Accuracy Guaranteed Bit-Width Optimization

Dong-U Lee, *Member, IEEE*, Altaf Abdul Gaffar, *Member, IEEE*, Ray C.C. Cheung, *Student Member, IEEE*, Oskar Mencer, *Member, IEEE*, Wayne Luk, *Member, IEEE*, and George A. Constantinides, *Member, IEEE*

***Abstract*— We present MiniBit, an automated static approach for optimizing bit-widths of fixed-point feedforward designs with guaranteed accuracy. Methods to minimize both the integer and fraction parts of fixed-point signals with the aim of minimizing circuit area are described. For range analysis, our technique identifies the number of integer bits necessary to meet range requirements. For precision analysis, we employ a semi-analytical approach with analytical error models in conjunction with adaptive simulated annealing to optimize the number of fraction bits. The analytical models enable us to guarantee overflow/underflow protection and numerical accuracy for all inputs over the user-specified input intervals. Using ASC, A Stream Compiler for field-programmable gate arrays (FPGAs), we demonstrate our approach with polynomial approximation, RGB to YCbCr conversion, matrix multiplication, B-splines and discrete cosine transform placed-and-routed on a Xilinx Virtex-4 FPGA. Improvements for a given design reduce area and latency by up to 26% and 12% respectively, over a design using optimum uniform fraction bit-widths. Studies show that MiniBit optimized designs are within 1% of the area produced from integer linear programming approach.**

***Index Terms*— Field programmable gate arrays, finite wordlength effects, fixed point arithmetic, optimization methods, simulated annealing.**

I. INTRODUCTION

ONE of the main objectives of hardware designers is to find the optimal design in terms of area, latency, throughput and power consumption. Bit-widths of signals are one of the parameters that designers can tweak to improve these metrics. In contrast to instruction processors, customizable hardware such as field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs) provide the freedom for bit-widths optimized for a given application. However, hardware designers face increasing difficulties choosing the best bit-widths. The objective is to find the minimal number of bits to represent a signal, while satisfying user-defined error constraints. A naive way to optimize bit-widths is to manually evaluate various combinations and observe the output for each design. This technique, however, involves an enormous search space and is not practical for large designs.

Bit-width optimization is an NP-hard problem [1] and has been the focus of numerous research contributions, especially over the past few years. The work in this area can be classified in many different ways, and one such classification is static analysis versus dynamic analysis. Dynamic analysis [2]–[7] relies on the use of stimuli input signals. Though this approach provides bit-widths closer to the optimal set for those particular stimuli when compared to static analysis techniques, it can be problematic since a large set of stimuli signals is required to analyze a design with sufficient confidence, possibly leading to prohibitively long simulation times and without guarantees for alternative input stimuli encountered in practice. Static analysis [8]–[12] is believed to give more conservative bit-width estimates than dynamic analysis. Static analysis is often more attractive than dynamic analysis especially for large designs, since only the characteristics of the input signals are needed. In this work, we adopt a static analysis technique based on affine arithmetic [13] and analytical error models to optimize both ranges and precisions for the signals in a fixed-point design.

Another way of classifying bit-width optimization involves an error metric. Most existing work is based on the signal to noise ratio (SNR) error criterion. The SNR criterion is popular with digital signal processing applications. On the other hand, many computer arithmetic and scientific applications require a maximum absolute error bound. This error metric is good for portability, especially when a module needs to be integrated into a larger design. Our criterion for evaluating the accuracy is the unit in the last place (ulp). The ulp of a fixed-point number with eight bits of fraction bit-width would be 2^{-8}. Faithful rounding means that results are accurate to 1 ulp (rounded to the nearest or the next nearest) and exact rounding means that results are accurate to 0.5 ulp (rounded to the nearest). Exact rounding is difficult to achieve, due to a problem known as the Table maker's dilemma [14] and has a large area penalty [15], hence we opt for faithful rounding in this initial work. Therefore, if the result has eight fraction bits, our approach guarantees a maximum absolute error of less than or equal to 2^{-8}.

The main contributions of this paper are:

- Analytical range and uniform fraction bit-width determination, based on affine arithmetic models introduced in [9].
- Multiple fraction bit-width determination via adaptive simulated annealing, using error models and area cost functions with guaranteed maximum absolute error bounds.
- Demonstration of our approach with five case studies: polynomial approximation, RGB to YCbCr conversion, matrix multiplication, B-splines and, DCT realized in a Xilinx Virtex-4 FPGA.
- The only reported static bit-width optimization technique that can guarantee 1 ulp maximum absolute error bound.

Manuscript received June 30, 2005; revised October 4, 2005; accepted November 5, 2005. This paper was recommended by the Associate Editor Raul Camposano.

D. Lee is with the Electrical Engineering Department, University of California, Los Angeles, USA. (Email: dongu@icsl.ucla.edu).

A. Abdul Gaffar and G.A. Constantinides are with the Department of Electrical and Electronic Engineering, Imperial College London, United Kingdom (Email: {altaf.gaffar, g.constantinides}@imperial.ac.uk.)

R.C.C Cheung, O. Mencer and W. Luk are with the Department of Computing, Imperial College London, United Kingdom (Email: {r.cheung, o.mencer, w.luk}@imperial.ac.uk).

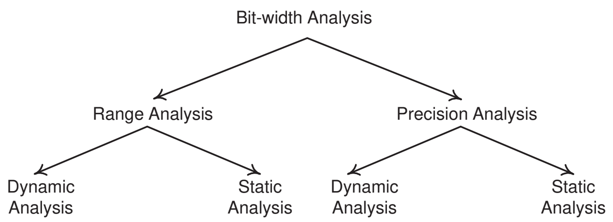


Fig. 1. Classification of bit-width analysis

The rest of this paper is organized as follows. Section II discusses background material and related work. Section III presents an overview of our MiniBit bit-width optimization approach. Section IV introduces affine arithmetic. Section V and Section VI presents our range analysis and precision analysis steps. Section VII describes how our bit-width analysis can be extended to cover resource sharing situations. Section VIII and Section IX present our case studies and their results with MiniBit. Section X gives conclusions and future work.

II. Background

As shown in Fig. 1, the problem of optimizing the design bit-widths can be split into two parts: range analysis and precision analysis.

Range analysis involves studying the data range of the computation and ensuring that the signals in the design have enough bits to accommodate this range. *Precision analysis* involves analyzing the sensitivity of the output from a computation to slight changes in the bit-widths. More specifically, the sensitivity of an output to the computational precision within an arithmetic unit. For both range and precision analysis, we can apply a dynamic or a static analysis method.

Dynamic analysis methods evaluate the data flow graph (DFG) of the design using input stimuli signals. However, *static analysis* methods propagate static characteristics of the inputs through the DFG and hence no input stimuli are required. The input data dependence of dynamic analysis and the input data independence of static analysis have consequences on the results of the bit-width analysis.

As discussed next in the review of existing work in bit-width optimization, both these methods have certain advantages and disadvantages over each other.

A. Existing Work

A large body of work has been produced by Sung *et al.* [4], [5] which uses simulation-based techniques for range and precision bit-width optimization. Their approach involves examining the mean and the standard deviations of the signals. Signals are grouped together during optimization to reduce simulation time. One of the drawbacks of this approach is that there is a danger of overflows for rare events.

The FRIDGE [7] project provides a bit-width optimization system for hardware/software co-design. FRIDGE uses interval arithmetic-based range propagation techniques for the range optimization and a simulation-based approach for the precision optimization. This approach relies on bit-width specification of some of the signals by the user, and derives the remaining signal bit-widths through what the authors describe as interpolation. In order to speed up the simulation time, the authors convert the hardware designs into integer based ANSI C descriptions before simulation.

In [3], Cmar *et al.* use interval arithmetic for range bit-width determination and a dynamic method based on simulation for precision bit-width optimization. The simulation operates by simultaneously performing the same calculation in a reference floating-point and a custom fixed-point format, and comparing the error between the two values. A heuristic is employed where the mean and the standard deviation of the error at each signal are examined for determining the precision bit-widths.

In [6] Shi and Brodersen describe a statistical modeling method based on perturbation theory. For bit-width optimization, the method is designed to target optimization of algorithms used in communications applications. One feature of this approach is that it is data dependent due to its use of statistical modeling.

Abdul Gaffar *et al.* [2] use a mathematical technique known as automatic differentiation to perform precision bit-width optimization for both fixed-point and floating-point designs. Automatic differentiation is used to monitor the sensitivity of the output signals with respect to the intermediate signals. The proposed method requires less simulation time than conventional dynamic approaches.

Nayak *et al.* [16] describe a data range propagation method designed for the MATCH [17] system, which converts Matlab designs to FPGA design descriptions. Their range optimization relies on data range propagation, while precisions are optimized by forward propagation of errors through the data flow graph. A set of error transfer functions are proposed to determine the error contribution of each node.

Constantinides *et al.* propose Synoptix [8], an optimization technique targeting linear time invariant digital signal processing systems using a novel resource binding technique. Synoptix uses a technique based on saturation arithmetic to perform the range bit-width optimization. In recent work [18] the proposed approach is extended to cover non-linear systems, but this extension requires input stimuli to operate.

Fang *et al.* [9], [10] employ affine arithmetic for modeling range and precision analysis. While the use of affine arithmetic to model precision error is demonstrated, the authors do not use it to optimize the actual bit-widths.

B. Discussion

In summary, for range analysis, most of the existing work considered use static analysis [3], [7]–[9], [16] with a few using dynamic analysis [2], [4]–[6]. For precision analysis, dynamic methods are employed by [2]–[7], while static methods are employed by [8], [9], [16]. Static analysis as we show in this paper is capable of providing guaranteed analytical bounds on the output error.

Similar to Fang *et al.*, our proposed technique, MiniBit, uses static range analysis based on affine arithmetic. However, we go a step further and tackle the problem of precision optimization problem as well. Our static precision optimization

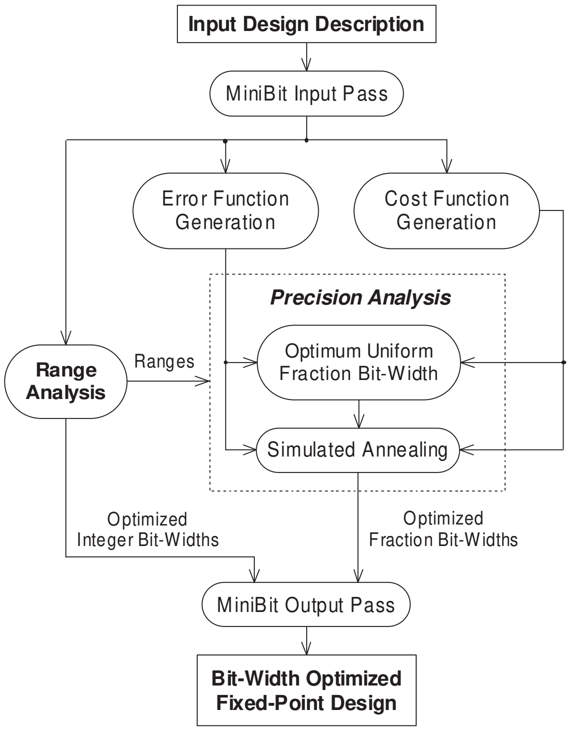


Fig. 2. Overview of the MiniBit automated bit-width optimization approach.

approach is able to guarantee a maximum absolute error bound analytically, which is what differentiates our work with the studies discussed in this section.

III. Overview

An overview of the MiniBit bit-width optimization framework is given in Fig. 2. MiniBit targets hardware designs using fixed-point representation. Fixed-point is often preferred over floating-point for hardware designs involving reasonable dynamic ranges due to its area and speed advantages. In fixed-point representation, a real number is represented by two parts: an integer part which represents the range, and a fraction part which represents the precision. Two's complement representation is assumed. The range of a signal x with IB_x integer bits and FB_x fraction bits is given by $[-(2^{IB_x-1} - 2^{-FB_x} + 1), 2^{IB_x-1} - 2^{-FB_x}]$.

We divide the bit-width optimization problem into two tasks: range analysis and precision analysis. Range analysis involves determining the integer bit-widths (IBs) and precision analysis involves the determination of the required fraction bit-widths (FBs) of fixed-point signals. Our approach is implemented as a series of compilation passes inside MiniBit, which is built on top of the BitSize bit-width analysis system [2]. The input to MiniBit is a design description in ASC [19], C/C++ or Xilinx System Generator [20]. The MiniBit input pass uses this design description together with user-supplied information including the output error specification and the range of the input of values to perform range and precision analysis.

We first perform range analysis, then pass the range results to the precision analysis phase. Precision analysis requires an error function and a cost function. The error function captures the output error as a function of the bit-width of the signals of the design. The cost function returns the area cost as a function of the signal bit-widths and their arithmetic operators.

Range analysis is performed via standard affine arithmetic. Precision analysis operates in two phases: (1) using the error function generated by MiniBit, we analytically find the optimum uniform fraction bit-width (UFB), which means the fraction bit-width for all signals are the same. The UFB serves as the initial set of parameters for the next phase. (2) we use both the error and cost functions to find the optimum multiple fraction bit-widths (MFBs), which in contrast to UFB means that the fraction bit-widths of the signals can be different. MFBs aim at minimizing the area cost function, while meeting the constraints of the error function. The MFBs are found by using adaptive simulated annealing (ASA) [21], [22].

Once the signals have been quantized, the ranges found in the range analysis phase will slightly differ due to finite precision effects. Hence, range is a function of precision. However, as it will be shown in Section VI, precision is a function of range. Since the actual range can marginally change after quantization, the range assumed during the precision analysis phase can no longer be guaranteed to be perfectly accurate. In combination, these factors could in theory lead to increased IB requirements and/or increased FB requirements. Both of these potential problems can be addressed by using more conservative range estimates. However, these problems are highly unlikely to occur since 1) only the ranges that are very close to a power of two can cause larger IB requirements, and 2) due to the conservative nature of the precision analysis (which assumes maximum quantization errors can happen at all signals concurrently), the slight inaccuracy in the range will have negligible impact. For the designs covered in this work, we have not encountered such problems.

Having found the optimized IBs and FBs to each signal, the MiniBit output pass compares the outputs produced from our bit-width optimized fixed-point design against the outputs produced from a software verification model. This step verifies that overflows/underflows do not occur and the user-specified error requirements are met. We finally synthesize and place-and-route the design to target technologies such as FPGAs or ASICs.

IV. Affine Arithmetic

Interval arithmetic (IA) [23] was invented in the 1960s by Moore to solve range problems, where each signal is represented by its interval. A signal x is represented by the interval $\bar{x} = [x_{min}, x_{max}]$, meaning that the true value of x lies between x_{min} and x_{max}. Thus, for instance, the difference of two intervals $\bar{x}$ and $\bar{y}$ is expressed as

$$\bar{x} - \bar{y} \quad = \quad [x_{min} - y_{max}, x_{max} - y_{min}]$$

The main disadvantage of IA is the assumption that all values of the arguments vary independently over the given intervals, potentially leading to drastic overestimation of the true range. As an extreme example, when evaluating the expression $x-x$, we get the interval $\bar{x}-\bar{x} = [x_{min}-x_{max}, x_{max}-x_{min}]$, which is twice as wide as the original interval $\bar{x}$, instead of

$[0,0]$, which is the true range. This overestimation effect can accumulate along the computation chain, resulting in an "error explosion".

Affine arithmetic (AA) [13] is a recent refinement to IA to address this problem. AA captures all of the features of IA with one significant improvement: it keeps track of correlations among intervals. In AA, the uncertainty of a signal x is represented by an affine form $\hat{x}$, which is a first degree polynomial

$$\hat{x} \quad = \quad x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots + x_n\varepsilon_n \quad \text{where } \varepsilon_i = [-1,1]$$

Each ε_i is an independent uncertainty source that contributes to the total uncertainty of the signal x. An ordinary IA interval $\bar{x} = [x_{min}, x_{max}]$ can be converted into an equivalent affine form $\hat{x} = x_0 + x_1\varepsilon_1$ with

$$x_0 = \frac{x_{max}+x_{min}}{2} \quad x_1 = \frac{x_{max}-x_{min}}{2}$$

The key feature of AA is that the sample noise symbol ε_i can contribute to the uncertainty of other signals in the computation chain, keeping correlations between them. Returning to the previous example where IA overestimated, if x has the affine form $\hat{x} = x_0 + x_1\varepsilon_1$ then $\hat{x} - \hat{x} = 0$, which is the correct result.

In affine arithmetic form, we write 1) addition/subraction, 2) constant multiplication and 3) addition/subraction with a constant as

$$\begin{aligned} \hat{x} \pm \hat{y} &= (x_0 \pm y_0) + \sum_{i=1}^{n}(x_i \pm y_i)\varepsilon_i \\ c\hat{x} &= (cx_0) + \sum_{i=1}^{n}(cx_i)\varepsilon_i \\ \hat{x} \pm c &= (x_0 \pm c) + \sum_{i=1}^{n} x_i\varepsilon_i \end{aligned}$$

For multiplication, we get

$$\begin{aligned} \hat{x}\hat{y} &= (x_0 + \sum_{i=1}^{n} x_i\varepsilon_i)(y_0 + \sum_{i=1}^{n} y_i\varepsilon_i) \\ &= x_0y_0 + \sum_{i=1}^{n}(x_0y_i + y_0x_i)\varepsilon_i + Q \\ &\quad \text{where} \quad Q = (\sum_{i=1}^{n} x_i\varepsilon_i)(\sum_{i=1}^{n} y_i\varepsilon_i) \end{aligned}$$

The equation above is not in affine form, due to the quadratic term Q. Hence, a conservative approximation is taken:

$$Q \quad \approx \quad uv\varepsilon_{n+1} \qquad \text{where} \quad u = \sum_{i=1}^{n}|x_i| \quad v = \sum_{i=1}^{n}|y_i|$$

Affine forms for other elementary operations such as division and square root are given in [13]. It has been shown that affine arithmetic gives tighter bounds than interval arithmetic for both fixed-point [9] and floating-point designs [10].

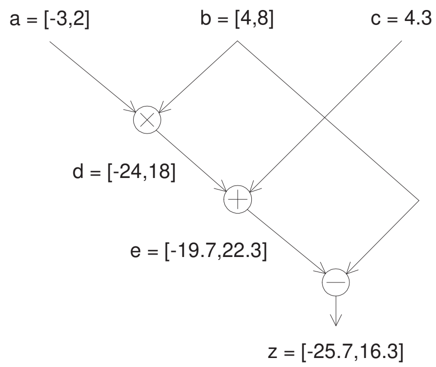


Fig. 3. An example circuit performing $z = ab + c - b$. The range of a signal is shown in the square brackets.

V. Range Analysis

The authors in [9] propose a single affine expression to capture both range and precision. However, we believe range and precision expressions should be kept separately. Precision is a function of range for operations such as multiplication and division, and hence, the number of error terms ε_i can easily explode. After range analysis, we obtain numerical values for the ranges, hence the affine expressions for precisions remain manageable.

We use affine arithmetic for the range analysis to minimize the integer bit-widths required for each signal. For instance, let us consider the evaluation of $z = ab + c - b$ as illustrated in Fig. 3. First, we assume the users know the exact range of all the input signals. Since we want to obtain the range for each signal, we set $d = ab$, $e = d + c$ and $y = e - b$. In affine form we get

$$\begin{array}{ll} \hat{a} = -0.5 + 2.5\varepsilon_1 & \hat{b} = 6 + 2\varepsilon_2 \\ \hat{c} = 4.3 & \hat{d} = -3 + 15\varepsilon_1 - 1\varepsilon_2 + 5\varepsilon_3 \\ \hat{e} = 1.3 + 15\varepsilon_1 - 1\varepsilon_2 + 5\varepsilon_3 & \hat{z} = -4.7 + 15\varepsilon_1 - 7\varepsilon_2 + 5\varepsilon_3 \end{array}$$

Hence, the ranges of the signals are $d = [-24, 18]$, $e = [-19.7, 22.3]$ and $z = [-25.7, 16.3]$. We perform range analysis on all signals for a given design and find the range for each signal. The integer bit-width (IB) required for a signal x is computed with

$$\begin{aligned} IB_x \quad &= \quad \lceil \log_2(\max(|x_{min}|, |x_{max}|)) \rceil + \alpha \qquad (1) \\ &\text{where } \alpha = \begin{cases} 1, & \mathrm{mod}(\log_2(x_{max}), 1) \neq 0 \\ 2, & \mathrm{mod}(\log_2(x_{max}), 1) = 0. \end{cases} \end{aligned}$$

However, AA does not always lead to a better range estimation than IA. For instance, applying IA to the example, we obtain $d = [-24, 16]$, which is narrower than the AA result. This is mainly due to the suboptimal fitting of an affine expression which is not due to the affine approach itself.

VI. Precision Analysis

We use affine arithmetic for precision analysis in a similar fashion as for range analysis. There are two main ways to quantize a signal: truncation and round-to-nearest. Truncation and round-to-nearest can cause a maximum error of 2^{-FB} (1 ulp) and 2^{-FB-1} (0.5 ulp), respectively. Truncation chops bits off the least significant bits and requires no extra hardware resources. Round-to-nearest involves a small adder followed by truncation. For simplicity, we shall perform round-to-nearest throughout this work. Hence, the quantized version $\tilde{x}$ of a signal x is given in affine form by

$$\tilde{x} = x + 2^{-FB_{\tilde{x}}-1}\varepsilon \quad \text{where } \varepsilon = [-1, 1]$$

where $FB_{\tilde{x}}$ is the fraction bit-width of $\tilde{x}$. Hence, the error at $\tilde{x}$ due to finite precision effects is given by

$$E_{\tilde{x}} = 2^{-FB_{\tilde{x}}-1}\varepsilon$$

For addition/subtraction, the affine error expression is given by

$$\begin{aligned} \tilde{z} &= \tilde{x} \pm \tilde{y} = x \pm y + E_{\tilde{x}} \pm E_{\tilde{y}} + 2^{-FB_{\tilde{z}}-1}\varepsilon_3 \\ \Rightarrow E_{\tilde{z}} &= E_{\tilde{x}} + E_{\tilde{y}} + 2^{-FB_{\tilde{z}}-1}\varepsilon_3 \end{aligned}$$

For multiplication:

$$\begin{aligned} \tilde{z} &= \tilde{x}\tilde{y} \\ &= xy + xE_{\tilde{y}} + yE_{\tilde{x}} + E_{\tilde{x}}E_{\tilde{y}} + 2^{-FB_{\tilde{z}}-1}\varepsilon_3 \\ \Rightarrow E_{\tilde{z}} &= xE_{\tilde{y}} + yE_{\tilde{x}} + E_{\tilde{x}}E_{\tilde{y}} + 2^{-FB_{\tilde{z}}-1}\varepsilon_3 \end{aligned}$$

Assuming that the input signals need to be rounded, the application of the error models to the circuit in Fig. 3 is shown below.

$$\begin{aligned} E_{\tilde{a}} &= 2^{-FB_{\tilde{a}}-1}\varepsilon_1 \\ E_{\tilde{b}} &= 2^{-FB_{\tilde{b}}-1}\varepsilon_2 \\ E_{\tilde{c}} &= 2^{-FB_{\tilde{c}}-1}\varepsilon_3 \\ E_{\tilde{d}} &= aE_{\tilde{b}} + bE_{\tilde{a}} + E_{\tilde{a}}E_{\tilde{b}} + 2^{-FB_{\tilde{d}}-1}\varepsilon_4 \\ E_{\tilde{e}} &= E_{\tilde{d}} + E_{\tilde{c}} + 2^{-FB_{\tilde{e}}-1}\varepsilon_6 \\ E_{\tilde{z}} &= E_{\tilde{e}} - E_{\tilde{b}} + 2^{-FB_{\tilde{z}}-1}\varepsilon_7 \end{aligned}$$

Note that $E_{\tilde{d}}$ would be at its maximum when the signals a and b are at their absolute maximum, i.e. $a = 3$ and $b = 8$.

Substituting the equations we get the following maximum error at the output $\tilde{z}$.

$$\begin{aligned} \max(E_{\tilde{z}}) &= 2^{-FB_{\tilde{b}}} + 2^{-FB_{\tilde{a}}+2} + 2^{-FB_{\tilde{a}}-FB_{\tilde{b}}-2} \\ &\quad +2^{-FB_{\tilde{d}}-1} + 2^{-FB_{\tilde{c}}-1} \\ &\quad +2^{-FB_{\tilde{e}}-1} + 2^{-FB_{\tilde{z}}-1} \end{aligned}$$

For faithful rounding, the output error $E_{\tilde{z}}$ needs to be less than or equal to 1 ulp, i.e.

$$\begin{aligned} 2^{-FB_{\tilde{z}}} &\geq \max(E_{\tilde{z}}) \\ \Rightarrow 2^{-FB_{\tilde{z}}-1} &\geq 2^{-FB_{\tilde{b}}} + 2^{-FB_{\tilde{a}}+2} + 2^{-FB_{\tilde{a}}-FB_{\tilde{b}}-2} \\ &\quad +2^{-FB_{\tilde{d}}-1} + 2^{-FB_{\tilde{c}}-1} + 2^{-FB_{\tilde{e}}-1} \end{aligned} \quad (2)$$

From Eqn. (2), we see that the aim is to find minimal FB for each signal that satisfy the inequality and results in minimal circuit area. Since each FB is an integral value, the constraint space of this optimization problem is non-convex. As a result, we choose the adaptive simulated annealing (ASA) package available from [22]. Traditional simulated annealing is very effective in discovering the global optimum, but its problem has been the slow convergence. ASA has been developed to statistically find the test global fit of a nonlinear constrained non-convex cost function over a D-dimensional space. It permits adaptation to changing sensitivities in the multi-dimensional parameter space, thus allowing significantly faster convergence times.

TABLE I
THE INTEGER BIT-WIDTHS (IBS), THE UNIFORM FRACTION BIT-WIDTHS (UFBS) AND THE MULTIPLE FRACTION BIT-WIDTHS (MFBS) TO THE EXAMPLE CIRCUIT IN FIG. 3.

Signal	$\tilde{a}$	$\tilde{b}$	$\tilde{c}$	$\tilde{d}$	$\tilde{e}$	$\tilde{z}$
IB	2	3	3	5	5	5
UFB	12	12	12	12	12	8
MFB	12	11	12	12	12	8

In ASA, the user supplies a constraint function and a cost function. Error functions such as the inequality above are supplied as the constraint function. Since our aim is to minimize circuit area in this work, we supply an area model of the circuit as a function of the signal bit-widths as the cost function. In this area model, a full adder is assumed to be the unit area, i.e. the area for the addition $x + y$ is modeled with $\max(IB_x + FB_x, IB_y + FB_y)$ and the area for the multiplication xy is modeled with $(IB_x+FB_x)(IB_y+FB_y)$. These area models are derived to correspond with the operator area usage of our hardware compilation system (ASC) [19]. Other area models can be used to target different hardware compilers and device technologies.

The annealing process can be accelerated significantly by supplying good initial variable values (FBs in our case). Optimum uniform FBs are analytically computed and used as the initial variable values. For Eqn. (2), substituting the fraction bit-widths in the computation chain with a uniform fraction bit-width UFB,

$$\boxed{2^{-FB_{\tilde{z}}-1} \geq 2^{-UFB} + 2^{-UFB+2} + 2^{-2UFB-2} + 3(2^{-UFB-1})}$$

Let $FB_{\tilde{z}} = 8$ bits. Solving the equation for the minimum value of UFB which satisfies the inequality gives us $UFB = 12$ bits, an analytical solution of the uniform bit-width selection problem.

The integer bit-widths (IBs), the uniform fraction bit-width (UFBs) and the multiple fraction bit-widths (MFBs) to the example circuit are summarized in Table I. The IBs are obtained by using the signal ranges found in Section V and Eqn. (2), and the MFBs are computed by ASA. We observe that ASA is able to reduce the multiplication operand b by one bit. The area cost of using UFBs and MFBs is found to be 244 and 230 units, respectively. By using MFBs, we have achieved an area saving of 6%.

When using UFBs we get a gap of 3.66×10^{-4} between the actual error and the requested error, since the degree of

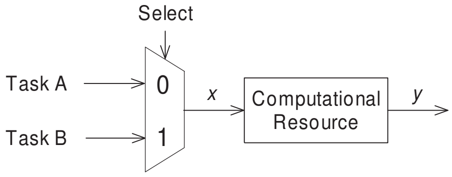


Fig. 4. A resource sharing scenario: Task A and Task B time-share a computational resource.

freedom is one dimensional. However with MFBs, we have a multi dimensional degree of freedom, resulting in smaller error gap of 1.22×10^{-4} and area cost. The differences between UFBs and MFBs are rather small for this example circuit, since the error gap is already rather small when using UFBs. We see more dramatic differences between UFBs and MFBs for designs with larger error gaps, as it will be demonstrated with the case studies in Section IX.

VII. Optimization under Resource Sharing

We often encounter a scenario such as the one depicted in Fig. 4 where we want to share a single resource for two (or more) tasks in a time-multiplexed manner. The range and precision analysis presented in the previous two sections can be trivially extended to cover these cases: signals that are shared among multiple tasks should be large enough to cover all tasks.

For the scenario in Fig. 4, we run range and precision analysis twice. In the first cycle, we assume that the select signal is set to zero, i.e. Task A is in execution. In the second cycle, we assume that the select signal is set to one. After the two bit-width analysis cycles, we obtain two sets of bit-widths, each corresponding to a task data path. For all signals that are shared (x, y and all signals within the shared computational resource), we take the maximum of each signal's tuple of bit-widths. For instance, if the signal x has the following set $IB_x = \{2, 4\}$ and $FB_x = \{12, 9\}$, we set $IB_x = 4$ and $FB_x = 12$. This scheme allows all signals to have sufficient bit-widths for every task.

VIII. Case Studies

For the five case studies below, we assume that the inputs use the same fraction bit-widths as the outputs. For the cases when multiple outputs are present, we use the same output precision for all outputs

For these cases studies, we develop a fully automated design flow which starts with designs captured in ASC [19] C++ syntax. All stages within MiniBit (Fig. 2) are written in C++ and integrated within the ASC system. The MiniBit input pass parses the input design description into an internal dataflow graph (DFG) representation for the range analysis pass. The error function generation pass produces an error model and the cost function generation pass produces an area cost model. The results from range analysis, together with the generated error function and cost function, are used to derive the UFB. Next, using ASA, we compute the MFBs. The MiniBit output pass converts the bit-width optimized DFG representations back into ASC design description. The verification output generation step produces a variable bit-width assigned design for simulation to verify the optimized results.

A. Polynomial Approximation

We examine the degree four polynomial for the approximation to $y = \log(1 + x)$ where $x = [0, 1)$. Horner's rule evaluates the polynomial:

$$y \quad = \quad ((c_d x + c_{d-1})x + \ldots)x + c_0$$

where $c_0 \ldots c_d$ are the polynomial coefficients. The coefficients are obtained in a minimax sense to minimize the maximum absolute error.

B. RGB to YCbCr

We consider the RGB to YCbCr color space converter specified by the JPEG 2000 standard [24]. The input signals R, G and B are assumed to be 8-bit unsigned integers. Shifts are used for the multiplications by 0.5.

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.16875 & -0.33126 & 0.5 \\ 0.5 & -0.41869 & -0.08131 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

C. Matrix Multiplication

The 2×2 matrix multiplication using Strassen's algorithm [25] is considered, which is commonly used as a basic processing element for large matrix multiplications. We assume the elements of the input matrices are over [0,1).

$$\begin{bmatrix} y_{00} & y_{01} \\ y_{10} & y_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}$$

The four quadrants of the result matrix can be calculated as follows:

$$\begin{aligned} y_{00} &= p_0 + p_3 - p_4 + p_6 \\ y_{01} &= p_2 + p_4 \\ y_{10} &= p_1 + p_3 \\ y_{11} &= p_0 + p_2 - p_1 + p_5 \end{aligned} \qquad \begin{aligned} p_0 &= (a_{00} + a_{11})(b_{00} + b_{11}) \\ p_1 &= (a_{10} + a_{11})b_{00} \\ p_2 &= a_{00}(b_{01} - b_{11}) \\ p_3 &= a_{11}(b_{10} - b_{00}) \\ p_4 &= (a_{00} + a_{01})b_{11} \\ p_5 &= (a_{10} - a_{00})(b_{00} + b_{01}) \\ p_6 &= (a_{01} - a_{11})(b_{10} + b_{11}) \end{aligned}$$

D. B-Splines

We examine uniform cubic B-splines, commonly used for image warping applications [26]. The B-spline basis functions, B_0, B_1, B_2 and B_3, are defined by

$$B_0(u) = \frac{(1-u)^3}{6} \qquad B_2(u) = \frac{-3u^3+3u^2+3u+1}{6}$$

$$B_1(u) = \frac{3u^3-6u^2+4}{6} \qquad B_3(u) = \frac{-u^3}{6}$$

where $u = [0, 1)$. For the implementation of this design, optimizations including shifts instead of multiplications, and sharing common intermediate results are carried out.

TABLE II
NUMBER OF ADDITIONS/SUBTRACTIONS, MULTIPLICATIONS AND SIGNALS TO BE OPTIMIZED.

Case Study	Add/Sub	Mult	Signals
Degree 4 Poly	4	4	12
RGB to YCbCr	6	7	19
2×2 Matrix Mult	18	7	21
B-Splines	15	6	22
8×8 DCT	32	32	55

E. Discrete Cosine Transform (DCT)

We consider the 8×8 discrete cosine transform (DCT) implemented according to [27]. A vector of input data $x_{0...7}$ can be transformed to DCT coefficients $y_{0...7}$ by

$$\begin{bmatrix} y_0 \\ y_2 \\ y_4 \\ y_6 \end{bmatrix} = \begin{bmatrix} c_0 & c_0 & c_0 & c_0 \\ c_2 & c_5 & -c_5 & c_2 \\ c_0 & -c_0 & -c_0 & c_0 \\ c_5 & -c_2 & c_2 & -c_5 \end{bmatrix} \begin{bmatrix} x_0 + x_7 \\ x_1 + x_6 \\ x_2 + x_5 \\ x_3 + x_4 \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_3 \\ y_5 \\ y_7 \end{bmatrix} = \begin{bmatrix} c_1 & c_3 & c_4 & c_6 \\ c_3 & -c_6 & -c_1 & -c_4 \\ c_4 & -c_1 & c_6 & c_0 \\ c_6 & -c_4 & c_3 & -c_1 \end{bmatrix} \begin{bmatrix} x_0 - x_7 \\ x_1 - x_6 \\ x_2 - x_5 \\ x_3 - x_4 \end{bmatrix}$$

where $c_{0...7}$ are trigonometric constants. We use 8-bit unsigned integers for the elements in the input vector $x_{0...7}$.

IX. RESULTS

We implement the five case studies with ASC, A Stream Compiler, for FPGAs [19]. ASC code makes use of C++ syntax and ASC semantics, which allow the user to program on the architecture-level, the arithmetic-level and the gate-level. Designs are synthesized with ASC and placed-and-routed with Xilinx ISE 6.3 on a Xilinx Virtex-4 XC4VLX100-11 FPGA. The device contains user-programmable elements known as slices, dedicated multiply-and-add units and embedded RAMs. In order to make fair comparisons, we implement designs using slices only and combinatorially without any pipelining. Table II shows the number of additions, subtractions, multiplications and signals to be optimized for the five case studies. We observe that the degree four polynomial is the least complex and the 8×8 DCT has the most complexity.

A. Comparisons of Uniform Fraction Bit-Widths and Multiple Fraction Bit-Widths

Table III shows the optimization times and error statistics of multiple fraction bit-width designs, and comparisons between uniform fraction bit-width (UFB) and multiple fraction bit-width (MFB) designs. The UFB and MFB designs use the same number of integer bits for all signals, as computed in our range analysis phase. The optimization times have been measured on an AMD Athlon XP 2600+ PC with 2GB DDR-SDRAM, and include both range and precision analysis times. Adaptive simulated annealing (ASA) is the dominant factor

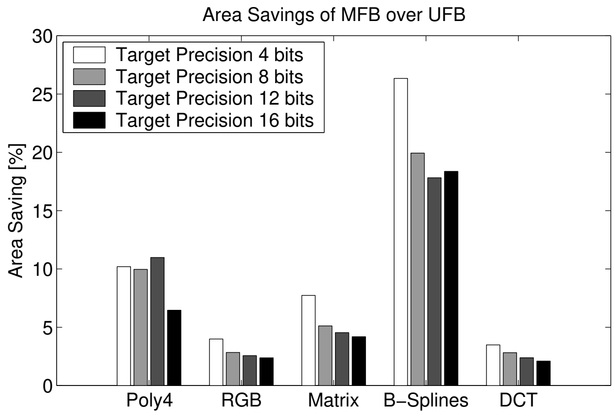


Fig. 5. Percentage area saving of MFB over UFB at different target precisions.

in the optimization process and we observe that more time is needed for designs with more signals, which is expected. Note that for all case studies, the optimizations times are a matter of seconds.

The ulp errors and SNRs are computed by simulating MFB optimized designs using random input vectors. IEEE 64-bit double-precision floating-point is assumed to be the true value for the error computations, since it is significantly more accurate than the precisions we are targeting. The maximum ulp error for all designs are well below 1 ulp indicating that the results are indeed faithfully rounded. Also, the average ulp error is less than 0.3 for all case studies. Looking at the area and speed comparisons, although we optimize designs for minimal area, the reduction in the multiplier and adder sizes leads to reductions in latencies as a byproduct. We note that designs using MFBs are always smaller and faster than designs using UFBs. Some of the savings may seem rather small, but we get these savings for free, as the MFB designs have the same error bound as the UFB designs.

Fig. 5 shows the percentage area saving of MFB over UFB for the five case studies at different target precisions. Generally, we obtain increased relative savings for lower precisions. Another trend is that designs with deeper computation chains can benefit more with MFBs: the depths can be deduced by examining the latency (combinatorial delay) results in Table III. An area saving of up to 26% is achieved in the case of B-Splines, which has a deep computation chain.

Fig. 8 shows the area variation for B-splines with increasing target precision. It can be seen that the area differences between UFB and MFB are increasing with the target precision. Fig. 6 and Fig. 7 show the area and latency variations for various polynomial degrees with target precision fixed at eight bits. Since we use Horner's rule to evaluate polynomials, one extra degree causes one more adder and one more multiplier. In order to make a fair comparison, the coefficients are set to the number π throughout. We can see that as we increase the depth of the computation chain (i.e. increase the polynomial degree), the area and latency differences between UFB and MFB increase.

Fig. 9 illustrates how the best adaptive simulated annealing

TABLE III
OPTIMIZATION TIMES AND ERROR STATISTICS OF MULTIPLE FRACTION BIT-WIDTH DESIGNS (MFB RESULTS), AND COMPARISONS BETWEEN UNIFORM FRACTION BIT-WIDTH AND MULTIPLE FRACTION BIT-WIDTH DESIGNS (UFB/MFB COMPARISONS).

Case Study		**MFB Results**				**UFB/MFB Comparisons**							
Application	Prec [bits]	Opt Time [s]	Max Err [ulp]	Avg Err [ulp]	SNR [dB]	Area [slices] UFB	MFB	Diff	Impr [%]	Latency [ns] UFB	MFB	Diff	Impr [%]
Degree 4 Polynomial	8	1.9	0.694	0.253	51.5	803	723	80	9.96	114.00	105.39	8.61	7.55
	16	2.0	0.731	0.256	99.5	1921	1797	124	6.45	168.55	151.81	16.74	9.93
RGB to YCbCr	8	8.9	0.662	0.260	97.2	1165	1132	33	2.83	37.47	36.95	0.52	1.39
	16	9.7	0.793	0.272	144.9	1641	1602	39	2.38	50.26	48.83	1.43	2.85
2×2 Matrix Multiplication	8	16.1	0.520	0.251	54.4	1896	1799	97	5.12	44.20	42.73	1.47	3.33
	16	19.5	0.528	0.247	102.5	4240	4072	168	3.96	59.22	56.14	3.08	5.20
B-Splines	8	27.7	0.716	0.267	49.8	1189	952	237	19.93	88.39	78.58	9.81	11.10
	16	32.8	0.774	0.278	96.6	2652	2165	487	18.36	130.11	114.03	16.08	12.36
8×8 DCT	8	154.3	0.702	0.254	103.1	5368	5217	151	2.81	54.83	50.73	4.10	7.48
	16	179.1	0.708	0.257	151.3	7320	7167	153	2.09	66.39	59.42	6.97	10.50

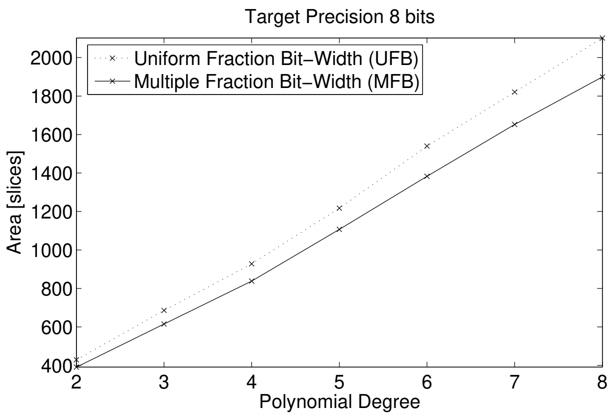


Fig. 6. Area variation for various polynomial degrees with target precision fixed at eight bits.

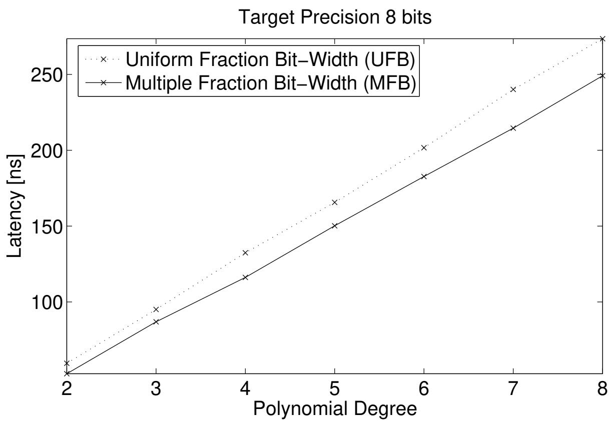


Fig. 7. Latency variation for various polynomial degrees with target precision fixed at eight bits.

(ASA) area cost varies with the number of iterations for B-splines with a target precision of eight bits. ASA first starts with UFB, which results in a cost value of 1303 units. We observe that by increasing the number of iterations, we gradually approach a set of MFBs that give the lowest cost value, which is 982 units in this case. The annealing time to approach this optimum cost value is 23 seconds.

B. Comparisons of Different Range Analysis Methods

We examine the impact of using different range analysis methods on design the area and latency. Comparisons to the five case studies using simulation, affine arithmetic (AA) and interval arithmetic (IA) for the range are shown in Table IV. The simulation-based range analysis is performed by feeding the designs with a large data set of random input samples and observing the data range. Although the simulation method gives smaller area and shorter latency, its drawback is that it can only guarantee overflow/underflow protection for the samples tested. It becomes quickly impractical for high input resolutions, since the run time increases exponentially with the resolution.

We also observe that AA gives slightly better results than IA. This is due to the fact that AA can exploit correlations between signals as discussed in Section IV.

C. Comparisons with Integer Linear Programming

In order to determine the optimality of the ASA solution used in MiniBit, we compare our results against the optimal bit-widths allocated through integer linear programming (ILP), which is known to give global optimum results. The ILP formulations are solved by using the ILOG CPLEX [28] solver. The optimization problem contains non-linear functions that are converted into equivalent linear functions following the approach in [29]. The obtained ILP results are used as optimal solutions to judge the quality of the ASA results. The ILP solutions presented here take several hours to several days on an AMD Athlon XP 2600+ PC with 2GB DDR-SDRAM.

TABLE IV

AREA AND LATENCY COMPARISONS USING DIFFERENT RANGE ANALYSIS METHODS FOR TARGET PRECISION OF EIGHT BITS.

Case Study	Area [slices]			Latency [ns]		
	Simulation	Affine Arithmetic	Interval Arithmetic	Simulation	Affine Arithmetic	Interval Arithmetic
Degree 4 Poly	701	723	732	103.21	105.39	107.37
RGB to YCbCr	1109	1132	1143	35.17	36.95	38.82
2×2 Matrix Mult	1747	1799	1838	40.71	42.73	43.68
B-Splines	937	952	978	76.01	78.58	89.86
8×8 DCT	5103	5217	5312	47.61	50.73	51.30

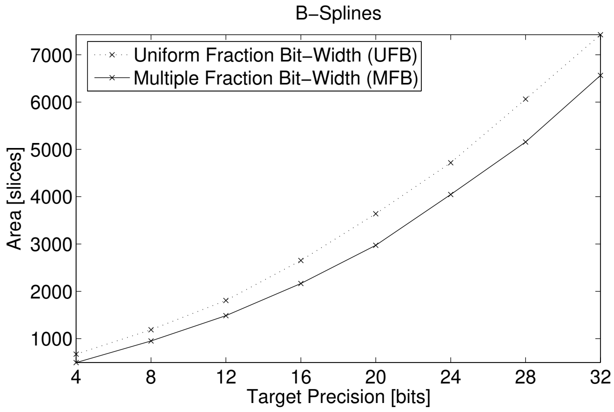


Fig. 8. Area variation for B-splines with increasing target precision.

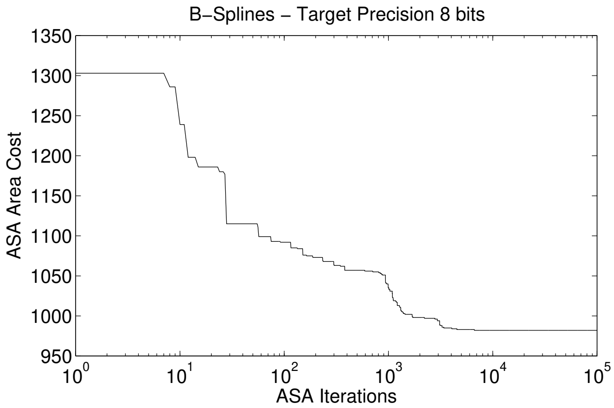


Fig. 9. Adaptive simulated annealing (ASA) area cost variation with ASA iterations for B-splines.

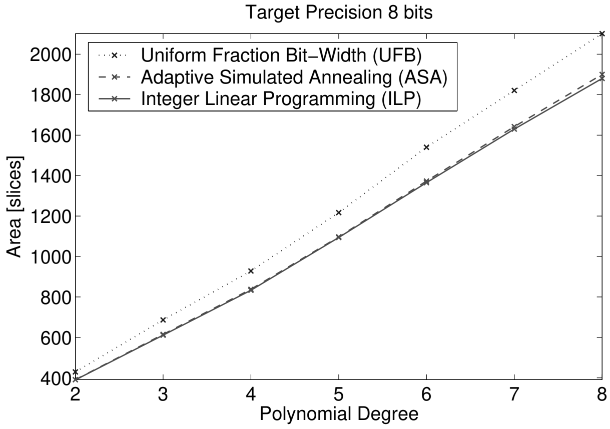


Fig. 10. Area comparison for various polynomial degrees with target precision fixed at eight bits.

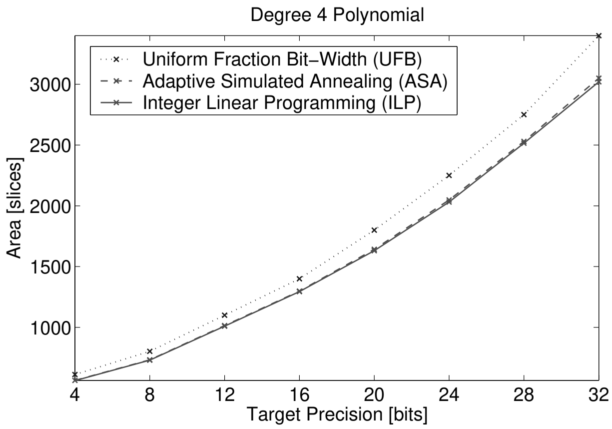


Fig. 11. Area variation for degree four polynomial with increasing target precision.

First, we consider the comparison between ILP and ASA for different design complexities. Fig. 10 presents the results for optimizing degree two to degree twelve polynomials using ILP, ASA and UFB solutions. Although the area savings of ILP over ASA improves slightly with design complexity, they are less than 1%. This indicates that with ASA, we can produce results that are close to the global optimum even with increasing design complexities.

Next, we study optimality of the ASA solution when the design constraint is changed using the degree four polynomial design. Fig. 11 shows the area cost for optimizing the design with different target precision requirements. We see a similar trend to Fig. 10 and again the differences are within 1%.

We conclude that although ILP provides global optimal solutions, we can achieve near optimal solutions using ASA with several orders of magnitude less run time.

X. CONCLUSIONS

We have presented MiniBit, an automated approach for optimizing bit-widths of fixed-point designs with static analysis, for designing fixed-point hardware with guaranteed accuracy.

We have described methods to minimize both integer and fraction parts of fixed-point signals based on affine arithmetic. For precision analysis, we employ a semi-analytical approach, where analytical error models in conjunction with adaptive simulated annealing (ASA) are used to find a local optimum number of fraction bits. The analytical models allow us to guarantee overflow/underflow protection and numerical accuracy for all possible inputs over the user-specified input intervals. We make the assumption that the maximum errors can happen at all nodes at the same time, however in practice, this will perhaps never be the case. This assumption will often result in pessimistic bit-widths. However, given that our approach can guarantee accuracy, we believe it is a small tradeoff to make.

Although our approach has been primarily designed for FPGA and ASIC applications, its principles could be applied to other applications such as fixed-point digital signal processors (DSPs). For DSPs, one could apply our range analysis to determine the integer bit-widths, and use our precisions analysis to calculate the maximum error bounds at the output of the computation chain. We can also apply MiniBit to configurable processors with extensible instructions for embedded applications such as Tensilica [30] by optimizing the bit-widths of the custom instruction blocks.

Two limitations of our approach are: (1) the search space for ASA can be vast for large designs, leading to slow optimization times; (2) although this initial work does not cover designs with feedback loop, we can extend the error model by using the technique described in [9]. For future work, we hope to use clustering techniques to optimize parts of a large design independently, which will result in suboptimal bit-widths but faster optimization time. Moreover, we hope to extend MiniBit to cover floating-point arithmetic.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their remarks. The support of the Jet Propulsion Laboratory, Xilinx Inc., the U.K. Engineering and Physical Sciences Research Council (Grant number GR/N 66599, GR/R 55931 and GR/R 31409) and the Croucher Foundation is gratefully acknowledged.

REFERENCES

[1] G. Constantinides and G. Woeginger, "The complexity of multiple wordlength assignment," *Applied Mathematics Letters*, vol. 15, no. 2, pp. 137–140, 2001.

[2] A. Abdul Gaffar, O. Mencer, W. Luk, and P. Cheung, "Unifying bit-width optimisation for fixed-point and floating-point designs," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, 2004, pp. 79–88.

[3] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens, "A methodology and design environment for DSP ASIC fixed point refinement," in *Proc. ACM/IEEE Design Automation and Test in Europe Conf.*, 1999, pp. 271–276.

[4] K. Kum and W. Sung, "Combined word-length optimization and high-level synthesis of digital signal processing systems," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 8, pp. 921–930, 2001.

[5] S. Kim and W. Sung, "Fixed-point error analysis and word length optimization of 8×8 IDCT architectures," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 8, pp. 935–940, 1998.

[6] C. Shi and R. Brodersen, "Automated fixed-point data-type optimization tool for signal processing and communication systems," in *Proc. ACM/IEEE Design Automation Conf.*, 2004, pp. 478–483.

[7] M. Willems, V. Bürgens, H. Keding, T. Grötker, and H. Meyr, "System level fixed-point design based on an interpolative approach," in *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 293–298.

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

[9] C. Fang, R. Rutenbar, and T. Chen, "Fast, accurate static analysis for fixed-point finite-precision effects in DSP designs," in *Proc. ACM/IEEE Int'l Conf. on Computer-Aided Design*, 2003, pp. 275–282.

[10] C. Fang, R. Rutenbar, M. Püschel, and T. Chen, "Toward efficient static analysis of finite-precision effects in DSP applications via affine arithmetic modeling," in *Proc. ACM/IEEE Design Automation Conf.*, 2003, pp. 496–501.

[11] D. Menard and O. Sentieys, "Automatic evaluation of the accuracy of fixed-point algorithms," in *Proc. ACM/IEEE Design Automation and Test in Europe Conf.*, 2002, pp. 1530–1591.

[12] S. Wadekar and A. Parker, "Accuracy sensitive. word-length selection for algorithm optimization," in *Proc. IEEE Int'l. Conf. on Computer Design*, 1998, pp. 54–61.

[13] L. de Figueiredo and J. Stolfi, "Self-validated numerical methods and applications," in *Brazilian Mathematics Colloquium monograph.* IMPA, Brazil, 1997.

[14] V. Lefevre, J. Muller, and A. Tisserand, "Toward correctly rounded transcendentals," *IEEE Trans. Computers*, vol. 47, no. 11, pp. 1235–1243, 1998.

[15] M.J. Schulte and E.E. Swartzlander Jr., "Hardware designs for exactly rounded elementary functions," *IEEE Trans. Computers*, vol. 43, no. 8, pp. 964–973, 1994.

[16] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee, "Precision and error analysis of Matlab applications during automated hardware synthesis for FPGAs," in *Proceedings of the DATE 2001 on Design, automation and test in Europe*, 2001, pp. 722–728.

[17] M. Haldar, A. Nayak, N. Shenoy, A. Choudhary, and P. Banerjee, "FPGA hardware synthesis from Matlab," in *Proc. Conf. VLSI Design,*, 2001, pp. 299–304.

[18] G. Constantinides, "Perturbation analysis for word-length optimization," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, 2003, pp. 81–90.

[19] O. Mencer, D. Pearce, L. Howes, and W. Luk, "Design space exploration with A Stream Compiler," in *Proc. IEEE Int'l Conf. Field-Programmable Technology*, 2003, pp. 270–277.

[20] *Xilinx System Generator User Guide v6.3*, Xilinx Inc., 2004, http://www.xilinx.com.

[21] L. Ingber, "Very fast simulated re-annealing," *Journal of Mathematical Computer Modelling*, vol. 12, no. 8, pp. 967–973, 1989.

[22] ——, *Adaptive Simulated Annealing (ASA) 25.15*, 2004, http://www.ingber.com/#ASA.

[23] R. Moore, *Interval Analysis*. Prentice-Hall, 1966.

[24] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 36–58, 2001.

[25] V. Strassen, "Gaussian elimination is not optimal," *Numerische Mathematik*, vol. 13, pp. 354–356, 1969.

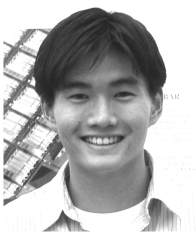
[26] J. Jiang, W. Luk, and D. Rueckert, "FPGA-based computation of free-form deformations in medical image registration," in *Proc. IEEE Int'l Conf. Field-Programmable Technology*, 2003, pp. 234–241.

[27] A. Madisetti and A.N. Willson Jr., "A 100 MHz 2-D 8×8 DCT/IDCT processor for HDTV applications," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 5, no. 2, pp. 158–165, 1995.

[28] *ILOG CPLEX 9.0, User's Manual*, ILOG SA., http://www.ilog.com/products/cplex, 2003.

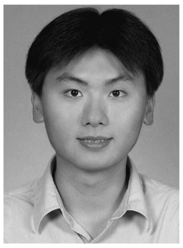
[29] G. Constantinides, P. Cheung, and W. Luk, "Optimum Wordlength Allocation," in *Proc. IEEE Symp. on Field Programmable Custom Computing Machines*, 2002, pp. 219 – 228.

[30] R. Gonzalez, "Xtensa: a configurable and extensible processor," *IEEE Micro*, vol. 20, no. 2, pp. 60–70, 2000.



Dong-U Lee (S'01-M'05) received the BEng degree in information systems engineering and the PhD degree in computing, both from Imperial College London in 2001 and 2004, respectively. He is currently a postdoctoral researcher at the Electrical Engineering Department, University of California, Los Angeles (UCLA), where he is working on channel codes and symbol timing synchronization for deep-space communications with the Jet Propulsion Laboratory, NASA. His research interests include computer arithmetic, communications, design automation, reconfigurable computing and video image processing. He is a member of the IEEE.

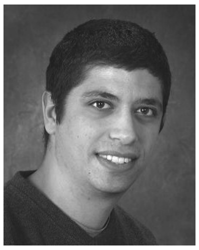
Altaf Abdul Gaffar (S'03-M'05) received the BEng degree in information systems engineering and the PhD degree in computing, both from Imperial College London in 2000 and 2005, respectively. He is currently working as a research assistant at the Electrical and Electronic Engineering Department at Imperial College London. His research interests include bit-width optimization for floating-point and fixed-point arithmetic, and high-level power estimation and optimization techniques. He is a member of the IEEE.



Ray C.C. Cheung (S'01) received the BEng degree and MPhil degree in Computer Engineering and Computer Science and Engineering from The Chinese University of Hong Kong (CUHK) in 1999 and 2001 respectively. In 2001, he worked as a system administrator in the Center of Large-Scale Computation (CLC) of Cluster Technology, HK. From Jan 2002 to Dec 2003, he was an instructor of the Department of Computer Science and Engineering, CUHK. He is now a PhD candidate, in the Custom Computing group, Department of Computing, Imperial College London. His current research interests are computer arithmetic hardware designs and design exploration of System-on-Chip (SoC) designs and embedded systems. He is a student member of the IEEE and the ACM, and is the newsletter and web editor of the SIGDA UK chapter.

Oskar Mencer received his PhD and MS in Electrical Engineering from Stanford University in 2000 and 1997 respectively, and a BS degree in Computer Engineering from the Technion/Israel in 1994. He founded MAXELER Technologies in 2003 after three years as member of Technical staff in the Computing Sciences Research Center at Bell Labs. He is a member of the academic staff in the Department of Computing, Imperial College London and with the Custom Computing group. His research interests span computer architecture, computer arithmetic, VLSI microarchitecture, VLSI CAD, and reconfigurable (custom) computing. More specifically, he is interested in exploring application specific representation of computation at the algorithm level, the architecture level, and the arithmetic level. He is a member of the IEEE.

Wayne Luk (S'85-M'89) received the MA, MSc, and DPhil degrees in engineering and computing science from the University of Oxford, Oxford, United Kingdom. He is Professor of Computer Engineering in the Department of Computing, Imperial College London and leads the Custom Computing Group there. His research interests include theory and practice of customizing hardware and software for specific application domains, such as graphics and image processing, multimedia, and communications. Much of his current work involves high-level compilation techniques and tools for parallel computers and embedded systems, particularly those containing reconfigurable devices such as field-programmable gate arrays. He is a member of the IEEE.



George A. Constantinides (S'96-M'01) received the MEng degree in information systems engineering and the PhD degree in electrical and electronic engineering from Imperial College, London, UK, in 1998 and 2001, respectively. Since 2002, he has been a Lecturer in digital systems, Electrical and Electronic Engineering Department, Imperial College. He is the author of "Synthesis and Optimization of DSP Algorithms" (Dordrecht, Germany: Kluwer, 2004). His research interests include reconfigurable computing and electronic design automation, with a particular focus on digital signal processing algorithms. Dr. Constantinides was program Co-Chair of the International Conference on Field-Programmable Logic and Applications in 2003, and serves on the Program Committees of FPL, FPT, ISCAS, ERSA, and ARC. He was the Founding Chair of the UK SIGDA Chapter, serving as General Chair in 2001, and Technical Chair from 2002 to 2003 of the annual workshop. He is a member of the IEEE and the ACM.