[[1]](#footnote-1)

Reordering Test Vectors with Particle Swarm Optimization

Jordan Richardson

Department of Electrical and Computer Engineering, Auburn University, Auburn, USA

Email: jar0012auburn.edu

*Abstract*—Power consumption during testing of combinational circuits is of concern in industry. For a given set of test vectors, power consumed during testing is related to the number of bits that change between each applied vector (a.k.a. Hamming distance). The amount of power consumed can thus be minimized if the test vectors are reordered in a way that minimizes the total Hamming distance. This manuscript proposes to apply Particle Swarm Optimization (PSO) to reorder test vectors. Experimental results show that test power is reduced by 23% when PSO is used to reorder vectors.

*Index Terms*—particle swarm optimization, PSO, traveling salesperson problem, TSP, low power, testing

# INTRODUCTION

Testing combinational circuits requires the sequential application of test vectors. Deriving a set of test vectors that cover all possible faults, and is small enough to be practical is a nontrivial task[1]. Once a suitable set of test vectors has been generated, the goal changes to minimizing the power consumed during testing. For combinational CMOS circuits, the power is consumed when changes in the input propagate through the circuit. If the input is static, then obviously no changes occur. This leads to the assumption that amount of power consumed when applying input vectors is proportional to the number of bits that are different between the vectors, otherwise known as the Hamming distance. Therefore, for a given set of test vectors, the power consumption will be minimized if the vectors are ordered such that the total Hamming distance is minimized.

This problem of test vector reordering is equivalent to the classis Traveling Salesperson Problem (TSP). Many different techniques have been applied to the TSP over its long history, so for the sake of brevity, this paper will focus on applying an optimization technique to reordering vectors, and evaluate the results.

The optimization algorithm that will be examined is a modified version of Particle Swarm Optimization (PSO). The algorithm will be implemented in the MATLAB programming environment, and the results evaluated using HSPICE to simulate power consumption in an ISCAS-85 C6288 16x16 bit multiplier (Fig. 1.).



Fig.1. ISCAS-85 C6288 Multiplier.

Image Credit: http://web.eecs.umich.edu/~jhayes/iscas.restore/c6288.html

The rest of this paper is organized as follows: Section II details PSO and the modifications to make it applicable to the TSP problem, Section III experimental results, and finally conclusions are drawn in Section IV.

# Brief Algorithm review

The goal of this paper was to implement and evaluate the suitability of Particle Swarm Optimization (PSO) to solve the Traveling Salesperson Problem (TSP). This section provides a review of the PSO algorithm, and the modifications implemented to make it applicable to the TSP.

## Original PSO

Particle Swarm Optimization[2] can be classified as a biologically inspired algorithm. Biologically inspired algorithms, such as PSO, Genetic Algorithms (GA), Ant Colony Optimization[3] (ACO), etc. This class of algorithm attempts to approximate some aspect of biological life in order to solve computational problems.

In particular, PSO can be thought of as an approximation of the way a flock of birds locate and congregate to the most abundant food sources. The birds do not know where the best food source is, but they can see where other birds are gathering, and adjust their flight paths to investigate. PSO is a simplified version of this type of behavior. Instead of birds, the problem space is filled with particles, which are attempting to minimize an objective function. The algorithm itself is very simple. The problem space is seeded with a user set number of particles. Each particle is given a random starting position and velocity. On each iteration of the algorithm, the fitness of every particle is evaluated, and the position of the best particle is recorded. Then, the velocity and position of each particle is updated according to and respectively.

|  |  |  |
| --- | --- | --- |
|  | $$V\_{i,k+1} =V\_{i,k}+α\*r\_{1}\*\left(PB-P\_{i,k}\right)+β\*r\_{2}\*\left(GB-P\_{i,k}\right)$$ |  |
|  | $$P\_{i,k+1} =P\_{i,k}+V\_{i,k}$$ |  |

 Where $V\_{i,k}$ is the velocity of particle $i$ on iteration $k$, $α$ and $β$ are learning constants, $r\_{1}$ and $r\_{2}$ are random numbers between zero and one, $PB$ is the best position ever obtained by particle $k$, and $GB$ is the best position ever obtained by any member of the population.

In practice, there are a few more settings/modifications, such a maximum allowed velocity used to limit the rate at which particle can move through the problem space, or an inertia factor that causes particle to resist changing their current velocity; but the above description covers the major portions of the algorithm.

## Discrete PSO

The original PSO algorithm works well for optimization problems where the solutions can be real numbers. For a path to be a valid solution to the TSP, it must only contain node values between one and the total number of nodes. In addition, the solution cannot have any repeats. These restrictions necessitate modifications to the PSO algorithm. There has been extensive research on modifying PSO for discrete problems, and for TSP. For a fairly recent literature review, see [4].

The modifications described below are modeled on those presented in [5], which should be read for a complete derivation. In brief, the modifications define a new swap operator, and redefines $+$ and $–$ operators.

For a sequence $S$ made up of $n$ nodes, the swap operator (SO) is defined as:

|  |  |  |
| --- | --- | --- |
|  | $$S^{'}=S+SO\left(S,i,j\right)$$ |  |

Where the new sequence $S'$ is equal to the old sequence $S$, with the node $i$ swapped with the node $j$. Furthermore, we can define a swap sequence $SS$ as a series of one or more swap operators performed sequentially. The $–$ operator is defined as the swap sequence necessary to transform one sequence into another.

|  |  |  |
| --- | --- | --- |
|  | $$SS=S\_{1}-S\_{2}$$ |  |
|  | $$S\_{1}=S\_{2}+SS$$ |  |

With these new definitions, the position and velocity update rules in and are modified as

|  |  |  |
| --- | --- | --- |
|  | $$V\_{i,k+1} =V\_{i,k}+α\*\left(PB-P\_{i,k}\right)+β\*\*\left(GB-P\_{i,k}\right)$$ |  |
|  | $$P\_{i,k+1} =P\_{i,k}+V\_{i,k}$$ |  |

Where $α$ and $β$ are now random numbers between zero and one, and control the probability of the all swap operators in the swap sequence being maintained or removed. With the new update rules in and , the PSO algorithm can now be implemented.

# Experimental Results

## Experimental setup

The ISCAS-85 C6288 16x16 multiplier was chosen as the benchmark circuit for simulation. The initial set of 10 test vectors can be seen on Table I. These vectors were generated in [1].

 Table I Initial Test Vector Order

|  |  |
| --- | --- |
| Vector # | 32 bit input vectors |
| 1 | 11011011011011011101111111111111 |
| 2 | 01101101101101101111111111111111 |
| 3 | 00000000000000000010111111111111 |
| 4 | 10110110110110111101111111111111 |
| 5 | 11111111111111111101010101010101 |
| 6 | 11111111111111110110101010101010 |
| 7 | 00111111111111011101010101010101 |
| 8 | 00111111111111011010101010101011 |
| 9 | 11101101101101100010111111111111 |
| 10 | 11011011011011001010101010101010 |

The experimental setup consisted of two parts. First, MATLAB code was written to perform PSO on the test vectors. Ten trials were performed. Each trial started with 100 random initial particles, and was allowed to operate for 50 iterations. After 50 iterations, the best solution was saved, and the next trial started. The result was a set of 10 different orders for the test vectors. For comparison, a set 10 random orders were also generated. The second part of the simulation utilized HSPICE to simulate a netlist of the c6288 circuit for all of the test vector orders, using 45nm PTM metal gate/high-k CMOS technology. The MATLAB code can be found in Appendix I, and the Verilog file used to generate the HSPICE netlist, sample input file, and PTM files can be found in Appendix II.

## Results

The first set of results was produced in MATLAB. Table II shows the total Hamming distance for the original order, the 10 random orders, and the 10 PSO orders. For the orderings used, and a sample wave form, see Appendix III.

Table II Hamming Distances

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trial | Original | Random | PSO |
| Distance | 1 | 128 | 118 | 79 |
| 2 | - | 126 | 83 |
| 3 | - | 108 | 82 |
| 4 | - | 122 | 81 |
| 5 | - | 141 | 79 |
| 6 | - | 142 | 77 |
| 7 | - | 128 | 79 |
| 8 | - | 122 | 85 |
| 9 | - | 117 | 77 |
| 10 | - | 118 | 83 |

It can be seen from Table I that all of the PSO produced orderings resulted in a smaller Hamming distance than the original, or the random orders.

The next step was to simulate the different test vector orderings with HSPICE. Table III shows the numerical results, while Fig. 3 is a graphical representation of the same data.

Table III Average Power

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trial | Original | Random | PSO |
| Average Power (mW) | 1 | 6.7780 | 5.5994 | 5.2648 |
| 2 | - | 6.4710 | 5.6193 |
| 3 | - | 5.8394 | 5.0161 |
| 4 | - | 7.0186 | 5.2858 |
| 5 | - | 6.4496 | 5.0907 |
| 6 | - | 6.5014 | 5.1154 |
| 7 | - | 5.7309 | 5.4610 |
| 8 | - | 5.9993 | 5.0148 |
| 9 | - | 6.3564 | 5.1154 |
| 10 | - | 5.6098 | 5.1429 |

Fig.2. Comparing path length with average power consumption

 It can be easily seen in Fig. 2 that the PSO vector orders produce lower power consumption than the random vectors, and improve on the original order. Table IV shows the percent reduction in power consumption for the random and PSO ordering when compared to the original.

Table IV Percent Reduction in Power

|  |  |  |  |
| --- | --- | --- | --- |
|  | Trial | Random | PSO |
| Percent Improvement (%) | 1 | 17.39 | 22.33 |
| 2 | 4.53 | 17.10 |
| 3 | 13.85 | 25.99 |
| 4 | -3.55 | 22.02 |
| 5 | 4.85 | 24.89 |
| 6 | 4.08 | 24.53 |
| 7 | 15.45 | 19.43 |
| 8 | 11.49 | 26.01 |
| 9 | 6.22 | 24.53 |
| 10 | 17.23 | 24.12 |

 The PSO orderings reduce power consumption by an average of 23%, and the random orderings reduce power by an average of 9%.

#  Conclusions and Future Work

Power consumed during testing of combinational circuits can be significant. The power consumed can be reduced by proper reordering of vectors to minimize Hamming distance. In this paper we have shown how to use Particle Swarm Optimization to reorder test vectors. Experimental results show a significant reduction in power consumption as a result.

There are still many areas that need more investigation. The PSO algorithm needs further testing and improvement. It would be interesting to examine the effect of increasing the number of particles to increasing the maximum number of iterations, and see which has the greater impact on the quality of the solution. Larger sample sizes, as well as different benchmark circuits/initial vector sets would provide useful insight into the scalability of the proposed approach.

Acknowledgement

I would like to thank Dr. Agrawal for his teaching and allowing me to work on this project. I would also like to thank Murali Dharan for his invaluable assistance with the simulations tools.

References

[1] K. R. Kantipudi and V. D. Agrawal, “A Reduced Complexity Algorithm for Minimizing N-Detect Tests,” in *, 20th International Conference on VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems*, 2007, pp. 492–497.

[2] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *, IEEE International Conference on Neural Networks, 1995. Proceedings*, 1995, vol. 4, pp. 1942–1948 vol.4.

[3] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Scituate, MA, USA: Bradford Company, 2004.

[4] W.-N. Chen, J. Zhang, H. S. H. Chung, W.-L. Zhong, W. Wu, and Y. Shi, “A Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization Problems,” *IEEE Trans. Evol. Comput.*, vol. 14, no. 2, pp. 278–300, Apr. 2010.

[5] K.-P. Wang, L. Huang, C.-G. Zhou, and W. Pang, “Particle swarm optimization for traveling salesman problem,” in *2003 International Conference on Machine Learning and Cybernetics*, 2003, vol. 3, pp. 1583–1585 Vol.3.

Appendix I

|  |  |
| --- | --- |
|  | clear all; V1=[1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1];V2=[0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1];V3=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1];V4=[1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1];V5=[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1];V6=[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0];V7=[0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1];V8=[0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,1];V9=[1,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,0,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1];V10=[1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0];V=[V1;V2;V3;V4;V5;V6;V7;V8;V9;V10]; % V=randi(2,50,32)-1;nvec=size(V,1);distanceMat=zeros(nvec,nvec); for i=1:nvec for j=1:nvec distanceMat(i,j)=sum(abs(V(i,:)-V(j,:))); endend% s=48315965;% control random seed for repeatability, comment out for random vectors% rng(s);ntrial=10; nparticles=100;max\_iter=50;VMAX=1;inertia=1;c1=0.5; c2=0.5;bestDists=zeros(ntrial,1);bestPaths=zeros(ntrial,nvec);numberOfIterationsToBest=zeros(ntrial,1);reorderedVecs=cell(1,ntrial);ticfor k=1:ntrial % distances=zeros(nparticles,1);% velocities=zeros(nparticles,nvec);particles=cell(nparticles,1);%% initialize particles globalBestDistance=inf;globalBestPath=nan\*ones(1,nvec);for i=1:nparticles% particle.presentPath=randperm(nvec); particle.presentPath=randperm(nvec+2,nvec)-1; particle.presentPath=fixParticle(particle.presentPath); particle.velocity=[]; particle.distance=evaluatePath(distanceMat,particle.presentPath); particle.personalBestPath=particle.presentPath; particle.personalBestDistance=particle.distance; if (particle.distance<globalBestDistance) globalBestDistance=particle.distance; globalBestPath=particle.presentPath; end particles{i}=particle;end  % bestPath=zeros(1,nvec);for i=1:max\_iter inertia=0.9-(0.9-0.4)\*i/max\_iter; for j=1:nparticles% [particles{j}.presentPath,particles{j}.velocity]=updateParticle(particles{j}.presentPath,particles{j}.velocity,globalBestPath,particles{j}.personalBestPath,c1,c2); particles{j}.distance=evaluatePath(distanceMat,particles{j}.presentPath); if (particles{j}.distance<particles{j}.personalBestDistance) particles{j}.personalBestDistance=particles{j}.distance; particles{j}.personalBestPath=particles{j}.presentPath; end   if (particles{j}.distance<globalBestDistance) globalBestDistance=particles{j}.distance; globalBestPath=particles{j}.presentPath; numberOfIterationsToBest(k)=i; end end for j=1:nparticles% [particles{j}.presentPath,particles{j}.velocity]=updateParticle(particles{j}.presentPath,particles{j}.velocity,globalBestPath,particles{j}.personalBestPath,inertia,c1,c2,VMAX);[particles{j}.presentPath,particles{j}.velocity]=updateParticleDiscrete(particles{j}.presentPath,particles{j}.velocity,globalBestPath,particles{j}.personalBestPath,inertia,c1,c2,VMAX); particles{j}.presentPath=fixParticle(particles{j}.presentPath); end if (globalBestDistance<=77) break end% inertia=endbestDists(k)=globalBestDistance;bestPaths(k,:)=globalBestPath;reorderedVecs{k}=V(globalBestPath,:);endtime=tocbestDist=min(bestDists)% ind=find(bestIter=numberOfIterationsToBest(bestDists==bestDist)averageDist=sum(bestDists)/length(bestDists)averageIter=sum(numberOfIterationsToBest)/length(numberOfIterationsToBest) % ntrial=50;% % nparticles=5000;% max\_iter=10;fname=strcat('nt',num2str(ntrial),'np',num2str(nparticles),'niter',num2str(max\_iter));% save fnamesave(fname) |
|  | function [particle,velocity]=updateParticleDiscrete(particle,velocity,gbest,pbest,inertia,c1,c2,VMAX)s1=createSwapSequence(pbest,particle);for i=1:size(s1,1)if (rand(1) <=c1) s1(i,:)=[0,0];endends2=createSwapSequence(gbest,particle);for i=1:size(s2,1) if (rand(1) <=c2) s2(i,:)=[0,0]; endend% if (rand(1) <=c2)% s2=[];% endfor i=1:size(velocity,1) if (rand(1) <=inertia) velocity(i,:)=[0,0]; endend% if (rand(1) >=inertia)% velocity=[velocity;s1;s2];% end% velocity=[velocity;c1\*rand(1)\*(createSwapSequence(pbest,particle));c2\*rand(1)\*(createSwapSequence(gbest,particle)); % velocity(velocity > VMAX)=VMAX;% particle=particle+round(velocity);if (rand(1)>inertia) velocity=[s1;s2];endparticle=swapSequence(particle,velocity); |
|  | function swapOrder=createSwapSequence(sequence1,sequence2)% clear all;% % s=12341;rng(s);% % sequence1=1:1:5% % sequence2=randperm(5)% sequence1=[1,2,3,4,5];% sequence2=[2,3,1,5,4];l1=length(sequence1);l2=length(sequence2);swapOrder=zeros(l1,2);cnt=0;for i=1:l1 for j=1:l2 if (sequence1(i)==sequence2(j) && i~=j) cnt=cnt+1; swapOrder(cnt,:)=[i,j];  sequence2=swapOperator(sequence2,i,j); end end% if (sequence1==sequence2)% break% endendswapOrder=swapOrder(1:cnt,:); % [C,ia,ib]=union(sequence1,sequence2); % test=swapSequence(sequence1,swapOrder) |
|  | function sequence=swapSequence(sequence,SO)for i=1:size(SO,1) sequence=swapOperator(sequence,SO(i,1),SO(i,2));end |
|  | function sequence=swapOperator(sequence,i1,i2)if (i1~=0 && i2 ~=0)sequence([i1,i2])=sequence([i2,i1]);end |

Appendix II

|  |  |
| --- | --- |
| 45nm\_MGK.pm | \* PTM 45nm Metal Gate / High-K .model n nmos level=54 +version = 4.0 binunit = 1 paramchk= 1 mobmod = 0 +capmod = 2 igcmod = 1 igbmod = 1 geomod = 1 +diomod = 1 rdsmod = 0 rbodymod= 1 rgatemod= 1 +permod = 1 acnqsmod= 0 trnqsmod= 0 +tnom = 27 toxe = 9e-010 toxp = 6.5e-010 toxm = 9e-010 +dtox = 2.5e-010 epsrox = 3.9 wint = 5e-009 lint = 2.7e-009 +ll = 0 wl = 0 lln = 1 wln = 1 +lw = 0 ww = 0 lwn = 1 wwn = 1 +lwl = 0 wwl = 0 xpart = 0 toxref = 9e-010 xl = -20e-9+dlcig = 2.7e-009 +vth0 = 0.3423 k1 = 0.2 k2 = 0 k3 = 0 +k3b = 0 w0 = 2.5e-006 dvt0 = 1 dvt1 = 2 +dvt2 = 0 dvt0w = 0 dvt1w = 0 dvt2w = 0 +dsub = 0.078 minv = 0.05 voffl = 0 dvtp0 = 1e-010 +dvtp1 = 0.1 lpe0 = 0 lpeb = 0 xj = 1.4e-008 +ngate = 1e+023 ndep = 6.5e+018 nsd = 2e+020 phin = 0 +cdsc = 0 cdscb = 0 cdscd = 0 cit = 0 +voff = -0.13 nfactor = 1.9 eta0 = 0.0055 etab = 0 +vfb = -1.058 u0 = 0.02947 ua = -5e-010 ub = 1.7e-018 +uc = 0 vsat = 159550 a0 = 1 ags = 0 +a1 = 0 a2 = 1 b0 = 0 b1 = 0 +keta = 0.04 dwg = 0 dwb = 0 pclm = 0.06 +pdiblc1 = 0.001 pdiblc2 = 0.001 pdiblcb = -0.005 drout = 0.5 +pvag = 1e-020 delta = 0.01 pscbe1 = 2.0e+009 pscbe2 = 1e-007 +fprout = 0.2 pdits = 0.01 pditsd = 0.23 pditsl = 2300000 +rsh = 5 rdsw = 105 rsw = 52.5 rdw = 52.5 +rdswmin = 0 rdwmin = 0 rswmin = 0 prwg = 0 +prwb = 0 wr = 1 alpha0 = 0.074 alpha1 = 0.005 +beta0 = 30 agidl = 0.0002 bgidl = 2.1e+009 cgidl = 0.0002 +egidl = 0.8 aigbacc = 0.012 bigbacc = 0.0028 cigbacc = 0.002 +nigbacc = 1 aigbinv = 0.014 bigbinv = 0.004 cigbinv = 0.004 +eigbinv = 1.1 nigbinv = 3 aigc = 0.018029 bigc = 0.0029 +cigc = 0.002 aigsd = 0.018029 bigsd = 0.0029 cigsd = 0.002 +nigc = 1 poxedge = 1 pigcd = 1 ntox = 1 +xrcrg1 = 12 xrcrg2 = 5 +cgso = 1e-010 cgdo = 1e-010 cgbo = 0 cgdl = 7.5e-013 +cgsl = 7.5e-013 clc = 1e-007 cle = 0.6 cf = 1.1e-010 +ckappas = 0.6 ckappad = 0.6 vfbcv = -1 acde = 1 +moin = 15 noff = 1 voffcv = 0 +kt1 = -0.154 kt1l = 0 kt2 = 0.022 ute = -1.1 +ua1 = 1e-009 ub1 = -1e-018 uc1 = -5.6e-011 prt = 0 +at = 33000 +fnoimod = 1 tnoimod = 0 noia = 6.25e+041 noib = 3.125e+026 +noic = 8.75e+009 em = 41000000 af = 1 ef = 1 +kf = 0 tnoia = 1.5 tnoib = 3.5 ntnoi = 1 +jss = 1.2e-006 jsws = 2.4e-013 jswgs = 2.4e-013 njs = 1 +ijthsfwd= 0.1 ijthsrev= 0.1 bvs = 10 xjbvs = 1 +jsd = 1.2e-006 jswd = 2.4e-013 jswgd = 2.4e-013 xjbvd = 1 +pbs = 1 cjs = 0.0018 mjs = 0.5 pbsws = 1 +cjsws = 1.2e-010 mjsws = 0.33 cjswgs = 2.1e-010 cjd = 0.0018 +cjswd = 1.2e-010 mjswd = 0.33 pbswgd = 1 cjswgd = 2.1e-010 +mjswgd = 0.33 tpb = 0 tcj = 0 tpbsw = 0 +tcjsw = 0 tpbswg = 0 tcjswg = 0 xtis = 3 +dmcg = 0 dmci = 0 dmdg = 0 dmcgt = 0 +dwj = 0 xgw = 0 xgl = 0 +rshg = 0.4 gbmin = 1e-010 rbpb = 5 rbpd = 15 +rbps = 15 rbdb = 15 rbsb = 15 ngcon = 1 .model p pmos level = 54 +version = 4.0 binunit = 1 paramchk= 1 mobmod = 0+capmod = 2 igcmod = 1 igbmod = 1 geomod = 1+diomod = 1 rdsmod = 0 rbodymod= 1 rgatemod= 1+permod = 1 acnqsmod= 0 trnqsmod= 0+tnom = 27 toxe = 9.2e-010 toxp = 6.5e-010 toxm = 9.2e-010+dtox = 2.7e-010 epsrox = 3.9 wint = 5e-009 lint = 2.7e-009+ll = 0 wl = 0 lln = 1 wln = 1+lw = 0 ww = 0 lwn = 1 wwn = 1+lwl = 0 wwl = 0 xpart = 0 toxref = 9.2e-010 xl = -20e-9+dlcig = 2.7e-009+vth0 = -0.23122 k1 = 0.2 k2 = -0.01 k3 = 0+k3b = 0 w0 = 2.5e-006 dvt0 = 1 dvt1 = 2+dvt2 = -0.032 dvt0w = 0 dvt1w = 0 dvt2w = 0+dsub = 0.1 minv = 0.05 voffl = 0 dvtp0 = 1e-011+dvtp1 = 0.05 lpe0 = 0 lpeb = 0 xj = 1.4e-008+ngate = 1e+023 ndep = 2.8e+018 nsd = 2e+020 phin = 0+cdsc = 0 cdscb = 0 cdscd = 0 cit = 0+voff = -0.13 nfactor = 1.9 eta0 = 0.0049 etab = 0+vfb = -1.058 u0 = 0.00391 ua = -5e-010 ub = 1.6e-018+uc = 0 vsat = 78000 a0 = 1 ags = 1e-020+a1 = 0 a2 = 1 b0 = 0 b1 = 0+keta = -0.047 dwg = 0 dwb = 0 pclm = 0.1+pdiblc1 = 0.001 pdiblc2 = 0.001 pdiblcb = 3.4e-008 drout = 0.6+pvag = 1e-020 delta = 0.01 pscbe1 = 2e+009 pscbe2 = 9.58e-007+fprout = 0.2 pdits = 0.08 pditsd = 0.23 pditsl = 2300000+rsh = 5 rdsw = 105 rsw = 52.5 rdw = 52.5+rdswmin = 0 rdwmin = 0 rswmin = 0 prwg = 0+prwb = 0 wr = 1 alpha0 = 0.074 alpha1 = 0.005+beta0 = 30 agidl = 0.0002 bgidl = 2.1e+009 cgidl = 0.0002+egidl = 0.8 aigbacc = 0.012 bigbacc = 0.0028 cigbacc = 0.002+nigbacc = 1 aigbinv = 0.014 bigbinv = 0.004 cigbinv = 0.004+eigbinv = 1.1 nigbinv = 3 aigc = 0.010687 bigc = 0.0012607+cigc = 0.0008 aigsd = 0.010687 bigsd = 0.0012607 cigsd = 0.0008+nigc = 1 poxedge = 1 pigcd = 1 ntox = 1+xrcrg1 = 12 xrcrg2 = 5+cgso = 1e-010 cgdo = 1e-010 cgbo = 0 cgdl = 3e-011+cgsl = 3e-011 clc = 1e-007 cle = 0.6 cf = 1.1e-010+ckappas = 0.6 ckappad = 0.6 vfbcv = -1 acde = 1+moin = 15 noff = 1 voffcv = 0+kt1 = -0.14 kt1l = 0 kt2 = 0.022 ute = -1.1+ua1 = 1e-009 ub1 = -1e-018 uc1 = -5.6e-011 prt = 0+at = 33000+fnoimod = 1 tnoimod = 0 noia = 6.25e+041 noib = 3.125e+026+noic = 8.75e+009 em = 41000000 af = 1 ef = 1+kf = 0 tnoia = 1.5 tnoib = 3.5 ntnoi = 1+jss = 2e-007 jsws = 4e-013 jswgs = 4e-013 njs = 1+ijthsfwd= 0.1 ijthsrev= 0.1 bvs = 10 xjbvs = 1+jsd = 2e-007 jswd = 4e-013 jswgd = 4e-013 xjbvd = 1+pbs = 1 cjs = 0.0015 mjs = 0.5 pbsws = 1+cjsws = 9.4e-011 mjsws = 0.33 cjswgs = 2e-010 cjd = 0.0015+cjswd = 9.4e-011 mjswd = 0.33 pbswgd = 1 cjswgd = 2e-010+mjswgd = 0.33 tpb = 0 tcj = 0 tpbsw = 0+tcjsw = 0 tpbswg = 0 tcjswg = 0 xtis = 3+dmcg = 0 dmdg = 0 dmcgt = 0 xgw = 0+xgl = 0+rshg = 0.1 gbmin = 1e-012 rbpb = 50 rbpd = 50+rbps = 50 rbdb = 50 rbsb = 50 ngcon = 1  |
| test.vec | ; start of Pattern Definition sectionRADIX 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1vname A[15] A[14] A[13] A[12] A[11] A[10] A[9] A[8] A[7] A[6] A[5] A[4] A[3] A[2] A[1] A[0] B[15] B[14] B[13] B[12] B[11] B[10] B[9] B[8] B[7] B[6] B[5] B[4] B[3] B[2] B[1] B[0] IO I I I I I I I I I I I I I I I I I I I I I I I I I I I I I I I Iperiod 0.5tunit nsslope 0.01vih 1vil 0voh 0.7vol 0.3;Vector Table1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 10 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 11 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 11 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 11 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 00 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 10 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 11 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 11 1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 |
| c6288b.v | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* \* \* VERILOG BEHAVIORAL DESCRIPTION OF THE ISCAS-85 BENCHMARK CIRCUIT c6288 \* \* \* \* Function: 16 x 16 Multiplier \* \* \* \* Written by: Mark C. Hansen \* \* \* \* Last modified: Nov 12, 1997 \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/module Circuit6288h (in256, in239, in222, in205, in188, in171, in154, in137,  in120, in103, in86, in69, in52, in35, in18, in1, in528, in511, in494, in477, in460, in443, in426, in409, in392, in375, in358, in341, in324, in307, in290, in273, out6287, out6288, out6280, out6270, out6260, out6250, out6240, out6230,  out6220, out6210, out6200, out6190,  out6180, out6170, out6160, out6150,  out6123, out5971, out5672, out5308,  out4946, out4591, out4241, out3895, out3552, out3211, out2877, out2548, out2223, out1901, out1581, out545); input in256, in239, in222, in205, in188, in171, in154, in137,  in120, in103, in86, in69, in52, in35, in18, in1, in528, in511, in494, in477, in460, in443, in426, in409, in392, in375, in358, in341, in324, in307, in290, in273; output out6287, out6288, out6280, out6270, out6260, out6250, out6240, out6230,  out6220, out6210, out6200, out6190,  out6180, out6170, out6160, out6150,  out6123, out5971, out5672, out5308,  out4946, out4591, out4241, out3895, out3552, out3211, out2877, out2548, out2223, out1901, out1581, out545; wire [15:0] A, B; wire [31:0] P; assign A[15:0] = {in256, in239, in222, in205, in188, in171, in154, in137,  in120, in103, in86, in69, in52, in35, in18, in1}, B[15:0] = {in528, in511, in494, in477, in460, in443, in426, in409, in392, in375, in358, in341, in324, in307, in290, in273}, {out6287, out6288, out6280, out6270, out6260, out6250, out6240, out6230,  out6220, out6210, out6200, out6190,  out6180, out6170, out6160, out6150,  out6123, out5971, out5672, out5308,  out4946, out4591, out4241, out3895, out3552, out3211, out2877, out2548, out2223, out1901, out1581, out545} = P[31:0];  TopLevel6288b Ckt6288b (A, B, P);endmodule /\* Circuit6288b \*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/module TopLevel6288b (A, B, P); input[15:0] A, B; output[31:0] P; assign P = A\*B;endmodule /\* TopLevel6288b \*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  |

Appendix III

|  |  |  |
| --- | --- | --- |
| Trial | PSO order | random Order |
| 1 | 7->5->1->10->8->6->4->2->9->3 | 4->1->2->9->5->6->10->3->7->8 |
| 2 | 8->10->6->5->7->4->1->2->9->3 | 1->8->4->10->2->3->9->7->5->6 |
| 3 | 4->3->9->2->8->6->10->1->5->7 | 9->4->7->5->1->10->6->3->2->8 |
| 4 | 10->8->6->5->7->4->1->2->9->3 | 2->3->10->1->4->9->6->5->8->7 |
| 5 | 3->9->2->4->6->8->10->1->5->7 | 5->8->6->3->4->1->2->10->7->9 |
| 6 | 8->6->10->1->5->7->4->2->9->3 | 8->2->1->10->5->3->4->6->9->7 |
| 7 | 7->5->6->8->10->1->4->2->9->3 | 9->10->3->5->4->1->8->6->2->7 |
| 8 | 10->1->7->5->4->6->8->2->9->3 | 6->10->9->4->3->8->2->5->1->7 |
| 9 | 8->6->10->1->5->7->4->2->9->3 | 5->6->3->4->1->7->8->10->9->2 |
| 10 | 3->2->9->6->10->8->7->5->4->1 | 1->2->3->7->4->5->8->6->10->9 |



1. [↑](#footnote-ref-1)