008.

[25] FFC, http://bwrc.eecs.berkeley.edu/people/grad_students/cchi/research/FFC/documentation.htm, Update tools, http://www.ee.ucla.edu/~dmgroup/optim/WLtool_DSPbook.zip.

[26] C. C. Wang, *Design and optimization of low-power logic*, M.S. thesis, Electrical Engineering Department, UCLA, 2009.

[27] C.-H. Yang and D. Marković, "A flexible DSP architecture for MIMO sphere decoding," *IEEE Transactions on Circuits and Systems I*, vol. 56, no. 10, pp. 2301–2314, 2009.

[28] V. Karkare, S. Gibson, and D. Marković, "A 130-uW, 64-channel spike-sorting DSP chip," in *Proceeding of the IEEE Asian Solid-State Circuits Conference*, pp. 289–292, November 2009.

[29] Z. Towfic, S. K. Ting, and A. Sayed, "Sampling clock Jitter estimation and compensation in ADC circuits," in *Proceeding of the IEEE International Symposium on Circuits and Systems (ISCAS '10)*, pp. 829–832, June 2010.

[30] S. K. Ting and A. Sayed, "Reduction of the effects of spurious PLL tones on A/D converters," in *Proceeding of the IEEE International Symposium on Circuits and Systems (ISCAS '10)*, pp. 3985–3988, June 2010.

[31] D. Marković, B. Nikolić, and R.W. Brodersen, "Power and area minimization of multidimensional signal processing," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 4, pp. 922–934, 2007.

[32] R. Nanda, C. H. Yang, and D. Marković, "DSP architecture optimization in matlab/simulink environment," in *Proceeding of the Internatonal Symposium on VLSI*, p. 192193, June 2008.