

GENETIC ALGORITHMS - OPTIMISATION OF LEAKAGE POWER AND DELAYS

Path : home/itvlsi26/ruc/

**Algorithm based transistor sizing of nanoscale digital circuits
(Determining values of W,L for which leakages and delays are
minimum)**

FACULTY : ZIA ABBAS

MENTOR : PRATEEK GUPTA

SATHYA SRAVYA.V (20161121)

RUCHITHA .VUCHA(20161139)

UNNIKRISHNAN .R

RADHAKRISHNA .V

Goal

The main goal of this project is to do Multi-objective optimisation of parameters Leakage power and Time delays of digital CMOS circuits.

Milestones

1. We started initially with the Rank based optimisation technique(Multi objective optimisation) of Leakage power and Delays.We implemented it on Inverter,where as there are very good chances of implementing it on different kinds of digital circuits.
2. Later, we started with Multi-objective optimisation using Non-Dominated Sorting Algorithm -II.We were able to decrease the leakage to considerable extent and kept Delay in particular bounds.We implemented it on Inverter.
3. Finally we implemented NSGA-II on Full Adder circuit, were able to optimise delays and leakage powers of 14 NMOS s, 14 PMOS s.
4. Thus we were able to obtain Pareto optimal solution for Leakage and Delay parameters of given circuit.

Specifications

We adopted a process in which , the Length of MOSFETS we use won't decrease below 45nm and the maximum limit is 60 nm.The values of Width of MOSFETS have minimum limit as 90nm,maximu limit as 1200 nm.

Applications and Motivation

These are robust adaptive optimization techniques based on biological paradigm.

- They perform efficient search on parameters along with maintaining ordered pool of them.
- New parameters are produced from existing ones by applying genetic operators like crossover and mutation.
- These searches are not greatly influenced by local optima or discontinuous functions.
- Microcode compaction can be modelled in the same way as these problems , which motivates us to use them in this field.
- Also these algorithms were used in many famous problems such as Travelling salesperson and scheduling job shops.
- These algos are used to obtain a better generation .

First Milestone

Genetic Algorithms

- Genetic algorithm is a stochastic optimisation technique that uses principles derived from evolutionary process in nature.
- In operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimisation and search problems by relying on bio-inspired operators such as mutation, crossover and selection.
- In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions.
- Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

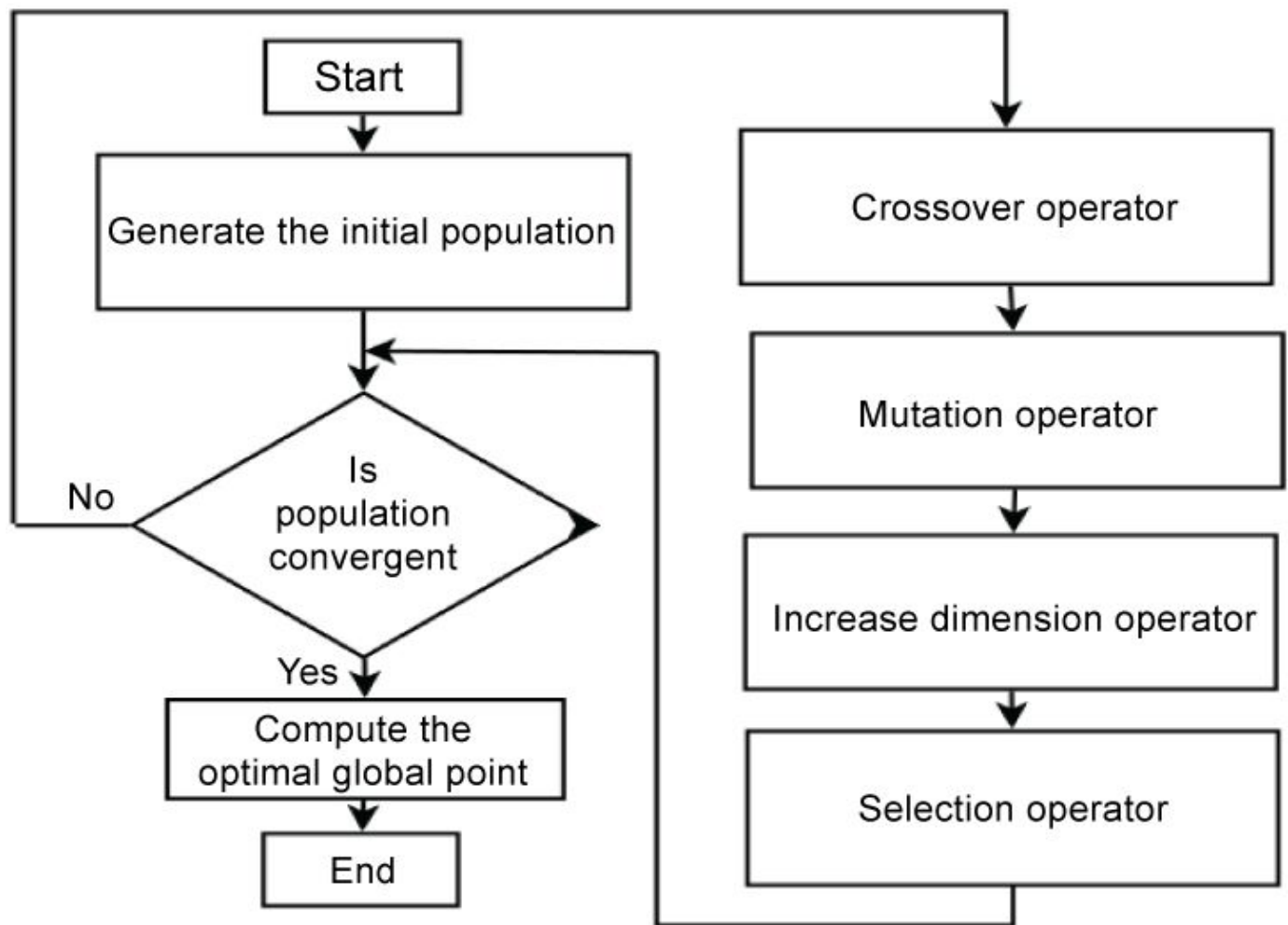


Figure 1: Flow diagram of the improved genetic algorithm.

- Here a Hashing technique is used.

Individual encoding:

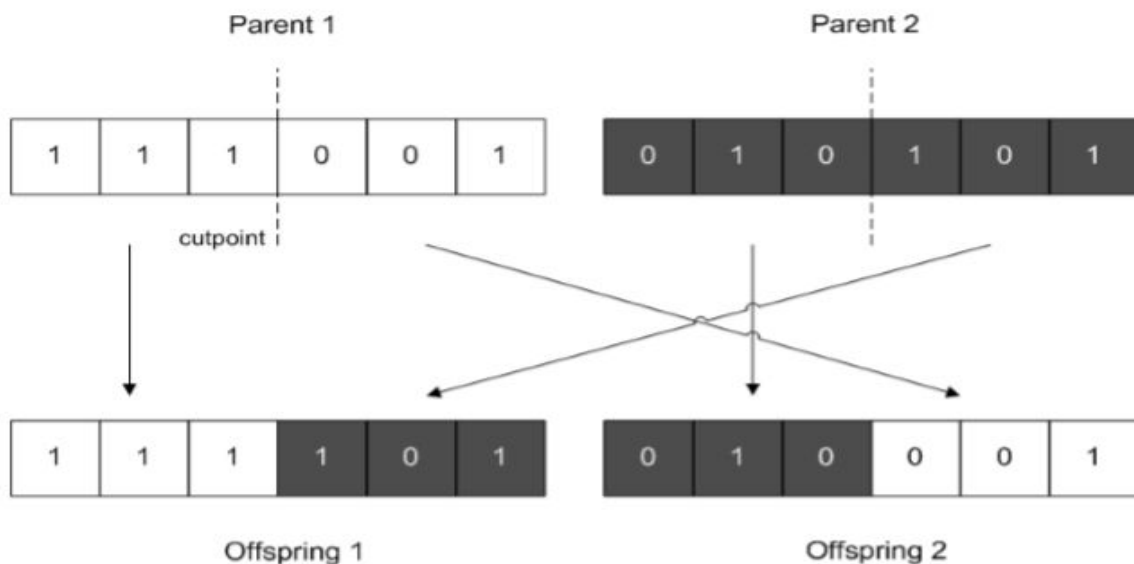
- The population of W,L s we operate with are encoded into

strings, here we use hashing technique to do that.

- The chosen encoding scheme should also be easy to decode using minimal time and memory resources.

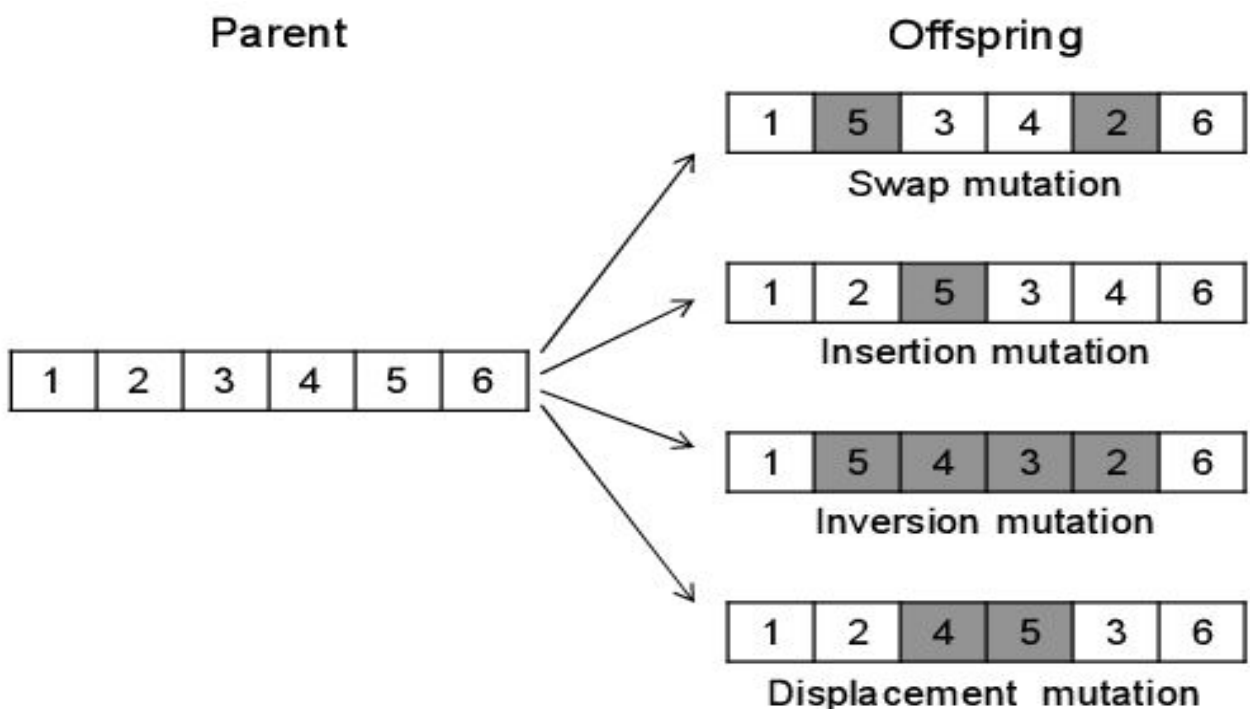
Cross over :

- The crossover operator selects two parent individuals from the population based on a selection scheme. It then produces an offspring individual by using certain information from the first parent and the rest of the information from the second parent.



Mutation :

- This operator introduces random error in individuals we deal with.
- The mutation operator plays a critical role in restoring lost genetic material or providing diversity in the current population. Thus, the mutation operator helps prevent convergence to local optima.



Fitness measure:

- In this part, we run hspice for different individual files, and get leakage and delay files.
- We extract the T_{hl} and T_{lh} from delay files and Leakage power when $V_{in} = 0$ and $V_{in} = 1$ from .ms0 files.

- We take average of both delays and both leakage powers and then assign a fitness measure.
- The fitness measure should reflect the quality of corresponding solution to the problem. It will also decide if the individuals(parameters) survive through generations.
- After applying genetic operations on individuals , the fittest (here the one with leakage power=0 , delay below a particular bound) will be considered for next generations.
- Here we adopted usage of a **profit function** to filter the ones with least leakage power and delays , we output the best 90% of given W,L s of mosfets for the CMOS inverter cell .

Termination:

- Thus we ended up with getting best values of W,L for which Leakage power and Delays are minimum .
- We understand the quality of individuals from their fitness measures .
- At the end we have a generation of individuals with their delays and leakages far less than the first generation .

Algorithm :

```

Genetic_algo()
{
    Initialize the generation;
    Calculate the fitness function for all;
    While (the number of generation > max)
    {
        Selection;(Random or Natural)
        Crossover;
        Mutation;
        Calculate the fitness ;
        Rank them on the basis of fitness by sorting;
    }
}

```

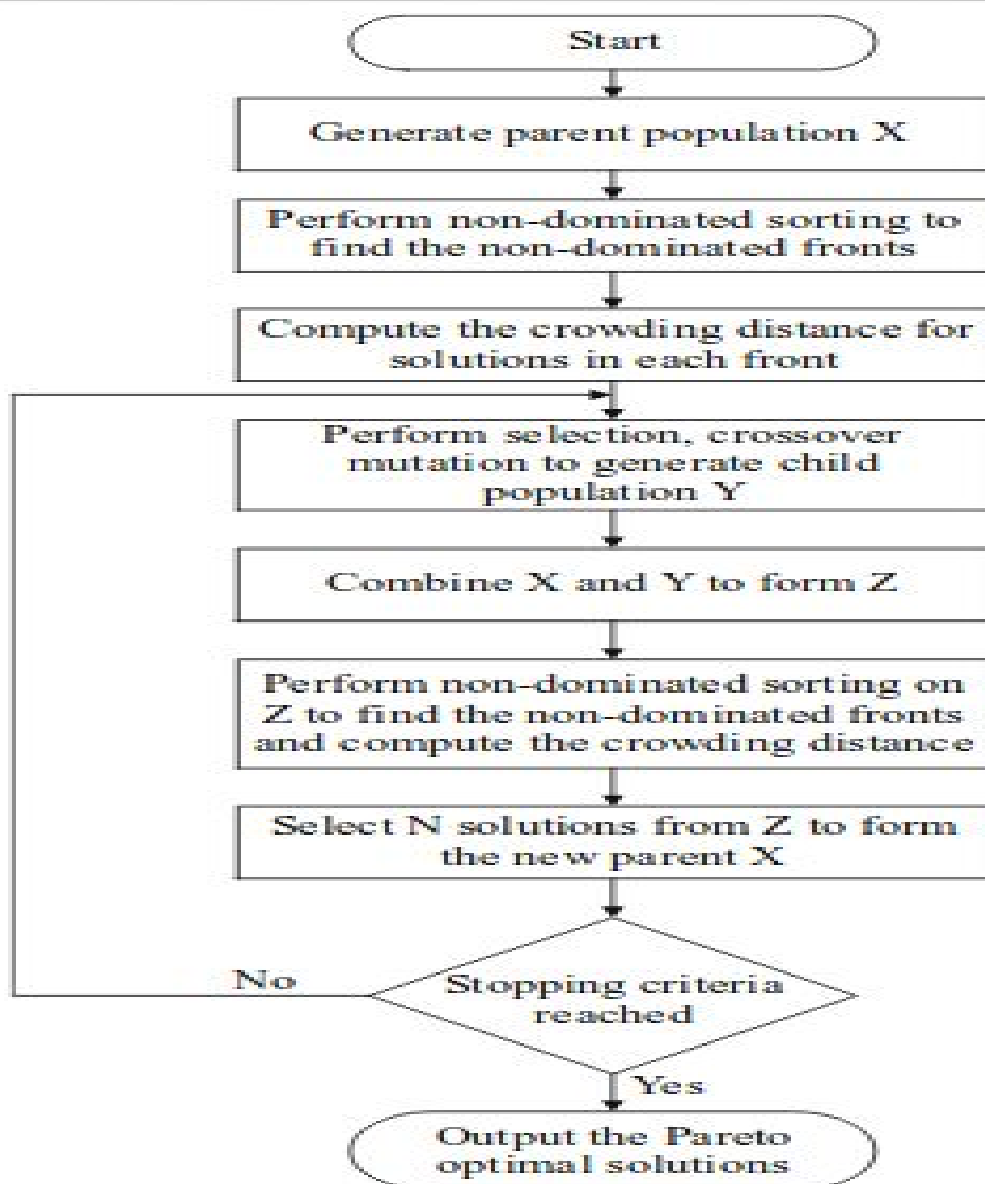
Conclusion :

- Genetic algorithms have been extensively researched and applied to a wide variety of single objective and multi-objective optimization applications.
- Here , the application of genetic algorithms to CMOS inverter cell is investigated.

Second milestone

Moving towards Multi objective optimisation using

NSGA-II ...Elitism is applied in order to preserve the best individuals by taking both, the previous and the current population into account.



NSGA-II employs a fast nondominated sorting approach with m number of objectives and N population to be sorted. It employs a fast nondominated sorting approach with a complexity of $O(mN^2)$ where m is the number of objectives and N is the population to be sorted.

- The first is to classify the population into various non domination fronts using a non dominated sorting procedure. (ranking acc to heuristic fn)
- The second is to estimate the density of solutions surrounding a particular point in the population by means of crowding distance computation.
- Nondominated sorting is used to classify the population to identify the solutions for the next generation
- The individuals are selected based on rank and crowding distance metric, every individual in the population is assigned a crowding distance value.

What happens in NSGA..?

- We will assign ranks to each individual based on how many individuals they dominate wrt each objective.
- Thus fronts of 10 individuals are formed, which will be further sent for crowding distance computations.

Crowding distance computation

- ❑ With respect to each objective crowding distance is computed, to represent how best the solution is wrt that particular parameter.
- ❑ The best solution at the end will be known by considering

Crowded Tournament Selection

- ❑ Within the first front, the one with highest crowding distance wrt leakage objective is taken as the best solution.(after running for 100 generations).

Crowding distance computation algorithm

```

1: procedure CROWDINGDISTANCE( $\mathcal{F}$ )
2:    $N = |\mathcal{F}|$ 
3:   for  $i = 1 \dots N$  do
4:      $\mathcal{F}[i]_{\text{dist}} = 0$ 
5:   end for
6:   for  $m = 1, \dots, M$  do
7:     SORT( $\mathcal{F}, m$ )
8:      $\mathcal{F}[1]_{\text{dist}} = \mathcal{F}[N]_{\text{dist}} = \infty$ 
9:     for  $i = 2 \dots N - 1$  do
10:       $\mathcal{F}[i]_{\text{dist}} = \mathcal{F}[i]_{\text{dist}} + \frac{(\mathcal{F}[i+1].m - \mathcal{F}[i-1].m)}{f_m^{\max} - f_m^{\min}}$ 
11:    end for
12:  end for
13: end procedure

```

Thus by applying crowding distance computation,

Crowding distance calculation is the determination of Euclidean distance between each individual in a front based on their m objectives in the m dimensional space. Since the individuals are selected based on rank and crowding distance metric, every individual in the population is assigned a

crowding distance value. The crowding distance is assigned front-wise. The crowding distance is calculated as follows:

Step 1: For all n individuals in each front F_u , initialize the distance to be zero.

$F_u(cd_i) = 0$, where i corresponds to the i th individual in front F_u .

Step 2: For each objective function m , sort the individuals in front F_u based on objective m :

- $L = \text{sort}(F_u, m)$.
- Assign a large distance to the boundary points in L , $L(cd_1) = L(cd_n) = \infty$
- For the other points $i = 2$ to $(n - 1)$ in L is given by Eq. 1.6:

$$L(cd_i) = L(cd_i) + \frac{L_{i+1}f_m - L_{i-1}f_m}{f_m^{\max} - f_m^{\min}} \quad (1.6)$$

$L_i f_m$ is the value of the m th objective function of the i th individual in L , f_m^{\max} is the maximum value of m th objective, and f_m^{\min} is the minimum value of m th objective.

Crowded Tournament Selection Selection of individuals using crowded tournament selection operator is as given in the following.

A solution x is said to win the tournament with another solution y if any of the following conditions are true:

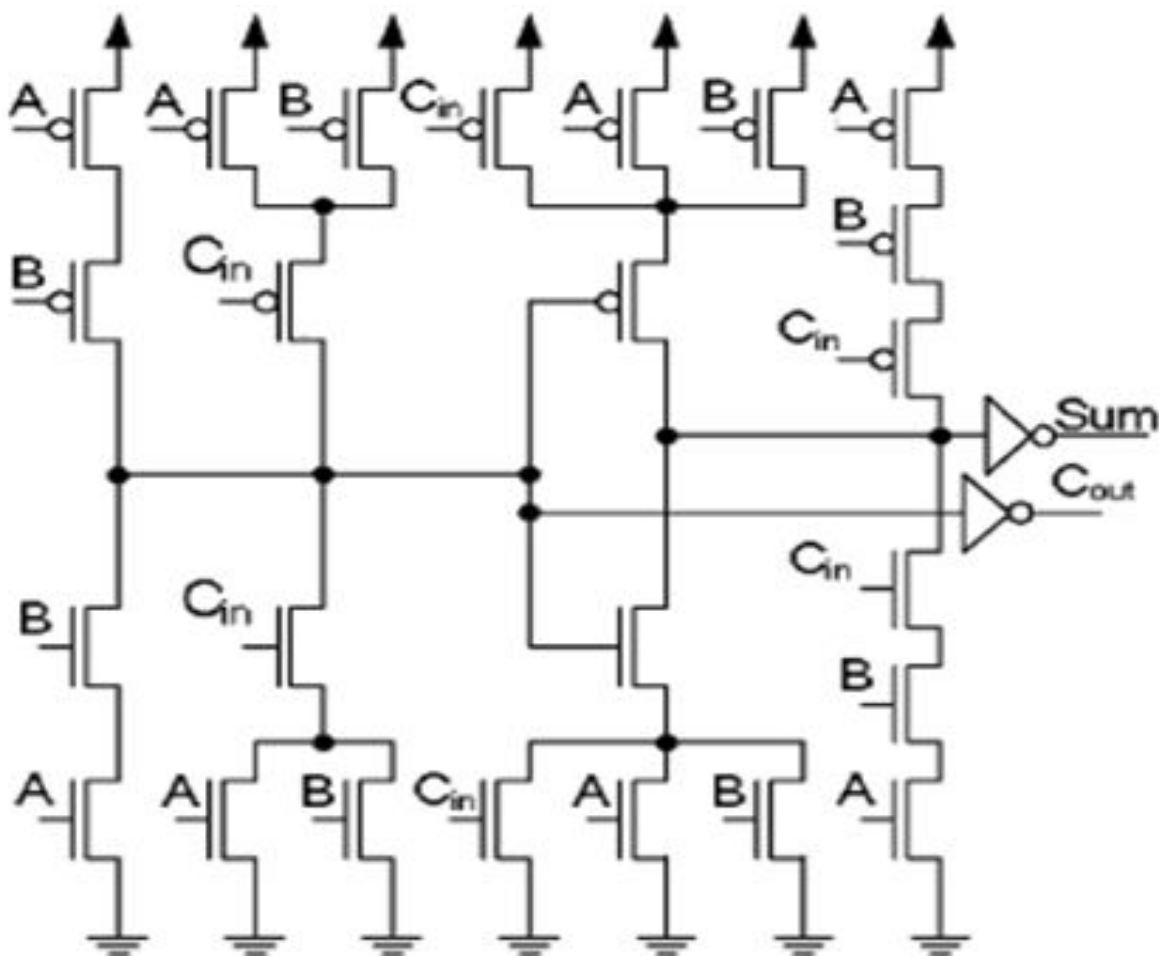
- If solution x has a better rank, that is, $r_x < r_y$
 - If x and y hold the same rank and solution x has a better crowding distance than solution y , that is, $r_x \leq r_y$ and $cd_x > cd_y$
- The first condition makes sure that the chosen solution lies on a better nondominated front. The second condition resolves the tie of both the

solutions being on the same nondominated front by deciding on their crowded distance. The solution residing in the less crowded area wins, that is, it has a larger crowding distance.

Third milestone

Implementation of NSGA-II algorithm on FULL ADDER

Full adder schematic



Inputs: sizes of generative Full Adder transistors w,l values of 14 pmos & 14 nmos.

Outputs: average Leakage power and max delay optimisation.

Objective: To reduce the average leakage power by 50% while keeping the maximum delay in bound.(10% of initial value)

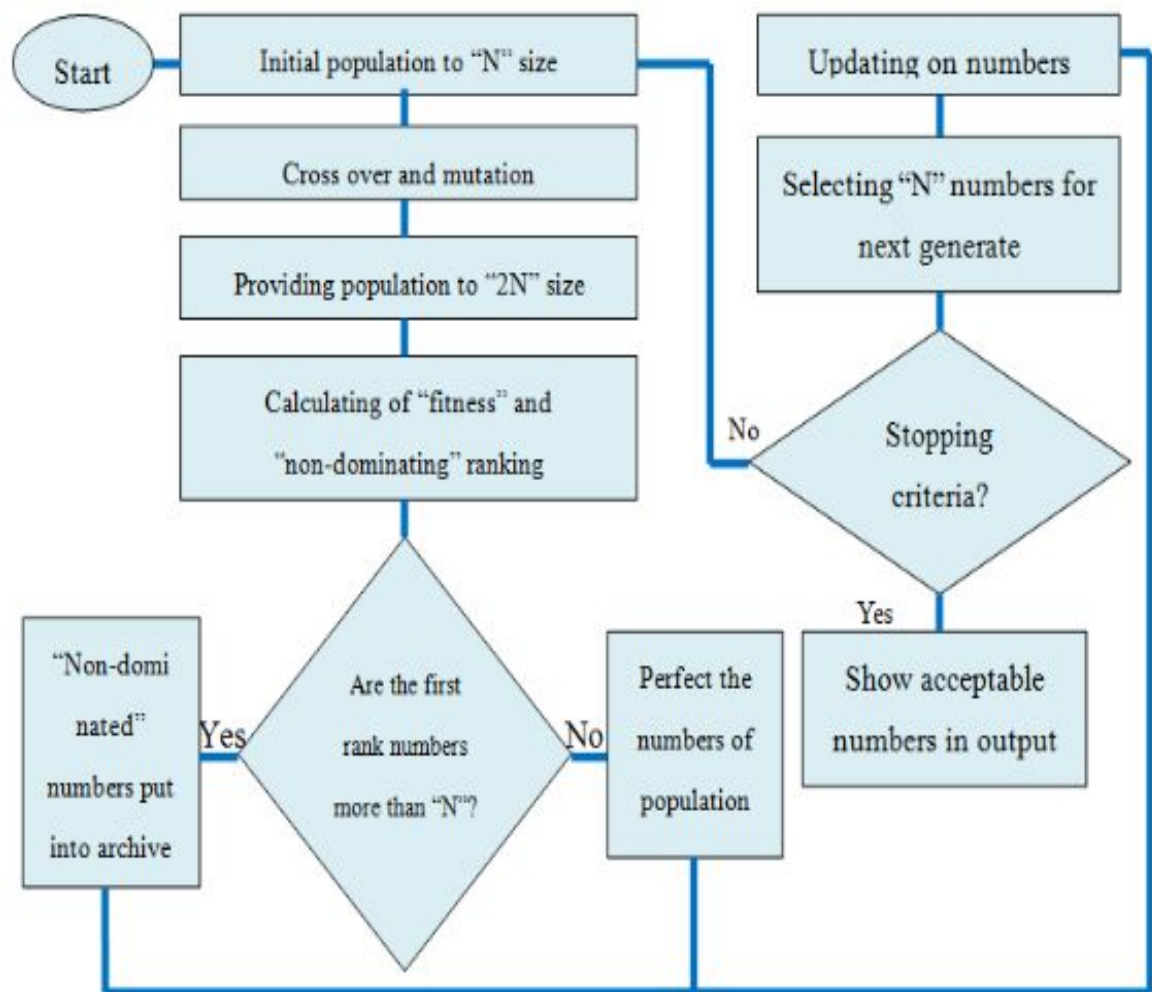
Method :Full Adder is made of two separate circuits each part is optimized.The number of population for both circuits are 100 and algorithm will continue to 20 generations.Plan of implementation , We follow a predefined flow,

Initial population Generation ->Non-domination sorting -> Binary tournament selection -> Recombination -> Mutation -> -> Pareto solution set (crowding distance assignment is used after reproduction steps)

Fast non-domination sorting has to be performed twice, i.e.

1. after population initialization
2. after parent and offspring population combination.

In selection process, crowded comparison operator is used based on non domination rank and crowding distance properties. So, crowding distance assignment need to be done before selection process.



Basic idea behind bringing a strong Pareto optimal solution ,

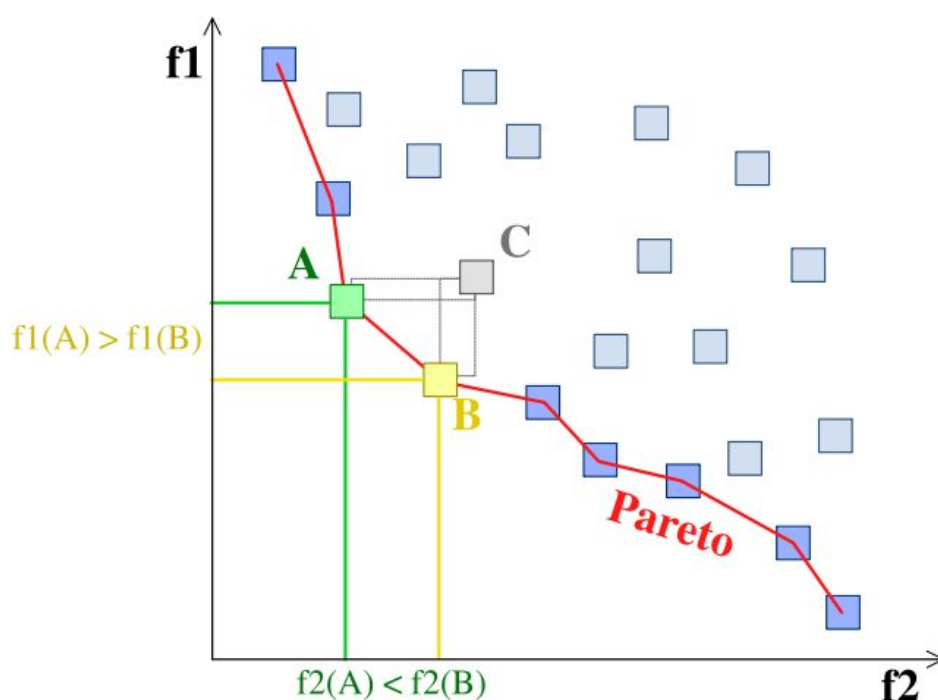
Pareto front

Idea:

The Pareto front (or Pareto frontier) is a framework for partially evaluating a set of "actions" with multi-dimensional outputs assuming a very weak "desirability" partial ordering which only applies only when one processes is better (or at least as good) for all the outputs. It is useful for reducing a

set of candidates prior to further analysis.

Given a set SS of possible actions (which might be a process, a technology, etc) one can discard any actions which are strictly dominated by another element of SS to get a reduced set of actions $S' \subseteq S$ which is called the Pareto front. This name is inspired by the way that, in N -dimensions, the remaining points are the “outer shell” of SS with the discarded points on the interior. This is shown in 2-D in the following plot:



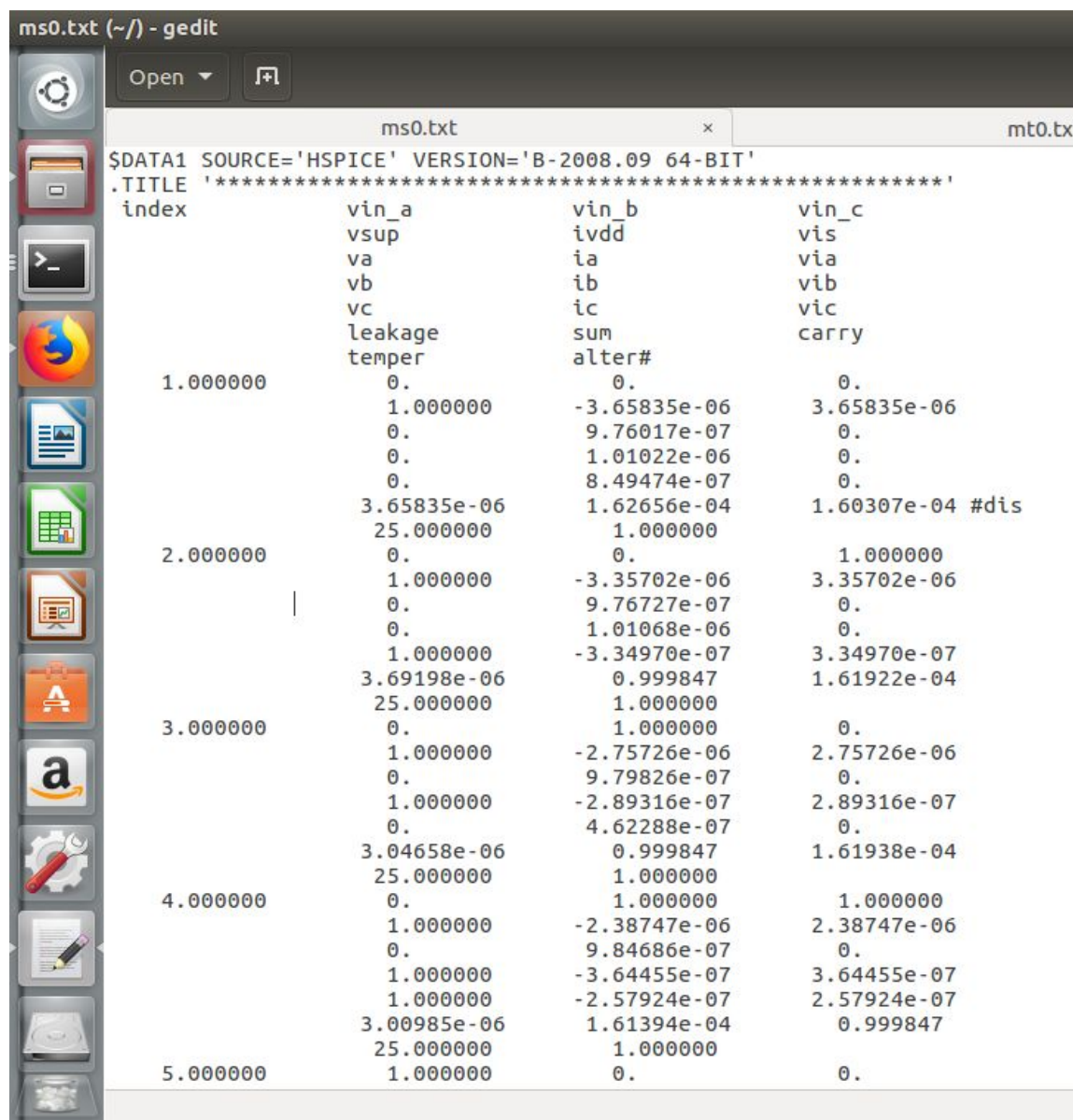
(Image from wikipedia page referenced below.) Another description of the Pareto front is as the set of *minimal elements*

The intuitive meaning of the Pareto front is that elements which are not on the front are never the best choice, because there's some element on the front which is at least as good on every variable. In contrast, elements which are on the front could be the best choice, depending how the user

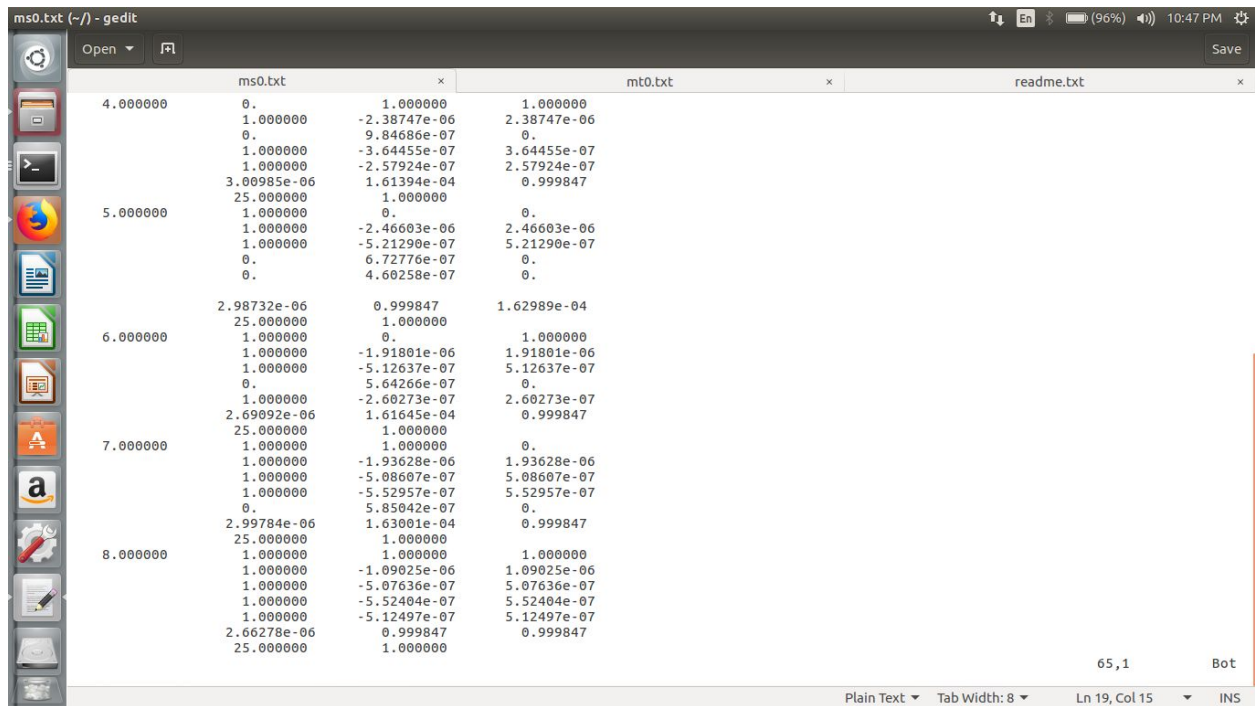
decides to trade-off the weights of the various variables.

By doing so, we arrive at Pareto optimal solution, since we take care of elitism perform non dominated sorting, we obtain a solution that is least dominated with respect to every parameter.

Initial values :



index	vin_a	vin_b	vin_c	
1.000000	0.	0.	0.	
	1.000000	-3.65835e-06	3.65835e-06	
	0.	9.76017e-07	0.	
	0.	1.01022e-06	0.	
	0.	8.49474e-07	0.	
	3.65835e-06	1.62656e-04	1.60307e-04	#dis
	25.000000	1.000000		
2.000000	0.	0.	1.000000	
	1.000000	-3.35702e-06	3.35702e-06	
	0.	9.76727e-07	0.	
	0.	1.01068e-06	0.	
	1.000000	-3.34970e-07	3.34970e-07	
	3.69198e-06	0.999847	1.61922e-04	
	25.000000	1.000000		
3.000000	0.	1.000000	0.	
	1.000000	-2.75726e-06	2.75726e-06	
	0.	9.79826e-07	0.	
	1.000000	-2.89316e-07	2.89316e-07	
	0.	4.62288e-07	0.	
	3.04658e-06	0.999847	1.61938e-04	
	25.000000	1.000000		
4.000000	0.	1.000000	1.000000	
	1.000000	-2.38747e-06	2.38747e-06	
	0.	9.84686e-07	0.	
	1.000000	-3.64455e-07	3.64455e-07	
	1.000000	-2.57924e-07	2.57924e-07	
	3.00985e-06	1.61394e-04	0.999847	
	25.000000	1.000000		
5.000000	1.000000	0.	0.	

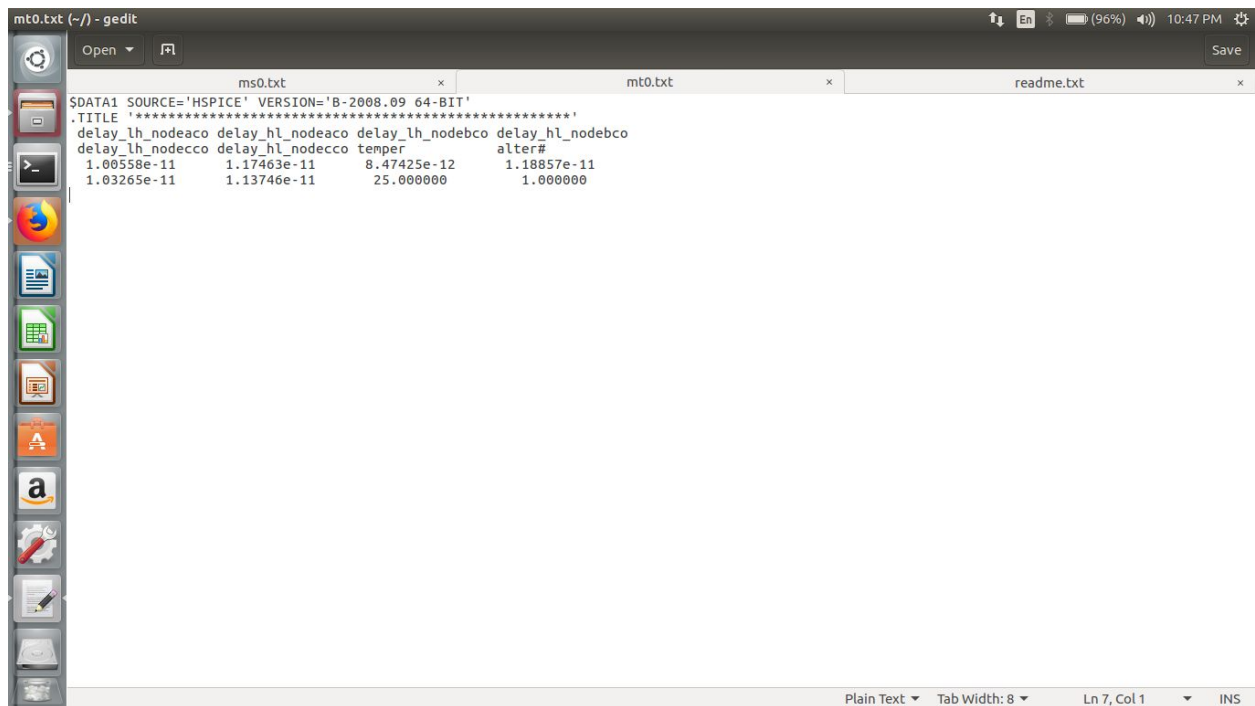


ms0.txt (~/) - gedit

	ms0.txt	mt0.txt	readme.txt
4.000000	0. 1.000000 0. 1.000000 1.000000 3.00985e-06 25.000000	1.000000 -2.38747e-06 9.84686e-07 -3.64455e-07 -2.57924e-07 1.61394e-04 1.000000	1.000000 2.38747e-06 0. 3.64455e-07 2.57924e-07 0.999847
5.000000	1.000000 1.000000 1.000000 0. 0.	0. -2.46603e-06 -5.21290e-07 6.72776e-07 4.60258e-07	0. 2.46603e-06 5.21290e-07 0. 0.
6.000000	2.98732e-06 25.000000 1.000000 1.000000 1.000000 0. 1.000000 2.69092e-06 25.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 2.99784e-06 25.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 2.66278e-06 25.000000	0.999847 1.000000 0. -1.91801e-06 -5.12637e-07 5.64266e-07 -2.60273e-07 1.61645e-04 1.000000 1.000000 -1.93628e-06 -5.08607e-07 -5.52957e-07 5.85042e-07 1.63001e-04 1.000000 1.000000 -1.09025e-06 -5.07636e-07 -5.52404e-07 -5.12497e-07 0.999847 1.000000	1.62989e-04 1.000000 1.91801e-06 5.12637e-07 0. 2.60273e-07 0.999847 0. 1.93628e-06 5.08607e-07 5.52957e-07 0. 0.999847 1.000000 1.09025e-06 5.07636e-07 5.52404e-07 5.12497e-07 0.999847

65,1 Bot

Plain Text Tab Width: 8 Ln 19, Col 15 INS



mt0.txt (~/) - gedit

```
$DATA1 SOURCE='HSPICE' VERSION='B-2008.09 64-BIT'
.TITLE '*****'
delay_lh_nodeaco delay_hl_nodeaco delay_lh_nodebco delay_hl_nodebco
delay_lh_nodecco delay_hl_nodecco temper alter#
1.00558e-11 1.17463e-11 8.47425e-12 1.18857e-11
1.03265e-11 1.13746e-11 25.000000 1.000000
```

Plain Text Tab Width: 8 Ln 7, Col 1 INS

itvlsi26@cvest:~/rucfinal/ruc			
	0.	2.02661e-07	0.
	7.75777e-07	9.30436e-05	4.71210e-05
2.000000	25.000000	1.000000	
	0.	0.	1.000000
	1.000000	-6.57906e-07	6.57906e-07
	0.	2.24300e-07	0.
	0.	2.40339e-07	0.
	1.000000	-1.05753e-07	1.05753e-07
	7.63659e-07	0.999826	4.71811e-05
	25.000000	1.000000	
3.000000	0.	1.000000	0.
	1.000000	-4.87453e-07	4.87453e-07
	0.	2.24886e-07	0.
	1.000000	-9.55844e-08	9.55844e-08
	0.	1.08568e-07	0.
	5.83038e-07	0.999826	4.71673e-05
	25.000000	1.000000	
4.000000	0.	1.000000	1.000000
	1.000000	-4.54604e-07	4.54604e-07
	0.	2.23333e-07	0.
	1.000000	-1.19329e-07	1.19329e-07
	1.000000	-8.10255e-08	8.10255e-08
	6.54958e-07	9.24653e-05	0.999633
	25.000000	1.000000	
5.000000	1.000000	0.	0.
	1.000000	-4.09623e-07	4.09623e-07
	1.000000	-1.80058e-07	1.80058e-07
	0.	1.41998e-07	0.
	0.	1.07617e-07	0.
	5.89681e-07	0.999826	4.72099e-05
	25.000000	1.000000	
6.000000	1.000000	0.	1.000000
	1.000000	-3.44962e-07	3.44962e-07
	1.000000	-1.82605e-07	1.82605e-07
	0.	1.20019e-07	0.
	1.000000	-8.18834e-08	8.18834e-08
	6.09450e-07	9.26321e-05	0.999633
	25.000000	1.000000	
7.000000	1.000000	1.000000	0.
	1.000000	-3.68608e-07	3.68608e-07
	1.000000	-1.81685e-07	1.81685e-07
	1.000000	-1.88103e-07	1.88103e-07
	0.	1.37033e-07	0.

```

itvlsi26@cvest:~/rucfinal/ruc
4.000000 0. 1.000000 1.000000
1.000000 -4.54604e-07 4.54604e-07
0. 2.23333e-07 0.
1.000000 -1.19329e-07 1.19329e-07
1.000000 -8.10255e-08 8.10255e-08
6.54958e-07 9.24653e-05 0.999633
25.000000 1.000000
5.000000 1.000000 0.
1.000000 -4.09623e-07 4.09623e-07
1.000000 -1.80058e-07 1.80058e-07
0. 1.41998e-07 0.
0. 1.07617e-07 0.
5.89681e-07 0.999826 4.72099e-05
25.000000 1.000000
6.000000 1.000000 0. 1.000000
1.000000 -3.44962e-07 3.44962e-07
1.000000 -1.82605e-07 1.82605e-07
0. 1.20019e-07 0.
1.000000 -8.18834e-08 8.18834e-08
6.09450e-07 9.26321e-05 0.999633
25.000000 1.000000
7.000000 1.000000 1.000000 0.
1.000000 -3.68608e-07 3.68608e-07
1.000000 -1.81685e-07 1.81685e-07
1.000000 -1.88103e-07 1.88103e-07
0. 1.37033e-07 0.
7.38396e-07 9.33960e-05 0.999633
25.000000 1.000000
8.000000 1.000000 1.000000 1.000000
1.000000 -1.52799e-07 1.52799e-07
1.000000 -1.79906e-07 1.79906e-07
1.000000 -1.87686e-07 1.87686e-07
1.000000 -1.59938e-07 1.59938e-07
6.80329e-07 0.999826 0.999634
25.000000 1.000000

[itvlsi26@cvest ruc]$ cat gen1208/mutation/delay/84_delay.mt0
SDATA1 SOURCE='HSPICE' VERSION='B-2008.09 64-BIT'
.TITLE '*****'
delay_lh_nodeaco delay_hl_nodeaco delay_lh_nodebco delay_hl_nodebco
delay_lh_nodecco delay_hl_nodecco temper alter#
1.28864e-11 1.00405e-11 1.20291e-11 1.16464e-11
1.29130e-11 1.01722e-11 25.000000 1.000000
[itvlsi26@cvest ruc]$

```

Initial max delay:1.18 e-11

Final max delay:1.29 e-11

Bounds:10%

Initial average leakage:3.09 e-06

Final average leakage:6.738 e-07

Reduction:78% (More than expected)

Challenges

- ❑ Since we are deal with Large population of values of W,L 's and we perform genetic operations and sorting techniques on the whole population,produce a generation of W,L s with optimised values of Leakages and Delays, it's hard for us to produce one single pair of W,L with least Leakage and Delay.