SOME EXPERIMENTS IN VLSI LEAF-CELL OPTIMIZATION

PRINCETON UNIV NJ DEPT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE K IWANO ET AL. 1984 ARO-18985.25-EL

UNCLASSIFIED DAAG29-82-K-0095

F/G 9/5 NL
**REPORT DOCUMENTATION PAGE**

1. **REPORT NUMBER**
   - ARO 18985.25-EL

2. **GOVT ACCESSION NO.**
   - N/A

3. **RECIPIENT'S CATALOG NUMBER**
   - N/A

4. **TITLE (and Subtitle)**
   - Some Experiments in VLSI Leaf-Cell Optimization

5. **TYPE OF REPORT & PERIOD COVERED**
   - Reprint

6. **PERFORMING ORG. REPORT NUMBER**

7. **AUTHOR(s)**
   - Kazuo Iwano, Kenneth Steiglitz

8. **CONTRACT OR GRANT NUMBER(s)**
   - DAAG29-82-K-0095

9. **PERFORMING ORGANIZATION NAME AND ADDRESS**
   - Princeton University

10. **CONTROLLING OFFICE NAME AND ADDRESS**
    - U.S. Army Research Office
    - Post Office Box 12211
    - Research Triangle Park, NC 27709

11. **MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)**

12. **REPORT DATE**

13. **NUMBER OF PAGES**

14. **UNCLASSIFIED/DECLASSIFIED/DEGRADED COPY**

15. **SECURITY CLASS. (of this report)**
   - Unclassified

16. **DISTRIBUTION STATEMENT (of this report)**
   - Approved for public release; distribution unlimited.

17. **DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)**
   - NA

18. **SUPPLEMENTARY NOTES**
   - The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official position, policy, or decision, unless so designated by other documentation.

19. **KEY WORDS (Continue on reverse side if necessary and identify by block number)**
   - Very Large Scale Integration
   - Arrays
   - Cells
   - Optimization
   - Logic Circuits
   - Delay

20. **ABSTRACT (Continue on reverse side if necessary and identify by block number)**
Abstract

This paper describes a method for local optimization of VLSI leaf cells, using the parameterized procedural layout language ALLENDE [5]. Tradeoffs among delay time, power consumption, and area are illustrated. Three different implementations of the 1-bit full adder are compared: a random logic circuit, a data selector, and a PLA. The fastest random logic 1-bit full adder has a time-power product about 1/3 that of the fastest data selector, and about 1/4 that of the fastest PLA. The 4-bit parallel adder is used to illustrate the effect of loading when leaf cells are combined.

1. Introduction

In the design of a custom VLSI chips it often happens that there is one cell that is used many times, usually in an array or a recursive structure. The fact that a cell is used many times means that there is a large potential payoff in its optimization, and that the problem can be made small enough to be manageable. Arrays of cells are especially common in digital signal processing applications, where regular structures, like systolic arrays, lead to designs that are easy to lay out efficiently, and have high throughput. As examples, bit-parallel and bit-serial multipliers can be constructed from one- and two-dimensional arrays of one-bit full adders, as can a wide variety of pipelined FIR and IIR filters (see [1], for example). As another example, a processor for updating one-dimensional cellular automata has been designed at Princeton which consists of a one-dimensional array of 5-input/1-output PLA's [10]. In such cases the problem of making most efficient use of a given piece of silicon breaks down into two distinct problems: 1) choice of the global packing strategy (the method of laying out and interconnecting leaf cells, and connecting them to power and clocks), and 2) the design of the iterated structure itself (which we call the leaf cell). In this paper we study the second problem: the design of efficient leaf cells. The example used throughout is the most common in digital signal processing, the 1-bit full adder.

There are three important measures of how good a leaf cell is: its time delay $T$; its peak or average power dissipation $P_{\text{max}}$ or $P_{\text{ave}}$; and its area $A$. 

This work was supported by National Science Foundation Grants ECS-8307965, U.S. Army Army Research Office, Durham, NC, under Grant DAAG29-82-K-0085, DARPA Contract N00014-82-K-0549, and ONR Grant N00014-83-K-0275.
Ideally, the designer should be able to trade off these measures, one against the other. For example, in one application the clock may be fixed at a known value $T_0$, and it would therefore be senseless to make the cell faster. On the other hand, peak power may be a real constraint because of heat dissipation limitations, and at the same time it may be important to keep the area small so as to fit as many cells on one chip as possible. We might therefore try to minimize some measure of the peak power and area (the product, for example), while enforcing the constraint $T \leq T_0$. In other applications speed may be critical, and it may be important to minimize $T$ while observing constraints on $P_p$ and $A$, and so on. In general, we would like to have enough information about the tradeoffs among the measures $T$, $P$ and $A$ to make intelligent design decisions. As we will see, the $P-T$ tradeoff is often of most interest, since the area is often a less sensitive function of design parameters (at least for fixed topology).

2. Formulation

The basic approach we take will be to search for local improvements on random initial designs. The search strategy will be to consider all single or double changes in element size along the critical path. When only single changes are tried, we call the procedure "1-change", when double changes are tried, "2-change". The idea is that the critical path indicates which parameters are most important to performance at any given point in the analysis.

We will limit the optimization to choice of pulldown widths. The method can be extended to choice of layers, orientation, and topologies. We will, however, study three radically different topologies for the full adder: the PLA, data-selector, and random logic.

The main analysis tools used in these experiments are the timing simulator CRYSTAL, and the power-estimation program POWEST, together with the rest of the Berkeley tool package.

Another essential component of the work is a procedural, constraint-based layout language for specifying VLSI layouts; in this case, we used the new language ALLENDE being developed at Princeton, a successor to ALG and CLAY. This allows us to specify circuit parameters and have a cifplot generated automatically.

3. The Critical-Path Optimization

Figure 1 shows how the optimization is performed in our experiments. In Figure 1 $faparm$ is an input parameter vector to ANALYSIS which has diffusion widths of nodes as described in section 4. The initial $faparm$ is generated at random by RANDOM according to its input file pattern. ANALYSIS takes $faparm$ as its input and generates an appropriate layout and its resulting $T$, $P$, and $A$ as well as the nodes on the critical path (hereafter called the critical path nodes). Since every node on the critical path has an associated parameter in $faparm$, CASEGEN can generate $faparm$ as subcases by using the one-(two-)change method. Here the one-(two-)change method changes one(two) parameter(s) associated with the critical path nodes by one step. (From here on the 1-change method is denoted by 1-opt or Random 1-opt, and the 2-change method by 2-opt or Random 2-opt.)

The optimization strategy is shown in the flowchart of figure 2. When the first improvement occurs, this case is picked up for the next iteration. If no improvement occurs but there exists a case which has the same cost and has
not yet been analyzed, this case is adopted next. Otherwise a new random
faparm is generated for the next iteration, to search for other locally optimal
points. We used two cost criteria for optimization: $T$, and $P_{\text{max}}T$ (hereafter
denoted by $PT$). Figure 3 shows an outline of the main procedures used in the
ANALYSIS loop. A short description of each follows below:

1) **ALLENDE** This procedural constraint-based VLSI layout language pro-
duces an integrated circuit layout in Caltech International Form (CIF) corresponding to the specified parameters [5].

2) **MEXTRA** MEXTRA reads CIF and extracts the nodes to create a circuit
description for further analyses [2].

3) **CRYSTAL** CRYSTAL is used for finding the worst-case delay time of the
circuit [2].

4) **POWEST** POWEST is used for finding the average and maximum power
consumption of the circuit.

5) **CRITICAL** CRITICAL reports the critical path nodes by using the output
of CRYSTAL.

6) **LIST** This command stores the vector of results ($T, P, A$) in the HIST-
ORY file for further optimization.

In figure 3 the squares surrounded by dotted lines are files used for inputs
or outputs of the above procedures.

1) **faparm** The faparm has parameters for layout genera-
tion: for example, the diffusion width of each
node, the permutation of product terms in a
PLA, etc.

2) **layout generating program** There are several ALLENDE programs imple-
menting desired circuit topologies such as the
PLA, random logic, etc. Each program requires
parameters in its corresponding faparm.

3) **the critical path nodes** The critical path nodes are extracted from the
output of CRYSTAL. Each node can be associ-
ated with parameters in faparm. This is done by
looking up a table for each topology, which
associates each node with its corresponding
parameter.

4. **Full-Adder Circuit Implementations**

As mentioned in the Introduction, we adopted the 1-bit full-adder circuit as
an example for experimentation, because it is relatively simple, but is a basic
arithmetic logic circuit. The 1-bit full-adder circuit can be implemented in
many ways. We chose three kinds of circuits: the PLA, Data Selector, and Ran-
dom logic. Each layout has several parameters. We will use the vector represen-
tation of these parameters: that is $d = (d_1, d_2, \ldots, d_n)$ means that the
diffusion width of node $i$ is $d_i$. We also use the vector $k = (k_1, k_2, \ldots, k_n)$ to
mean that the pullup to pulldown ratio of the inverter, NOR, or NAND circuit in
which node $i$ exists is $k_i$. The vector $k$ is fixed for each circuit.

1) **PLA**

-3-
Figure 4 shows the full-adder circuit diagram implemented by a programmable logic array (PLA) [7]. This layout has the following 17 parameters and 2 permutations.

\[ d = ( d_{\text{and}_1}, \ldots, d_{\text{and}_p}, d_{\text{or}_1}, \ldots, d_{\text{or}_q}, d_{\text{in}_1}, \ldots, d_{\text{in}_s}, d_{\text{out}_1}, d_{\text{out}_p}, \pi_1, \pi_2 ) \]

\[ k = ( 4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4 ) \]

- 7 pulldown diffusion widths of the AND plane.
- 2 pulldown diffusion widths of the OR plane.
- 6 pulldown diffusion widths for inputs.
- 2 pulldown diffusion widths for outputs.
- 1 permutation of product terms in the AND plane.
- 1 permutation of outputs.

In the optimization process, the two permutations are fixed for the sake of simplicity. However those two permutations are chosen in advance in order to give the best result before the optimization by doing experiments based on various random permutations as inputs.

2) Berkeley PLA

The PLA generated by using mkpla of the Berkeley VLSI tools [2,8] is used for the purpose of cost comparison with the PLA implemented in 1). This PLA is not optimized, but uses the following fixed parameter vector.

\[ d = ( 4,4,4,4,4,4,4,4,8,8,8,8,8,8,8,8 ) \]

\[ k = ( 4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4 ) \]

3) Data Selector

Figure 5 shows the full-adder circuit diagram of a Data Selector implementation [9]. The following truth table is used.

<table>
<thead>
<tr>
<th>( C_i )</th>
<th>B</th>
<th>S</th>
<th>( C_o )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>A</td>
<td>( C_i ) (or ( B ))</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>( \overline{A} )</td>
<td>( A )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>( \overline{A} )</td>
<td>( A )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>A</td>
<td>( C_i ) (or ( B ))</td>
</tr>
</tbody>
</table>

This circuit selects inputs \( (A, \overline{A}, \text{or } C_i) \) instead of calculating \( S \) and \( C_o \). Here \( C_i \) is the input carry signal, \( C_o \) is the output carry signal, and \( S \) is the output sum signal. \( A \) and \( B \) denote the two other inputs. This layout has the following 6 parameters.

\[ d = ( d_A, d_B, d_{C_i}, d_1, d_2, d_3, d_{C_o}, d_S ) \]

\[ k = ( 4,4,4,4,8,8,8,8 ) \]

- 3 pulldown diffusion widths for input inverters.
- 3 pulldown diffusion widths for internal inverters.
- 2 pulldown diffusion widths for output inverters.

4) Random Logic

Figure 6 shows the circuit diagram of the Random Logic Implementation [6].
This layout has the following 4 parameters.
\[ d = (d_1, d_2, d_3, d_4) \]
\[ k = (8, 12, 4, 4) \]
- 2 pulldown diffusion widths for internal inverters.
- 2 pulldown diffusion widths for output inverters.

All the circuits above were verified by ESIM [2] or SIMULATE [5].

5. Parameterization

The diffusion width of the pullup in each stage is automatically determined and implemented by ALLENDE in the following way. Suppose that the current parameter vector is \( d = (d_1, d_2, \ldots, d_n) \), and the pullup-to-pulldown ratio vector of the specified layout is \( k = (k_1, k_2, \ldots, k_n) \). (The choice of pullup-to-pulldown ratio is discussed in [7].) For each node \( i \), define the variables \( Z_{pu}, Z_{pd}, \) and a pullup-to-pulldown ratio \( K \) as follows.

\[
Z_{pu} = \frac{L_{pu}}{W_{pu}}, \quad Z_{pd} = \frac{L_{pd}}{W_{pd}}, \quad K = \frac{Z_{pu}}{Z_{pd}}
\]

where
- \( L_{pu} (L_{pd}) \) is the length of pullup (pulldown).
- \( W_{pu} (W_{pd}) \) is the width of pullup (pulldown).
- \( W_{pd} = d_i, \quad K = k_i \) and \( L_{pd} = 2 \).

\( L_{pu} \) and \( W_{pu} \) are determined as follows.

If \( W_{pd} \leq 2K \)

\[
W_{pu} = 2 \quad \text{or} \quad L_{pu} = \frac{4K}{W_{pd}}.
\]

If \( W_{pd} > 2K \)

\[
W_{pu} = \frac{W_{pd}}{K} \quad \text{or} \quad L_{pu} = \frac{2KW_{pu}}{W_{pd}}.
\]

We adopted following choices.

1) \( \lambda = 2 \mu \)

2) The timing estimation program CRYSTAL uses an input pulse which is 1 nsec wide.

6. Results

Table 1 shows a comparison of the performance of our implementations. Each row represents one locally optimal point using as criterion the item indicated by *. The units of \( A, P_{ave}, P_{max}, T, APT \) and \( PT \) are \( \lambda^2, (10^{-6} \cdot W), (10^{-6} \cdot W), ns, (10^{-12} \cdot \lambda^2 \cdot W \cdot ns) \) and \( (10^{-6} \cdot W \cdot ns) \) respectively in all tables. Figure 7 shows \( P_{max} \) vs \( T \) curves for different topologies, while figure 8 shows several \( P_{max} \) vs \( T \) trajectories obtained during the process of optimization using the 1-change and 2-change methods for the Data Selector and the Random
### Table 1. Performance Comparison (1 Bit Full Adder)

<table>
<thead>
<tr>
<th>type</th>
<th>$A$</th>
<th>$P_{\text{ave}}$</th>
<th>$P_{\text{max}}$</th>
<th>$T$</th>
<th>$\text{APT}$</th>
<th>$PT$</th>
<th>parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLA</td>
<td>21560</td>
<td>6472</td>
<td>10183</td>
<td>12.8*</td>
<td>2802</td>
<td>1303</td>
<td>1)</td>
</tr>
<tr>
<td></td>
<td>21840</td>
<td>5678</td>
<td>9241</td>
<td>15.3*</td>
<td>3087</td>
<td>1413</td>
<td>2)</td>
</tr>
<tr>
<td></td>
<td>21762</td>
<td>5503</td>
<td>8616</td>
<td>14.9*</td>
<td>2794</td>
<td>1284</td>
<td>3)</td>
</tr>
<tr>
<td>PLA (Berkeley)</td>
<td>22176</td>
<td>7314</td>
<td>11749</td>
<td>12.8*</td>
<td>3339</td>
<td>1504</td>
<td>4)</td>
</tr>
<tr>
<td>Data Selector</td>
<td>8100</td>
<td>3765</td>
<td>6117</td>
<td>15.8*</td>
<td>763</td>
<td>988</td>
<td>8 8 8 8 8 8 8 8</td>
</tr>
<tr>
<td></td>
<td>8100</td>
<td>3529</td>
<td>5645</td>
<td>16.5*</td>
<td>754</td>
<td>931</td>
<td>8 6 8 4 8 8 8 8</td>
</tr>
<tr>
<td></td>
<td>8190</td>
<td>3764</td>
<td>6116</td>
<td>15.9*</td>
<td>796</td>
<td>972</td>
<td>12 8 8 8 8 8 8 8</td>
</tr>
<tr>
<td>Random Logic</td>
<td>7742</td>
<td>1331</td>
<td>1957</td>
<td>16.5*</td>
<td>392</td>
<td>323</td>
<td>16 12 3 2</td>
</tr>
<tr>
<td></td>
<td>9600</td>
<td>1683</td>
<td>2427</td>
<td>16.4*</td>
<td>382</td>
<td>396</td>
<td>16 24 2 3</td>
</tr>
<tr>
<td></td>
<td>9600</td>
<td>1644</td>
<td>2329</td>
<td>16.4*</td>
<td>376</td>
<td>362</td>
<td>16 24 2 2</td>
</tr>
<tr>
<td></td>
<td>9600</td>
<td>1723</td>
<td>2508</td>
<td>16.5*</td>
<td>397</td>
<td>413</td>
<td>16 24 3 3</td>
</tr>
<tr>
<td></td>
<td>5194</td>
<td>705</td>
<td>1096</td>
<td>22.8</td>
<td>128</td>
<td>248*</td>
<td>8 8 2 2</td>
</tr>
<tr>
<td></td>
<td>4704</td>
<td>626</td>
<td>1018</td>
<td>25.9</td>
<td>124</td>
<td>264*</td>
<td>4 6 3 2</td>
</tr>
<tr>
<td></td>
<td>5136</td>
<td>744</td>
<td>1174</td>
<td>22.9</td>
<td>138</td>
<td>269*</td>
<td>8 8 2 3</td>
</tr>
</tbody>
</table>

1) $d = (4, 4, 4, 4, 4, 4, 3, 4, 8, 8, 8, 4, 4, 4, 8, 2)$  
2) $d = (4, 2, 3, 3, 3, 3, 4, 3, 8, 8, 4, 4, 8, 2)$  
3) $d = (3, 3, 3, 3, 3, 3, 4, 3, 3, 8, 8, 4, 4, 4, 3)$  
4) $d = (4, 4, 4, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8)$

### Table 2. Performance Comparison (4 Bit Parallel Adder)

<table>
<thead>
<tr>
<th>type</th>
<th>$A$</th>
<th>$P_{\text{ave}}$</th>
<th>$P_{\text{max}}$</th>
<th>$T$</th>
<th>$\text{APT}$</th>
<th>$PT$</th>
<th>parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Selector</td>
<td>4131T</td>
<td>16538</td>
<td>28218</td>
<td>75.3*</td>
<td>877761</td>
<td>212482</td>
<td>4 8 8 8 16 8 16 16</td>
</tr>
<tr>
<td></td>
<td>44556</td>
<td>16538</td>
<td>28218</td>
<td>84.1*</td>
<td>1057230</td>
<td>237313</td>
<td>4 8 8 8 16 8 24 16</td>
</tr>
<tr>
<td></td>
<td>45409</td>
<td>16534</td>
<td>28213</td>
<td>84.3*</td>
<td>1079990</td>
<td>266836</td>
<td>4 8 8 16 16 16 16 16</td>
</tr>
<tr>
<td></td>
<td>44523</td>
<td>13248</td>
<td>21841</td>
<td>91.0*</td>
<td>878805</td>
<td>196933</td>
<td>4 8 8 8 16 4 8 4</td>
</tr>
<tr>
<td></td>
<td>42945</td>
<td>12301</td>
<td>19748</td>
<td>92.5*</td>
<td>782645</td>
<td>182669</td>
<td>4 8 4 4 16 8 6 4</td>
</tr>
<tr>
<td></td>
<td>43747</td>
<td>11382</td>
<td>17865</td>
<td>94.9*</td>
<td>741806</td>
<td>189567</td>
<td>4 8 4 4 16 8 6 4</td>
</tr>
<tr>
<td></td>
<td>43805</td>
<td>12354</td>
<td>20692</td>
<td>98.0*</td>
<td>884229</td>
<td>202782</td>
<td>2 8 4 8 16 8 4 8</td>
</tr>
<tr>
<td></td>
<td>45441</td>
<td>11865</td>
<td>19753</td>
<td>100.8*</td>
<td>904777</td>
<td>199110</td>
<td>2 8 4 8 16 8 4 4</td>
</tr>
<tr>
<td></td>
<td>44523</td>
<td>12305</td>
<td>19755</td>
<td>101.1*</td>
<td>889227</td>
<td>199723</td>
<td>4 8 4 8 16 4 8 4</td>
</tr>
<tr>
<td></td>
<td>44849</td>
<td>11831</td>
<td>18908</td>
<td>103.2*</td>
<td>886631</td>
<td>194099</td>
<td>4 8 4 4 16 8 4 4</td>
</tr>
<tr>
<td></td>
<td>43747</td>
<td>11382</td>
<td>17868</td>
<td>103.8*</td>
<td>809812</td>
<td>185112</td>
<td>4 8 4 4 16 4 8 8</td>
</tr>
<tr>
<td>Random Logic</td>
<td>35552</td>
<td>6577</td>
<td>10335</td>
<td>41.1*</td>
<td>151014</td>
<td>42476</td>
<td>16 12 8 2</td>
</tr>
<tr>
<td></td>
<td>34848</td>
<td>6734</td>
<td>10849</td>
<td>41.4*</td>
<td>153634</td>
<td>44087</td>
<td>16 12 8 3</td>
</tr>
</tbody>
</table>

- 6 -
Logic circuit. Each point takes about 1.5 minutes of cpu time on a VAX 11/750. Many of the locally optimal solutions have identical parameter values on the critical path, but differ in other coordinates because of different random starting values.

7. Parallel Adder: The effect of loading factors

The preceding results did not take the loading on the output of the circuit into account. When these circuits are used in arrays, this may become important. To study this problem, we implemented two circuits for a 4-bit parallel adder, using the Data Selector and the Random Logic 1-bit full adders of the previous section. The results are shown in Table 2.

6. Discussion of Results

6.1. $P_{max} \times T$ tradeoff

Figure 8 shows $P_{max} \times T$ trajectories followed by the critical path optimization process, when minimizing $T$ for the Random Logic circuit. The dotted envelope shows the final tradeoff curve for $P$ vs $T$. Notice that the locally optimal point obtained by using PT as the cost criterion lies very close to the trajectory obtained when minimizing $T$. (See point a, with $P = 12.5\text{mW}$, and $T = 22.4\text{ns}$.) For comparison, the optimization for PT gave us a locally optimal point b with $P = 10.9\text{mW}$ and $T = 22.6\text{ns}$, very close to point a. Thus, optimization using the two criteria is consistent.

6.2. Performance comparison among the PLA, Data selector, and Random logic.

<table>
<thead>
<tr>
<th>type</th>
<th>$A$</th>
<th>$P_{max}$</th>
<th>$Pmax$</th>
<th>$T$</th>
<th>$APT$</th>
<th>$PT$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Logic</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Data Selector</td>
<td>105</td>
<td>283</td>
<td>313</td>
<td>96</td>
<td>200</td>
<td>299</td>
</tr>
<tr>
<td>PLA</td>
<td>278</td>
<td>486</td>
<td>520</td>
<td>78</td>
<td>715</td>
<td>403</td>
</tr>
<tr>
<td>PLA (Berkeley)</td>
<td>286</td>
<td>550</td>
<td>600</td>
<td>78</td>
<td>852</td>
<td>486</td>
</tr>
</tbody>
</table>

Table 3 shows a normalized performance comparison of the best locally optimal point for each layout, minimizing $T$. The Random Logic seems to be the best choice in all respects except $T$. However, it is the fastest among the 4-bit parallel adder implementations. The $T$ of the 4-bit parallel adder using Random Logic is less than 4 times the $T$ of the 1-bit full adder, while in the other layouts it is more than 4 times the $T$ of the 1-bit full adder. The reason is that this Random Logic 1-bit full adder circuit calculates the carry signal and propagates it before the calculation of the sum signal, so the carry ripple propagates faster than the sum. As a result, the 4-bit parallel adder takes only 2.5 times as much time as the 1-bit full adder. Figure 7 shows the $P-T$ tradeoff curve of each layout. The curve for the Random Logic circuit is below the one for the Data Selector, which is below that for the PLA. Hence we can order the layouts with Random logic best, Data Selector next, and PLA last. This result agrees with our intuition because this order is the same as the order of circuit specialization.

6.3. Comparison between our PLA and the Berkeley PLA

Both PLAs have almost the same costs, except for $P$. The reason is that our locally optimal point occurs at the choice $d = (4,4,4,4,4,4,3,4,4,8,8,8,4,4,4,8,2)$. 

-7-
while the Berkeley PLA adopts \( d = (4,4,4,4,4,4,8,8,8,8,8,8) \). The Berkeley PLA is therefore very close to locally optimum with respect to \( T \).

### 8.4 Comparison with Myers' work

Myers did similar performance comparisons of various 1-bit full adder implementations [9], but did not use any optimization. His results, shown in Table 4 below, are quite different from ours, shown in Table 3. Our results show that an appropriate choice of layout and its optimization makes the Random Logic circuit better than the Data Selector, and that the PLA can be made very fast at the expense of Power.

**Table 4. 1-bit full adder normalized performance comparison (Myers[9])**

<table>
<thead>
<tr>
<th>type</th>
<th>( A )</th>
<th>( P_{\text{max}} )</th>
<th>( T )</th>
<th>( APT )</th>
<th>( PT )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Logic</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Data Selector</td>
<td>45</td>
<td>50</td>
<td>125</td>
<td>28</td>
<td>72.5</td>
</tr>
<tr>
<td>PLA</td>
<td>105</td>
<td>110</td>
<td>170</td>
<td>196</td>
<td>187</td>
</tr>
</tbody>
</table>

#### 8.5. 4-bit Parallel Adder

Tables 1 and 2 show that the locally optimal point of the 1-bit full adder is attained with a pull-down diffusion width of the carry output stage \( d_{c_0} = 2 \) or 3, while the corresponding width for the 4-bit parallel adder is \( d_{c_0} = 8 \). The pullup width remains 2. This suggests that the critical path passes through the pull-down of the output carry stage, which is indeed the case.

On the hand, for the Data Selector, the critical path passes through the pullup of the output carry stage, and in fact it is the pullup width that expands during optimization of the 4-bit parallel adder.

#### 8.6. Comparison of the 1-change and 2-change methods

Figure 8 and Table 5 show a comparison between the 1-change and the 2-change methods when applied to the Random Logic implementation. Table 5 is discussed in the next section. The slope of the 2-change method is steeper than that of the 1-change method, but the 2-change method reaches better locally optimal points. Hence in this case the 2-change method works better than the 1-change method does. However, the 2-change method does not work as well as the 1-change method for the Data Selector, which has many more parameters. The 2-change method took more iterations than the 1-change method and did not obtain better locally optimal points.

#### 8.7. Effectiveness of our optimization: Cost Improvement ratio

Table 5 below shows the average initial delay times \( T_0 \) (obtained from random starts), the average locally optimal delay time \( T_{\text{opt}} \), the average percent improvement of the delay time \( T \), and the best locally optimal delay time \( T_{\text{best}} \). We can see from this that 2-opt performs much better than 1-opt. We should note that it is very important to choose a good order in which to try improvements, because this saves unnecessary search time evaluating changes that are unlikely to be improvements. For example, we chose the diffusion widths of the 3-input NAND gate as the first parameters tried for the Random Logic circuit.
Table 5 Cost improvement of our optimization methods

<table>
<thead>
<tr>
<th>type</th>
<th>opt</th>
<th>criterion</th>
<th>T&lt;sub&gt;initial&lt;/sub&gt;</th>
<th>T&lt;sub&gt;opt&lt;/sub&gt;</th>
<th>% improvement</th>
<th>T&lt;sub&gt;best&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Logic</td>
<td>1-opt</td>
<td>T</td>
<td>29.7</td>
<td>19.2</td>
<td>33</td>
<td>19.1</td>
</tr>
<tr>
<td>Random Logic</td>
<td>2-opt</td>
<td>T</td>
<td>29.7</td>
<td>16.8</td>
<td>42</td>
<td>16.4</td>
</tr>
<tr>
<td>Data Selector</td>
<td>1-opt</td>
<td>T</td>
<td>24.3</td>
<td>17.7</td>
<td>25</td>
<td>15.8</td>
</tr>
<tr>
<td>Data Selector</td>
<td>2-opt</td>
<td>T</td>
<td>23.5</td>
<td>18.0</td>
<td>23</td>
<td>15.8</td>
</tr>
<tr>
<td>PLA</td>
<td>1-opt</td>
<td>T</td>
<td>19.3</td>
<td>16.3</td>
<td>16</td>
<td>12.8</td>
</tr>
</tbody>
</table>

9. References


Figure 4. PLA

Figure 5. Data Selector

Figure 6. Random Logic
END

FILMED

9-85

DTIC