Smart Bit-width Allocation for Low Power Optimization in a SystemC based ASIC Design Environment[1]

Arindam Mallik, Debjit Sinha, Prith Banerjee†, Hai Zhou

Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208, USA
{arindam, debjit, haizhou}@ece.northwestern.edu

†College of Engineering
University of Illinois at Chicago,
Chicago, IL 60607, USA
prith@uic.edu

Abstract

The modern era of embedded system design is geared towards design of low-power systems. One way to reduce power in an 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 design environment. We propose an algorithm for optimal bit-width precision for two variables and a greedy heuristic which 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 on the trade-offs between quantization error, power consumption and hardware resources used are reported on a set of four SystemC benchmarks that are mapped onto 0.18 micron ASIC cell library from Artisan Components. We demonstrate that it is possible to reduce the power consumption by 50% on average by allowing round-off errors to increase from 0.5% to 1%.

1 INTRODUCTION

Power consumption has become a primary design criterion for modern embedded systems. The majority of low power design and analysis tools in the EDA industry target low levels of chip design, such as the transistor, gate and RTL levels. 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 result in an immediate gain in power savings and area of a design. However, this adversely affects the accumulation of quantization errors due to finite precision arithmetic. Most practical 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 IP exchange. It has been developed as a standardized modeling language intended to enable system level design and IP exchange at multiple abstraction levels, for systems containing both hardware and software components.

[1] This research was supported by NASA under contract 276685, and DARPA under contract no F33615-01-C-1631.

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

This paper describes an algorithm 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 which 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.

The rest of the paper is organized as follows. The next section describes related work. Section 3 presents the area, delay and error models used in our SystemC designs. Section 4 gives the details of the proposed optimization algorithms. Section 5 discusses the experimental setup and results. Section 6 concludes the paper by summarizing our contributions.

2 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 [2, 3]. The other approach is based on bit-true simulation techniques used by hardware designers [4].

There has been some work in the recent literature on automated compiler techniques for conversion of floating point representations to fixed-point representations [5, 6].

The BITWISE compiler [7] determines the precision of all input, intermediate and output signals in a synthesized hardware design from a C program description. The MATCH compiler [8] develops precision and error analysis techniques for MATLAB programs. Synopsys has a commercial tool called the Cocentric Fixed-Point Designer [6], which automatically converts floating-point

3-9810801-0-6/DATE06 © 2006 EDAA

computations to fixed point within a C compilation framework. However, the code generated is not synthesizable. Constantinides [10] has developed a design tool to tackle both linear and nonlinear designs. Chang et al. [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 which the compiler is unable to infer. Recently, Roy et al. [12] 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.

3 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.

3.1 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++. The behavioral synthesis tools available for SystemC such as the Synopsys Cocentric compiler [9] do not support the synthesis of fixed-point data types.

We propose the use of the *sc_int*<> data type which represents the fixed precision signed integer to preserve the data precision. When we scale any 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 a parameter. We can specify the number of bits which 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*<> type. sc_int<12> sample_tmp;

The above 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. Note that, we use scale-factors that are always powers of two. This facilitates the task of allocating extra bits to the variables. During the first pass over the program (details are provided in Section 4.1) we formulate several relations for bits allocated to each variable in the program. Due to the scaling of the input variables by a SCALE_FACTOR, if any variable in the program gets scaled by $SCALE\_FACTOR^n$ (n=0, 1, 2 ..) we allocate $\log_2(SCALE\_FACTOR^n)$ bits to it.

3.1.1 Error Model

The introduction of the SystemC model enables convenient and fast simulation of designs by using the lightweight cycle-based simulation kernel. In our approach we compute the errors using simulations and consider non-increasing 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

$E = | Output_{float} - Output_{fixed} | / Output_{float}$

where, $Output_{float}$ denotes the vector of floating point output values for training set inputs and $Output_{fixed}$ denotes the vector of corresponding output values after scaling.

3.2 Area and Power Model Analysis

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

We assume that 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. Consequently, optimizing the bit-length of different parameters of a design results in low power consumption. Moreover, bit-length optimization of different parameters reduces the interconnect overhead of the circuit. We use two abstract function 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 co-ordinates. Hence, a design X in that solution space can be expressed as $(x_1, x_2, x_3, \ldots\ldots x_k)$. 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 obtained for the configuration X, then we can claim that,

$A(X) \leq A(Y)$ If $x_i \leq y_i$, for all i = 1, 2, 3, … k.
$P(X) \leq P(Y)$ If $x_i \leq y_i$, for all i = 1, 2, 3, … k.
$E(X) \geq E(Y)$ If $x_i \leq y_i$, for all i = 1, 2, 3, … k.

These relations conclude that the abstract area function f and power function g are monotonically non-decreasing 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 - $A_{ADDER} = f_{ADDER}(n_a, n_b)$ (1)

Then, the power consumption can be modeled as –

$P_{ADDER} = A_{ADDER}.V_{DD}^2.f(activity) = g_{ADDER}(n_a,n_b)$ (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 ni bits as input and no 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,

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

$P_{MULT} = g_{MULT}(CW, n_i, n_o)$ (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 ,

$A_{REGISTER} = f_{REGISTER}(n_i)$ (5)

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

These models satisfy the above monotoic properties

4 ALGORITHM FOR POWER OPTIMIZATION

We now describe our automated algorithm, QUANTASMART, for power optimization while constraining the round off errors. It consists of a compiler pass followed by simulation and synthesis for the optimization purpose. First we propose a heuristic Greedy search algorithm followed by a Smart Search Algorithm which gives the optimal solution for two variables. A flow diagram describing the steps involved in the whole process of optimization is shown in Figure 1.

We subdivide the process into three basic steps :

- Data profiling and Inequality formation
- Solution space recognition
- Optimal point search within given error constraint

4.1 Data Profiling and Inequality Formation

The quantization algorithm can be used for various types of applications such as speech processing applications, filter applications. A small percentage of the actual inputs to a system are used for 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. Thus we now test the hardware with actual inputs and verify that the actual quantization error is within the acceptable limit. While training the system we can use a factor of safety for the Error Metric (E) constraint (defined in section 3.3). For example if our E constraint of the actual system is 5%, and we use a factor of safety=10%, then the E constraint fed to the quantization algorithm is 5*(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 very low. As we train the system with a very small percentage of the actual inputs, this process is fast. During the synthesis of the design we use only integer values to preserve the precision of the parameters (as described in Section 3.1). If the inputs of the design do not contain any integer part, we scale it.

Next we perform a compiler pass over the program to assign bit values to each variable in the program. We propose the following set of rules while assigning bit-length to any variable.

Addition rule: a = b + c;

$Bitlength_a = maximum(Bitlength_b, Bitlength_c) + 1$

Multiplication rule : a = b * c;

$Bitlength_a = (Bitlength_b + Bitlength_c)$

Loop rule : For an accumulator variable within a loop which iterates N (a fixed number known at compile time) times,

for (i = 0; i < N; i++)

accumulator +=sample;

$Bitlength_{accumulator} = (Bitlength_{sample} + Log_2 N)$

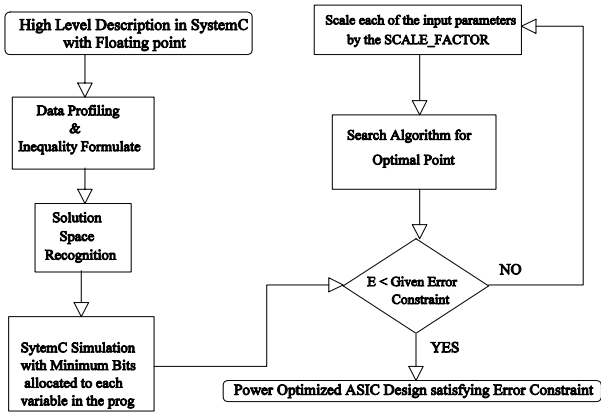

Figure 1. Flow Diagram of the optimization algorithm.

We illustrate our approach using the example SystemC code for an FIR filter in Figure 2. 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. Then we can form the inequalities:

$sample_{BIT} \geq 7$ (i)

$coeff_{BIT} \geq 8$ (ii)

```
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();
};
```

Figure 2. Example of a SystemC code segment.

We next obtain the following inequalities:

$sample\_tmp_{BIT} = sample_{BIT}$ (iii)

$pro_{BIT} = shift_{BIT} + coeff_{BIT} + 1$ (iv)

$acc_{BIT} = pro_{BIT} + log_2(NUMTAPS)$ (v)

$shift_{BIT} = sample_{BIT}$ (vi)

$result_{BIT} = acc_{BIT}$ (vii)

An analysis of the equations reveals that 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:

$32 \geq result_{BIT}$ (viii)

$32 \geq acc_{BIT}$ (ix)

4.2 Solution Space Recognition

The set of inequality relations obtained 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 ($sample_{BIT}$, $coeff_{BIT}$). We also modify the bit-lengths used by the intermediate variables every time we make a change in the independent variable bit-lengths. By solving the 9 inequalities given in the example we get a range of bit-lengths within which $sample_{BIT}$ and $coeff_{BIT}$ can vary. In effect this defines the solution space of the optimization problem. From the above example (for NUMTAPS = 4) we get the solution space:

$19 \geq sample_{BIT} \geq 7$ (x)

$20 \geq coeff_{BIT} \geq 8$ (xi)

4.3 Search Algorithm for Optimal 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 point.

4.3.1 Greedy Search Algorithm

The first method that we use is a heuristic greedy algorithm followed by refined local search techniques. We model the problem as a search for bit precisions in an N-dimensional search space of independent variables

```
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;
}
```

Figure 3. Pseudo code of the Greedy Search Algorithm.

4.3.2 Sub-optimal point search

We start with the search for a sub-optimal point in the solution space that involves a Greedy Search Stepwise algorithm. In this step we try to reach a near optimal solution with the help of fast SystemC simulation. We narrow down the search for an optimal solution with a given error constraint into a very small solution space. We start with the minimum permissible bit-length allowed for the independent variables and adjust the intermediate variables using the inequality relations.

Assuming the design has n independent variables we allocate extra $\log_2$(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 which affects the Error value most favorably (i.e. made it decrease most) is picked and marked 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 got a point which satisfies the given Error Constraint on the design.

4.3.3 Local Search

We can further optimize the solution by searching the neighbors of the sub-optimal point in the solution space. To perform that, we implement a local search at the sub-optimal point. Here we search for other solution candidates among the neighbors and ultimately we would settle for the solution for which given error-bound is met and power consumption is the minimum.

To do that, we scale down the independent variables along each dimension by a constant FINER_SCALE which is less than the SCALE_FACTOR. We calculate the Error Metric E value and the power consumed by each of these new designs. In the example given in Figure 2 we scale both the variables $sample_{BIT}$ and $coeff_{BIT}$ by 64 to reach the sub-optimal point. Now during local search procedure it is observed that if we save a single bit in the sampleBIT variable, i.e. scale down it by 2 (< SCALE_FACTOR = 4) we can get a solution that has the least power consumption. Hence for the given example, a design with $sample_{BIT}$ scaled by 32 and $coeff_{BIT}$ scaled by 64 gives us the optimized solution.

```
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];
      }
 P = Minimum of P[i];
```

Figure 4. Pseudo code of the Local Search Algorithm.

4.3.4 Smart Search Algorithm

In the greedy search we started looking for the solution from the configuration where all the independent variables have least bit-length. This section discusses another approach which 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 sub-optimal solution exists along the solution space boundary, it won't give an optimized solution. A possible modification can be traversing the outer boundary of the solution space to check if any other optimal solution exists along the boundary. This results a significant increase in complexity. We propose a smart algorithm which 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.

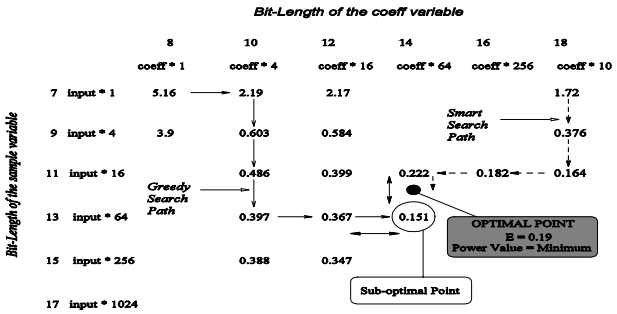

Figure 5. Illustration of the Search Algorithm.

4.4 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 deduced that X has a range of m bits for variance ($x_1 < x_2 < \ldots < x_m$) and Y has a range of n bits ($y_1 < y_2 < \ldots < y_n$). The solution space can be assumed to be a grid of (n x m) cells where each cell corresponds to a particular configuration of the design. The smart algorithm is as follows:

(1) Start searching from either (x_1, y_n) or (x_m, y_1) position. One of the variables will have the highest permissible precision and the other will have 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 lower than previously recorded value, store it and decrease the bit-length of the High Precision variable by one. Go back to step 2.

(4) Otherwise, allocate an extra bit to the Low Precision variable and go back to step 2.

The process will end when the High Precision variable can't be reduced or the Low Precision variable can't be decreased. The optimum configuration is recorded during the traversal.

Proof of optimality

We now prove that our proposed algorithm will always give 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 while searching we started from (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:OPT =BEST(record, BEST$\{(x_k,y_l) \mid i \le k \le m, 1 \le l \le j\}$)

where, OPT = Optimal solution of the problem

record = valid configuration with minimum power consumption checked so far.

BEST(S) = 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 $\{(x_k, y_l) \mid 1 \le k \le m, 1 \le l \le n\}$ is indeed the optimal solution.

At the end of the search we would have i > m or j < 1. Thus $\{(x_k , y_l) \mid i \le k \le m, 1 \le l \le j\} = \{\}$. Hence, record=OPT. The algorithm actually gives the optimal solution.

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.

Case 1 : The algorithm will move along the Y direction, i.e. the High Precision variable. 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 to be an optimal solution. The record will be updated accordingly. Thus after the point moves in Y direction 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 won't change and also the unexplored solution space will reduce. Additionally, we won't exclude any point which could be a possible candidate of the optimal solution. The property P is still true after the increase of i.

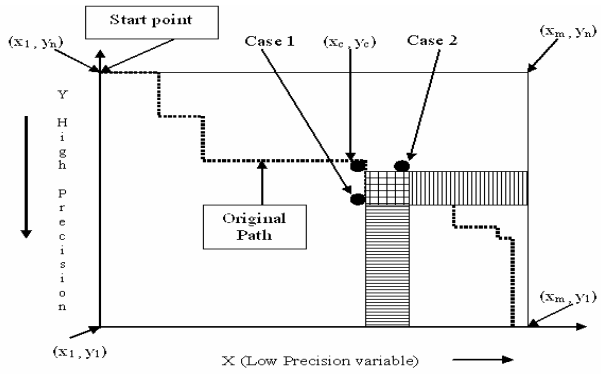

Figure 6. Illustration of the Smart Search Algorithm

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 towards reduction and Low Precision variable towards 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).

4.4.1 Selection of High and Low Precision variable

We use a heuristic to select the High and Low Precision variable for the algorithm. We consider two configurations – Least bit-length and Largest possible bit-lengths for both the independent variables. For both the configurations we change the bit-length along each of the variable and observe the effect on 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 which has a smaller range of bit variance as the High Precision variable.

4.4.2 Extension of the Algorithm to Higher Dimensions

The smart algorithm can be proved 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 a near exponential time complexity. Thus, we propose a heuristic approach for designs having more than two independent variables. Using the same heuristic as discussed in section 4.3.6 we would pick up 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 variable ranked next in the precision sensitivity list. This step would be continued till we fix all the independent variable bit-length.

5 EXPERIMENTAL RESULTS

We now report the experimental results on various benchmark SystemC benchmarks.

- A 16 tap Finite Impulse Response filter (fir)
- An Interpolation FIR filter (intfir)
- A Decimation in Time FIR filter (decfir)
- A LMS adaptive filter (lms)

We 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 sinusoidal wave input of size 2048. Subsequently, we used the Cocentric Behavioral Compiler for SystemC [9] to generate RTL VHDL from the SystemC benchmarks. Using the same tool where Synopsys Design compiler runs in the backend, we synthesized all the designs into 0.18 micron technology ASIC cell library from Artisan Components. This gave us an idea about the area consumed by the optimized design. Finally, we used 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.

Table 1 shows the optimal bit-length of input parameters selected by the smart search algorithm followed for E constraints of 0.5%, 1% and 5% using simulation with sinusoidal inputs. We demonstrate that it is possible to reduce the power consumption by 200% in the best case by allowing round-off 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. However for smaller error constraints (0.5%) it took more simulations to reach the solution point.

6 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 which 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 trade-off the quantization error with the hardware 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 FPGAs.

7 REFERENCES

[1] The Open SystemC™ Initiative (OSCI), www.systemc.org

[2] K. H. Chang, and W. G. Bliss, "Finite word-length effects of pipelined recursive digital filters," IEEE Transactions on Signal Processing, Aug. 1994 Page(s): 1983 –1995

[3] R. M. Gray, D. L. Neuhoff, "Quantization", IEEE Transactions on Information Theory, Volume: 44 Issue: 6, October 1998, pp. 2325 –2383.

[4] H. Keding, M. Willems, M. Coors, and H. Meyr, "FRIDGE: a fixed-point design and simulation environment" In Proc. of Design Automation Test in Europe, 1998, pp. 429 –435, 1998.

[5] Cocentric SystemC Compiler, www.synopsys.com

[6] Cocentric Fixed Point Designer, www.synopsys.com

[7] M. Stephenson and J. Babb and S. Amarasinghe, "Bitwidth Analysis with Application to Silicon Compilation". In Proc. of the SIGPLAN conference on Programming Language Design and Implementation, Vancouver, British Columbia, June 2000.

[8] A. Nayak, M. Haldar, A. Choudhary, P. Banerjee, "Precision And Error Analysis Of MATLAB Applications During Automated Hardware Synthesis for FPGAs," In Proc. of Design Automation and Test in Europe, Mar. 2001, Berlin, Germany.

[9] P. Banerjee, D. Bagchi, M. Haldar, A. Nayak, V. Kim, R. Uribe, "Automatic Conversion of Floating Point MATLAB Programs into Fixed Point FPGA Based Hardware Design," In Proc. of FPGA Based Custom Computing Machines (FCCM), FCCM 2003, Napa Valley, CA

[10] G.A. Constantinides, "Perturbation Analysis for Word-length Optimization," In Proc. of FPGA Based Custom Computing Machines (FCCM), 2003, Napa, CA

[11] M.L. Chang, S. Hauck, "Precis: A Design-Time Precision Analysis Tool," In Proc. of FPGA Based Custom Computing Machines (FCCM), 2002, Napa, CA.

[12] S. Roy and P. Banerjee, "An Algorithm for Converting Floating Point Computations to Fixed Point Computations in MATLAB based Hardware Design," In Proc. of Design Automation Conference (DAC 2004), San Diego, Jun. 2004.

Table 1: 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%)

Error Constraint	Bit-width of the Independent Variables	Power Consumed (104 nW)	Area
FIR16			
E <= 5%	input= 7; coeff = 8	91.1	12454
E <= 1%	input=9; coeff= 9	101	13243
E <= 0.5 %	input=12; coeff=14	376	19844
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
Greedy Search Algorithm gave the optimal solution in 11 out of 12 cases but with extra number of simulations			