COMPUTATIONAL MESH WITH PULL-UPS

William Bricken
January 2002

FUNCTIONAL MODEL ILLUSTRATED TOUR

Illustrations of functional Comesh circuits follow. They include the mesh architecture, the most simple logic gates and registers, common three input circuits, and a brief example of Comesh composition.

CONTENTS

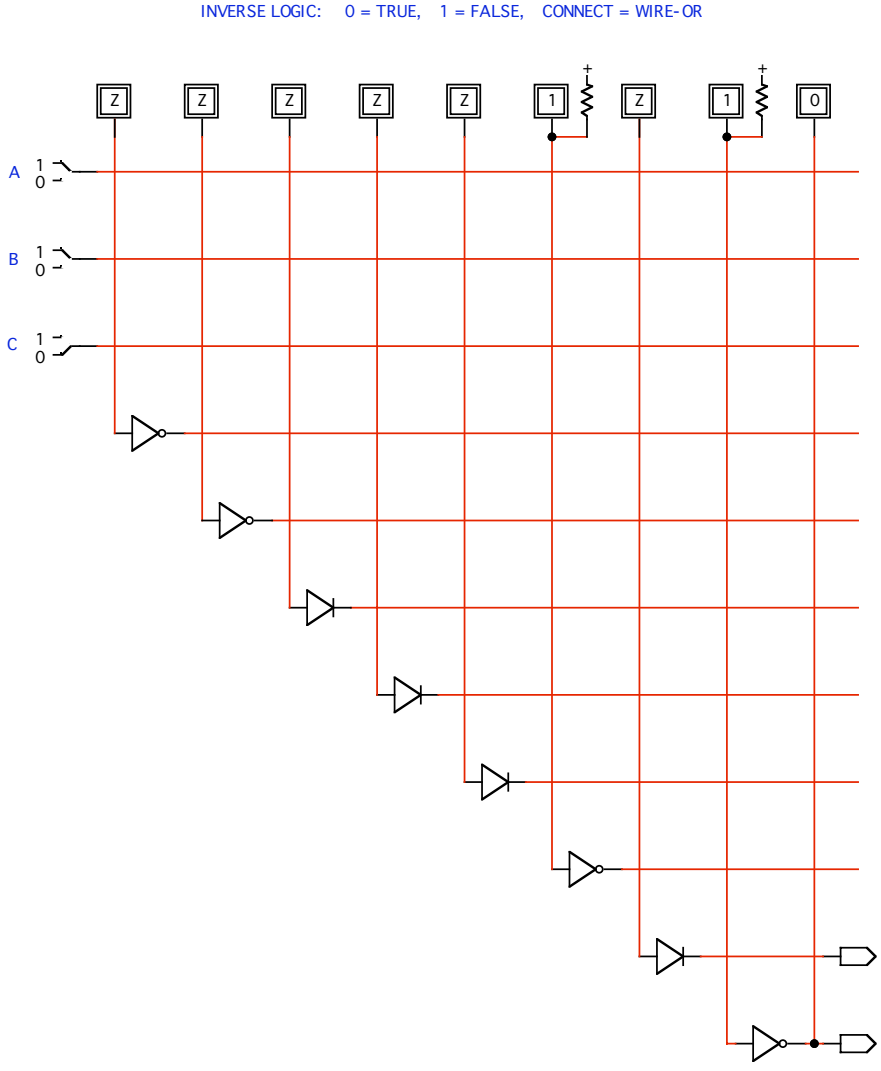
Mesh

The mesh consists of an orthogonal array of potential interconnect points. It is a reconfigurable triangular matrix with inverters on the diagonals.

Multiple connections on a column wire is wire-OR. A wire-OR line is pulled-up to 1. All diagonal inverters of rows that are wire-ORed are Open-Collection Buffers (OCB). Single connections to a column wire are simple negations, converting the wire-OR column to a NOR.

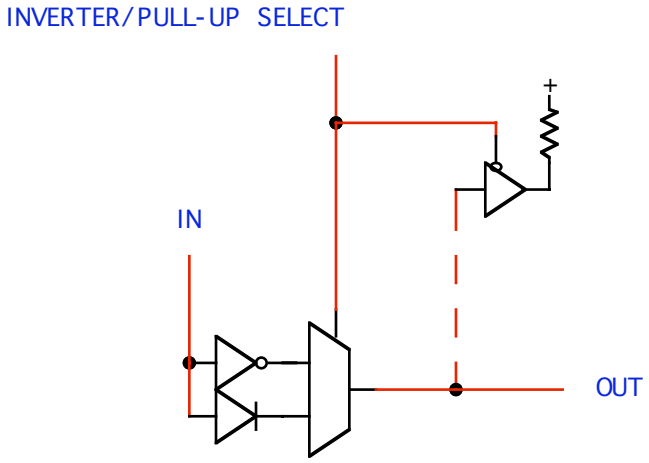
Multiple connections on a row are fanout of the row signal. Only a row signal of 1 creates fanout into columns.

I/O value	*Meaning*	*Wire-OR*	*Pull-up column*
0	True	pull-down	pulled-down
1	False	Z	pulled-up



Mesh Diagonal Element

For reconfigurability, each column line can be accessed from its accompanying row via an ordinary NOT or via an OCB. For one connection on the column line, NOT is used. The switch which chooses OCB also activates the pull-up resistor on the column line.



Comesh uses *negative logic* at input and output.

0 line-value is True
1 line-value is False

Incoming and outgoing signals from the Comesh can be inverted to return to positive logic.

Multiple column line connections are *wire-or* under the negative logic interpretation. In negative logic, a 0 signal value will pull-down a column value. Each 1 signal will be converted into Z by the OCB. When all row connections are Z (i.e. none are 0), the column pull-up resistor will set the column signal value to 1. When a column is pulled-up to 1 (as False), it becomes irrelevant as the <void>.

Crosspoints

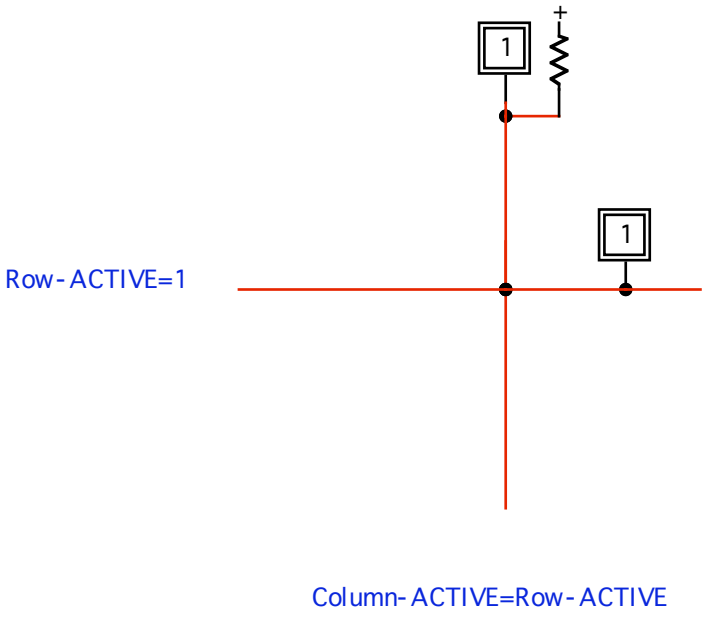
Crosspoints use negative logic: 0 = True, 1 = False.

Crosspoints are indicated symbolically by a dot at the intersection of a row and a column (crosspoint-cell value = 1). Non-dotted intersections are not connected (crosspoint-cell value = 0)

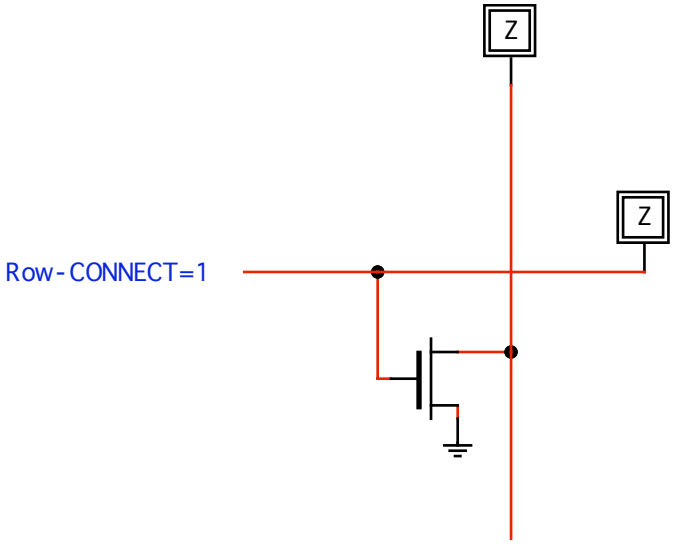
Row crosspoints spread a particular signal value to multiple columns. Column crosspoints form a logical wire-OR. In boundary terms, row crosspoints identify the containers of the row output, column crosspoints indicate contents of the column input.

In a void-based model, any 0 row value propagating through an open crosspoint (encoded as 1) will dominate (Dominion) the column of a crosspoint. Other potential inputs to the column are irrelevant. The dominant signal will propagate to the column diagonal, where it is inverted to 1 or to Z depending upon the type of diagonal connection. When all row input values to a column equal Z, the pull-up on the column will set the column value to 1.

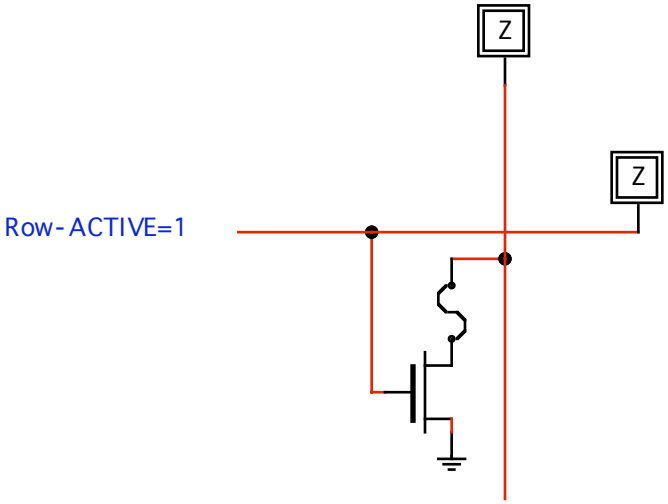
Generic Crosspoint



ROM Crosspoint



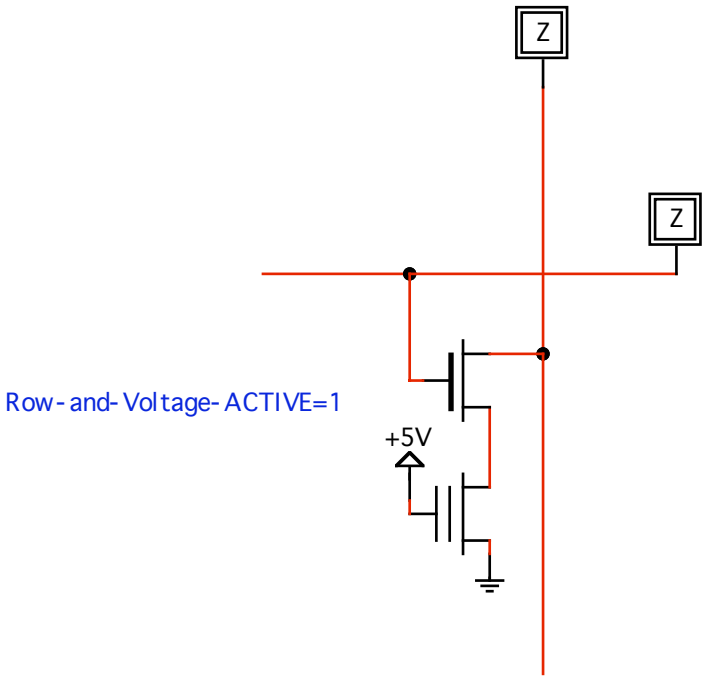
PROM Crosspoint



Column-ACTIVE=Row-ACTIVE

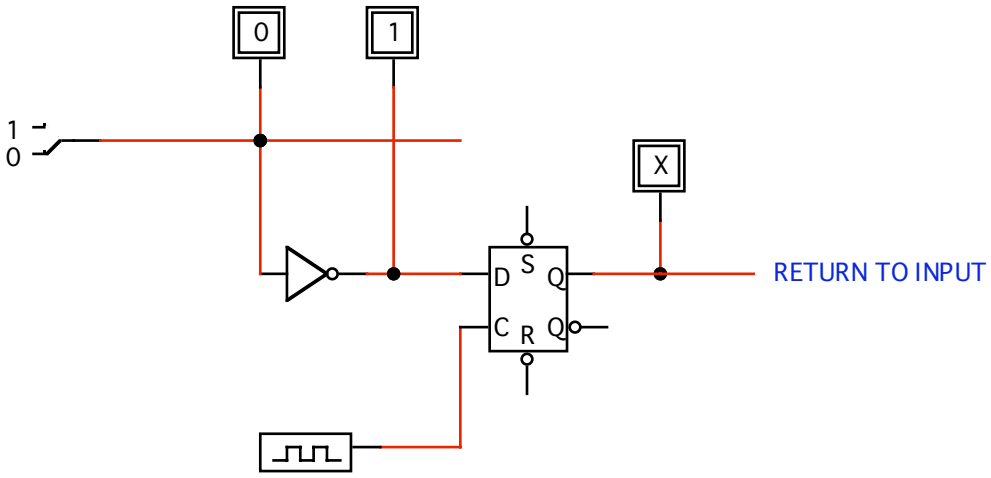
Fuse-INTACT=1

EEPROM Crosspoint

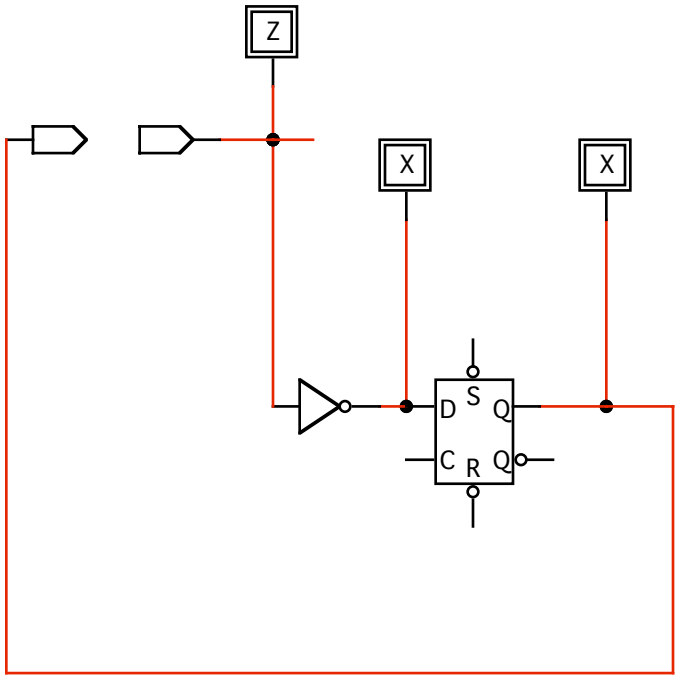


Registers

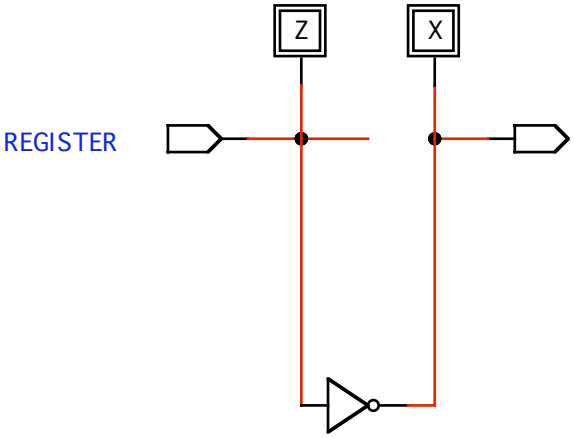
Each row that is either an output row or an abstract register row is terminated by a single register. In the case of an abstract register row, the output is the stored value of the abstract register, which is converted to the stored value in the actual terminal register. The registers are physically wired to input rows, to be reintroduced into the array on the next array processing cycle.



The feedback loop from a register row is indicated by a feedback wire into an input terminal associated with that particular register row. Register outputs connect only to the same register input and not to any other row.



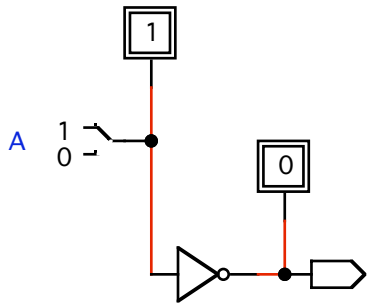
We will use a shorthand notation for the feedback loop, by indicating which row the output of a register should reenter. The loop wiring is implicit. The register below returns a signal to itself, after the signal has been processed through the connections in the array.



GATES

1-input NOR

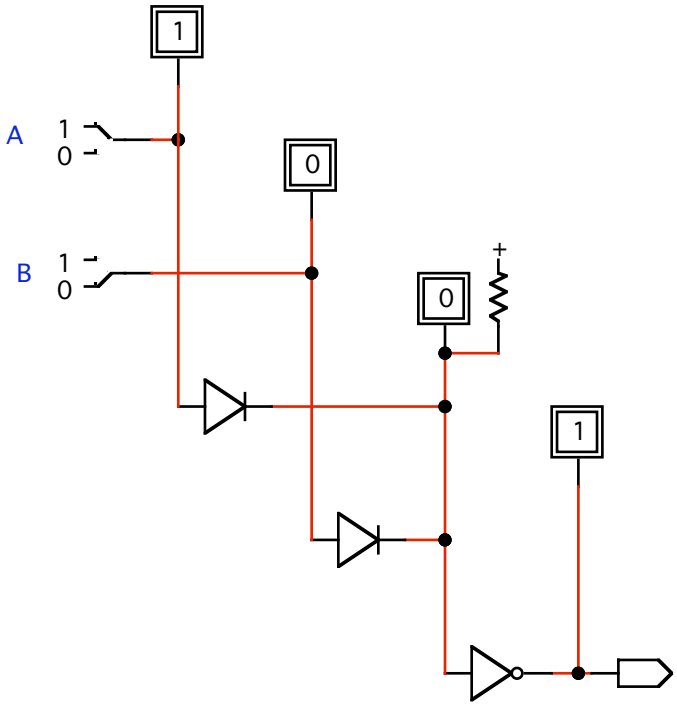
Parens: (A)



The configuration for elementary NOT.

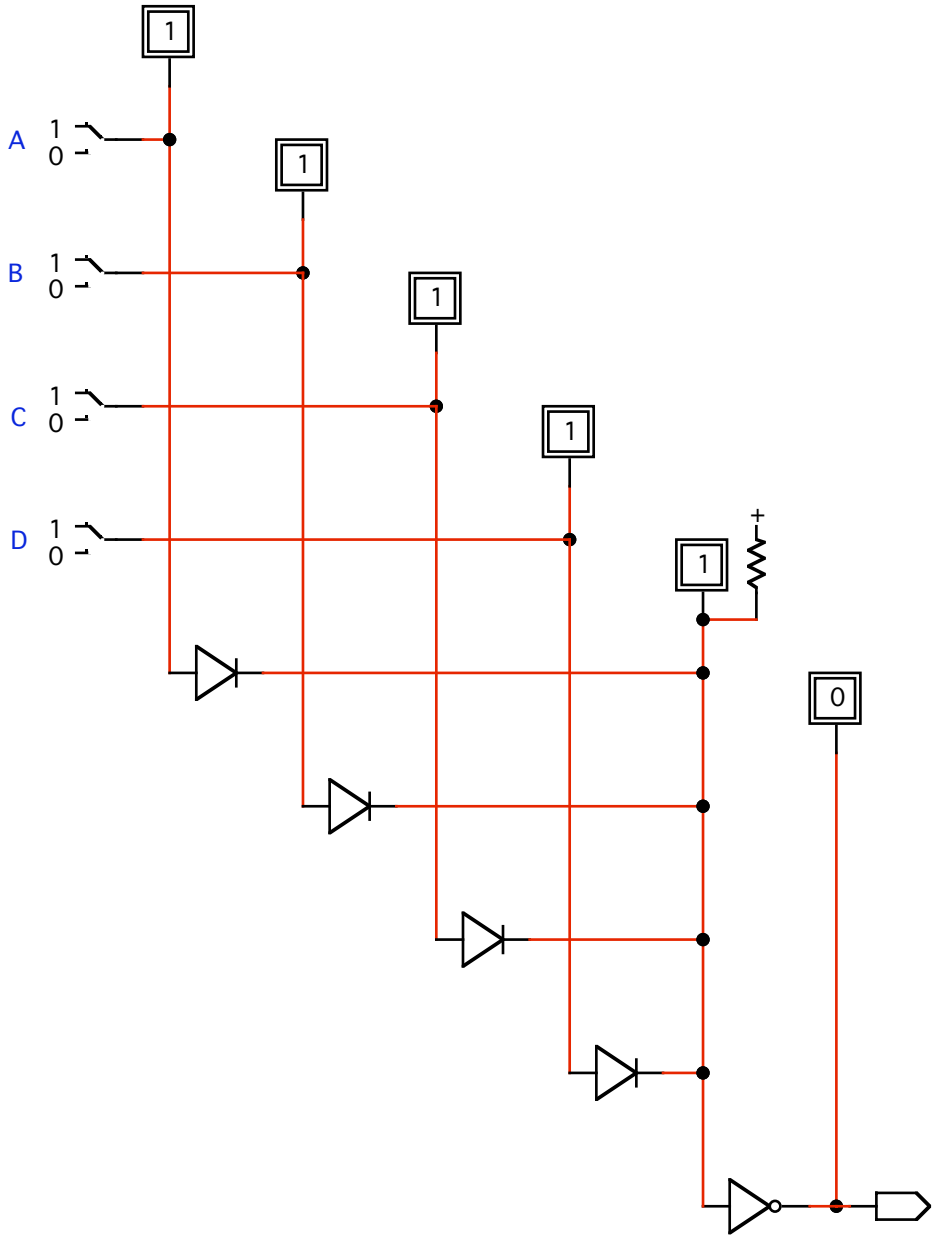
2-input NOR

Parens: (A B)



4-input NOR

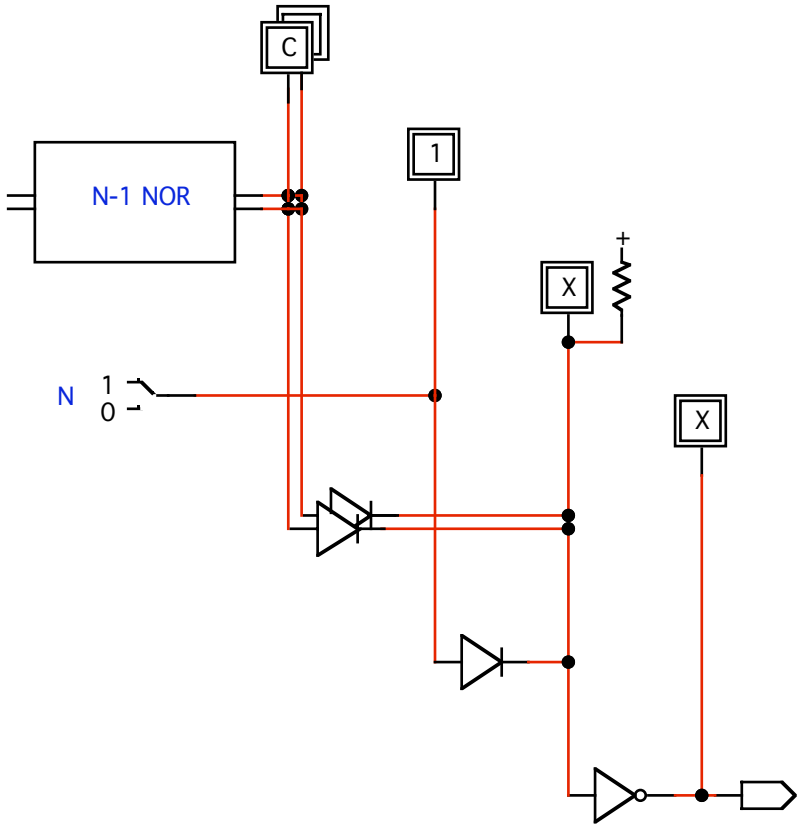
Parens: (A B C D)



Arity is simply the opportunity for any one of several row signals to dominate a column.

N-input NOR

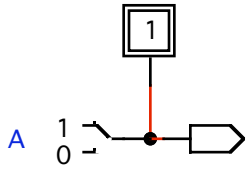
Parens: (..N-1.. N) This notation is descriptive, not functional.



The N-input NOR illustrates how to add another input to a wire-OR configuration.

1-input OR

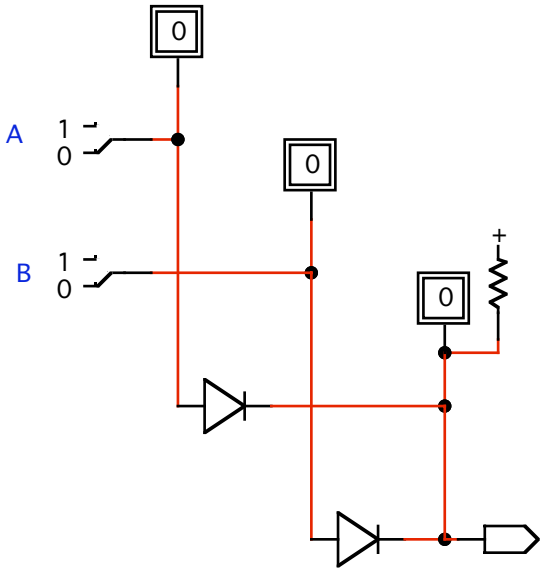
Parens: A



One-input OR is simply a wire.

2-input OR

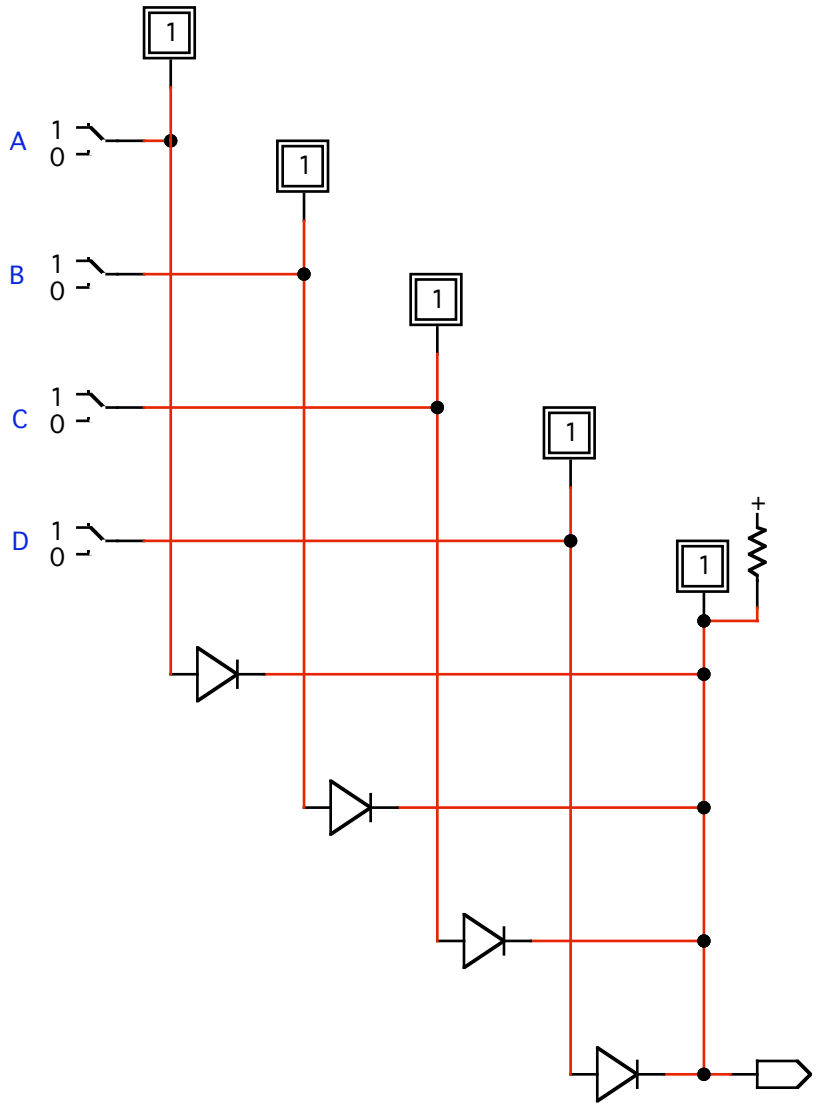
Parens: A B



The only difference between OR and NOR is that the final inverter of NOR is absent in OR.

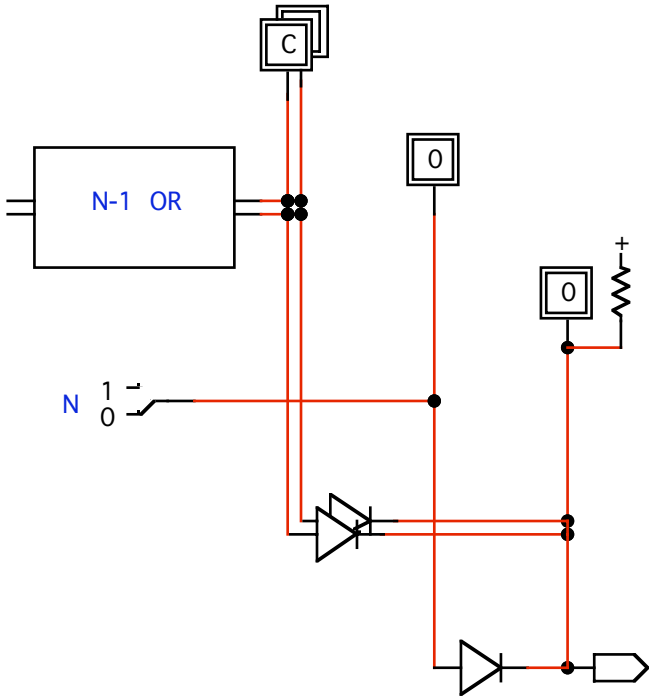
4-input OR

Parens: A B C D



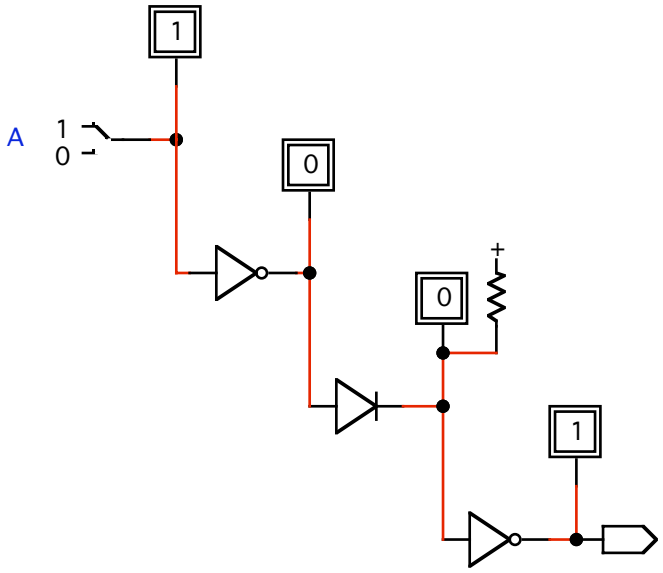
N-input OR

Parens: ..N-1.. N



1-input AND

Parens: ((A))

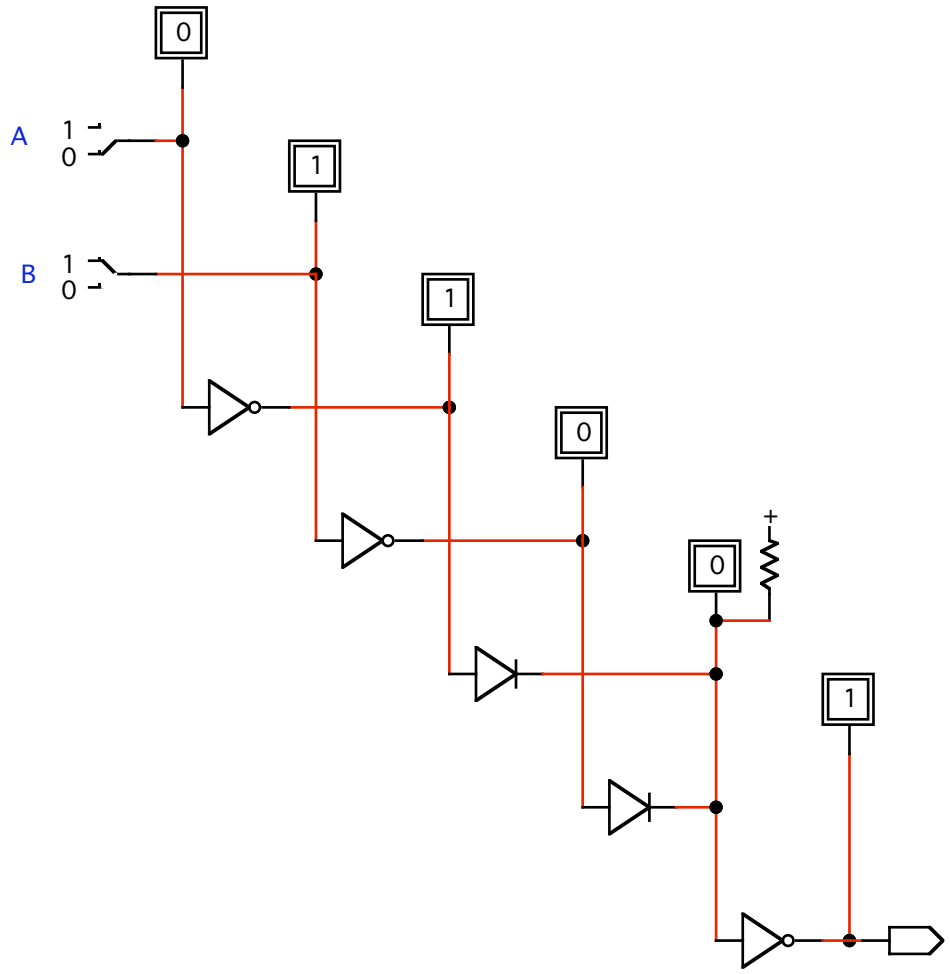


One-input AND is double negation.

Note the behavior of the pull-up column. When A is 1, the first inverter changes it to 0. The OCB propagates the 0 value and the final inverter changes it back to the original value of A. When A is 0, it is inverted to 1 and changed to Z by the OCB. Lacking a dominant signal on the pull-up column, the Z is passed as 1, which is changed to 0 by the final inverter. Essentially, pull-up columns are *transparent* to logical intent.

2-input AND

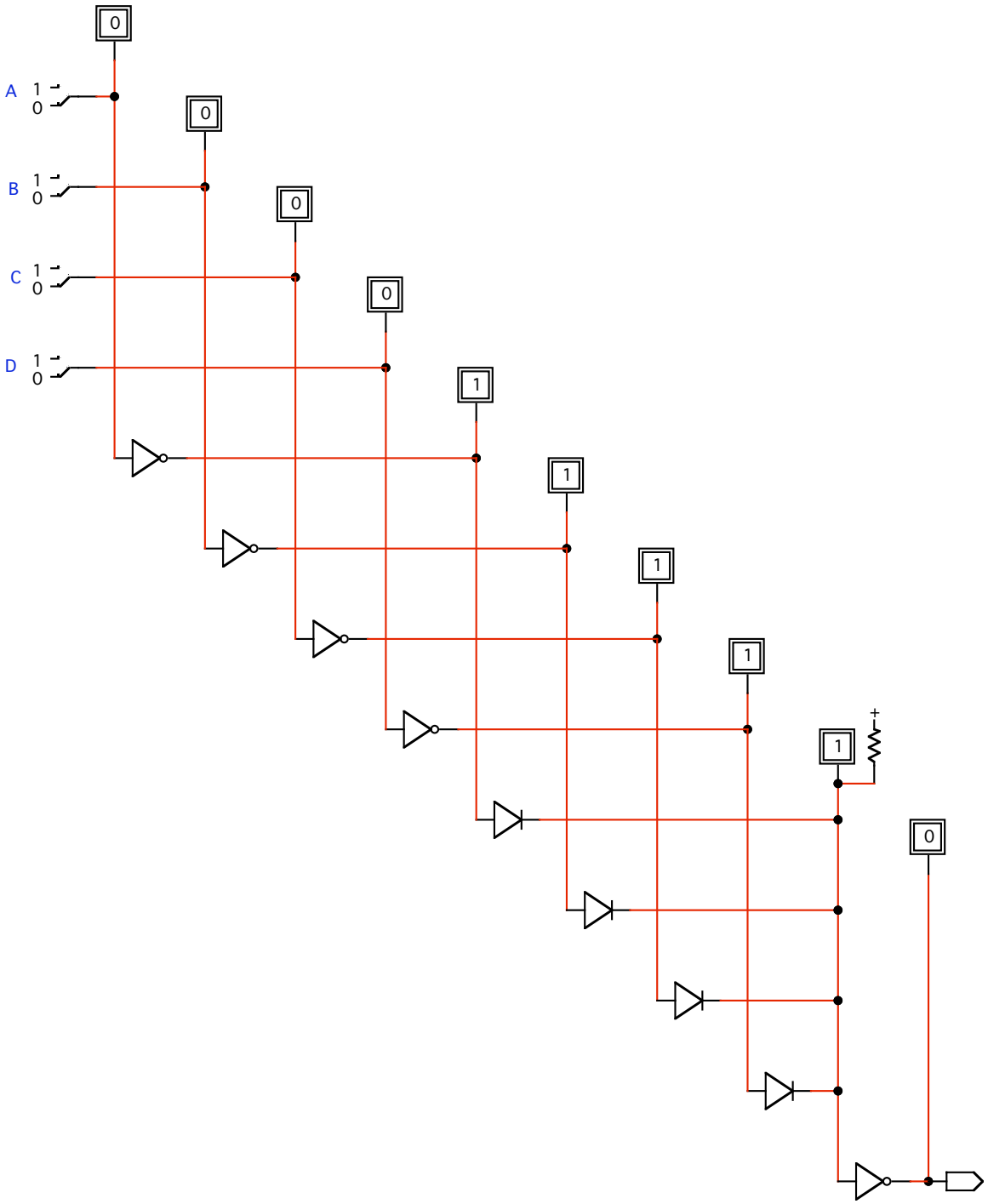
Parens: ((A)(B))



The difference between AND and NOR is that the AND inputs are first passed through inverters. Aside from the type of diagonal element and the doubling of row counts caused by input inversion, AND and NOR are structurally identical.

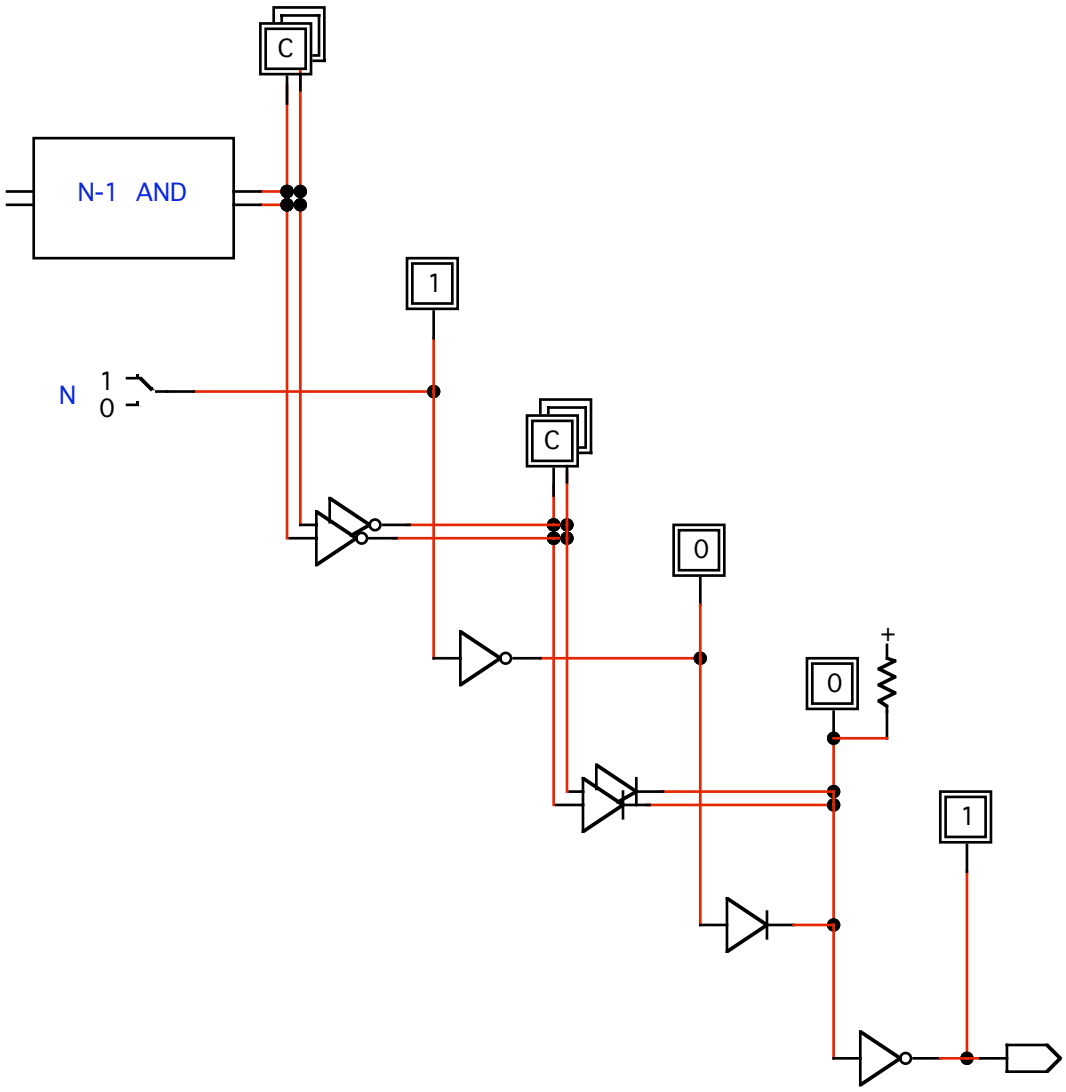
4-input AND

Parens: ((A)(B)(C)(D))



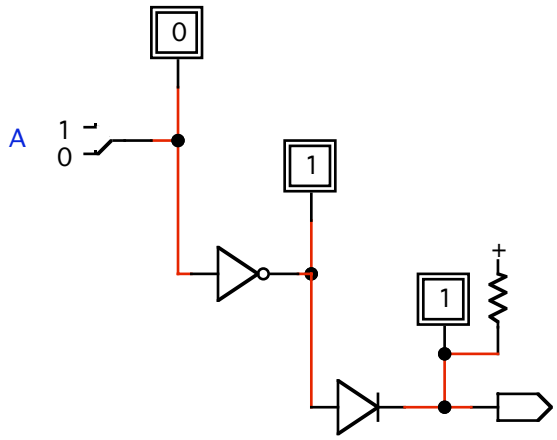
N-input AND

Parens: ((..N-1..) (N))



1-input NAND

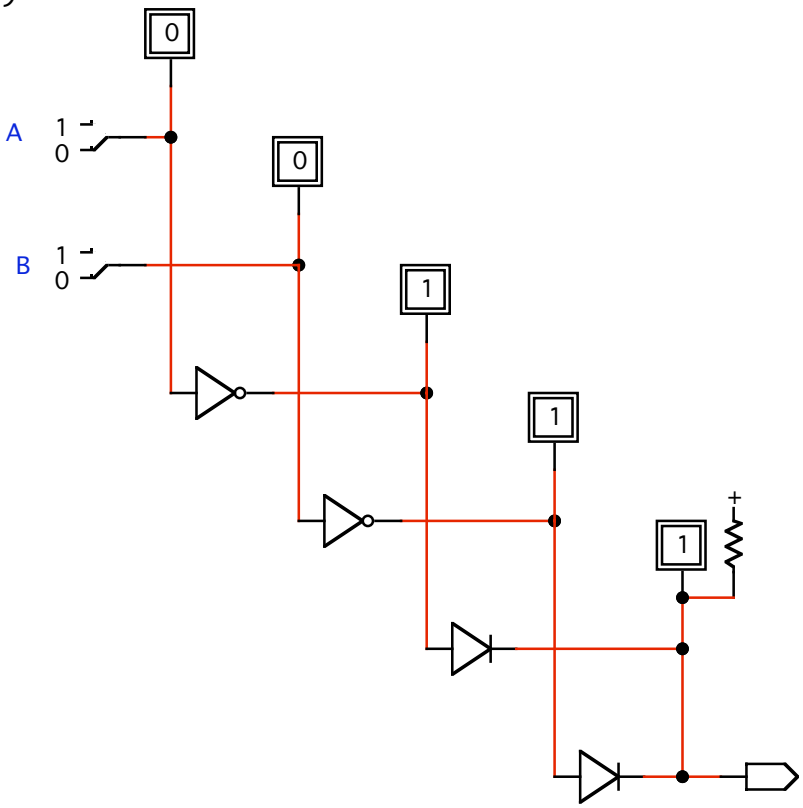
Parens: (A)



One-input NAND is a simple NOT, and is equivalent to a one-input NOR. This is a formal representation of the transparency of pull-up columns.

2-input NAND

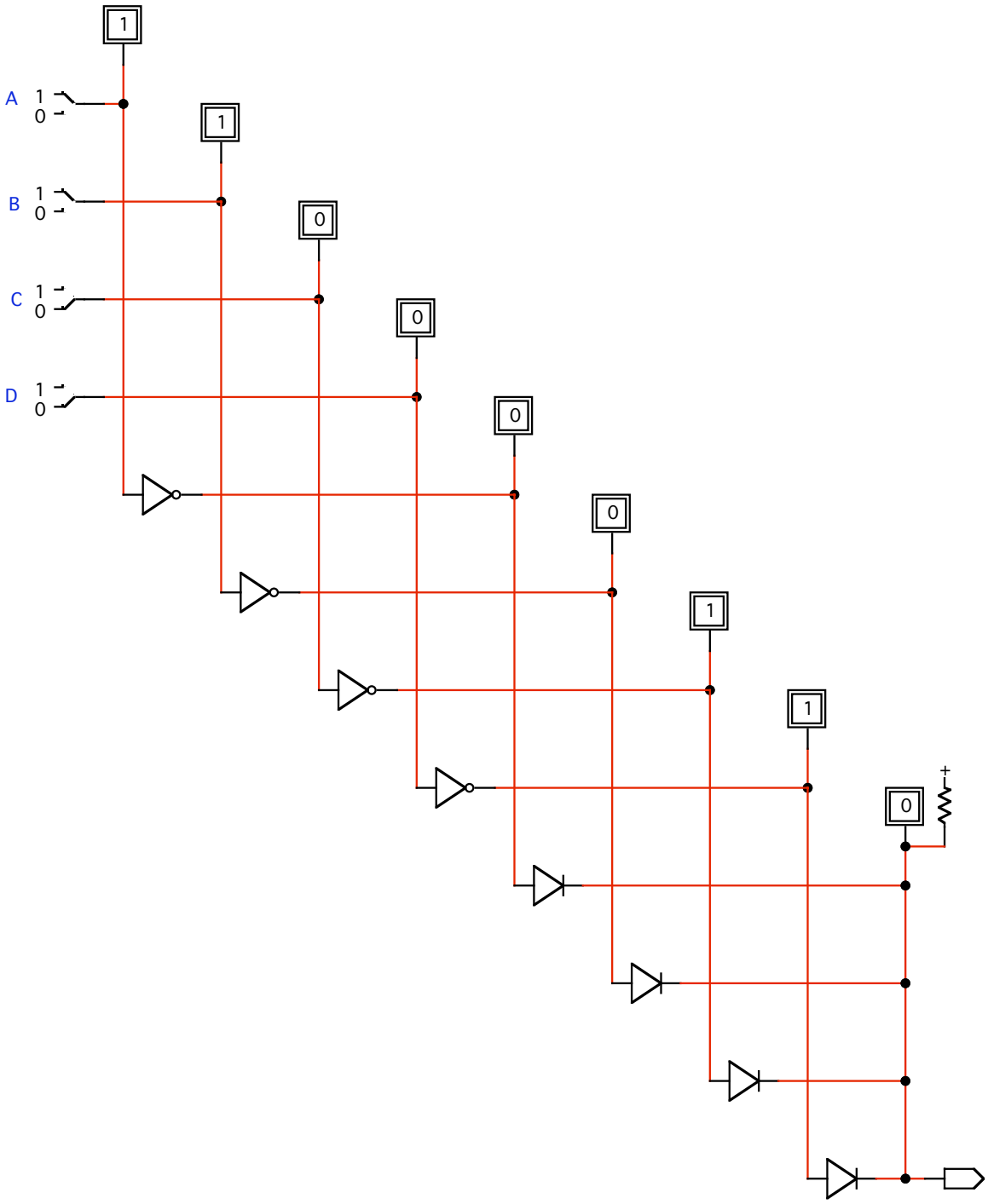
Parens: (A)(B)



The only difference between AND and NAND is that the final inverter of AND is absent in NAND.

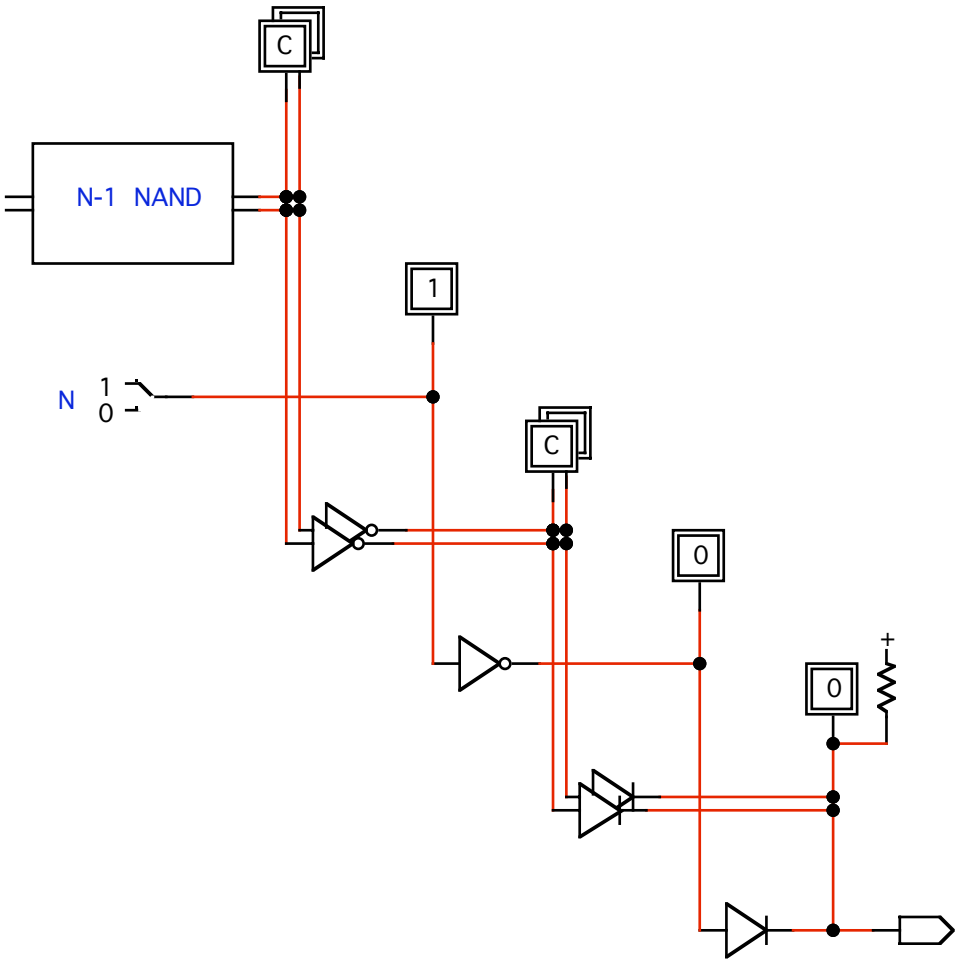
4-input NAND

Parens: (A)(B)(C)(D)



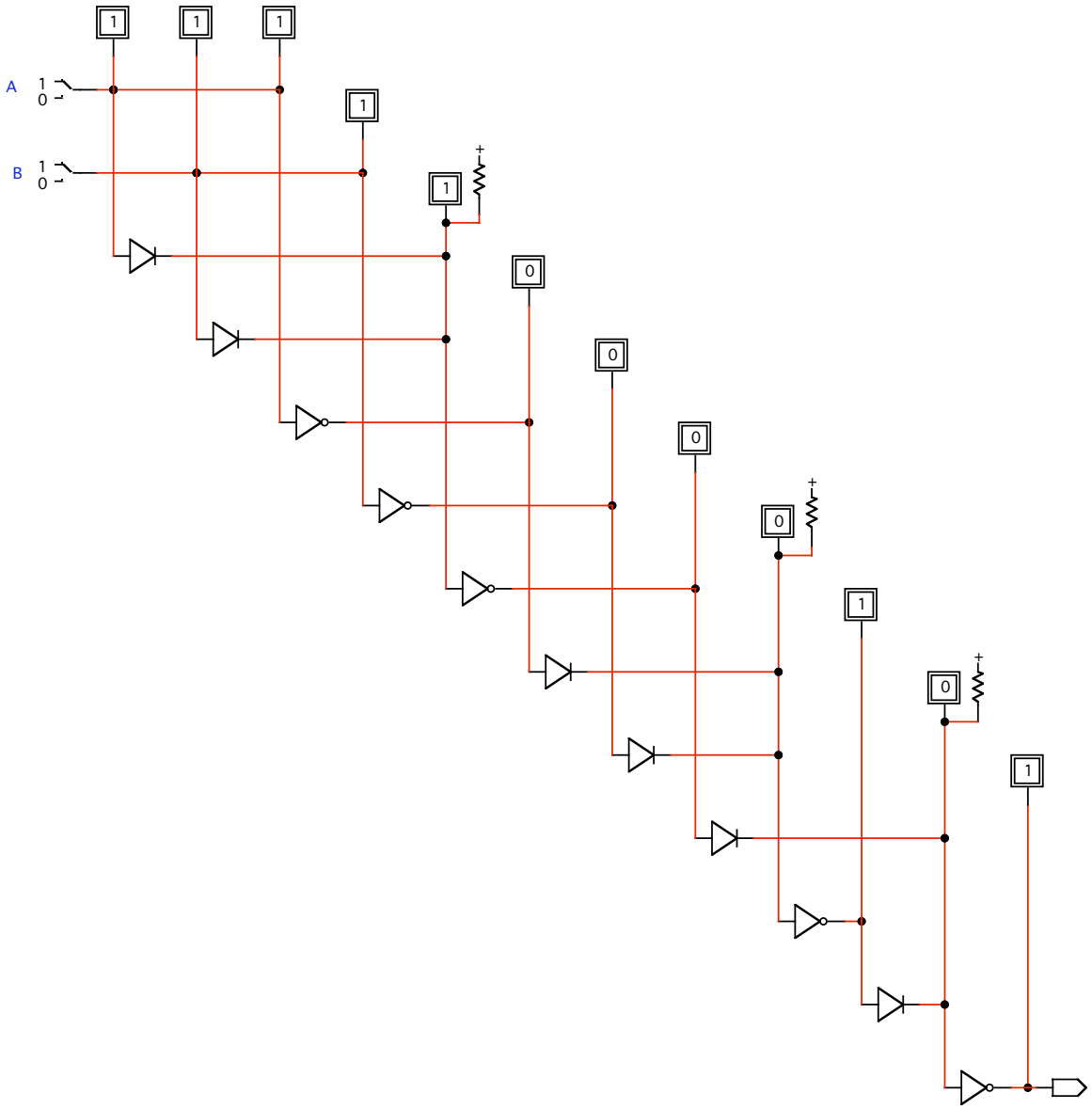
N-input NAND

Parens: (..N-1..)(N)



2-input XOR, Version I

Parens: ((A B)((A)(B)))

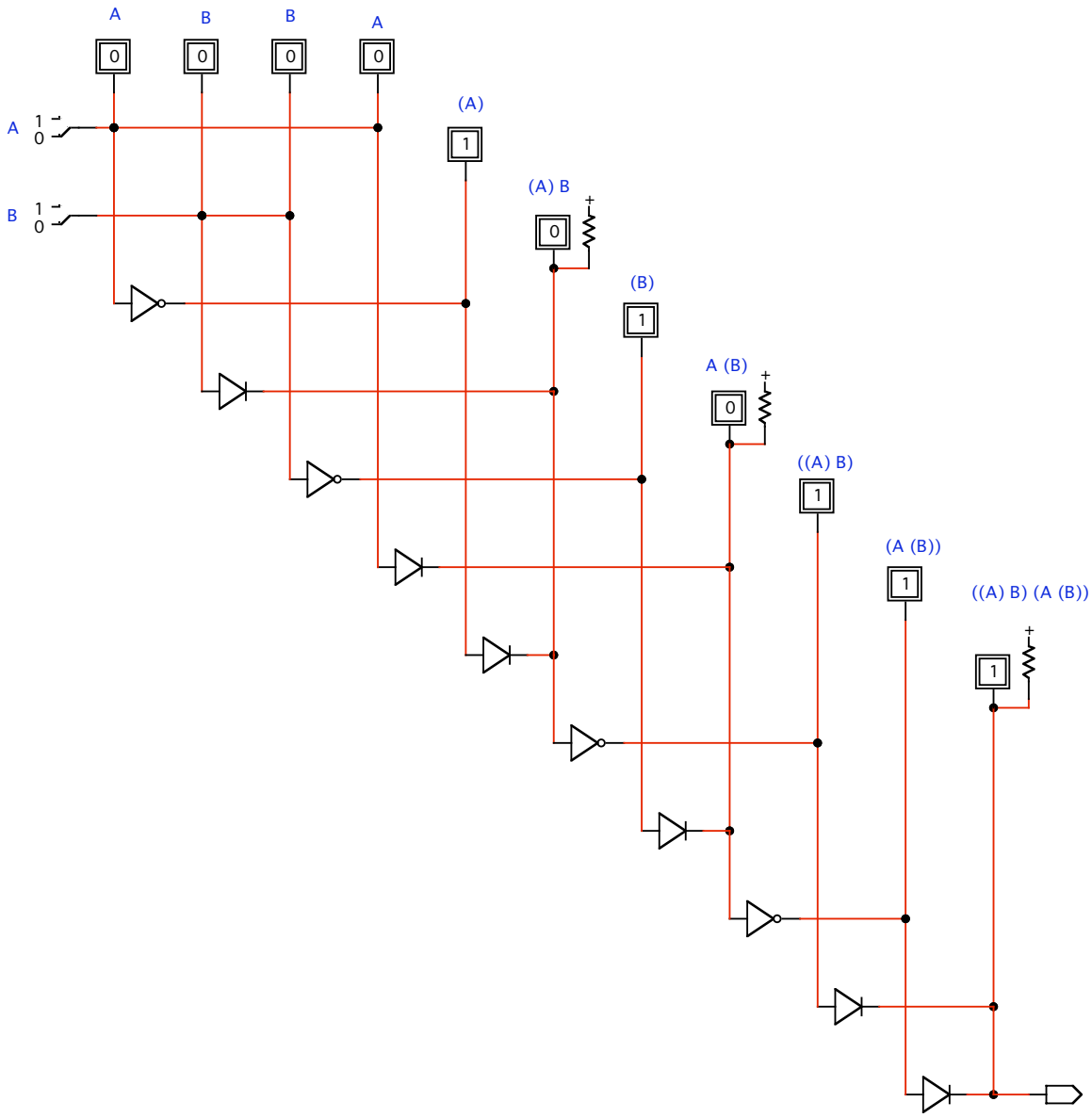


This XOR is the AND of NOR and NAND. The parens reads:

Both:
at least one of A or B
at least one of not-A or not-B

2-input XOR, Version II

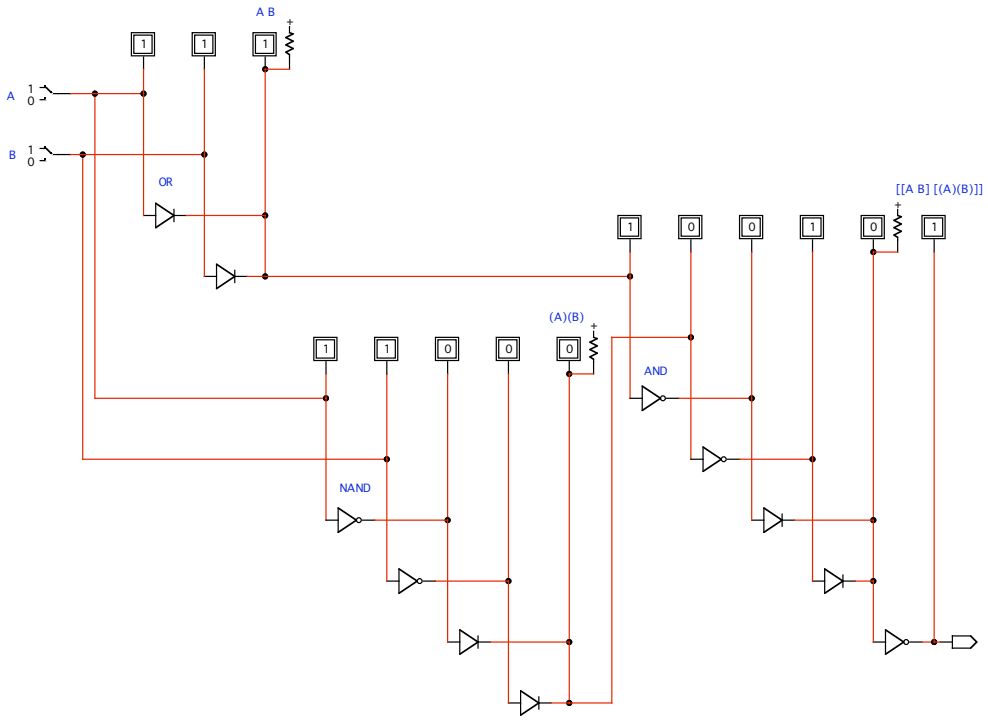
Parens: ((A) B) (A (B))

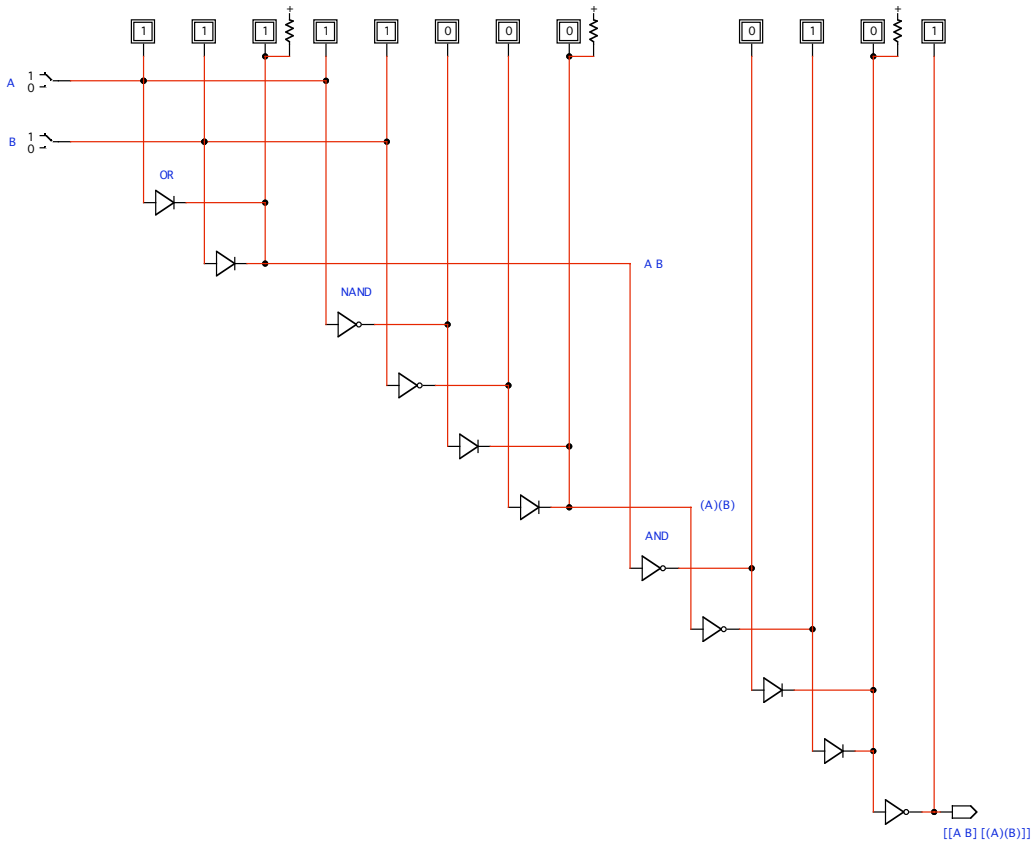


An alternative version of XOR has one less column so is slightly more efficient. The parens reads:

One of two cases:
A does not imply B
B does not imply A

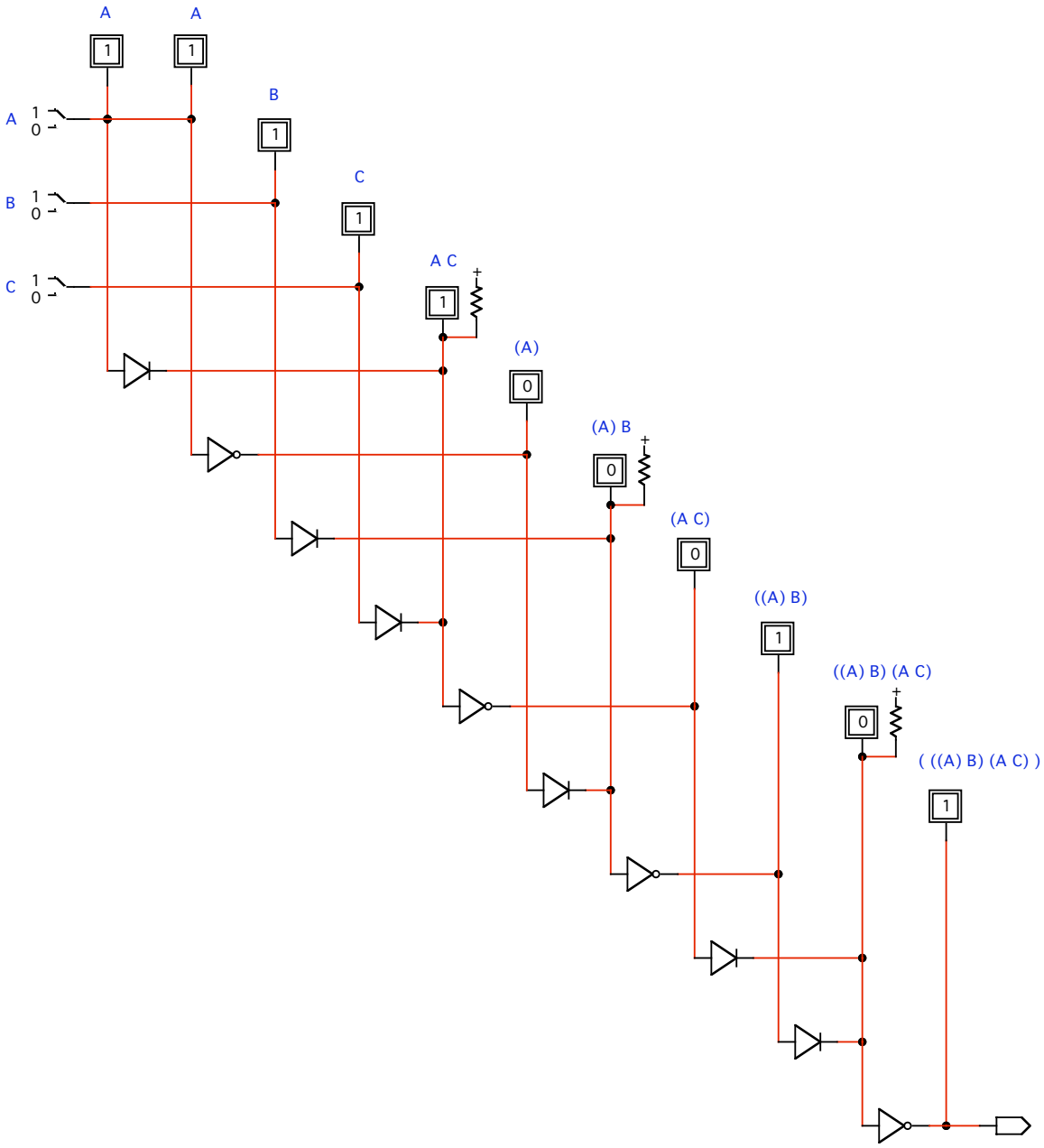
2-input XOR, composed of the AND of NOR and NAND





2to1 MULTIPLEXER (MUX)

Parens: (((A) B)(A C))



The parens reads:

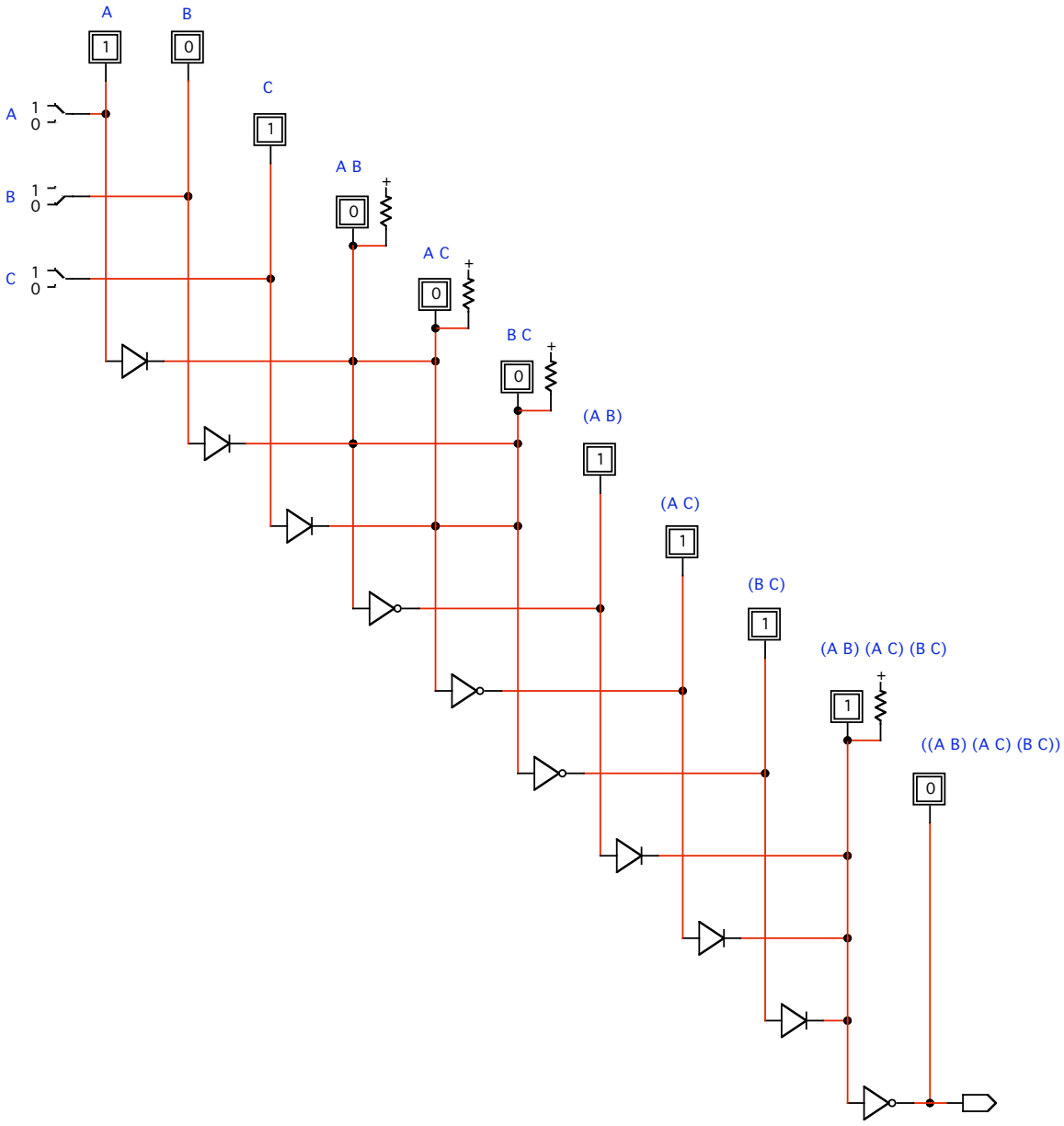
Both:

A implies B

at least one of A or C

2/3 MAJORITY, Version I

Parens: ((A B)(A C)(B C))

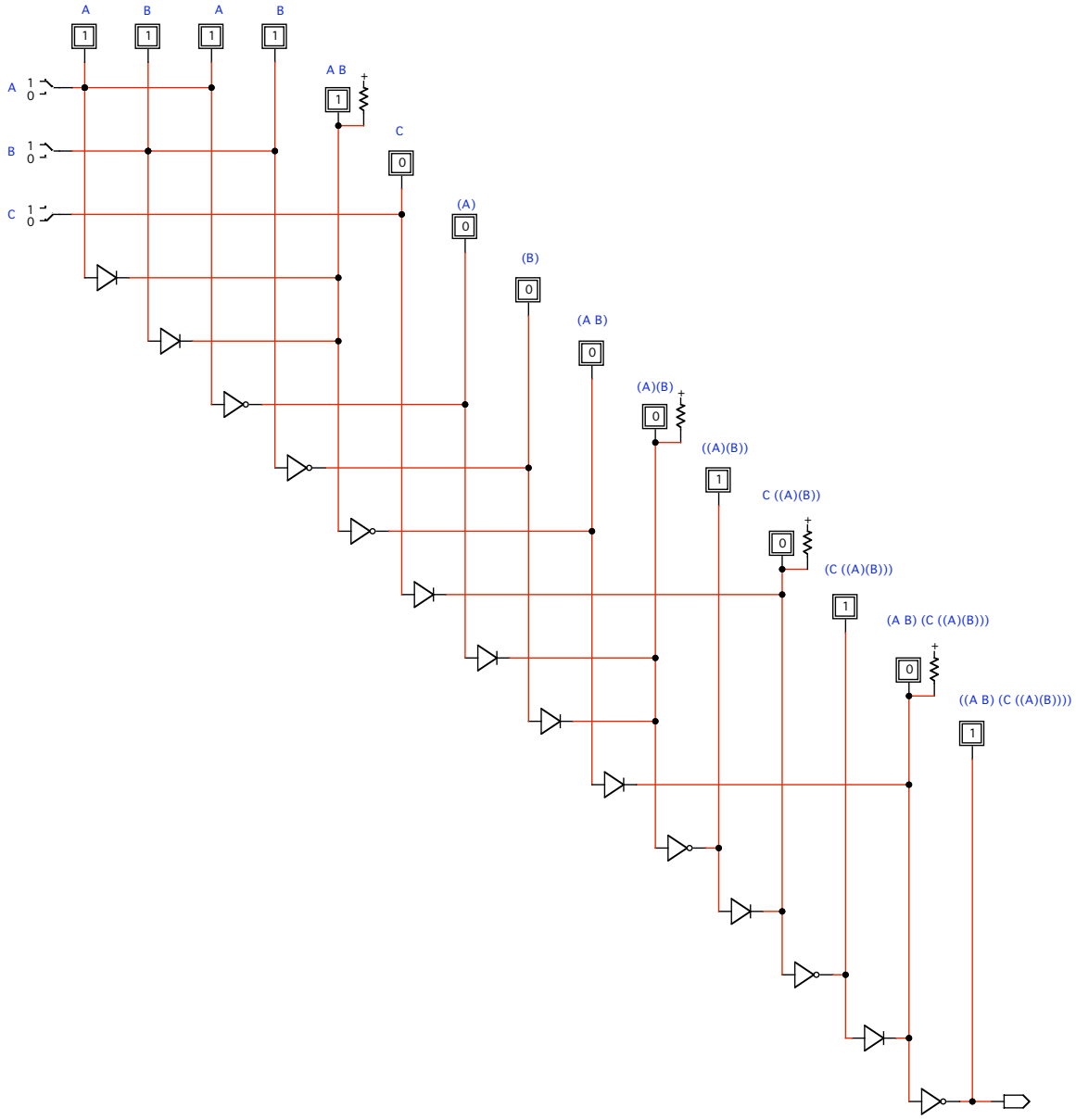


2/3 MAJORITY combines all pairs of incoming rows into another row. The set of connections in the three binary wire-OR columns requires a one-directional flow of current (or may lead to conflict). The parens reads:

All of
at least one of each possible pair.

2/3 MAJORITY, Version II

Parens: ((A B)(C ((A)(B))))

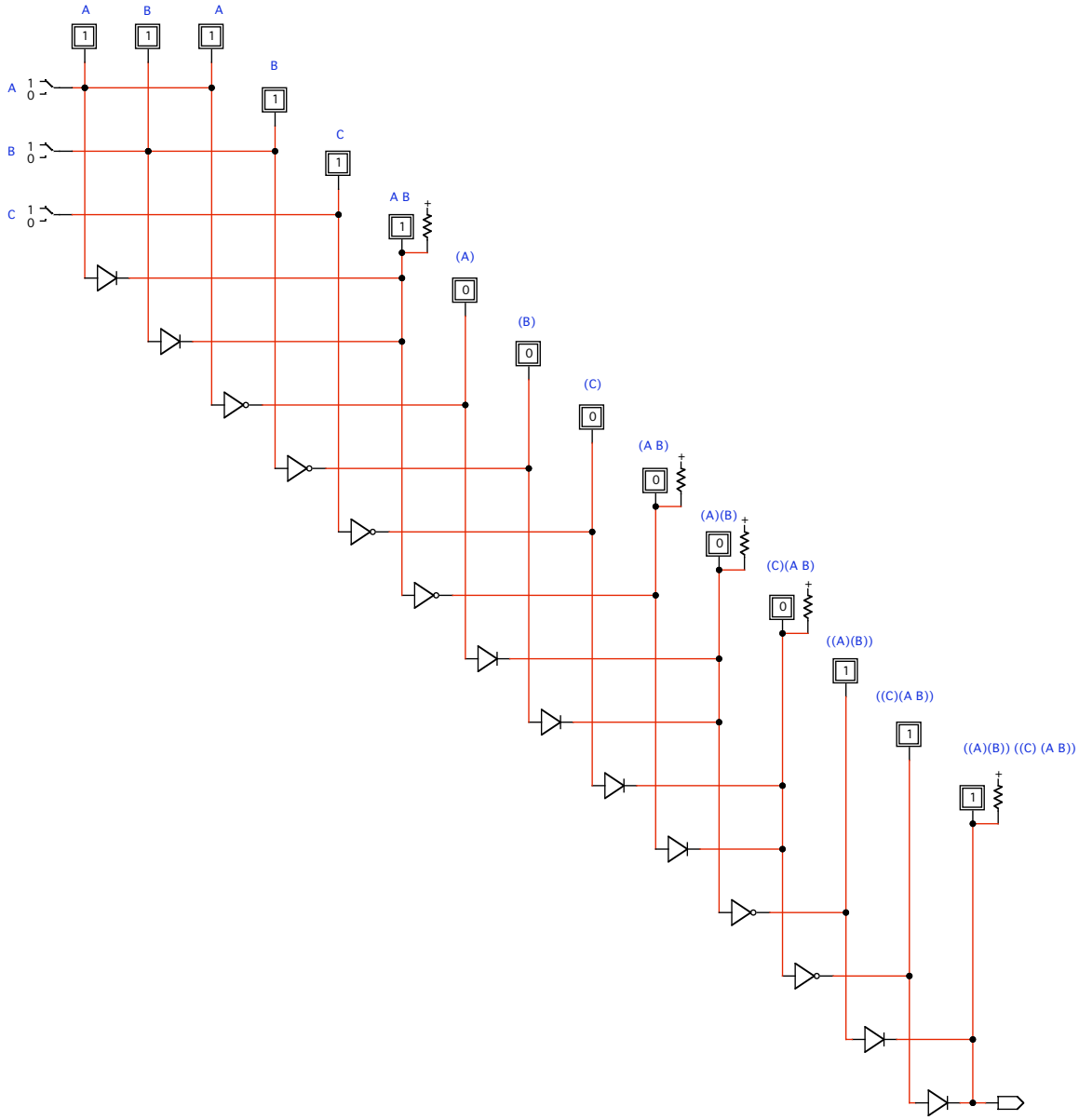


This 2/3 MAJORITY is the NXOR of A and B, partially distributed over C. The parens reads:

Both
at least one of A or B
at least one of C or (both A and B)

2/3 MAJORITY, Version III

Parens: ((A)(B))((C)(A B))

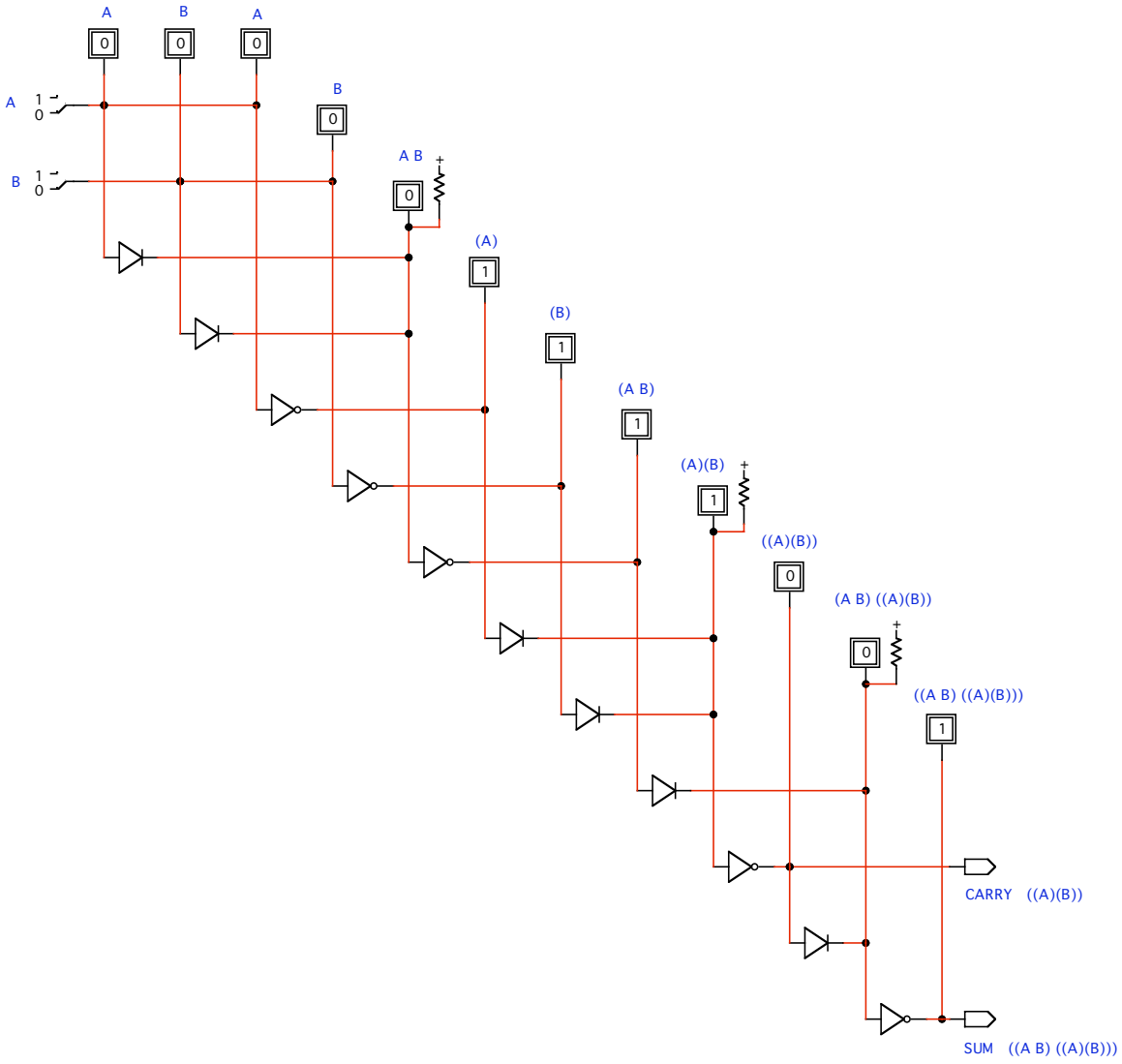


This 2/3 MAJORITY is the Flex of Version II. Here the AND of A and B is independent of the C value. The parens reads:

One of two cases:
both A and B
C and (at least one of A or B)

HALF-ADDER

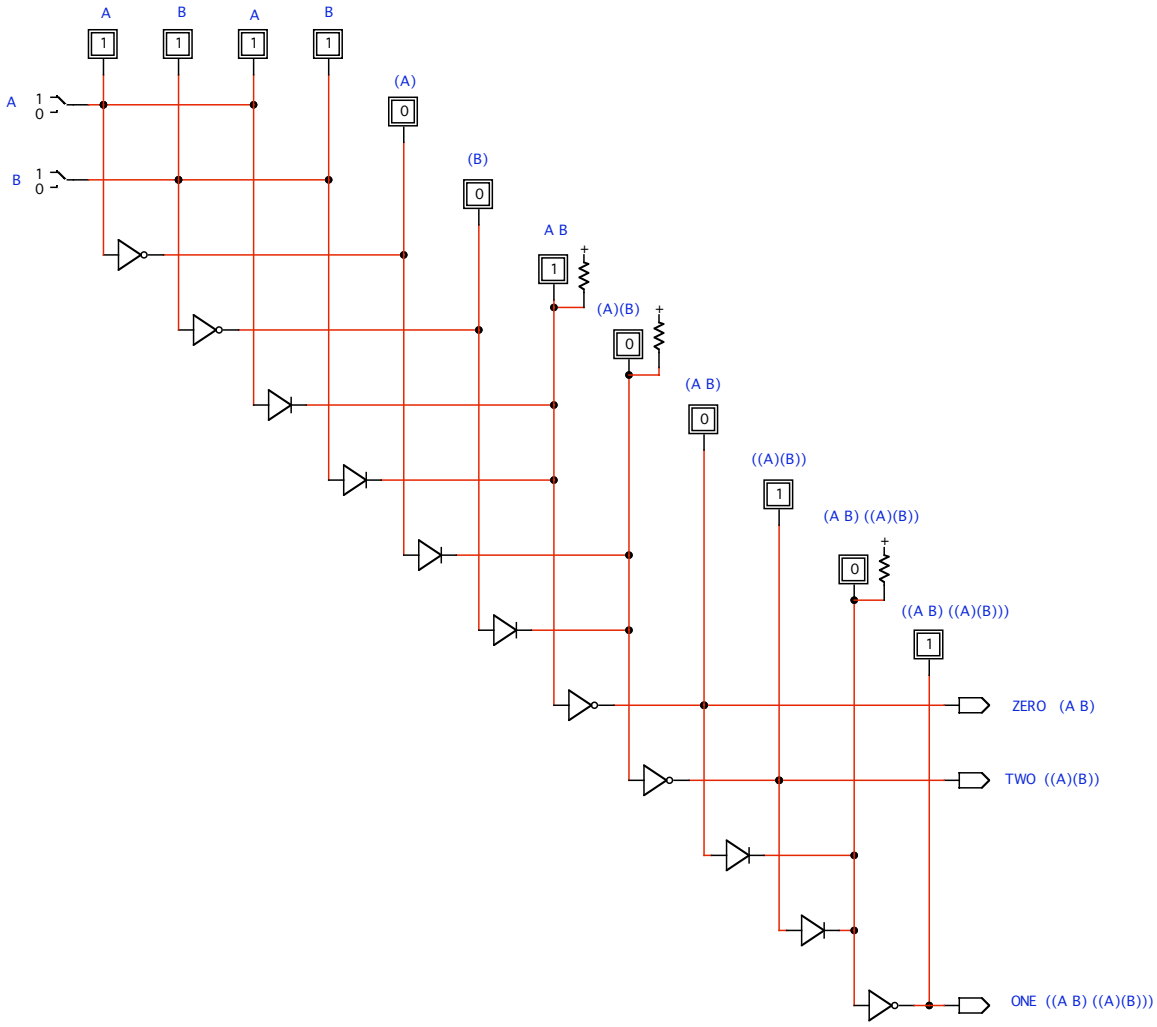
```
Parens:  sum   = ((A B)((A)(B)))
         carry =       ((A)(B))
```



A half-adder is XOR with two outputs. Here two separate functions (sum and carry) are combined into one comesh. They happen also to share structure, the carry output row branches to two rows.

2-bit TALLY

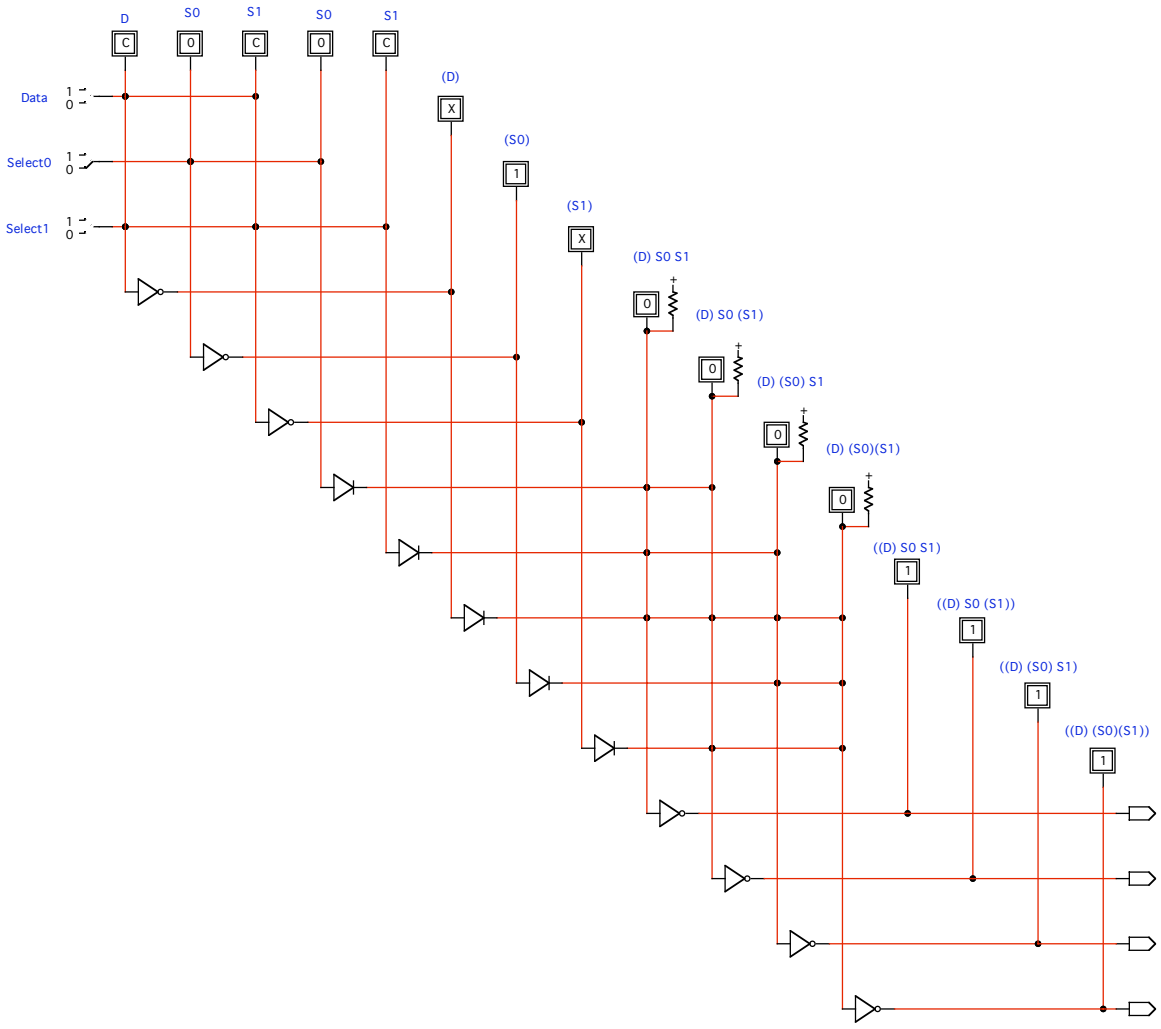
```
Parens:  zero = (A B)
         one  = ((A B)((A)(B)))
         two  = ((A)(B))
```



The two-bit tally is similar to a half-adder, except three structure-sharing functions are combined. The half-adder sum also means exactly one of A or B. The new zero output is neither A or B.

1to4 DEMULTIPLEXER

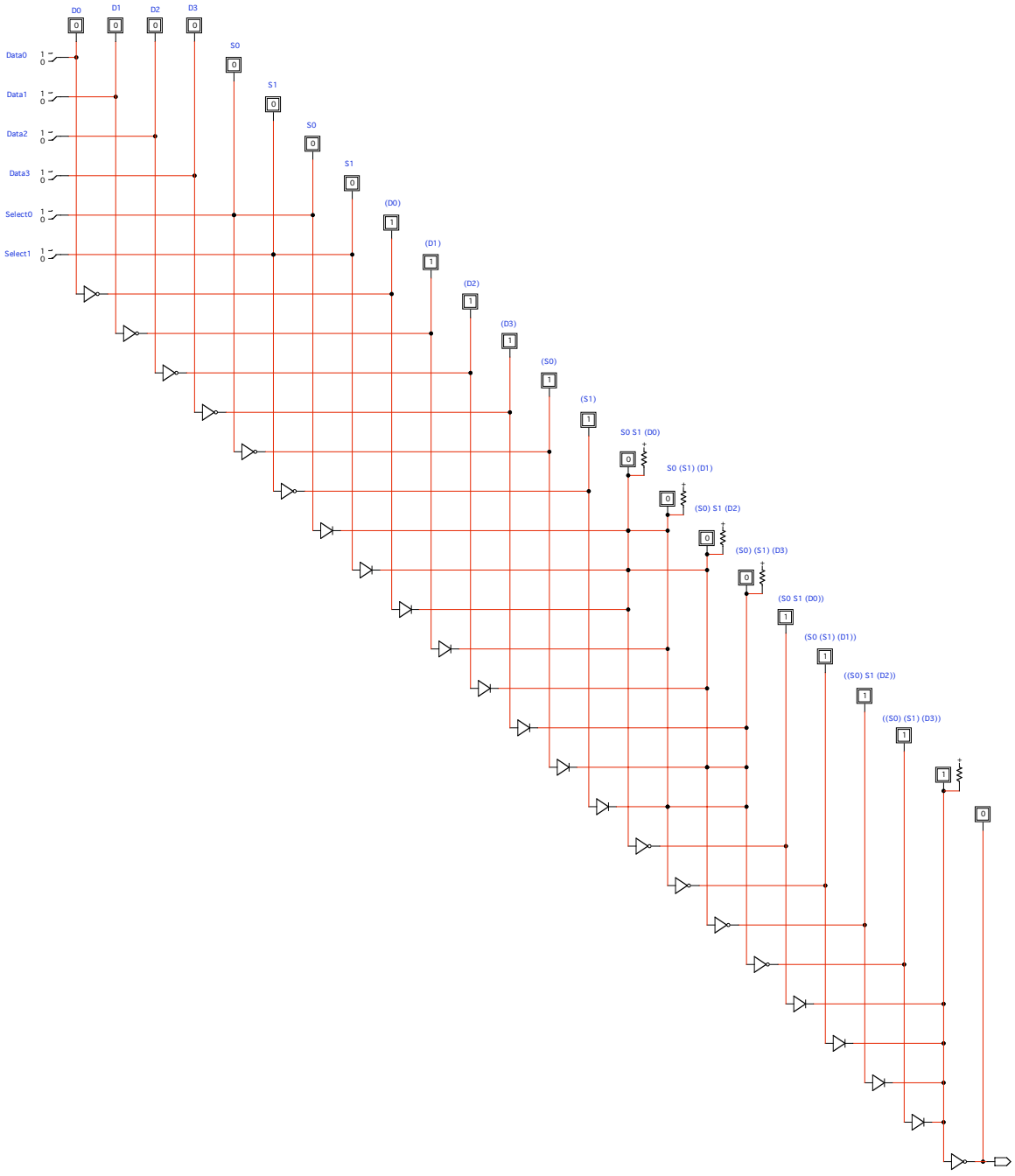
```
Parens:  out1 = ((D) S0  S1 )
         out2 = ((D) S0 (S1))
         out3 = ((D)(S0) S1 )
         out4 = ((D)(S0)(S1))
```



The four possible values of two select rows each combine with the data D. Each configuration of select values voids three of the four output lines. If D is False, all output lines will be 0.

4to1 MULTIPLEXER

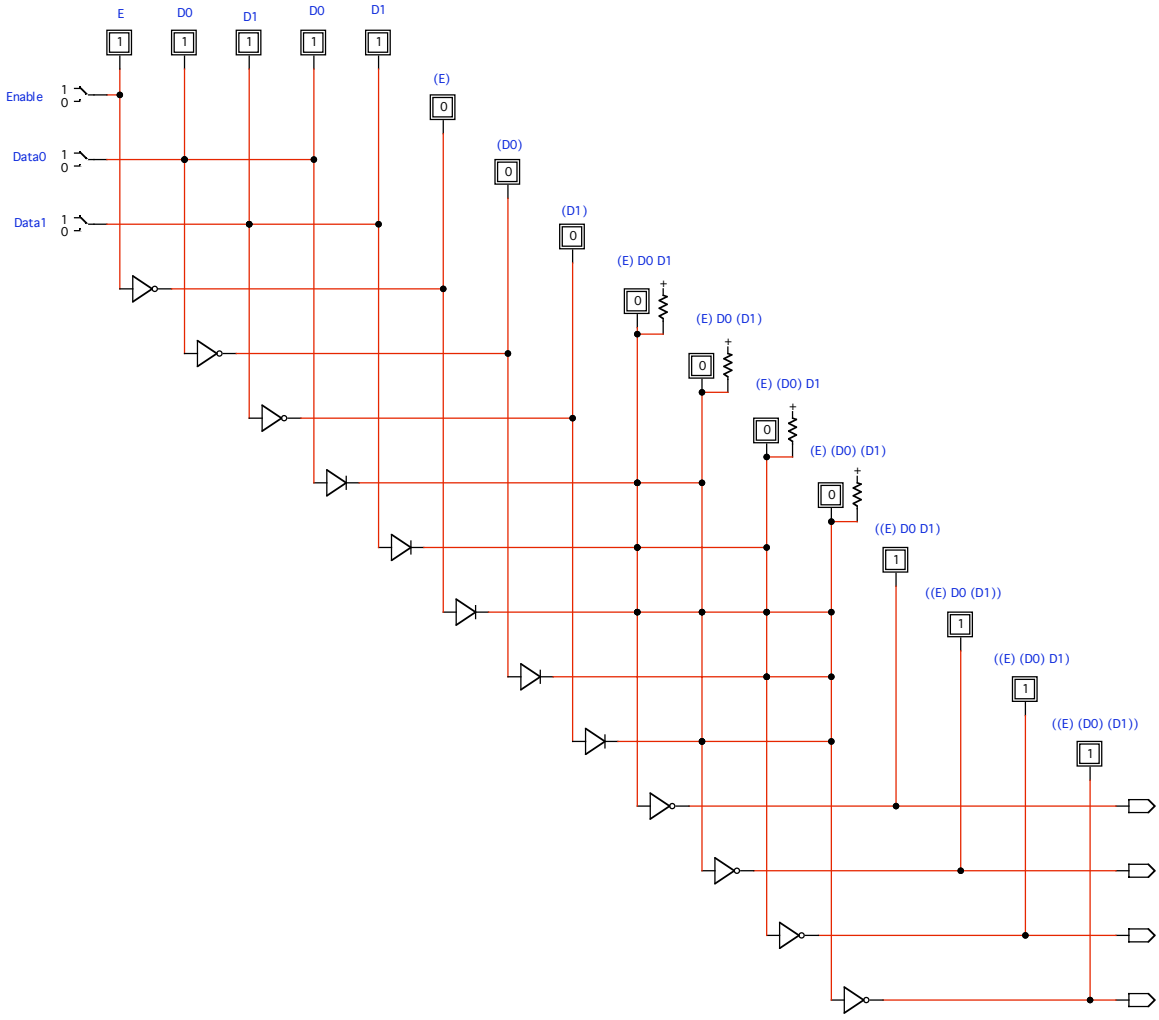
Parens: ((S0 S1 (D0)) (S0 (S1)(D1)) ((S0) S1 (D2)) ((S0)(S1)(D3)))



The four combinations of 2 select rows associate uniquely with one of four input rows. The active input row becomes output.

2to4 DECODER

```
Parens:  out0 = ((E) D0  D1 )
         out1 = ((E) D0 (D1))
         out2 = ((E)(D0) D1 )
         out3 = ((E)(D0)(D1))
```



All outputs are 0 when Enable is 0. When Enable is 1, the one output that corresponds to the four possible configurations of two data rows is high. This circuit is identical to the 1to4 demultiplexer, except that input rows have different names.