Contents

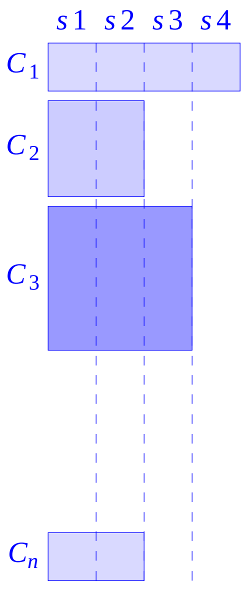
CHAPTER 54

COMPONENT FOLDING

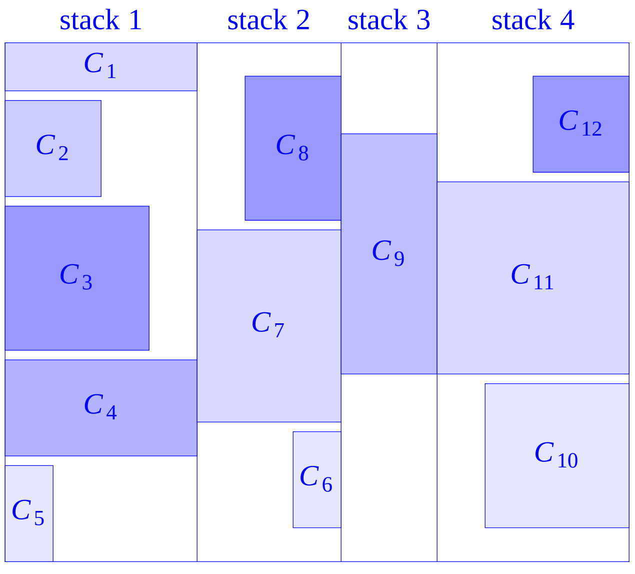
54.1 COMPONENT FOLDING

Engineers follow several different design styles when they design electronic circuits. Two of these styles are **bit-slice designs** and **standard-cell designs**. In the former the electronic circuit is first designed as a stack of components, as shown in Figure 54.1(a). Each component C_i has a width w_i and a height h_i. The width of the component is an integer number of **slices**. The design of Figure 54.1(a) is a four-slice design. The wires that connect components follow the slice. That is, wires may connect from slice j of component C_i to slice j of component C_{i+1}. If one of these components isn't j slices wide, then there are no slice j wires between them. When the bit-slice design of Figure 54.1(a) is to be realized as part of a larger system, it is allocated some amount of space on a (very large scale integrated) VLSI chip. This allocation is done either by placing a restriction on the width of the space or on its height. The problem now is to fold the stack into the allocated space so as to minimize the unspecified dimension (i.e., if the height is restricted to H, the width W of the area into which the stack is folded is to be minimized). Minimizing this dimension is equivalent to minimizing the area because the other dimension is fixed.

A component stack is folded in a snake-like manner. At each fold point the components get rotated by 180 degrees. In the example of Figure 54.1(b), a 12-component stack has been folded into four vertical stacks. The fold points are C_6, C_9, and C_{10}. The width of a folded stack is the number of slices required by



(a) A four-slice stack



(b) A folding

Figure 54.1 A component stack and a folded layout

the widest component in the slice. In the example of Figure 54.1(b), the stack widths are 4, 3, 2, and 4. The height of a folded stack is obtained by summing the heights of the components in each stack and taking the maximum of these sums. In Figure 54.1(b) the sum of component heights is maximum for stack 1. The height of this stack determines the height of the rectangle needed to enclose all stacks.

In realistic cases of the component folding problem, we need to account for the additional space needed to carry the wires from one stack to the next. For example, in Figure 54.1(b) the wires between C_5 and C_6 get bent because C_6 is a fold point. These wires need vertical space below C_5 and C_6 so they can cross from stack 1 to stack 2. Let r_i denote the height allowance required in case C_i is a fold point. Now the height needed to accommodate stack 1 is $\sum_{i=1}^{5} h_i + r_6$. The height needed by stack 2 is $\sum_{i=6}^{8} h_i + r_6 + r_9$.

In standard cell designs, an electronic circuit is first designed as a linearly ordered list of components that have the same height. Suppose that the components are C_1, $\cdots$, C_n in this linear order. Next the components are folded into equal-width rows as in Figure 54.2. In this figure 12 standard cells are folded into four equal-width rows. The fold points are C_4, C_6, and C_{11}. Between pairs of adjacent standard cell rows, we use a routing channel to make the electrical connections between cells in different rows. The fold points detrmine the required channel heights for the routing. Let l_i denote the channel height needed when C_i is a fold point. In the example of Figure 54.2, the height of routing channel 1 is l_4, that of channel 2 is l_6, and that of channel 3 is l_{11}.

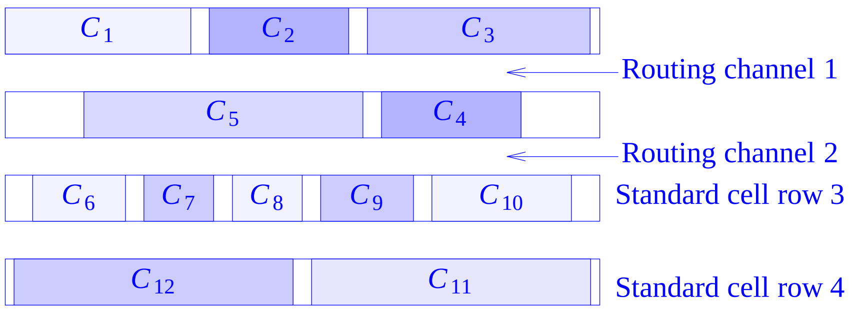


Figure 54.2 Standard cell folding

The folding of bit-slice stacks, as well as of standard cells, leads to several problems that may be solved using dynamic programming.

Folding of Equal-Width Bit-Slice Components

Define $r_1 = r_{n+1} = 0$. The height requirement of a stack composed of the components C_i through C_j is $\sum_{k=i}^{j} h_k + r_i + r_{j+1}$. Suppose that all components of a bit-slice design have the same width w. First consider the case when the height H of the rectangle into which the folding is to be done is given, and we are to minimize its width. Let W_i be the width of a minimum-width folding of the components C_i through C_n into a rectangle of height H. In case such a folding is not possible (for example, when $r_i + h_i > H$), let $W_i = \infty$. Notice that W_1 is the width of the best folding possible for all n components.

When folding the components C_i through C_n, we need to decide on the fold points. Suppose these decisions are made for the stacks in left-to-right order. If the first decision is to fold at C_{k+1}, then C_i through C_k are in the first stack. For the overall folding to have minimum width, the remaining components C_{k+1} through C_n must be folded in an optimal manner. Hence the principle of optimality holds, and dynamic programming may be used to solve the problem. When the first fold point $k+1$ is known, we get the equality

$$W_i = w + W_{k+1} \tag{54.1}$$

Since we do not know the first fold point, we need to try out all feasible fold points and select the one for which Equation 54.1 gives the minimum W_i. Let $hsum(i,k) = \sum_{j=i}^{k} h_j$. For $k+1$ to be a feasible fold point, $hsum(i,k) + r_i + r_{k+1}$ must not exceed H. Using this observation, we obtain the following dynamic-programming recurrence:

$$W_i = w + \min\{W_{k+1} | hsum(i,k) + r_i + r_{k+1} \leq H,\ i \leq k \leq n\} \tag{54.2}$$

Here $W_{n+1} = 0$ and in case there is no feasible fold point $k+1$, then W_i is ∞. Equation 54.2 may be solved iteratively for W_1 by computing the W_is in the order W_n, W_{n-1}, $\cdots$, W_1. The computation of W_i involves the examination of at most $n-i+1$ W_{k+1}s and can be done in $O(n-k)$ time. So the time needed to compute all W_is is $O(n^2)$. By saving the values of k that yield each minimum in Equation 54.2, we can use an $O(n)$ traceback to compute the optimal fold points.

Let us consider another folding problem involving equal-width components. This time the width W of the rectangle into which the folding is to be done is known, and we are to minimize its height. Since each folded stack has width w, the maximum number of stacks in the folded layout is $s = W/w$. Let $H_{i,j}$ be the height of a minimum-height folding of C_i, $\cdots$, C_n into a rectangle of width jw. $H_{1,s}$ is the minimum height into which all n components can be folded. When $j = 1$, no folding is permitted, and so

$$H_{i,1} = hsum(i,n) + r_i,\ 1 \leq i \leq n$$

Also, when $i = n$, there is only one component and no folding is possible. Therefore,

$$H_{n,j} = h_n + r_n,\ 1 \leq j \leq s$$

For other $H_{i,j}$, folding is possible. If the first fold is at $k+1$, then the first stack has a height of $hsum(i,k) + r_i + r_{k+1}$. The remaining components must be folded using a width of at most $(j-1) * w$. To ensure optimality of the overall folding, the remaining components should be folded with minimum height. So

$$H_{i,j} = \max\{hsum(i,k) + r_i + r_{k+1},\ H_{k+1,j-1}\} \tag{54.3}$$

Since we do not know the first fold point, we try out all possible fold points and pick the one for which the right side of Equation 54.3 is minimum. The resulting recurrence is

$$H_{i,j} = \min_{i \leq k < n} [\max\{hsum(i,k) + r_i + r_{k+1}, H_{k+1,j-1}\}] \tag{54.4}$$

This recurrence may be solved iteratively for $H_{i,j}$, $1 \le i \le n$, $1 \le j \le s$ by computing the $H_{i,j}$s first for $j = 2$, then $j = 3$, and so on. The time needed to compute the $H_{i,j}$s for each j is $O(n^2)$, so it takes $O(sn^2)$ time to compute all $H_{i,j}$s. By saving the k values that yield the minimums in Equation 54.4, we can determine the optimal fold points by a tracback procedure of complexity $O(n)$.

Folding of Variable-Width Bit-Slice Components

First consider the case when the height H of the rectangle into which the folding is to be done is given and its width is to be minimized. Let W_i be as in Equation 54.2. Using an argument similar to that used to derive Equation 54.2, we obtain

$$W_i = \min\{wmax(i,k) + W_{k+1} | hsum(i,k) + r_i + r_{k+1} \le H,\ i \le k \le n\} \qquad (54.5)$$

Here $W_{n+1} = 0$, and $wmax(i,k) = \max_{i \le j \le k}\{w_j\}$. This recurrence may be solved in a manner similar to Equation 54.2. The time needed is $O(n^2)$.

When the width, W, of the folded layout is given, the minimum-height folding may be obtained using a binary search over the $O(n^2)$ posssible values, $h(i,j) + r_i + r_{j+1}$, for this height. For each height checked, we use Equation 54.5 to determine whether there is a folding with width $\le W$. The total time needed to find the minimum height for which the folding can be done into a rectangle of width at most W is $O(n^2 \log n)$.

Standard Cell Folding

We shall use w_i to denote the width of cell C_i. Each cell has height h. When the width W of the standard cell rows is fixed, the layout area is minimized by minimizing the height of the folding. Consider any minimum-height folding of the components C_i through C_n. Suppose that the first fold point is C_{s+1}. The folding of the components C_{s+1} through C_n must use minimum height; otherwise, we can use a lesser height folding of these components and thereby get an even lesser height folding of C_i through C_n. So the principle of optimality holds, and the dynamic-programming method may be used.

Let $H_{i,s}$ be the minimum height when components C_i through C_n are folded into a rectangle of width W with the first fold being at C_{s+1}. Let $wsum(i,s) = \sum_{j=i}^{s} w_j$. We may assume that no component has width greater than W; otherwise, there is no feasible folding. For $H_{n,n}$, we see that there is just one component, and no routing is needed. Therefore, $H_{n,n} = h$. For $H_{i,s}$, $1 \le i < s \le n$, we see that if $wsum(i,s) > W$, the folding is infeasible. If $wsum(i,s) \le W$, components C_i and C_{i+1} are in the same standard cell row and the height of the routing channel just below is l_{s+1} (define $l_{n+1} = 0$). As a result

$$H_{i,s} = H_{i+1,s} \tag{54.6}$$

When $i = s < n$, the first standard cell row contains only C_i. The height of this row is h, and the height of the routing channel just below is l_{i+1}. The cells C_{i+1} through C_n are folded optimally. Therefore,

$$H_{i,i} = \min_{i<k\leq n}\{H_{i+1,k}\} + l_{i+1} + h \tag{54.7}$$

To find the minimum height-folding, we first use Equations 54.6 and 54.7 to determine $H_{i,s}$, $1 \leq i \leq s \leq n$. The height of the minimum-height folding is given by $\min\{H_{1,s}\}$. We can use a traceback procedure to determine the fold points that result in the minimum-height folding.

EXERCISES

1. Write an $O(n^2)$ iterative Java program to find optimal fold points for bit-slice stacks with equal-width components. Use Equation 54.2.

2. Do Exercise 1 using Equation 54.4. This time the complexity of your code should be $O(sn^2)$.

3. Write a Java program to find a minimum-width folding of a stack of variable-width components. Use Equation 54.5. The complexity of your program should be $O(n^2)$.

4. Use the development of Section 54.1 to arrive at an $O(n^2 \log n)$ algorithm to find a minimum height folding into a rectangle of width W. The bit-slice components have different widths.

5. Use Equations 54.6 and 54.7 to determine a minimum height folding of n standard cells. Your code should run in $O(n^2)$ time. Can you think of a way to use these equations to obtain an $O(n)$ time method for this problem?

54.2 REFERENCES AND SELECTED READINGS

Our discussion of bit-slice and standard-cell folding is based on the papers "Optimal Folding of Bit Sliced Stacks" by D. Paik and S. Sahni, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12, 11, 1993, 1679–1685; "Folding a Stack of Equal Width Components" by V. Thanvantri and S. Sahni, *IEEE Transactions on CAD of ICAS*, 14, 6, 1995, 775–780; and "Optimal Folding of Standard and Custom Cells" by V. Thanvantri and S. Sahni, *ACM Transactions on Design Automation and Electronic Systems*, 1996. The second paper uses **parametric search** to obtain faster algorithms than the dynamic programming algorithms described in this book. The last paper shows how Equations 54.6 and 54.7 may be solved in $\Theta(n)$ time. These three papers also consider other variants of the folding problem.