Yield-Area Optimizations of Digital Circuits Using Non-dominated Sorting Genetic Algorithm (YOGA)

Vineet Agarwal
Electrical and Computer Engineering
The University of Arizona,Tucson. AZ
vagarwal@ece.arizona.edu

Janet Wang
Electrical and Computer Engineering
The University of Arizona,Tucson. AZ
wml@ece.arizona.edu

ABSTRACT

With shrinking technology, the timing variation of a digital circuit is becoming the most important factor while designing a functionally reliable circuit. Gate sizing has emerged as one of the efficient way to subside the yield deterioration due to manufacturing variations. In the past single-objective optimization techniques have been used to optimize the timing variation whereas on the other hand multi-objective optimization techniques can provide a more promising approach to design the circuit. We propose a new algorithm called YOGA, based on multi-objective optimization technique called Non-dominated Sorting Genetic Algorithm (NSGA). YOGA optimizes a circuit in multi domains and provides the user with *Pareto-optimal set* of solutions which are distributed all over the optimal design spectrum, giving users the flexibility to choose the best fitting solution for their requirements. YOGA overcomes the disadvantages of traditional optimization techniques, while even providing solutions in very stringent bounds.

1. INTRODUCTION

With recent shrinking of transistor feature size, uncertainty in circuit response has become the most menacing issue for the circuit designers. Data statistics indicate that delay variation of a simple gate can go as high as 10%-15% of the expected value. If no compensating techniques are employed to encounter these variations, the yield of nanometer digital circuits can decrease drastically, thereby increasing the manufacturing cost by multiple folds. In recent past the technique of scaling the transistor W/L ratio, also known as *gate sizing*, has emerged as one of the convincing technique for combating these uncertainty due fickle fabrication processes. In this technique, various algorithms and heuristics are used to scale transistor sizes, so as to decrease the timing variations of the circuit and increase the yield. Thus gate sizing presents a promising approach to counter the diabolic affects of manufacturing process uncertainty.

A survey of previous literature reveals that gate sizing techniques have been in lime light for the past decade. The issue of calculating the timing yield and optimizing it has been addressed numerous number of times. Most generally gate sizing problem has been viewed as minimizing a timing variability while having constraints on area or delay [1]. Yield optimizations has also carried out using optimization tools [2] under similar constraints, through modular neural networks [3], geometric programming [4] and also in pre-synthesis steps [5]. Yield issues for sequential circuits has also been addressed in [6]. Song in [7] proposed an approach in which gate sizing is done on statistical critical paths and offers a trade off between area and variation deduction. But most of these work concentrated on trade-off between area overhead and timing variance reduction. Recently Orshansky *el al* proposed an stochastic algorithm for power minimization under variability with a guarantee of power and timing yield [8]. Thus power now played an vital role in designing the circuit, while keeping an constraint on timing yield. Thus the traditional framework for optimization can be summarized as:

$$\begin{array}{rl} \min & f(x) \\ \textit{Subject to} & g(x) \le g_{\max} \\ & h(x) \le h_{\max} \end{array} \tag{1}$$

Doing optimization through techniques just mentioned in Equation (1), which has a single objective function, have various disadvantages. First and foremost, the final solution of single objective optimization process will depend on the constraints placed on other parameter (g_{max}, h_{max}), which are user defined. Thus if these values are not properly chosen then the resulting solution may not be the most appropriate one. For example if we define penalty as the increase in $g(x)$ and $h(x)$, and if the bounds are set very loose, the optimization may minimize $f(x)$ while using all the penalty it is allowed to incur. A second solution may be present with similar objective function value but with considerable less penalty. In those cases the traditional techniques may not deliver the second solutions and thus may not spot the most 'attractive' solution. Secondly, all the optimization technique have been only single-objective function. Thus if more than one objective have to be optimized, the traditional techniques, will optimize the first objective function first and then redo the optimization on this solution with second objective as primary aim. Thus it has to be done in a 'sequential' sense. Multi-objective optimization pose a better approach in which multi objectives are simultaneously optimized. Thus using multi-objective optimization both $f(x)$ and penalties on $g(x)$, $h(x)$ can be optimized (minimized) simultaneously. Thirdly, if the bounds on $g(x)$ and $h(x)$ are set too tight, then there may not exist any feasible solution which satisfies all the constraints. In such cases, the traditional techniques will flounder ending up in giving no solution. Thus optimization techniques are required which can minimize multiple objectives, which are user independent and which don't crumple under too stringent constraints.

Thus we propose our technique, Yield-area Optimization using Genetic Algorithm, YOGA, which encounters the above stated disadvantages. YOGA attempts to minimize all criterion i.e, area, mean delay and delay variance simultaneously. YOGA at the simulation end provides the user with multiple solution spread over the optimum design spectrum. All the solutions in the final set are called Pareto-optimal solutions [9, 10], in which none of the solutions are better than any other in the set, thus any one solution can be chosen without hesitation of optimality. Thereby it gives the designers the flexibility to choose the most appropriate solution according to their needs. YOGA's application can be easily extended to simultaneously optimize power consumption of a circuit. Just by including an extra objective of minimizing power stochastically [8] the designers can get similar trade-off information between power and timing yield and find the best fitting solution or otherwise get trade-off information of a larger picture including area, power and timing yield all together.

Our rest of the paper is organized as follows. Section 2 provides a preliminary introduction to technique of gate sizing. In Section 3, we describe the backbone algorithm of *Non-dominated Sorting Genetic Algorithm*. Section 4 provides a demonstration of the technique while YOGA implementation for gate sizing is discussed in Section 5. The experimental results are listed in Section 6 and Section 7 concludes our paper.

2. BASIC GATE SIZING TECHNIQUE

The traditional framework of gate sizing technique can be summarized as:

$$\begin{array}{rll} \textit{find} & \mathbf{s} & \\ \min & \sigma^2(d_{\mathcal{O}}) & \\ \textit{Subject to} & \mu(d_{\mathcal{O}}) & \leq \mu_{\max} \\ & \pi(\mathcal{G}) & \leq \pi_{\max} \end{array} \quad (2)$$

where

1. $\mu(x)$ is the mean and $\sigma^2(x)$ is the variance of the random variable x.
2. $\mathbf{s} = \{s_1, s_2, \ldots, s_n\}$, where s_i is the size of gate g_i, $\mathcal{G}$ is the set of all gates and $g_i \in \mathcal{G}$. Size s for any gate denotes, the ratio of area of the gate to that of a minimum sized inverter.
3. $\mathcal{O}$ is the set of all the outputs of the circuit.
4. d_i is the delay of the i^{th} output of the circuit and $i \in \mathcal{O}$
5. $\pi(\mathcal{G})$ denotes the area of circuit containing gates $\mathcal{G}$
6. $\pi_{\max}$ and $\mu_{\max}$ are the maximum allowable Area and Mean output delay of the circuit.

Thus a regular gate sizing algorithm will solve the non-linear optimization problem stated in Equation 2 and provide the end user with a single **s**. The constraints place an upper bound on the *penalty* entailed while trying to optimize the objective function, where *penalty* can be in form of area or delay overhead.

3. NON-DOMINATED SORTING GENETIC ALGORITHM

In this section we provide an overview of Non-dominated Sorting Genetic algorithm (NSGA) which will be used as the base for our gate sizing algorithm. Extensive study has been done in multi-objective programming using genetic algorithms in the past [9, 10, 11, 12]. One of the most influential work has been done by Deb in [13], in which he has coined the primary prototype of NSGA.

Algorithm 1 NSGA Algorithm

```
1: $pop \leftarrow$ GenerateInitialPopulation
2: if $generation \leq$ max $generation$ then
3:    $rank \leftarrow$ NonDominatedRanking($pop$)
4:    $fitness \leftarrow$ FitnessAssignment($pop$, $rank$)
5:    for $i = 1$ to N step 2 do
6:       $parent_1 \leftarrow$ Selection($pop$, $fitness$)
7:       $parent_2 \leftarrow$ Selection($pop$, $fitness$)
8:       ($child_1$, $child_2$) $\leftarrow$ Crossover($parent_1$, $parent_2$)
9:       $newpop_i \leftarrow$ Mutation($child_1$)
10:      $newpop_{i+1} \leftarrow$ Mutation($child_2$)
11:   end for
12:   $pop \leftarrow newpop$
13:   $generation \leftarrow generation + 1$
14: end if
15: $final\_rank \leftarrow$ NonDominatedRanking($pop$)
16: NonDominatedSolutions $\leftarrow pop_i, \forall i \in final\_rank_i = 1$
17: return NonDominatedSolutions
```

3.1 Pareto-optimal Solution

A solution is called Pareto-optimal solution, if there exists no other solution for which at least one of its criterion has a better value while values of remaining criteria are the same or better. In other words, one can not improve any criterion without deteriorating a value of at least one other criterion. Thus a design vector x^* is Pareto-optimum if and only if, for an minimization problem and for any x and i,

$$\begin{array}{rl} f_j(x) \leq f_j(x^*) & \forall j = 1, \ldots, m, j \neq i \\ \Rightarrow f_i(x) \geq f_i(x^*) & \end{array}$$

where $f_i(x)$ is the value of i^{th} objective function.

NSGA gives rise to a set of such *Pareto-optimal* solutions. In a multi-optimization problem, the designer may be interested in more than one of such solutions because of the equal *quality* of the solutions. The concept of NSGA is discussed in next section.

3.2 NSGA Algorithm

The algorithmic flow of NSGA is summarized in Algorithm 1. A Model multi-objective optimization problem can be summarized as.

$$\begin{array}{lll} & \min f_i(\mathbf{x}) & \forall i \in \{1, M\} \\ \text{Subject to :} & x_j^L \leq \mathbf{x}_j \leq x_j^U & \forall j \in \{1, P\} \end{array} \quad (3)$$

where f_i is the i^{th} objective function for $i = 1, \ldots, M$ and $\mathbf{x} = \{x_1, \ldots, x_P\}$ where x_j is the j^{th} design variable and $x_j \in [x_j^L, x_j^U]$. Henceforth all optimization will pertain to a minimization problem, unless otherwise stated. The various steps in Algorithm 1 are explained below.

3.2.1 GenerateInitialPopulation *Operator*

This process generates *pop*, an array of size $N \times P$, which contains N (population size) members, each of which is a vector for length P. This population is created by generating uniformly distributed random numbers bounded by $[x_j^L, x_j^U]$ for each x_j.

3.2.2 NonDominatedRanking *Operator*

Definition 1. *Say operator '$\lhd$' denotes worse and '$\rhd$' denotes better solution i.e, for a minimization problem $a \lhd b$ means $a > b$ while for a maximization problem $a \lhd b$ means $a < b$ and vice-versa for '$\rhd$' operator. Thus a solution $\mathbf{x}^a$ is said to dominate $\mathbf{x}^b$ if both of following condition is true:*

Algorithm 2 Non-Dominated Ranking Algorithm

```
$\chi \leftarrow 1$ {comment: *pop* is the current population and $\{x^a, x^b\} \in pop\}$
**repeat**
  $N \leftarrow length(pop)$
  **for** $a = 1$ to $N$ **do**
    **if** $\exists\ b$ such that $x^b \triangleright x^a$ **then**
      push($x^a$, dominated)
    **end if**
  **end for**
  **for** $a = 1$ to $N$ **do**
    **if** $x^a \notin$ dominated **then**
      $rank_{x^a} \leftarrow \chi$
      remove($x^a$, $pop$)
    **end if**
  **end for**
  $\chi \leftarrow \chi + 1$
  *clear* dominated
**until** $pop = \{\emptyset\}$
**return** NonDominatedRanking
```

1. *$\mathbf{x}^a$ is no worse than $\mathbf{x}^b$ for any of the objective functions. In other words, $f_i(\mathbf{x}^a) \ntriangleleft f_i(\mathbf{x}^b)$ for all $j = 1, \ldots, M$.*
2. *The solution $\mathbf{x}^a$ is strictly better than $\mathbf{x}^b$ in at least one objective. In other words, $\exists j$ for which $f_j(\mathbf{x}^a) \triangleright f_j(\mathbf{x}^b)$.*

Now for a set of N solution vectors, the non-dominated ranking can be done according to Algorithm 2.

3.2.3 `FitnessAssignment` *Operator*

After the population of size N has been ranked, fitness values are assigned each of the solution. At first all the solutions in a particular front (having same rank) are assigned same fitness value (f_k) and then those fitness value is *shared* with other solutions in the same front. The sharing procedure for a solution x^a in the k^{th} front is performed as follows:

i. Compute Euclidean distance measure with another solution x^b in the k^{th} front as:

$$d_{ab} = \sqrt{\sum_{p=1}^{P} \left(\frac{x_p^a - x_p^b}{x_p^U - x_p^L} \right)^2}$$

ii. The Sharing function value is then computed as [12]:

$$S(d_{ab}) = \begin{cases} 1 - \left(\frac{d_{ab}}{\xi}\right)^2, \text{if} & d_{ab} \le \xi \\ 0 & \text{otherwise} \end{cases}$$

where $\xi \approx \frac{0.5}{\sqrt[P]{10}}$

iii. The niche count is then computed as:

$$n_a = \sum_{b=1}^{p_k} S(d_{ab})$$

where p_k is the number of solutions in the k^{th} front.

iv. Then the new fitness value is computed as:

$$f_i' = \frac{f_k}{n_a}$$

This procedure is repeated for all the fronts with $f_{k+1} = \min(f_k) - \kappa$ where κ is a small positive value.

3.2.4 `Selection` *Operator*

After fitness has been assigned to all the solutions of the current population, one solution is randomly picked up from the pool. The probability of selection of a solution x^a with fitness value f_a is given by [12]:

$$P(x^a) = \frac{f_a}{\sum\limits_{j=1}^{N} f_j}$$

3.2.5 `Crossover` *Operator*

Crossover is the fundamental mechanism of genetic rearrangement for both real organisms and genetic algorithms. After two parents are selected by the Selection operator, these two parents are crossed over with a probability of 0.9 to give rise to two new children. The Crossover operation can be summarized as:

i. Calculate β_q as:

$$\beta_q = \begin{cases} (u\alpha)^{\frac{1}{\eta_c+1}}, & \text{if} u \le \frac{1}{\alpha} \\ (\frac{1}{2-u\alpha})^{\frac{1}{\eta_c+1}} & \text{otherwise} \end{cases}$$

where $\eta_c = 30$, u is a random number between 0 and 1 and $\alpha = 2 - \beta^{-(\eta_c+1)}$ where β is calculated as follows:

$$\beta = 1 + \frac{2}{x_i^b - x_i^a} \min[(x_i^a - x_i^L), (x_i^U - x_i^b)]$$

ii. The children are then calculated as follows:

$$y_i^a = 0.5[(x_i^a + x_i^b) - \beta_q \left| x_i^b - x_i^a \right|]$$
$$y_i^b = 0.5[(x_i^a + x_i^b) + \beta_q \left| x_i^b - x_i^a \right|]$$

3.2.6 `Mutation` *Operator*

Mutations modify a small fraction of the solution variables: roughly one in every 10,000. Mutation alone does not generally advance the search for a solution, but it does provide insurance against the development of a uniform population incapable of further evolution. After the two children are generated by the Crossover operator, they are mutated by using the polynomial probability distribution [13]. The following steps are used on each y_i for $i \in \{1, P\}$ with a probability p_m:

i. Parameter δ_q is calculated as:

$$\delta_q = \begin{cases} [2u + (1-2u)(1-\delta)^{\eta_m+1}]^{\frac{1}{\eta_m+1}} - 1 \\ \quad \text{if } u \le 0.5 \\ 1 - [2(1-u) + 2(u-0.5)(1-\delta)^{\eta_m+1}]^{\frac{1}{\eta_m+1}} \\ \quad \text{otherswise} \end{cases}$$

where u is a random number between 0 and 1, $\delta = \min[(y_i^a - y_i^L), (y_i^U - y_i^a)]$, $\eta_m = 100+$iteration number.

ii. The mutated child portion is calculated as follows:

$$z_i^a = y_i^a + \delta_q (y_i^U - y_i^L)$$

The mutation probability p_m is linearly varied from $\frac{1}{P}$ till 1.0.

For minimization problems in `NSGA`, the objective function of the form $f_j(\overrightarrow{x}) \le t_j$ is modified to $\min\langle f_j(\overrightarrow{x}) - t_j \rangle$ where the operator $\langle x \rangle$ returns x if x is positive or 0 otherwise. A much detailed explanation for these operators can be found in [10]

4. NSGA DEMONSTRATION

We shall now demonstrate the application of NSGA on a sample non-linear multi objective optimization problem. The sample problem is formulated as:

$$\begin{array}{rl} goal & f_1 = 10x_1 \leq 2 \equiv \min \langle f_1 - 2 \rangle \\ goal & f_2 = \frac{10+(x_2-5)^2}{10x_1} \leq 2 \equiv \min \langle f_2 - 2 \rangle \\ Subject\ to\ \mathbb{F} \equiv & (0.1 \leq x_1 \leq 1, 0 \leq x_2 \leq 10) \end{array} \quad (4)$$

Figure 1 shows a values of f_1 and f_2 for randomly chosen values of x_1 and x_2. It is noticed that there exists no feasible solution to this problem, since no values of (x_1, x_2) can make both the objective function meet their goal. In this situation

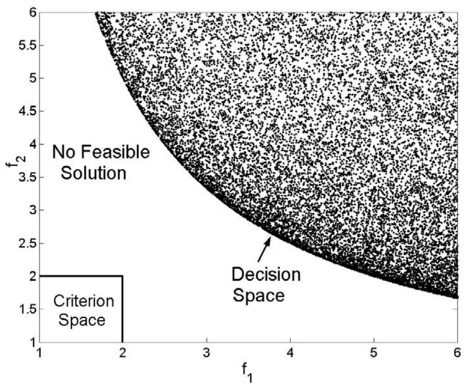

Figure 1: Non-overlapping Criterion Space and Decision space of the problem stated according to Equation 4

the traditional optimization technique will cease to give any fruitful results, and on the other hand the solutions provided by NSGA are plotted in Figure 2. Thus NSGA provides solutions which are concentrated in a certain neighborhood and tries to bring both the objective functions near their goals and also the tries to keep the penalty of the solutions to a minimum. Thus instead of providing no-feasible-solution, it provides the solutions from the decision space which are closest to the required non-feasible objective function point. Thus the advantages of NSGA over the traditional techniques

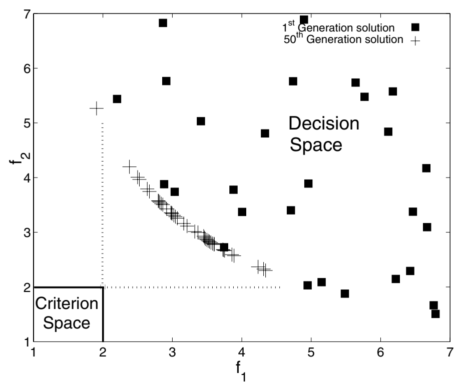

Figure 2: NSGA solution to the Equation 4

are evident from this example. YOGA can thus use NSGA in order to do circuit optimization in the similar way, which is explained in further details in next section.

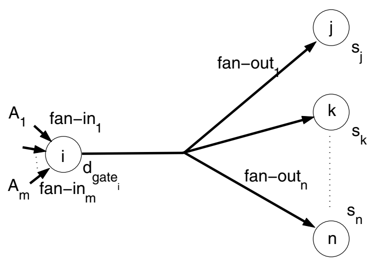

Figure 3: Depth based Nodal delay model for STA

5. YOGA IMPLEMENTATION

NSGA in [13] concentrated on convergence of such Pareto-optimal set, while YOGA maintains wide diversity in the solution set, so as to provide the designer with ample different solutions distributed over design spectrum. Thus for using NSGA in YOGA, the problem is formulated as in Equation 5.

$$\begin{array}{rl} find & \mathbf{s} \\ Minimize & \max(\overrightarrow{\sigma}_{\mathcal{O}} / \overrightarrow{\mu}_{\mathcal{O}}) \\ Minimize & \pi(\mathcal{G}) \\ Subject\ to & s_j^L \leq s_j \leq s_j^U, \quad \forall j \in [1, N], g_j \in \mathcal{G} \end{array} \quad (5)$$

where $\mathcal{O}$ is the set of all the outputs of the circuit, $\mathcal{G}$ is the set of all the N gates in the circuit, $\overrightarrow{\mu}_{\mathcal{O}}$ is a vector of mean delay values of all the outputs in $\mathcal{O}$, $\overrightarrow{\sigma}_{\mathcal{O}}$ is a vector of delay deviation of all the outputs in $\mathcal{O}$ and g_i is the i^{th} gate with a sizing factor of s_i.

Thus the solution vector of YOGA will be set of vectors each of length N, where each element of each solution will correspond to the sizing factor of the respective gates.

5.1 Gate Delay model

In the past, for the purpose of Statistical Timing Analysis (STA) of digital circuit, the delay was modelled as a random variable to incorporate the manufacturing uncertainty. Song in [7] modelled the variation in delay as sum of a factor directly proportional to delay and another random factor whereas Raj in [2] assumed variation to be 15% of the expected value. In our work we model the delay as:

$$d_i = \max(A_{i1}, \ldots, A_{in}) + d_{gate_i} + d_{load_i} \quad (6)$$

where d_i is the delay at wire i, $(A_{i1}, \ldots, A_{in})$ are the arrival times of fan-ins of node i, d_{gate_i} is the intrinsic delay of $gate_i$ and d_{load_i} is the delay incurred due to fan-out loads of node i. The situation is depicted in Figure 3. Here the scaling factors $(s_1, \ldots, s_N)$ are modelled as random variables. Intuitively it can be comprehended that a smaller the sizing factor, larger is the uncertainty in manufacturing. Thus the factor (σ/μ) decreases with increasing size (where σ_s and μ_s are the variance and mean of sizing factor respectively). With this kind of delay model and delay characterization (Section 5.2.1), delay at node i can be computed fairly accurately by considering effects of arrival time, intrinsic delay, the loading effect.

5.2 Statistical Timing Analysis (STA)

Block based and event propagation techniques [14, 15] are efficient ways for doing the STA for the circuit. Our methodology is based on techniques mentioned in [15]. At first arrival times are assigned to all the primary inputs of the circuits depending on the fan-out load for each of the inputs. Then the whole circuit is divided into different depths.

The gates which receive their inputs from the primary inputs of the circuit are assigned a depth 1. If $\mathcal{F}_i$ is the set of fan-ins of the gate g_i, and $\mathcal{D}_i$ is the depths of members of $\mathcal{F}_i$, then the depth of a gate g_i, is given by $\max(\mathcal{D}_i) + 1$. A gate is not assigned its depth unless and until all the elements in $\mathcal{F}_i$ have been assigned their corresponding depths. After the depth assignment is completed, the mean and variance values of delay are computed for all the gates having depth 1. Then the delay values are propagated from depth 1 to depth 2 and so on from depth k to depth $k+1$. The delay at the output of any gate g_i is given by Equation 6. The $\max(A_{i1}, \dots, A_{in})$ operation is performed with the use of formulation for $z = \max(x, y)$ [16] according to Equation 7.

$$\begin{aligned} a^2 &= \sigma_x^2 + \sigma_y^2 - 2\sigma_x\sigma_y\rho \\ \alpha &= (\mu_x - \mu_y)/a \\ \nu_1 &= \mu_x\phi(\alpha) + \mu_y\phi(-\alpha) + a\ \varphi(\alpha) \\ \nu_2 &= (\mu_x^2 + \sigma_x^2)\phi(\alpha) + (\mu_y^2 + \sigma_y^2)\phi(-\alpha) + (\mu_x + \mu_y)\ a\ \varphi(\alpha) \end{aligned} \tag{7}$$

where $x \sim N(\mu_x, \sigma_x)$ and $y \sim N(\mu_y, \sigma_y)$ and $\rho = r(x, y)$ where $r()$ is the correlation operator. Thus we can get $\mu_z = \nu_1$ and $\sigma_z = \sqrt{\nu_2 - \nu_1^2}$.

Thus the delay values are propagated from depth 1 till the last front which are the primary outputs of the circuit. After the mean and variance values are computed at the primary output, the factor $\max(\sigma_1/\mu_1, \dots, \sigma_L/\mu_L)$, where L is the number of outputs of the circuit, is computed and returned to YOGA as the value of one of it objective function. Though this is not a very accurate technique of doing STA, but since our main concentration in this work is the effectiveness of YOGA, we can compromise with a less complicated model for doing STA.

The area of the whole circuit is simply calculated as :

$$Area(\mathcal{G}) = \sum_{j=1}^{N} \alpha_j s_j$$

where N is the number of gates in the circuit. s_j is the size of gate i which has a weighting factor of α_j. The value of α_j for any particular gate depends on the number of its input and the type of the gate. For example a 2-input XOR gate has an α value of 24 units while a 3-input NAND gate has α value of 15 units. The weighting factor α depicts the size of a minimum sized gate of that type.

5.2.1 *Delay Characterization*

For shorter run time while doing the STA of any circuit, delay characterization is done before hand. In this technique, the effect on delay at node i due to variations in s_i of gate g_i and size of fan-outs can be captured simultaneously. Initially a sample circuit is constructed consisting a chain of 2 inverters which have size as s_D, s_R respectively. Then the sample circuit is simulated using SPICE for some set of random values pairs of s_D and s_R and the delay of 1^{st} inverter is computed for each value pair. From Elmore delay analysis we conclude that the delay is of the form:

$$d = \kappa_1 + \kappa_2\left(\frac{s_R}{s_D}\right) + \kappa_3\left(\frac{s_R}{s_D}\right)^2. \tag{8}$$

Any curve fitting technique, like Response Surface Method (RSM) or Singular Value decomposition (SVD) can be used to estimate the three coefficients of Equation 8. This delay model can be used for STA. If 50 different values are used for characterization, then the error in estimated delay is around 0.0001% of the actual value, which is fairly acceptable.

6. EXPERIMENTAL RESULTS

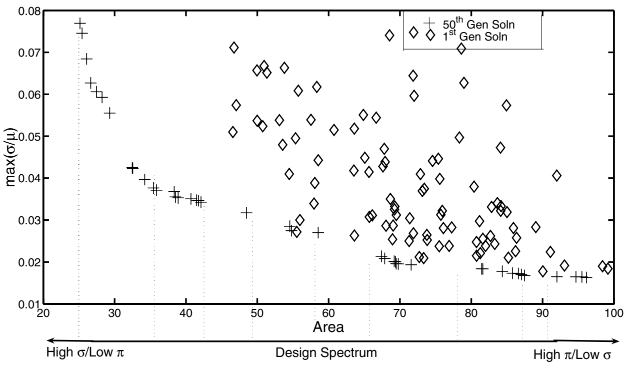

Figure 4: Results of YOGA on ISCAS C17 Benchmark

This section presents the results of application of YOGA algorithm on the variety of benchmark circuits of ISCAS family [17] and few circuits for which the types of gate in it were chosen arbitrarily. The STA was done using Berkeley PTM [18] $70nm$ model cards for NMOS and PMOS transistors. The results of YOGA on c17 circuit is shown in Figure 4. It can be inferred from the figure that given a random set of gate sizes, YOGA delivers a variety of solutions at the designers disposal. All the solutions marked with '+' in figure 4 consist the *Pareto-optimal set* in which no one solution is better than any other. Few of the Pareto-optimal solutions for other benchmarks circuits are listed in Table 1.

If traditional single objective function techniques were used to solve this optimization problem, in a way to minimize $\max(\sigma/\mu)$ and to keep total area under 120 units, then the solution provided will be a single set of gate sizes which will lie on the right extreme section of design spectrum of Figure 4. But it can be noticed from the figure, that other solutions exist which can minimize objective function to the same amount but can still account for lesser area of the circuit. Single objective function techniques fail to see these solutions and thus may not provide solutions which are optimum in multi-sense. On the other hand, YOGA gives a wide range of solutions and designers can pick any of the pareto-optimal solution according to their need. For example low area solutions can be selected from left section of design spectrum, while least variation solutions can be picked from right section. The pareto-optimal solutions provide the users with visible trade-offs for choosing their solutions. If they don't wish to loose much on both end of area and delay variability, then they can choose solutions from middle sections of spectrum. This flexibility is be provided by traditional techniques.

For the case of c17 benchmark the designer has varied solutions to pick any one of them according to their requirements. For example if the area is of more concern then he can choose solutions such as (area = 25.1485, $\max(\sigma/\mu)$ = 0.0769) or (area = 32.4708, $\max(\sigma/\mu)$ = 0.0425), where he gets low area solution but has to compensate by getting a high variance circuit along with. But on the other hand of timing variance is of more importance then he can choose solution from the other end of the spectrum, such as (area = 71.5729, $\max(\sigma/\mu)$ = 0.0193) or (area =96.1451, $\max(\sigma/\mu)$ = 0.0163) where he has to compromise on area to get a low variance circuit. Whereas if he needs to keep both in objective in control then he can choose the solution which are midway in the design spectrum such as (area=54.8, $\max(\sigma/\mu) = 0.0275$).

Similarly, solutions from different portion of design spectrum for any given circuit can be provided to suit the designers needs and requirements. The only input which is required is circuit topology and the minimum and maximum allowable gate sizes.

7. CONCLUSION

In this paper, we have shown the advantages of Multi-objective optimization techniques over the traditional single objective techniques. A new algorithm YOGA was proposed to use multi-objective optimization technique for gate sizing domain of digital circuits and flexibility obtained by it was demonstrated for various benchmark circuits.

Table 1: Few solutions of *Pareto-optimal set* obtained by YOGA for ISCAS benchmark circuits

Circuit Name	No. of gates	Area $= \sum_{j=1}^{N} \alpha_j s_j$	$\max(\sigma/\mu)$	Run time(s)
c17	6	25.1485	0.0769	4.42
		32.4708	0.0425	
		54.7947	0.0275	
		71.5729	0.0193	
		96.1451	0.0163	
c432	160	769.5831	0.0538	43.43
		1054.2467	0.0317	
		2131.5073	0.0169	
		3028.1891	0.0142	
		3798.7005	0.0121	
c499	202	1418.9510	0.0231	51.37
		2025.2311	0.0189	
		3869.6068	0.0145	
		4932.4860	0.0135	
		6420.8214	0.0126	
c880	383	1818.4447	0.0618	94.25
		3102.6154	0.0432	
		4756.7919	0.0281	
		6281.9218	0.0228	
		7554.7382	0.0205	
c1355	546	2681.1763	0.0117	137.18
		3817.4057	0.0085	
		3965.4355	0.0068	
		8815.9168	0.0057	
		10672.4103	0.0042	
c1908	880	3218.8936	0.0247	227.20
		5476.0578	0.0202	
		8981.3529	0.0163	
		12692.8132	0.0148	
		15834.0099	0.0131	
c270	1193	6973.1389	0.0687	341.68
		9281.2028	0.0534	
		16315.1987	0.0323	
		22180.6038	0.0241	
		24872.2772	0.0213	
c3540	1669	7281.1988	0.0357	470.75
		14672.0153	0.0248	
		21915.7468	0.0162	
		29180.4415	0.0151	
		34971.9318	0.0138	
circuit$_1$	20	172.4660	0.0631	7.67
		275.4186	0.0421	
		438.8642	0.0284	
		569.5252	0.0231	
		661.2028	0.0208	
circuit$_2$	30	252.6721	0.0648	10.20
		628.8381	0.0408	
		912.0196	0.0256	
		1181.6013	0.0237	
		1263.3795	0.0228	

8. REFERENCES

[1] S. H. Choi, B. C. Paul, and K. Roy, "Novel sizing algorithm for yield improvement under process variation in nanometer technology," in *DAC*, 2004.

[2] S. Raj, S. Vrudhula, and J. Wang, "A methodology to improve timing yield in the presence of process variations," in *DAC*, 2004, pp. 448–453.

[3] A.A.Ilumoka, "Optimal transistor sizing for cmos vlsi circuits using modular artificial neural networks," in *Twenty-Ninth Southeastern Symposium on System Theory*, 1997.

[4] J. Singh, V. Nookala, Z.-Q. Luo, and S. Sapatnekar, "Robust gate sizing by geometric programming," in *DAC*, 2005.

[5] A. Nardi and A. L. Sangiovanni-Vincentelli, "Synthesis for manufacturability- a sanity check," in *Design Automation and Test in Europe*, 2004.

[6] M. Pan, C. C. N. Chu, and H. Zhou, "Timing yield estimation using statistical static timing analysis," in *Intl Symposium on Circuits and Systems*, 2005.

[7] O. Neiroukh and X. Song, "Improving the process-variation tolerance of digital circuits using gate sizing and statistical techniques," in *Design Automation and Test in Europe*, 2005.

[8] M. Mani, A. Devgan, and M. Orshansky, "An efficient algorithm for statistical minimization of total power under timing yield constraints," in *DAC*, 2005.

[9] C. M. Foncesa and P. J. Flemming, "Genetic algorithms for multi-objective optimization: Formulaton, discussion and generalization," in *5th Intl Conf on Genetic Algorithms*, 1993, pp. 416–423.

[10] N. Srinivas and K. Deb, "Multi-objective function optimization using non-dominated sorting genetic algorithms," in *Evolutionary Computation*, 1994.

[11] A. G. Cunha, P. Oliveira, and J. A. Covas, "Use of genetic algorithms in multicriteria optimization to solve industrial problems," in *Proceedings of the Seventh International Conference on Genetic Algorithms*, 1997, pp. 682–688.

[12] D. E. Goldberg, *Genetic algorithms for search, optimization, and machine learning.* MA: Addison-Wesley., 1989.

[13] K. Deb, "Non-linear goal programming using multi-objective genetic algorithms," *Evolutionary Computation Journal*, 1994.

[14] A. Devgan and C. Kashyap, "Block-based static timing analysis with uncertainty," in *ICCAD*, 2003.

[15] J. Liou, K. Cheng, S. Kundu, and A. Krstic, "Fast statistical timing analysis by probabilistic event propagation," in *DAC*, 2001, pp. 661–666.

[16] C. Clark, "The greatest of a finite set of random variables," *Operations Research*, vol. 9, no. 2, 1961.

[17] M. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the iscas-85 benchmarks: A case study in reverse engineering," *IEEE Design and Test*, 1999.

[18] "Berkeley ptm." [Online]. Available: http://www-device.eecs.berkeley.edu/~ptm/