Logical Circuit Gate Sizing using PSO guided by Logical Effort – An Examination of the 4-Stage Half Adder Circuit

A. Johari, S. Mohamed, A. K. Halim, I. M. Yassin
Faculty of Electrical Engineering
Universiti Teknologi Mara
Shah Alam, Selangor
ihsan_yassin@salam.uitm.edu.my

H. A. Hassan
Faculty of Electrical Engineering
Universiti Industri Selangor
Bestari Jaya, Selangor
ellyza90210@yahoo.com

Abstract— Automated Complementary Metal Oxide Semiconductor (CMOS) logic circuit design leads to the reduction in costs associated with manpower and manufacturing time. Conventional methods use repetitive manual testing guided by Logical Effort (LE). LE provides an easy way to compare and select circuit topologies, choose the best number of stages for path and estimate path delay. In this paper, we propose the Particle Swarm Optimization (PSO) algorithm as a method to automate the process of CMOS circuit design by approaching the design process as an optimization problem. In our work, we choose gate widths inside the circuit as parameters to be optimized in order to achieve the target delay, and its fitness is guided by the LE method. Various parameters, such as swarm size and iterations were tested under different initialization conditions to verify PSO’s performance on a 4-stage half-adder circuit. Results have indicated that the PSO algorithm was an effective method to apply to the circuit design problem, with high convergence rates observed.

Keywords- Logical Effort, Automated circuit design, Particle swarm optimization, half-adder

I. INTRODUCTION

Logical Effort (LE) [1] is a method for fast evaluation of delays in logic paths. This method is a straightforward technique to estimate delay and to choose the best number of stages in a Complementary Metal Oxide Semiconductor (CMOS) circuits [2]. LE has since been adopted as a basis for many Computer-Aided Design (CAD) tools. It helps designer to quickly estimate the minimum possible delay in a circuit, and to choose the appropriate gate sizes to achieve this delay. However, the sizing process still requires repetitive tests by the designer to achieve the desired delay [1].

In this paper, the application of the Particle Swarm Optimization (PSO) algorithm is proposed as an alternative method for automated means for CMOS circuit design. We extend the works in [3], but apply it to a more complex circuit topology.

PSO is a population-based stochastic optimization technique based on Swarm Theory (ST) and Evolutionary Computation (EC) [4]. It is inspired by the seemingly intelligent swarming behavior of animals in nature [5]. PSO has a successful track record in solving many optimization problems [6-10]. The algorithm is simple and computationally inexpensive, yet fast and efficient [11-14].

This paper is organized as follows: Some previous works are presented in Section II. The theoretical background to the proposed approach is presented in Section III. The experimental setup is presented in Section IV, and the results and discussions are presented in Section V. Finally, Section VI presents the conclusions.

II. PREVIOUS WORKS

A. Logical Effort (LE)

Circuit delay is an important factor to consider in digital circuit design. Failure to consider the delay may lead to circuit malfunctions and non-compliance to timing specifications. Such failures are called delay faults [15].

The LE method [1] has been introduced to allow for both quick and accurate analysis of digital circuit design. LE defines the overall circuit delay in concrete mathematical terms, combined to a linear Resistance-Capacitance (RC) delay mode [1]. This method can be used for comparative analysis of existing circuits, as well as to guide transistor sizing and estimate the optimal number of stages in the circuit.

B. Adders and previous works on adders

Fast adders are widely used in CMOS circuits as part of the Arithmetic-Logic Unit (ALU). Various types of adders have been designed: half-adder, full-adder and ripple-carry adder. All adders offer different tradeoffs between delay, area, and power consumption. Analytical delay models help designers evaluate these tradeoffs, but simply counting logic levels is inadequate because circuit delay also depends on fan-out [16].

Half-adders are the basic building blocks for more complex adders, such as full-adders and ripple-carry adders. Previous works on adders design involves increasing its speed by...
adjusting its topologies [17], gate sizing and routing [16] and others.

C. PSO

PSO is a population-based stochastic optimization technique inspired by animal swarming behavior in nature [6]. The exploration of the solution space is performed by simple agents called particles. The solution is achieved by the competitive and cooperative behavior of agents.

The original PSO algorithm generates solutions in continuous-form. The algorithm is simple and computationally inexpensive, yet fast and efficient [5, 12-14].

PSO has a successful track record in solving many optimization problems [18]. Since PSO agents (particles) are parallel in nature, PSO allows efficient and fast optimization of the problem [19]. Furthermore, PSO requires only basic mathematical operators (plus, minus and multiply) to perform optimization [5]. It is memory-inexpensive [14], as only a few parameters need to be stored to update future iterations. Another benefit of PSO is that it requires constant computational and memory costs for each iteration [12].

III. THEORETICAL BACKGROUND

A. Implementation of Logic Gates at Transistor Level

Transistors form the basis of any logic gate. In CMOS, these transistors can be arranged in ways to perform a desired logic function. A list of several logic gate implementations is shown in Table I.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Transistor Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inverter</td>
<td>![Inverter Diagram]</td>
</tr>
<tr>
<td>Buffer</td>
<td>![Buffer Diagram]</td>
</tr>
<tr>
<td>NAND</td>
<td>![NAND Diagram]</td>
</tr>
<tr>
<td>NOR</td>
<td>![NOR Diagram]</td>
</tr>
</tbody>
</table>

Transistors are annotated with widths measured in arbitrary units so that each pull down stack has unit effective resistance [16]. The size of these transistors affects the delay characteristics of the logic gate. When the gates are combined in a logic circuit, these transistors must be sized accordingly to achieve the desired circuit delay.

B. The Half Adder Logic Circuit

A half adder is a logic circuit that performs 1-bit addition, and is implemented using the logic gates described in Table I.

A half adder takes two inputs, A and B and produces two outputs sum and carry. An implementation of the half adder is shown in Fig. 1 [20].

C. LE

The logical effort LE is the ratio of the input capacitance of the gate input to the input capacitance (3 units) of an inverter with the same unit effective resistance [16]. Table II lists the logical effort and parasitic delay of each gate in the half adder circuit. The parasitic delay (p) is estimated by counting the total transistor width on the output node, assuming diffusion and gate capacitance are approximately equal [16].

<table>
<thead>
<tr>
<th>Gate</th>
<th># of Input</th>
<th>LE</th>
<th>PD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inverter</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Buffer</td>
<td>1</td>
<td>4/3</td>
<td>2</td>
</tr>
<tr>
<td>NAND</td>
<td>2</td>
<td>4/3</td>
<td>2</td>
</tr>
<tr>
<td>NOR</td>
<td>2</td>
<td>4/3</td>
<td>2</td>
</tr>
</tbody>
</table>

The single stage delay, \( d \), is measured by adding the stage effort (\( f \)) and \( p \) together:

\[
d = f + p,
\]

\[
f = gh
\]

\[
h = C_{out}/C_{in}
\]

where \( g \) and \( h \) are the logical and electrical effort of the gates. Eq. (1) can be extrapolated to cater for multi-stage circuits. In the case of multi-stage circuits, \( f \) must be multiplied by the branching effort, \( b \):

\[
b = (C_{on} + C_{off})/C_{on}
\]

where \( C_{on} \) is the capacitance in the critical path, and \( C_{off} \) is the capacitance which doesn’t lay in the critical path. The branching effort (\( b \)) is important in multistage design, because
it allows us to model a branch in a given network as its single-path equivalent. Eq. (1) thus becomes:

\[ D = GBH + P \]  

(3)

where the uppercase letters denote multi-stage application. To find the minimum path delay, \( F_{min} \), the effort for each stage must be equal. This is done by setting \( h \) to \( h_{min} \). Rearranging Eq. (1) produces:

\[ C_{in} = \frac{6C_{out}}{F_{min}} \]  

(4)

Once \( C_{in} \) is known, this value can be distributed among transistors in the stage.

D. The PSO algorithm

PSO is a population-based stochastic optimization algorithm inspired by animal swarming behavior in nature [6]. PSO iteratively searches for solutions in the problem space by taking advantage of the cooperative and competitive behavior of simple agents called particles.

The constriction factor variant of the PSO (PSO CF) algorithm was used in this paper due to its superior convergence properties. PSO CF improves particle convergence by gradually decreasing particle velocities as iterations progress, so that particle movements near the optimum are localized.

The PSOCF algorithm search is directed by its velocity (\( V_{id} \)) which modifies the particle’s position (\( X_{id} \)):

\[ V_{id} = \chi[V_{id} + C_1(pBest - X_{id}) \times \text{rand}_1 - C_2(gBest - X_{id}) \times \text{rand}_2] \]  

(5)

\[ X_{id} = X_{id} + V_{id} \]  

(6)

\( V_{id} \) = particle velocity.
\( X_{id} \) = particle position.
\( pBest \) = particle’s best fitness so far.
\( gBest \) = best solution achieved by the swarm so far.
\( C_1 \) = cognition learning rate
\( C_2 \) = social learning rate.
\( \text{rand}_1 \) and \( \text{rand}_2 \) = uniformly distributed random numbers between 0 and 1.
\( \chi \) = constant set according to \( C_1 \) and \( C_2 \).

E. Modifications on PSO CF to perform delay optimization

Works by [3, 4] have used a scaling equation to rescale the particle values in Eq. (2) with significant success in optimization of discrete variables. In this paper, a similar method was used. However, since the optimization was done in continuous form, the scaling equation in [3, 4] was modified by removing the rounding operator, which produces Eq. (7). Eq. (7) was originally used in [21] to transform neural network datasets to between a predefined range:

\[ y = \left( \frac{y_{max} - y_{min}}{r_{max} - r_{min}} \right) + y_{min} \]  

(7)

where:
\( y \) = particle position.
\( y_{min} \) = lowest integer value position.
\( y_{max} \) = highest integer value position.
\( r_{min} \) = lowest range for continuous-valued solution.
\( r_{max} \) = upper range for continuous-valued solution.

To use this equation, the continuous-valued solution should be in the range of \([r_{min}, r_{max}]\). The pseudo-code for the PSO CF is shown in Fig. 2.

For each particle:
1) Do a velocity update as directed by Eq. (5).
2) Modify the particle position according to Eq. (6).
3) Check the particle value, \( x_{id} \):
   a) If the value is more than \( x_{Max} \), then pull the value of \( x_{id} \) back to \( x_{Max} \), and set \( vid \) to 0 to stop searching in that direction.
   b) If the value is less than \( x_{Min} \), then pull the value of \( x_{id} \) back to \( x_{Min} \), and set \( vid \) to 0 to stop searching in that direction.
4) Rescale the solution using Eq. (7) and evaluate the fitness function.
5) Compare particle’s fitness with \( gBest \):
   a) If the fitness is better than \( gBest \), then do not update \( gBest \).
   b) If the fitness is worse than \( gBest \), then update \( gBest \) with the new fitness value.
6) Resume optimization for next particle at 1).
7) Finish optimization either when the maximum iterations reached, or when the objective has been met.

IV. METHODOLOGY

A. Description of Experiments

For the problem presented, several experiments were performed. For all experiments, the initial swarm was uniformly distributed inside \([0.2, 5.5]^D\), where \( D \) is 4, the corresponding dimension for the number of gate widths to be optimized. The swarm sizes tested is shown in Table III.

For each swarm size, the experiments were repeated 5 times (using different random initialization values generated using the Mersenne-Twister Algorithm (MTA). The initial MTA state (for the first repetition) starts from 0 and increased by 50,000 for next repetitions. This was done to measure the convergence with different initial particle values, as well as to ensure repeatability of the experiments.
### TABLE III. Swarm Sizes Used for PSO Optimization

<table>
<thead>
<tr>
<th>Swarm size</th>
<th>Max. Iteration</th>
<th>Interval [ymin, ymax]</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10, 15, 20, 25, 30</td>
<td>[0.2, +5.5]</td>
</tr>
<tr>
<td>10</td>
<td>10, 15, 20, 25, 30</td>
<td>[0.2, +5.5]</td>
</tr>
<tr>
<td>15</td>
<td>10, 15, 20, 25, 30</td>
<td>[0.2, +5.5]</td>
</tr>
<tr>
<td>20</td>
<td>10, 15, 20, 25, 30</td>
<td>[0.2, +5.5]</td>
</tr>
<tr>
<td>25</td>
<td>10, 15, 20, 25, 30</td>
<td>[0.2, +5.5]</td>
</tr>
<tr>
<td>30</td>
<td>10, 15, 20, 25, 30</td>
<td>[0.2, +5.5]</td>
</tr>
</tbody>
</table>

### B. PSO Parameter Setting

For the PSO algorithm, the constriction factor method was used. The values of $C_1$ and $C_2$ were both set to 2.05 [22]. The minimum and maximum values of $X_{id}$ were set to 0 and 1, respectively, to constrain the particle values between 0 and 1. The dynamic range of $V_{id}$ was set to between -1 (when $X_{id}$ moves from 1 to 0) and +1 (when $X_{id}$ moves from 0 to 1). The process of LE optimization can be described with a flow chart in Fig. 2.

### C. Fitness function

The optimization objective was set to $18.4031 \tau$ based on electrical effort where the input capacitance, $C_{in}$, and output capacitance, $C_{out}$, were 1 and 16, respectively. However, this delay value can be set according to the requirements of the circuit design. The delays for several types of $H$ are shown in Table IV.

### TABLE IV. Delay of 4-Stage Half Adder

<table>
<thead>
<tr>
<th>Electrical Effort</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>18.4031\tau</td>
</tr>
<tr>
<td>8</td>
<td>16.4297\tau</td>
</tr>
<tr>
<td>4</td>
<td>14.7703\tau</td>
</tr>
<tr>
<td>2</td>
<td>13.3749\tau</td>
</tr>
<tr>
<td>1</td>
<td>12.2015\tau</td>
</tr>
</tbody>
</table>

During optimization, the gate widths were set according to PSO values, and the corresponding delay is calculated. The difference between the calculated delay ($d_{PSO}$) and the desired delay ($d_{desired}$) was determined as:

$$fitness = |d_{desired} - d_{PSO}|$$

A low fitness value indicates a good solution (as the fitness minimizes the difference between the desired delay and the delay calculated using PSO), and vice versa. This fitness value was then fed back to the PSO algorithm for it to optimize its search in the following iterations.

### V. RESULTS & DISCUSSIONS

The convergence of the PSO algorithm was tested for different swarm sizes, iterations and initial seeds. For each swarm size, the experiment was repeated 5 times with different initial particle values generated using MTA. If PSO managed to achieve the optimal the optimization is considered a success. Else, the optimization is considered a failure.

From the half adder circuit in Fig. 1, the path delay was first minimized with maximum electrical effort in Table. Then we used this minimum delay as our timing constraint for minimizing area. The best convergences plots of the half adder delay by using PSO algorithm are shown in Fig. 4 and summarized in Table V.

### TABLE V. Summary of Convergence Plot

<table>
<thead>
<tr>
<th>Result</th>
<th>Particles</th>
<th>Seed</th>
<th>Iteration</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best 1</td>
<td>30</td>
<td>100000</td>
<td>20</td>
<td>4.81 x 10^{-05}</td>
</tr>
<tr>
<td>Best 2</td>
<td>30</td>
<td>100000</td>
<td>25</td>
<td>4.81 x 10^{-05}</td>
</tr>
<tr>
<td>Best 3</td>
<td>30</td>
<td>100000</td>
<td>30</td>
<td>4.81 x 10^{-05}</td>
</tr>
<tr>
<td>Best 4</td>
<td>30</td>
<td>150000</td>
<td>15</td>
<td>5.94 x 10^{-05}</td>
</tr>
<tr>
<td>Best 5</td>
<td>30</td>
<td>150000</td>
<td>20</td>
<td>5.94 x 10^{-05}</td>
</tr>
</tbody>
</table>

From Table V, it can be seen that the swarm size of 30, 20 iterations at seeds equal to 100,000 gave the optimum delay since it gave the lowest difference between the PSO-optimized delay and the objective delay which was $4.81 \times 10^{-05} \tau$. The convergence of PSO with different swarm sizes was relatively consistent, indicating that small swarm sizes were sufficient to approximate the solution to the given problem.

After optimization, all trials converged to $18.4031 \tau$, which...
was the optimal fitness value. All trials managed to find the optimal gate sizes for all logic gates in the full adder path. This result indicates that the proposed method has successfully been implemented to solve the test problem, even with different initialization values.

The gate widths, without considering the wire capacitance for the best five test cases are shown in Table VI. As shown in the table, the continuous sizing area is expressed in terms of capacitance. As expected, the area and delay results are almost the same. The PSO algorithms managed to produce consistent results (indicated by similar gate widths found even with different seeds, swarm sizes and iterations).

<table>
<thead>
<tr>
<th>Gate</th>
<th>Delay</th>
<th>C/G</th>
<th>Delay</th>
<th>C/G</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nand1</td>
<td>0.6154</td>
<td>0.6111</td>
<td>0.6083</td>
<td>2.1733</td>
<td>4.9608</td>
</tr>
<tr>
<td>Buf1</td>
<td>1.4311</td>
<td>1.8711</td>
<td>3.6773</td>
<td>2.4511</td>
<td>1.8979</td>
</tr>
<tr>
<td>Nand2</td>
<td>2.2188</td>
<td>2.7868</td>
<td>4.2704</td>
<td>5.2411</td>
<td>3.1827</td>
</tr>
<tr>
<td>Buf2</td>
<td>5.1611</td>
<td>4.1008</td>
<td>4.7451</td>
<td>4.3722</td>
<td>2.3422</td>
</tr>
<tr>
<td>Total</td>
<td>18.4031</td>
<td>18.4031</td>
<td>18.4031</td>
<td>18.4032</td>
<td></td>
</tr>
</tbody>
</table>

VI. CONCLUSIONS

A novel PSO method by [3, 4] was extended to find the gate widths of a 4-stage Half Adder circuit. The method used the PSO algorithm to automatically find the gate widths for the full adder logic gates based on a predefined delay. Experiments showed that the PSO method was capable to solve the LE problem well, even with increased complexity of the test circuit, with consistent results throughout different swarm sizes, iterations and initial conditions.

REFERENCES