An Efficient Transistor Folding Algorithm for Row-Based CMOS Layout Design

Jaewon Kim
Quickturn Design Systems, Inc.
440 Clyde Av.
Mountain View, CA 94043

S. M. Kang
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Abstract

In timing-driven layout synthesis, transistor sizes tend to be significantly different from each other and thus the use of conventional layout approaches can cause inefficient area utilization. We propose an efficient algorithm to find the optimal transistor folding sizes in row-based designs. Our algorithm finds optimal folding sizes given a CMOS circuit with $m$ pairs of pMOS and nMOS transistors in $O(m^2 \log m)$ time complexity with the effective reduction of the solution space. MCNC benchmark circuits are used to demonstrate the area-efficiency of the physical layouts with optimal folding sizes.

1 Introduction

Among various design attributes, timing delay has become one of the most important constraints in the high-performance circuit design. In order to meet all the timing requirements, transistors with various current driving capabilities are required. Judicious increase of certain transistor sizes can reduce the circuit delay at the expense of additional chip area[1]. Many optimization techniques have been introduced to solve the transistor sizing problem[2, 3, 4]. Most approaches try to minimize the area subject to the constraints on the maximum circuit delay or the area-delay product term. Even if transistor sizes are optimized to minimize the total diffusion area, the variations in the transistor sizes may make the circuit layout worse than the one with non-optimal uniform transistor sizes, depending on the layout design methodology. Among various design methodologies that have been developed recently, row-based design has become most popular. In row-based layout synthesis, the variation in transistor sizes may cause non-uniform cell heights and the non-uniform cell heights may lead to significant waste of the layout area. In order to utilize the chip area more efficiently, a transistor folding scheme should be introduced. The objective of transistor folding is to keep the cell height of cells uniform and concurrently to reduce the total layout area. Traditionally, the transistor folding procedure has been assigned to layout experts exclusively for high-performance custom cell or standard cell designs, while it has been ignored by cell generation tools. Recently, several researchers have worked on transistor folding [5, 6]. However, their works are based on the fixed cell height or cell-by-cell local optimization, which are not guaranteed to lead to optimal layout area.

In this paper, we present an efficient algorithm that can find the optimal transistor folding size for given transistor sizes in a circuit. We first eliminate redundant folding sizes from the solution space. Then, we apply the modified exhaustive method with the time complexity of $O(m^2 \log m)$ to find the optimal folding size, where $m$ is the number of transistors in the circuit. In the automatic synthesis of custom VLSI chips, the transistor folding scheme contributes significantly to the total layout area reduction.

Copyright © 1997 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept., ACM Inc., fax +1 (212) 869-0481, or permissions@acm.org.

Design Automation Conference ©

2 Problem Definition

Let $C$ be a CMOS circuit with $m$ transistor pairs. The transistor sizes of $C$ are divided into two sets according to the transistor types. Let $P = \{p_1, p_2, \ldots, p_m\}$ and $N = \{n_1, n_2, \ldots, n_m\}$ be the sets of the sizes of pMOS transistors and the sizes of nMOS transistors, respectively. Each transistor size is assumed to be in multiples of $\lambda$, the minimum resolution size. A transistor pair with the same index, $p_i$ in $P$ and $n_i$ in $N$, have the CMOS duality. If $p_i$ ($n_i$) does not have its dual transistor, $n_i$ ($p_i$) is set to zero.

In order to satisfy the design rules, each transistor size should be at least equal to the minimum transistor size, $PMIN$ for pMOS transistor and $NMIN$ for nMOS transistor, which are also multiples of $\lambda$. Each element in $P$ and $N$ should satisfy the minimum size constraint as

\[
p_i \geq PMIN \text{ for all } p_i \in P,
\]

\[
n_i \geq NMIN \text{ for all } n_i \in N.
\]

The folding size limits the maximum height of any transistor layout. If the specified transistor size exceeds the folding size of its type, the transistor should be folded. In order to determine the optimal folding size for each transistor type precisely, it is desirable to generate physical layout for every possible folding size. However, since it is excessively time-consuming to generate actual layouts, an area estimate function is introduced. The total layout area is estimated by summing the cell areas. Since the routing area is difficult to estimate precisely, it is considered an overhead. The estimation of the total cell area is determined by

\[
\text{Area} = (\max(P) + \max(N) + \text{VertOverhead}) \times (\text{TotalColumn} + \text{HoriOverhead}).
\]

VertOverhead includes the overhead for the power rails, the minimum distance between pMOS and nMOS transistors and the estimation of the minimum intra-cell interconnections including terminal positions. TotalColumn represents the total number of transistors multiplied by the minimum column width specified in the design rules. HoriOverhead represents the horizontal area for gate output and the intra-cell interconnections.

The direct layout synthesis with the given transistor sizes without folding may waste large chip area due to the wide variation of the transistor sizes. In the above estimation, the height of cells is estimated to be $\max(P) + \max(N) + \text{VertOverhead}$, but this height may not be optimal in terms of area because most cells in the same row have shorter heights.

If we limit the maximum transistor height, it can improve the overall area efficiency. Let the folding sizes be $h_p$ and $h_n$ for pMOS and nMOS transistor, respectively. Any pMOS(nMOS) transistor size is larger than $h_p(h_n)$, the transistor should be folded. The amount of the wasted area can be reduced as long as the horizontal expansion does not incur the net area increase.

If a transistor is folded with folding size $h_n$, the transistor is divided in $[p_i/h_p] \times [n_i/h_n]$. The objective is to minimize the area function, which is represented as

\[
\text{area}_{\text{col}} = (h_p + h_n + \text{VertOverhead}) \times \text{width},
\]

where \text{width} is represented in terms of $h_p$ and $h_n$, as

\[
\text{width} = \sum_i \max([p_i/h_p], [n_i/h_n]) + \text{HoriOverhead}.
\]
The reason why we apply the max operation in the above formula is that pMOS and nMOS transistors share the same input signal on vertical poly-silicon layer, so both types can not be considered separately. This is due to the fact that our scheme is targeted to automatic CMOS layout synthesis in which gate column sharing is done to reduce layout area. The upper bound of \( h_p(h_n) \) because any pMOS (nMOS) transistor should be larger than or equal to \( \text{PMIN} (\text{NMN}) \), which is specified by the design rule of the corresponding CMOS technology.

Therefore, the problem can be formulated as a minimization problem with the double-sided constraints.

\[
\begin{align*}
\text{minimize} & \quad (h_p + h_n + \text{VertOverhead}) \\
\text{subject to} & \quad \sum_i \max \left( \frac{P_i}{h_p}, \frac{N_i}{h_n} \right) + \text{HoriOverhead}, \\
\text{where} & \quad \text{PMIN} \leq h_p \leq \max(P), \\
& \quad \text{NMN} \leq h_n \leq \max(N).
\end{align*}
\]

If two adjacent diffusion areas are electrically equivalent, then layout area can be reduced by merging the diffusions. Otherwise, a diffusion break is required between them. In the above function, we do not consider the diffusion breaks which may be caused by transistor folding for two reasons. First, we do not know how many diffusion breaks are required before actual cell synthesis. Normally, the number of diffusion breaks are determined by transistor ordering algorithms. All good transistor ordering algorithms try to minimize the diffusion breaks between transistors. Second, transistor folding in conjunction with any good transistor ordering algorithms introduces new diffusion breaks because folded transistors are usually abutted as shown in Fig. 1.

![Figure 1: Folding example.](image)

### 3 Optimal Folding

A simple method to solve the problem would be, albeit inefficient, to consider all possible folding sizes exhaustively, calculate the areas and choose the pair of folding sizes with the minimum area, as shown in Fig. 2. The time complexity of line 1 and 2 are determined by the maximum size of the transistor in the circuit because the minimum size of the transistors are fixed. Let the size of the solution space for \( h_p \) and \( h_n \) be \( s \). Since line 3 takes the time that is proportional to the number of transistor pairs, \( m \), the total time complexity of the algorithm becomes \( O(s^2m) \). However, depending on the circuit behavior, \( s \) can be constant or dependent on the number of transistor pairs. In the design of full custom VLSI circuits, each transistor will be customized to have its own optimal transistor size in terms of layout area and timing delay. Furthermore, if a transistor sizing algorithm is used in the synthesis, every transistor can have a different size. If transistor sizes in a circuit are almost uniform and the solution space of \( s \) is very small, we may not need any transistor folding scheme. However, this is not the case in real custom designs. Thus, we assume that \( s \) is \( m \) for the worst case in which all transistor sizes are different and the time complexity becomes \( O(m^3) \). With such time complexity, the solution process of the optimal folding sizes for large circuits may take prohibitively long time.

**Reduction of solution space**

In the previous section, we defined the solution space of \( h_p \) and \( h_n \) for the optimum area. Even though the size of the solution space is dependent on the circuit behavior, the space is usually very large due to the variation of the transistor sizes in the circuit. Hence, it is desirable to minimize the solution space to reduce the run time. However, the reduction process should not remove the possible optimal solutions of the folding sizes. The following definitions and theorems are applied to both types of transistors. Here, we only mention the pMOS transistor case without loss of generality.

**Definition 1** Let \( S_p^c \) be an ordered set of the possible folding sizes of transistor \( p_i \) in \( P \). \( S_p^c \) is composed of all \( \left[ \frac{p_i}{k} \right] \) that are greater than or equal to \( \text{PMIN} \). \( S_p^c \) is sorted in descending order.

The ordered folding size set \( S_p^c \) is a partial set of the entire solution space of folding sizes of \( p_i \). Since \( k \) is a positive integer, every element in \( S_p^c \) is smaller than or equal to \( \max(P) \).

**Theorem 1** Folding size \( s_p^c \), that satisfies \( \text{PMIN} \leq s_p^c \leq \max(P) \) but is not in \( S_p^c \) does not yield the minimum area solution.

Proof: Suppose \( S_p^c = \{ s_1, s_2, \cdots, s_n \} \) be an ordered set of possible folding sizes of transistor \( p_i \) in \( P \). Let \( k \) be a positive constant and let \( \left[ \frac{p_i}{k} \right] \) be \( s_l \) in \( S_p^c \). Since \( S_p^c \) is sorted in descending order, we can divide the situation into three cases:

(i) \( \left[ \frac{p_i}{k} \right] = \left[ \frac{p_i}{k+1} \right] \)  
(ii) \( \left[ \frac{p_i}{k} \right] = \left[ \frac{p_i}{k+1} \right] + 1 \)  
(iii) \( \left[ \frac{p_i}{k} \right] = \left[ \frac{p_i}{k+1} \right] + c \) where \( c > 1 \).

Obviously, there is no other possible integer between \( \left[ \frac{p_i}{k} \right] \) and \( \left[ \frac{p_i}{k+1} \right] \) in the first and the second cases. In the third case, let \( \left[ \frac{p_i}{k} \right] \) and \( \left[ \frac{p_i}{k+1} \right] \) be \( a + c \) and \( a \), respectively. \( a + c \) and \( a \) are elements of \( S_p^c \), but \( a + c - 1, a + c - 2, \cdots, a + 1 \) are not contained in \( S_p^c \) according to the definition of \( S_p^c \). We need to justify why \( a + c - 1, a + c - 2, \cdots, a + 1 \) are not better than \( a \) in terms of the cell area. \( p_i \) can be represented by other constants,

\[ p_i = (a + c - 1)k + b, \quad \text{where} \quad 0 < b \leq k. \]

If we apply the folding size, \( a + c - 1 \) to \( p_i \),

\[ \left[ \frac{p_i}{a + c - 1} \right] = k + \left[ \frac{b}{a + c - 1} \right]. \]

Since \( b \) is greater than 0,

\[ \left[ \frac{p_i}{a + c - 1} \right] \geq k + 1. \]

This induces

\[ \left[ \frac{p_i}{a + c - 1} \right], \left[ \frac{p_i}{a + c - 2} \right], \cdots, \left[ \frac{p_i}{a} \right] \geq k + 1. \]  

(1)

\( p_i \) can be also represented as

\[ p_i = (a - 1)(k + 1) + d, \quad \text{where} \quad 0 < d \leq k + 1. \]

If we apply the folding size, \( a \) to \( p_i \),

\[ \left[ \frac{p_i}{a} \right] = k + 1 + \left[ \frac{d}{a} \right]. \]

Since \( d \) is less than or equal to \( k + 1 \),

\[ \left[ \frac{p_i}{a} \right] \leq k + 1. \]

This also induces

\[ \left[ \frac{p_i}{a + c - 1} \right], \left[ \frac{p_i}{a + c - 2} \right], \cdots, \left[ \frac{p_i}{a} \right] \leq k + 1. \]  

(2)
According to (1) and (2),
\[
\begin{bmatrix}
p_i / a + c - 1 \\
p_i / a + c - 2 \\
\vdots \\
p_i / a
\end{bmatrix} = k + 1.
\]
\(a+c-1, a+c-2, \ldots, a+1\) produce the same folded transistor numbers as \(a\) does, but the estimated area with these folding sizes are obviously greater than the estimated area with \(a\) because \(a+c-1, a+c-2, \ldots, a+1\) that are greater than \(a\) cost more area with the same folded numbers. Note that the multiplication of the folding size and the folded number mainly contributes to the area estimate. Hence, \(a+c-1, a+c-2, \ldots, a+1\) that are not elements of \(S^p\) can be ignored in the area minimization process.

Definition 2 \(S^p\) is the set of ordered folding sizes for transistor set \(P\), if \(S^p\) contains all elements in \(S^p_1 \cup S^p_2 \cup \cdots \cup S^p_n\) and is sorted in descending order.

Theorem 2 The folding size \(s^p_k\), that satisfies \(\text{PMIN} \leq s^p_k \leq \max(P)\) but is not contained in \(S^p\), is an element of \(S^p\).

Proof: Let \(S^p = \{s^p_1, s^p_2, \ldots, s^p_n\}\) be an ordered set of the folding sizes for \(P\) and \(x\) be an integer that satisfies \(s^p_k > x > s^p_{k-1}\), but is not an element of \(S^p\). According to the above theorem, \(x\) generates the same folded transistor numbers as \(s^p_{k-1}\) does because \(x\) does not correspond to \(\frac{p}{a}\) for any \(p\) in \(P\) and any positive integer \(k\).

\[
x * \text{folded number}(x) > s^p_{k+1} * \text{folded number}(s^p_{k+1}),
\]

because both folded numbers for \(x\) and \(s^p_{k+1}\) are equivalent, \(x\) can be ignored in \(S^p\) without degradation of the solution quality.

With the theorems described above, the continuous solution space is reduced and converted to the discrete solution space.

Algorithm optimal-folding 2:

```
optimal-folding() {
  initialize column
  nMOS-phase
  end optimal-folding

  pMOS-phase() {  
    sort \(P\) and \(N\) in descending order of \(p\) in \(P\)
    2. for each \(s^p\) in \(S^p\)
      3. divide \(P\) and \(N\) into sections
      4. calculate section number \(g_k\) of each section
      5. sort each section in descending order of \(n\) in \(N\)
      6. for each \(s^p_k\) in \(S^p\)
        7. for each \(k\)
          if \(g_k > \frac{\max(P)}{s^p_k}\), where \(n_{k1}\) is the first nMOS transistor in section \(k\)
            column\([s^p_k, s^p_k]\) = column\([s^p_k, s^p_k]\) + \(g_k * [g_k]\)
          else if \(g_k < \frac{\max(P)}{s^p_k}\), where \(n_{k2}\) is the last nMOS transistor in section \(k\)
            do nothing
          else
            find first \(n_{k3}\) that satisfies \(\frac{\max(P)}{s^p_k}\) < \(g_k\)
            column\([s^p_k, s^p_k]\) = column\([s^p_k, s^p_k]\) + \(g_k * [g_k]\)
        end for
      end for
    end for
  end pMOS-phase
```

Figure 3: Algorithm 2.

Optimal folding size

After the sets for the reduced solution space for pMOS and nMOS transistors are found, the optimal folding sizes for pMOS and nMOS transistors can be chosen from the reduced sets. In order to avoid the time complexity of \(O(m^n)\) suggested in Fig. 2, the proper algorithm and data structure should be established. The main idea is to count the folded numbers of pMOS transistors and nMOS transistors efficiently for given folding sizes. With the given folding sizes, we only need to count the dominant folded number between a pair of dual transistors.

The optimal folding size problem is divided into two phases; the pMOS phase and the nMOS phase. In the pMOS phase, the folded numbers of the pMOS transistors that are greater than or equal to the folded numbers of the nMOS transistors are counted. Likewise, in the nMOS phase, the folded numbers of the nMOS transistors that are greater than the folded numbers of the pMOS transistors are counted. The algorithm is shown in Fig. 3. Since the only major difference between the pMOS phase and the nMOS phase is whether there exists an equality in line 8, only the pMOS phase is described in Fig. 3.

Since we count the folded numbers in a two-phase operation, the storage for the folded numbers is needed during the entire operation. column\([s^p, s^p]\) is a two dimensional array whose indices represent the folding sizes of each transistor type. Figure 4 shows an example of transistor sizes in a circuit that has 12 transistor pairs. The same indices represent the dual transistors between pMOS and nMOS transistors. The initial state of transistor arrays is shown in Fig. 5(a). In the pMOS phase, the transistor arrays are sorted in descending order of pMOS transistor sizes. With any folding sizes of pMOS transistors which is given in line 2 of Fig. 3, the array is divided into several sections within which the folded numbers of the transistors are the same. Suppose that \(s^p_k = 10\), then the array is divided into three sections because the maximum of the folded numbers with \(s^p_k\) is 3 and the minimum is 1. Section numbers indicate the folded transistor numbers. In Fig. 5(b), transistors \(p_1, p_2, p_3, p_4, p_5, n_1, n_2, n_3, n_4\) are contained in the first section whose section number is 3. After sectioning, each section is sorted again in descending order of nMOS transistor sizes as shown in Fig. 5(c).

After all sections are sorted, the arrays are ready for the calculation of the folded transistor numbers. Each section will be in one of the following three situations with given folding size of nMOS transistor \(s^p_k\):

1. The section number is greater than the folded number of the first nMOS transistor in the section.
2. The section number is smaller than the folded number of the last nMOS transistor in the section.
3. None of the above.

The first represents the situation that the folded number of the pMOS transistors in a section is greater than the folded numbers of any other nMOS transistors in the section. Since the larger folded number dominates the smaller one, we can ignore the folded numbers of nMOS transistors in such sections. The total sum of folded number is calculated as

\[
\text{column}(s^p_k, s^p_k) = \text{column}(s^p_k, s^p_k) + g_k * [g_k],
\]

where \(g_k\) and \([g_k]\) are the folded number of section \(k\) and the number of elements in section \(k\), respectively. In Fig. 5(c), the section with section number 3 corresponds to this situation. The second case is the opposite situation of the first one. This
case should be handled in the nMOS phase. If we calculate the folded numbers of nMOS transistors in pMOS phase, the time complexity is proportional to the number of transistors, while it takes only constant time in nMOS phase. The section with section number 1 in Fig. 5(c) belongs to this situation. The third case represents the situation wherein the folded number of the pMOS transistors in a section is a median value of the folded numbers of nMOS transistors. In order to calculate the partial sum of the folded numbers, the section point that satisfies

\[
\frac{\sum_{i=k}^{n} |a_i|}{s_k} \leq g_k,
\]

should be found in a section. We can calculate the partial sum of the folded numbers for the partial section that satisfies the above situation as,

\[
column[s_i, s_j] \leftarrow column[s_i, s_j] + g_k \times |g_k|,
\]

where \(|g_k|\) is the number of elements that satisfy the above condition in section \(k\). The section with section number 2 corresponds to this situation in Fig. 5(c). The rest of the elements in the section that do not contribute to the partial sum, are considered in the nMOS phase. However, the above condition is slightly different in the nMOS phase, because the equality condition is already considered in the pMOS phase. The condition in the nMOS phase is

\[
\left|\frac{D_m}{s_j}\right| < g_b.
\]

In Fig. 5(c), \((p_1, n_1)\) is the section point which matches the given folding sizes. Only the transistor pairs, \((p_1, n_1), (p_0, n_0), (p_2, n_2)\) that satisfy the above condition in the section with section number 2, are considered in the calculation of the partial sum of the folded numbers in the pMOS phase.

**Time Complexity**

In order to measure the performance of the algorithm shown in Fig. 3, the time complexity should be compared to that of the algorithm in Fig. 2. Let the number of transistor pairs and the number of possible folding sizes in \(S^p\) be \(m\) and \(s\), respectively. The sorting of the transistor arrays in line 1 takes \(O(m \log m)\) with a reasonable sorting algorithm. The second sorting process in line 3 linearizes the entire transistor arrays, which costs \(O(sm)\). Since a typical transistor is folded only a finite number of times and is often smaller compared to the folding size, the maximum folded number is assumed to be a constant. The maximum number of sections is also set to a constant, line 4 takes \(O(s)\). The sorting process in each section takes \(O(sm \log m)\). While line 7 costs \(O(s^2)\), line 8 takes \(O(sm \log m)\) in the worst case due to the binary searching. Hence, \(O(sm \log m)\) or \(O(s^2)\) can represent the time complexity of the optimal folding algorithm depending on the situation. It is hard to define the relation between \(s\) and \(m\). However, we expect that they are dependent on the applied design methodologies. With a given design rule and many planning cells, an orthogonal fold approach, the size of \(s\) is a small constant and irrelevant to \(m\). Standard cell approach may have moderately big size of \(s\), which is partially dependent on \(m\). The major concern of this paper is fully customized design cell that has the big size of \(s\). In the worst case of this approach, every transistor width can be different from the others. If we assume the folding size is proportional to the number of transistor pairs in the circuit as we did in the first algorithm, we can get the worst case complexity of \(O(m^2 \log m)\), which is significantly smaller than \(O(m^p)\) of the algorithm in Fig. 2.

### 4 Experimental Result

We implemented the optimal transistor folding algorithms and compared the performance in terms of the run time and the area using a layout generation system [8]. First, in order to measure how the optimal transistor folding affects the layout area, the physical layouts of several circuits including MCNC benchmarks were generated with cell generation, placement and routing as shown in Table 1. For circuit eldckt which is a full custom design, a significant amount of area was saved by the optimal transistor folding scheme. Others are benchmark circuits of the standard cell approach. For the transistor sizes specified in the MCNC standard cell library, which are relatively uniform compared to customized designs, the area ratios are less than that of eldckt.

In order to consider the running time of the algorithms, algorithm 1 shown in Fig. 2 without solution space reduction and algorithm 2 shown in Fig. 3 with the solution space reduction were implemented. Both algorithms were used to generate the optimal transistor folding sizes of the benchmark circuits. Table 1 shows the comparison result on run time. While both algorithms produce the same folding sizes for all designs, the run times of the second algorithm are significantly smaller than those of the first algorithm. This run time data was measured on a SPARCstation 10/30 with the algorithms described in C language.

We found the optimal folding sizes of each transistor type and they were fed to the custom cell synthesis system to generate physical layouts of the circuits. In order to observe the effect of the folding sizes, we applied various folding sizes to generate the physical layouts of fract. Through placement and routing tools, the circuit layouts were automatically synthesized. Table 2 shows the area ratios of fract with the various folding sizes. The layout with the worst area was generated with \(P = 80\) and \(N = 10\), while the layout with the best area was generated with \(P = 30\) and \(N = 30\) or \(N = 20\). The actual best or worst area terms may be obtained with the median folding sizes. Not all possible folding sizes are shown in Table 2 due to the space limitations of the table. From Table 1, we can observe the global view of the area variation with the folding sizes specified. We note that the area difference can be 30% depending on the folding sizes of transistors.

### Table 1: Algorithm results on MCNC benchmarks.

<table>
<thead>
<tr>
<th>circuit</th>
<th>tr. name</th>
<th>layout area ((\mu m^2))</th>
<th>area run time (sec)</th>
<th>speed</th>
<th>algorithm 1</th>
<th>algorithm 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>eldckt</td>
<td>192</td>
<td>3884000</td>
<td>268200</td>
<td>0.69</td>
<td>2.4</td>
<td>0.39</td>
</tr>
<tr>
<td>highway</td>
<td>156</td>
<td>131469</td>
<td>114168</td>
<td>0.99</td>
<td>2.64</td>
<td>0.63</td>
</tr>
<tr>
<td>fact</td>
<td>598</td>
<td>6680000</td>
<td>69200</td>
<td>0.91</td>
<td>10.87</td>
<td>2.12</td>
</tr>
<tr>
<td>struct</td>
<td>8890</td>
<td>1.238x10^9</td>
<td>1.069x10^9</td>
<td>0.88</td>
<td>173.07</td>
<td>14.61</td>
</tr>
<tr>
<td>sqw small</td>
<td>373</td>
<td>1.08x10^9</td>
<td>93x10^9</td>
<td>0.91</td>
<td>225.79</td>
<td>22.98</td>
</tr>
<tr>
<td>sqw large</td>
<td>88258</td>
<td>1.199x10^9</td>
<td>1.111x10^9</td>
<td>0.93</td>
<td>322.45</td>
<td>31.43</td>
</tr>
</tbody>
</table>

### Table 2: fract layout area comparison.

<table>
<thead>
<tr>
<th>pMOS folding size ((\mu m))</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>nMOS</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>folding size ((\mu m))</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>fract</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>area ratio</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

### REFERENCES


