Bit-Width Selection for Data-Path Implementations*

Carlos Carreras Juan A. López Octavio Nieto-Taladriz
Dept. Ingeniería Electrónica, E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid
Ciudad Universitaria s/n, 28040 Madrid, Spain
carreras@die.upm.es, juanant@die.upm.es, nieto@die.upm.es

Abstract

Specifications of data computations may not necessarily describe the ranges of the intermediate results that can be generated. However, such information is critical to determine the bit-widths of the resources required for a data-path implementation. In this paper, we present a novel approach based on interval computations that provides, not only guaranteed range estimates that take into account dependencies between variables, but estimates of their probability density functions that can be used when some truncation must be performed due to constraints in the specification. Results show that interval-based estimates are obtained in reasonable times and are more accurate than those provided by independent range computation, thus leading to substantial reductions in area and latency of the corresponding data-path implementation.

1. Introduction

The silicon area and the latency of a data-path implementation depend directly on the bit-widths of its functional, communication and storage resources. While input ranges or bit-widths may be part of the specification, it is usually the case that the designer has to select the bit-widths of the remaining variables. Eventually, when allowed by the specification, this selection may imply range truncations due to area constraints (e.g. fixed-point implementation of a floating-point algorithm).

Computing the exact ranges in a sequence of operations is a complex task when there are dependencies between them. The only general method is the exhaustive simulation of all the possible input vectors. However, this is usually unfeasible due to the size of the input space, $S = \prod_{i=1}^{I} N_i$ vectors, where I is the number of inputs and N_i is the number of possible values of input i assuming integer arithmetic. Estimates of the ranges can still be obtained through partial simulation of the input space, but these estimates are not guaranteed to include all possible results (e.g. worst case), and it must be assumed that the stimuli vectors truly represent the expected input values [8].

Another common approach is to compute the range of each operation from the ranges of its operands. This does not consider the dependencies between the values of the operands. This method is fast and guaranteed, but produces significantly oversized bit-widths [6]. It is used for worst case estimation in the in [9], and it is applied in [5] combined with the partial simulation technique.

This paper presents a novel approach which uses interval computations for bit-width dimensioning in data-path implementations. In particular, we consider the general problem of obtaining estimates of the probability density functions (PDFs) of the intermediate and output variables in a sequence of arithmetic operations from estimates of the input PDFs. Such estimates are represented in the form of histograms where each bar describes the probability mass associated with a particular range of values. Considering PDFs instead of ranges allows evaluating the impact of bit-width truncations, prepares the road for future tools capable of estimating the execution flow in algorithms that include data-dependent control constructs [3].

Interval computations have been used extensively to deal with uncertain data, including methods to find the enclosures of real functions [1, 7]. Approaches that consider PDFs instead of ranges are less frequent [2]. They are aimed to obtain exact probabilistic results in the form of cummulative distribution functions, and use predefined partitions of the input and ouput domains to represent the PDFs as histograms. Instead, we develop approximations that allow the analysis of sequences of dependent operations and provide guaranteed ranges with approximate PDFs of all intermediate and output results. In particular, the contributions in this paper include the formal definition of the interval models used in the computation, a new geometric model that provides several advantages over previous linear models, and

*This work was funded in part by projects ESPRIT 24137 and CICYT TIC97-0928.

the application of the interval method to bit-width dimensioning in data-paths which may result in substantial area and latency reductions for guaranteed implementations.

The following sections introduce the computation method, the interval models, and some other applications of the interval approach. Then, experimental results of range, PDF and bit-width computations are presented, with conclusions following.

2. Basic Computation Method

Considering a sequence of dependent integer arithmetic operations with known independent input PDFs in the form of histograms, the goal is to obtain histogram descriptions of the output PDFs. Histogram descriptions favour an interval approach. It has been mentioned that exhaustive integer simulation is exact but usually unfeasible. On the other hand, independent range computation leads to oversized results because it ignores dependencies. An interval approach fills the gap between these two extremes by providing the means of computing the operations in terms of subranges (bars), thus reducing the size of the input space, and considering dependencies between intervals. Histogram bars are described in terms of the following definition.

Definition 1. An interval $[a, b]/p$ is the set of integers x that verify $a \leq x \leq b$ with an associated probability p uniformly distributed inside the interval, so the probability of any x is $p/(b - a + 1)$.

Considering a uniform PDF inside each interval allows that approximate PDFs can be computed through a sequence of operations. This is different from [2] where nothing is said about the shape of PDFs in the intervals so only bounds for the cummulative distributions can be obtained. In our case, uniformity is an assumption based on the uncertain nature of the input data, where bit-widths are frequently the only available information, and the goal of obtaining only approximate output PDFs.

The basic algorithm to compute histograms is adapted from [2] as:

1. Consider the input space $N_1 \times \ldots \times N_I$ where N_i is the set of intervals describing the histogram of input i.
2. For each vector $(\ldots, [a_{ij}, b_{ij}]/p_{ij}, \ldots)$ of the input space, where $[a_{ij}, b_{ij}]/p_{ij}$ represents the j-th bar of the histogram describing input i:
   (a) Compute its probability $P = \prod_{i=1}^{I} p_{ij}$.
   (b) Execute the operations using interval arithmetic.
   (c) Assign P to each resulting interval.
   (d) Collect the results in output histograms using (*split* and) *merge* operations (see below).

The key approximation for computing sequences of operations is that P is also uniformly distributed in the output intervals. Interval operations are computed from the endpoints of the interval operands [4]. Addition, subtraction, multiplication, division, and exponentiation with constant positive exponents are considered here. Division by 0 is a special case that can occur due to the approximations. In the current implementation it is automatically transformed into a division by 1.

The process that collects results from each input vector into global histograms (2.d) uses the uniformity assumption to collect two overlapping intervals, for example $[a_i, b_i]/p_i$ and $[a_j, b_j]/p_j$ with $a_i < a_j < b_i < b_j$. An exact result generates three disjoint intervals $[a_i, a_j - 1]/(a_j - a_i)p$, $[a_j, b_i]/(b_i - a_j + 1)p$, and $[b_i + 1, b_j]/(b_j - b_i)p$, where $p = p_i/(b_i - a_i + 1) + p_j/(b_j - a_j + 1)$. However, this procedure tends to increase the number of output intervals so the complexity of the collection process can also increase to unfeasible proportions. Instead, a *merge* operation is applied so the overlapping intervals are collected as one interval with probabilities added, $[a_i, b_j]/(p_i + p_j)$.

The problem with this approach is that PDF information is lost in the merge operation and output histograms tend to flatten. One way to reduce this effect is to use predefined bars for the output histograms. First, the meaning of the merge operation is modified so that any two interval results falling inside the same predefined output bar are represented by the minimum interval that contains them with probabilities added, $[min(a_i, a_j), max(b_i, b_j)]/(p_i + p_j)$. This provides a mechanism to control the number of intervals in the output histogram so it corresponds to the number of predefined bars. Second, a *split* operation is applied before the merge when an interval result overlaps several predefined output bars. The split operation divides the original interval into as many subintervals as overlapped bars, with probabilities proportional to the overlaps. This allows maintaining part of the PDF information in the output representation. A similar operation called *proportional assignment* is used in [2] to transform continuous PDFs into user-defined histograms. However, user-defined partitions (bars) of the output histograms are not practical when ranges are unknown, so predefined fixed partitions are preferred here, and the obvious choice is to use predefined intervals (bars) of equal size. This leads to the class of *linear* models described in the next section. However, if the specification includes multiplications or exponentiations, the intervals resulting from the computation will span over many of such equal-sized intervals, so the collection process becomes inefficient. Therefore, new models are required to achieve a good compromise between accuracy and computation times.

3. Complex Interval Models

Interval models are classified here into *simple* and *complex*. A simple model only uses the merge operation for

overlaps, so it is fast but output histograms tend to flatten. Its accuracy with respect to dependencies is controlled through the number of bars used to represent the inputs.

Complex models are based on predefined fixed partitions of the integer axis and use the split and merge operations to represent histograms in terms of the model. A partition of the integer axis is an infinite set of disjoint intervals that we call *maxintervals* whose union gives the axis. A class of complex models is described by its set of maxintervals. A particular model within a class is identified by a grain or granularity g and a center of symmetry c. The value of g is related to the sizes of the maxintervals. A granularity 1 corresponds to the standard integer partition of the axis (one integer per maxinterval) so, in general, g takes integer values greater than 1. The center c is always an interval with only one integer, so using a center $[c, c]$ different from the origin, $[0, 0]$, provides greater accuracy around c in the PDF. Therefore, the accuracy of a complex model is tuned through its class, grain and center. The two classes of complex models used here are based on the following definitions.

Definition 2. The class of *linear* interval models has maxintervals $[A, B]$, uniquely identified by integer n, that verify one of the following identities:

$$[A,B] = \begin{cases} [c+gn+1, c+g(n+1)] & (n<-1) \\ [c-g+1, c-1] & (n=-1) \\ [c,c] & (n=0) \\ [c+1, c+g-1] & (n=1) \\ [c+g(n-1), c+gn-1] & (n>1) \end{cases}$$

Linear models provide intervals of equal size (except around the center) and are recommended for sequences of additions and subtractions. However, it has been mentioned that multiplications and exponentiations can make the collection process inefficient. So a new *geometric* class of models is defined. It describes a partition using maxintervals with increasing exponential sizes as they separate from the center of symmetry.

Definition 3. The class of *geometric* interval models has maxintervals $[A, B]$, uniquely identified by integer n, that verify one of the following identities:

$$[A,B] = \begin{cases} \left[c-g^{-n}+1, c-g^{-(n+1)}\right] & (n<0) \\ [c,c] & (n=0) \\ \left[c+g^{(n-1)}, c+g^{n}-1\right] & (n>0) \end{cases}$$

Geometric models allow for large reductions in the size of the input space and compensate for the range expansion produced by multiplications and exponentiations, at the cost of coarser probabilistic descriptions for values away from the center. It should be noticed that the model with grain 2 and center 0 is closely related to the binary representation of numbers and, thus, to bit-widths.

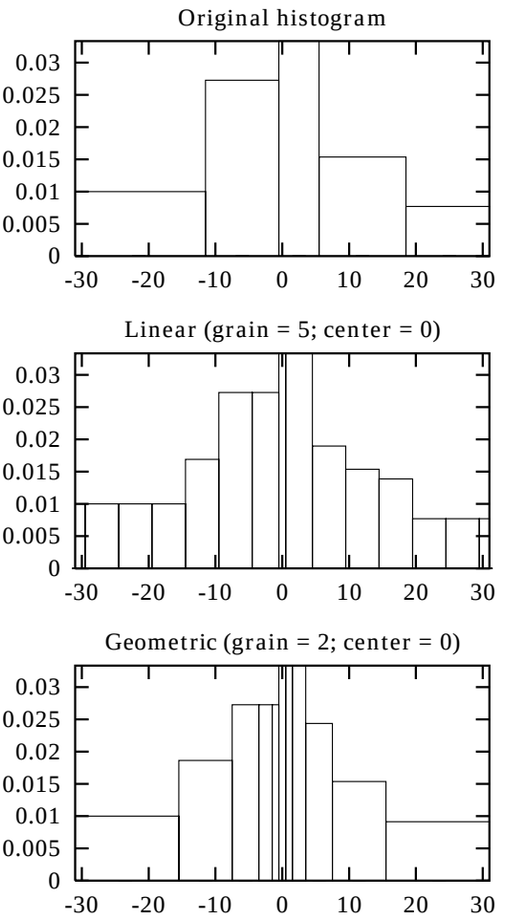

Figure 1. Linear and geometric models

Complex models with intervals bounded by maxintervals facilitate the automatization of the estimation process (the user selects a model instead of defining a custom axis partition), and avoid in part the flattening that appears in simple models, thus gaining accuracy at the cost of some more computational complexity. So they provide clear advantages when input space sizea are dominated by large ranges.

Figure 1 contains examples of representations using a linear and a geometric model. It can be observed that both models, but specially the geometric one, are more accurate around the center of symmetry than in the rest of the axis.

4. Other Applications of the Interval Approach

When the number of inputs is also an important contributor to the large size of the input space, interval methods have a limited impact on the input space reduction. In this case, the overall simulation time can still be reduced by avoiding redundant computations. The goal is to identify

independent blocks of operations that can be computed separately. When despite this strategy exhaustive simulation is still unfeasible even with large granularities in the models, the partitioning into blocks can be applied further even if the blocks are not totally independent. Of course, this will cause oversized results, but the bounds are still guaranteed. The extreme case is represented by independent range computation, where each operation is treated like a block and all dependencies are ignored.

Another approach to deal with a situation where exhaustive simulation is unfeasible is to use partial simulation to obtain hints for bit-width dimensioning, although output ranges will not be guaranteed bounds in this case. Standard partial simulation of the integer input space is based on random generation of the input vectors to obtain statistical parameters of the results. When using complex interval models, the random generation strategy cannot be used because (1) maxintervals have different sizes (particularly in the geometric model) and randomness is not easily achieved, and (2) the number of values involved is greatly reduced so precise statistical parameters cannot be inferred from the results. However, complex models allow a different strategy with more advantages than random input generation. In [4], the intervals representing each input histogram (a particular case of the geometric model defined here) are sorted with decreasing probability masses while input vectors are generated in an ordered fashion. This causes that the input vectors with the highest probabilities are simulated first, so the interval results tend towards their true values faster than the integer results with random input generation. Consequently, interval simulation yields smaller errors than integer random simulation when considering a fixed simulation time. It is clear that this sorting is prohibitive in computation times when integers are considered.

5. Experiments and Results

The goal of the following experiments is to compare ranges and PDFs obtained from exhaustive integer simulation, independent range computation, and interval simulation with linear and geometric models. Although it is possible to assign a different model to each variable in a computation, the next simulations use a common model for all the variables so the results can be easily interpreted.

The first experiment evaluates the errors in ranges and PDFs with respect to the exact results, called reference, obtained from exhaustive simulation of the integer input space.

If R_r is the reference range which includes n_r integers, and R_a is the approximate range with n_a integers, it is verified that $R_r \subset R_a$, and the range error is computed as $\epsilon_r = (n_a - n_r)/n_r$.

Two types of errors are defined to evaluate PDF results. The first one, ϵ_m, represents the difference of bar probability masses between the exact and approximate PDFs when compared in terms of the particular interval model. It is computed with the following algorithm:

1. Represent the reference in terms of the interval model as histogram $h1$.
2. Multiply by -1 the bar probabilities of the histogram obtained using the interval model to build a histogram $h2$ with negative bars.
3. Merge the intervals of $h1$ and $h2$ to obtain a difference histogram $h3$.
4. Compute $\epsilon_m = 1/2 \sum_N \mid prob \mid$, where N is the set of model maxintervals and *prob* is the probability of each bar of $h3$.

The factor $1/2$ in step 4 accounts for the fact that each misplaced result causes a difference in the histograms of twice its probability. The relative error introduced by independent range computation is obtained similarly but overall ranges are considered instead of the N maxintervals. The previous algorithm compares histograms in terms of the probability of each output bar, so ϵ_m is relative to the model under consideration and values from different models cannot be compared.

The other type of PDF error, ϵ_i, is defined in order to compare results from different models. It is computed using a similar algorithm, but instead of representing the reference in terms of the interval model, the histogram obtained using the interval model is represented in terms of the reference (individual integer values) before obtaining the difference histogram $h3$. Of course, this yields large errors because the interval models are not expected to be accurate at the level of individual integers, but allows comparison across models. It should be noted that in the case of independent range computation $\epsilon_m = \epsilon_i$.

The benchmarks used in this experiment are the following simple arithmetic expressions frequently used in data computations.

$$\begin{aligned} f_1 &= a(a+b) \\ f_2 &= (a+b)(a-b) \\ f_3 &= K(a-b)+ab \\ f_4 &= (a+b)^2 - ab \end{aligned}$$

Moderate integer input spaces must be considered so exhaustive simulations to obtain the references can be performed in reasonable times. Expressions with only two inputs are used, so ranges are the dominant factor in the input space sizes and results are not masked by the effect of more inputs. 8-bit inputs uniformly distributed in the range $[-128, 127]$ are used. The constant K in f_3 is also set to 128.

Output results are collected in table 1. Intermediate results are not included due to space limitations. Model cen-

$f_{i,g}$	Range	ϵ_r	ϵ_m	ϵ_i	Time
$f_{1,ref}$	$[-4096, 32768]$	-	-	-	99.6
$f_{1,10}$	$[-4692, 32768]$	0.02	0.14	0.57	18.8
$f_{1,100}$	$[-12573, 32768]$	0.23	0.29	0.67	0.4
$f_{1,2}$	$[-8128, 32768]$	0.11	0.08	0.60	0.4
$f_{1,4}$	$[-8128, 32768]$	0.11	0.12	0.61	0.1
$f_{1,ind}$	$[-32512, 32768]$	0.77	0.83	0.83	0.0
$f_{2,ref}$	$[-16384, 16384]$	-	-	-	127.5
$f_{2,10}$	$[-17399, 17399]$	0.06	0.18	0.64	31.5
$f_{2,100}$	$[-28829, 28829]$	0.76	0.34	0.77	0.6
$f_{2,2}$	$[-18336, 18336]$	0.12	0.16	0.68	0.6
$f_{2,4}$	$[-21392, 21392]$	0.31	0.13	0.70	0.2
$f_{2,ind}$	$[-65280, 65280]$	2.98	0.93	0.93	0.00
$f_{3,ref}$	$[-48896, 16384]$	-	-	-	179.6
$f_{3,10}$	$[-48896, 18416]$	0.03	0.19	0.62	70.4
$f_{3,100}$	$[-48896, 28956]$	0.19	0.25	0.70	1.2
$f_{3,2}$	$[-48896, 28416]$	0.18	0.16	0.66	0.7
$f_{3,4}$	$[-48896, 28544]$	0.19	0.06	0.67	0.2
$f_{3,ind}$	$[-48896, 49024]$	0.50	0.82	0.82	0.0
$f_{4,ref}$	$[0, 49152]$	-	-	-	180.5
$f_{4,10}$	$[-77, 51136]$	0.04	0.24	0.79	88.6
$f_{4,100}$	$[-9797, 55536]$	0.33	0.29	0.83	1.8
$f_{4,2}$	$[0, 60420]$	0.23	0.05	0.80	0.8
$f_{4,4}$	$[-2945, 61440]$	0.31	0.10	0.83	0.2
$f_{4,ind}$	$[-16384, 81920]$	1.00	0.93	0.93	0.0

Table 1. Range and PDF errors

ters are always $[0, 0]$, so models are identified by their granularities, g, which are included as an index in the name of the benchmark, $f_{i,g}$. In the table, granularities 10 and 100 correspond to linear models, while 2 and 4 correspond to geometric models. The words *ref* and *ind* are used to identify exhaustive integer simulation and independent range computation respectively. CPU times are expressed in seconds. They are given only for relative comparison since new developments in the tool are showing that these values can be substantially reduced.

It is observed that independent range computation generates oversized ranges that can be very costly in data-path implementations. Although the impact on bit-widths is relatively small when these expressions are evaluated alone, it is clear that the differences can increase when these expressions are part of more complex computations. Results also show that linear models have small range errors if g is also small, but they are inefficient during the collection process due to multiplications and exponentiations, leading to long simulation times. Errors increase substantially if larger granularities are used. On the other hand, geometric models provide great reductions in computation times while preserving some of the features of the output PDFs, despite the approximations in the split and merge operations and the uniformity assumption. This PDF information can be used to evaluate the impact of truncations. For example, the range [0,60420] provided by $f_{4,2}$ requires 16 bits. If a truncation to 15 bits (range [0,32767]) is performed, the PDF of $f_{4,2}$ indicates that the probability mass being discarded is 0.064, which is an acceptable approximation of the exact 0.035 obtained from the PDF of $f_{4,ref}$.

The next experiment compares the interval approach based on geometric models (grains 2, 3 and 4) with the independent range computation method. In this case, larger input spaces with more inputs are evaluated since no exact reference is obtained and only range reductions are analyzed. The following expressions are estimated.

$$
\begin{aligned}
f_5 &= (a+b)^2/ab \\
f_6 &= (a-b)(b-c)(c-a)/K \\
f_7 &= (a+b+c+d)(a+b-c-d)+c(a+b) \\
f_8 &= (ab+c)(ad+e)/a(b^2+d^2)
\end{aligned}
$$

Table 2 describes the input ranges used in the computation and the corresponding integer input space sizes. The interval input space sizes are presented in table 3. All these sizes are given in vectors and provide an indication of the simulation time and storage requirements for each method. The value of K in f_6 is set to 2048^2.

f	Inputs	Input Ranges	Integer Vectors
f_5	2	[-65536,65535]	1.7×10^{10}
f_6	3	[-2048,2047]	6.8×10^{10}
f_7	4	[-512,511]	1.1×10^{12}
f_8	5	[-16,15]	3.3×10^{7}

Table 2. Input characteristics

f	$g=2$	$g=3$	$g=4$
f_5	1156	529	324
f_6	13824	3375	2197
f_7	8000	2197	1331
f_8	100000	16807	7776

Table 3. Input space sizes

Again, uniform distributions in the input ranges are assumed. As the number of inputs increases, smaller reductions of the input space size can be achieved and larger granularities must be used. An alternative approach would be to divide the algorithm into blocks to be computed independently as mentioned in the previous section.

The simulation results are shown in table 4. Independent range computation produces oversized results in all cases. In particular, f_8 shows very significant differences due to the fact that a division is included in the expression, as divisions can cause large errors when the divisor includes small values around 1 (the division does not reduce the range of the dividend while, in reality, reductions might occur due to dependencies). In f_7, granularity 4 provides better results than granularity 3. This situation is not unique and has appeared in some other tests showing that, in some cases, there are granularities that seem more appropriate to estimate a

$f_{i,g}$	Range	Time
$f_{5,2}$	$[-131064, 4.295 \times 10^9]$	3.9
$f_{5,3}$	$[-177135, 4.295 \times 10^9]$	1.7
$f_{5,4}$	$[-262128, 4.295 \times 10^9]$	1.0
$f_{5,ind}$	$[-1.718 \times 10^{10}, 1.718 \times 10^{10}]$	0.0
$f_{6,2}$	$[-4603, 4603]$	81.9
$f_{6,3}$	$[-5273, 5273]$	20.6
$f_{6,4}$	$[-5371, 5371]$	11.9
$f_{6,ind}$	$[-16372, 16372]$	0.0
$f_{7,2}$	$[-1.766 \times 10^6, 2.159 \times 10^6]$	1365.7
$f_{7,3}$	$[-1.808 \times 10^6, 2.195 \times 10^6]$	237.9
$f_{7,4}$	$[-1.768 \times 10^6, 2.161 \times 10^6]$	110.9
$f_{7,ind}$	$[-4.713 \times 10^6, 4.715 \times 10^6]$	0.0
$f_{8,2}$	$[-272, 272]$	749.9
$f_{8,3}$	$[-320, 320]$	124,4
$f_{8,4}$	$[-465, 465]$	52.6
$f_{8,ind}$	$[-73441, 73441]$	0.0

Table 4. Geometric models vs. ranges

particular expression. So, in general, it is recommended to try several grains in the estimation process.

In the last experiment, the specification of a block from a phase equalizer used in communication circuits (see table 5) is synthesized with no optimization. The goal is to evaluate the impact of bit-width estimates from (1) independent range computation and (2) the interval method with a geometric model of grain 2 as they evolve in a specification. Table 5 shows the bit-widths estimates of some of the variables involved when the inputs are modeled as $x_i = x + \Delta_i$, where $x \in [-13, 12]$ and $\Delta_i \in [-3, 3]$, and $K = 16$. The results from the synthesis show that using

Block	Ind.	$g = 2$
$f_1 = (x_1^2 + x_2^2 + x_3^2)/K$	8	8
$f_2 = (x_1x_2 + x_2x_3 + x_3x_4)/K$	8	8
$f_3 = (x_1x_3 + x_2x_4 + x_3x_5)/K$	8	8
$g_1 = (f_1^2 - f_2^2)/K$	10	10
$g_2 = f_2(f_3 - f_1)/K$	11	9
$g_3 = (f_2^2 - f_1f_3)/K$	11	9
$g_4 = (f_1^2 - f_3^2)/K$	10	10
$h_1 = (g_1x_3 + g_2x_2 + g_3x_1)/K$	12	11
$h_2 = (g_2x_3 + g_4x_2 + g_2x_1)/K$	11	10
$h_3 = (g_3x_3 + g_2x_2 + g_1x_1)/K$	11	10
$z_1 = (h_3x_2 - h_2x_3 - h_1x_4)/K$	14	12
$z_2 = (-h_3x_3 - h_2x_4 - h_1c_5)/K$	14	12

Table 5. Expressions and bit-width estimates

the estimates from the interval method allows a 7.6% area reduction and a 8.8% latency reduction in the implementation. While these numbers can be different when considering other inputs or other specifications, it is clear that estimates from interval methods can improve significantly the synthesized data-path.

6. Conclusions

We have presented an interval method for fast estimation of ranges and histogram PDFs of arithmetic functions. It is based on the definition of specific interval models, including new geometric models that reduce the complexity of the computation while keeping the errors in the estimates within acceptable margins. The interval method computes guaranteed bounds for intermediate and output ranges much more accurate than those obtained from the independent range computation of each operation. PDF information is also provided and can be used when evaluating the impact of range truncations. On the other hand, computation times are significantly reduced with respect to times required for exhaustive simulation using integers. Further research in the application of the interval method in conjuntion with the partitioning of the arithmetic specification is proposed to achieve even shorter simulation times. Results show that bit-width selections based on the range estimates provided by this interval method can clearly improve the area and the latency of data-path implementations with respect to those based on independent range computations.

References

[1] N. Asaithambi, S. Zuhe, and R. Moore. On Computing the Range of Values. *Computing*, 28:225–237, 1982.

[2] D. Berleant. Automatically Verified Reasoning with Both Intervals and Probability Density Functions. *Interval Computations*, 1993(2):48–70, 1993.

[3] C. Carreras, J. C. López, M. L. López, C. Delgado-Kloos, N. Martínez, and L. Sánchez. A Co-Design Methodology Based on Formal Specification and High-Level Estimation. In *Proc. Workshop on HW/SW Co-Design*, 1996.

[4] C. Carreras, I. Walker, O. Nieto, and J. Cavallaro. Robot Reliability Estimation Using Interval Methods. In *MISC'99 International Workshop on Applications of Interval Analysis to Systems and Control*, Girona, Spain, Feb 1999.

[5] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens. A Methodology and Design Environment for DSP ASIC Fixed Point Refinement. In *Proceedings of the DATE*, pages 271–276, Paris, 1999.

[6] Eero Hyvonen. Evaluation of Cascaded Interval Functions. In *Proceedings of Intervational Workshop on Constrain-Based Reasoning, 8th Florida AI Research Symposium*, April 1995.

[7] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions*. Ellis-Horwood, Chichester, 1988.

[8] W. Sung and K. Kum. Simulation-Based Word-Length Optimization Method for Fixed-Point Digital Signal Processing Systems. *IEEE Transactions on Signal Processing*, 43:3087–3090, 1995.

[9] M. Willems, V. Bursgens, H. Keding, T. Grotker, and H. Meyr. System Level Fixed-Point Design Based on an Interpolative Approach. In *Proceedings of the DAC*, pages 293–298, Anaheim, CA, 1997.