2015 IEEE Symposium Series on Computational Intelligence

Transistor Sizing Using Particle Swarm Optimisation

Lyndon White*, Lyndon While†, Ben Deeks† and Farid Boussaid*
*School of Electrical, Electronic & Computer Engineering
The University of Western Australia, Perth, Australia
Email: lyndon.white@research.uwa.edu.au, farid.boussaid@uwa.edu.au
†School of Computer Science & Software Engineering
The University of Western Australia, Perth, Australia
Email: lyndon.while@uwa.edu.au, ben.deeks@graduate.uwa.edu.au

***Abstract*—We describe an application of particle swarm optimisation to the problem of determining the optimal sizing of transistors in an integrated circuit. The algorithm minimises the total area of silicon utilised by a given circuit, whilst maintaining the propagation delay of the circuit within a hard limit. It assesses designs using the well-known circuit simulation engine SPICE, making allowance for the inability of SPICE to assess poorly-designed circuits within a reasonable timeframe. Experiments on three different types of circuits demonstrate that the algorithm is able to derive excellent designs for a range of problem instances, including several problems where the Monte Carlo method is unable to find any feasible solutions at all.**

I. Introduction

Circuit design in micro-electronics can be considered as a two-stage process[13], [9]: designing a circuit schematic for a given specification, and then selecting a (usually large) number of parameters to determine the layout of the circuit in silicon. The second step is sometimes further split into two: determining the smallest possible size for the transistors in the circuit, subject to other design constraints; then determining a good layout for the transistors, given the connections required between them. Important objectives to be optimised throughout include the silicon area of the circuit, which directly influences production costs; and the propagation delay in the circuit, which directly influences its performance. The second step is a difficult task for designers, due to the number of parameters involved and the interactions and dependencies between them: even small changes can have major effects on the circuit's performance that are hard to predict. This task is commonly done manually by assigning parameters intuitively, and testing designs using a circuit simulation engine such as SPICE[6], [8]; or sometimes with the aid of simple tools such as Monte Carlo simulation[5]. Neither of these approaches is likely to produce designs that are optimal. The task seems ripe for the use of global optimisation technology.

The principal contribution of this paper is an implementation of particle swarm optimisation (PSO)[3] that takes a defined integrated circuit as input, and that determines dimensions for each of the transistors making up the circuit. PSO has a population of "particles" that move through the search space sampling different solutions: the particles remember the best solutions that they have already found, and they influence each other's movement by communicating these findings. SPICE is used to assess the performance of the designs generated. The goal is to minimise the total area of silicon utilised for the transistors, whilst observing tight constraints on the propagation delay in the circuit. Experiments on three circuit-types of varying complexity show that the algorithm derives excellent results for all of them; it easily out-performs the Monte Carlo method, and in some cases it returns feasible solutions where Monte Carlo fails to do so. The designs returned by the algorithm could be used as a good benchmark for manual designers of final circuit layouts, or they could serve as input to a second process that performs the layout process automatically.

The rest of the paper is structured as follows. Section II discusses previous work in this area. Section III defines the exact problem that we address here, and Section IV describes the PSO implementation that we have constructed. Section V presents the three circuits that we use to test our algorithm, and the results achieved. Section VI concludes the paper.

II. Previous Work

Transistor optimisation is a difficult task, where humans struggle to create good solutions, let alone optimal ones. The traditional approach of relying on designer intuition and experience, combined with simple heuristic methods and "trial and error" testing using circuit simulation programs such as SPICE[6], is difficult for novice designers and not necessarily optimal even with the most experienced design team. As such, many algorithms have been developed in order to automate the process, saving time and money while being a useful tool for those who are unable to take the traditional approach.

There are two broad categories of algorithms and methods which have been developed and documented for completing this optimisation task. Methods in the first category opt not to use a simulation engine: instead they build up and use a simplified model of the circuit using analytical techniques[2], or they use a combination of simulation-based macromodels[4] which has propagation delay embedded in the model, allowing for the information to be easily analysed to create optimal solutions. The second category of algorithms are those that make use of a simulation engine, most commonly the SPICE engine, to evaluate the performance and other details about the circuit given the transistor size parameters. This method can include using a non-linear optimization tool set such as in DELIGHT.SPICE[7], or stochastic elements such as Monte Carlo or genetic algorithms[10] to choose solutions to be tested by the simulator. The PSO approach fits into this latter category of simulation approaches.

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.46

The model-based approach has advantages over the other category of algorithms in its ability to easily extract propagation delay information for the model that is generated, reducing the amount of computation time needed. Not being reliant on a third-party simulation engine has additional optimisation benefits through not incurring the significant overhead required to interface with external systems. While the mathematical solving of the modelled system provides a definite optimal solution, the solving itself can be computationally intensive, and the solution produced is optimal only in the scope of the model: its actual performance depends on how accurately the model agrees with reality.

On the other hand, solutions generated by the simulation-driven approach are more likely to accurately reflect reality, due to the legacy of high accuracy which circuit simulators boast, SPICE in particular. This accuracy comes at the price of overheads in the algorithm due to the need to interface with external systems and longer evaluation times resulting from the computational intensity of the simulator engine. As a result, the algorithms in this category focus more heavily on the intelligent choice of solutions to test with the simulator, as opposed to the creation of an accurate and easy-to-solve model of the system. Within the category of simulation-based optimisation, approaches employing non-linear optimisation techniques tend to frequently choose consecutively better solutions, but are often limited to finding local optima in the solution space; while approaches employing stochastic elements will often evaluate less-optimal solutions as part of a global search of the solution space.

The PSO[3] algorithm presented here sits squarely within the simulator-driven category for the automatic optimisation of transistor-sizing. However, the technique incorporates stochastic elements, making the search more likely to find a global optimum.

III. The Problem

The most widely-used transistor today is the Metal Oxide Semiconductor Field Effect Transistor (MOSFET), which is the building block for Complementary Metal Oxide Semiconductor (CMOS) Logic[9]. MOSFETs are basically switches that come in two main types, P-channel and N-channel, that pass current on a low or high input signal, respectively. The transistors are not ideal switches: their operation involves a propagation delay due to the resistances and capacitances induced. Both increasing resistance and increasing capacitance will increase the delay, and assuming a rectangular transistor with fixed length and with width W, the capacitance $C \propto W$, while the resistance $R \propto 1/W$. Thus there is a trade-off in the sizing of MOSFETs: W must be not so large as to create high capacitance, but not so small as to create high resistance. Additionally, when several transistors are connected in a circuit, their individual characteristics must be designed such that they work together well.

The convention in the industry is to fix the length of a transistor at the minimum feature size available, and to treat propagation delay as a constraint to be kept below a threshold determined by the context in which the circuit will operate. Thus we can express the circuit design problem formally as follows.

> Given a set of transistors T making up a circuit C, determine a width for each transistor such that $\sum_{t \in T} width(t)$ is minimised, and such that the propagation delay for C is less than a pre-defined threshold $t_{p,max}$.

Note that we are ignoring here the requirement to produce an actual layout for the transistors. The designs produced by our algorithm could be used by manual designers as part of the process of deriving a layout, or they could be used as input to a subsequent automated layout process. Future work will include incorporating the layout requirement with sizing in a single optimisation process.

IV. The Algorithm

This section describes the structure of our algorithm for designing circuits: the basics of the PSO technology, the representation used for solutions, fitness calculations using SPICE, and dealing with invalid solutions generated.

A. PSO Basics

Particle swarm optimisation[3] is inspired by the behaviour of flocking birds. If we imagine a flock of birds searching for the best food source in a field, the birds view each point in the field as a potential solution, and each bird flies around the field sampling solutions over time. The birds choose where to fly next by two main methods: individually, by remembering the location of the best food source that they have seen so far, and favouring locations near that; and collectively, by communicating to other birds when they find new good sources, and also favouring locations near those. This is formalised in PSO by modelling birds as particles in n dimensions, by modelling time as a sequence of generations, by each bird remembering its best solution so far ($pbest$), and by the group remembering the best solution that any of them has seen so far ($gbest$). In each generation, each bird modifies its current velocity to favour both (its own) $pbest$ and $gbest$, each bird samples a new solution, and the birds update (their own) $pbest$ and $gbest$ appropriately. The process terminates either after a fixed number of generations, or when the flock has converged on the current location of $gbest$.

The principal parameters that must be set when applying PSO to a given problem are the size of the population, and three parameters that determine how each particle modifies its velocity in each generation: how much weight is given to its current velocity (intuitively the "inertial" aspect[11]), how much weight is given to $pbest$ (intuitively the "cognitive" aspect), and how much weight is given to $gbest$ (intuitively the "social" aspect). The values used in our experiments for these latter three parameters are $0.9, 2.0, 2.0$ respectively, as recommended in [11].

Initialisation of the population is also important. In common with other optimisation technology, initialisation in PSO is a balance between randomness and prior knowledge: more of the latter can accelerate the search process, but it can also limit innovation. In this work we just initialise our particles randomly.

Other aspects of PSO that must be determined on a problem-specific basis include the representation used for solutions, and the fitness function that is used to assess their performance.

B. Solution Representation

We use a straightforward representation of circuit designs. For each transistor in the circuit, we store the width of that transistor, thus a design with d transistors will be searching in d-dimensional space. The width of each transistor is restricted to the range $[1.2, 1000.0]\mu m$: the lower limit is the minimum feature size of the $1.2\mu m$ technology that we assume in our experiments, while the upper limit is a very high value chosen to allow a thorough search of the available space. We note that it isn't normally possible to manufacture transistors at every value in this range, but that the PSO algorithm and the fitness function assume that the range is continuous.

C. Fitness Calculations

The fitness of a circuit C that uses a set of transistors with widths W is given by Equation 1.

$$fitness(W) = \begin{cases} L \sum_{w \in W} w, & \text{if } t_p \leq t_{p,max} \\ A_{max} + t_p, & \text{otherwise} \end{cases} \quad (1)$$

L is the length of a transistor (in our case $1.2\mu m$), t_p is the propagation delay of C, $t_{p,max}$ is the threshold cut-off, and $A_{max} = 1000Ld$ is the maximum possible area of a circuit. Thus the fitness of a feasible solution (one that beats the threshold) is its area; and the fitness of an infeasible solution is the maximum area plus its delay.

Using this definition:

- all designs that beat the threshold are ranked better than all those that don't;
- designs that beat the threshold are ranked by their areas, as expected;
- designs that don't beat the threshold are ranked by their propagation delays, encouraging the particles to move towards feasible solutions.

We use the circuit simulation engine SPICE to calculate the propagation delay of a circuit, sourced from [12]. SPICE defines many levels at which MOSFETs can be modelled, from simple resistor/capacitor models, up to advanced models describing individual electrons and quantum effects. In this work, focused on $1.2\mu m$ technology, we use the Level 3 model, which models capacitive effects down to hundreds of picofarads. This is sufficient to expose the trade-offs which we want to optimise. We interface with SPICE through the use of MATLAB: Figure 1 illustrates this process.

D. Invalid Solutions

With complex circuits such as those discussed in Sections V-B–V-C, many points in the search space represent "invalid" solutions which SPICE is unable to simulate. The two most common reasons for this are that the iterative process used by SPICE fails to converge, or that the propagation delay is so high that the simulation does not complete in the allowed time. In either case, the solution is given an infinite fitness, and thus it will play no further part in the operation of the algorithm.

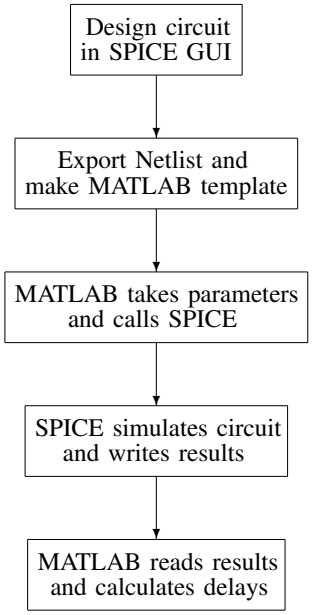


Fig. 1. The MATLAB/SPICE interface. The first two steps are performed once per circuit: the others are performed for each simulation.

The principal problem this causes is when it happens in the initialisation phase. A particle which has seen only invalid solutions has no sensible $pbest$, and if all particles have seen only invalid solutions, there will be no sensible $gbest$ either. In practice, we found that the latter problem never arose. We solved the former problem by simply re-initialising each particle until it landed on a valid solution. This approach may have implications for the algorithm's ability to explore the edges of the valid space: this is a topic for further investigation.

V. Experiments

This section describes the three circuit-types that we use to test our algorithm, and for each one it presents and discusses the results obtained.

- The inverter in Section V-A is a very simple circuit for which we can perform an exhaustive search, and which demonstrates that the PSO delivers an optimal result.
- The inverting buffer in Section V-B is the main experimental workhorse: it demonstrates that the PSO works for examples with difficult settings and constraints.
- The ripple carry adder in Section V-C is a larger circuit which demonstrates that the PSO works for more complex examples.

All experiments were run on a Windows server 2008r2 VM, with 4Gb of RAM and a 2.39GHz virtual CPU. For the two main experiments, each collection of runs with a given cut-off took about 18–20 hours.

A. *Inverter*

Figure 2 shows the inverter circuit that we use.

The inverter takes one bit as input and outputs its negation. It comprises two transistors, one P-channel with width `WP` and one N-channel with width `WN`. Therefore the problem has only two parameters, and an exhaustive search is possible. The PSO algorithm was run with twenty particles, each run taking around 11 minutes.

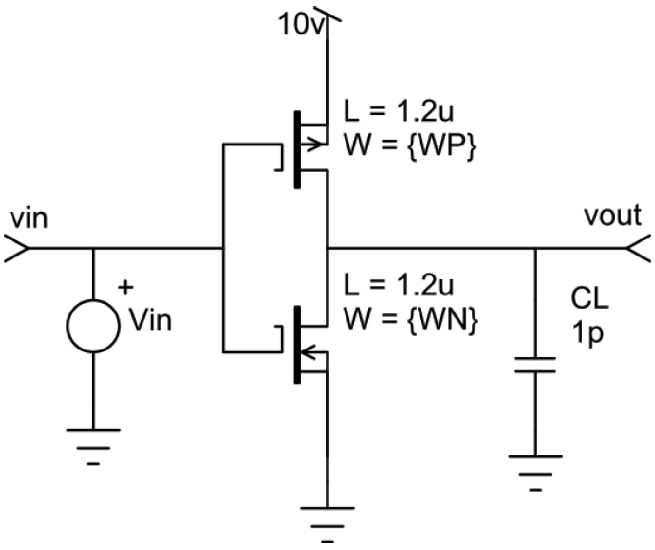


Fig. 2. The inverter. `vin` is negated to produce `vout`. Two transistors are used.

Figure 3 shows the results of an exhaustive search with a cut-off of $0.1ns$. The colours represent a "heat-map" of feasibility and performance: the background brown colour represents infeasible solutions, and the other colours represent the fitness at each feasible point in the search space, with bluer colours in the lower-left region denoting better fitness. The shape of the feasible region shows the asymmetry between the two transistors, due to the different relative permittivity of the P-type and the N-type channels.

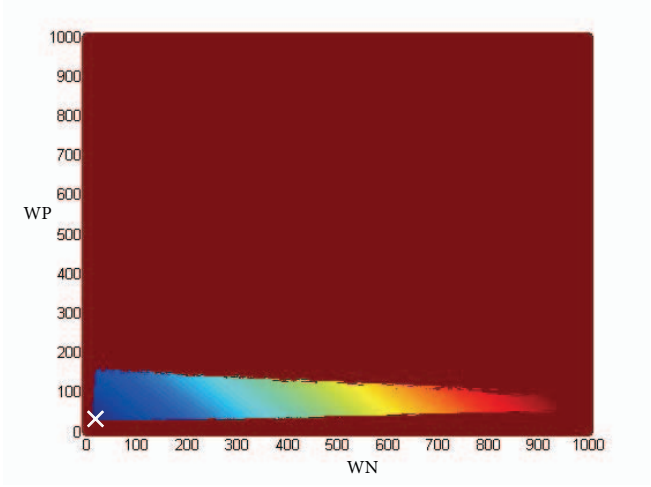


Fig. 3. The search space for the inverter, and the result of the exhaustive search. The background brown colour represents infeasible solutions, and the other (lighter) colours represent the fitness at each feasible point in the search space: bluer implies a better fitness, in the lower-left corner of the graph. The white $\times$ marks the location of the PSO result.

The exhaustive search reports an optimal fitness of $29.82\mu m^2$: the PSO algorithm locates a solution with a fitness of $29.83\mu m^2$, denoted by the white $\times$ on Figure 2. As noted above, these two likely represent the same solution when fabricated.

B. *Inverting Buffer*

Figure 4 shows the inverting buffer circuit that we use.

The buffer takes one bit as input and repeatedly inverts it. With an odd number of stages the output is the negation of the input: with an even number of stages the output is the same as the input, and it acts as a non-inverting buffer. Buffers have several uses in electronics, including imposing a fixed delay in a circuit, and passing strong current to charge large capacitances in a subsequent part of the circuit[1].

Figure 5 illustrates the timing flowing through an inverting buffer: the figure reported as the propagation delay for the entire circuit is the worst-case of its reaction time to either a change in the input signal from low to high, or from high to low.

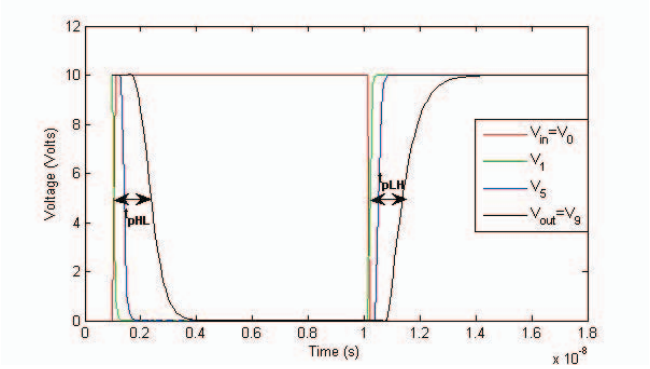


Fig. 5. Propagation of signal through the inverting buffer. The changes in `V0` denote changes in the input to the circuit, and the changes in `V1`, `V5`, and `V9` show the times at which these transistors respond. The overall delay in the circuit is given by the larger of t_{pHL} and t_{pLH}.

The buffer shown in Figure 4 has nine inverters, each with two transistors, therefore there are eighteen parameters. All experiments on the buffer were run with eighty particles: each run took 45–60 minutes.

We test our algorithm against this circuit with three different thresholds on the propagation delay.

1) A generous cut-off of $5ns$*:* Figure 6(a) shows twenty runs of the PSO algorithm with a cut-off of $5ns$, plus the best result achieved by Monte Carlo using the same total number of SPICE invocations. It is clear that PSO substantially outperforms Monte Carlo: fourteen of the twenty runs return a (roughly) optimal result of $28\mu m^2$, and even the other six achieve significantly better fitness.

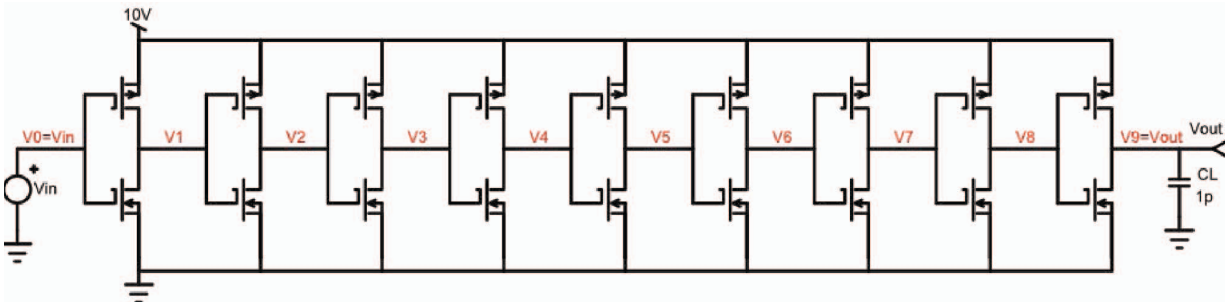
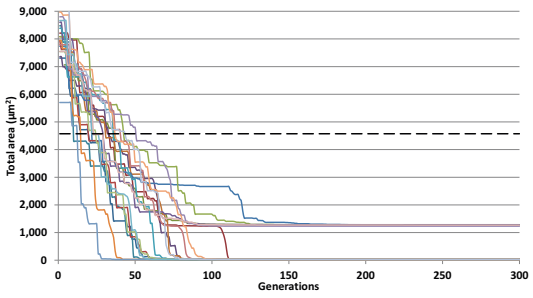


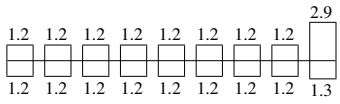
Fig. 4. The inverting buffer. `vin` is negated repeatedly to produce `vout`. Two transistors are used for each stage, thus n stages implies a total of $2n$ transistors.



(a) Twenty runs of the PSO. The dashed line indicates the best fitness achieved by Monte Carlo in 480,000 attempts.

27.979	27.990	27.993	27.995	1,268
27.980	27.990	27.993	27.995	1,275
27.990	27.991	27.994	1,232	1,278
27.990	27.992	27.995	1,265	1,284

(b) The fitness of the best solution returned in each run, in μm^2.



(c) The best design from the twenty runs. For each transistor, the top number is the width of the P-channel and the bottom number is the width of the N-channel, both in μm.

Fig. 6. Results for the inverting buffer with a cut-off of $5ns$.

Figure 6(b) shows the actual fitnesses achieved by PSO, and Figure 6(c) shows the best design returned.

2) A tighter cut-off of $2.5ns$*:* Figure 7(a) shows twenty runs of the PSO algorithm with a cut-off of $2.5ns$. Again, fourteen of the twenty runs return a (roughly) optimal result around $38.5\mu m^2$: using the same total number of SPICE invocations, Monte Carlo is unable to locate even one feasible design.

Figure 7(b) shows the actual fitnesses achieved by PSO, and Figure 7(c) shows the best design returned.

3) A very tight cut-off of $1.5ns$*:* This is a difficult constraint, specifically chosen to produce more-complex designs: it should be noted that even when optimising solely for minimum delay (ignoring area), the PSO algorithm could only get down to $1.3ns$. Once again Monte Carlo is unable to locate even one feasible solution. Figure 8(a) shows sixteen runs of the PSO algorithm with a cut-off of $1.5ns$, omitting four runs that failed to locate any feasible solutions. Of the successful runs, eight achieve good fitness and appear to still be improving after 300 generations: the eight others are probably stuck in local optima.

Figure 8(b) shows the actual fitnesses achieved by PSO, and Figure 8(c) shows the best design returned.

4) Observations: We note the following.

- For generous cut-offs, the best answer is simply for all transistors except the last to be the minimum size. The PSO finds these solutions.
- As the cut-off shrinks, the size required of the last transistor grows, until its size creates a capacitance that requires the penultimate transistor to be non-minimal. This effect can propagate backwards through the buffer, as each transistor causes its predecessor to grow.
- The fact that the algorithm sometimes falls into local optima may be partly down to the large upper limit $(1,000\mu m)$ on transistor widths. However such a large limit will be needed for some circuits, for example one that was driving a device requiring a significant current.

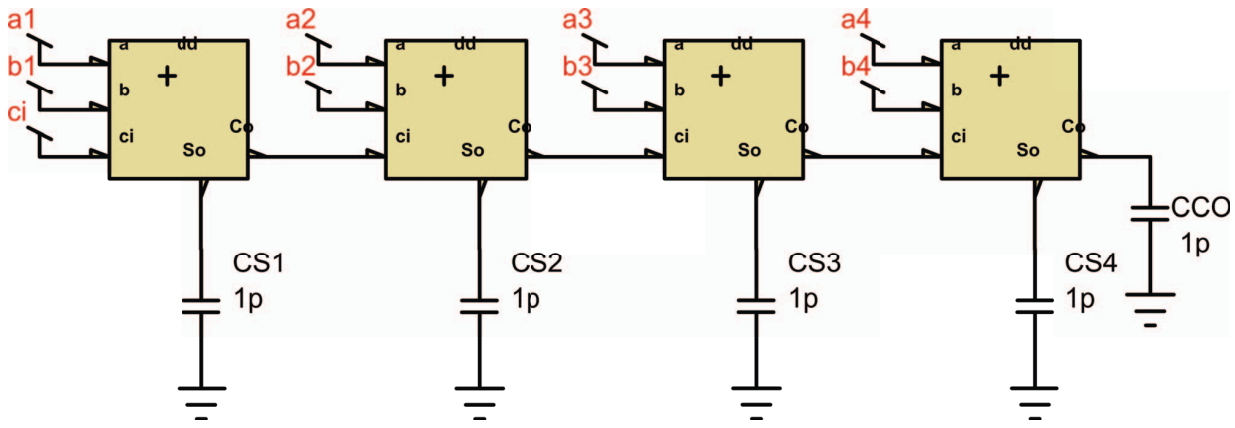


Fig. 9. The ripple carry adder. Each stage takes one bit from each summand (a_i and b_i) plus the carry bit c_i from the previous stage, and it outputs the sum bit s_i and the next carry bit c_{i+1}. 28 transistors are used for each stage, thus n stages implies a total of $28n$ transistors.

C. Ripple Carry Adder

Figure 9 shows the ripple carry adder circuit that we use.

The ripple carry adder takes two n-bit numbers and adds them by "rippling" the carry through n full adders, from the least-significant bit to the most-significant. Each full adder takes as input one bit from each summand and a carry-in, and it outputs the sum and the carry-out for the next stage. We assume that the delay for the circuit is that for propagating carries all the way to the most-significant bit. This is usually true, and is adequate for the purposes of our algorithm.

The circuit shown in Figure 9 has four adders in series, each with 28 transistors, therefore there are 112 parameters in total. All experiments on the ripple carry adder were run with forty particles: each run took 6–7 hours.

We test our algorithm against this circuit with two different thresholds on the propagation delay.

1) A generous cut-off of $4ns$: Figure 10(a) shows three runs of the PSO algorithm with a cut-off of $4ns$, plus the best result achieved by Monte Carlo using the same total number of SPICE invocations. Again it is clear that PSO outperforms Monte Carlo in all three runs, even with only forty particles: in every case it generates improved solutions in less than forty generations.

Figure 10(b) shows the actual fitnesses achieved by PSO.

2) A tighter cut-off of $2ns$: Figure 11(a) shows three runs of the PSO algorithm with a cut-off of $2ns$, plus the best result achieved by Monte Carlo using the same total number of SPICE invocations. Again PSO outperforms Monte Carlo in all three runs: it is also clear from the graphs that in both experiments the PSO is still improving in all runs.

Figure 11(b) shows the actual fitnesses achieved by PSO.

VI. Conclusions

We have described an implementation of PSO that determines the sizes of transistors required for a given integrated circuit, optimising for minimal area whilst observing hard limits on the propagation delay in the circuit. We have described experiments on three different circuit-types of varying complexity which demonstrate that the algorithm is able to derive excellent results, and in many cases is able to derive results where Monte Carlo simulation fails. The occasional inconsistency in the results derived can easily be overcome by employing multiple runs, a standard ploy with stochastic optimisation technology. These results could be used as input either to a manual circuit layout process, or to a subsequent automatic optimisation process to produce an actual circuit layout.

Future work will focus on three main areas. Firstly, we aim to combine the circuit layout problem with the sizing problem to produce a single optimisation process that generate layouts directly. Secondly, we aim to use a multi-objective approach to investigate the trade-off surface for silicon area vs. propagation delay: this may lead to a more-generally useful tool for circuit designers. Thirdly, we will investigate the application of PSO to other circuit design problems, such as power minimisation or analogue amplifier design.

References

[1] S. D. Brown and Z. G. Vranesic. *Fundamentals of Digital Logic with VHDL Design*, volume 70125910. McGraw-Hill, 2000.

[2] J. P. Fishburn and A. E. Dunlop. Tilos: A polynomial programming approach to transistor sizing. In *The Best of ICCAD*, pages 295–302. Springer, 2003.

[3] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE Int. Conf. on Neural Networks*, volume 4, pages 1942–1948, November 1995.

[4] M. Matson and L. Glasser. Macromodeling and optimization of digital MOS VLSI circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 5(4):659–678, October 1986.

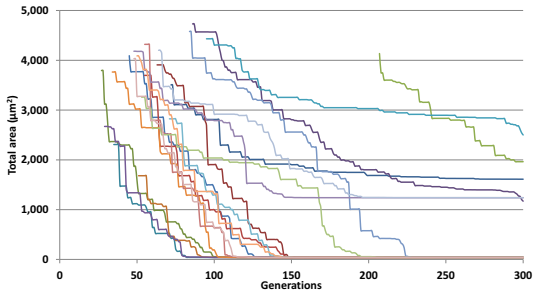
[5] C. Mooney. *Monte Carlo Simulation*. Sage Publications, 1997.

[6] L. Nagel and D. Pederson. SPICE (simulation program with integrated circuit emphasis), Memo. ERL-M382, University of California, Berkeley, April 1973.

[7] W. Nye, D. Riley, A. Sangiovanni-Vincentelli, and A. Tits. DELIGHT.SPICE: an optimization-based system for the design of integrated circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 7(4):501–519, Apr 1988.

[8] J. Rabaey. The SPICE page, *http://bwrcs.eecs.berkeley.edu/Classes/IcBook/SPICE*.

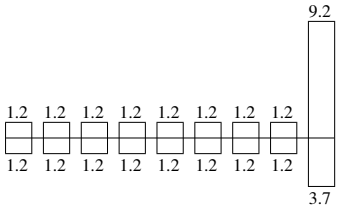
[9] J. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits*. Prentice Hall, 2002.



(a) Twenty runs of the PSO. Each line starts from the first generation when the algorithm finds a feasible solution. Monte Carlo failed to find a feasible solution in 480,000 attempts.

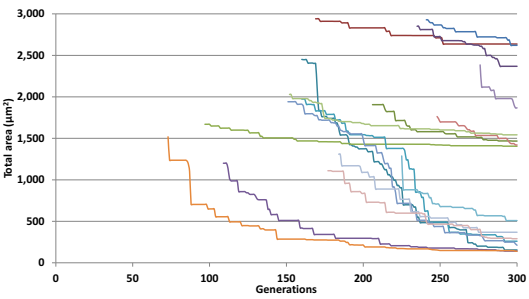
38.511	38.519	38.535	38.581	1,240
38.512	38.519	38.544	38.657	1,612
38.514	38.529	38.544	1,171	1,963
38.518	38.530	38.577	1,240	2,500

(b) The fitness of the best solution returned in each run, in μm^2.



(c) The best design from the twenty runs. For each transistor, the top number is the width of the P-channel and the bottom number is the width of the N-channel, both in μm.

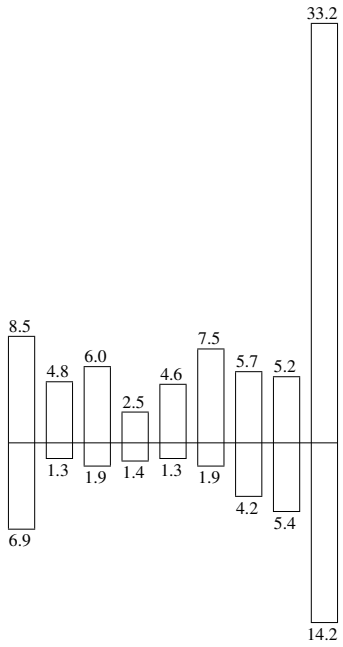
Fig. 7. Results for the inverting buffer with a cut-off of $2.5ns$.



(a) Sixteen runs of the PSO. Each line starts from the first generation when the algorithm finds a feasible solution. Four other runs failed to find a feasible solution in 300 generations. Monte Carlo failed to find a feasible solution in 480,000 attempts.

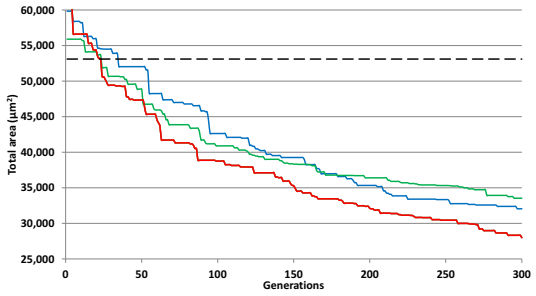
140	260	1,405	1,867	failed
142	291	1,421	2,368	failed
155	368	1,467	2,621	failed
212	509	1,542	2,636	failed

(b) The fitness of the best solution returned in each run, in μm^2.



(c) The best design from the twenty runs. For each transistor, the top number is the width of the P-channel and the bottom number is the width of the N-channel, both in μm.

Fig. 8. Results for the inverting buffer with a cut-off of $1.5ns$.

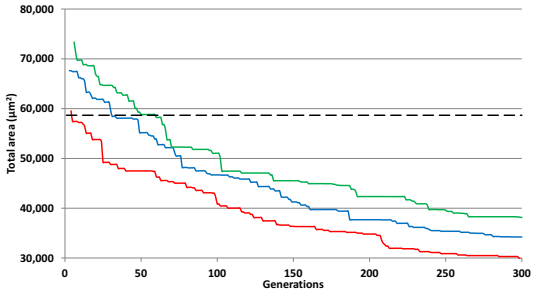


(a) Three runs of the PSO. The dashed line indicates the best fitness achieved by Monte Carlo in 36,000 attempts.

27,996	32,057	33,538

(b) The fitness of the best solution returned in each run, in μm^2.

Fig. 10. Results for the ripple carry adder with a cut-off of $4ns$.



(a) Three runs of the PSO. The dashed line indicates the best fitness achieved by Monte Carlo in 36,000 attempts.

29,707	34,228	38,161

(b) The fitness of the best solution returned in each run, in μm^2.

Fig. 11. Results for the ripple carry adder with a cut-off of $2ns$.

[10] R. Rogenmoser, H. Kaeslin, and T. Blickle. Stochastic methods for transistor size optimization of CMOS VLSI circuits. In *Parallel Problem Solving from Nature*, pages 849–858. Springer, 1996.

[11] Y. S. and R. Eberhart. A modified particle swarm optimizer. In *IEEE World Congress on Computational Intelligence*, pages 69–73, May 1998.

[12] J. Van der Spiegel. SPICE models for selected devices and components, *http://www.seas.upenn.edu/jan/spice/spice.models.html*.

[13] J. Wakerly. *Digital Design Principles and Practices*. Prentice Hall, 2005.