VLSI Leaf Cell Design by Understanding Circuit Structures *

Wen-Jeng Lue and L. P. McNamee
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024

Abstract

A new approach utilizing MOS circuit structures extracted from a circuit net-list for designing VLSI leaf cells is described. A circuit structure is explicitly present in a circuit schematic diagram on which a designer relies for drawing a layout. However, it is absent in the net-list input to an automatic layout system. In this paper, how to extract schematic like information from a net-list and how to apply it for automatic leaf cell design are discussed.

1 Introduction

The cell layout problem involves the construction of geometric artwork at different semiconductor layers that define the circuit components and interconnections for the cell from its circuit net-list description and boundary constraints. The net-list describes the circuit components, their interconnections, and the underlying fabrication technology. Cell boundary constraints specify the topological and geometric requirements around the outside boundary of the layout cell.

The circuit structure is understood through a hierarchical net-list partitioning and signal flow directions among circuit components. The circuit network is partitioned in such a way that basic circuit building blocks can be identified in most cases. A dependency-directed backtracking search mechanism is employed to assign signal flow direction for each circuit component by making use of various knowledge about MOS circuit signal flow. This general search scheme is capable of incorporating more knowledge, e.g. the set of rules discussed in [1], to become more competent in assigning signal flow directions

The major use for the extracted net-list structure to generate a layout is its implication on layout cell dissection. This approach explicitly establishes a relationship between the net-list structure and the layout placement. A *slicing-structure* spatial arrangement is adopted. This has been widely used both in manual design and expert layout design systems[3; 2; 4; 15]. However, all such systems lack an understanding of the circuit structure. They are able to use only local connection patterns, but not global ones, in the circuit net-list for making layout decisions.

In the following sections, the circuit network partitioning method, a search mechanism for assigning signal flow directions, and the use of net-list structure for making layout decisions are discussed. Finally, these ideas are demonstrated in a walk-through example followed by some discussions. Although this approach is based on CMOS circuits, the same principles can be easily applied to circuits in other MOS technologies.

2 Circuit Network Partitioning

Partitioning can be described in terms of a series of graph operations. A graph is obtained from the input net-list. It is then split into several graph components along some specific nodes. Each component is characterized by the set of signal nodes associated. Unions of components are established depending on their characteristics. The set of graph components so obtained defines the circuit network partitioning.

2.1 Circuit Graph

A **circuit graph** can be derived directly from the input net-list which is assumed to be in a SPICE input format[7]. A net-list $\Omega(S,T)$ consists of a set of signal $S(\Omega)$, abbreviated as S, connected through a set of transistors $T(\Omega)$, abbreviated as T. Each element in T can be either an *n-type* transistor or a *p-type* transistor, which is associated with three signals in S through its *source*, *drain* , and *gate* nodes. A signal in S is *active* if there are at least one p-type transistor and at least one n-type transistor with either source node or drain node connected to it. In general, output signals of CMOS circuits are active. A signal is *normal* if it is not active. Some signals may be declared as *input, output*, or *bias* in the input specification independent of the circuit structure. There are only two different bias signals, *vdd* and *vss*. Other signals not specified are taken as *interior*.

A **circuit graph** $C(V,E)$ is defined according to a net list $\Omega(S,T)$. The vertex set V is defined such that each vertex corresponds to either one signal in S or one transistor in T and vice versa. A signal connecting to a transistor, by its source, drain, or gate node, defines an edge in E.

2.2 Blocks

Net-list partitioning is done by splitting the **circuit graph** into several subgraphs and then combining them

*This research has been supported in part by the State of California MICRO program and the Rockwell International Incorporated.

to form *blocks*. This is obtained by two steps, graph splitting and graph union. A set of subgraphs of the circuit graph C can be obtained through the following steps:

1. Remove all edges corresponding to connections between signals and transistor gate nodes from C.
2. Remove all *isolated vertices* after step 1, which denote signal vertices connected only to gate nodes of transistors.
3. Split vertices of active and bias signals in the remaining graph.

The above steps originate from the following two observations. First, transistor gate signal connections in digital MOS circuits always function as inputs to the transistors connected, therefore they do not constitute connections within a functional block of transistors. Second, functional blocks connect with each other through transistor source nodes or drain nodes at either active or bias signals.

Let $\Theta = \{\Theta_1, \Theta_2, \ldots, \Theta_l\}$ be the set of resulting subgraphs. An important set of vertices characterizing each subgraph Θ_i are all active and bias signal vertices in Θ_i such as

$$\tilde{V}(\Theta_i) = V(\Theta_i) \cup ACTIVE(C) \cup BIAS(C)$$

where $ACTIVE(C)$ and $BIAS(C)$ are two sets of nodes in the circuit graph C corresponding to active signals and bias signals respectively. Note that if Θ_i comes from a combinational logic gate in the original circuit, it will contain both bias and active signal vertices. The only active signal vertex is the output signal of this logic gate. If Θ_i is obtained from a pass logic gate, it will not, in general, contain bias signal vertices and consist of more than one active signal vertex.

A *block* is a union of several subgraphs by joining identical signal vertices. Let $\tilde{\Theta}_k = \Theta_{k1} \cup \Theta_{k2} \cup \ldots \cup \Theta_{kj}$ be a block, then subgraphs $\{\Theta_{k1}, \Theta_{k2}, \ldots, \Theta_{kj}\} \subset \Theta$ satisfy the following conditions:

1. Either all $\tilde{V}(\Theta_{ki})$ contain bias signal vertices or all of them don't contain bias signal vertices.
2. $\tilde{V}(\Theta_{k1}) - BIAS(C) = \tilde{V}(\Theta_{k2}) - BIAS(C) = \ldots = \tilde{V}(\Theta_{kj}) - BIAS(C)$

This is because subgraphs in the same block are expected to share the same active signal vertices that are not bias. Furthermore, each subgraph of a block would contain bias signal vertices if that block contains bias signal vertices. The latter case is manifested in combinational logic circuits.

A partition of the input net-list can hence be defined according to the set of blocks obtained. An observation of the CMOS circuit characteristics is that functional gates, static or dynamic, can usually be partitioned in this way.

A **block graph** B can be built based on the set of blocks obtained. Each vertex in B is either a block vertex, corresponding to a subnet-list as a block, or a signal vertex, corresponding to a signal that connects transistors in different blocks. An edge is defined only between a signal vertex and a block vertex signifying a connection between the signal and some transistors inside the block.

2.3 Serial-Parallel Connections

Serial-parallel connection patterns are important structures in CMOS circuits as discussed in [5]. Such structures are to be extracted for every block. The extraction of serial-parallel connections between transistor source/drain nodes in each block can be done in a manner similar to the approach discussed in [5] with the constraint that no vertices corresponding to active or bias signals can be used for serial connections. A new graph can be obtained by extracting all serial-parallel connectioned edges into serial-parallel trees. Each edge of the resulting graph corresponds to either a transistor or a serial-parallel tree representing a serial-parallel connection pattern.

3 Signal Flow Assignment

3.1 Signal Flow Constraints

A set of constraints for assigning signal flow directions between a signal and a block based on the **block graph** B are described as follows:

C1 A signal should flow along one and only one direction between a signal vertex and block vertex.

C2 There is at least one signal input to each block through a transistor source or drain node.

C3 There is at least one signal output from each block through transistor source or drain node.

C4 If a signal is declared as input, it must go into some blocks.

C5 If a signal is declared as output, it must come from some blocks.

C6 An interior signal must go into some blocks.

C7 An interior signal must come from some blocks.

Besides this set of constraints, two heuristics are employed:

H1 If a signal connects to a block through transistor gate nodes, it goes into that block.

H2 A bias signal always goes into the block it connects.

C1 states that a signal should flow along one and only one direction between each signal and a connected block. Even though some signals may travel bidirectionally in an actual circuit, it is reasonable to consider only one direction for identifying a global circuit structure. **C2** and **C3** consider signal flows only through transistor sources or drains because transistor gates are usually regarded as receptive site for controlling signal flows. Note that if input and output signals are are not specified, all signals would be considered interior. It is quite likely that not all the constraints can be satisfied in this case. Eventually, some constraints will have to be relaxed. **H1** reflects the commonly known fact that gate nodes of CMOS transistors in digital circuits are used mainly for signal input. **H2** states that bias signals, namely, vss and vdd, are considered as input signals to

all blocks connected to them. These heuristics play very important roles in determining an initial partial signal flow specifications.

3.2 Dependency-Directed Backtracking Search

Given the knowledge about how a signal flow direction should be assigned locally, a search is needed to find a direction for each undirected edge of **block graph** B guided by a set of constraints. This constraint satisfaction problem[13] can be solved through a *dependency-directed backtracking* scheme[12], where domain knowledge can be applied to regulate the control of the search and to reduce further the computational cost. A *reasoning maintenance system*(RMS)[6] is employed to detect the presense of contradictions and determine the set of search decisions responsible. Each signal flow direction assignment may come directly from an assumption made during the search or deducted from the set of constraints and all assumptions made. The underlying RMS performs such deductions whenever an assumption is asserted, records how an assignment is obtained, detects conflicts, and reports all possible sets of assumptions causing the conflicts if they occur.

Selecting one edge to assign a signal flow direction from the set of undirected edges observes the following order:

1. If two edges with the same signal vertex are undirected, one connecting to a *combinational block* and the other connecting to a *pass block*, set the signal flow for the latter edge into the *pass block*.
2. If an edge with an input signal vertex is undirected, set the signal flow out of the signal vertex.
3. If an edge with an output signal vertex is undirected, set the signal flow into the signal vertex.
4. If an edge is undirected, set the signal flow into the block vertex connected.

The above rules are valid according to heuristics of CMOS circuit properties. The first rule comes from the commonly occurred feedback-loop structure. A block is defined *combinational* if: 1) it contains the same number of p-type transistors and n-type transistors; 2) it connects to both vdd and vss; and 3) of all signals that are not bias, it connects to only one active signal. A *pass block* refers to a block not connecting to any bias signal vertex. These definitions are heuristic in the sense that all combinational logic gates and pass logic gates will satisfy such conditions, but not necessarily vice versa. The second and third rules imply the natural tendency of directing signal flow towards output signals and from input signals. The last rule simply selects one undirected edge arbitrarily and assigns to it a signal flow direction towards the block.

There are two sources of conflicts possible during the search for signal flow directions:

1. The initial assumptions.
2. The set of input/output signals assigned for a block.

Currently, the initial assumptions are made very conservatively. The most common case it reports as a conflict is when a signal is assigned as input to all blocks it connects. This usually happens because the signal is actually an input signal but not declared as such. The assignment of input/output signals for a block has to satisfy some requirements based on the transistor source/drain node connections in the block. A conflict occurs when they are not satisfied. More detailed discussion on such a requirement is to be discussed later.

In the presence of conflicts, the search should retract some assertions, either assumptions about signal flow directions or the constraints themselves. Which one to choose is strongly dependent on the characteristics of the particular problem. The following is a priority list according to the assumption properties (from high to low):

1. A constraint about a signal which connects to all its adjacent blocks through transistor gate nodes.
2. A constraint about a signal which connects to all its adjacent blocks through transistor source/drain nodes.
3. A constraint about a block caused directly by initial assertions.
4. A signal flow direction assertion.

The above priority order in retracting assertions reflects some knowledge of the circuit structure. In the first case, the only possible reason is that the signal concerned is actually an input signal, because no signal would flow out of transistor gate nodes. The second one is established for the possibility that the signal can actually be an output signal, because output signals connect only through transistor source/drain nodes in most cases. The third one may also occur because of the lack of output signal information. The last selection is for the sake of completeness so that an initial signal flow assumption can be retracted rather than the constraint itself if no knowledge applies.

3.3 Signal Flow Assignment for Each Transistor

The signal flow directions along transistor source/drain nodes are specified based on the input and output signals of each block. Consider a graph defined through the transistor source/drain connections with all serial-parallel patterns extracted inside a block. The assumed requirements for a set of satisfactory transistor signal flow assignments are that:

1. For any input signal vertex, there exists a directed path to an output signal vertex that doesn't go through any other output vertex.
2. For any output signal vertex, there exists a directed path to an input signal vertex that doesn't go through any other output vertex.
3. Each edge has to belong to a directed path from an input vertex to an output vertex.

Such assumptions originate from the understanding that signals flow through transistor source/drain nodes from

input vertices to output vertices regulated by transistor gate nodes to perform some function.

One way to obtain signal flow directions satisfying the above requirements is through a *breadth-first search* that proceeds from input vertices towards output vertices and directs the edges accordingly.

```
1.  Initialize a queue as a list of
    all input vertices.
2.  Label all output vertices.
3.  Until the queue is empty, remove
    the first element q from the queue,
    3.a.  If q is an output vertex,
          do nothing.
    3.b.  Otherwise, for all v
          adjacent to q and
          v is not labeled,
          3.b.1 direct edge e from q to v,
          3.b.2 label vertex v,
          3.b.3 add v to the back
                of the queue.
```

A *tree* can be constructed by adding to the set of all labeled vertices and directed edges a new vertex *root* and new set of directed edges from *root* to every input vertex.

If the input and output vertices assignments are satisfactory, the above constructed tree will be a *spanning* tree that includes all vertices of the transistor source/drain graph of a block. In this case, each remaining undirected edge can then be directed from the vertex associated at a higher *level* of the tree to the other one at a lower *level*. Those with two vertices of the same *level* are directed arbitrarily.

However, if the constructed tree doesn't cover all the vertices, the input/output signal assignments for the block are unsatisfactory. In other words, some vertices can only be reached from the input vertices through output vertices, which violate the previous mentioned assumptions. This will be reported as a contradiction, and a new set of input and output vertices are to be found through the dependency-directed backtracking scheme.

4 Block and Transistor Ordering

Further ordering information about a circuit net-list structure can be obtained by using the block partitions and signal flow assigned for each transistor.

4.1 Loops and Topological Order

Loops among blocks reflect possible feedback paths existing in the net-list. This is an important structure appearing in many circuits, e.g. flip-flops, ring oscillators, and most clocked sequential circuits. In general, a block may belong to many loops. It is desired that all possible loops are identified. An algorithm based on the approach proposed in [14] is applied. Each loop is associated with a set of blocks constituting the loop.

An acyclic digraph can be obtained from a new partition of the net-list according to the directed **block graph** on the set of *strongly connected components*(SCC)[10]. Two vertices v, w are said to be *strongly connected*, $v \sim w$, if there is a directed path from v to w and vice versa. Clearly, $\sim$ is an equivalence relation and it partitions the vertex set into nonempty subsets $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_p$, such that if $v \in \mathcal{V}_i$ and w is strongly connected to v, then $w \in \mathcal{V}_i$. Each subset $\mathcal{V}_i$ is called a strongly connected component. With all loops identified, an SCC can be established by a block, if it doesn't belong to any loop; a loop, if it doesn't share constituent blocks with other loops; or a set of *connected* loops. *Connected* is an equivalence relation and two loops are *connected* if they share at least one block or are *connected* through other loops.

Each SCC $\mathcal{V}_i$ is associated with a subnet-list $\Omega_{\mathcal{V}_i}$ which is the union of all blocks corresponding to all strongly connected vertices in $\mathcal{V}_i$. Note that signal flow between $\mathcal{V}_i$s' is *acyclic*, i.e. it doesn't contain any directed cycles. Therefore there exists a topological order among net-list partitions $\Omega_{\mathcal{V}_1}, \Omega_{\mathcal{V}_2}, \ldots, \Omega_{\mathcal{V}_p}$.

4.2 Transistor Ordering

Transistors can be ordered inside each block according to the spanning tree identified by the transistor source/drain connections. Consider the spanning tree $\mathcal{T}_{\Omega_i}$ established for a block Ω_i. Let $l(v)$ be the level of the vertex $v \in \mathcal{T}_{\Omega_i}$, then one property of the spanning tree obtained through the *breadth-first search* based algorithm is

$$\text{if } e = < v_1, v_2 > \in E(\mathcal{T}_{\Omega_i}),$$

$$\text{then } l(v_2) = l(v_1) \text{ or } l(v_2) = l(c_1) + 1$$

Note that each edge e corresponds to a transistor in Ω_i. This results in a natural ordering of transistors according to the levels of the two vertices associated with their source and drain nodes.

5 Use of Net-List Structure in Cell Layout

The extracted net-list structure and the partial ordering relationships defined through assigned signal flow directions can be readily applied for generating a CMOS leaf cell. Its use is basically twofold: one is to provide a hierarchical partition of the cell layout area, and the other is to help determine the ordering relationships between different partitions, hence the layout placement. The leaf cell layout would then be completed with subsequent routing.

Considering layout problem as a search through all possible spatial arrangements to arrive at a satisfactory solution following a particular layout architecture, the extracted net-list structure provides a set of constraints to assist in finding a proper solution. It defines a reasonable initial solution following the natural properties of the input circuit. Some of the routing considerations are taken into account through both the clustering of circuit devises in the hierarchical structure and the ordering relationships derived from signal flow directions.

To simply the problem, all layouts will be addressed only at the symbolic level[8] where all geometrical mask

information is coalesced into symbols for circuit components and lines for connection wires. The layout problem consists of two separate, but not independent subproblems, placement and routing. Placement is of the major concern here, while routing can be done by applying existing algorithms.

5.1 Cell Slicing Structure Representation

A common way to represent cell layout area is by partitioning it into subareas. This is also one way to reduce the complexity of the cell layout problem applied both in automatic and manual design[15; 2]. Among various partitioning methods, the *slicing structure*[9] representation for rectangular dissection is the most widely used one, both for its simplicity, and more importantly, its suitability for a hierarchical description. The structure extracted from the circuit net-list implies exactly the adoption of such a cell representation scheme.

A slicing structure is characterized by a *decomposition tree* which represents a hierarchy of the regions in the layout rectangle. The root of the tree is the top level of the hierarchy and represents the enclosing rectangle. Each node corresponds to a subarea of the layout rectangle. The children of a node is created by a slicing operation, either horizontal or vertical. The recursive subdivision ends with nodes corresponding to rectangles that contain already completed layouts called *layout patterns.* In general, there are more than one slicing structure for one rectangular dissection.

5.2 Initial Placement

Cell layout placement can be obtained through the following two steps: first, determining a cell slicing structure to partition the input circuit, second, assigning the orientation and order of children slices for each slice.

Instanciating Slicing Structure A cell slicing structure can be determined by adopting the net-list structure as a decomposition tree with each node assigned either horizontal or vertical slicing for its children. To obtain a CMOS layout architecture similar the one presented above, the nodes in the decomposition tree are all assigned horizontal slicing except for those corresponding to blocks of the original circuit partition. These nodes are associated with vertical slicings for separating pmos transistors from nmos transistors if both types are included. Each node at the bottom level corresponds to a symbolic layout for a single transistor. A decomposition tree may also be obtained by modifying the original net-list structure. This happens when imposing layout patterns to the cell and hence requiring circuit clusters different from those in the original net-list structure.

Layout Pattern A layout pattern constrains a small number of transistors to be placed in one slice following a particular manner. One example is to pile up the same type of transistors sharing one gate signal. This is intended to either make use of extra vertical layout space or make the resulting cell higher and thinner. This type of placement arrangement is also used in *gate-matrix*[16] layout, where it is used as part of the layout constraints that have to be always followed.

In general, all possible patterns are extracted and selected according to layout heuristics. In case of insufficient heuristics, each of them will be considered to derive different final layouts and evaluated through layout performance measurement. Currently, no heuristics are included and multiple layout results are generated according to layout patterns utilized.

The major effect of imposing a layout pattern is the modification of the decomposition tree. If a pattern contains n non-overlapped subtrees of the original decomposition tree, the modification requires that these n subtrees to be removed from the original tree. The new subtree representing the pattern will be attached to one of the n positions in the original tree left open by the removal. Layout knowledge is needed to select one of the n positions, which is currently done arbitrarily.

Each pattern is associated with a set of conditions. They include net-list structural characteristics, electrical properties, and the input layout specification. Consider the pattern currently adopted. The set of conditions to activate it are: 1) each transistor should be of the same type; 2) the gate node signal of each transistor should be the same; and either 3) there are other transistors with large sizes compared with the ones considered; or 4) the width of the final layout cell is expected to be small.

With more layout knowledge, more layout patterns can be added to make the design more competent. However, the cost for extracting and selecting a proper set of layout patterns would also increase significantly. Different layout patterns may interact with each other, i.e. overlapping on the set of transistors included. Extra design knowledge is then needed to resolve such conflicts. Currently, no such knowledge is used and the resolution is arbitrarily done.

Order and Orientation Besides partitioning, a layout placement has to specify the ordering relationships between all sibling slices and the orientations of all bottom level patterns. Initially, the ordering relationships are arbitrarily specified except for those where the derived signal flow direction implies an ordered sequence, or others determined through layout heuristics. For example, p-type transistors are always placed above p-type transistors. Slices with circuit clusters serially connected are also ordered according to their sequence of connection. The transistor symbolic layout is initially oriented with its source node on the left and drain node on the right.

5.3 Layout Placement Optimization

An optimized layout placement based on the initial result can be obtained by applying a set of spatial operations to change the orientation of each slice and the order of its children. A simple evaluation function is used to calculate a heuristic concerning the signal wire routing. The hill-climbing procedure searches through the possible layout solutions under the same decompo-

sition tree. Even though the size of such a search space is still exponential, it is structured in a way consistent with the characteristics of the circuit.

Spatial Movement Two types of operations are defined for each slice to change the spatial arrangements of the circuit cluster included. One is *flipping* which reverses the order of its children slices along either horizontal or vertical direction and requires each of its children slice to be flipped along the same direction too. This proceeds recursively until the bottom symbolic transistor layout, which simply reverses its node positions along the flipping direction. Another operation is *swapping* that alters the ordering relationship of the children slices by exchanging the positions of any two of them. Either operation incurs only very local change of the layout placement. The first one is meant to vary the orientations of sibling slices, and the second one to examine different ordering relationships. Applying combinations of these two operations makes it possible to obtain any desired placement from any initial arrangement under the same decomposition tree.

Cost Evaluation A cost based on the change of estimated routing distance is calculated for each operation. The routing cost is measured according to Manhattan distance between signal nodes inside the slice considered and those outside the slice. If more than one pair exist for routing one signal, the one with minimum distance is selected. The final cost is the sum of the cost for each signal inside the slice.

Such distance measure is very crude since the coordinates at the symbolic level only exhibit relative positions rather than actual geometrical dimensions. However, since the measurement is used only for comparison, it is able to find some desired features for placement. Two prominent features are *alignment* and *abutment*. Alignment means that a set of routing pairs do not intersect with each other. Abutment refers to the situation that a routing pair of the same layer between adjacent slices are placed next to each other, and can therefore be connected without wiring. The cost associated with these two routing states are always minimum, although locally, with the cost function used. However, the currently adopted evaluation function may not always be able to distinguish other placement arrangements from these two desired situations.

Control Starting from the initial layout, each slice is examined for possible application of optimization operations to decrease the routing cost. This process continues until either it achieves a local minimum that no improvement is possible by applying any single operation, or it exceeds a preset limit on the number of iterations for this process. Such a limit on the amount of optimization operations is necessary because there is no guarantee that the simple evaluation function adopted will not result in a loop. However, this hasn't yet been found in all cases examined.

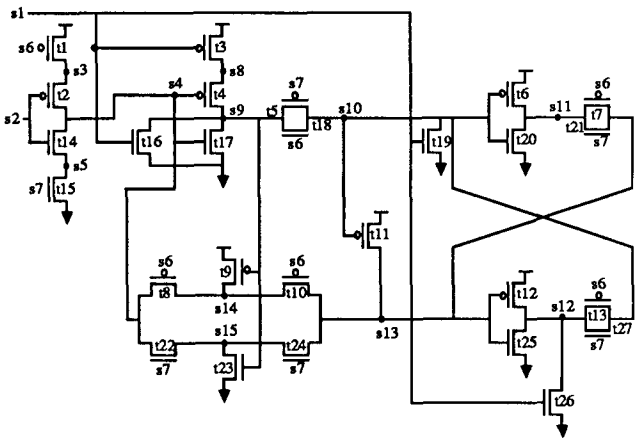

Figure 1: D-Flip-Flop Circuit

5.4 Routing

The channel routing scheme[11] is used to complete all signal net connections in a bottom-up fashion following the slicing structure. Each slice is considered to be routed only when all its children slices are routed. Except for slices which are layout patterns, routing channels are assigned between each adjacent sibling slices. The number of channels is decided in a conservative way such that all routing paths can be completed. To further simplify the routing problem, two layers of metal lines are employed. All vertical wires are in poly or metal-1 layers while horizontal ones are in metal-2 layer. The I/O ports specified by the boundary constraints can be connected after all routings inside the cell are completed.

6 An Example

Consider the CMOS network shown in *fig.* 1 as a working example. This is a *d-flip-flop* circuit consisting of 17 signals $S = \{s1, s2, \ldots, s15, vdd, vss\}$ and 27 transistors $T = \{t1, t2, \ldots, t27\}$. $T_p = \{t1, t2, \ldots, t13\}$ is the set of all p-type transistors and $T_n = \{t14, t15, \ldots, t27\}$ that of all n-type transistors. The set of active signals S_{active} includes $\{s4, s9, s10, s11, s12, s13\}$ and the rest will be normal, $S_{normal} = S - S_{active}$. Assume that the *input* and output signals are not specified and that all signals except the bias ones {vdd, vss} are taken as interior.

The set of blocks identified from partitioning the circuit network is illustrated in *fig.* 2. Functional implications exist in some of the blocks obtained. Block a is a *clocked inverter*; b is a *nor-gate*; g is an *inverter*; h is the combination of an *inverter* and a *pass-gate* for *reset* purposes; and the remainder are *pass-gates*.

The signal flow assignment procedure first asserts that signals flow out of bias signal vertices and into blocks if connected through transistor gate nodes. These assertions are shown in *fig.* 3. The constraints are then propagated and nine more directions are inferred accordingly. For example, arc $(a, s4)$ is deduced from the constraint that there has to be at least one signal flows out of a through vdd, vss, or $s4$. Since sig-

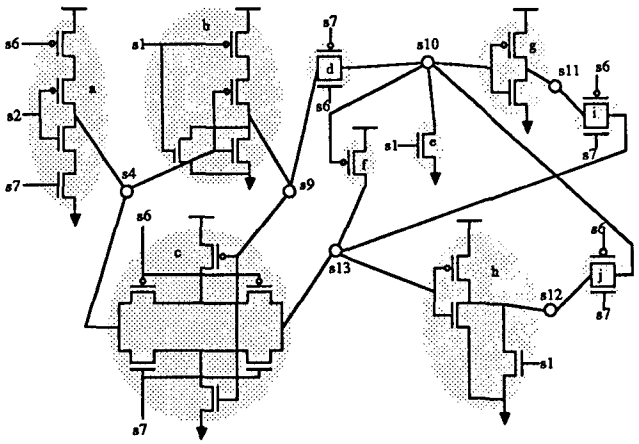

Figure 2: Blocks of D-Flip-Flop Circuit

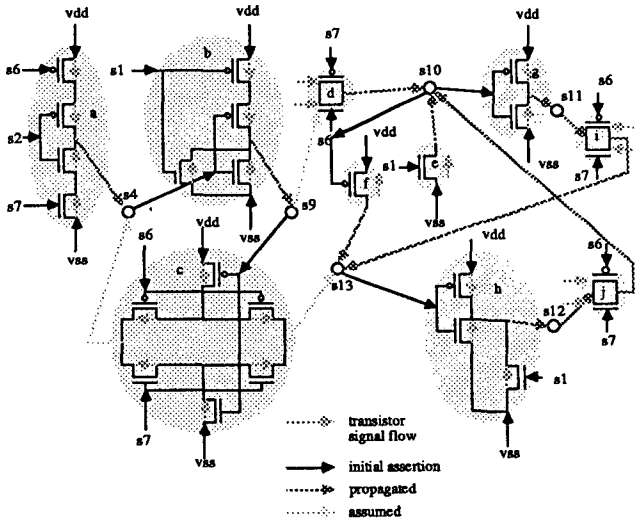

Figure 3: Signal Flow Directions Assignment

nals have to flow into a through vdd and vss, as initially asserted, the signal is therefore constrained to flow out of a through $s4$.

The DDB (*dependency-directed backtracking*) is then applied to resolve possible conflicts and to guide the assignment of the rest of signal flow directions. Three conflicts have been incurred so far. The constraint that each signal should have at least one input signal flow (*constraint* **C6**) is violated at all three signal vertices $s1$, $s6$, and $s7$. This is because they are all connected to transistor gate nodes. The underlying RMS (*reasoning maintenance system*) identifies the causes of each conflict to retract. For the violation at $s1$, it is reported that either assertion $s1 \rightarrow j$, assertion $s1 \rightarrow i$, assertion $s1 \rightarrow d$, assertion $s1 \rightarrow a$, or the constraint itself is at fault, where $s \rightarrow a$ denotes that signal s flows into block a and $a \rightarrow s$ means signal s flows out of block a. The heuristic that *constraints*, rather than *assertions* will be retracted has been implemented in the DDB. Therefore, this constraint at signal net $n11$ is retracted to remove one contradiction. Similar actions are taken for signals

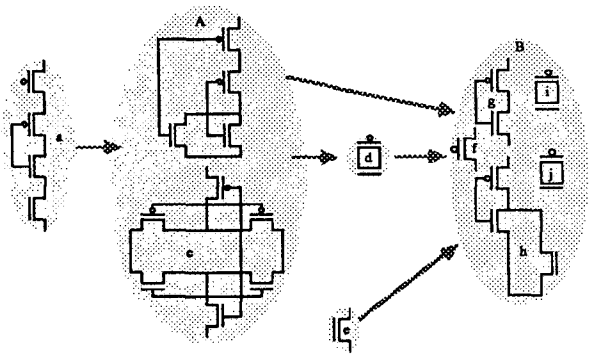

Figure 4: Topological Ordering

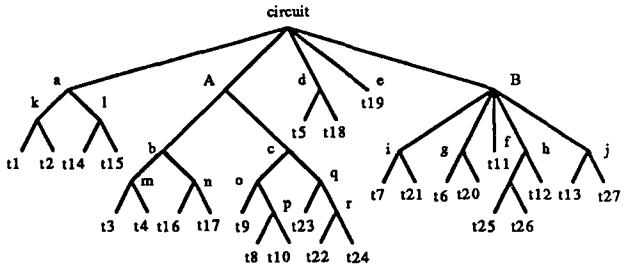

Figure 5: Extracted Net-List Hierarchy

$s6$ and $s7$.

After all contradictions are resolved by relaxing constraints, the search proceeds to assign signal flow directions for those edges not yet directed. According to the control heuristic **H1**, the edge incident to a vertex for pass block that connects to a combinational block should be considered first. Hence $s9 \rightarrow d$ is asserted, and from this $d \rightarrow s10$ is inferred. The last two assignments $c \rightarrow s4$ and $c \rightarrow s13$ are done arbitrarily but satisfy all constraints. Signal flows between transistor source and drain nodes are all determined according to the input/output signals of each block without any inconsistency as shown in *fig.*3. Since all signal flows are assigned and no constraints are violated, the DDB stops with an acceptable solution.

Three loops are found among the blocks in *fig.* 3. They are $L_1 = \{b, c\}$, $L_2 = \{h, j, f\}$, and $L_3 = \{g, i, h, j\}$. L_3 is basically composed of a flip-flop. All blocks in a loop are strongly connected; therefore loop L_3 defines a strongly connected component B. Since loops L_2 and L_3 share two blocks,h and j, they both define another strongly connected component A. A topological order can be found among the new circuit network partitions a, e, d, A, and B as shown in *fig.* 4

The extracted net-list structure is shown in *fig.*5. Nodes $\{a, e, d, A\}$, and B correspond to the partitions shown in *fig.*4. Blocks include nodes $\{a, b, c, d, e, i, g, f, h, j\}$. There are three clusters defined by serial connections and two defined by parallel connections, which are nodes $\{k, l, m\}$, and nodes $\{n, s\}$, respectively.

Two different cell layout placements are exemplified in *fig.*6 and *fig.*7. The former has a decomposition tree

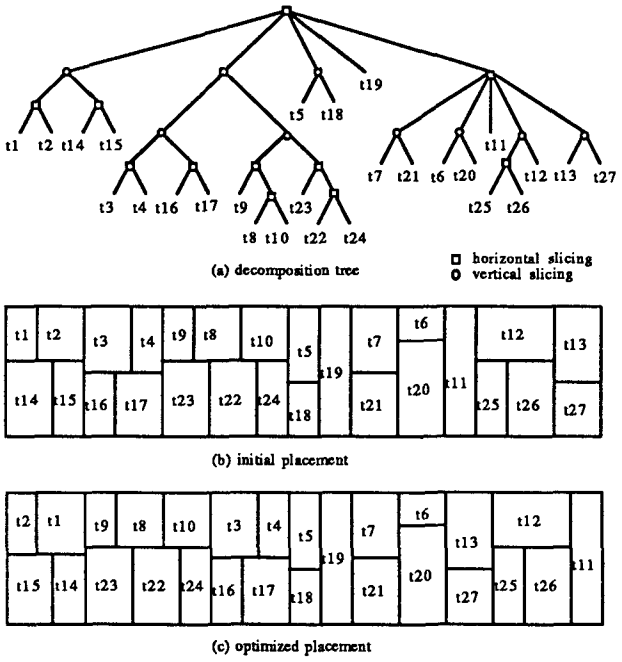

Figure 6: Placement Example 1

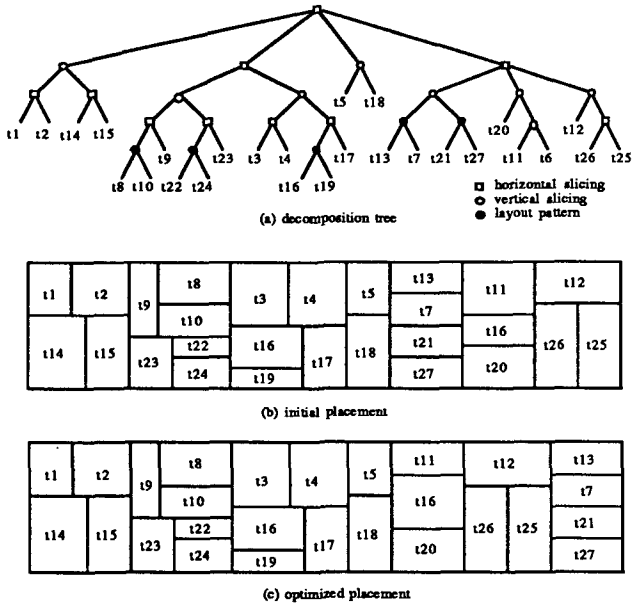

Figure 7: Placement Example 2

exactly the same as the net-list structure. It assigns all nodes with horizontal slicing except for those representing blocks where two different types of transistors are partitioned vertically. In *fig.*7, several layout patterns are imposed, therefore the decomposition tree shown in (a) is slightly different from the net-list structure. The initial placements are shown in part (b) of both figures, while the optimized results are shown in part (c). Note that both placements in (b) and (c) in the two examples share the same decomposition tree for cell dissection, differing only in the orientation and order for each slice. The orientation of each transistor is considered in the placement although not indicated in the figure. Two final symbolic layouts completed with routing based on the two optimized placements are shown in *fig.*8 and *fig.*9. The number of horizontal tracks in *fig.*8 is 30, while that for those in *fig.*9 is 24, as expected for the imposed layout patterns. However, this can only be used as heuristic comparison. The exact evaluation should still be based on final mask layouts.

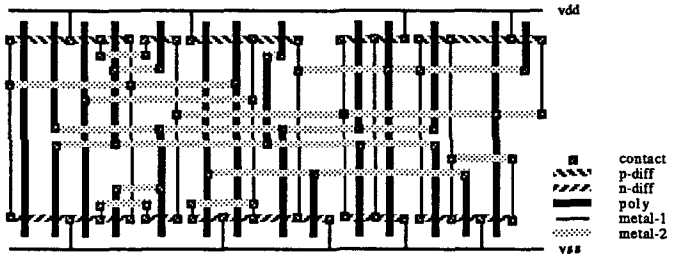

Figure 8: Completed Symbolic Layout 1

7 Implementation and Discussion

This system is implemented on a Sun4/280S using Kyoto Common Lisp. All algorithms involved are of polynomial time complexity except the one searching for the signal flow direction of each transistor. In the worst case, assigning either of two possible signal flow directions for each transistor will be of exponential time complexity, i.e. $O(2^n)$ where n is the number of transistors. However, with the help of heuristic constraints, only a very small number of possible solutions need to be considered. Of the examples tested, a completed symbolic

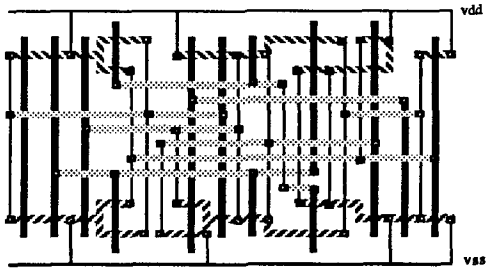

Figure 9: Completed Symbolic Layout 2

layout placement takes about 2 minutes for the *d-flip-flop* circuit (consisting of 27 transistors) presented above (both with and without imposing layout patterns), and 30 seconds for a circuit with 10 transistors. The routing time is not considered because it is independent of the net-list structure used. Among the 2 minutes computation time, about 15 seconds are spent on the extraction of net-list structure, 10 seconds for the initial placement, and the rest 95 seconds for optimization. The ratio may differ for different cases, but the optimization cost is significantly higher than others. This is because the cycle intensive nature of such computation. The initial placement examined doesn't take too much time because only one type of layout pattern is considered. It can be expected that as more and more layout patterns participate in the design, this portion of computation will increase considerably.

Although the flexibility of identifying a set of reasonable signal flow assignments has been demonstrated, even without specifying the input/output signals of the circuit, the above analysis of net-list is by no means complete. In all the examples tested, the signal flows assigned are consistent with the original intent of the circuit. However, it is not guaranteed to be so for all other circuits. With only a small number of heuristics rules, there are many sets of possible signal flow assignments that satisfy all constraints. If more specific results are desired, more knowledge is needed to narrow down the number of possible solutions. Many circuits, designed with electrical rather than functional considerations, haven't been taken into account in this work. The general search scheme used for identifying signal flow directions is capable of incorporating more MOS circuit knowledge to achieve better results for both net-list partitioning and signal flow assignments. One such set of knowledge directly applicable can be found in [1].

Many improvements are possible for generating the layout from the net-list structure. One example is transistor sizing. Although transistor sizing has been considered in enacting the simple layout pattern applied, the placement still assumes uniform size for each transistor. Representation of different sizes of transistor at the symbolic level would require different evaluation function for optimization. Another area to improve is the use of more layout patterns from the real design experience. Other spatial movements for optimization may also be introduced, e.g. rotating a slice, for further exploration of a better layout.

8 Conclusion

An approach for VLSI leaf cell layout based on an understanding about MOS circuits is presented. Various knowledge about the circuit is applied to extract a net-list structure and assign signal flow directions between circuit components. Such structure and signal flow relationship are then used to generate a layout placements and control subsequent optimization procedure to obtain the resulting layout. The final evaluation of this approach has to be done through performance measurement of the final mask layout derived from the symbolic layout generated. Although this hasn't been completed at this stage, the plausibility of utilizing knowledge of MOS circuit for generating cell layout is demonstrated.

References

[1] N. P. Jouppi. Derivation of Signal Flow Direction in MOS VLSI. *IEEE Transactions on Computer-Aided Design*, CAD-6(3):480–490, 1987.

[2] J. H. Kim. *Use of Domain Knowledge in Computer Aid for IC Cell Layout Design.* PhD thesis, Carnegie-Mellon University, 1985.

[3] P. Kollaritsch and N. Weste. A Rule-Based Symbolic Layout Expert. *VLSI Design*, 62–66, Aug. 1984.

[4] Y. S. Lin and D. D. Gajski. LES: A Layout Expert System. In *Proceedings, 24th ACM/IEEE Design Automation Conference*, pages 672–678, 1987.

[5] W. Lue and L. McNamee. PLAY - Pattern-Based Symbolic Cell Layout. In *Proceedings, 24th ACM/IEEE Design Automation Conference*, pages 659–665, 1987.

[6] D. V. McDermott. *Assimilation of New Information by A Natural Language-Understanding System.* Technical Report AI-TR291, MIT AI Lab., 1974.

[7] L. W. Nagel. *SPICE2: A Computer Program to Simulate Semiconductor Circuits.* Memo ERL-M520, University of California, Berkeley, 1975.

[8] A. Newton. Symbolic Layout and Procedural Design. In G. D. Micheli, A. Sangiovanni-Vincentelli, and P. Antognetti, editors, *Design Systems for VLSI Circuits*, pages 65–112, Martinus Nijhoff, 1987.

[9] R. Otten. Automatic Floor-Plan Design. In *Proc. 19th Design Automation Conf.*, pages 261–267, June 1982.

[10] V. B. Rao and T. N. Trick. Network Partitioning and Ordering for MOS VLSI Circuits. *IEEE Trans. on Computer-Aided Design*, CAD-6(1):128–144, Jan. 1987.

[11] J. Soukup. Circuit Layout. *Proc. of the IEEE*, 69(10):1281–1304, 1981.

[12] R. Stallman and G. Sussman. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artificial Intelligence*, 9:135–196, 1977.

[13] D. Waltz. Understanding line drawings fo scenes with shadows. In P. Winston, editor, *The Psychology of Computer Vision*, McGraw-Hill, 1975.

[14] H. Weinblatt. A New Search Algorithm for Finding the Simple Cycles of A Finite Directed Graph. *J. ACM*, 19(1):43–56, Jan. 1972.

[15] N. Weste and K. Eshraghian. *CMOS VLSI Design.* Addison-Wesley, 1984.

[16] O. Wing, S. Huang, and R. Wang. Gate Matrix Layout. *IEEE Trans. on CAD*, CAD-4:220–231, July 1985.