# Low-Power Optimization by Smart Bit-Width Allocation in a SystemC-Based ASIC Design Environment

Arindam Mallik, *Student Member, IEEE*, Debjit Sinha, *Member, IEEE*, Prithviraj Banerjee, *Fellow, IEEE*, and Hai Zhou, *Senior Member, IEEE*

***Abstract*—The modern era of embedded system design is geared toward the design of low-power systems. One way to reduce power in an application-specified integrated circuit (ASIC) implementation is to reduce the bit-width precision of its computation units. This paper describes algorithms to optimize the bit widths of fixed-point variables for low power in a SystemC-based ASIC design environment. We propose an optimal bit-width allocation algorithm for two variables and a greedy heuristic that works for any number of variables. The algorithms are used in the automation of converting floating-point SystemC programs into ASIC synthesizable SystemC programs. Expected inputs are profiled to estimate errors in the finite precision conversions. Experimental results for the tradeoffs between quantization error, power consumption, and hardware resources used are reported on a set of four SystemC benchmarks that are mapped onto a 0.18-$\mu$m ASIC cell library from Artisan Components. We demonstrate that it is possible to reduce the power consumption by 50% on the average by allowing roundoff errors to increase from 0.5% to 1%.**

***Index Terms*—Fixed-point arithmetic, high-level synthesis, low-power design, quantization.**

## I. Introduction

POWER consumption has become a primary design criterion for modern embedded systems. The majority of low-power design and analysis tools in the electronic design automation industry target low levels of chip design, such as transistor level, gate level, and register transfer level (RTL). However, it is widely recognized that the largest gains in power savings occur at the system level.

Reduction of the bit-width precision of computation units such as adders and multipliers results in an immediate gain in power savings and area of design. However, this adversely affects the accumulation of quantization errors due to finite precision arithmetic. Most practical application-specified integrated circuit (ASIC) designs of embedded applications are limited to fixed-point arithmetic due to the cost and complexity of floating-point hardware. Consequently, we consider system-level tradeoffs in quantization errors with the area and power consumption of the hardware implementation by varying the bit-width precision of individual operators.

SystemC [1] is a relatively new modeling language based on C++ that is intended to enable system-level design and Internet Protocol exchange. It has been developed as a standardized modeling language for systems containing both hardware and software components.

Fixed-point numbers are frequently used in DSP applications that target both hardware and software implementations. However, common behavioral synthesis tools available in the market for SystemC (e.g., the Synopsys Cocentric Compiler) are unable to synthesize a fixed-point data type. Naturally, there is a growing demand for solutions to this problem.

This paper describes algorithms for trading off quantization error with power consumption and hardware resources in a SystemC-based ASIC design environment. We use high-level power consumption and area usage models to formulate an error-constrained optimization problem. We propose an algorithm for optimal bit-width precision for two variables and extend the approach to a greedy heuristic that works for any number of variables. The power optimization process involves profiling input vectors and fast simulations in addition to high-level synthesis for estimating quantization errors and optimized power values.

It should be noted that our input profiling approach for improved hardware synthesis is similar to approaches used in improving branch prediction in high-performance microarchitectures [17], approaches used in parallel computing to increase thread parallelism using data dependence analysis via value prediction [16], and in the area of high-level power estimation and synthesis, where power consumption is estimated based on input switching activity [18].

The rest of this paper is organized as follows. Section II describes related work. Section III presents the area, delay, and error models used in our SystemC designs. Section IV gives the details of the proposed optimization algorithms. Section V discusses the experimental setup and results. Section VI concludes this paper by summarizing our contributions.

Manuscript received April 3, 2006; revised August 11, 2006 and September 27, 2006. This work was supported in part by the National Aeronautics Space Administration (NASA) under Grant 276685 and in part by the Defense Advanced Research Projects Agency (DARPA) under Grant F33615-01-C-1631. This paper was recommended by Guest Editor D. Sciuto.

A. Mallik and H. Zhou are with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208-3118 USA (e-mail: arindam@ece.northwestern.edu; haizhou@northwestern.edu).

D. Sinha was with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208-3118 USA. He is now with IBM Microelectronics, East Fishkill, NY 12533 USA.

P. Banerjee was with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208-3118 USA. He is now with the School of Engineering, University of Illinois, Chicago, IL 60607 USA.

Digital Object Identifier 10.1109/TCAD.2006.888291

## II. Related Work

The strategies for solving floating-point conversion and precision issues can be roughly categorized into two groups. The first one is an analytical approach used by algorithm developers who analyze finite word length effects due to fixed-point arithmetic. This approach started from attempts to model quantization error statistically; then it was expanded to specific linear time invariant (LTI) systems such as digital filters, fast Fourier transform, etc. In the past three decades, numerous papers have been devoted to this approach [2], [3], [19]–[21], [24]. The other approach is based on bit-true simulation techniques used by hardware designers [4], [14]. It has the ability to handle non-LTI systems as well as LTI systems.

There has been some recent work on automated compiler techniques for the conversion of floating-point representations to fixed-point representations [5], [6], [25]. However, power optimization is not considered in these approaches. The BITWISE compiler [7] determines the precision of all input, intermediate, and output signals in a synthesized hardware design from a C program description. It uses both forward and backward range propagations. In comparison, our approach only uses forward range propagation. We aim to optimize the bit length of the design variables with an error constraint. Belanovic and Rupp proposed the "fixify" [26] environment that automates the floating-point to fixed-point conversion in DSP designs. They proposed optimized algorithms for such automation. However, their work does not consider the power aspect of such designs. We have looked into the power consumption of the design during the optimization process. The MATCH compiler [8] develops precision and error analysis techniques for MATLAB programs. Synopsys has a commercial tool called Cocentric Fixed-Point Designer [6] that automatically converts floating-point computations to fixed point within a C compilation framework. However, the code generated is not synthesizable. Constantinides *et al.* [10], [22], [23] have developed a design tool to tackle both linear and nonlinear designs. Chang and Hauck [11] have developed a tool called PRECIS for precision analysis in MATLAB. An algorithm for automating the conversion of floating-point MATLAB to fixed-point MATLAB was presented in [9] using the AccelFPGA compiler. Their approach needs the default precision of variables and constants specified by the user that the compiler is unable to infer. Recently, Roy *et al.* [12], [13] proposed automated algorithms to convert floating-point MATLAB programs into fixed-point MATLAB programs using input profiling. The work is used to tradeoff area and performance for quantization error. Power is not explicitly considered in their approach.

## III. Area, Power, and Error Models

This section provides an overview of the SystemC library and the area, power, and error models used by our algorithm.

### A. SystemC Fundamentals

The SystemC class library provides necessary constructs to model system architecture including hardware timing, concurrency, and reactive behavior that are unavailable in standard C++. Common behavioral synthesis tools available for SystemC do not support the synthesis of fixed-point data types.

We propose the use of the sc_int$\langle\rangle$ data type, which represents a fixed precision signed integer to preserve a given data precision. When we scale any data or parameter, a part of its fractional part is converted into the integer part. This enables us to take into consideration the effect of the fractional part for that parameter. We can specify the number of bits that would be used to store a particular integer variable. Higher precision in the design is introduced by adding more bits to a parameter of sc_int$\langle\rangle$ type. We consider the following declaration of a variable sample_tmp

$$\text{sc\_int}\langle 12\rangle \ \text{sample\_tmp}.$$

This declaration allocates 12 bits to the variable sample_tmp during synthesis. In our algorithm, we scale the data to increase the precision and allocate extra bits to the scaled variable accordingly. We always employ scale factors that are powers of two, since this facilitates the task of allocating extra bits to the variables. During the first pass over the program (details in Section IV-A), we formulate several relations for bits allocated to each variable in the program. The scaling of input variables by some SCALE_FACTOR may result in the scaling of an internal program variable by SCALE_FACTOR$^n (n = 0, 1, 2, \ldots)$. In this case, we allocate $\log_2(\text{SCALE\_FACTOR}^n)$ bits to that internal variable.

### B. Error Model

The introduction of the SystemC model enables convenient and fast simulation of designs by using its lightweight cycle-based simulation kernel. In our approach, we compute the errors using simulations and consider nonincreasing error values with increase in resources in the form of the bit length of the parameters involved in a particular design. We define an error metric $E$ as

$$E = |\text{Output}_{\text{float}} - \text{Output}_{\text{fixed}}| \,/\, |\text{Output}_{\text{float}}|$$

where $\text{Output}_{\text{float}}$ denotes the vector of floating-point output values for training set inputs, and $\text{Output}_{\text{fixed}}$ denotes the vector of corresponding output values after scaling. Thus, the error metric is the aggregated absolute error.

### C. Area and Power Model Analysis

In order to formulate an error-constrained optimization problem, we need high-level models of power consumption [15] for each type of primary operations in system-level descriptions of algorithms. Adders, multipliers, and registers are considered to be the primary candidates of power consumption. We do not consider power consumption in interconnects and other input/output constructs.

We assume that the power consumption in CMOS circuits consists of dynamic power, leakage power, and static power. At a high level, the total power consumption is directly proportional to the area, which implies that optimizing the bit length of different parameters in a design results in reduced power

consumption. Moreover, bit-length optimization of different parameters reduces the interconnect overhead of the circuit. We use two abstract functions $f$ and $g$ to express the relation between the bit widths of the parameters to area and power consumption, respectively.

It can be observed that there exists a partial relation among different design configurations within a solution space. For a design with $k$ independent variables, the solution space of the design is $k$-dimensional, where a particular design can be represented using $k$ coordinates. Hence, a design $X$ in that solution space can be expressed as $(x_1, x_2, x_3, \ldots, x_k)$, where $x_i$ denotes the bit width of the $i$th independent variable.

If $A(X)$, $P(X)$, and $E(X)$ denote the area, power consumption, and error metric, respectively, obtained for some configuration $X$, we can claim that

$$\begin{aligned} A(X) &\le A(Y) \quad \text{if } x_i \le y_i \qquad \text{for all } i = 1, 2, 3, \ldots, k \\ P(X) &\le P(Y) \quad \text{if } x_i \le y_i \qquad \text{for all } i = 1, 2, 3, \ldots, k \\ E(X) &\ge E(Y) \quad \text{if } x_i \le y_i \qquad \text{for all } i = 1, 2, 3, \ldots, k. \end{aligned}$$

These relations conclude that the abstract area function $f$ and the power function $g$ are monotonically nondecreasing with bit width of variables.

The area of an adder is dependent on the bit width of the variables involved in the operation. Assuming we need $n_a$ and $n_b$ bits for two operands, we can formulate the area of an adder as

$$A_{\text{ADDER}} = f_{\text{ADDER}}(n_a, n_b). \tag{1}$$

The power consumption can now be modeled as

$$P_{\text{ADDER}} = A_{\text{ADDER}} \cdot V_{\text{DD}}^2 \cdot \text{fn(activity)} = g_{\text{ADDER}}(n_a, n_b). \tag{2}$$

It has been experimentally proven that a "coefficient blind" area model does provide good results in practice [10]. The number of additions required to implement a constant coefficient multiplier is assumed to be proportional to the coefficient word length $CW$. If we consider a multiplier with $n_i$ bits as input and $n_o$ bits as output, each multiplier would involve $(n_o + 1)$ bit addition. Hence, we can formulate the area and the power of a multiplier as

$$A_{\text{MULT}} = f_{\text{MULT}}(CW, n_i, n_o) \tag{3}$$

$$P_{\text{MULT}} = g_{\text{MULT}}(CW, n_i, n_o). \tag{4}$$

The area of a unit sample delay is implemented as a register. Hence, with an input $i$, the area and power model of a register can be formulated as

$$A_{\text{REGISTER}} = f_{\text{REGISTER}}(n_i) \tag{5}$$

$$P_{\text{REGISTER}} = g_{\text{REGISTER}}(n_i). \tag{6}$$

These models satisfy the previous monotonic properties.

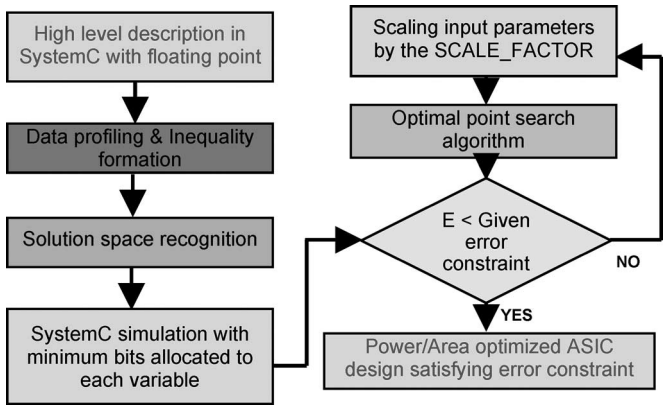


Fig. 1. Flow diagram of the optimization algorithm.

## IV. Algorithm for Power Optimization

We now describe our automated algorithm "QUANTA-SMART" for power optimization while constraining the round-off errors. The algorithm consists of a compiler pass followed by simulation and synthesis for the optimization purpose. We first propose a heuristic "greedy" search algorithm followed by a "smart" search algorithm that gives the optimal solution for two variables. A flow diagram describing the steps involved in the optimization process is shown in Fig. 1.

We subdivide the process into three basic steps:

1) data profiling and inequality formation;
2) solution space recognition;
3) optimal point search within given error constraint.

### A. Data Profiling and Inequality Formation

The quantization algorithm can be used for various types of applications including those for speech processing and filter applications. A small percentage of the actual inputs to the system are used in our algorithm. The quantization algorithm gives the optimal set of quantizers using the sample input. This is known as "training of the system." Once we develop an optimized hardware using our algorithm, we use a larger percentage of the actual inputs to test our system. We test the hardware with actual inputs and verify that the true quantization error is within the acceptable limit. During training of the system, we can employ a factor of safety for the previously defined error metric $(E)$ constraint. For example, if our $E$ constraint of the system is 5%, and we use a 10% factor of safety, then the $E$ constraint fed to the quantization algorithm is $= 5 \times (1 - 0.1) = 4.5\%$. For a system in which a smaller sample closely resembles the input signal, the factor of safety can be kept low. As we train the system with a very small percentage of the actual inputs, the process is efficient in terms of run times. During the synthesis of the design, we use only integer values to preserve the precision of the parameters (as described in Section III-A). If the inputs of the design do not contain any integer part, we scale it.

We next perform a compiler pass over the program to assign bit length (or width) values to each variable in the program. The following set of rules is proposed while assigning bit length to

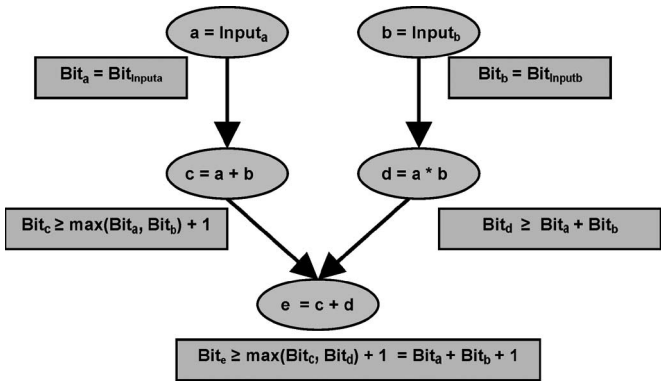


Fig. 2. Bit length assignment rules.

any variable. Fig. 2 illustrates the flow of the different rules within a design block

Addition rule: $a = b + c$;
$\text{Bitlength}_a = \text{maximum}(\text{Bitlength}_b, \text{Bitlength}_c) + 1$
Multiplication rule: $a = b \times c$;
$\text{Bitlength}_a = (\text{Bitlength}_b + \text{Bitlength}_c)$
Loop rule: For an accumulator variable within a loop that iterates $N$ (a fixed number known at compile time) times
for $(i = 0; i < N; i++)$ {
accumulator$+=$ sample;
}
$\text{Bitlength}_{\text{accumulator}} = (\text{Bitlength}_{\text{sample}} + \log_2 N)$

We illustrate our approach using the example SystemC code for a finite-impulse response (FIR) filter in Fig. 3. In the code, we have two input parameters ("sample" and "coeff"). We assign $x$ and $y$ bits to these variables, respectively. Let us assume that for a given data input set, we need at least 7 and 8 bits to represent the input variables. The minimum bits needed to represent the inputs can be calculated using range propagation on the training set.

Our required inequalities can therefore be formulated as

$$\text{sample}_{\text{BIT}} \geq 7 \tag{7}$$
$$\text{coeff}_{\text{BIT}} \geq 8 \tag{8}$$
$$\text{sample\_tmp}_{\text{BIT}} = \text{sample}_{\text{BIT}} \tag{9}$$
$$\text{pro}_{\text{BIT}} = \text{shift}_{\text{BIT}} + \text{coeff}_{\text{BIT}} + 1 \tag{10}$$
$$\text{acc}_{\text{BIT}} = \text{pro}_{\text{BIT}} + \log_2(\text{NUMTAPS}) \tag{11}$$
$$\text{shift}_{\text{BIT}} = \text{sample}_{\text{BIT}} \tag{12}$$
$$\text{result}_{\text{BIT}} = \text{acc}_{\text{BIT}}. \tag{13}$$

An analysis of the equations reveals that the variables acc and result would store the largest value. Hence, these two variables have to maintain the constraint on the largest ⟨sc_int⟩ that can be synthesized. This constraint gives us the inequalities

$$32 \geq \text{result}_{\text{BIT}} \tag{14}$$
$$32 \geq \text{acc}_{\text{BIT}}. \tag{15}$$

```
while(1) {
  output_data_ready.write(false);
  wait_until(input_valid.delayed() == true);
  sample_tmp = sample.read();  ← sample_tmpBIT = sampleBIT = x
  acc = 0;    acc = sample_tmp*coefs[0];
  for(int i=NUMTAPS; i>0; i--) {
   pro = shift[i-1]*coefs[i];        ← proBIT = shiftBIT + coeffBIT + 1
   acc = acc + shift[i-1]*coefs[i]; ← accBIT = proBIT + log2(NUMTAPS)
  };
   for(int i=NUMTAPS-1; i>=0; i--)
        shift[i+1] = shift[i];
  shift[0] = sample_tmp;  ←  shiftBIT = sampleBIT
  result.write(acc);            ←  resultBIT = accBIT
  output_data_ready.write(true);
  wait();
 };
```

Fig. 3. Example of a SystemC code segment.

```
While (E > ERROR_CONSTRAINT)  {
 For i = 1 to N = No_of_variables {
       Scale  the independent  variable  along
       co-ordinate i by the SCALE_FACTOR and
       simulate the design to get Error Metric E[i];
     }
 E = Minimum of E[i];
Change the variable along dimension i into the new scaled value;
}
```

Fig. 4. Pseudocode of the greedy search algorithm.

### B. Solution Space Recognition

The previous set of inequality relations describes the solution space satisfying the given error constraints. When we solve the inequalities, we come up with a range within which we can vary the bit length of the independent variables (here $\text{sample}_{\text{BIT}}$ and $\text{coeff}_{\text{BIT}}$). We also modify the bit lengths used by the intermediate variables whenever we make a change in the independent variable bit lengths. By solving these inequalities, we obtain a range of bit lengths within which $\text{sample}_{\text{BIT}}$ and $\text{coeff}_{\text{BIT}}$ can vary. In effect, this defines the solution space of the optimization problem. From the example (for NUMTAPS $=$ 4), we get the solution space as

$$19 \geq \text{sample}_{\text{BIT}} \geq 7 \tag{16}$$
$$20 \geq \text{coeff}_{\text{BIT}} \geq 8. \tag{17}$$

### C. Search Algorithm for Optimal Fix Point

Once we have narrowed down the solution space using solutions of the inequalities, we start to simulate the synthesizable code to get an error estimation of the optimized design. We propose two different algorithms for searching the optimal bit-length allocations.

*1) Greedy Search Algorithm:* The first method that we use is a heuristic greedy algorithm followed by refined local search techniques (Fig. 4). We model the problem as a search problem for bit precisions in an $N$-dimensional search space of independent variables.

*Suboptimal point search:* We start with the search for a suboptimal point in the solution space that involves a greedy

```
For i = 1 to N = No_of_variables {
    Scale down the independent variable along
      dimension i by the FINER_SCALE ;
      Simulate the design to get Error Metric E.
    If  (E< ERROR_CONTRAINT)
          Estimate the power P[i];
    }
```

Fig. 5. Pseudocode of the local search algorithm.

stepwise search algorithm. Our aim is to reach a near-optimal solution with the help of fast SystemC simulation. The search is narrowed down for an optimal solution with a given error constraint into a very small solution space. Subsequently, we start with the minimum permissible bit length allowed for the independent variables and adjust the intermediate variables using the inequality relations.

Assuming that a design has $n$ independent variables, we allocate extra $\log_2(\text{SCALE\_FACTOR})$ bits to one of them. The input data corresponding to that independent variable is scaled by SCALE_FACTOR, and the error metric $E$ is recorded after simulating that particular configuration. The input variable that affects the error value most favorably (i.e., causes the maximal decrease) is chosen as the next step. We repeat the same steps for the new configuration obtained to reach a better solution, which takes extra precision into account with the help of scaling. The process ends when we obtain a point that satisfies the given error constraint for the design.

*Local search:* We can further optimize the solution by searching the neighbors of the suboptimal point in the solution space. To perform that, we implement a local search algorithm (Fig. 5) at the suboptimal point. Here, we search for other solution candidates among the neighbors and finally settle for the solution that satisfies the given error bound and minimizes power consumption. This is done by scaling down the independent variables along each dimension by a constant FINER_SCALE that is less than SCALE_FACTOR. We calculate the error metric $E$ value and the power consumed by each of these new designs. In the example given in Fig. 3, we scale both variables $\text{sample}_{\text{BIT}}$ and $\text{coeff}_{\text{BIT}}$ by 64 to reach the suboptimal point. During the local search procedure, it is observed that if we save a single bit in the $\text{sample}_{\text{BIT}}$ variable, i.e., scale it down by 2 ($<$ SCALE_FACTOR $= 4$), we can get a solution that has the least power consumption. Hence, for the given example, a design with $\text{sample}_{\text{BIT}}$ scaled by 32 and $\text{coeff}_{\text{BIT}}$ scaled by 64 gives us the optimized solution. Note that, in the worst case, this greedy search algorithm has an exponential time complexity.

*2) Smart Search Algorithm:* In the greedy search approach, we started looking for the solution from the configuration where all the independent variables have the least bit length. This section discusses another approach that results in a provably optimum solution. It can be argued that the heuristic greedy search algorithm discussed in the previous section is not guaranteed to give an optimal solution.

If one or more suboptimal solutions exist along the solution space boundary, the optimal solution is not obtained. A possible modification would be to traverse the outer boundary of the solution space in search for an optimal solution. This results in a significant increase in complexity. We propose a smart

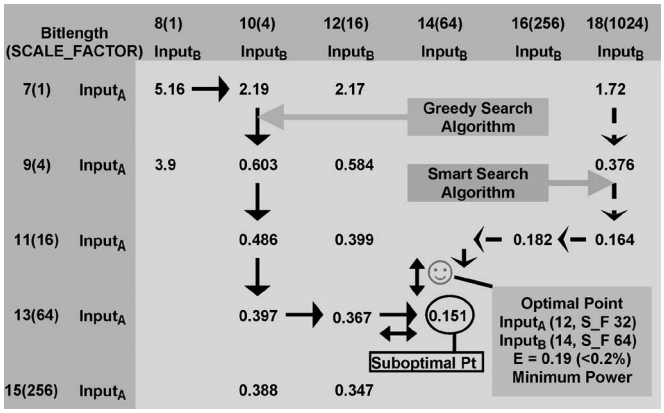


Fig. 6. Illustration of the search algorithm.

algorithm (Fig. 6) that can be proven to be optimal for designs where we have two independent variables. The algorithm is trivially true for designs with a single independent variable. For designs having more than two independent variables, we propose a heuristic solution.

*Designs with two independent variables:* Let us consider a design with two independent variables $X$ and $Y$. After the analysis in the first two steps, we can deduce some $m$ bit range for $X$ as $(x_1 < x_2 < \ldots < x_m)$ and an $n$ bit range for $Y$ as $(y_1 < y_2 < \ldots < y_n)$. The solution space can be assumed to be a grid of $(n \times m)$ cells, where each cell corresponds to a particular configuration of the design. The smart algorithm is described as follows.

1) Start searching from either $(x_1, y_n)$ or $(x_m, y_1)$ position. One of the variables has the highest permissible precision while the other has the least precision.
2) Calculate the error metric $E(i, j)$ for the point $(x_i, y_j)$ using simulation.
3) If $E(i, j)$ satisfies the error constraint and is lower than the previously recorded value, store it and decrease the bit length of the higher-precision variable by one. Revert to step 2).
4) Otherwise, allocate an extra bit to the lower-precision variable and go back to step 2).

The process will end when the higher-precision variable can no longer be reduced or the low-precision variable can no longer be increased. The optimal configuration is recorded during the traversal.

*Proof of optimality:* We next prove that our proposed algorithm will always yield the optimal solution. We will also prove that $(m + n - 1)$ simulations are sufficient to find the optimal solution using the smart algorithm.

Without any loss of generality, let us assume that we start from the $(x_1, y_n)$ position. So $Y$ is the high-precision variable and $X$ is the low-precision variable. We claim that when the algorithm reaches a point $(x_i, y_j)$, the optimal solution OPT of the problem can be expressed as

$$P\text{:OPT} = \text{BEST}\left(\text{record}, \text{BEST}\left\{(x_k, y_l) | i \le k \le m, 1 \le l \le j\right\}\right)$$

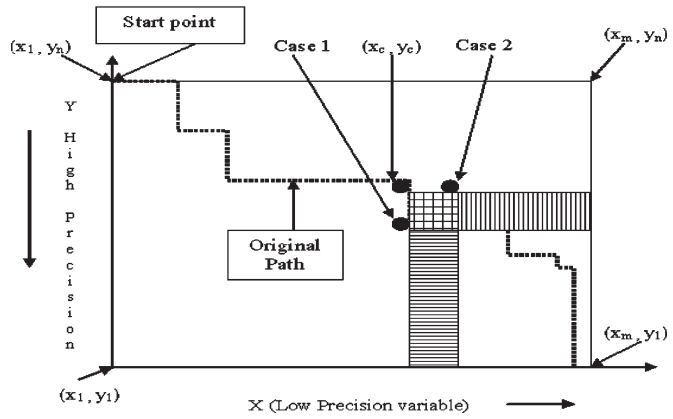


Fig. 7. Illustration of the smart search algorithm.

where record is a valid configuration with minimum power consumption checked so far, and BEST$(S)$ denotes a solution in $S$ satisfying the precision constraint and having the minimal power consumption.

We can show that the above claim is an invariant property. At the start of the search, the property $P$ holds true as record $= \{\}$ and $i = 1,\ y = n$. So, the best solution of $\{(x_k, y_l) | 1 \leq k \leq m, 1 \leq l \leq n\}$ is indeed the optimal solution. At the end of the search, we would have $i > m$ or $j < 1$. Thus, $\{(x_k, y_l) | i \leq k \leq m, 1 \leq l \leq j\} = \{\}$, and hence record $=$ OPT.

Let us assume that the property holds for a point $(x_i, y_j)$ in the solution space. In the next iteration, the algorithm can move in either of the two directions in the solution space (Fig. 7).

Case 1) The algorithm will move along the $Y$-direction, i.e., the high-precision variable direction. The partial order relation among the points in solution space suggests that all the points for which $(y = y_j)$ and $(x > x_i)$ will also satisfy the error constraint. However, they will consume no less power, which excludes them from being the optimal solution. The record will be updated accordingly. After the point moves in the $Y$-direction, it is thus evident that the property $P$ still holds.

Case 2) The algorithm will move along the $X$-direction i.e., the low-precision variable. From the existing partial order, we can exclude all invalid points for which $(y < y_j)$ and $(x = x_i)$. Thus, the record value will not change, and the unexplored solution space will reduce. Additionally, we will not exclude any point that could be a possible candidate of the optimal solution. The property $P$ is still true after the increase of $i$.

Therefore, using the principle of induction, we can prove that for all points traversed by the property, $P$ will hold, and hence at termination, it is bound to give the optimal solution.

Further, the algorithm always forces the high-precision variable direction toward reduction and the low-precision variable toward an increase. This ensures that we can traverse at most $(m + n - 1)$ simulation points in the solution space. Thus, the algorithm executes in linear time $O(m + n)$.

*Selection of high- and low-precision variables:* We use a heuristic to select the high- and low-precision variables for the algorithm. We consider two configurations, namely: 1) least bit lengths and 2) largest possible bit lengths, for both independent variables. For both configurations, we change the bit length along each of the variable and observe the effect on the error metric. If they suggest the same variable as more sensitive to precision, then we pick that variable as the high-precision variable. Otherwise, we pick the variable that has a smaller range of bit variance as the high-precision variable.

*Extension to higher dimensions:* The smart algorithm is proven to be optimal for designs with two independent variables. For designs having more than two independent variables, it can be proved that a similar approach would result in an exponential time complexity. Thus, we propose a heuristic approach for designs having more than two independent variables. Using the same heuristic as discussed earlier, we propose to choose two most sensitive variables for better precision. All other variables would be fixed to the highest permissible value and get the optimal solution. We would fix the two selected variables to the optimal solution configuration value and repeat the step with picking up two variables ranked next in the precision sensitivity list. This step would be continued until we fix all the independent variable bit lengths.

## V. Experimental Results

In this section, we report experimental results on the following SystemC benchmarks:

1) a 16-tap FIR filter (fir);
2) an interpolation FIR filter (intfir);
3) a decimation in time FIR filter (decfir);
4) a least mean square adaptive filter (lms).

We have used SystemC (version 2.0.1) to simulate the fixed/floating-point codes. We took measurements for optimal quantizers and $E$ corresponding to the inputs. We have used a sinusoidal wave input of size 1024. Subsequently, we used the Cocentric Behavioral Compiler for SystemC [9] to generate RTL VHSIC Hardware Description Language from SystemC benchmarks. Using the same tool where the Synopsys Design Compiler runs in the backend, we synthesized all the designs into 0.18-$\mu$m technology ASIC cell library from Artisan Components. This gave us an idea about the area consumed by the optimized design. Finally, we used the Synopsys Power Compiler to get the power values related to each synthesized design. We selected the design with the least amount of power consumption and having an $E$ within a given ERROR_CONSTRAINT as the final optimized ASIC design.

Tables I and II show the optimal bit length of input parameters selected by the greedy and smart search algorithm, respectively, for $E$ constraints of 5%, 1%, and 0.5% using simulation with sinusoidal inputs. We demonstrate that it is possible to reduce the power consumption by 50% on the average by allowing errors to increase from 0.5% to 1%.

The greedy search heuristic gave the optimal solution in 11 out of the 12 cases for sinusoidal input (Benchmark FIR16, for $E \leq 0.5\%$). However, for smaller error constraints (0.5%), it

TABLE I
EXPERIMENTAL RESULTS OF TRADEOFFS BETWEEN QUANTIZATION ERROR AND POWER AREA FOR FOUR BENCHMARKS WITH SINUSOIDAL INPUT (TRAINING SET OF 5% AND TESTING SET OF 95%; FACTOR OF SAFETY = 10%)

**GREEDY SEARCH ALGORITHM**

| Error Constraint | Bit-width of the Independent Variables | Power Consumed ($10^4$ nW) | Area ($10^{-12}$ $m^2$) |
|---|---|---|---|
| **FIR16** | | | |
| E ≤ 5% | input= 7; coeff = 8 | 91.1 | 12454 |
| E ≤ 1% | input=9; coeff= 9 | 101 | 13243 |
| E ≤ 0.5% | input=14; coeff=12 | 404 | 20449 |
| **INTFIR** | | | |
| E ≤ 5% | input = 7; coeff =13 | 450 | 104209 |
| E ≤ 1% | input=10; coeff= 13 | 565 | 138909 |
| E ≤ 0.5% | input=13; coeff=14 | 1070 | 157775 |
| **DECFIR** | | | |
| E ≤ 5% | input=7; coeff= 13 | 6790 | 23690 |
| E ≤ 1% | input=9; coeff= 13 | 8980 | 32959 |
| E ≤ 0.5% | input=11;coeff= 13 | 16100 | 42674 |
| **LMS** | | | |
| E ≤ 5% | input = 10 | 30900 | 132984 |
| E ≤ 1% | input = 13 | 40700 | 175692 |
| E ≤ 0.5% | input = 14 | 45400 | 197479 |

TABLE II
EXPERIMENTAL RESULTS OF TRADEOFFS BETWEEN QUANTIZATION ERROR AND POWER AREA FOR FOUR BENCHMARKS WITH SINUSOIDAL INPUT (TRAINING SET OF 5% AND TESTING SET OF 95%; FACTOR OF SAFETY = 10%)

**SMART SEARCH ALGORITHM**

| Error Constraint | Bit-width of the Independent Variables | Power Consumed ($10^4$ nW) | Area ($10^{-12}$ $m^2$) |
|---|---|---|---|
| **FIR16** | | | |
| E ≤ 5.0% | input= 7; coeff = 8 | 91.1 | 12454 |
| E ≤ 1.0% | input=9; coeff= 9 | 101 | 13243 |
| E ≤ 0.5% | input=12; coeff=14 | 376 | 19844 |
| **INTFIR** | | | |
| E ≤ 5.0% | input = 7; coeff =13 | 450 | 104209 |
| E ≤ 1.0% | input=10; coeff= 13 | 565 | 138909 |
| E ≤ 0.5% | input=13; coeff=14 | 1070 | 157775 |
| **DECFIR** | | | |
| E ≤ 5.0% | input=7; coeff= 13 | 6790 | 23690 |
| E ≤ 1.0% | input=9; coeff= 13 | 8980 | 32959 |
| E ≤ 0.5% | input=11;coeff= 13 | 16100 | 42674 |
| **LMS** | | | |
| E ≤ 5.0% | input = 10 | 30900 | 132984 |
| E ≤ 1.0% | input = 13 | 40700 | 175692 |
| E ≤ 0.5% | input = 14 | 45400 | 197479 |

took more simulations to reach the solution point, as shown in Table III.

Finally, we present the execution time for the test cases in Table IV. For comparison purposes, we looked at the performance of our approaches with respect to the best results obtained in earlier work done by Roy and Banerjee [13]. Fig. 8 plots the relative change in run times for similar benchmarks used in both studies (for randomized input of size 2048). We can see in all the three cases that our approach outperforms the earlier work.

TABLE III
COMPARATIVE STUDY BETWEEN GREEDY SEARCH ALGORITHM AND SMART SEARCH ALGORITHM FOR NUMBER OF SIMULATIONS REQUIRED TO REACH AN OPTIMAL POINT UNDER GIVEN ERROR CONSTRAINTS

| Error Constraint | Number of Simulations Required to Reach the Optimal Point | |
|---|---|---|
| | Greedy Search | Smart Search |
| **FIR16** | | |
| E ≤ 5.0% | 4 | 4 |
| E ≤ 1.0% | 7 | 6 |
| E ≤ 0.5% | 17 | 9 |
| **INTFIR** | | |
| E ≤ 5.0% | 1 | 4 |
| E ≤ 1.0% | 5 | 3 |
| E ≤ 0.5% | 23 | 17 |
| **DECFIR** | | |
| E ≤ 5.0% | 1 | 3 |
| E ≤ 1.0% | 3 | 2 |
| E ≤ 0.5% | 5 | 3 |
| **LMS** | | |
| E ≤ 5.0% | 4 | 3 |
| E ≤ 1.0% | 5 | 2 |
| E ≤ 0.5% | 6 | 1 |

TABLE IV
COMPARATIVE STUDY BETWEEN GREEDY SEARCH ALGORITHM AND SMART SEARCH ALGORITHM FOR TOTAL EXECUTION TIME (PROJECTED) REQUIRED TO REACH AN OPTIMAL POINT FOR GIVEN ERROR CONSTRAINTS

| Error Constraint | Simulation Time Required to Reach the Optimal Point (in seconds) | | | |
|---|---|---|---|---|
| | Sinusoidal Input | | Random Input | |
| | Greedy Search | Smart Search | Greedy Search | Smart Search |
| **FIR16** | | | | |
| E ≤ 5.0% | 1.52 | 1.51 | 2.32 | 2.25 |
| E ≤ 1.0% | 2.66 | 2.28 | 4.05 | 4.05 |
| E ≤ 0.5% | 6.46 | 3.42 | 8.55 | 5.4 |
| **INTFIR** | | | | |
| E ≤ 5.0% | 0.755 | 3.02 | 0.45 | 2.25 |
| E ≤ 1.0% | 3.775 | 2.265 | 3.15 | 1.35 |
| E ≤ 0.5% | 17.365 | 12.835 | 12.15 | 7.2 |
| **DECFIR** | | | | |
| E ≤ 5.0% | 0.12 | 0.36 | 0.5 | 1.7 |
| E ≤ 1.0% | 0.36 | 0.24 | 1.5 | 1.2 |
| E ≤ 0.5% | 0.60 | 0.36 | 2.25 | 1.8 |
| **LMS** | | | | |
| E ≤ 5.0% | 5.28 | 3.96 | 1.8 | 1.35 |
| E ≤ 1.0% | 6.6 | 2.64 | 2.25 | 0.9 |
| E ≤ 0.5% | 7.92 | 1.32 | 2.7 | 0.45 |

## VI. CONCLUSION

This paper describes algorithms to optimize the bit widths of fixed-point variables for low power in a SystemC design environment. We propose an algorithm for optimal bit-width precision for two variables and a greedy heuristic that works for any number of variables. The algorithms are used in the automation of converting floating-point SystemC programs into ASIC synthesizable SystemC programs. The results show that it is possible to tradeoff quantization error with the hardware

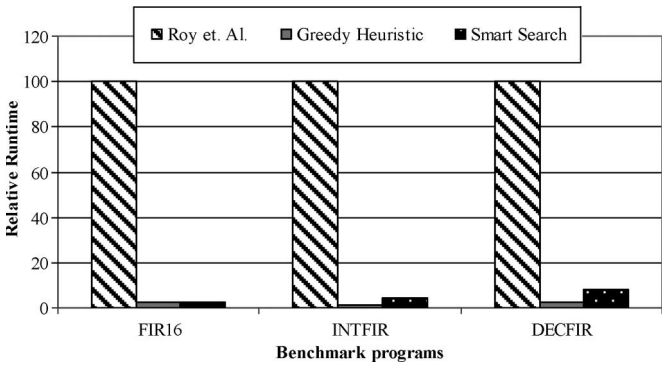


Fig. 8. Runtime comparison among different approaches (randomized input with 5% error constraint).

resources used in the ASICs very effectively. The ideas introduced in this paper can be extended to other programming languages such as MATLAB and SIMULINK and to other technologies such as field-programmable gate arrays.

## REFERENCES

[1] P. R. Panda, "SystemC: A modeling platform supporting multiple design abstractions," in *Proc. Int. Symp. Syst. Synthesis*, Montréal, QC, Canada, 2001, pp. 75–80.
[2] K. H. Chang and W. G. Bliss, "Finite word-length effects of pipelined recursive digital filters," *IEEE Trans. Signal Process.*, vol. 42, no. 8, pp. 1983–1995, Aug. 1994.
[3] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE Trans. Inf. Theory*, vol. 44, no. 6, pp. 2325–2383, Oct. 1998.
[4] H. Keding, M. Willems, M. Coors, and H. Meyr, "FRIDGE: A fixed-point design and simulation environment," in *Proc. Des. Autom. Test Eur.*, 1998, pp. 429–435.
[5] Synopsys Inc. (2001). *CoCentric SystemC Compiler Behavioral Modelling Guide.* [Online]. Available: www.synopsys.com
[6] Synopsys Inc. (2000). *Synopsys CoCentric Fixed-Point Designer Datasheet.* [Online]. Available: www.synopsys.com
[7] M. Stephenson, J. Babb, and S. Amarasinghe, "Bitwidth analysis with application to silicon compilation," in *Proc. SIGPLAN Conf. Program. Language Des. and Implementation*, Vancouver, BC, Canada, 2000, pp. 108–120.
[8] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee, "Precision and error analysis of MATLAB applications during automated hardware synthesis for FPGAs," in *Proc. Des. Autom. Test Eur.*, 2001, pp. 722–728.
[9] P. Banerjee, D. Bagchi, M. Haldar, A. Nayak, V. Kim, and R. Uribe, "Automatic conversion of floating point MATLAB programs into fixed point FPGA based hardware design," in *Proc. FCCM*, 2003, pp. 263–268.
[10] G. A. Constantinides, "Perturbation analysis for word-length optimization," in *Proc. FCCM*, 2003, pp. 81–90.
[11] M. L. Chang and S. Hauck, "Precis: A design-time precision analysis tool," in *Proc. FCCM*, 2002, pp. 229–238.
[12] S. Roy, D. Sinha, and P. Banerjee, "An algorithm for trading off quantization error with hardware resources for MATLAB based FPGA design," in *Proc. Int. Symp. FPGA*, 2004, pp. 886–896.
[13] S. Roy and P. Banerjee, "An algorithm for converting floating point computations to fixed point computations in MATLAB based hardware design," in *Proc. DAC*, 2004, pp. 484–487.
[14] D. Brooks and M. Martonosi, "Dynamically exploiting narrow width operands to improve processor power and performance," in *Proc. HPCA*, 1999, pp. 13–22.
[15] S. Gupta and F. N. Najm, "Power macromodeling for high level power estimation," in *Proc. 34th Des. Autom. Conf.*, 1997, pp. 365–370.
[16] L. Codrescu and S. Wills, "Profiling for input predictable threads," in *Proc. ICCD*, 1998, pp. 558–565.
[17] R. B. Hildendorf, G. J. Heim, and W. Rosenstiel, "Evaluation of branch prediction using traces for commercial applications," *IBM J. Res. Develop.*, vol. 43, no. 4, pp. 579–593, 1999.
[18] S. Gupta and F. N. Najm, "Power macromodeling for high level power estimation," in *Proc. 34th Des. Autom. Conf.*, 1997, pp. 365–370.
[19] L. B. Jackson, "On the interaction of roundoff noise and dynamic range in digital filters," *Bell Syst. Tech. J.*, vol. 49, no. 1, pp. 159–183, Feb. 1970.
[20] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1999.
[21] L. B. Jackson, K. H. Chang, and W. G. Bliss, "Comments on finite word-length effects of pipelined recursive digital filters," *IEEE Trans. Signal Process.*, vol. 43, no. 12, pp. 3031–3032, Dec. 1995.
[22] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Optimum wordlength allocation," in *Proc. FCCM*, 2002, pp. 219–228.
[23] ——, "The multiple wordlength paradigm," in *Proc. IEEE Symp. Field-Programmable Custom Comput. Mach.*, 2001, pp. 51–60.
[24] C. Shi, "Statistical method for floating-point to fixed point conversion," M.S. thesis, Elect. Eng. and Comput. Sci. Dept., Univ. California, Berkeley, CA, 2002.
[25] A. G. Braun, B. F. Jan, J. Gerlach, and W. Rosenstiel, "Automated conversion of SystemC fixed-point data types for hardware synthesis," in *Proc. IFIP VLSI SOC*, 2003, pp. 119–132.
[26] P. Belanovic and M. Rupp, "Automated floating-point to fixed-point conversion with the fixify environment," in *Proc. Int. Workshop RSP—Vortrag*, Montreal, QC, Canada, 2005, pp. 172–175.

**Arindam Mallik** (S'03) received the B.E. degree in computer science and engineering from Jadavpur University, Calcutta, India, in 2002, and the M.S. degree in computer engineering from Northwestern University, Evanston, IL, in 2003. He is currently working toward the Ph.D. degree in electrical engineering and computer science at Northwestern University.

He spent the summer of 2005 as a summer intern with Cswitch Corporation, Santa Clara, CA, working on reconfigurable network processors. In 2006, he worked on many core processor power modeling with the System Technology Group at Intel Corporation, Hillsboro, OR. His research interests include microarchitecture, network processors, embedded systems, and reliability issues in high-performance computing.

Mr. Mallik is a Royal Cabell Fellow of Northwestern University for the year 2006–2007. He received the Walter P. Murphy Fellowship Award from Northwestern University in 2002. He was a national scholar from 1996 to 2002 under the National Talent Search Scheme, India.

**Debjit Sinha** (M'06) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Kharagpur, India, in 2001, and the Ph.D. degree in electrical engineering and computer science from Northwestern University, Evanston, IL, in 2006.

Between 2001 and 2002, he worked on a project for National Semiconductor Corporation involving the research and development of software tools for compact RISC processors. He was a Summer Intern with the Advanced Technology Group, Synopsys Inc., where he worked on statistical approaches for timing analysis and optimization, and study of process technologies on interconnect parasitics in 2004 and 2005, respectively. He is currently an Advisory Engineer with IBM Microelectronics, East Fishkill, NY. His research interests include algorithms and computer-aided design for very large scale integrated circuits, especially related to crosstalk noise analysis, and statistical timing analysis and optimization.

**Prithviraj Banerjee** (S'82–M'84–SM'90–F'95) received the B.Tech. degree in electronics and electrical engineering from the Indian Institute of Technology, Kharagpur, India, in 1981, and the M.S. and Ph.D. degrees in electrical engineering from the University of Illinois, Urbana-Champaign, in 1982 and 1984, respectively.

He was the Director of the computational science and engineering program and a Professor of electrical and computer engineering with the University of Illinois at Urbana-Champaign. He was also a Walter P. Murphy Professor and the Chairman of the Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL. In 2000, he founded AccelChip Inc., a developer of products and services for electronic design automation. He served as President and CEO of AccelChip while on leave from Northwestern University from 2000 to 2002. In addition, he has been on the Technical Advisory Boards of many companies including Atrenta, Calypto Design Systems, and Ambit Design Systems. He is currently the Dean of the College of Engineering and a UIC Distinguished Professor with the University of Illinois at Chicago (UIC). He is the author of about 300 research papers and a book entitled *Parallel Algorithms for VLSI CAD*. He has supervised 35 Ph.D. and 40 M.S. student theses thus far. His research interests are in VLSI computer-aided design, parallel computing, and compilers.

Dr. Banerjee became a Fellow of AAAS and ACM in 2006 and 2000, respectively. He has served as the Program Chair, General Chair, and Program Committee of more than 50 conferences in the past 20 years and has served as the Associate Editor of four journals. He has received numerous awards and honors during his career. He received the President of India Gold Medal from the Indian Institute of Technology in 1981, the IBM Young Faculty Development Award in 1986, the National Science Foundation's Presidential Young Investigators' Award in 1987, the Senior Xerox Research Award in 1992, and the University Scholar award from the University of Illinois for in 1993. He was the recipient of the 1996 Frederick Emmons Terman Award of ASEE's Electrical Engineering Division. He received the IEEE Taylor L. Booth Education Award from the IEEE Computer Society in 2001.

**Hai Zhou** (M'04–SM'04) received the B.S. and M.S. degrees in computer science and technology from Tsinghua University, Beijing, China, in 1992 and 1994, respectively, and the Ph.D. degree in computer science from the University of Texas, Austin, in 1999.

He was with the Advanced Technology Group, Synopsys Inc. He is currently an Assistant Professor of electrical engineering and computer science with Northwestern University, Evanston, IL. His research interests include very large scale integrated computer-aided design, algorithm design, and formal methods.

Dr. Zhou has served on the technical program committees of many conferences on very large scale integrated circuits and computer-aided design. He was the recipient of the CAREER Award from the National Science Foundation in 2003.