

# Exclusion Constraints, a new application of Graph Algorithms to VLSI Design

Kevin Karplus  
Computer Science Department  
(also in the School of Electrical Engineering)  
Cornell University  
Ithaca, NY 14853

*This paper presents a new paradigm for analyzing digital MOS circuits, based on boolean expressions for paths in the switch graph. A new circuit representation, the inverter graph, is described.*

*A new set of electrical design rules are introduced, based on path expressions in the switch graph and cycles in the inverter graph. A program has been written that generates the exclusion constraints implied by the rules.*

## Introduction

This paper presents a new set of design rules for switch circuits. Since the rules are based on graph algorithms and no information is needed about switch sizes or parasitics, the rules can be applied before layout. Problems are spotted early in the design, before corrections become expensive. The six rules in this paper are simple enough to be applied automatically.

Two graphs are used in checking the rules: the switch graph and the inverter graph. The switch graph is input by the user and the inverter graph is derived from it. Each graph is explained in more detail below.

The rules apply to both nMOS and cMOS circuits, and use a simple switch model of the circuits. Some useful circuits violate the rules, but these circuits are often not handled correctly by current CAD tools. By pointing out the places where the implicit assumptions of the tools are violated, the rules can focus attention on those parts of the circuit that need more detailed modeling.

Each rule generates boolean equations (*exclusion constraints*) that should be satisfied by the signals in the circuit. A program has been written to generate the constraints defined by the rules. Although the program can be applied to entire chips, it is probably better applied to individual sub-circuits before they have been assembled.

The next two sections of the paper discuss the switch graph and the inverter graph. The section after that discusses the representation of boolean expressions. The six rules are then introduced, and some comments are made about the program that generates constraints

## Switch Graph and Path Expressions

The first five rules are checked on a *switch graph*, in which each signal is a vertex, and each switch is an edge connecting the source and drain. Edges are labeled with the signal on the gate of the switch (negated if the switch is pMOS). For switches in parallel, the multiple edges are merged into a single edge whose label is the OR of the labels for each switch. Static load devices (pullups) are included in the switch graph, but are specially marked as they need different treatment. A path in the switch graph represents a possible DC connection between two signals. Figure 1 shows a CMOS nand-gate and its switch graph.

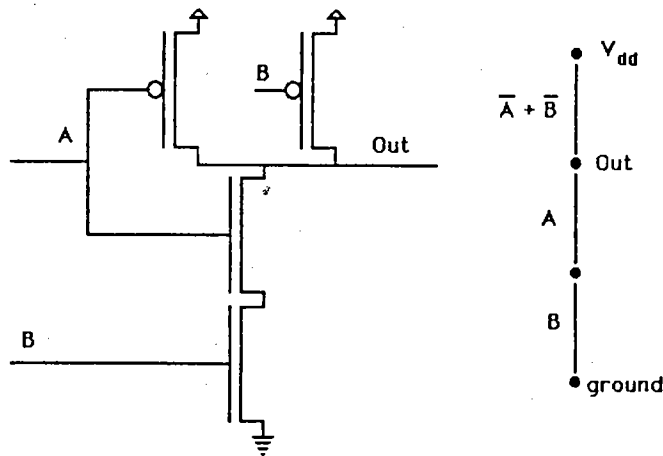


Figure 1: CMOS nand-gate and its switch graph.

The switch graph is a standard representation for MOS circuits, used by switch simulators, electrical rule checkers, and other tools. For compatibility with other tools, the exclusion constraint generator uses esim[8] format for input of the switch graph.

The rules are based on boolean expressions that summarize paths in the graph. Certain expressions are so useful that they are pre-computed for all nodes. These expressions are the ones that summarize all paths from any node  $n$  to  $V_{dd}$  ( $V_n$ ), ground ( $G_n$ ), a pullup ( $P_n$ ), a high input ( $In1_n$ ), or a low input ( $In0_n$ ).

The expressions can be quickly generated using LU-decomposition on the adjacency matrix for the switch graph[7]. Each element of the matrix is the label for the edge between the corresponding vertices ( $M_{i,j} = \text{label}(i,j)$ ). Since each vertex is trivially connected to itself, we add the diagonal ( $M_{i,i} = 1$ ). Because each switch is bi-directional, the graph is undirected and the matrix is symmetric.

Each element of  $M$  is an expression for paths of length 0 or 1 between the corresponding vertices. All paths from any vertex to any other vertex could be found by taking the transitive closure of  $M$ , but the resulting matrix may contain expressions too large for practical applications. We can find all paths to a particular node more cheaply. Let  $E_i$  be the vector with all zeros except a 1 in the  $i^{\text{th}}$  position. If we solve  $Mx = E_i$ , replacing each multiplication with an AND and each addition or subtraction with an OR, the solution vector will consist of all paths to vertex  $i$ . That is,  $x_j$  will be all paths from  $j$  to  $i$ .

When generating paths, we are not interested in paths that go through ground or  $V_{dd}$ . One way to eliminate such spurious paths from consideration would be to make the edges touching the power supplies directed edges (for example, edges might be directed out of  $V_{dd}$  and into ground). Unfortunately, directing edges makes the adjacency matrix for the graph asymmetric, and sparse matrix algorithms are much simpler for symmetric matrices (Cholevsky factorization rather than general LU-decomposition). This is the only instance I know where the bi-directionality MOS switches makes a CAD algorithm simpler.

The approach taken in the constraint generator is to partition the edges of the switch graph into four sets: the edges incident on ground, the pullups, the other edges incident on  $V_{dd}$ , and all other switches. The last group of switches are all potentially bi-directional, so have a symmetric adjacency matrix. LU-decomposition is done only on the symmetric matrix. To compute paths from all nodes (except  $V_{dd}$ ) to ground we solve  $MG = G_{\text{switch}}$  for  $G$ , where each  $G_{\text{switch}_i}$  is the label on the edge from  $i$  to ground. Paths from all nodes to  $V_{dd}$  can be computed similarly ( $MV = V_{\text{switch}}$  and  $MP = \text{Pullup}$ ). The

paths from  $V_{dd}$  to ground can be computed by either the dot product  $G \cdot V_{switch}$  or  $V \cdot G_{switch}$ . Note that the adjacency matrix needs to be factored only once, but that several right-hand sides may need to be solved for.

The constraint generator currently uses a slightly modified version of SPARSPAK[2] to do the matrix factoring and the solving to get path expressions. Eventually the algorithms will be re-written in C, so that the adjacency matrix and temporary arrays can be dynamically allocated. The SPARSPAK sparse matrix representation is compact and allows rapid solving for path expressions. Unfortunately, finding all neighbors of a given node (necessary for the charge-sharing rule) is slow. Furthermore, the program uses the SPARSPAK interface to build the representation, so the matrix itself is not accessible. Re-writing the sparse matrix algorithms in C should alleviate the second problem, and finding all neighbors will probably not be the bottleneck in the final tool.

Two considerations are important when using sparse matrix techniques: how sparse is the original matrix? and how sparse are the factors? Our original matrices are very sparse, with approximately as many edges as vertices. Although the degree of some nodes is high (buses for example), most nodes have degree zero or one after the  $V_{dd}$  and ground switches have been removed.

*Fill-in*, the difference between the number of non-zeros in the factors of the matrix and the original matrix, is a widely studied measure for sparse matrix methods. Vertex-ordering techniques are used with sparse matrix methods to try to minimize fill-in. One popular technique, quotient minimum degree (QMD), is a simple greedy method that chooses vertices one at a time, always picking the one that introduces the least immediate fill-in. Since choosing a degree 0 or 1 node in a quotient graph introduces no fill-in, QMD generates no fill-in for forests. After removing  $V_{dd}$  and ground switches, the switch graph is often a forest. Besides trees, one expects to find series-parallel graphs as components of switch graphs. When starting from a series-parallel graph, QMD never has to choose a vertex with degree higher than two, so the fill-in at most doubles the density of the matrix. (Sketch of proof: a series-parallel graph always has a vertex of degree 2 or less. The quotient graph after selecting a degree one or two vertex from a series-parallel graph is again series-parallel.)

Despite all the nice properties of QMD, the constraint generator

use  
nes  
alge  
but  
tho  
mo:  
pro  
and  
squ  
to r  
tha  
at r  
sma

In

for  
diff  
com  
as s  
to 1  
gra

K

eve:  
labo  
of s  
whe  
min

