COMPUTATIONAL MESH

William Bricken
October 2001

OVERVIEW

Section I (Descriptions) describes the architectural components and benefits of Comesh technology. Section II (Illustrated Tour) provides dozens of simple and intermediate examples of Comesh circuits, with emphasis on comparison of different array configurations. Section III (Implementations) provides the dnet code generated by Losp which accompanies the illustrations.

CONTENTS

Comesh

Computational mesh (Comesh) is an array-based technique for the construction of both combinational and sequential semiconductor circuitry. The specialized array is a triangular matrix of bit storage sites with inverters on the diagonal.

The mesh is an array of row and column wires which can be connected at selected intersection points (cross-points). The functionality of the target circuit is encoded in the configuration of connected cross-points. Essentially, each connected cross-point in a Comesh corresponds to a physical wire in a conventional ASIC. Row connections are fanout of a signal; column connections are wire-ORs of a collection of signals. Diagonal inverters manage the flow of computation, by inverting each column, they create a network of NOR logic. In combination, NOR logic constructed from wire-OR columns and diagonal inversions fully implements all combinational and sequential logic networks.

Comesh is not composed of logic gates in the conventional sense. Each column and inverter does implement the functionality of a NOR gate with an arbitrary number of inputs. Each row does implement the fanout of that abstract NOR gate. Input is defined by designated input rows; output is defined by designated output rows. However, Comesh cross-point configurations are explicitly structures of Iconic Logic. Rows are boundary objects, either atoms or containers. Connections in a column indicate which boundary objects share a common space. A column then is *a bounded space*. Connections in a row indicate which spaces contain the row boundary object. The diagonal inverter specifically represents the act of crossing a boundary, moving computational focus from the column space to the row container of that space.

Comesh "registers" are also rows which generate output signals; the difference between logic and registers is that register rows outputs are reentered into the array as inputs (together with external input signals) at each internal clock cycle. Thus Comesh registers are not flip-flops, and do not behave as flip-flops, rather their behavior is equivalent to that of Comesh gates. This makes timing the Comesh exceedingly simple.

A Comesh replaces both conventional wiring and conventional timing with its own internal structures. All timing is Comesh pipelined, all wiring is stored in Comesh cross-points. Comesh provides multilevel logic functionality in a PLD-like device, without the exponential explosion of flat AND-OR logic. Since timing is defined by the Comesh rather than by wiring connecting arbitrary sequences of gates, Comesh timing design is highly predictable. Since wiring is defined by the location of cross-point connections, place-and-route is equivalent to loading the mesh configuration. Performance is therefore independent of wiring complexity, and largely insensitive to the timing complexity in the original design.

Cross-points

The types of connectivity at each cross-point vary across types of desired performance, providing a wide choice of functional and manufacturing properties. Comesh cross-points are essentially bit locations, cross-point architectures mimic those available in commercial memories. Non-reconfigurable meshes have hardwired cross-points. Hardwiring can be achieved either by factory masking (Mask ROM), by write-once fuse technologies in the field (PROM), or by limited writable reconfigurable EEPROMs.

Reconfigurable meshes require dynamic cross-points. The simplest, most dense, and most familiar is DRAM, which uses one transistor per intersection. More elaborate cross-point architectures include SRAM and Flash-ROM. Some characteristics of these bit location architectures are listed below:

Mask ROM: non-reconfigurable factory hardwired connections
very fast, high density, very low power, inexpensive, non-volatile
Flash RAM: readable/writable RAM
inexpensive, high density, moderately reconfigurable, non-volatile
SRAM: static read/write memory
fast, low density, high power, expensive, volatile
DRAM: dynamic read/write memory
high density, inexpensive, low power, volatile

Comesh is *not* a memory, it is a bit-array. No address decoding is necessary, none of the control or read/write logic of memory is relevant to the computation. All input lines are activated concurrently. All output lines become available concurrently.

Configurations

Custom software (Losp/Pun) minimizes the target circuit and maps it into a *dnet*, which is the internal format for Losp/Pun Comesh data-structures. Losp provides Boolean minimization and conversion to dnet format. Pun provides management of network connectivity such as structure sharing and fanout. Losp/Pun functionality is completely automated. Configuration choices are parameterized to achieve desired structural properties.

Although the mesh can be considered to be square, the number of rows is greater than the number of columns by the number of input literals. I.e.

rows = columns + input-literals

There is one row/column for each dnet cell in the function, and one row for each input literal (positive and negative polarity inputs).

Registers are a single Comesh row/column pair. The input of a Comesh "register" is a row which is similar to any other input row. This row is connected to a column which has no other connection points, it is the register inversion column. When this column is inverted into a new row, that row is the register input prior to the next clock cycle, the next state stored in the register. The value in the register output row is treated as any output, it is asserted as output into an output vector of conventional registers (the Output Buffer). On the following clock cycle, these register outputs are reentered into the Comesh as inputs, equivalent to any other inputs entering the mesh.

The number of dnet cells in a Comesh configuration, that is the number of rows, is roughly equivalent to the number of n-input NOR gates plus registers in an equivalent multilevel circuit. Comesh columns are insensitive to fanin, so that the gate count of an equivalent circuit should include any amount of fanin; gate counts are not determined by two-input gates. Similarly, Comesh rows are insensitive to fanout, so an appropriate comparison would be circuits with unlimited fanout. Since almost all netlists use low fanin and fanout gates, Comesh optimization improves significantly over conventional gate counts. Very roughly, one Comesh row is equivalent to about three conventional gates.

Comesh rows are coupled with their corresponding columns. However the relative position of each row is completely free. This allows Comesh rows, particularly input and output rows to be arranged to fit whatever pin locations are required from the global connectivity context.

Comesh is fully composable. Each segment can be treated as a complex logic block with inputs and outputs which compose by connection. Composition can be achieved a Comesh blocks, or if preferred, within a single Comesh block.

Flattening a circuit to two-levels (SOP format) increases both fanout and wire-OR loads of a Comesh. Unlike a PLD, these increases are in the number of cross-points, not in the number of physical wires or gates. The exponential growth associated with SOP forms is limited to growth in clauses, that is increases in the number of bounded groups of inputs, the AND gates of a conventional PLD.

Comesh is optimized for deeply nested structures, since deep multilevel nesting of gates minimizes the number of gates and thus the number of rows in an equivalent Comesh circuit. In deeply nested designs, propagation across diagonals corresponds to the critical path of a conventional multilevel design. A Comesh requires one diagonal propagation step for each conventional gate in a multilevel path. Like conventional circuits, all Comesh combinational logic is asynchronous. Unlike conventional circuits, propagation delays through a Comesh configuration are highly predictable.

Diagonal Cells

Diagonal cells are inverters. Wire-ORs feeding the inverter creates NOR-logic. The entire Comesh is unclocked, except at outputs.

A possible clocking regime would be to clock each diagonal inverter. Clocking each inverter turns the entire mesh into a fine-grain clocked network. The BILD Engine implements a version of diagonal element clocking, enhancing predictability of performance at the cost of slower operation and more hardware mechanism.

A hardwired Comesh ROM-like configuration requires two transistors per row, the two transistors comprising the diagonal inverter. Thus, all Comesh gate-equivalent rows require two transistors. A 10K gate circuit literally requires 20K transistors in the Comesh architecture. This makes non-volatile Comesh ROM extremely small and extremely inexpensive to manufacture.

Functional Behavior

Inputs (and their inversions) enter the Comesh concurrently. Active cross-points in each row define the fanout of the incoming signals. Cross-points in each column define the inputs to a dnet cell. Low signals on a row have no effect on a computation, they represent disconnected elements. High signals on a row are propagated to each column that the row is connected to via a cross-point. hen a high column signal reaches the diagonal inverter, it is in effect cancelled by the inverter, creating a corresponding row with no signal. When the Comesh settles and a column has no signal, the pull-up on the inverter of that column triggers the inverter to emit a high signal to the corresponding row, in effect propagating an active signal.

Hierarchical Subdivision

Comesh area can be reduced by reclaiming unused cross-points in the sparse array of connectivity. Several techniques are possible to reduce area:

1) Compose the Comesh of several smaller meshes. A 100 row mesh might be made from tiling ten 10 row meshes, ten across the row. (Thus, the 100x100 mesh would be composed of one hundred 10x10 meshes.) Submeshes with no connections can be abstracted to a single meta-non-connect.

2) Compose larger Comesh functions from smaller Comesh functions by direct wiring.

3) Connect an array of Comesh submeshes using conventional interconnect technology. Thus, a submesh would be similar to a smaller look-up table in a CPLD architecture.

4) Compose limited mesh resources over time. A large function would be partitioned into several smaller mesh configurations, these smaller configurations would be dynamically swapped by reconfiguring the mesh connectivity.

5) Use multi-bit cross-point storage techniques to carry several different functional configurations in the same mesh.

6) Reconfigure the connectivity in Losp by changing the structural characteristics of the dnet. An example would be to triangulate the mesh (assumed standard), another would be to form a SOP dnet.

7) With an appropriate cross-point architecture, reclaimed area can be explicitly used as memory, providing a homogeneous computation/memory fabric.

Performance and Metrics

The essential advantage of a Comesh network is that it performs as a PLD without the combinational explosion encountered in all other PLD architectures. Roughly,

2^N PLD gates ==> N^2 Comesh bits

Specifically,

N = number of input literals plus registers
M = number of internal gates, regardless of fanins and fanouts, plus registers

CPLD gates, worst case: 2^N

Comesh rows, worst case: N+M
area: (N+M)*M

Multilevel circuits originated in order to combat the exponential explosion in PLD gate count. Comesh provides the functional and manufacturing benefits of any PLD, while retaining the structural economy and performance of any multilevel ASIC.

Benefits

In a sentence: Comesh combines the best of both ASICs and PLDs, with the drawbacks of neither. In particular,

Unlike ASICs, the design, verification and layout of Comesh functions is both easy and rapid. Comesh is homogeneous, and thus has the manufacturability

characteristics of memory and of homogeneous gate arrays rather than custom ASICs. Comesh performance is slightly slower than an equivalent ASIC design.

Unlike PLDs, Comesh performs at ASIC speeds while reconfiguring at memory-write speeds. Internally Comesh requires no physical timing or routing. Comesh performance is always better than an equivalent FPGA design, and is roughly equivalent to a PLD design. However, Comesh design sizes are determined by the number of multilevel any-fanin gates.

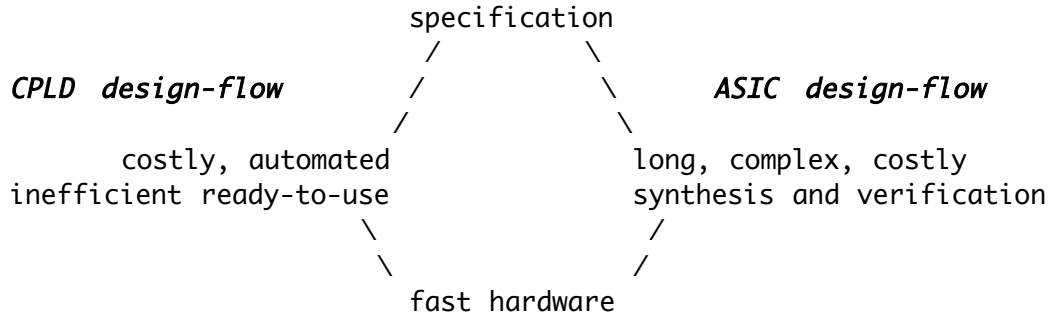
Comesh area (die-size) is slightly smaller than a functionally equivalent FPGA, since it does not require routing resources. This area is significantly larger than an equivalent ASIC.

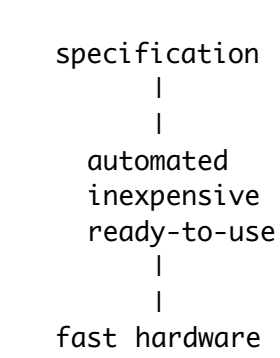
Reconfiguration

Reconfiguration is accomplished in software, at memory-write speeds. Thus RAM-based and flash-RAM Comesh is extremely adaptable, providing dynamic reconfiguration combined with ease of design.

Design flow

Here is a comparison of standard and Comesh design flows:



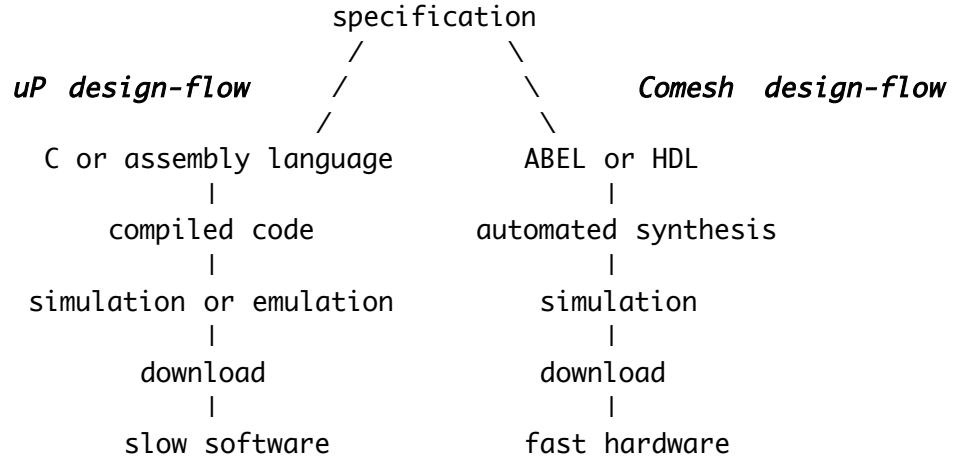


For large designs, conventional CPLDs are too inefficient in space and wiring. Two solutions have developed, FPGAs and uPs. FPGA design-flow is very similar to CPLD design, except that the end product is slow, relatively expensive hardware. uP design is discussed below.

Microprocessors

Comesh is competitive with embedded microprocessors with a fixed variety of functions. The flexibility of software instruction sets is matched by the rapid reprogramability of Comesh. The trade-off is HDL design (for Comesh) vs. instruction set design (for a vonNeumann uP). Both are equally difficult, but designing in HDL gives immediate hardware speedup, in the range of two to three orders of magnitude.

Comparing instruction set and Comesh design:



uP compiler technology is relatively advanced, so that timing and resource allocation is fully automated. Comesh has the same automated "compiler" synthesis, but the resource allocation problem it addresses is far simpler, so simple that the designer can just set design parameters such as delay and area.

Instruction set design is complex since hardware and software characteristics interact. Software verification is at best horrible, algorithmic and formal verification of software is not yet understood. Software engineering imposes strict design regimes which again make software design difficult and highly error prone. Worse, there are few techniques know to remove bugs; it is a software industry standard that all software is shipped riddled with both known and unknown bugs. From a hardware perspective, this is intolerable.

Comesh synthesis is completely formal, and fully verified at every design step.

Both software and hardware approaches make download easy, neither require place and route. The cost of using a software approach is much slower performance.

Many uPs use the flexibility of instruction set encoding to provide multiple functionality. Dynamic reconfiguration of hardware FPGA architectures is still in its infancy, it is not particularly easy to exchange functionality. Comesh is reconfigurable at the same speed of reprogramming a software-based uP. Reconfiguration design is easier than with software. Thus Comesh provides all the benefits of instruction set programming, with none of the performance disadvantages. Reconfigurable Comesh uses flash-ROM, DRAM or SRAM cell technology for bit storage, which adds a small additional cost over non-volatile Comesh ROM which is hardwired.

Manufacturability

Comesh is dominated by bit storage cells, which have a standard known architecture. Memory (ROM, flash-ROM, DRAM, SRAM) is increasing in price/density ratio faster than ASICs (an increase of 1.6 per year rather than 1.25), thus costs of manufacturing Comesh will decrease faster than all competing ASIC and FPGA products.

Layout control

Layout control is equivalent to place and route for conventional reconfigurable hardware. However, Comesh layout is simply the identification of which cross-points in the mesh to make active, either by storing in a DRAM or flash-ROM-like cell, or by hardwiring the connection.

Pun provides parametric control of

- the desired size of the array,
- the degree of wire fanin/out (wire-OR loading),
- the number of diagonal gates traversed during the computation (process time), and
- the contextual location of inputs and outputs.

Given that these parameters are all highly mutually interdependent, the design space is actually quite small. The dnet form can vary over these parameters, however a minimized dnet also places significant constraints on connectivity design. This means that the Pun "synthesis and placement" algorithms can be very efficient, as well as very definitive about resulting behavioral characteristics such as propagation time and power usage.

Since Comesh is composable by wiring outputs to inputs, it provides maximum flexibility for extending and reusing resources. Complex functions can even be dynamically partitioned and assembled later, using a single small Comesh.

IP cores

Configuration encoding of Comesh can be treated as IP, similar to any software (instruction set) IP.

Summary of Advantages

-- pin i/o location is flexible

-- memory and computation can be mixed in the same fabric

-- only the output register and the diagonals change,
thus very low power and noise

-- configurations are easily decomposed for time/space trade-offs

-- since disconnections are propagated, very low power use

-- inherent Iconic Logic simplicity

-- wiring is virtual,

-- connected cross-points define paths though i/o space,
rather than paths through gates

-- two transistors per logic function (plus cross-point architecture)

-- all techniques are backward compatible

-- 60% yearly growth in memory technology

-- registers are virtual and integrated into the Comesh configuration

-- freely partitionable for cross-mesh integration

-- timing is predictable

-- differentiable not incremental technology

ILLUSTRATION TOUR

Over fifty illustrations of Comesh circuits follow. They include the most simple logic gates, common smaller circuits, and examples of Comesh composition and timing.

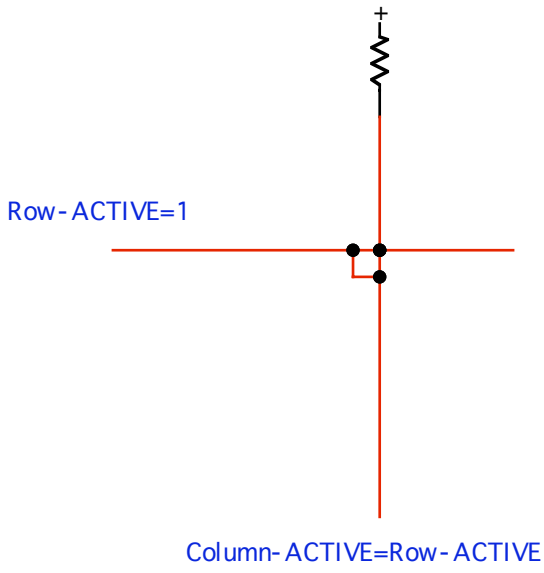
Cross-points

Cross-points are indicated symbolically by an angle join of a row and a column.

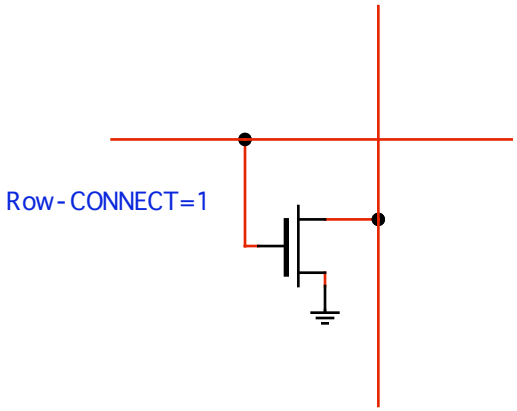
Row cross-points spread a particular signal value to other rows, via column collection. Column cross-points form a logical wire-OR. In boundary terms, row cross-points identify the containers of the row output, column cross-points indicate contents of the column input.

In a void-based model, any Hi row value will dominate (Dominion) both the row and the column of a -point. Other potential inputs to row or column are irrelevant. The dominant signal will propagate through all row cross-points to all connected columns. These will propagate to the diagonal, where they will be inverted into void. Lo signals do not propagate and do not change the state the Comesh.

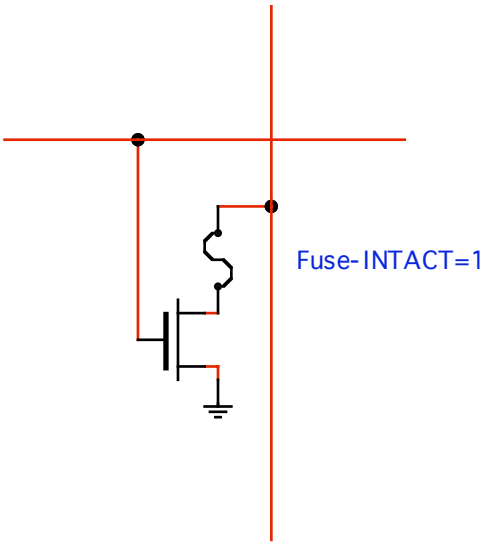
Generic Cross-point



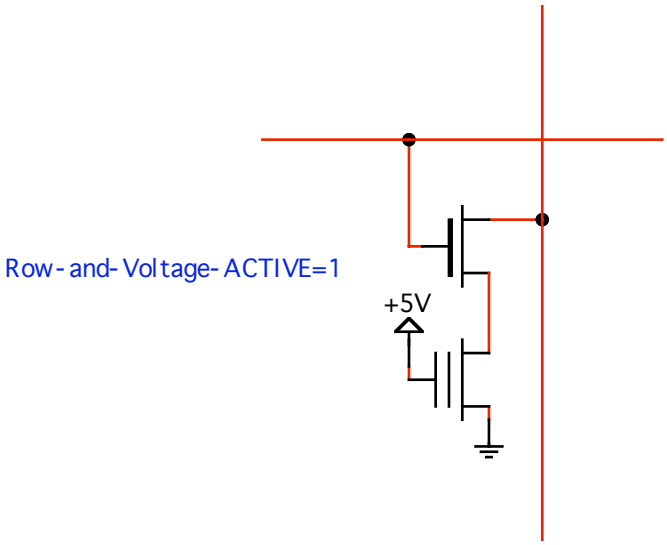
ROM Cross-point



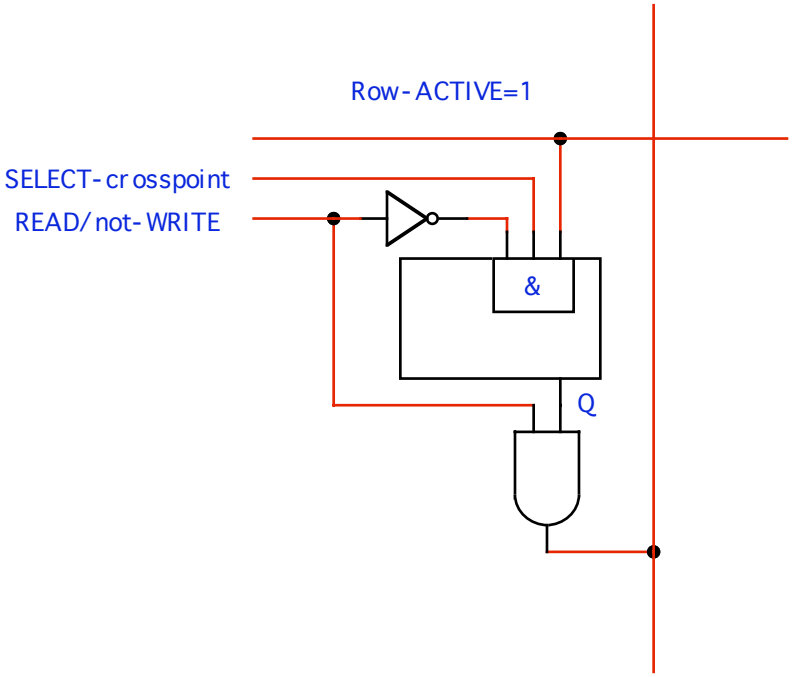
PROM Cross-point



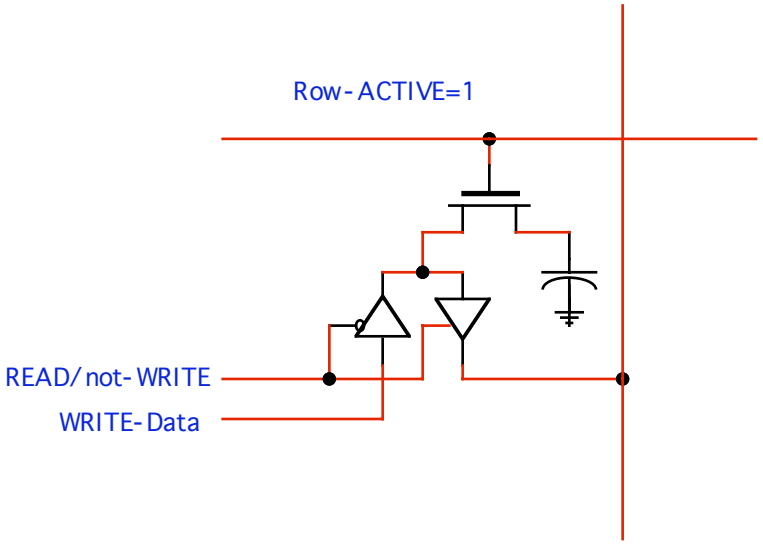
EEPROM Cross-point



SRAM Cross-point

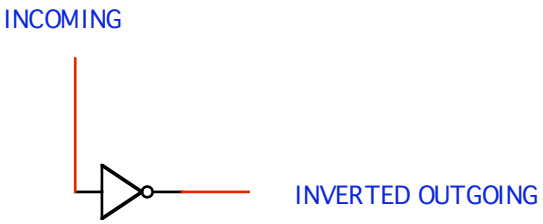


DRAM Cross-point



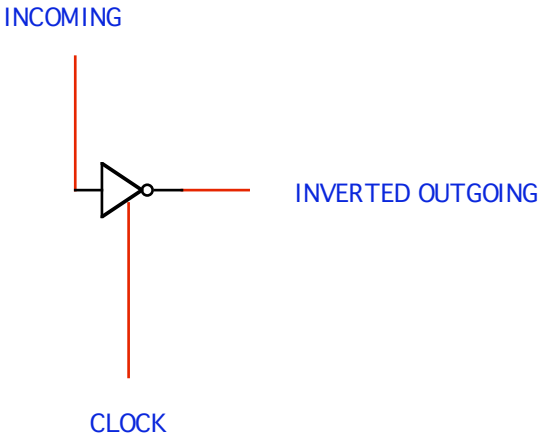
Diagonal Elements

Simple inverter



Hi incoming is voided. Lo incoming is made dominant.

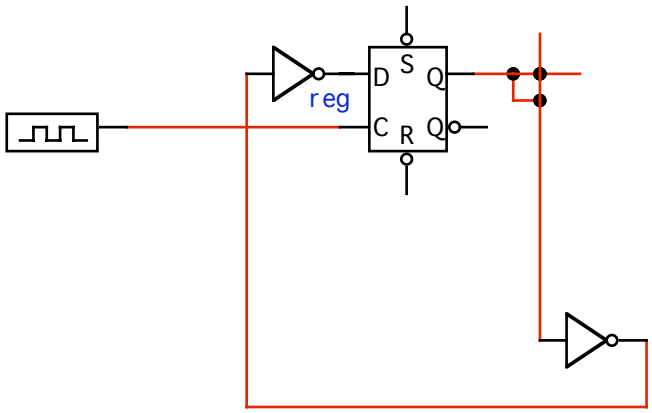
Clocked inverter



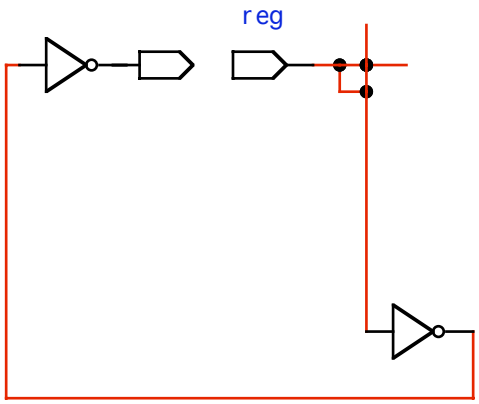
The diagonal element is clocked to tightly control propagation.

Registers

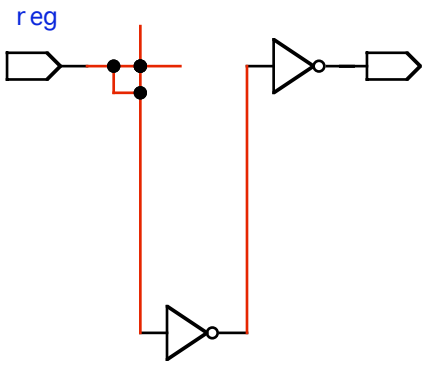
A single register



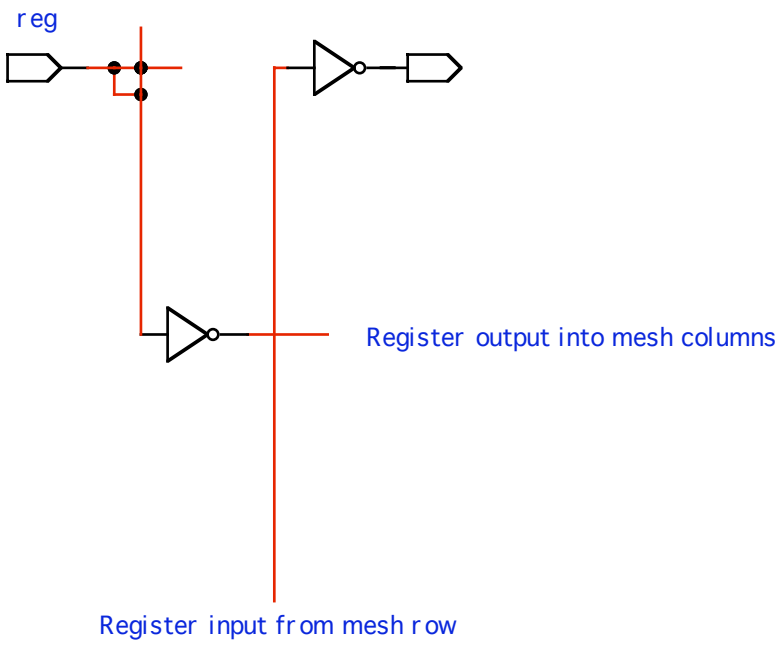
Shorthand notation using a feedback loop



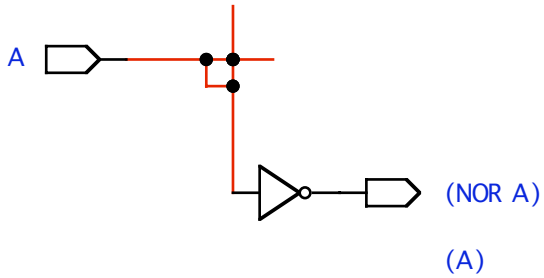
Shorthand notation using an implicit loop. This register is connected to itself.



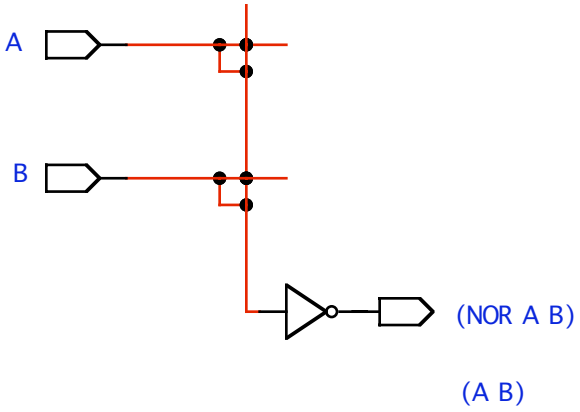
The representation of a generic register.



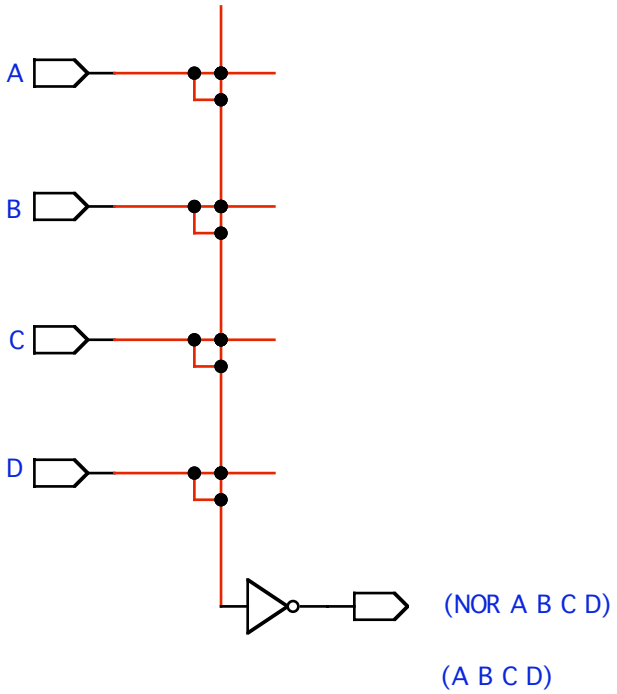
1-input NOR



2-input NOR

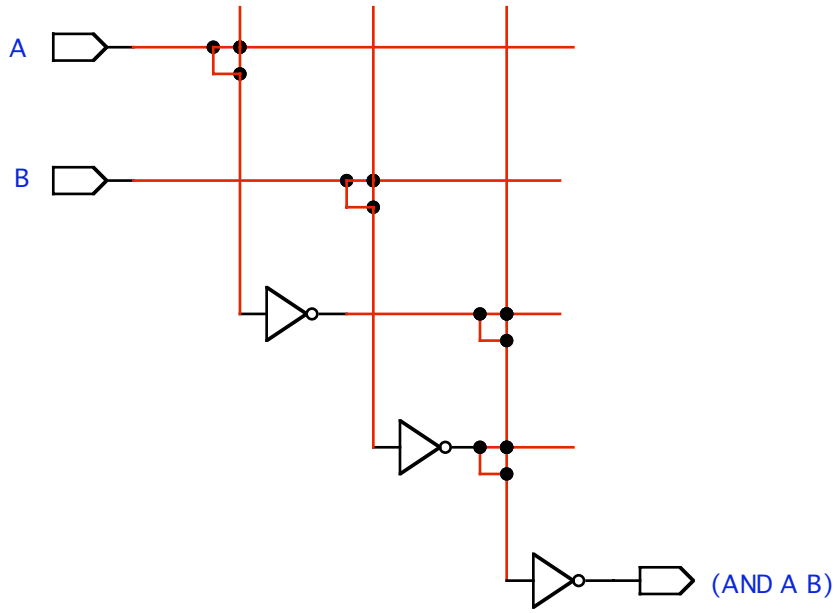


4-input NOR

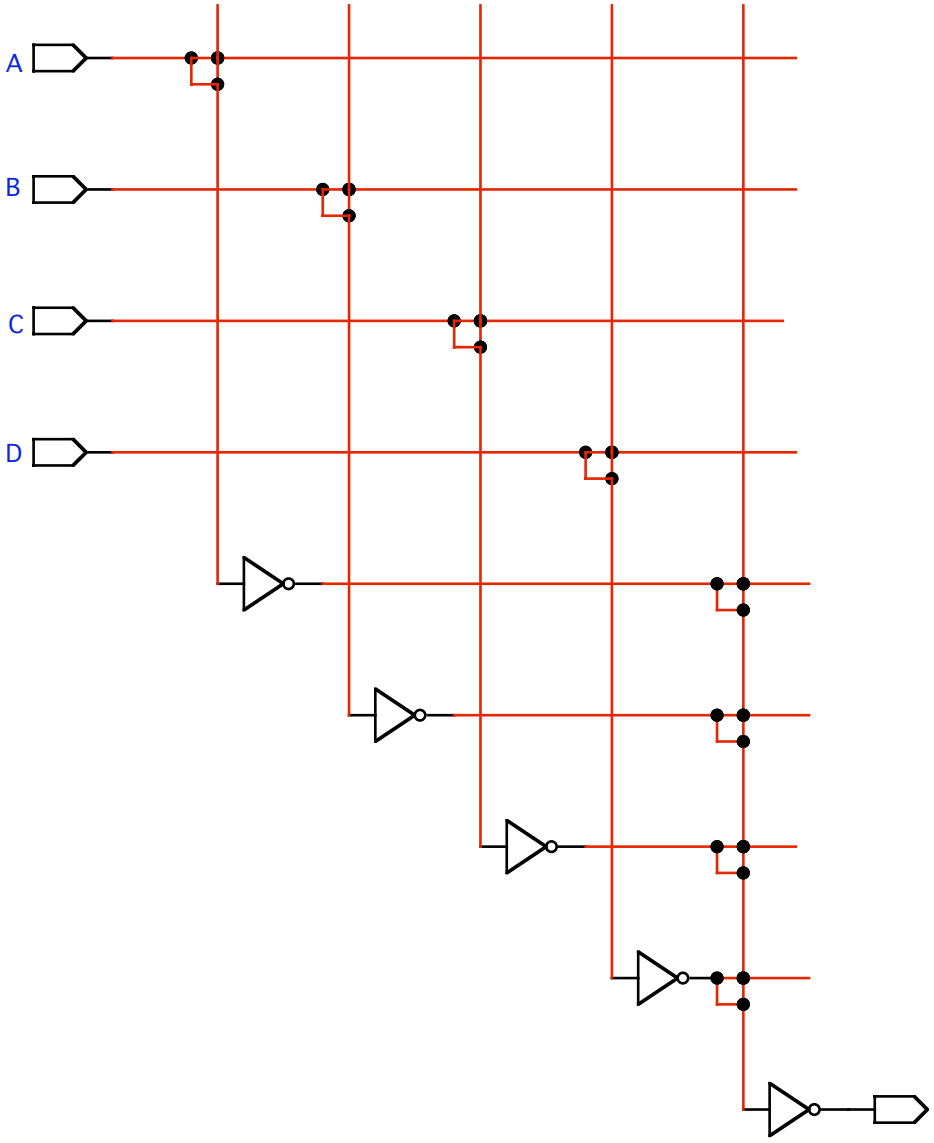


Arity is simply the opportunity for any one of several row signals to dominate a column. Once a column is dominated, it becomes irrelevant since the Hi signal is inverted into void.

2-input AND

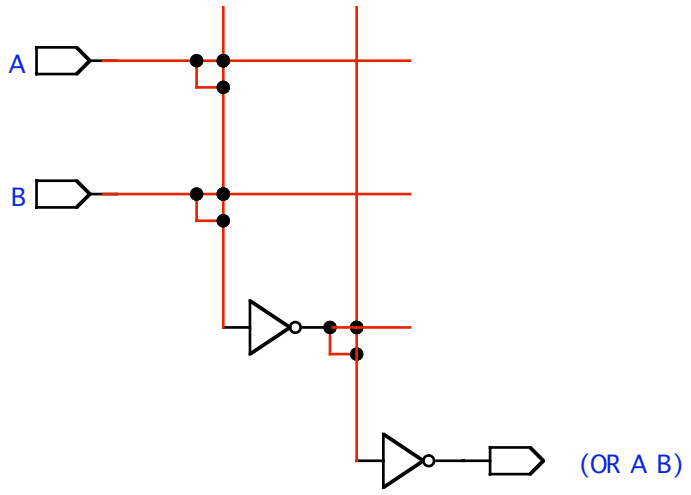


4-input AND



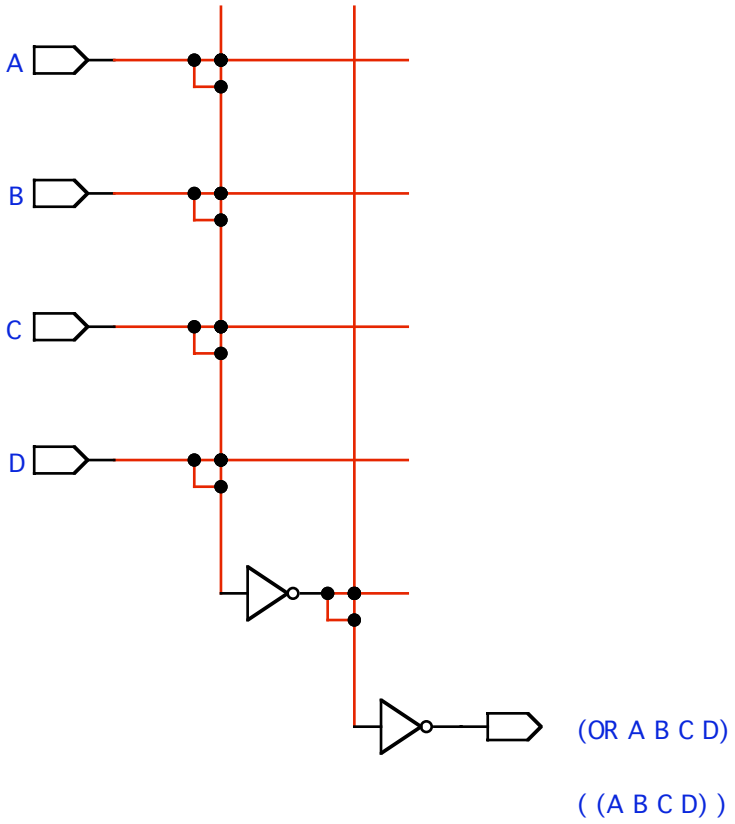
AND and NOR have the same column structure; however AND rows come from inverters.

2-input OR

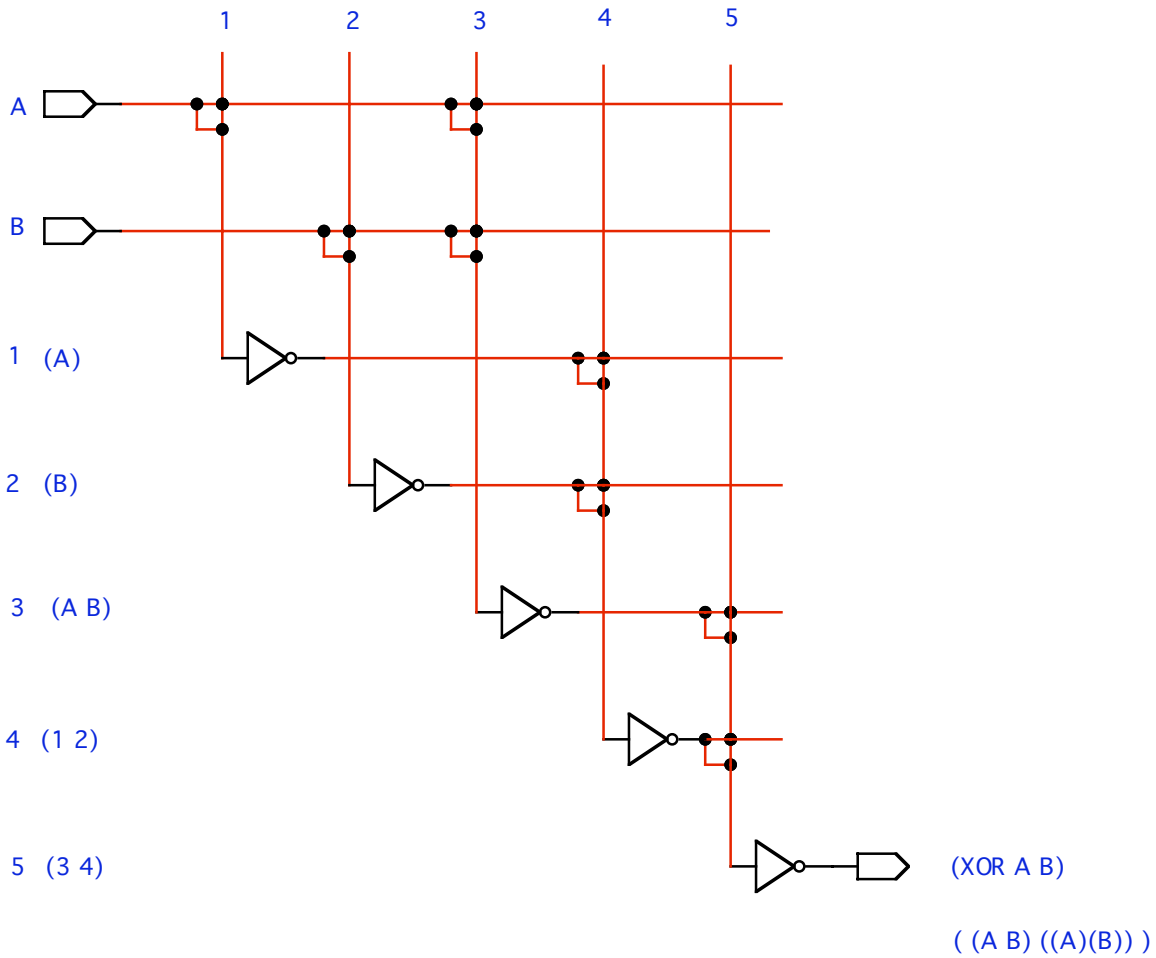


OR has the same structure as NOR, but its column signal is inverted twice, and thus kept active.

4-input OR

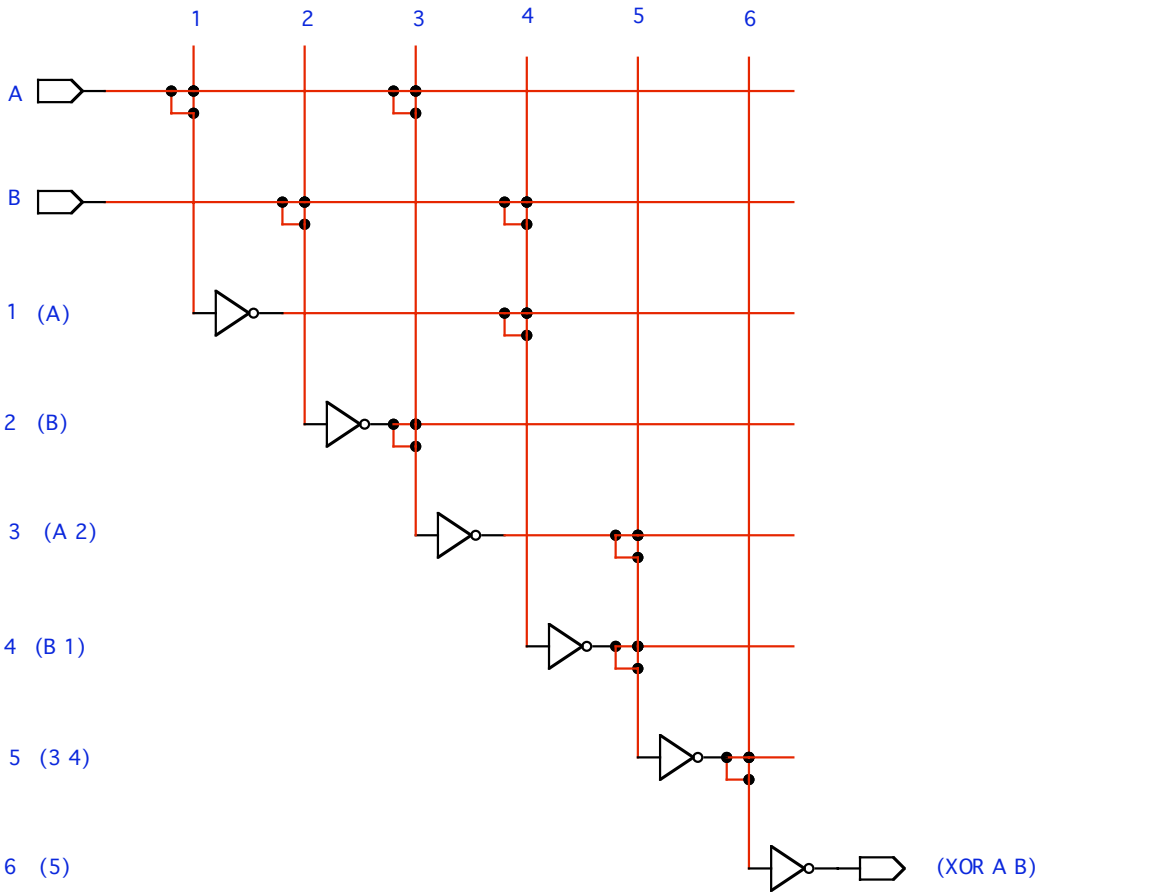


2-input XOR, Version I



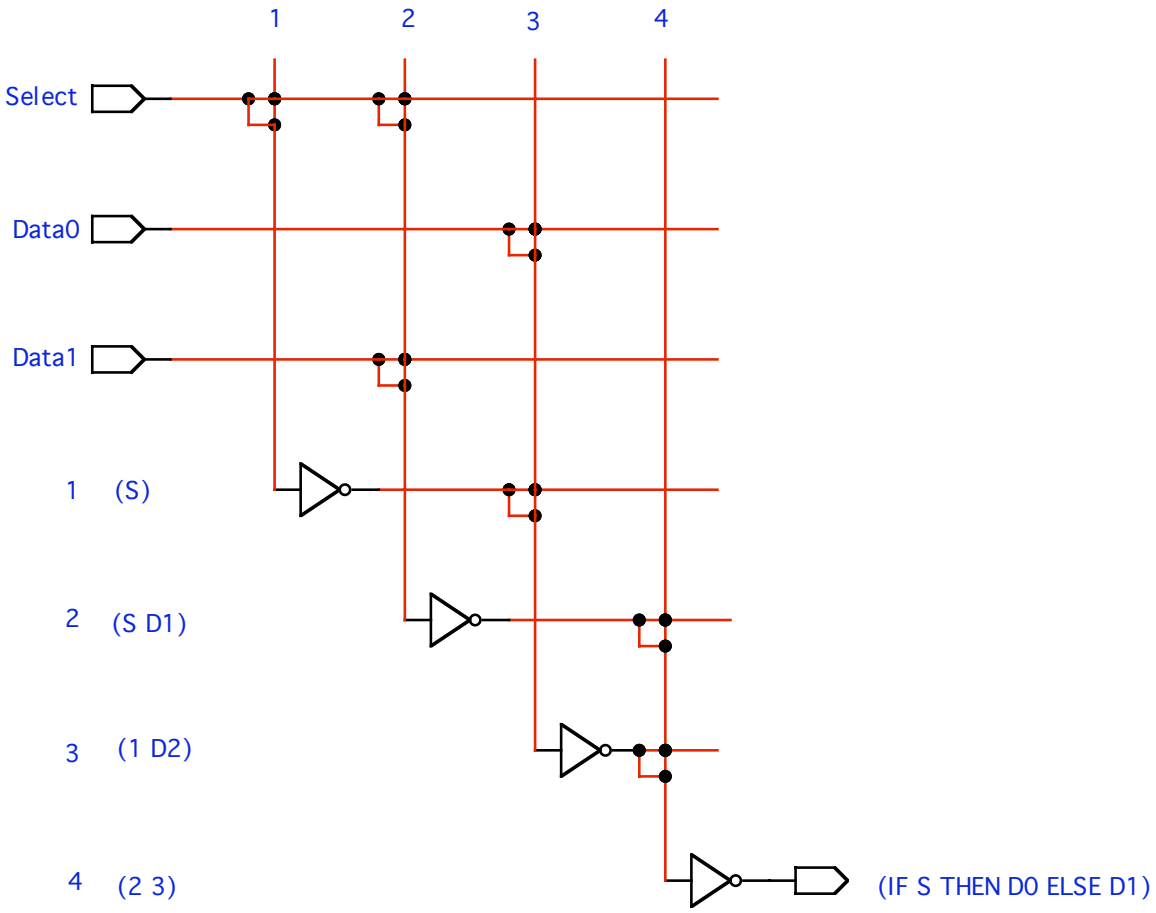
XOR takes two NOR columns, one with positive signals and one with inverted signals, and combines them in another column.

2-input XOR, Version II

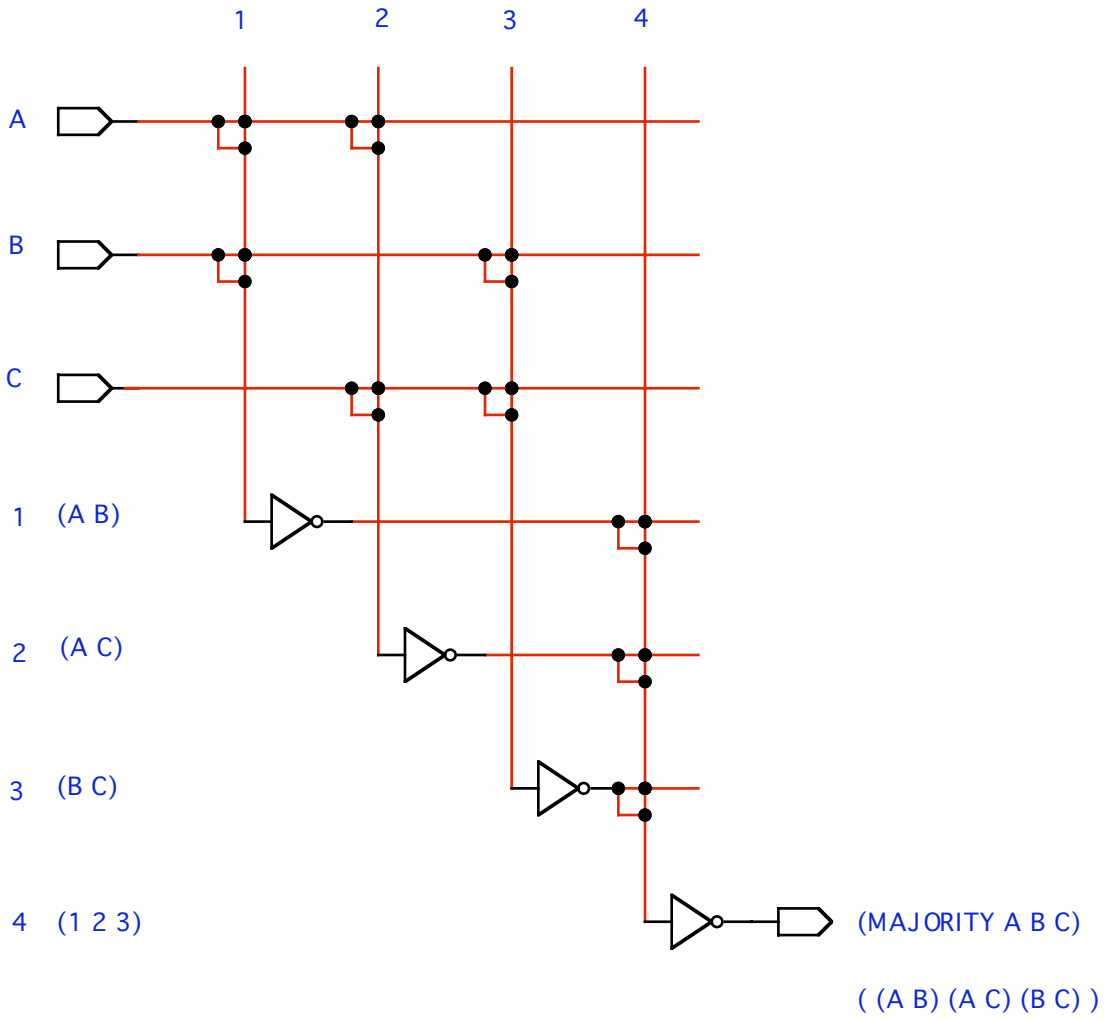


An alternative version of XOR, slightly less efficient.

2to1 MULTIPLEXER (MUX)

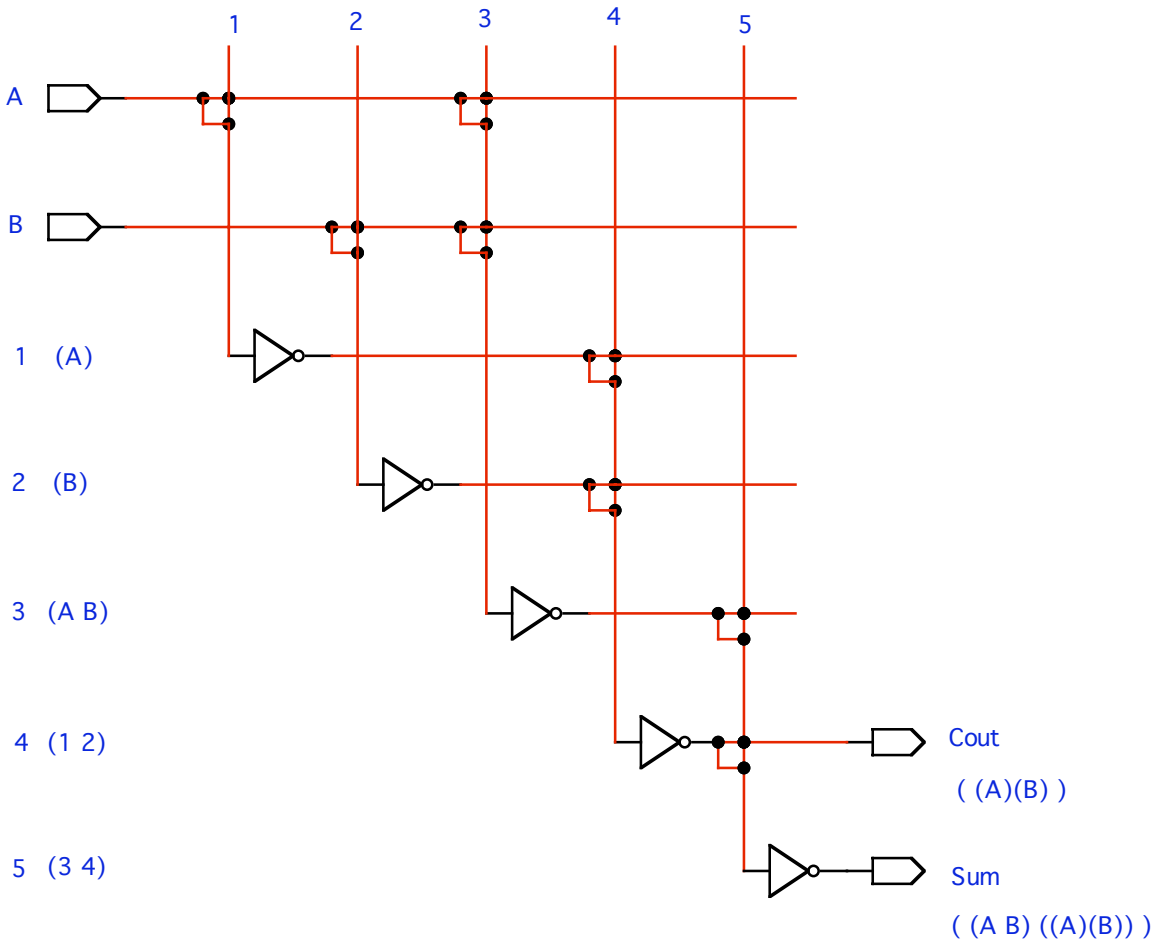


2/3 MAJORITY



2/3 MAJORITY combines all pairs of incoming rows into another row.

HALF-ADDER

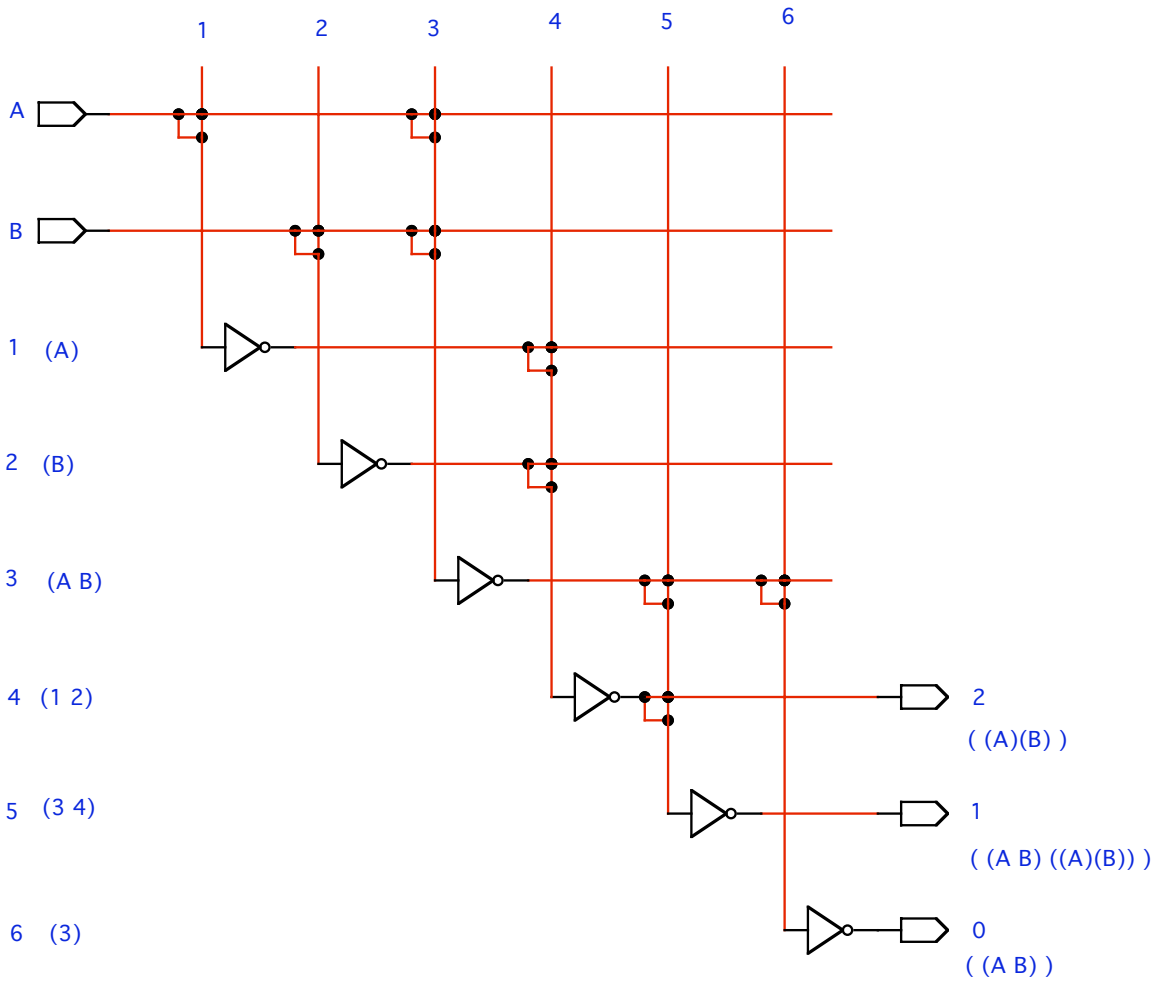


A half-adder is XOR with two outputs. Here two separate networks are combined into one Comesh. They happen also to share structure, row 4 is the branch point. Row 3 also provides a branch point to both outputs, a possibility not usually incorporated in conventional half-adders.

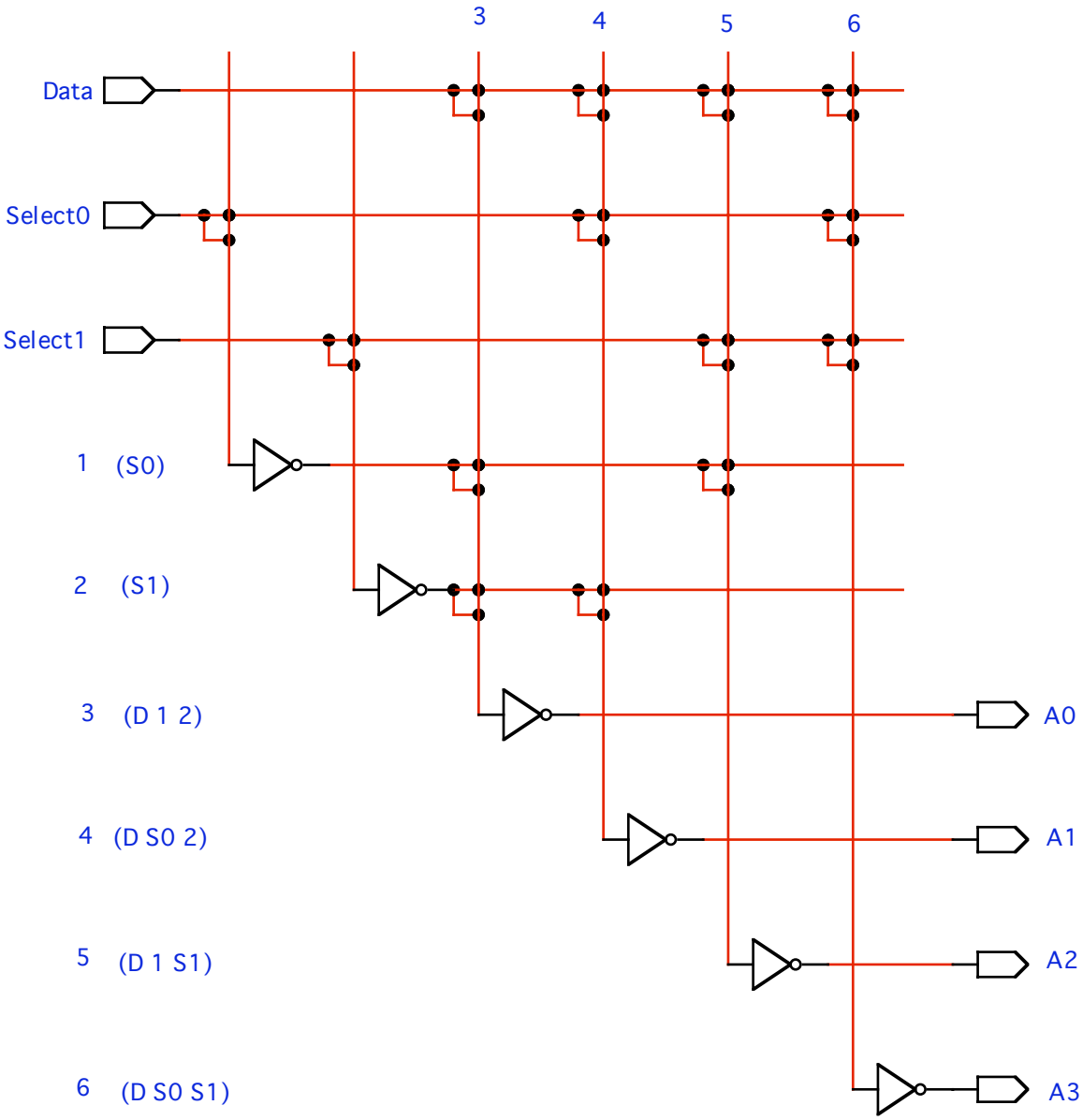
Reading the Comesh:

Row 3 is relevant only when neither input dominates (both 0), in which case the carryout is Hi and the sum is void.

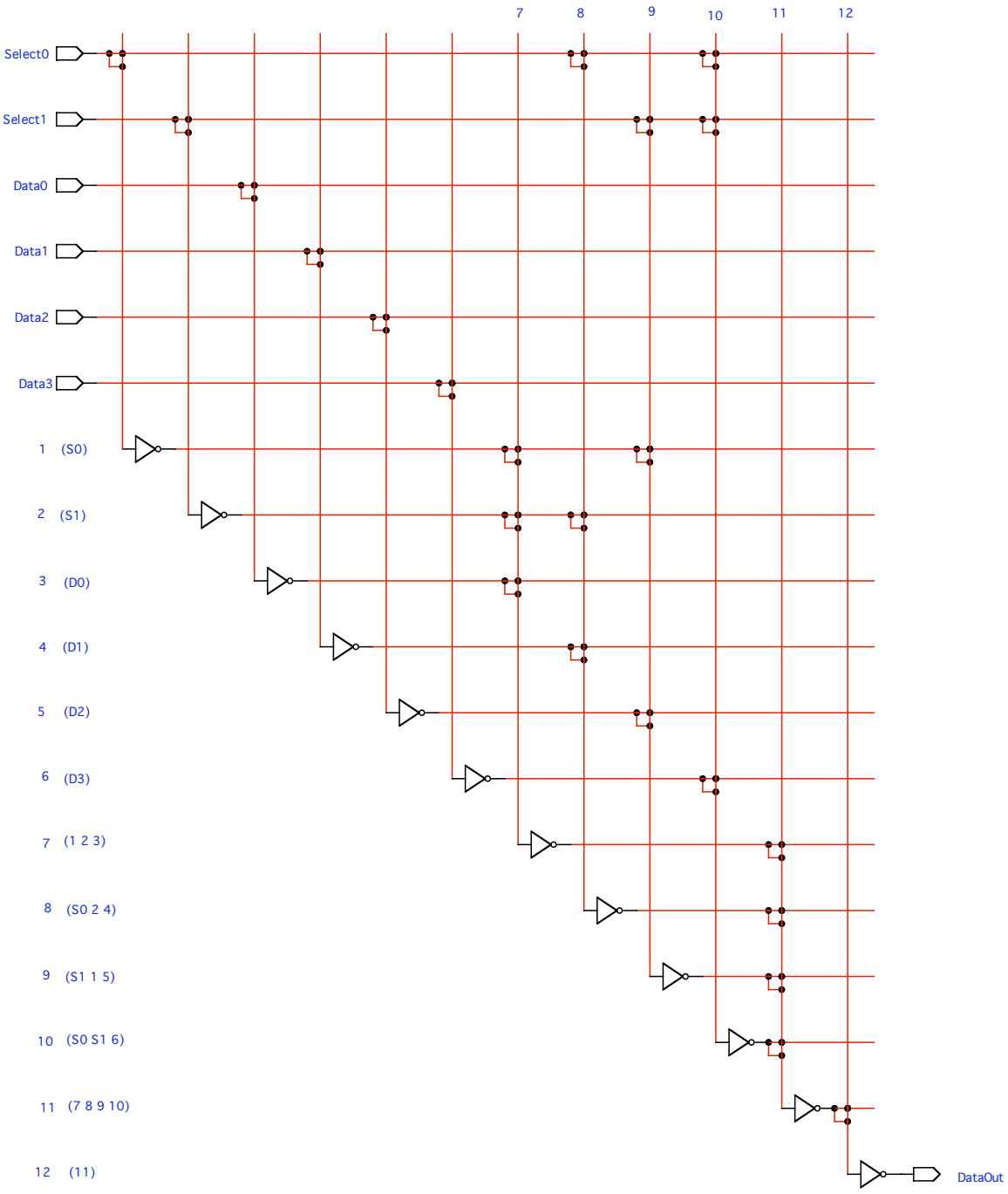
2-bit TALLY



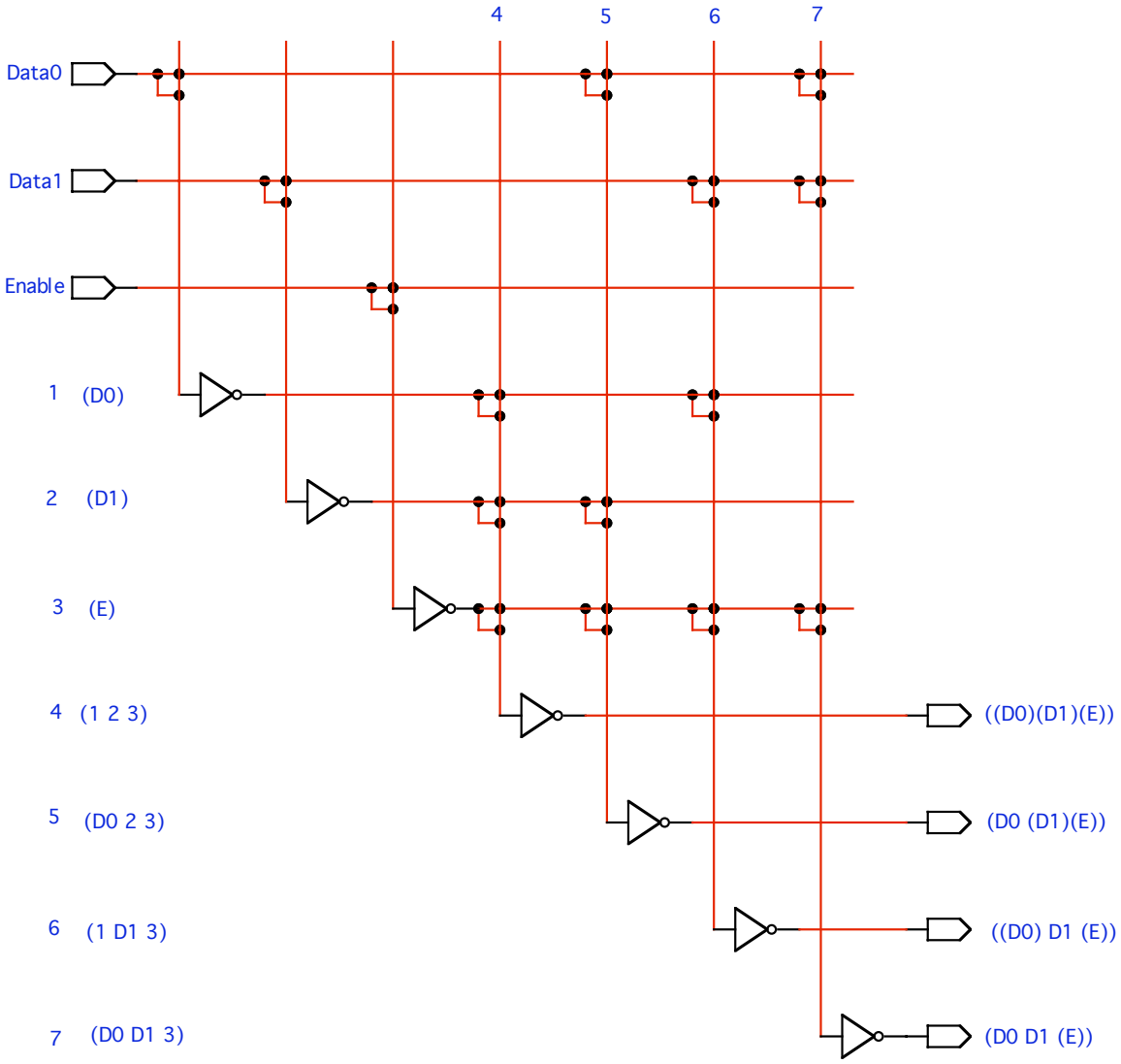
1to4 DEMULTIPLEXER



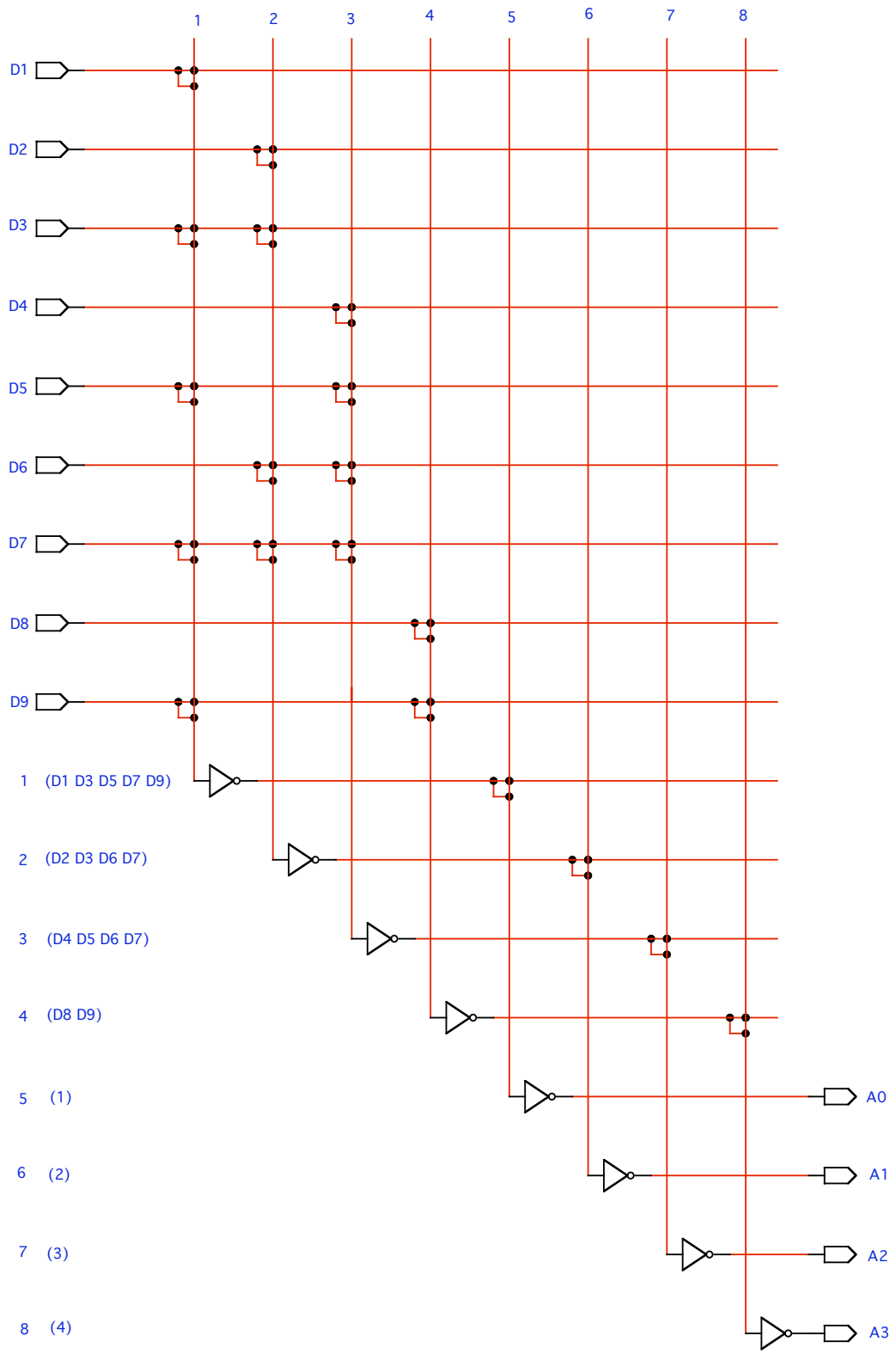
4to1 MULTIPLEXER



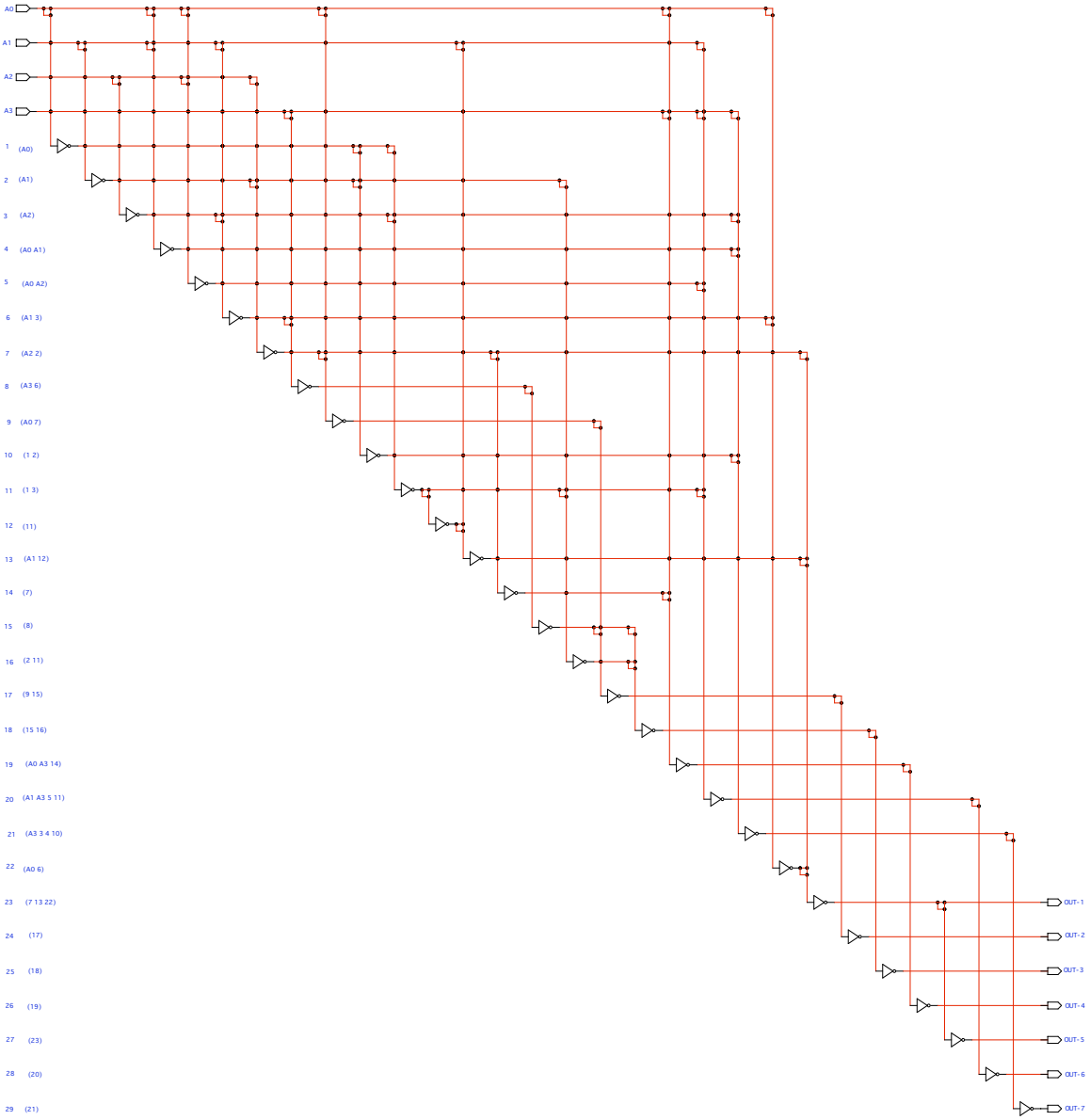
2to4 DECODER



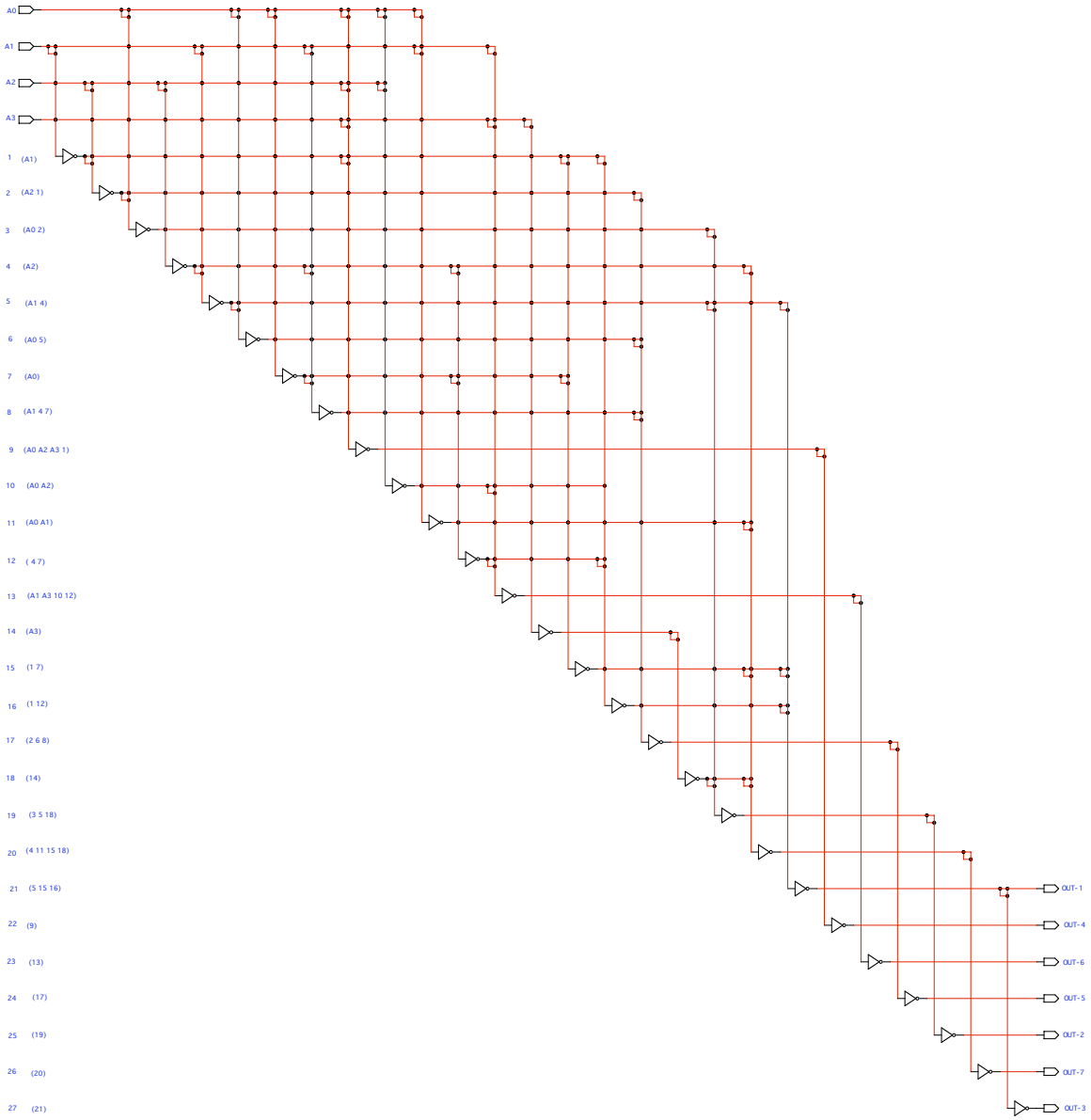
DECIMAL-TO-BCD ENCODER



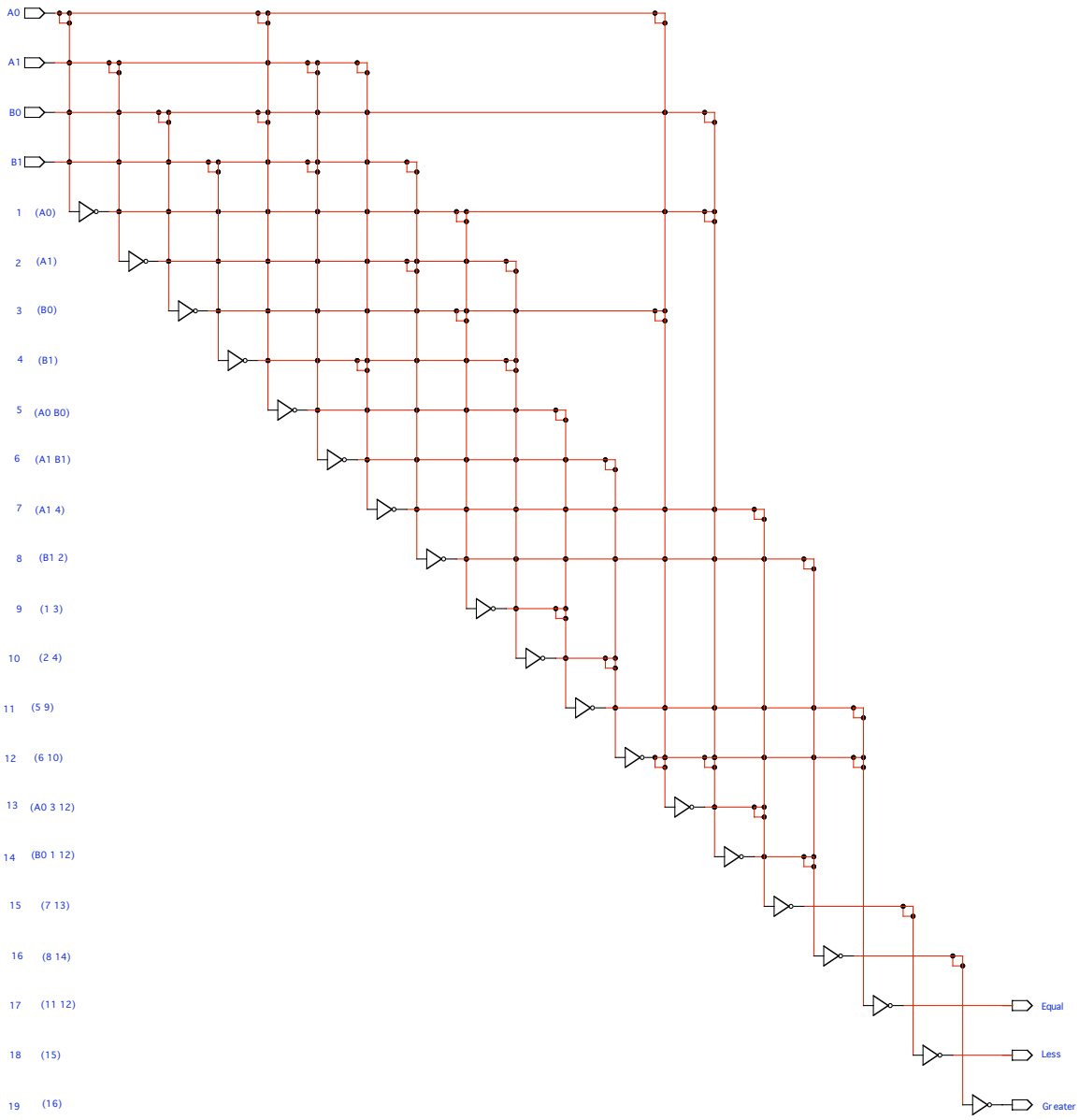
BCD-TO-7SEGMENT ENCODER



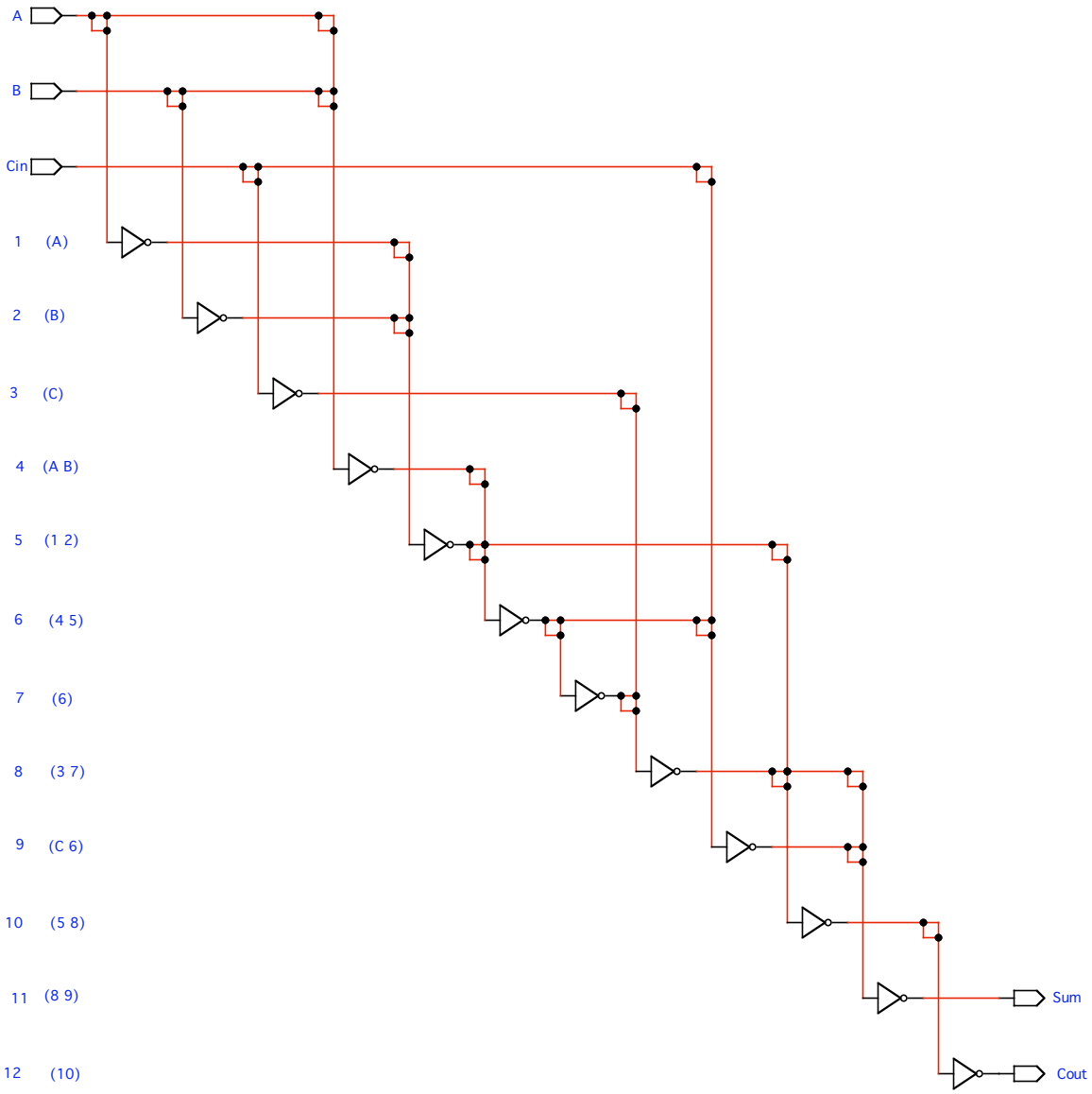
BCD-TO-7SEGMENT ENCODER *diagonalized*



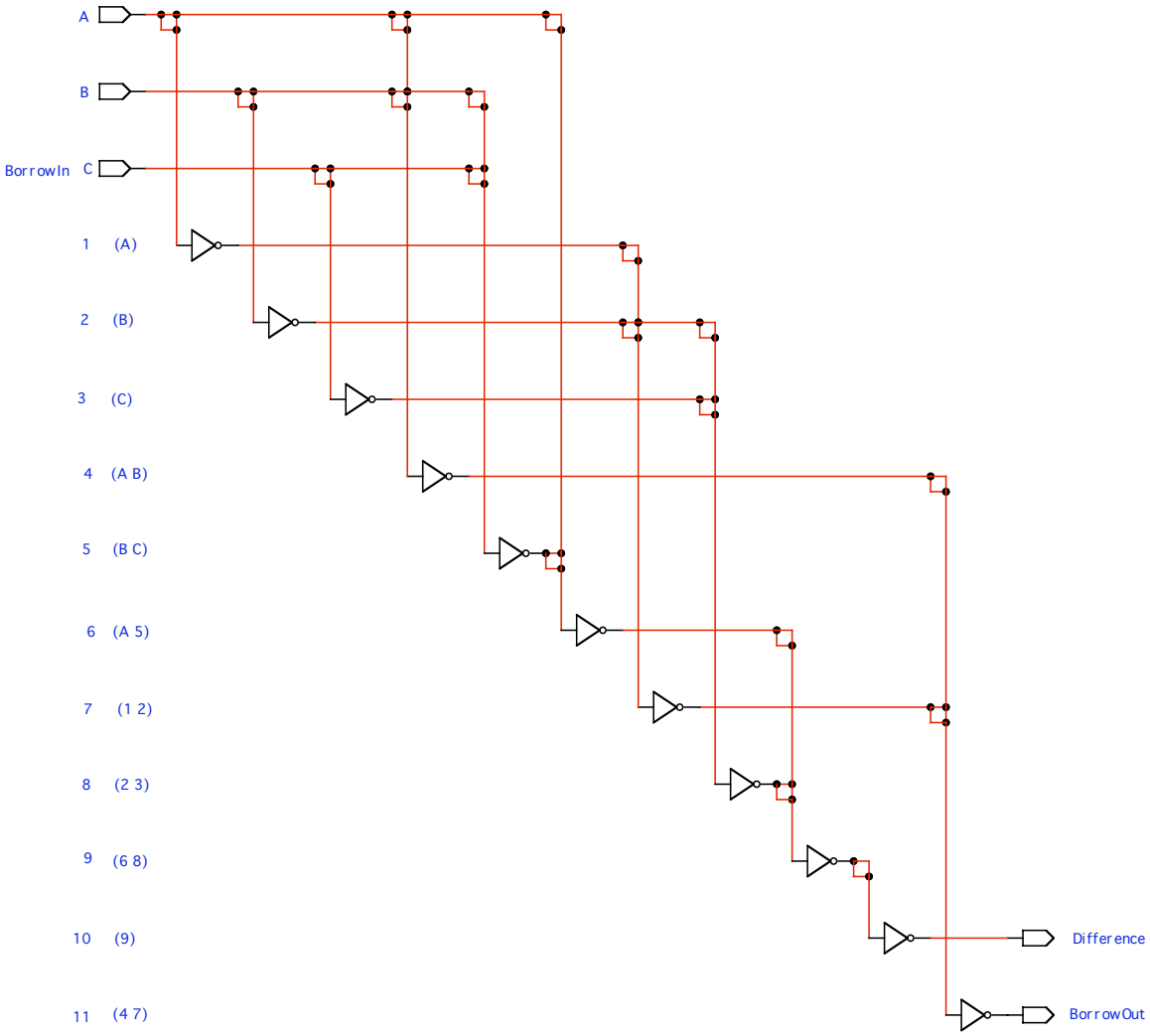
2-bit COMPARATOR



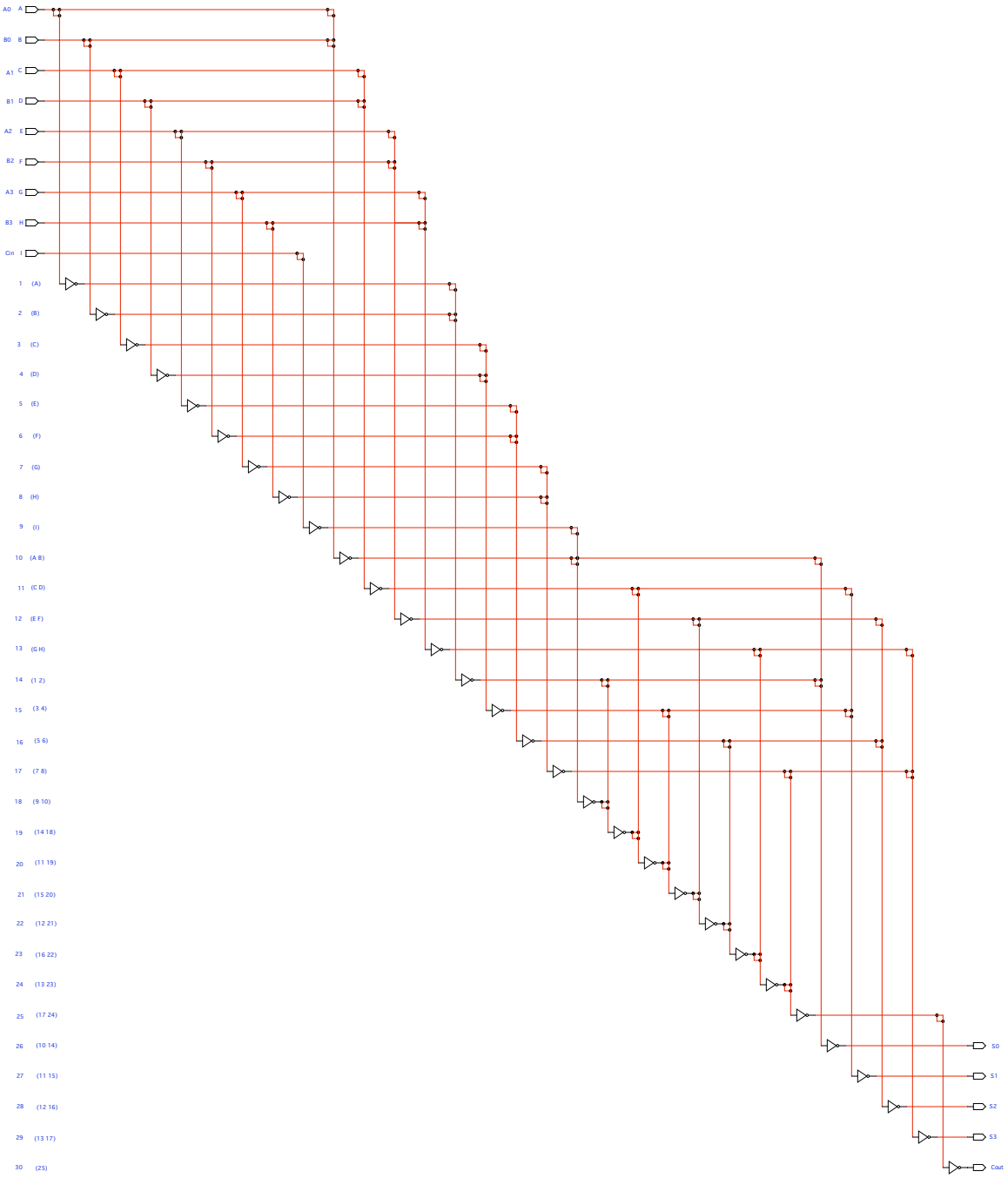
1-bit FULL-ADDER



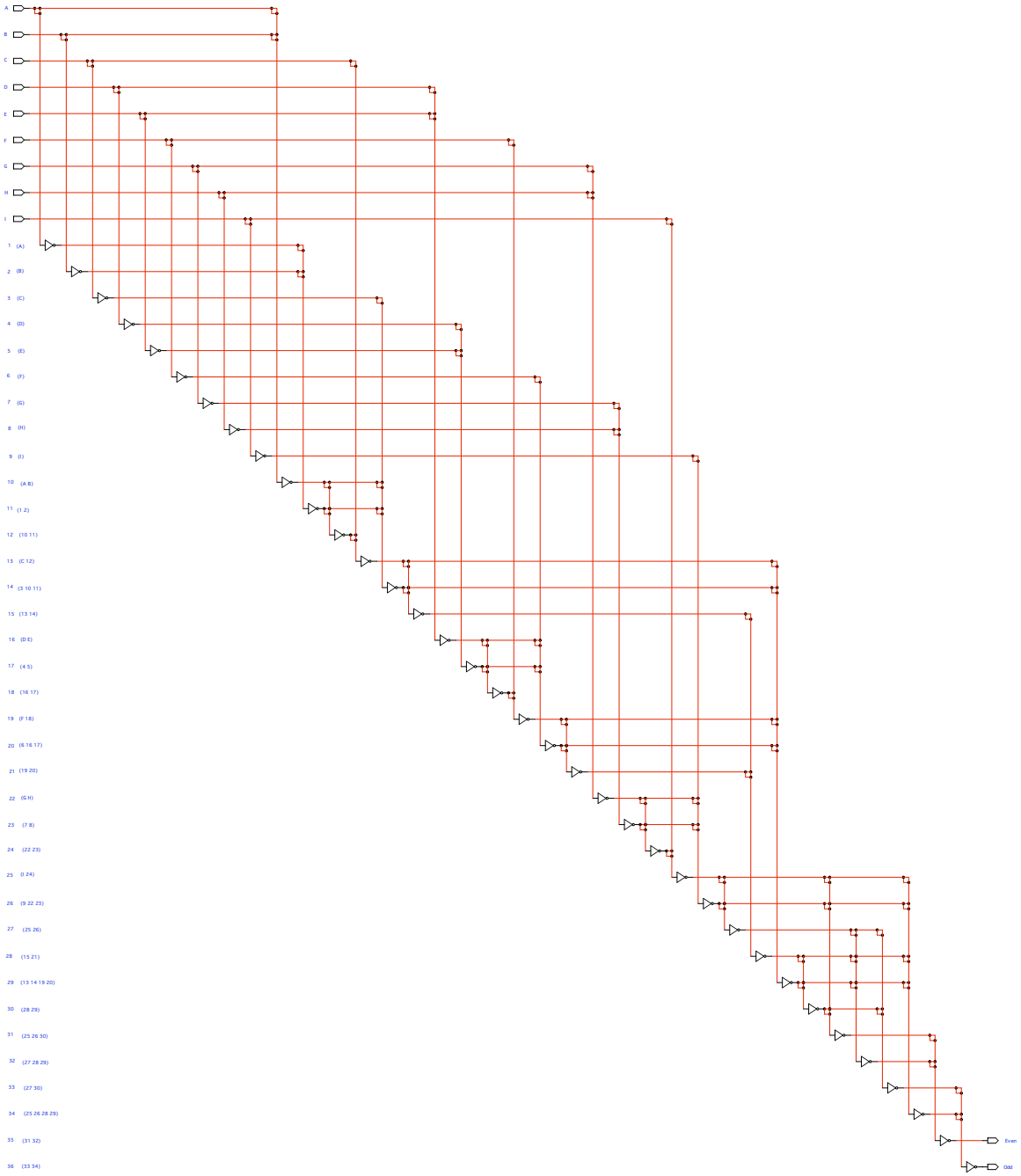
1-bit SUBTRACTOR



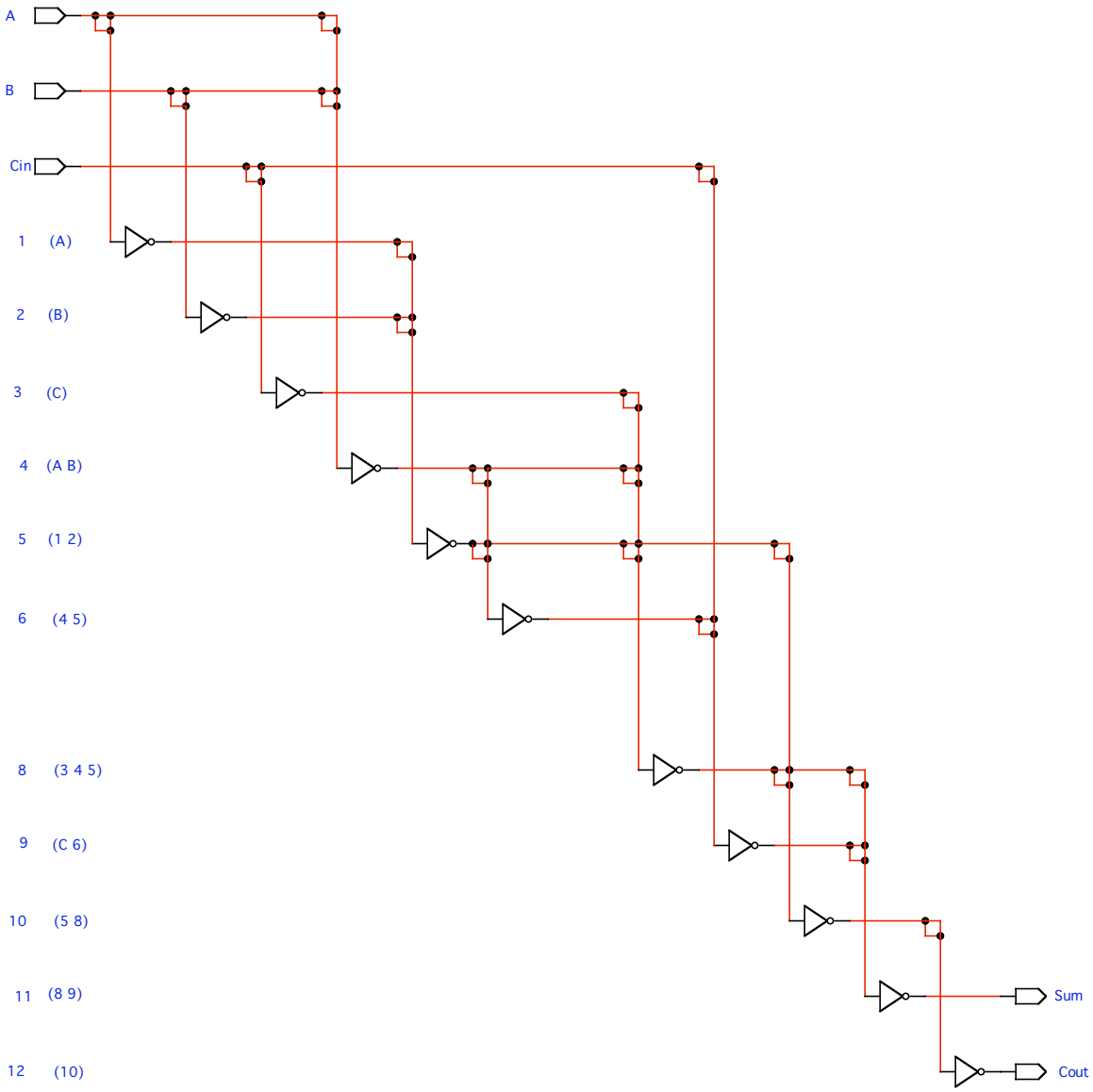
4-bit ADDER


A0 A
B0 B
A1 C
B1 D
A2 E
B2 F
A3 G
B3 H
Cin I
1 (A)
2 (B)
3 (C)
4 (D)
5 (E)
6 (F)
7 (G)
8 (H)
9 (I)
10 (A B)
11 (C D)
12 (E F)
13 (G H)
14 (1 2)
15 (3 4)
16 (5 6)
17 (7 8)
18 (9 10)
19 (14 18)
20 (11 19)
21 (15 20)
22 (12 21)
23 (16 22)
24 (13 23)
25 (17 24)
26 (10 14)
27 (11 15)
28 (12 16)
29 (13 17)
30 (25)
S0
S1
S2
S3
Cout

9-bit PARITY GENERATOR

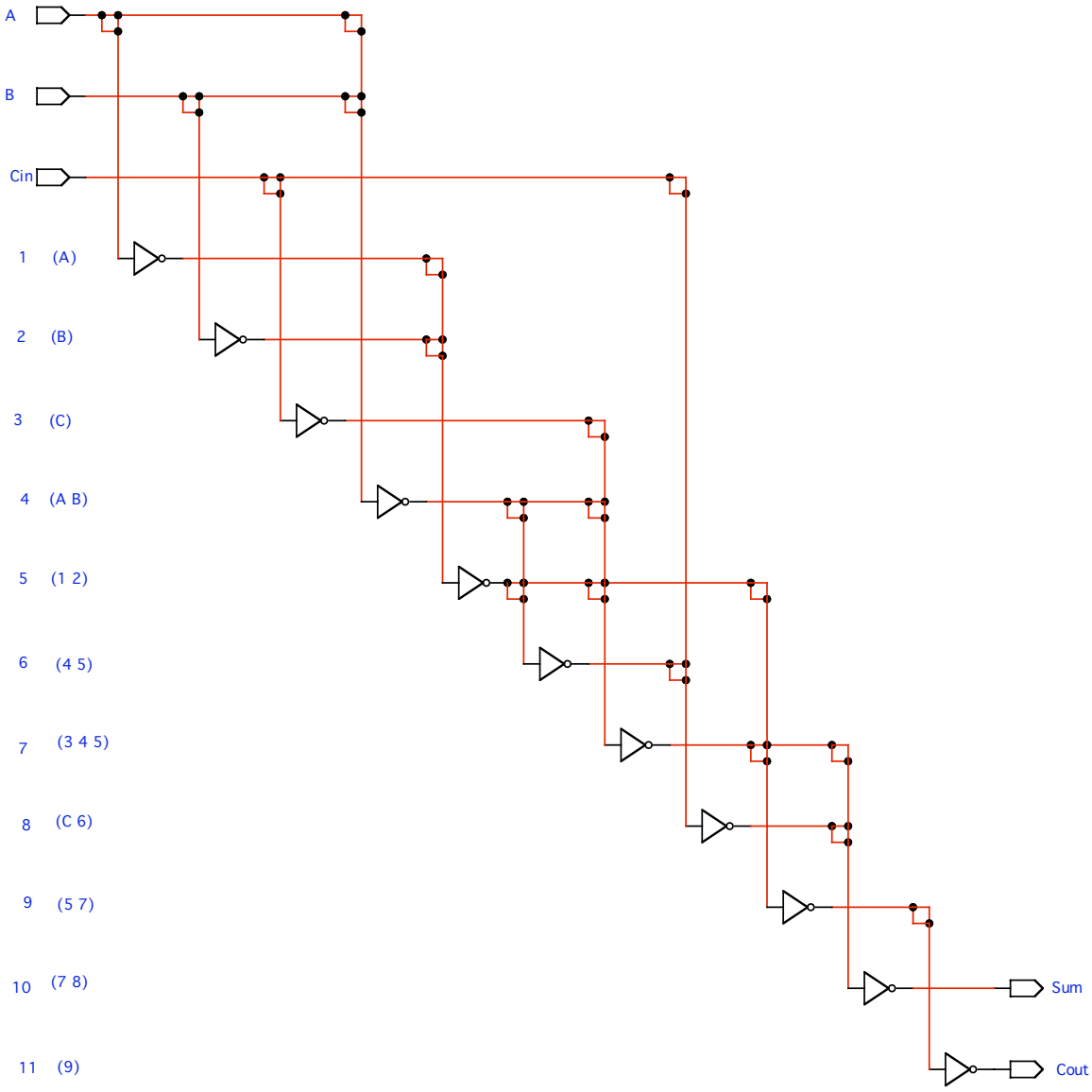


FULL-ADDER simplified

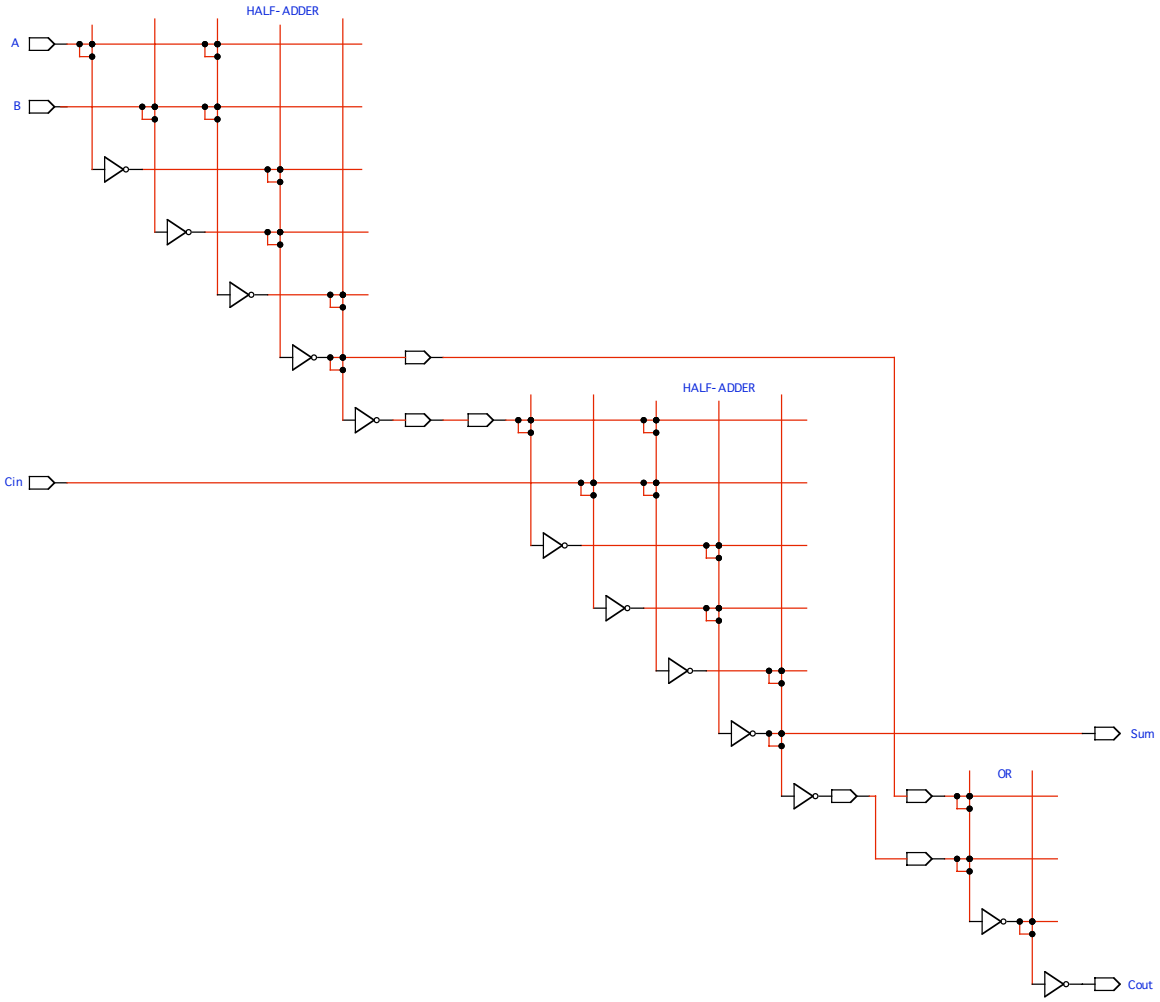


Structure sharing reduces the number of rows for a full-adder.

FULL-ADDER simplified and tidy

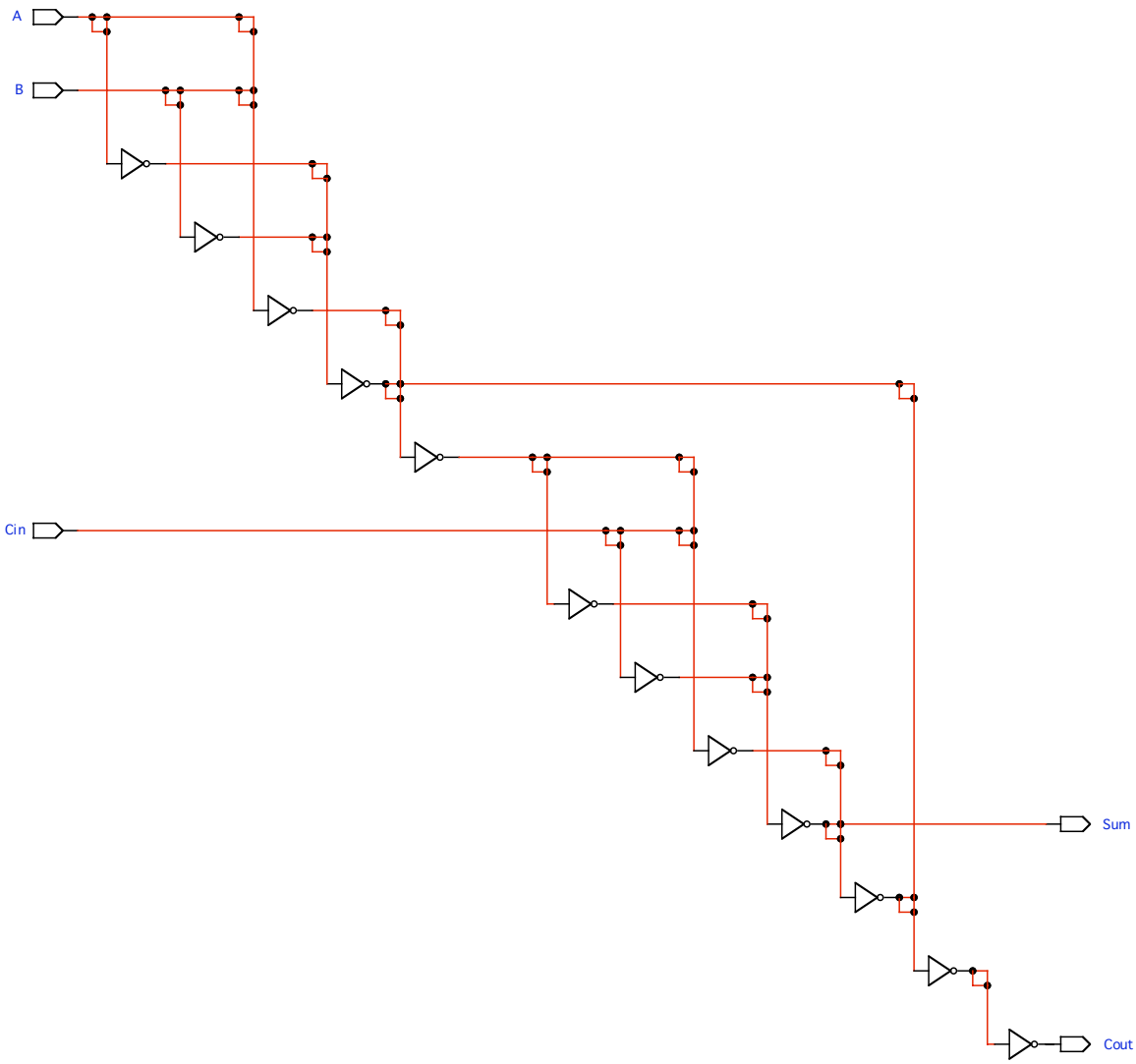


FULL-ADDER composed of two HALF-ADDERs



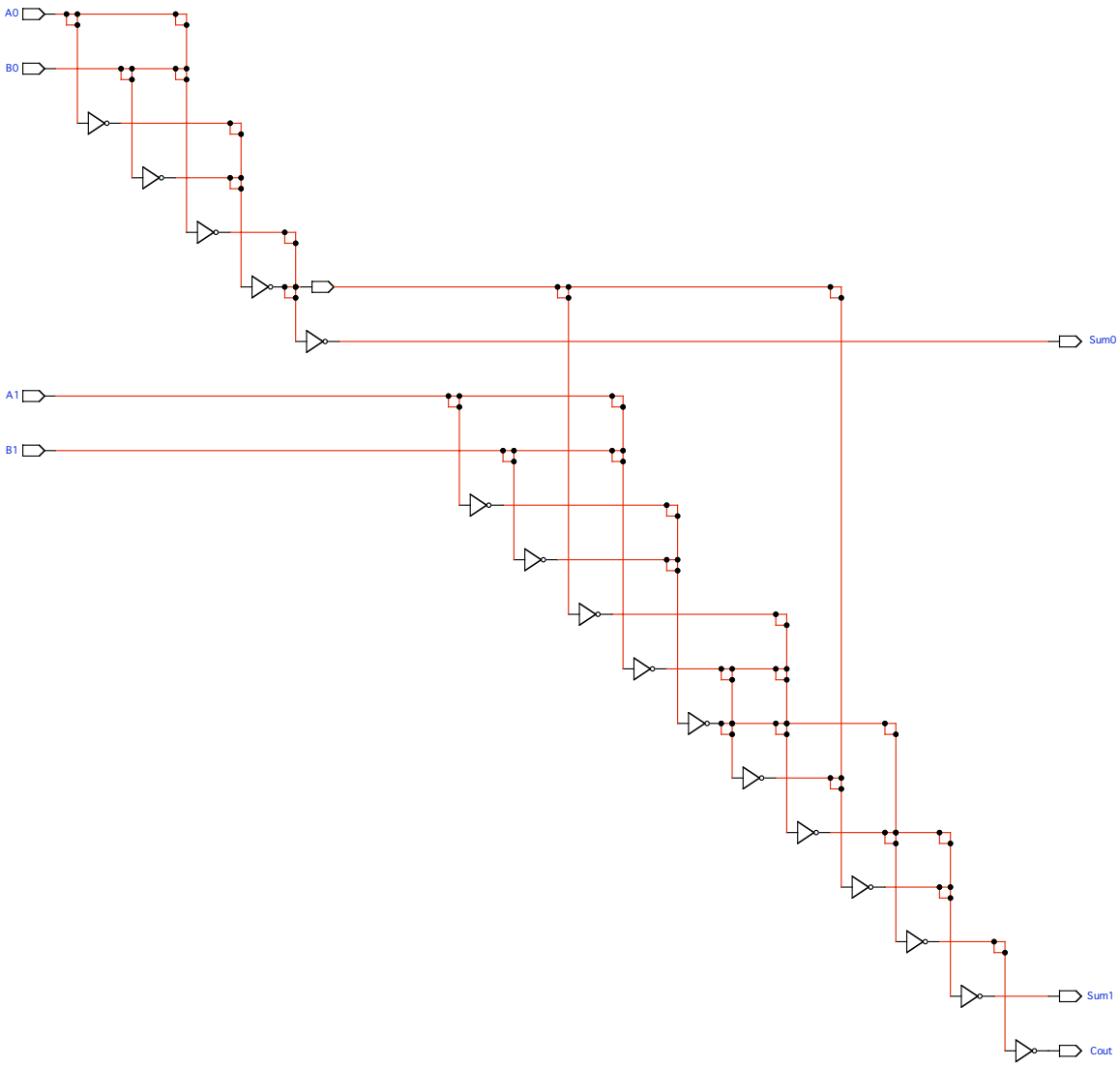
Two half-adders are ORed together by connecting Comesh i/o pins.

FULL-ADDER composed and tidy

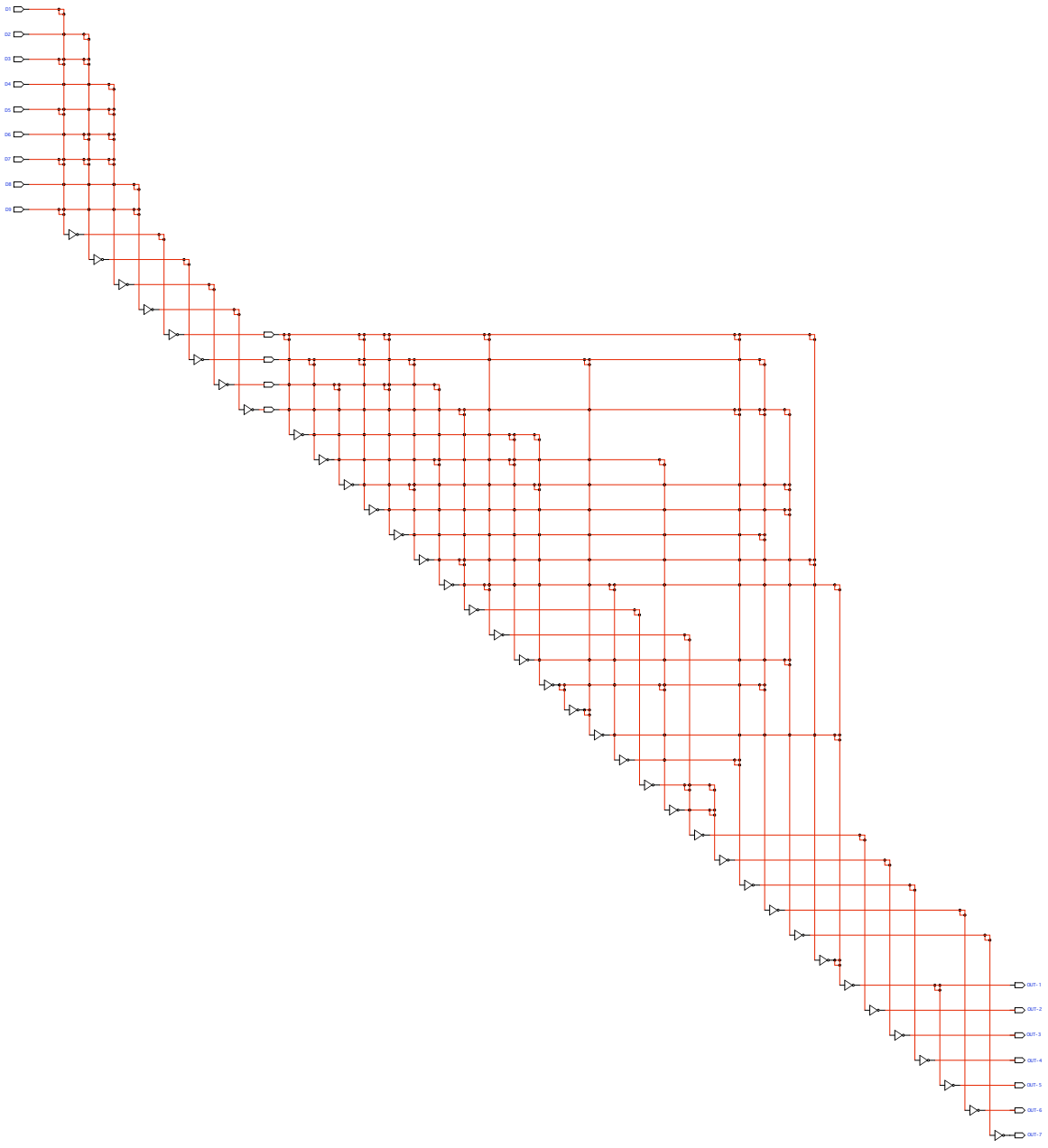


The composed full-adder with i/o pins removed.

2-bit ADDER composed of HALF-ADDER and FULL-ADDER

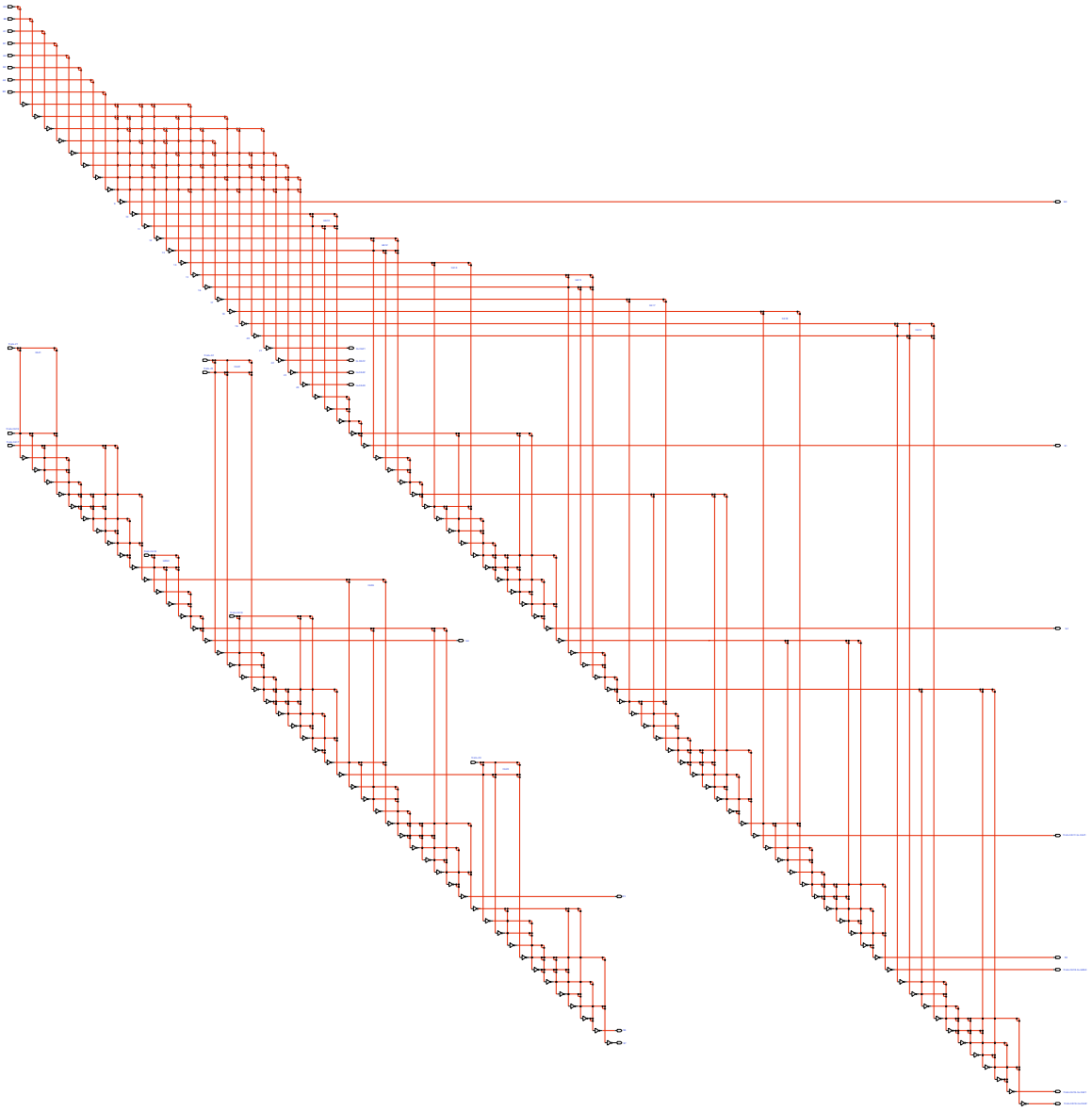


DECIMAL-TO-7SEGMENT-ENCODER composed



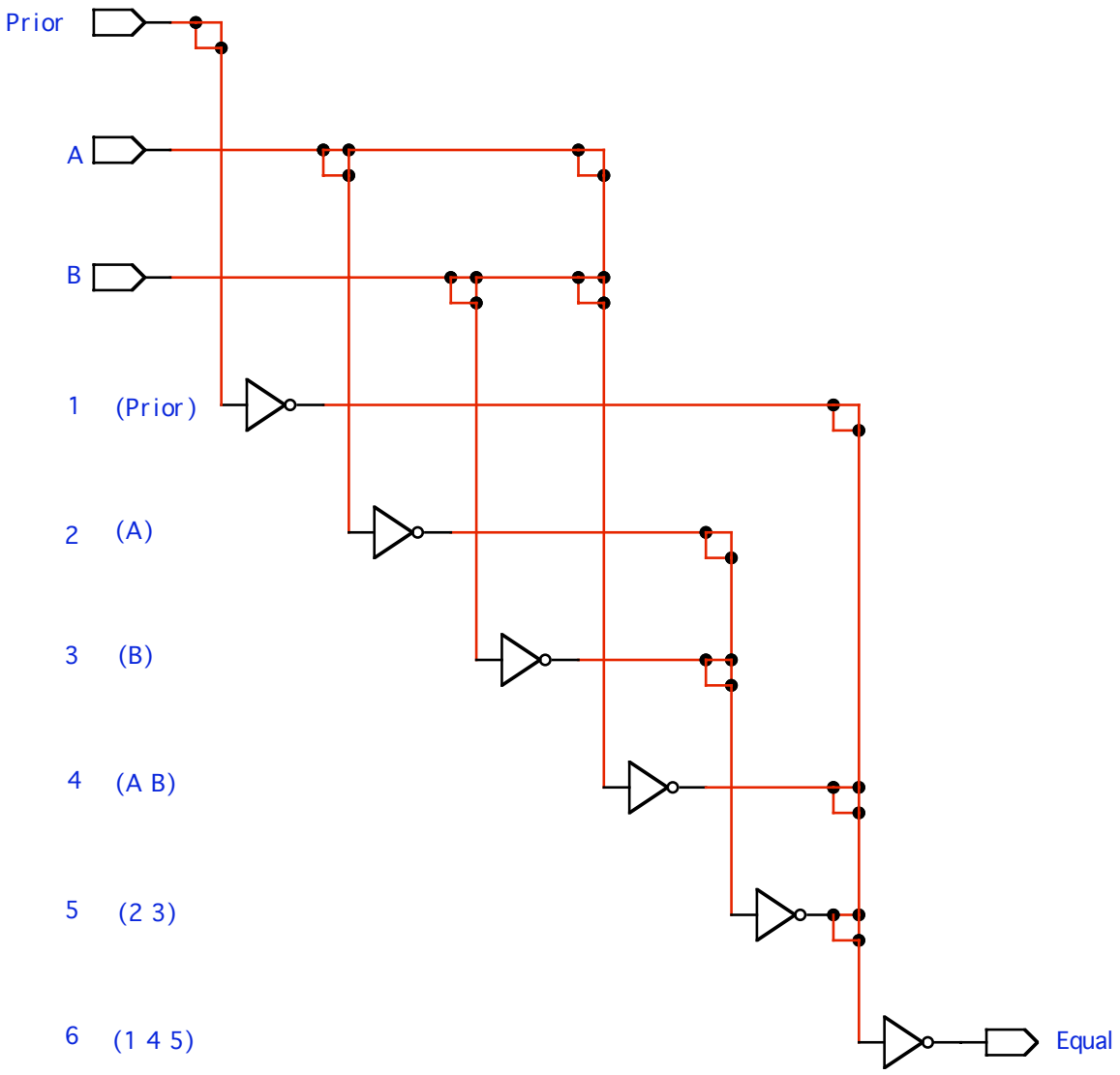
Decimal-to-BCD and BCD-to-7segment are composed by connecting i/o pins.

4-bit MULTIPLIER composed



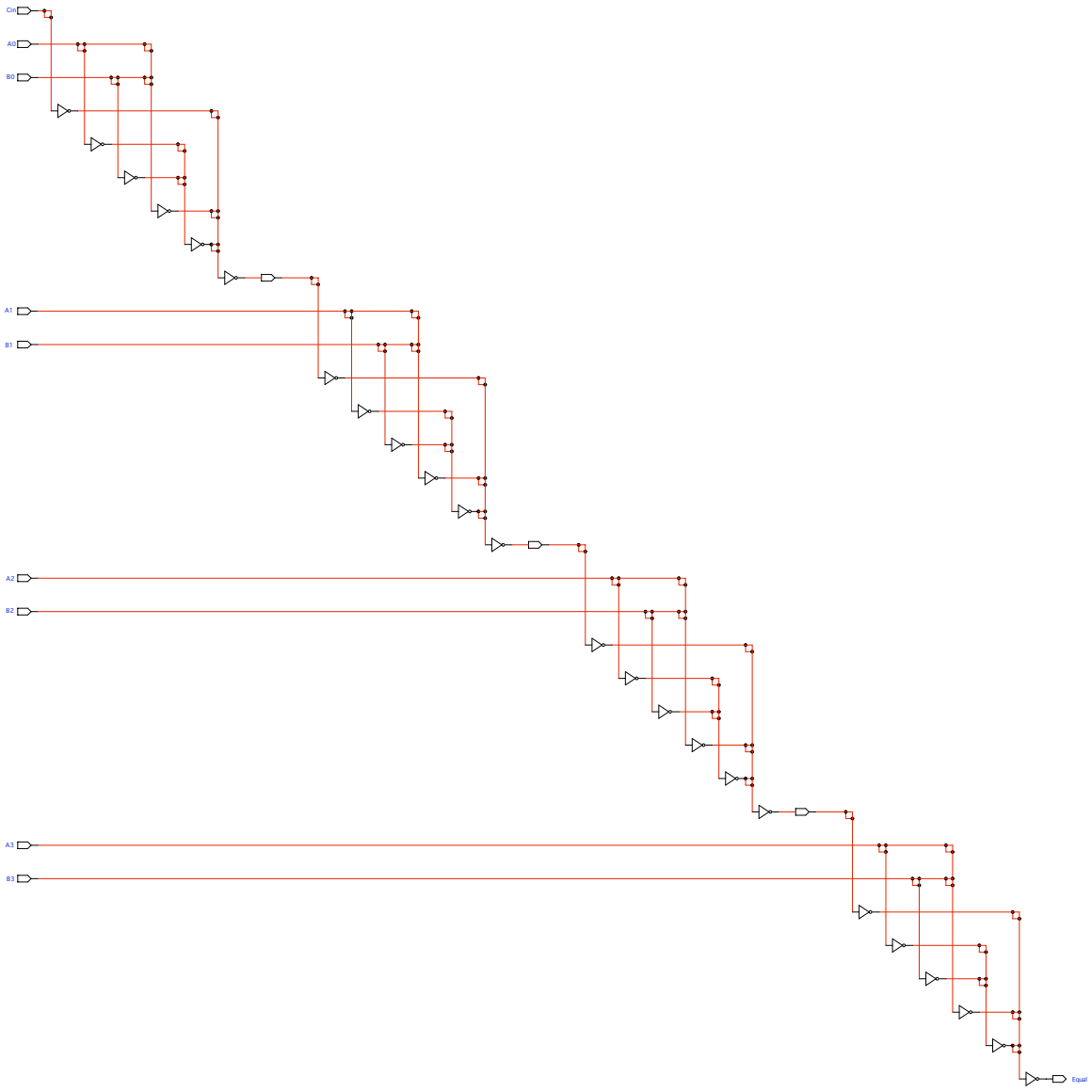
This illustration is too large to fit on one page; the discontinuous lower submesh is a continuation of the upper submesh.

1-bit COMPARATOR iterated



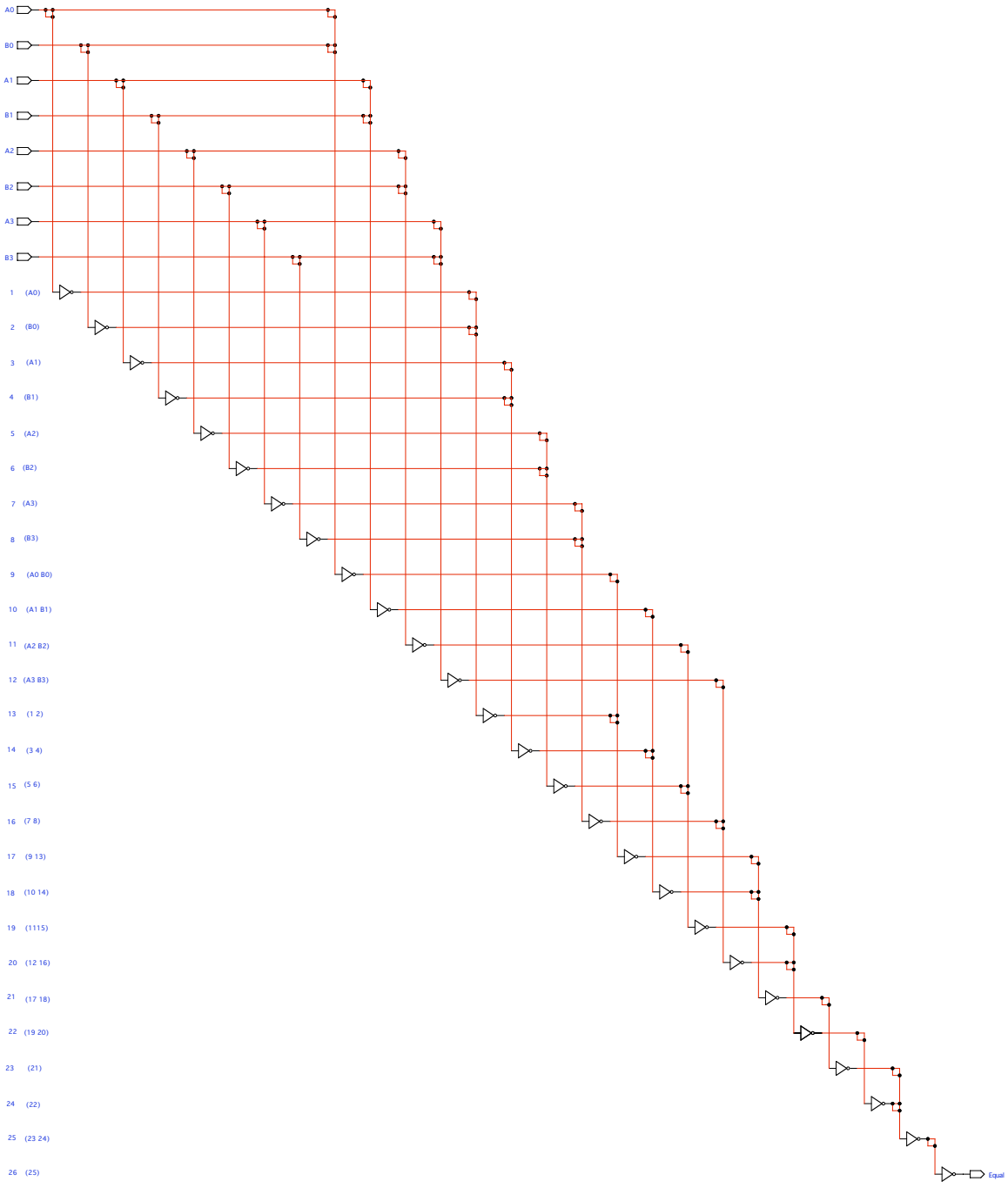
An iterative unit for an N-bit comparator.

4-bit COMPARATOR iteratively composed



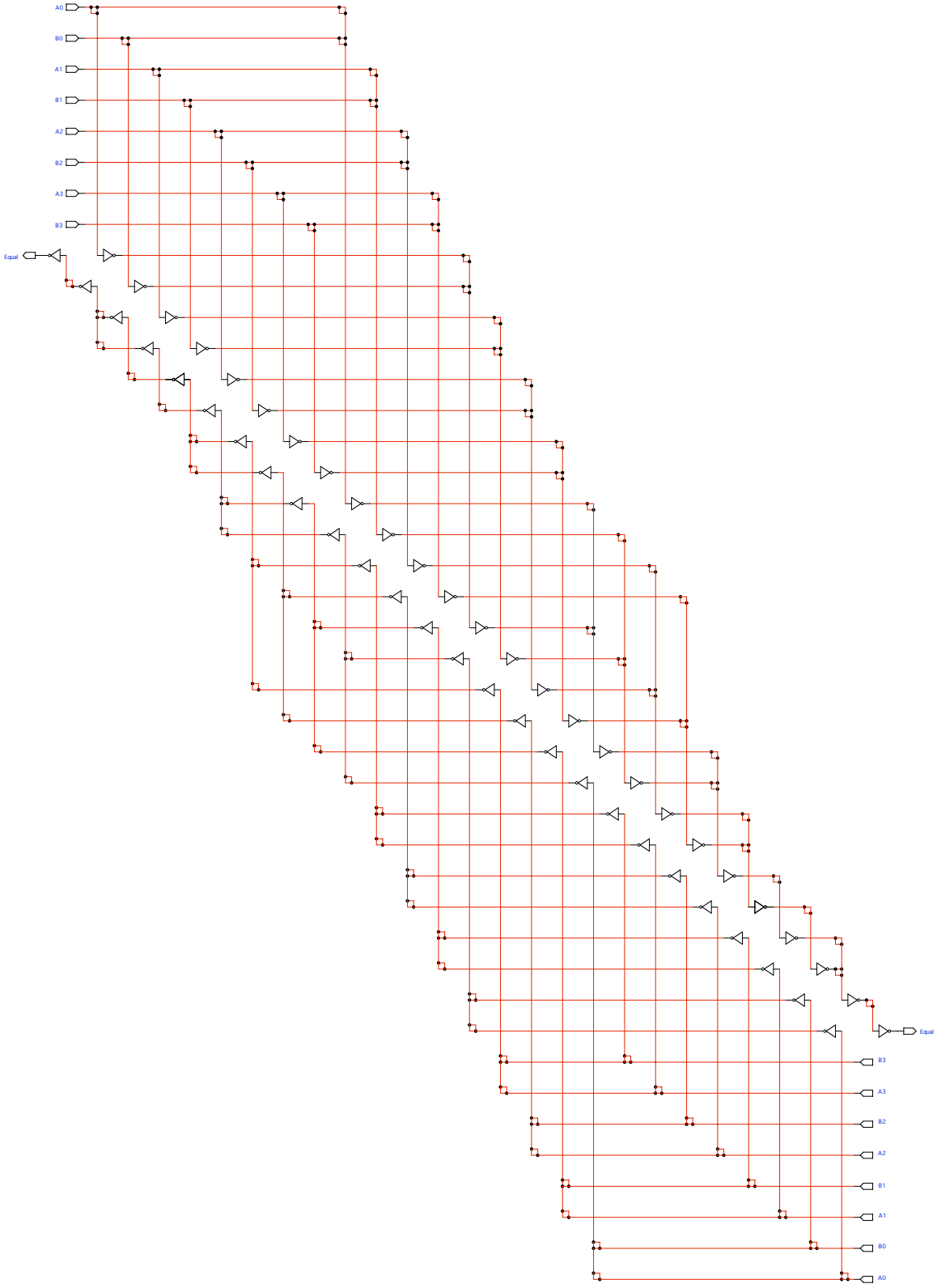
Composing four unit comparators to yield a 4-bit comparator.

4-bit PARALLEL COMPARATOR

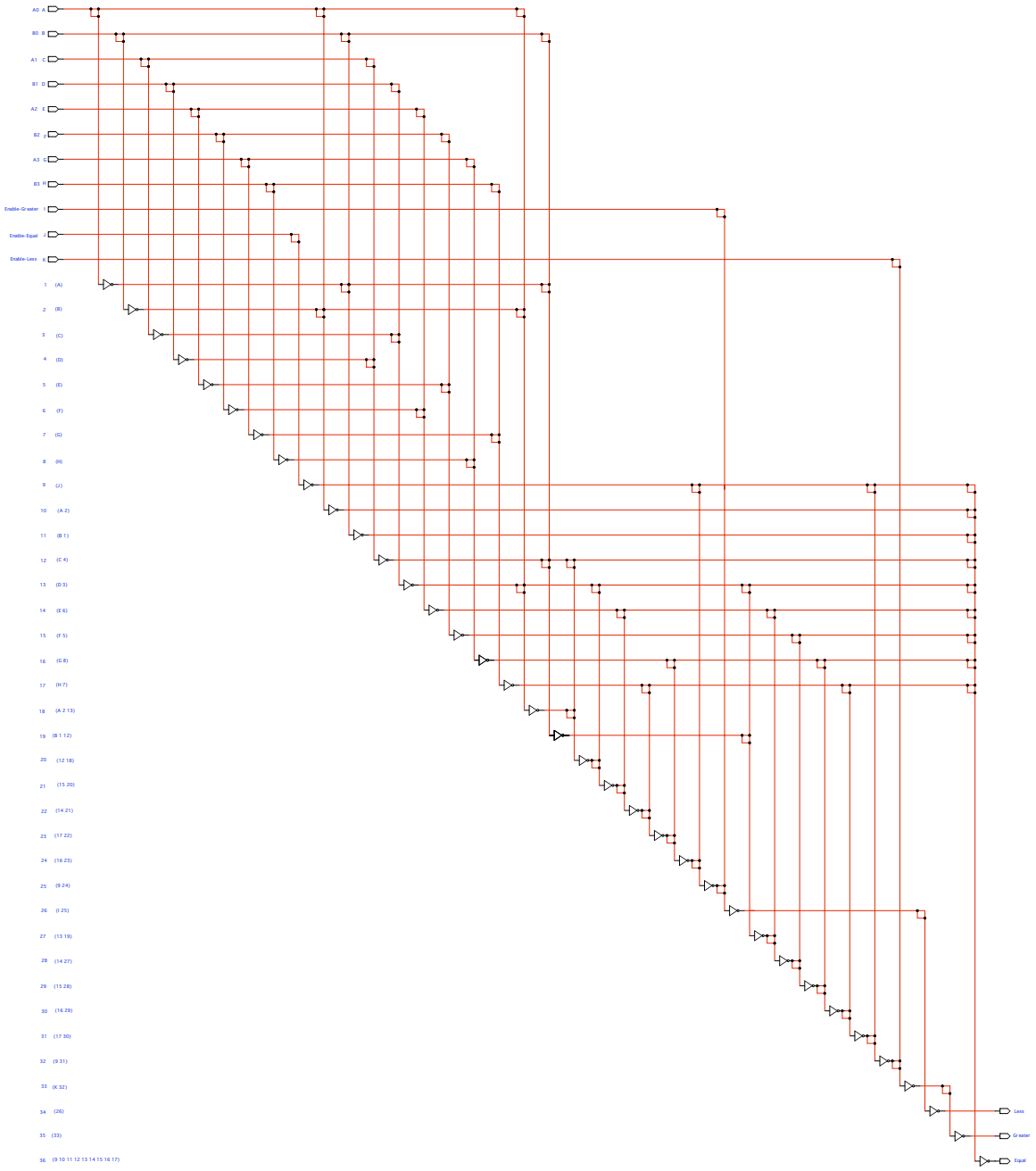


In contrast to composition, this 4-bit comparator has been optimized by Losp.

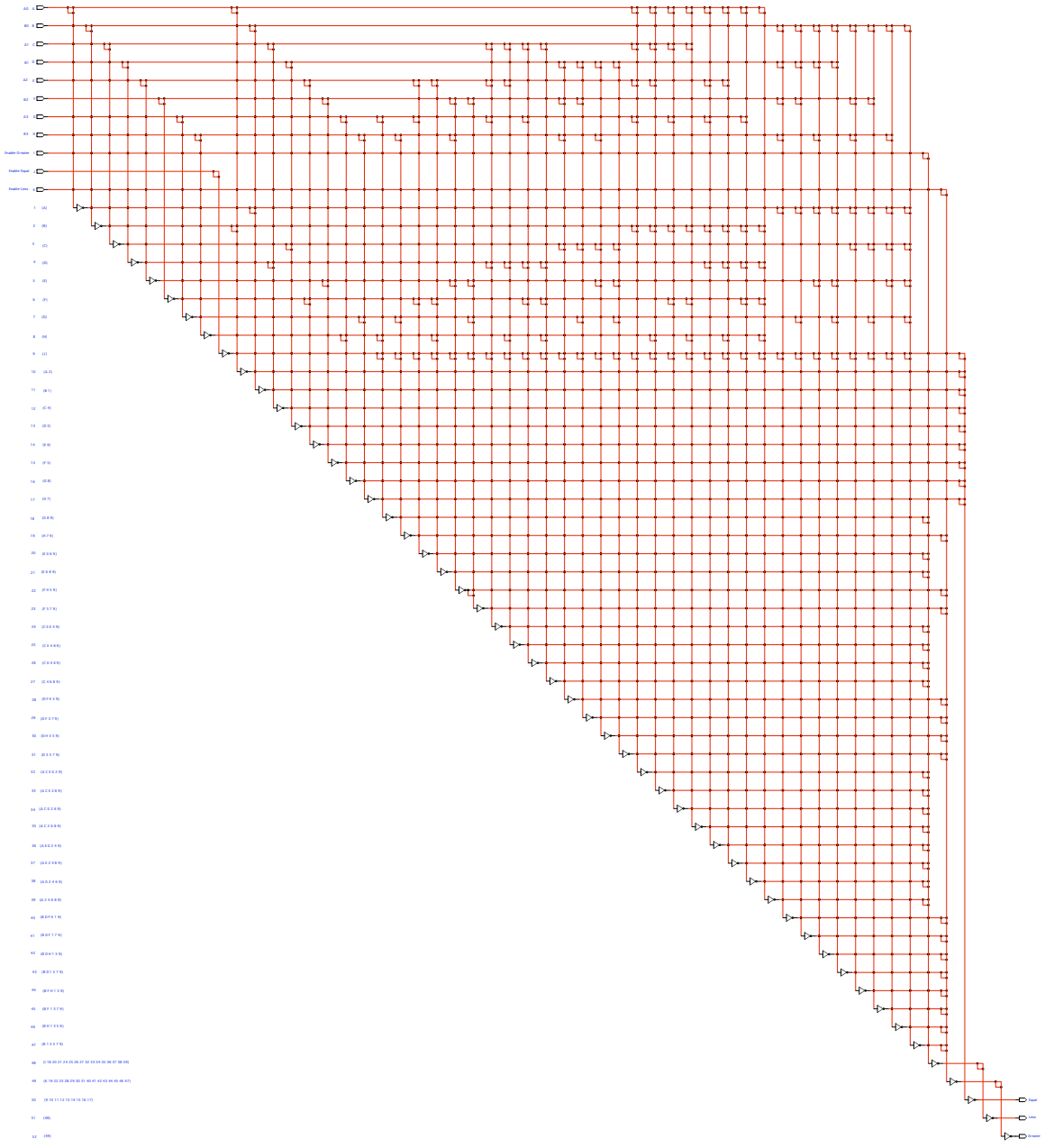
4-bit PARALLEL COMPARATOR *SQUARE (2 comparators)*



4-bit MAGNITUDE COMPARATOR with enables

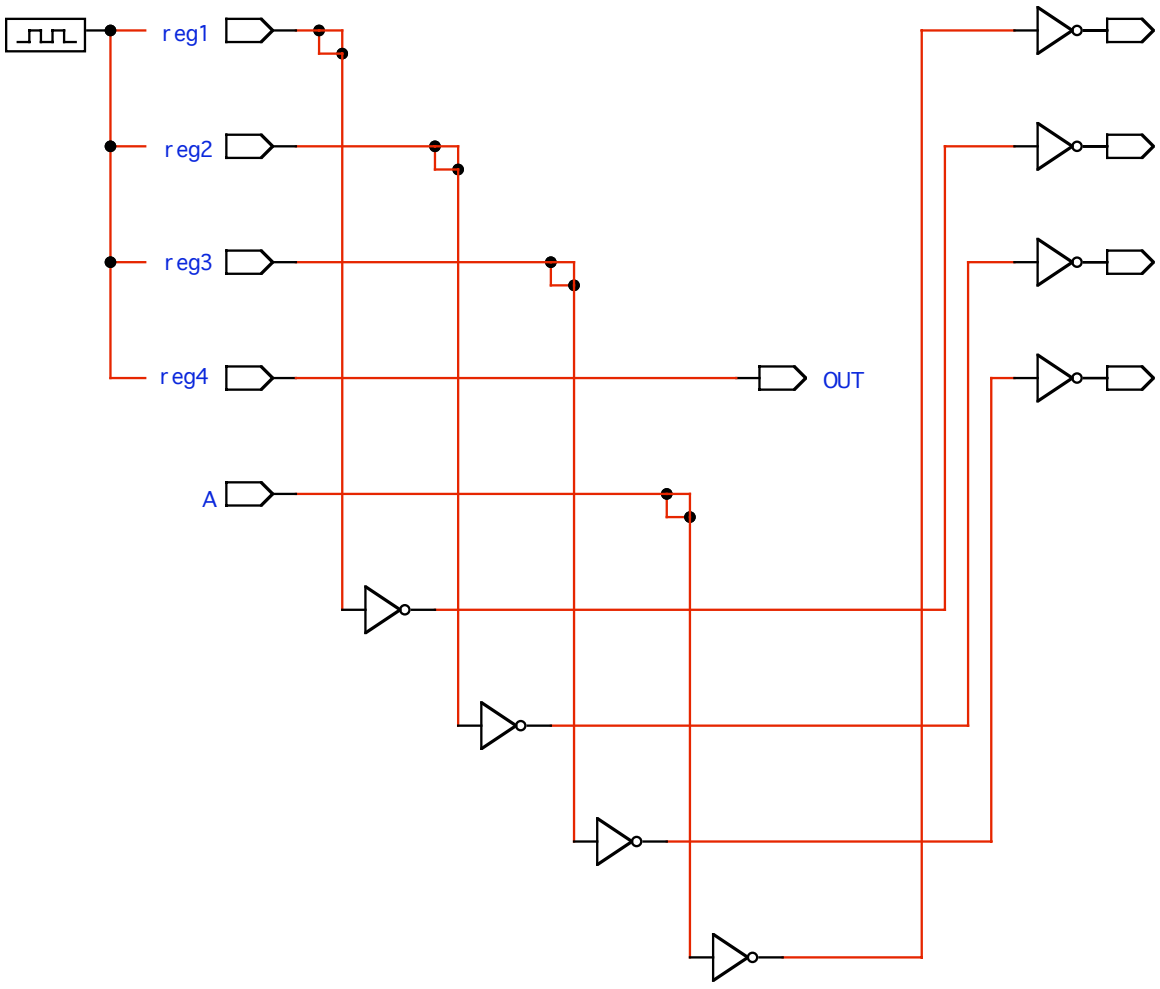


4-bit MAGNITUDE COMPARATOR SOP

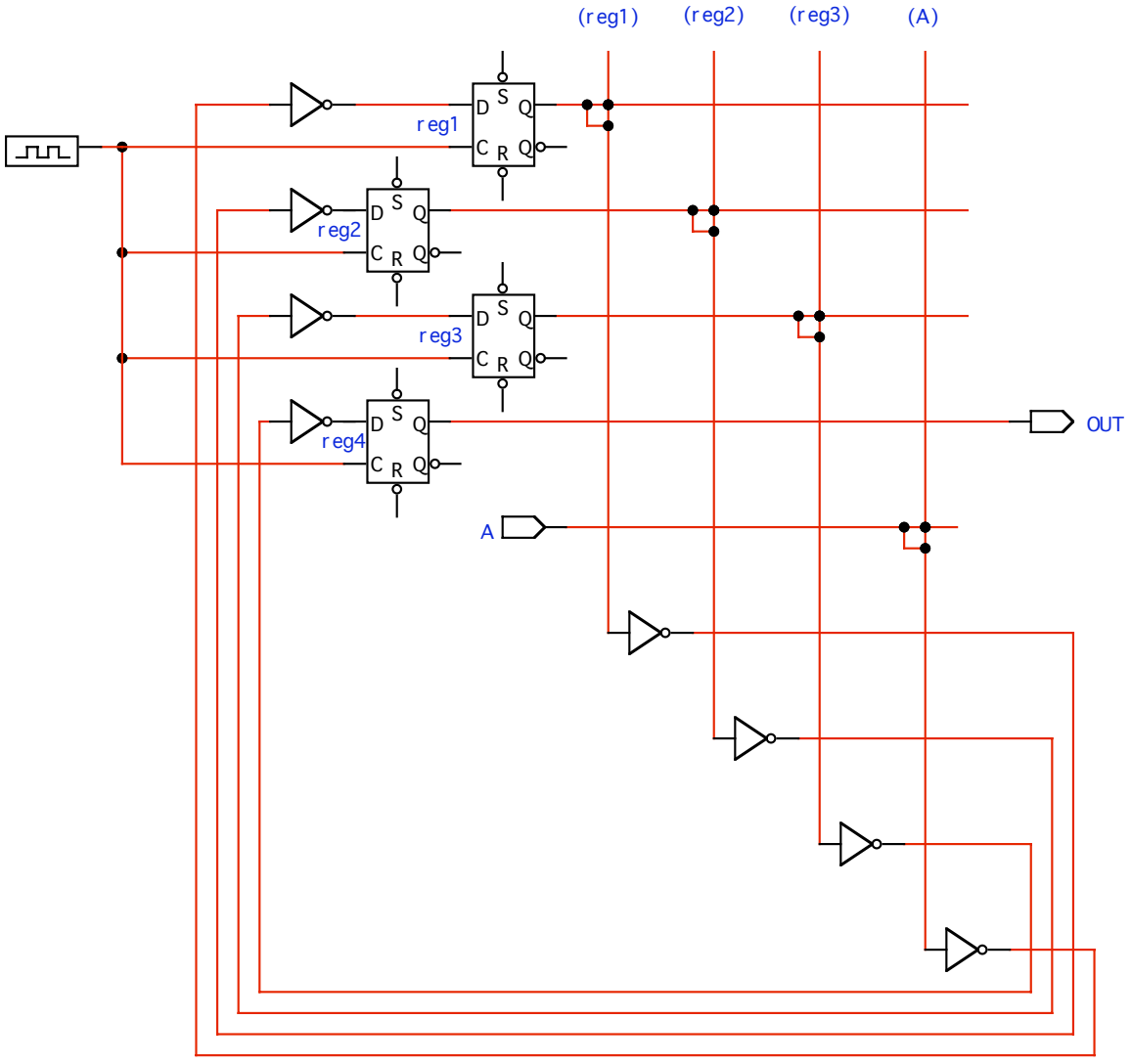


The two-level SOP form restricts all connections to the input rows and to three columns.

4-bit SHIFT-REGISTER serial-in, serial-out

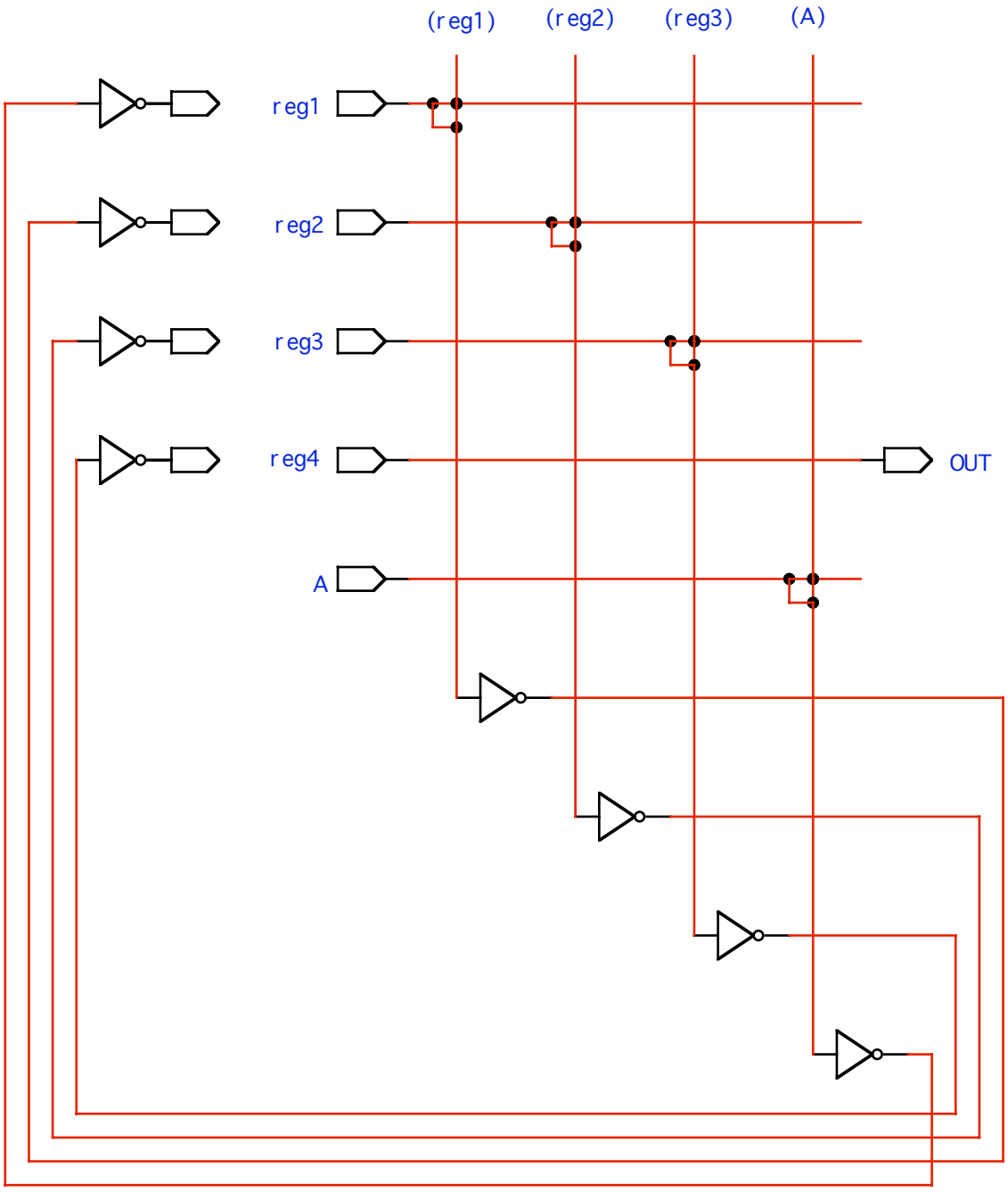


4-bit SERIAL SHIFT-REGISTER with register bank



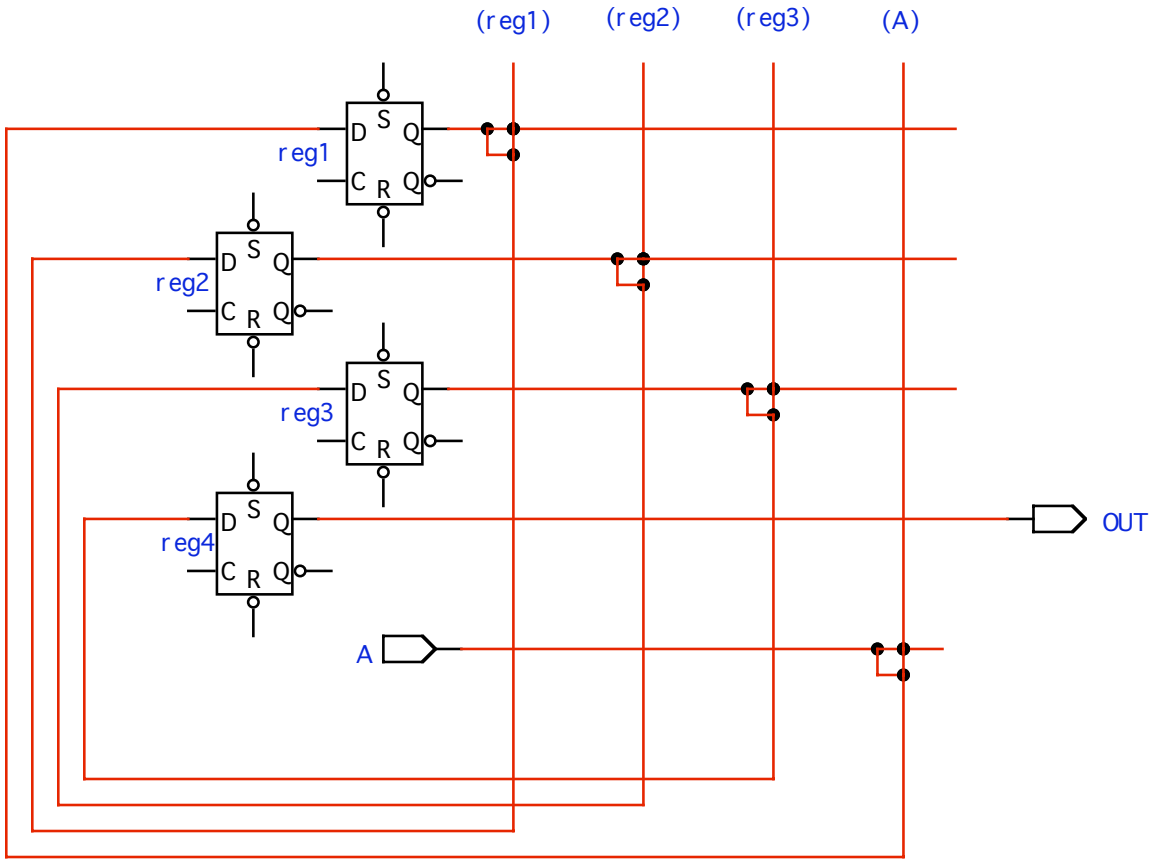
A more explicit illustration of the shift register, showing the register bank and the clock.

4-bit SERIAL SHIFT-REGISTER with looped connection



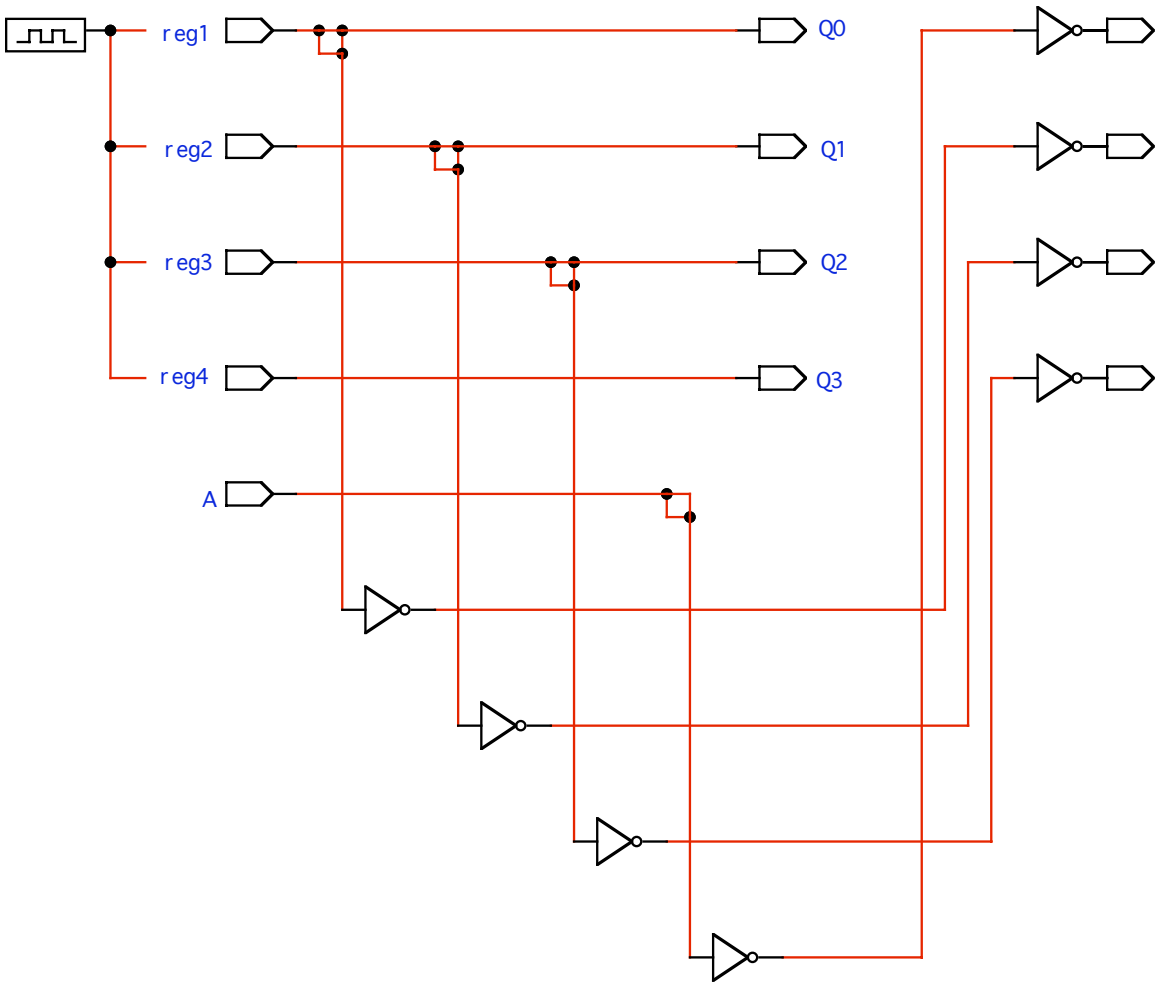
Another illustration of the shift-register, showing the loop wiring for re-entering register values.

4-bit SERIAL SHIFT-REGISTER, direct wired, not inverted

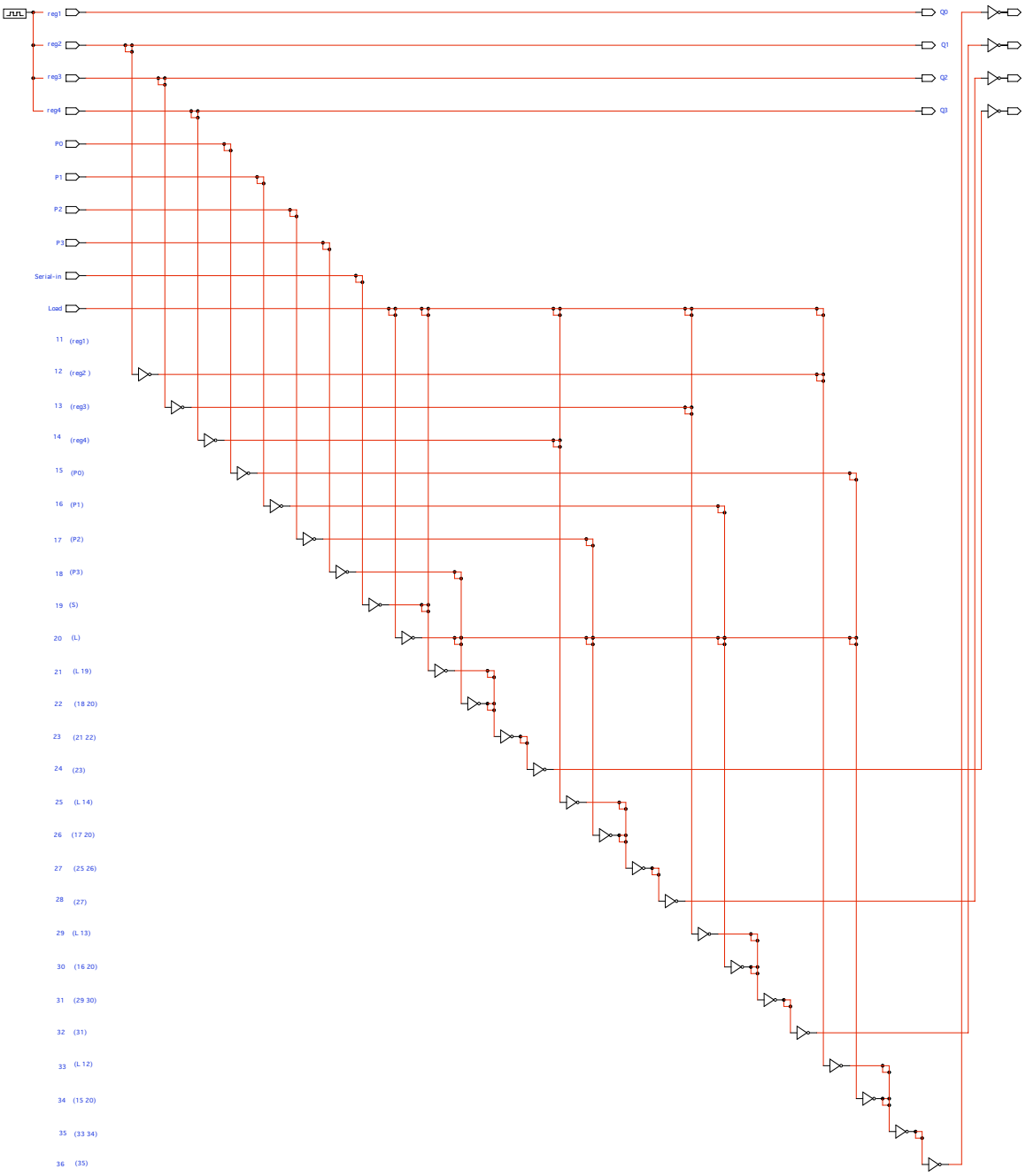


Direct wiring of registers avoids the double inversion while undermining the homogeneous structure of the mesh.

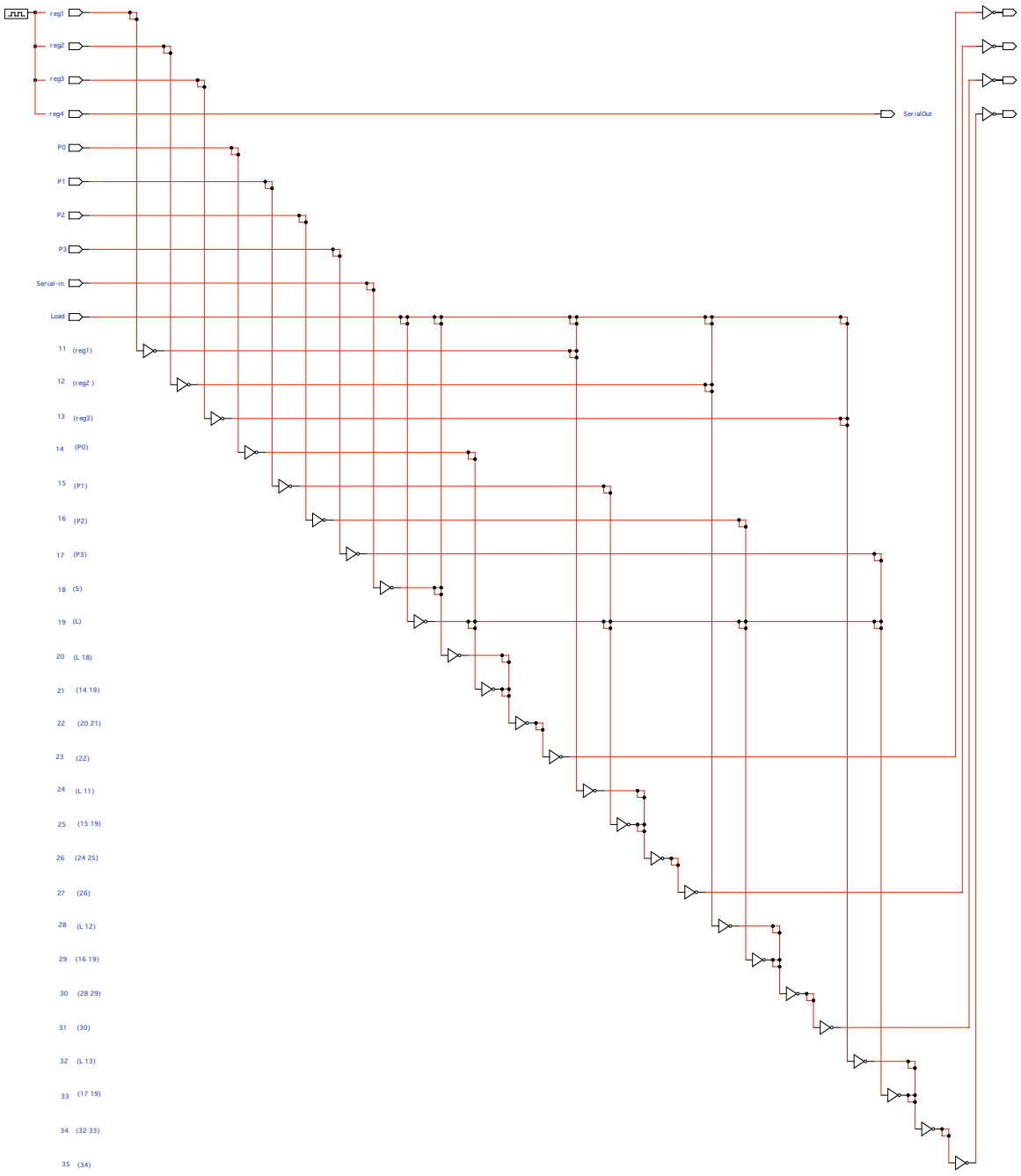
4-bit SHIFT-REGISTER serial-in, parallel-out



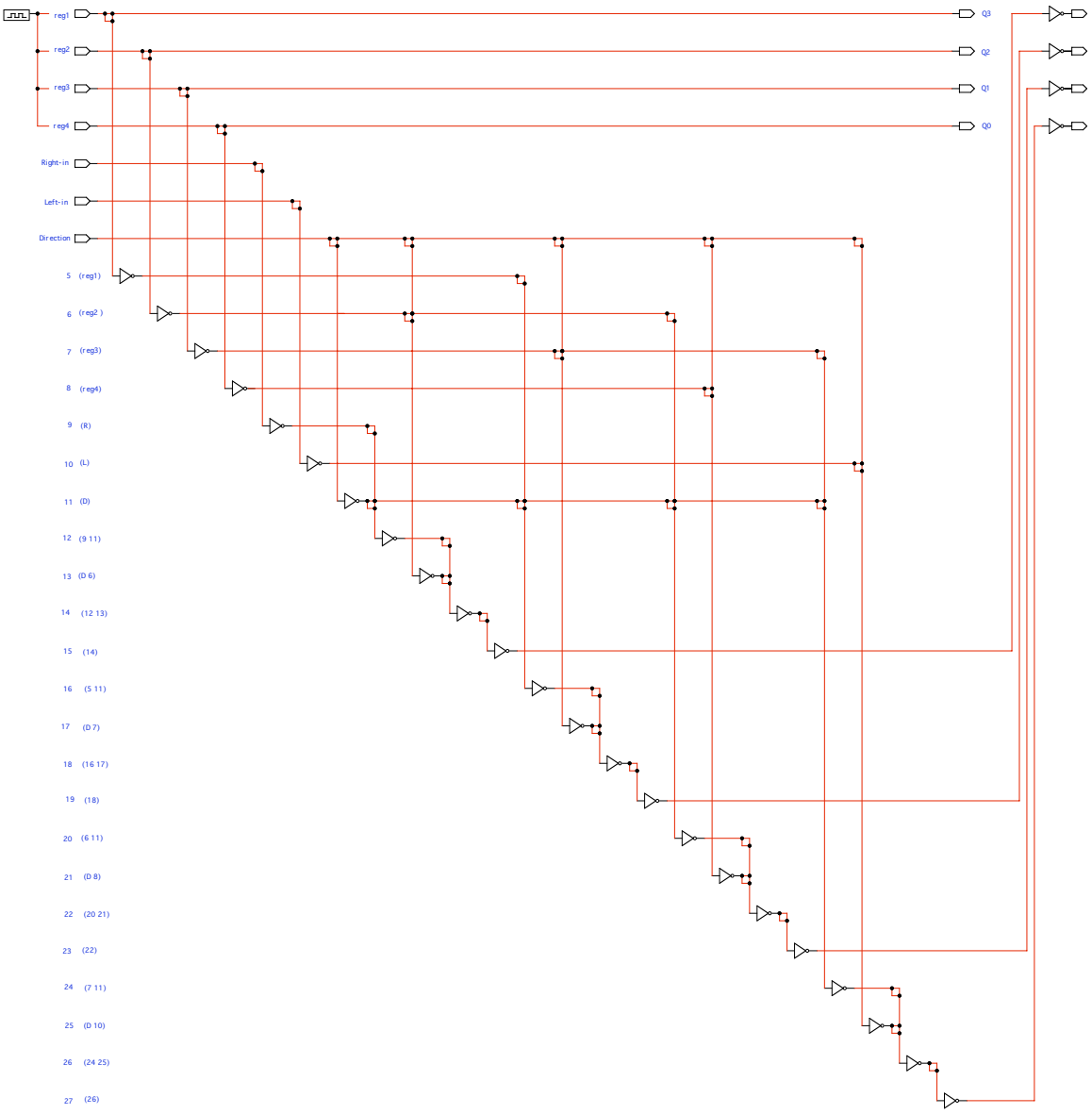
4-bit SHIFT-REGISTER serial-in, parallel-out with load



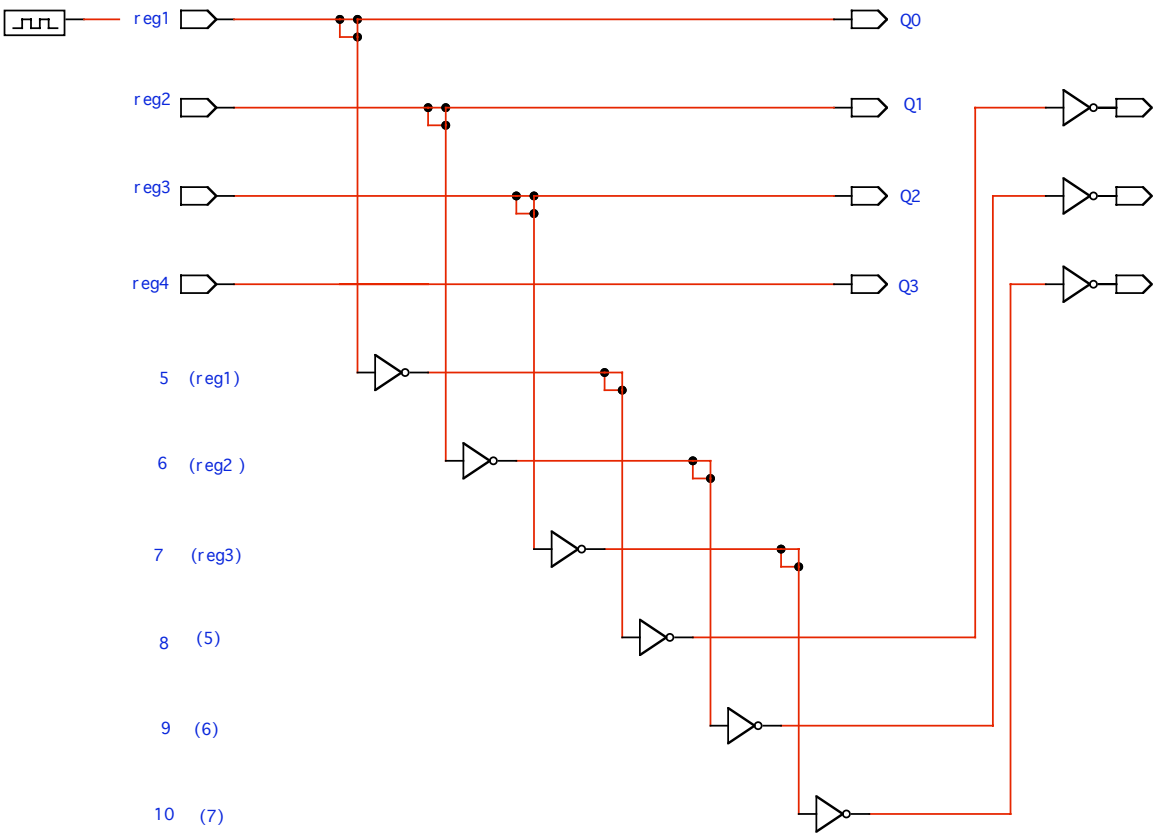
4-bit SHIFT-REGISTER parallel-in, serial-out with load



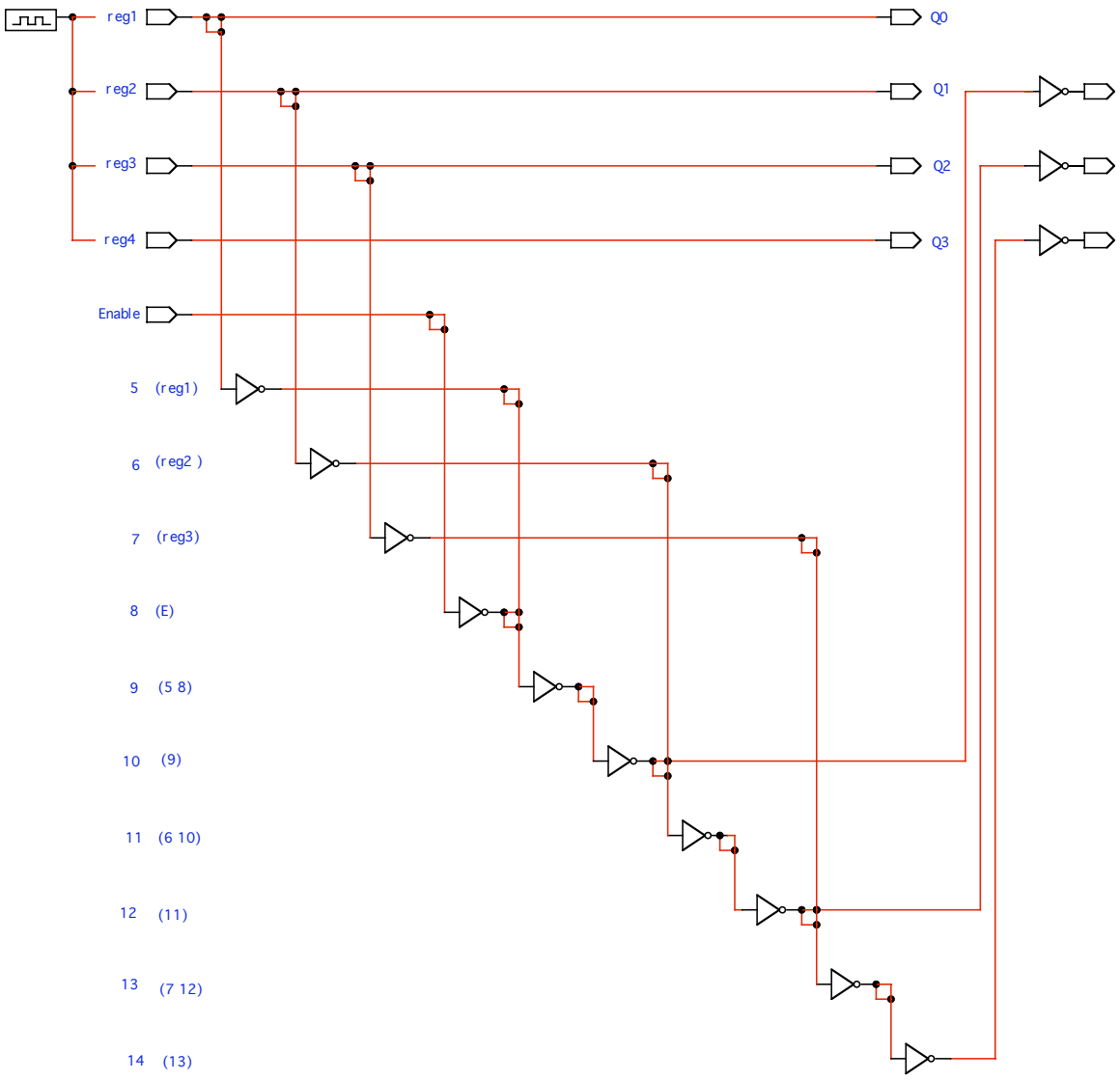
4-bit SHIFT-REGISTER bidirectional



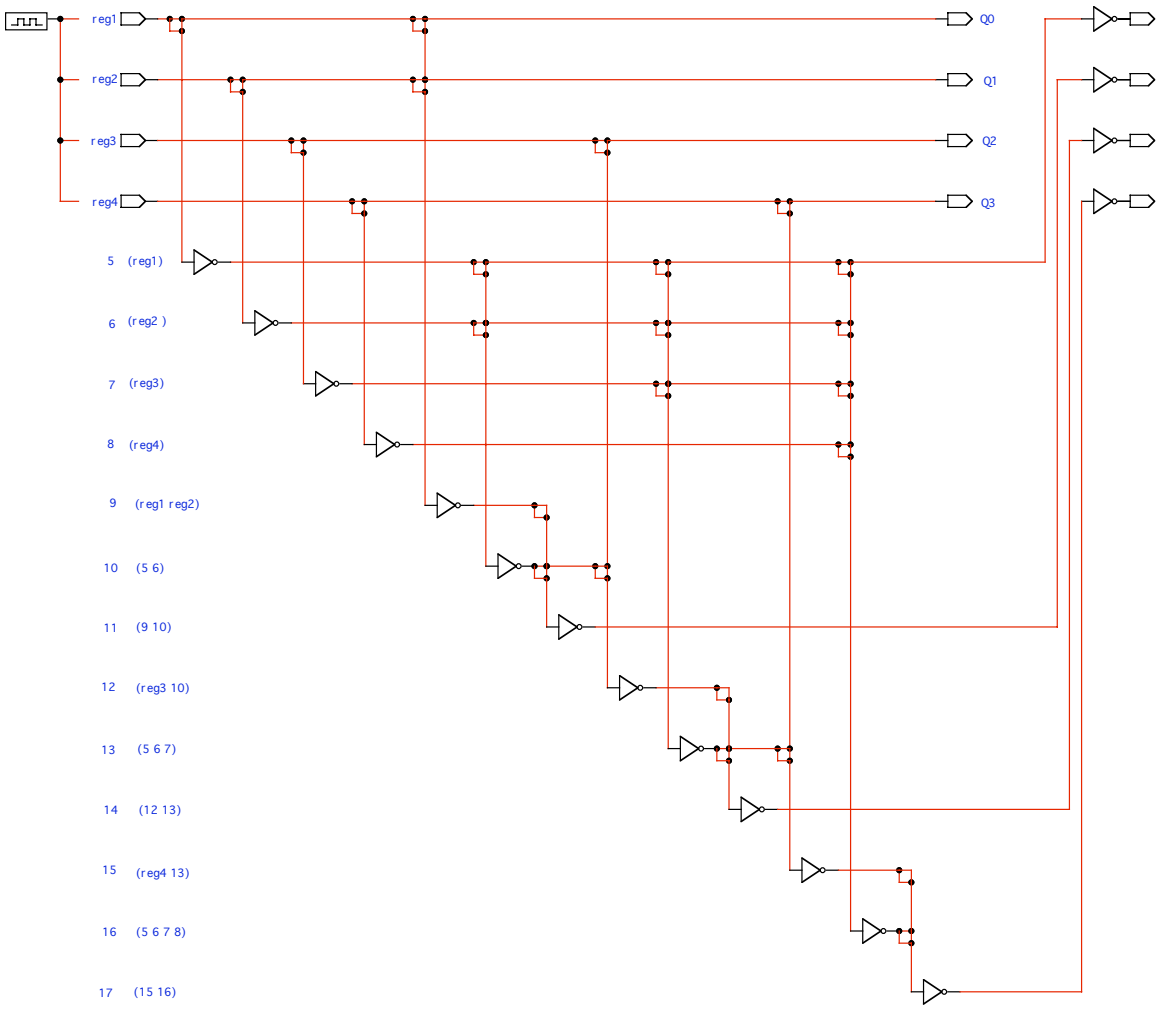
4-bit RIPPLE COUNTER



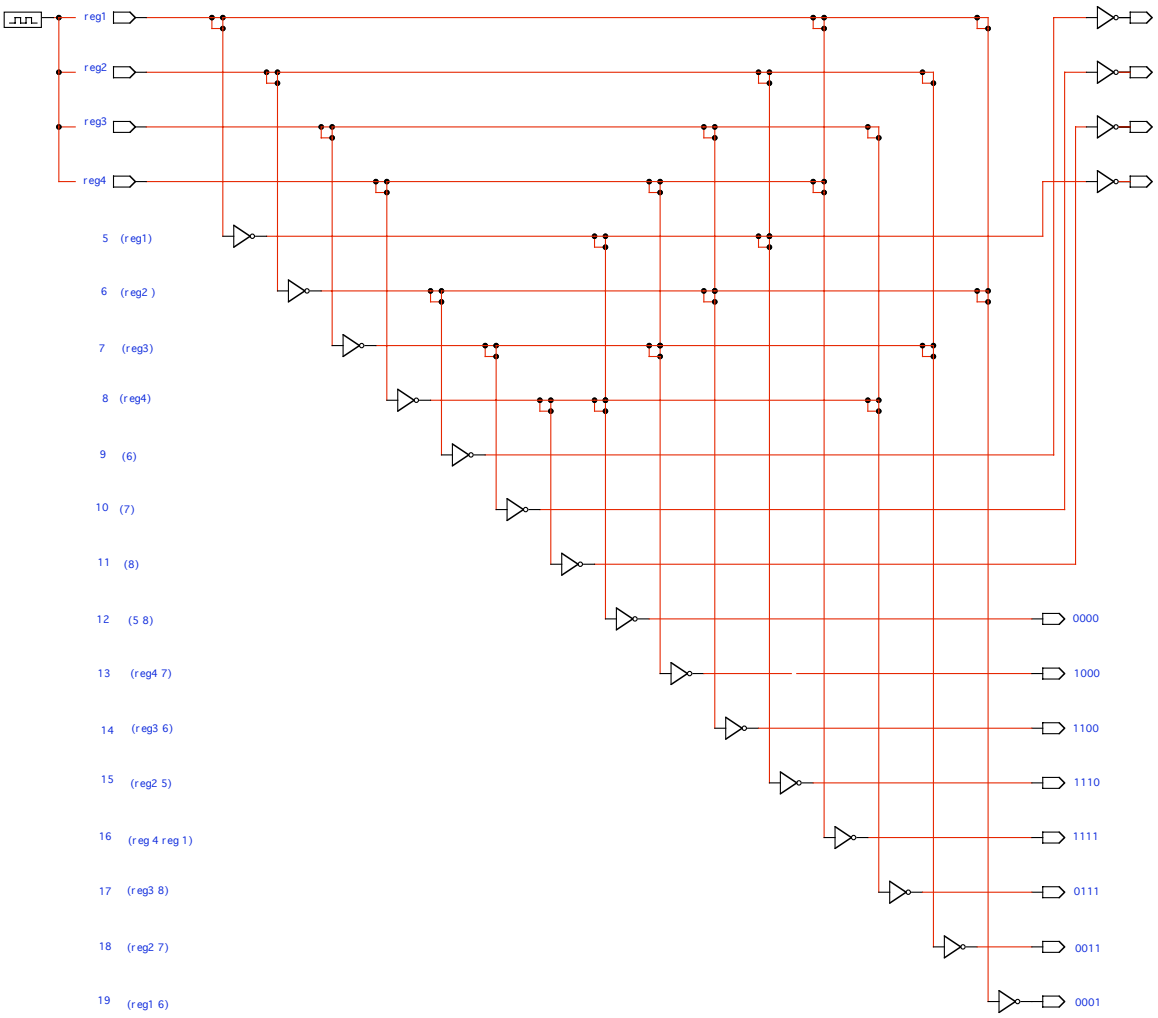
4-bit SYNCHRONOUS COUNTER with serial enable



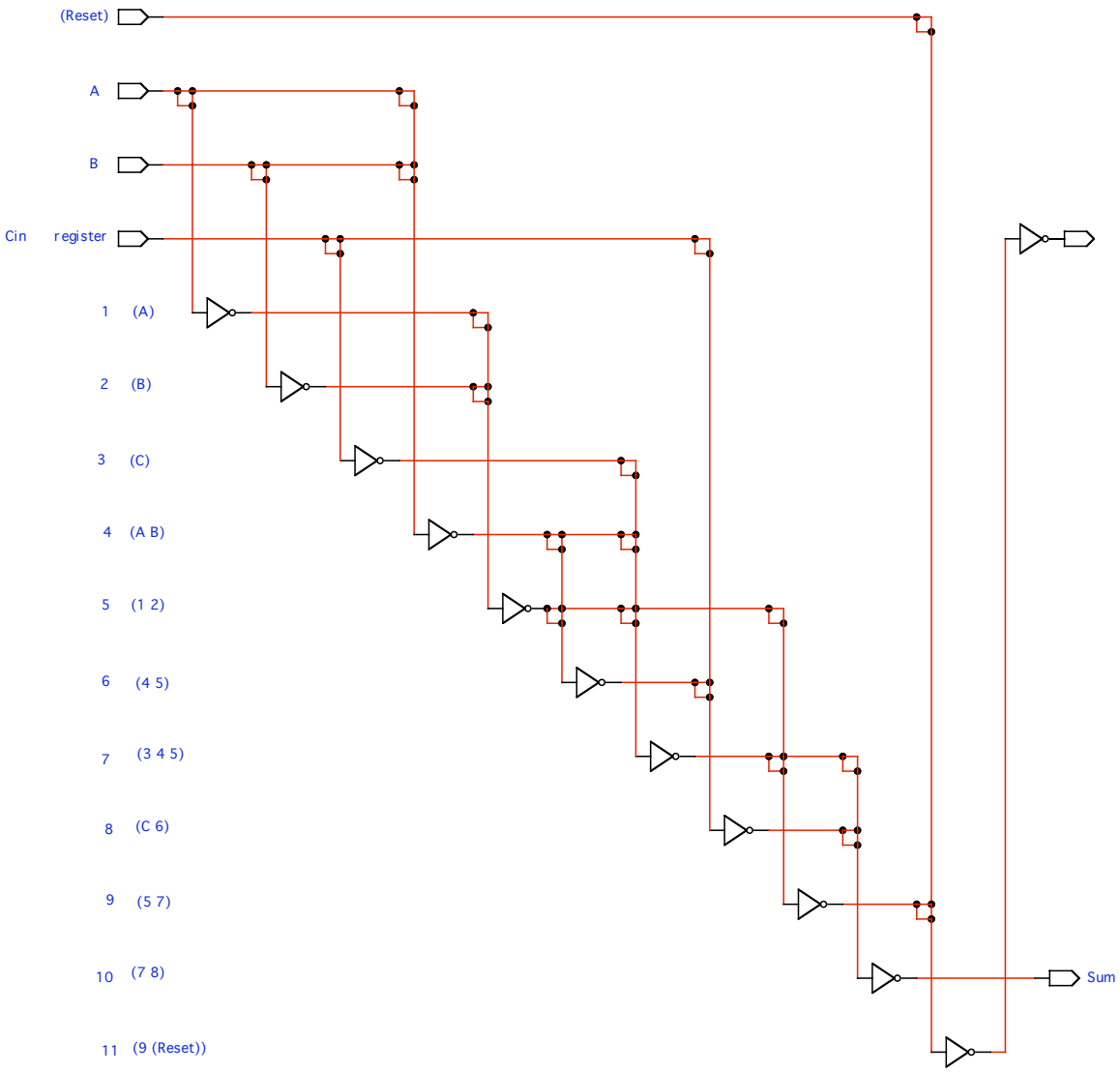
4-bit SYNCHRONOUS COUNTER



4-bit JOHNSON COUNTER with DECODER

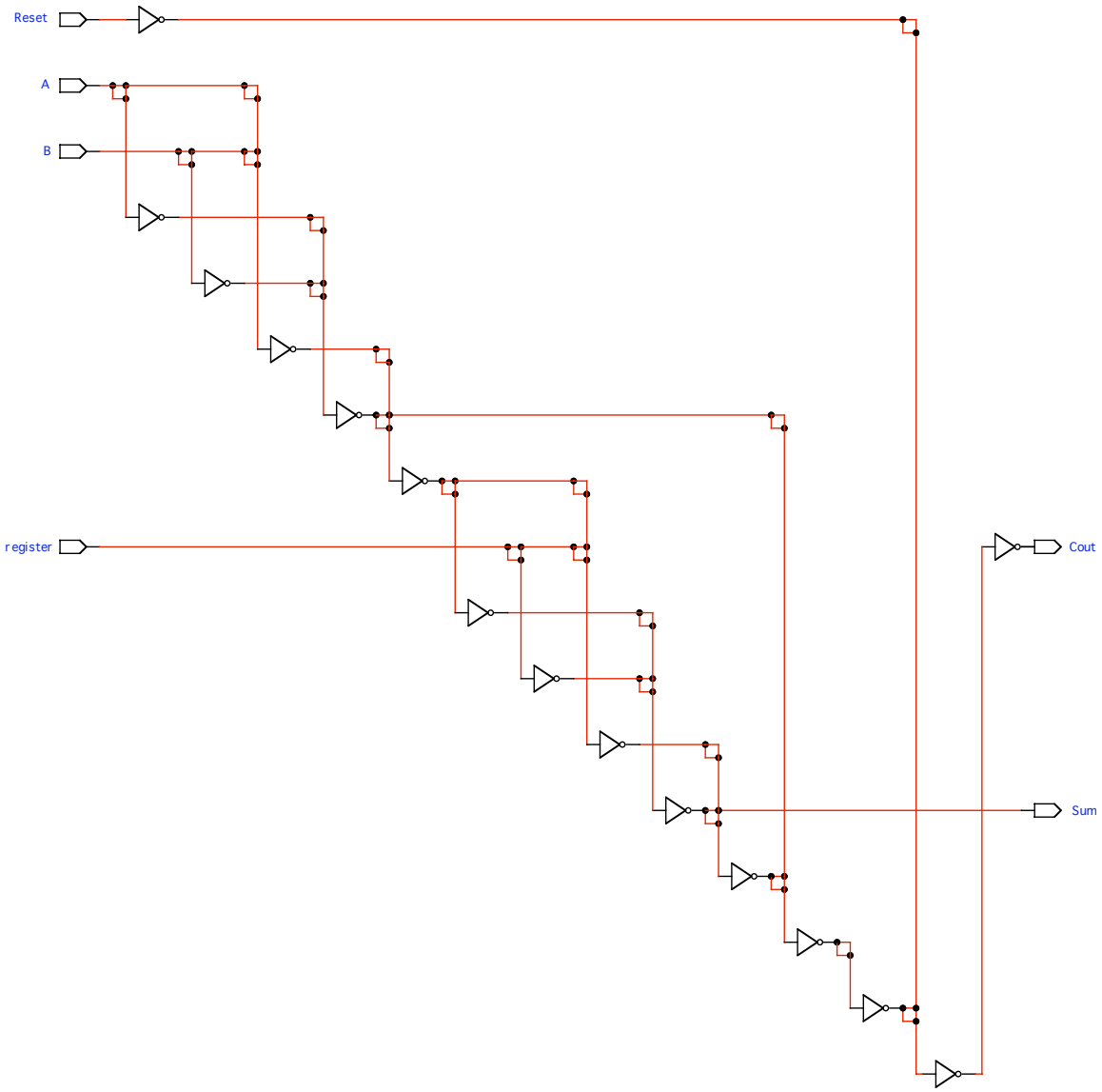


FULL-ADDER serial



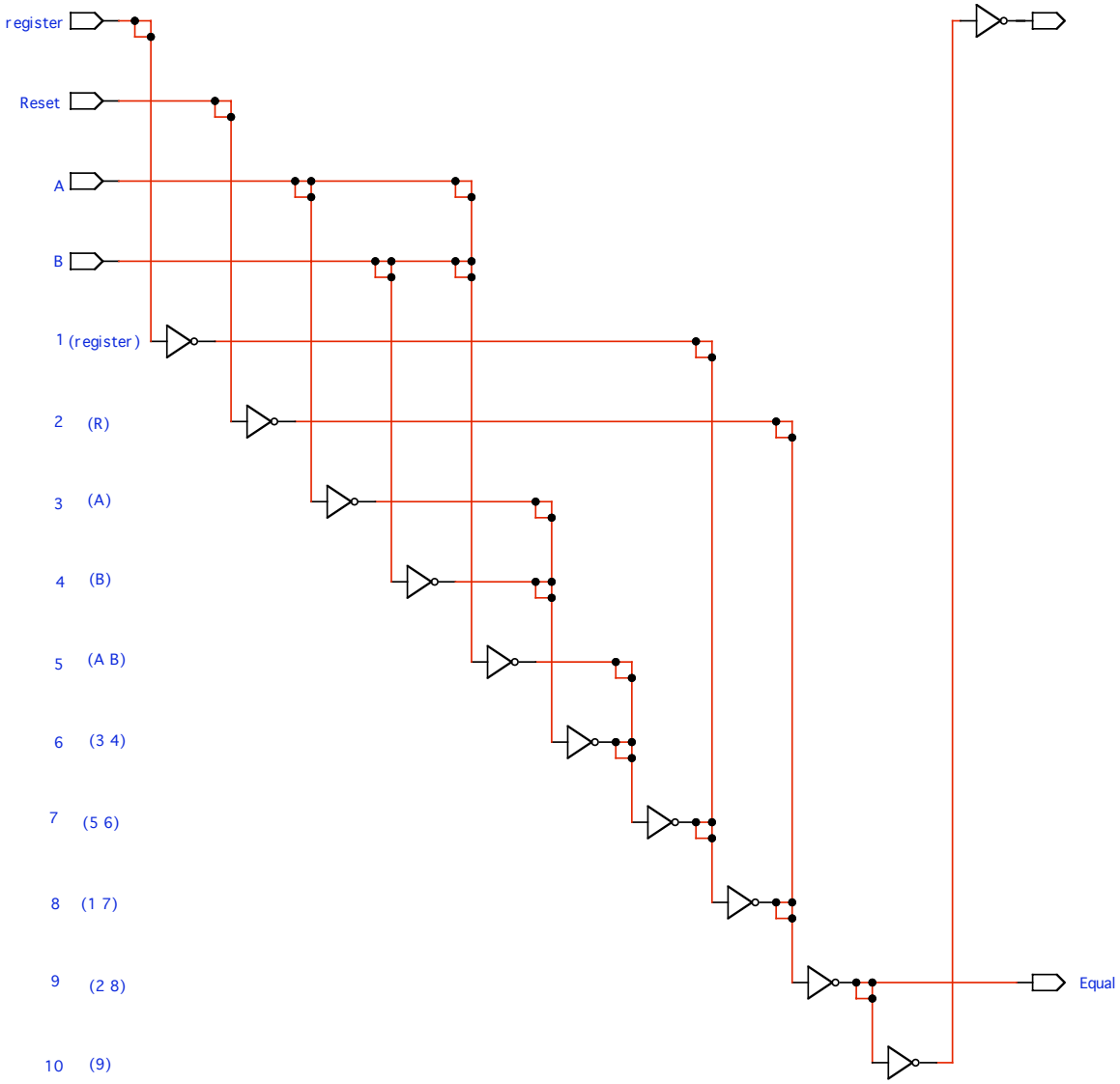
A full-adder unit for sequential composition.

FULL-ADDER serial and composed



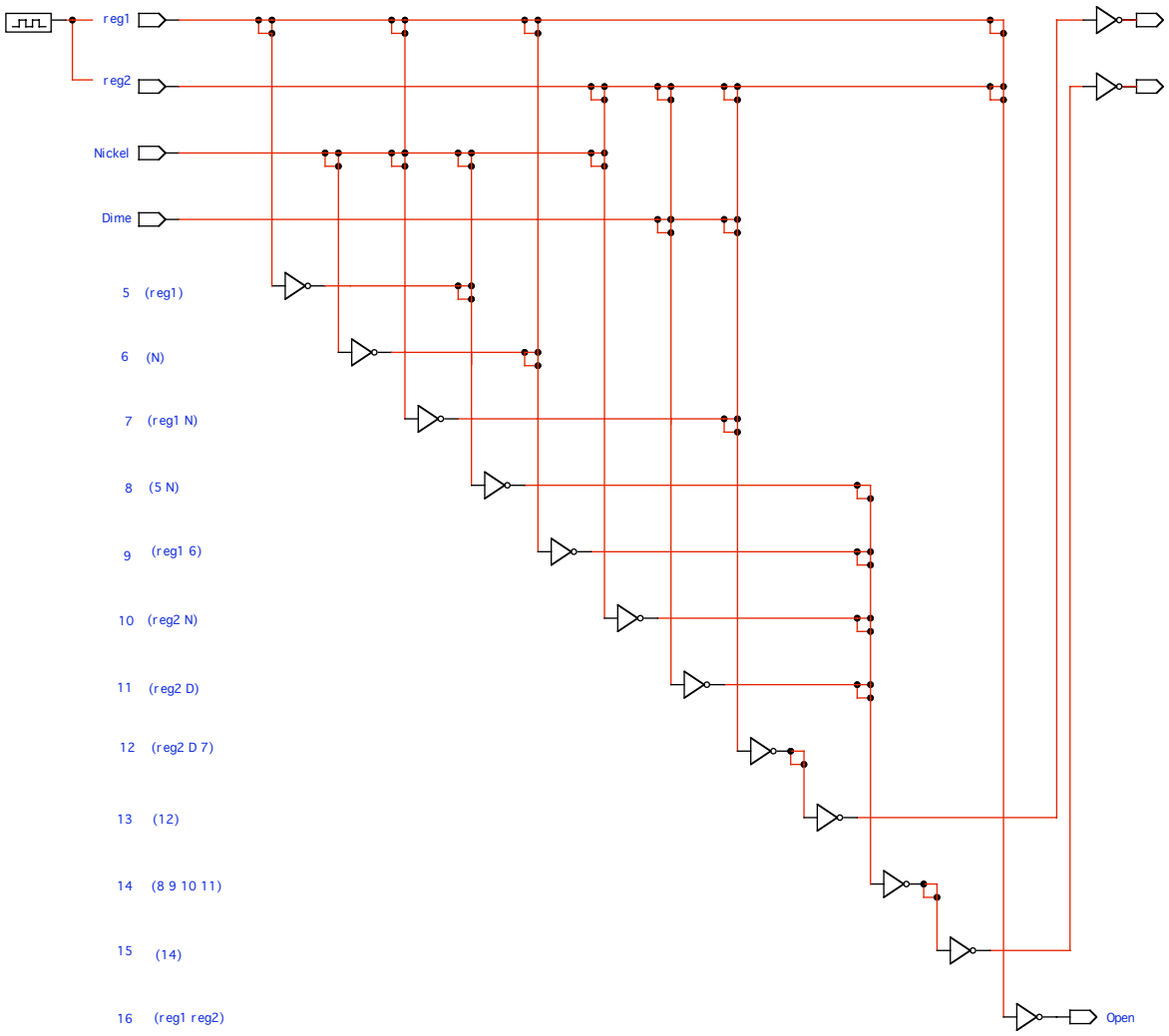
A full-adder for serial composition, itself composed of two half-adders.

2-bit COMPARATOR, serial

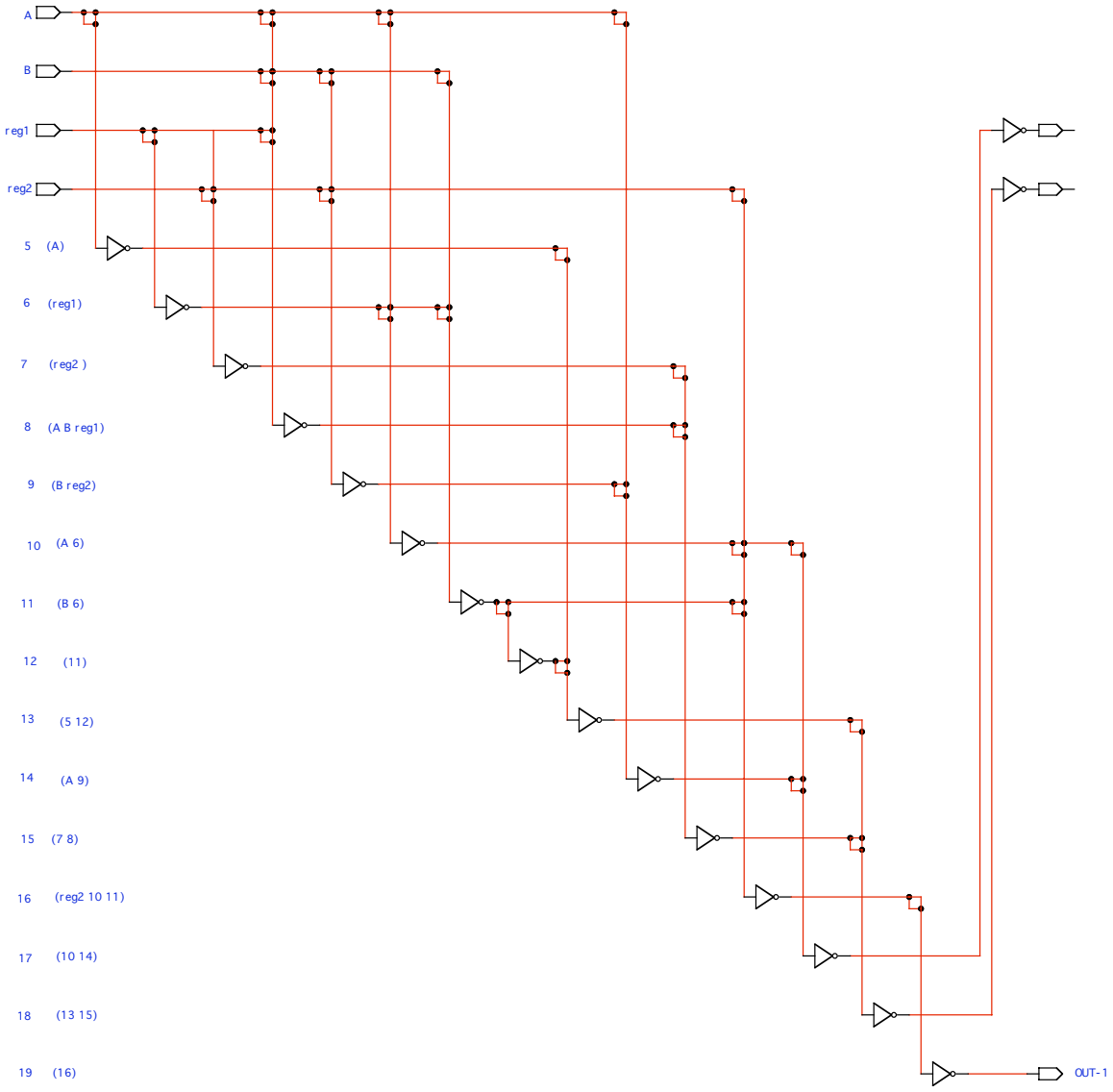


A 2-bit comparator for serial composition.

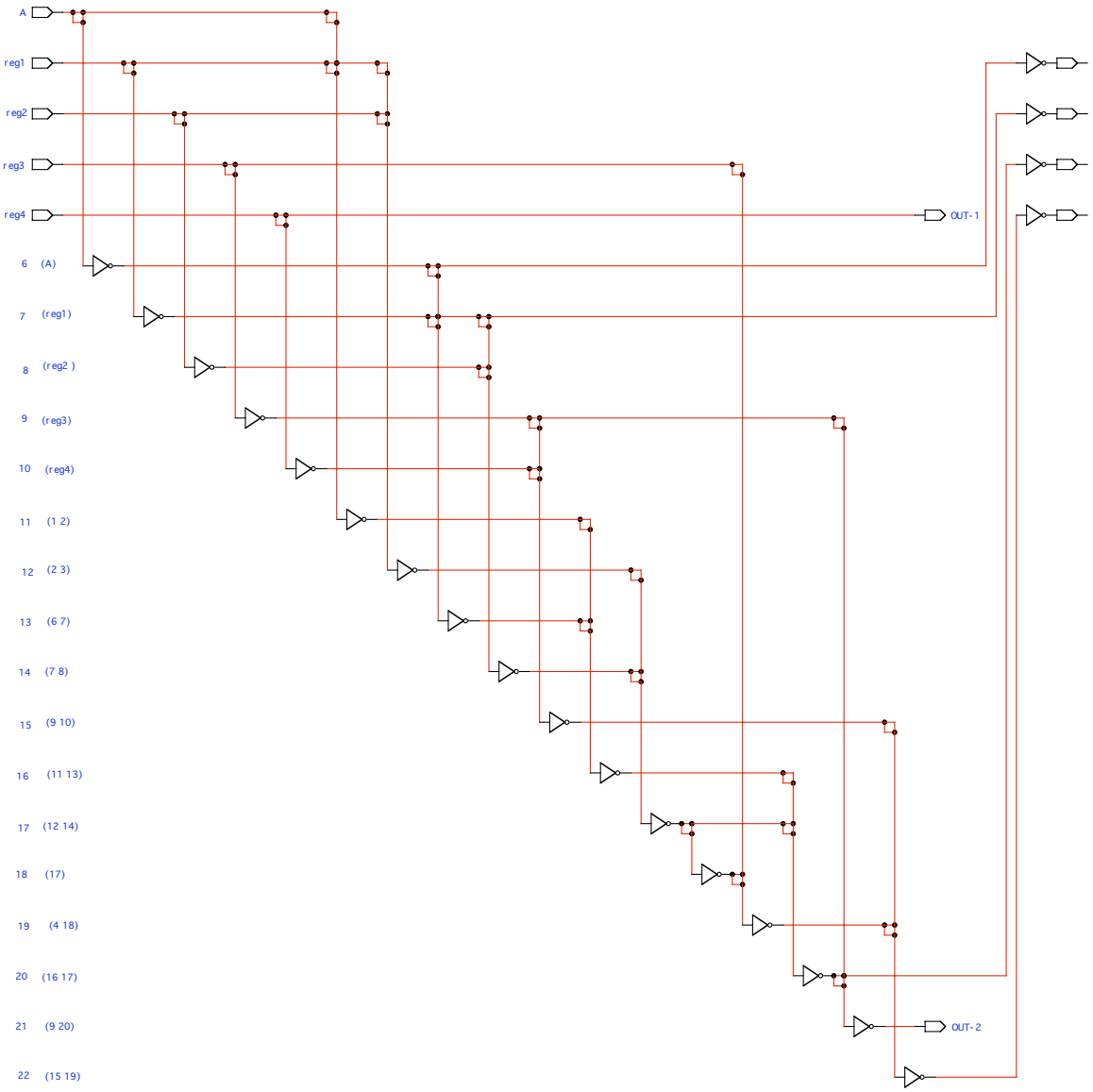
15cent vending machine FSM



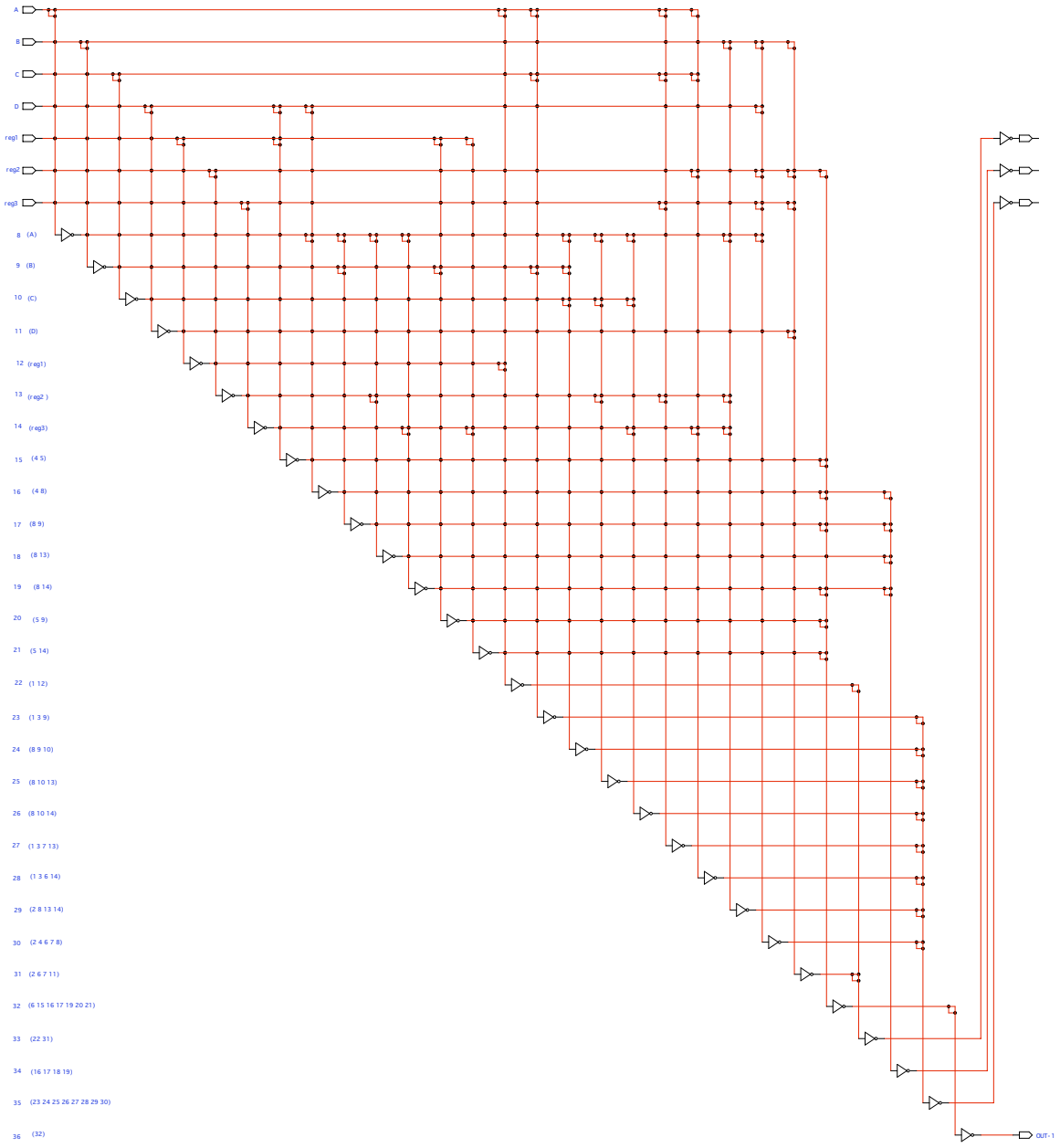
LION


A
B
reg1
reg2
5 (A)
6 (reg1)
7 (reg2)
8 (A B reg1)
9 (B reg2)
10 (A 6)
11 (B 6)
12 (11)
13 (5 12)
14 (A 9)
15 (7 8)
16 (reg2 10 11)
17 (10 14)
18 (13 15)
19 (16)
OUT-1

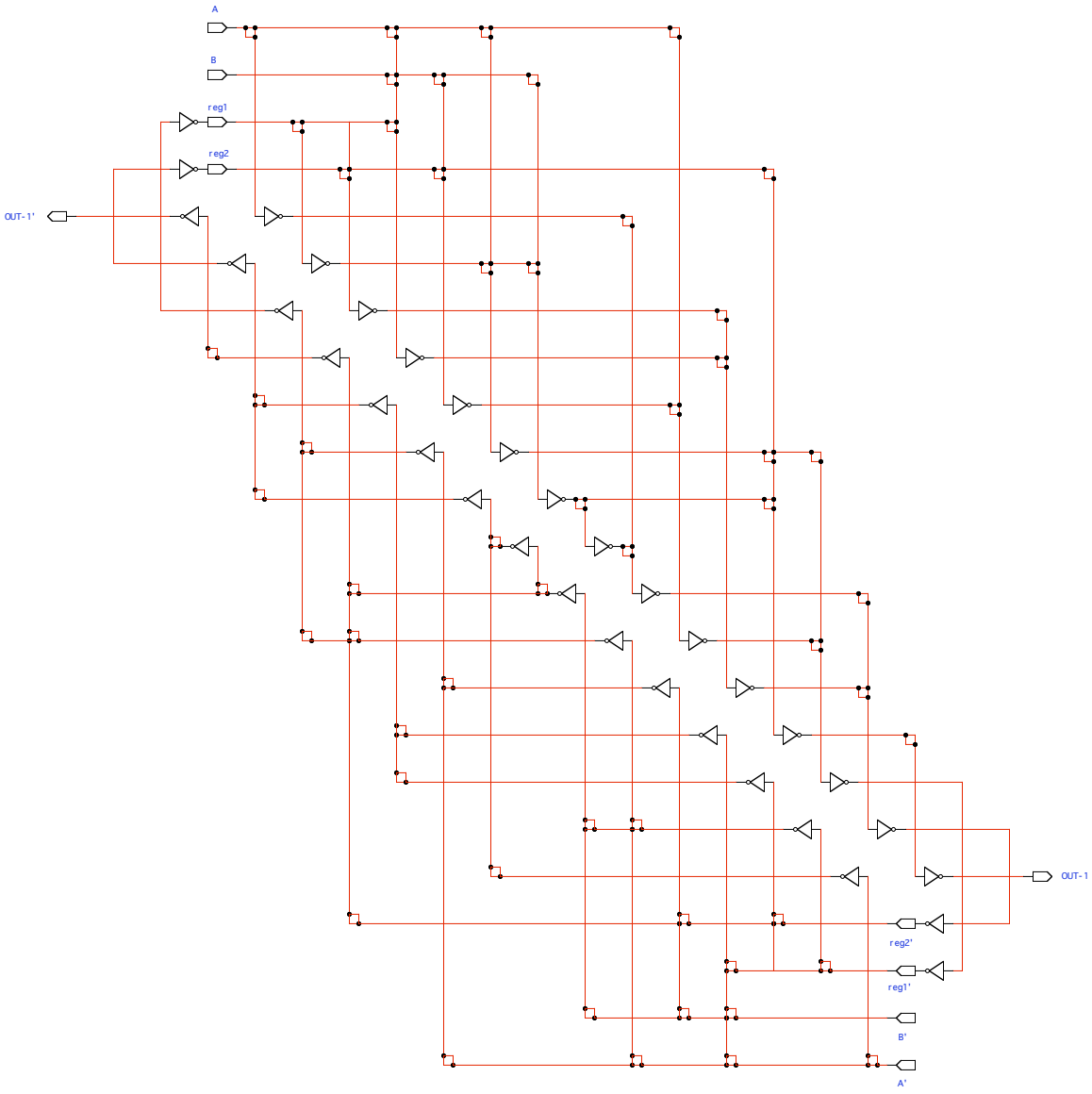
DAIO


A
reg1
reg2
reg3
reg4
6 (A)
7 (reg1)
8 (reg2)
9 (reg3)
10 (reg4)
11 (1 2)
12 (2 3)
13 (6 7)
14 (7 8)
15 (9 10)
16 (11 13)
17 (12 14)
18 (17)
19 (4 18)
20 (16 17)
21 (9 20)
22 (15 19)
OUT-1
OUT-2

S27


A
B
C
D
reg1
reg2
reg3
8 (A)
9 (B)
10 (C)
11 (D)
12 (reg1)
13 (reg2)
14 (reg3)
15 (4 5)
16 (4 8)
17 (8 9)
18 (8 13)
19 (8 14)
20 (5 9)
21 (5 14)
22 (1 12)
23 (1 3 9)
24 (8 9 10)
25 (8 10 13)
26 (8 10 14)
27 (1 3 7 13)
28 (1 3 6 14)
29 (2 8 13 14)
30 (2 4 6 7 8)
31 (2 6 7 11)
32 (6 15 16 17 19 20 21)
33 (22 31)
34 (16 17 18 19)
35 (23 24 25 26 27 28 29 30)
36 (32)
OUT-1

LION *SQUARE* (2 LION circuits)



APPENDIX:

Diagonalized CM85A Pun, used as MUX

```
((cm85a canon-renumber pun22 pun22 44-sort
  (stats cm85a lits 70 conn 73 dnode 36 gate 36) 0.29 (2001 9 5 22 4 50))
 ((main)
  ((a unk)(b unk)(c unk)(d unk)(e unk)(f unk)(g unk)(h unk)(i unk)(j unk)(k unk))
  ((oa ~35) (ob ~36) (oc ~34))

  ((~1     (a)   )
   (~2     (b)   )
   (~3     (c)   )
   (~4     (d)   )
   (~5     (e)   )
   (~6     (f)   )
   (~7     (g)   )
   (~8     (h)   )
   (~9     (j)   )
   (~10    (a ~2)   )
   (~11    (b ~1)   )
   (~12    (c ~4)   )
   (~13    (d ~3)   )
   (~14    (e ~6)   )
   (~15    (f ~5)   )
   (~16    (g ~8)   )
   (~17    (h ~7)   )
   (~18    (a ~2 ~13)   )
   (~19    (b ~1 ~12)   )
   (~20    (~12 ~18)   )
   (~21    (~13 ~19)   )
   (~22    (~14 ~21)   )
   (~23    (~15 ~22)   )
   (~24    (~15 ~20)   )
   (~25    (~14 ~24)   )
   (~26    (~16 ~23)   )
   (~27    (~17 ~25)   )
   (~28    (~17 ~26)   )
   (~29    (~16 ~27)   )
   (~30    (~9 ~28)   )
   (~31    (~9 ~29)   )
   (~32    (k ~30)   )
   (~33    (i ~31)   )
   (~34    (~32)   )
   (~35    (~33)   )
   (~36    (~9 ~10 ~11 ~12 ~13 ~14 ~15 ~16 ~17)   )  )) )
```