BOUNDARY LOGIC: The Design of Computation

Laws of Form 50th Anniversary Conference
August 9, 2019

William Bricken

william@iconicmath.com

Copyright © 2019 William M. Bricken. All rights reserved.

Style

– 15 years of exploration within several companies,
  focussing on work with Dick Shoup

– trying to find the most challenging applications for LoF logic
  optimization of 100,000 logic gate benchmark and industrial circuit designs

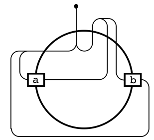
– applications to computational software and silicon hardware only;
  no philosophy, no pure mathematics, no infinite excursions

– pure boundary logic and technique only, no hybrid systems

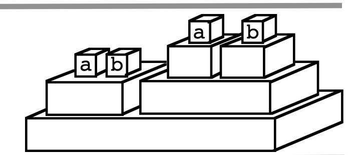
The presentation is a broad overview
that shows technical applications of LoF from
software engineering
logic optimization
semiconductor design
software/hardware co-design.

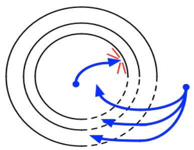
This presentation can be downloaded at iconicmath.com

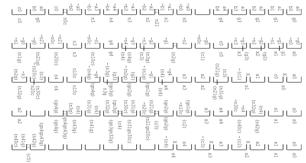
Copyright © 2019 William M. Bricken. All rights reserved.

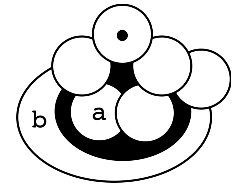


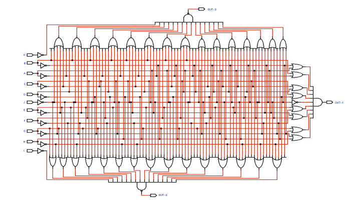
Contents

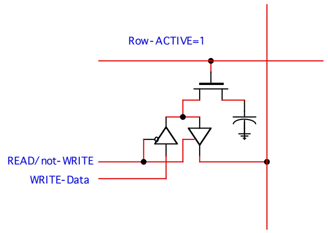












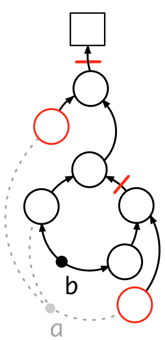
Grounding

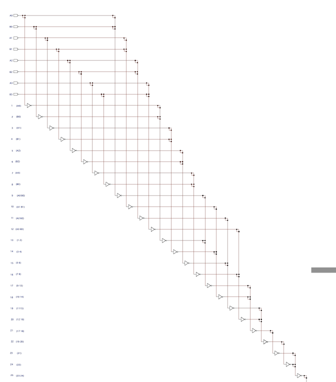
(A B C D) ==> (((A B)) ((C D)))

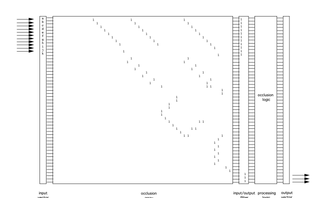
```
(A ( )) =
 ((A))   = A
A {A B} = A {B}
```

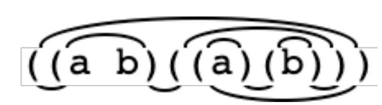
```
(((a)((a) b))) b
  (a)((a) b)   b
  (a)(      )  b
     (      )
```

Applications



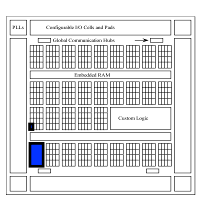


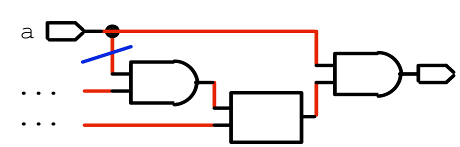


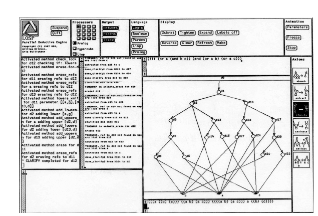


(B) = (A ( )) (B (A ( ) (A ( ))))

```
((!V1  (@ =V2)                       )
 (V3   ((V4)(q))                     )
 (V5   (((V3)(!V1)) (V5s (V3 !V1)))  )
 (=V6  (((V3  V5s)(( V3)(V5s))))     )
 (=V2s (((=V6 !V1)((=V6)(!V1))))     ))
```







α, (α |= β) |= β --> (((α) ((α) β))) β

Copyright © 2019 William M. Bricken. All rights reserved.

Applications of LoF

Advanced Decisions Systems (1984-1988)

- intelligent software editor and behavioral query language
- parallel deduction engines
- artificial intelligence inference and contradiction maintenance

logic engines

Interval Research Corporation (1993-2000)

- propositional and predicate logic deduction engines
- combinational circuit synthesis and optimization
- circuit design generation
- form abstraction
- hardware/software design integration
- sequential circuit synthesis and optimization

hardware/software
co-design

Boundary Institute, Unary Computers, BTC (2001-2006)

- circuit design generation, mapping and routing
- abstraction, partitioning and layout optimization
- novel reconfigurable hardware architectures
- iconic logic optimizing compiler

design automation
and
hardware integration

Copyright © 2019 William M. Bricken. All rights reserved.

Dedicated Funding

Interval Research Corporation

- Paul Allen's silicon valley research company, 1993-2000
- 50-80 researchers pursuing their own agendas
- $80 million/year budget
- all research and development held in trade secrecy

The Natural Computing Project *led by Dick Shoup*

If you could redesign silicon computation,
without concern for backward compatibility, what would you build?

- Foundational math and theory (15 person-years)
- Language and interface (15 person-years)
- Architecture and tools (15 person-years)
- Applications and commercialization (5 person-years)

LoF was *the central organizing principle* for

- design of new mathematical foundations for computation
- integration of interactive software tools
- development of visual specification languages
- construction of reconfigurable hardware architectures

Copyright © 2019 William M. Bricken. All rights reserved.

New Foundations

Goals

- mathematics that directly supports formal verification
- hierarchical algebraic design language
- unification of hardware and software design
- formal verification of benchmark and industrial semiconductor designs

Logic engines — *Bricken and James* ← *today*

- propositional and predicate calculus engines
- distinction networks
- hierarchical and functional abstraction
- multilevel combinational and sequential circuit optimization

Transition analysis — *Shoup and Furtek*

- computation as signal propagation and change (vs objects and states)
- sequential and behavioral verification

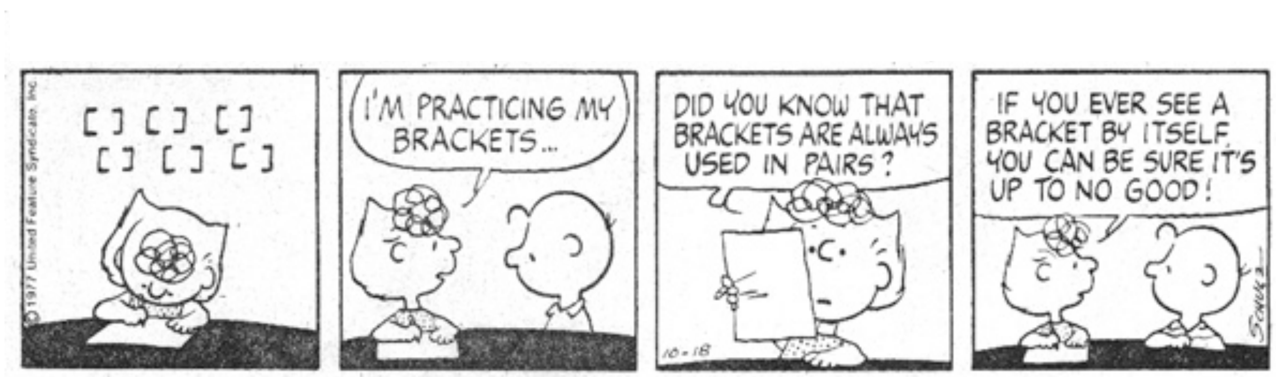
Link theory — *Etter and Shoup*

- a general theory of formal structure
- connectivity defines information and independence

Copyright © 2019 William M. Bricken. All rights reserved.

FUNDAMENTAL CONCEPTS

Peanuts



Copyright © 2019 William M. Bricken. All rights reserved.

Take Nothing Seriously

Empty containers permit the semantic use of syntactic non-existence.

( ) contains *nothing* on the inside.

Void has *no properties* and supports *no relations*.

Void-equivalence (A ( )) =

– forms and patterns can be equated to *void*

Void-substitution (B (A ( ))) = (B)

– substitution of *void* for a void-equivalent form returns nothing to non-existence

Void-based pattern transformation (B) = (A ( )) (B (A ( ) (A ( ))))

– void-equivalent forms can be deleted at will
– void-equivalent forms can be constructed anywhere throughout a form

~~~ The Principle of Void-Equivalence ~~~

Void-equivalent forms are syntactically irrelevant and semantically inert.

but they can still be used to catalyze change

Copyright © 2019 William M. Bricken. All rights reserved.
~~~

Void-based Reasoning

Garfield

Structure

- all forms are containers
- boundaries distinguish their contents
- contents are inherently independent
- logic boundaries are semipermeable
- many multidimensional options for representation

Computational technique

- pattern-directed structural transformation
- deletion of irrelevant structure rather than collection of facts
- depth insensitive operations across boundaries
- non-intrusive, query-based identification of valid deletions
- proof is reduction of form to *void*

*single concept system**
`contains` *serves as*
a ground
an object
a unary operator
a binary relation
a data structure
a transformation pattern

replication provides diversity

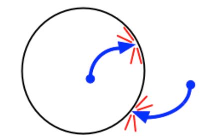
replication is the source of complexity

* W. Bricken (2017) Distinction is Sufficient, *Cybernetics and Human Knowing*, **24**(3-4), p.29-74.

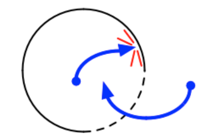
Copyright © 2019 William M. Bricken. All rights reserved.

Boundary Permeability

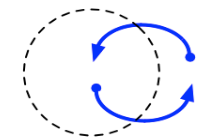
Impermeable boundaries do not permit forms to cross.
– a model for numerics



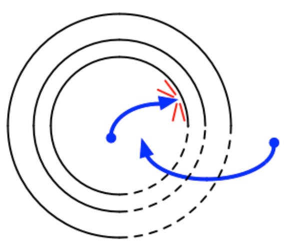
Semipermeable boundaries permit crossing in one direction only.
– a model for logic



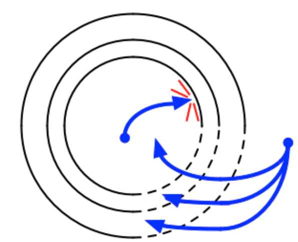
Fully permeable boundaries do not distinguish their contents.
– a model for imaginary forms



Logic boundaries are *transparent to their context*.
Forms on the outside are arbitrarily present in every interior space.



Pervasion $A\ \{A\ B\} = A\ \{B\}$



curly braces denote any intervening structure

By convention, the semantic viewpoint is on the outside.*
(*We* are outside of the space of representation.)
Crossing without permission changes intent.

* W. Bricken (1994) Inclusive Symbolic Environments, in K. Duncan & K. Krueger (eds) *Proceedings of the 13th World Computer Congress*, v3, Elsevier Science, p.163-170.

Copyright © 2019 William M. Bricken. All rights reserved.

Parallel and Sequential Partitioning

Forms in the same container are independent.
They do not interact and can be processed in parallel.

```
(A
   B
      C
         D
            E)
```

parallel processes

Parallel partitions are structured by containment width.

Forms nested within other forms are structurally dependent
and require sequential processing.

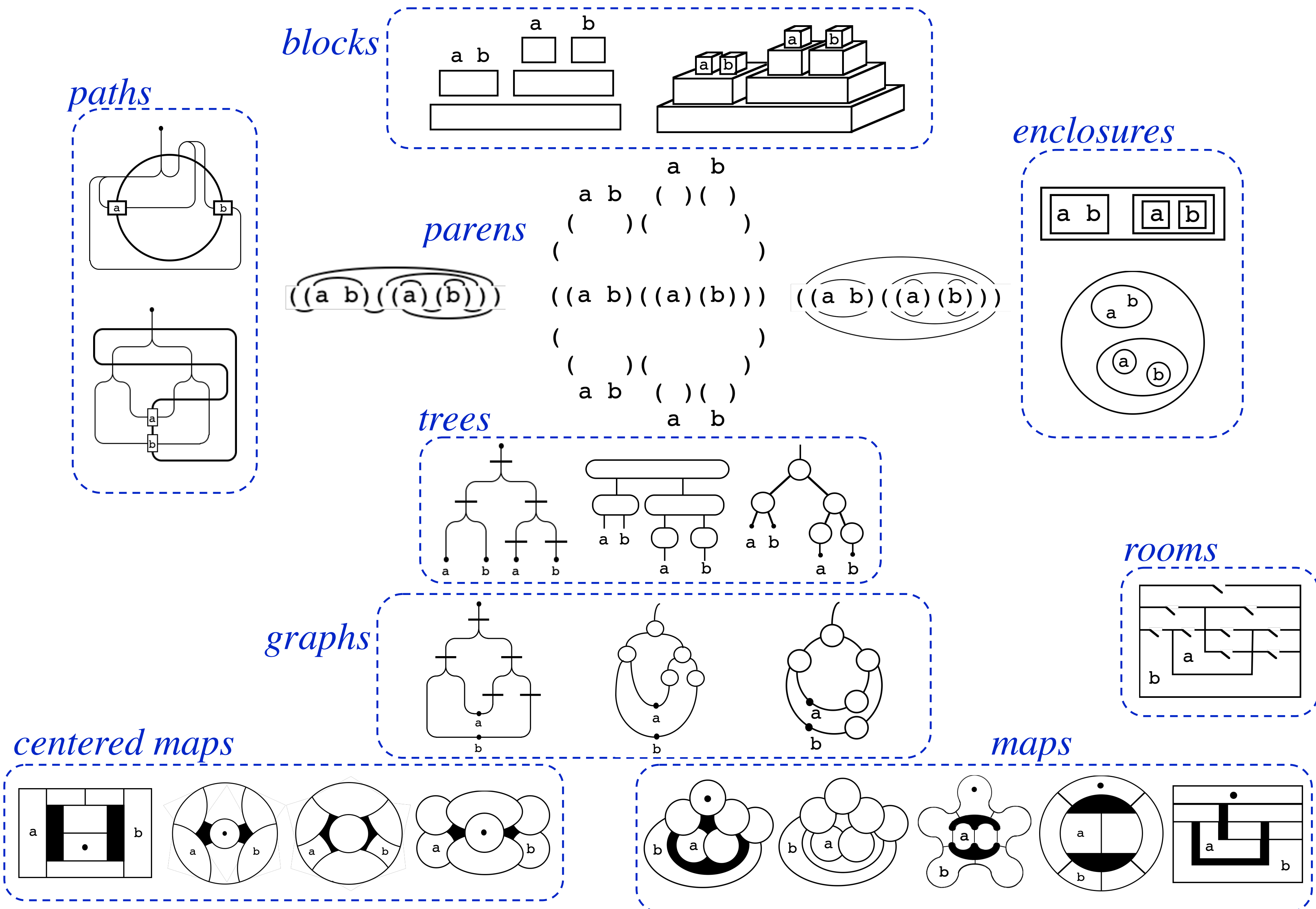
```
(A (B (C (D (E)))))
```

sequential processes

Sequential partitions are structured by containment depth.

Copyright © 2019 William M. Bricken. All rights reserved.

Syntactic Variety



W. Bricken (2006) Syntactic Variety in Boundary Logic, in D. Barker-Plummer *et al* (eds) *Diagrams 2006* LNAI 4045, Springer-Verlag, p.73-87.

Copyright © 2019 William M. Bricken. All rights reserved.

Boundary Logic Algorithms and Research

1977–2007: Hawaii, Palo Alto, Sausalito, Seattle

- algebraic theory development
- LISP implementation
- rigorous applications
- pragmatic applications
- application generalization
- visual and experiential languages

Copyright © 2019 William M. Bricken. All rights reserved.

Algebraic Pattern-Equations

Axioms

			Spencer-Brown:
`(A ( )) =`	*halting condition*	**Occlusion**	*void occlusion*
`((A)) = A`	*boundary deletion*	**Involution**	*reflexion*
`A {A B} = A {B}`	*form deletion*	**Pervasion**	*extended generation*

Curly braces refer to *any* deeper intervening structure.
Each pattern proceeds from left to right by deletion of structure.
There is no analogy in conventional mathematical technique.

Useful Theorems

to manage structural tangles

`(A) {B (A N)} = (A) {B}`	**Subsumption**	
`((A B)(A C)) = A ((B)(C))`	**Distribution**	*transposition*
`((A (B))(C (A))) = (A B) ((A)(C))`	**Pivot**	

Copyright © 2019 William M. Bricken. All rights reserved.

Boolean and Boundary Logic

Dual Boolean	Boolean	Boundary
TRUE	FALSE	
FALSE	TRUE	( )
NOT a	NOT a	(a)
a AND b	a OR b	a b
NOT (a AND b)	NOT (a OR b)	(a b)
NOT (IF b THEN a)	IF a THEN b	(a) b
a OR b	a AND b	((a)(b))
a NOT EQUALS b	a EQUALS b	(a b)((a)(b))

The boundary logic "constant": ( )

The boundary logic "function": (a)

The boundary logic "relation": $(a)_b$

object and operator are subsumed by pattern

Copyright © 2019 William M. Bricken. All rights reserved.

One-to-Many Mapping

One boundary form represents *many* different conventional logic expressions.

A one-to-many mapping is necessary for one system to be *simpler*.

The particular logical interpretation of a given boundary form is a *free choice*.

( )	((a)(b))
1	a AND b
NOT 0	b AND a
1 OR 0	NOT (NOT a OR NOT b)
0 OR 1 OR 0	NOT a NOR NOT b
0 NOR 0	NOT (a NAND b)
(NOT 0) OR 0	(a AND b) OR 0
NOT (0 OR 0)	NOT (a NAND (0 OR b)) OR 0
NOT (0 OR 0) OR (0 OR 0)	NOT (b NOR 0) OR NOT a OR 0
...	...

void

0 = 0 OR 0 = 0 OR 0 OR 0 = ...

Copyright © 2019 William M. Bricken. All rights reserved.

Table of Non-Correspondence

LoF is not Boolean.

	BOOLEAN	BOUNDARY	DIFFERENCE
symbols	strings	icons	linear vs spatial
constants	{0,1}	{ ( ) }	two vs one
duality	objects	spaces	existence
mapping	functional	structural	values vs patterns
unary operator	NOT	none	existence
binary operator	AND, OR	contains	two vs one
arity	specific	variary	countable vs not
commutativity	linear	none	existence
associativity	binary	nesting	non-associative
rearrangement	distribution	pervasion	regroup vs construct
computation	rearrange	delete	void-equivalence
random valuation	50% TRUE	~64.7% ( )	symmetric vs asymmetric

Copyright © 2019 William M. Bricken. All rights reserved.

Proof of *Modus Ponens*

$\alpha, (\alpha \models \beta) \models \beta$ --> (((α) ((α) β))) β

values
expressions
collections

(a AND (a IMPLIES b)) IMPLIES b *modus ponens*

Transcribe

(a AND (a IMPLIES b)) IMPLIES b	
(a AND (a) b) IMPLIES b	a IMPLIES b --> (a) b
((a) ((a) b)) IMPLIES b	a AND X --> ((a)(X))
(((a) ((a) b))) b	X IMPLIES b --> (X) b

Reduce

(((a)((a) b))) b	transcription	
(a)((a) b) b	involution	((A)) ==> A
(a)( ) b	pervasion	A (A B) ==> A (B)
( )	dominion	A ( ) ==> ( )

Interpret

TRUE ( ) --> TRUE

Copyright © 2019 William M. Bricken. All rights reserved.

Virtual Insertion Technique

Outer forms pervade all inner spaces.
Their *hypothetical presence as queries* can trigger structural deletions.

Simple Subsumption `(a) (a b) =?= (a)` *virtual forms in red*

```
(a) ((a) a b)          insert (a)
(a) ((  ) a b)         extract a
(a)                    occlusion
```

Broad Subsumption `(a b c) (a (d (b (c e)))) =?= (a b c) (a (d (b)))`

```
(a b c) (a (d (b (c e) (a b c)))))      insert (a b c)
(a b c) (a (d (b (c e) (    c)))))      extract a b
(a b c) (a (d (b       (    c)))))      subsume (c e)
(a b c) (a (d (b               )))      discard (c)        virtual!
```

Copyright © 2019 William M. Bricken. All rights reserved.

Equivalence by Query

((a)(b)) (a (b (c))) =?= b (a c) *binate*

A = B *iff* A B = A B = *virtual extraction method*

```
((a)(b))(a (b (c)))          b (a c)          A B
((a)( ))(a (  (c)))          b (a c)          extract b
        (a (  (c)))          b (a c)          occlusion
        (a     c  )          b (a c)          involution
                             b (a c)          extract (a c)
          void                                discard b (a c)
```

```
  b (a c)           ((a)(b))(a (b (c)))        B A
  b (a c)           ((a)( ))(a (  (c)))        extract b
  b (a c)                   (a (  (c)))        occlusion
  b (a c)                   (a     c  )        involution
  b                         (a     c  )        extract (a c)
  b                                            discard (a c)
((b)                  )                        involution
((b) ((a)(b))(a (b (c))))                      insert A
((b) ((a)   )(a (b (c))))                      extract (b)
((b) ((a)   )(a        ))                      subsume (b (c))
((b) (      )(a        ))                      extract (a)
          void                                 occlusion
```

Copyright © 2019 William M. Bricken. All rights reserved.

Artificial Intelligence Programming

1981–1988: Advanced Decision Systems & Stanford University

- propositional theorem prover
- intelligent program editor (semantic debugger for Ada)
- behavioral query language
- LoF-based programming language
- software optimization
- AI inference engine
- inference with contradictions
- asynchronous parallel computation (Intel Hypercube)

Copyright © 2019 William M. Bricken. All rights reserved.

LoF Deductive Engines

Pure boundary logic data structures and algorithms.

First-order Logic

- predicate calculus with quantification
- built-in theory of equality
- skolemization, unification, demodulation
- Boolean minimization and symmetry detection
- selected domain theories

used for code optimization rather than theorem proving

Configurable Computation

- partial case analysis, partial function evaluation
- generate counter-examples if possible
- identify parallel and sequential components
- parallel propositional logic implemented on a 16-core processor

Inconsistency Maintenance

- capture, isolate and use contradiction without degradation

Copyright © 2019 William M. Bricken. All rights reserved.

Executable Code

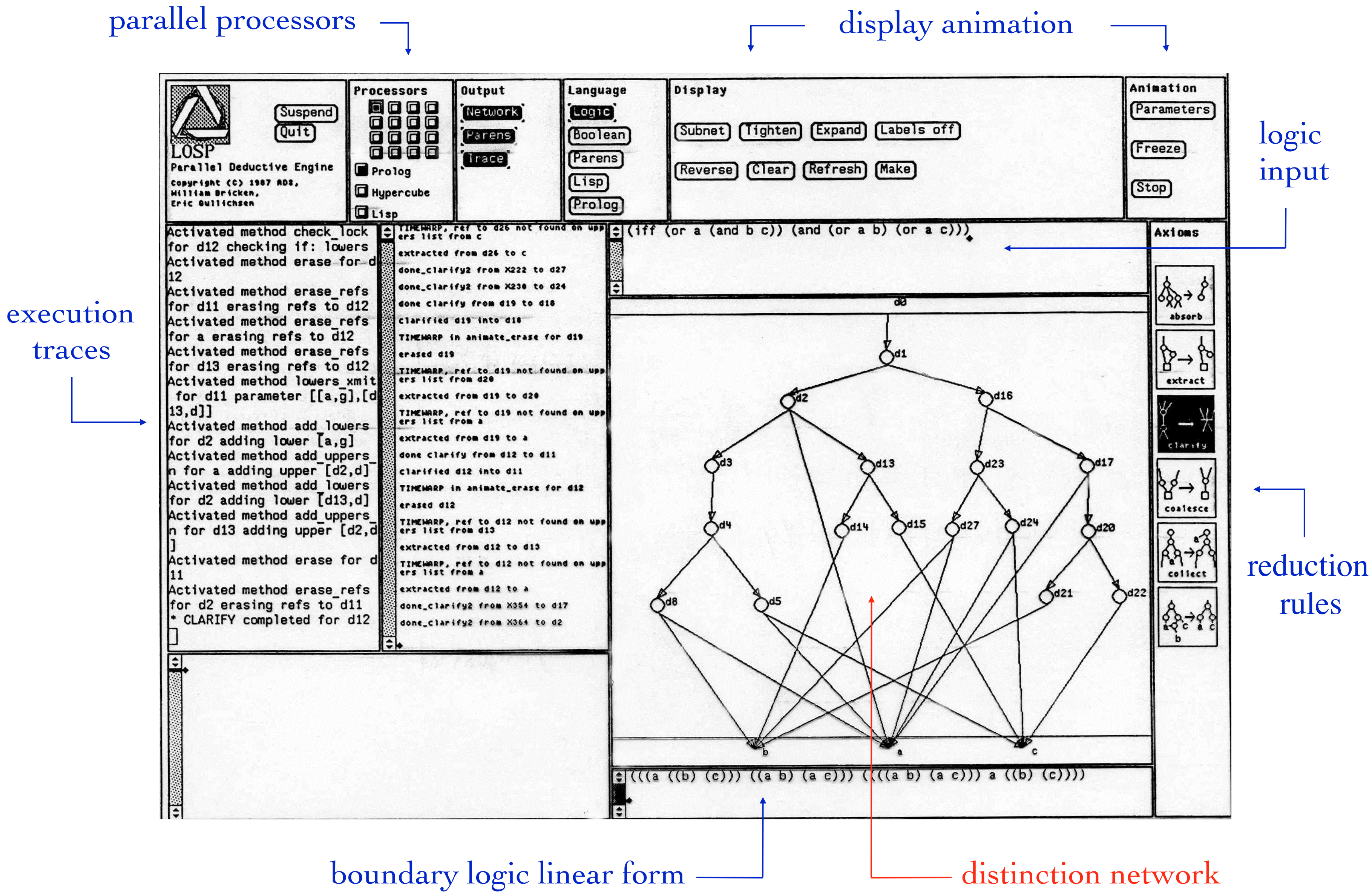
This very efficient LISP code implements *Occlusion* and *Involution* recursively to simplify and evaluate logic expressed as parens forms.

Readability is achieved by renaming common LISP functions.

```
(instructions-to apply-atomic-deletion-reduction
  (with-any (form)
    (take-these-steps
      ((if-its-an-atom form) form)
      ((if-theres-a-ground-mark-inside-the form) nothing)
      ((if-its-a-compound form)
        (simplify
          (the-result-of
            (the simplification-of-each-part-of-the) form)))
      ((if-its-an-atom (inside-of-the form)) form)
      ((if-theres-a-ground-mark-inside-the (inside-of-the form)) ground-mark)
      ((if-the-contents-are-compound form)
        (simplify
          (the-container-of
            (the-result-of
              (the simplication-of-each-part-of-the) (inside-of-the form)))))
      (otherwise (apply-atomic-deletion-reduction
                   (to-whats-in-the-double-container-of-the form))))))
```

Copyright © 2019 William M. Bricken. All rights reserved.

Asynchronous Parallel Deduction Engine (1987)



W. Bricken and E. Gullichsen (1989) An Introduction to Boundary Logic with the Losp Deductive Engine, *Future Computing Systems* 2(4), p.1-77.

Copyright © 2019 William M. Bricken. All rights reserved.

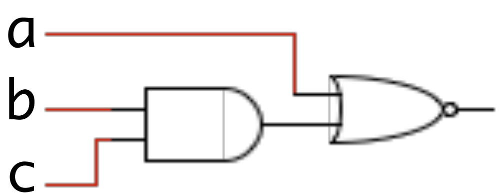
Semiconductor Optimization

1994–2000: Interval Research Corporation & Seattle University

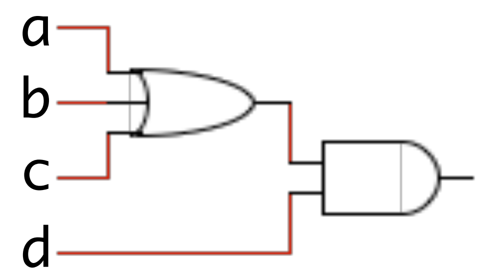
- Boolean satisfiability, Boolean minimization
- predicate calculus deductive engine
- combinational and sequential circuit optimization (area and delay)
- mapping to reconfigurable hardware

Copyright © 2019 William M. Bricken. All rights reserved.

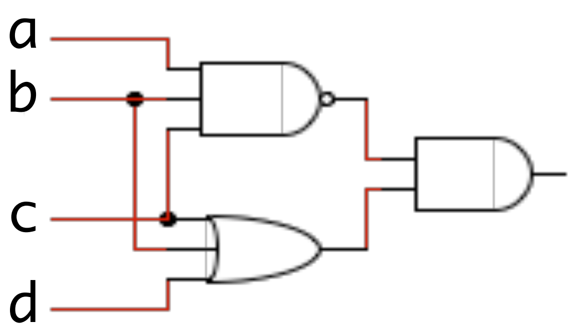
Circuit Structures in Boundary Logic



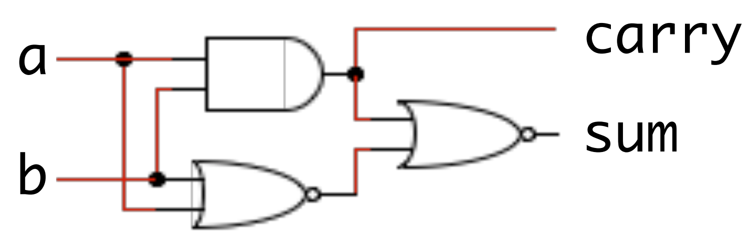
(a ((b)(c)))



((d)(a b c))



((b c d) ((a)(b)(c)))



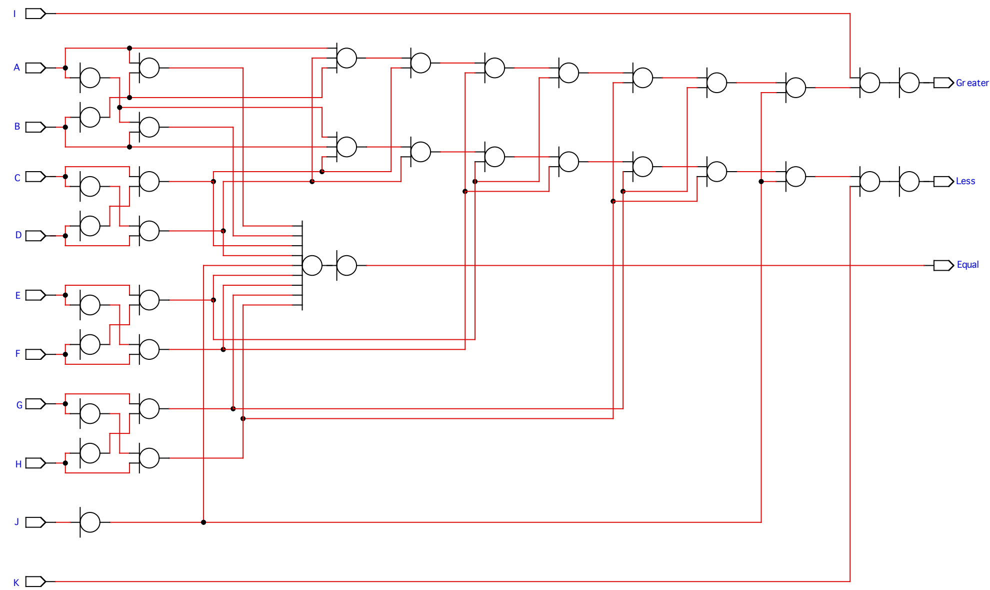
sum   = (carry (a b))
carry = ((a)(b))

Copyright © 2019 William M. Bricken. All rights reserved.

Distinction Networks

A distinction network (dnet) circuit propagates disconnects.

eleven inputs



three outputs

4-bit Magnitude Comparator with enables *fully expanded*

```
((eq 1) (gt 2) (lt 3))
 ((1  ((j)(a (b))(b (a))(c (d))(d (c))(e (f))(f (e))(g (h))(h (g)))          )
  (2  ((i ((j)((g (h))((h (g))((e (f))((f (e))((c (d))(a (b)(d (c))))))))))  )
  (3  ((k ((j)((h (g))((g (h))((f (e))((e (f))((d (c))(b (a)(c (d))))))))))  )))
```

Copyright © 2019 William M. Bricken. All rights reserved.

Structure Sharing

Multilevel circuits fanout from logic gates to share computational resources.

Fanout in represented by the number of references to a particular dnet cell.

Distinction network format

- Each row is a *cell*.
- A cell consists of a label and a boundary logic form.
- Letters are input labels.
- Numbers are cell labels.
- To expand a cell, *substitute a form for a label*.
- The circuit is technology mapped when the form in each cell matches a library form.

Technology library

{ INV, NOR2, AND2, OR3, OR5 } fanout = 3

{(A),(A B),((A)(B)),((A B C)),((A B C D E))}

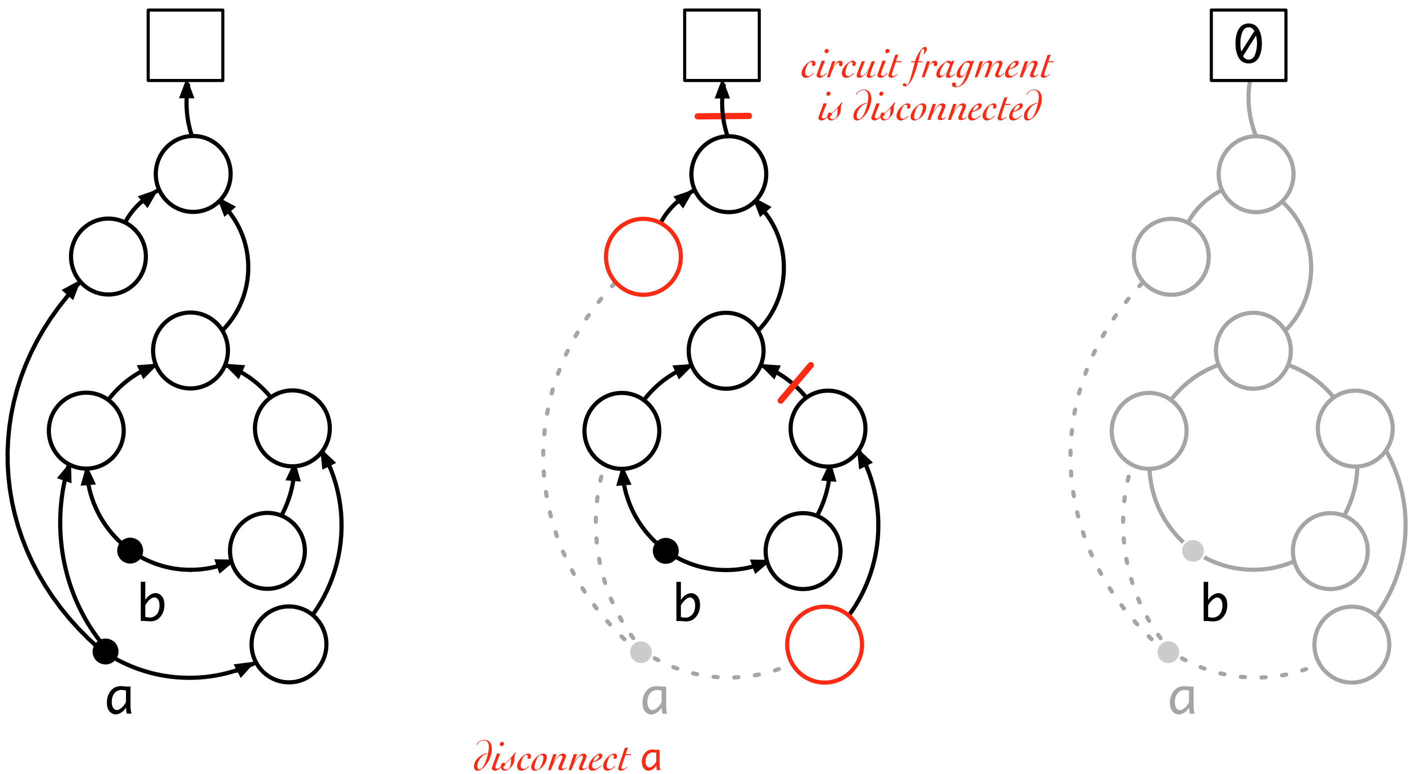
```
((eq 12) (gt 34) (lt 35))     -- output pins

 ((1    ( a)              )
  (2    ( b)              )
  (3    ( c)              )
  (4    ( d)              )
  (5    ( e)              )   -- INVERTERS
  (6    ( f)              )
  (7    ( g)              )
  (8    ( h)              )
  (9    ( j)              )
  (10   (30)              )
  (11   (33)              )
  (12   (31)              )
  (13   (32)              )
  (14   (a 2)             )
  (15   (b 1)             )
  (16   (c 4)             )
  (17   (d 3)             )   -- NOR2 gates
  (18   (e 6)             )
  (19   (f 5)             )
  (20   (g 8)             )
  (21   (h 7)             )
  (22   (( j)(20))        )
  (23   (( j)(21))        )
  (24   ((11)(16))        )
  (25   ((11)(17))        )   -- AND2 gates
  (26   ((10)(18))        )
  (27   ((10)(19))        )
  (28   ((13)(14))        )
  (29   ((13)(15))        )
  (30   (( 9 20 21))      )
  (31   ((14 15 32))      )   -- OR3 gates
  (32   ((16 17 33))      )
  (33   ((18 19 30))      )
  (34   ((i 22 24 26 28)) )   -- OR5 gates
  (35   ((k 23 25 27 29)) )))
```

Copyright © 2019 William M. Bricken. All rights reserved.

Evaluation by Occlusion

sufficient for evaluation

Occlusion `(A ( )) =`



Inputs are either deleted (0) or asserted as a distinction (1).

Evaluation is asynchronous and strongly parallel.

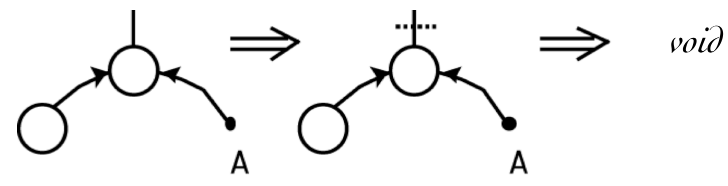
Copyright © 2019 William M. Bricken. All rights reserved.

Distinction Network Optimization

sufficient for reduction

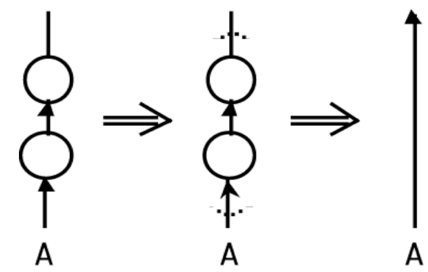
Occlusion

(A ( )) =



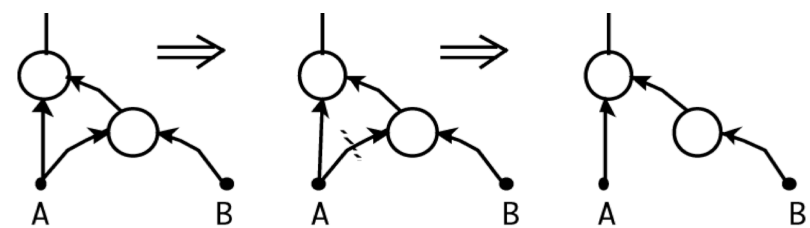
Involution

((A)) = A



Pervasion

A {A B} = A {B}



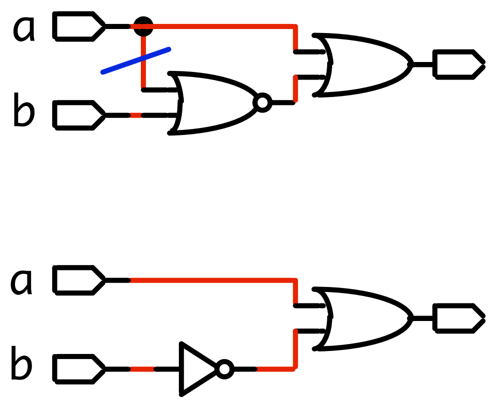
Communication between nodes is local with no global coordination.

Transformation is asynchronous and strongly parallel.

W. Bricken (1995) Distinction Networks, in I. Wachsmuth *et al* (eds) *KI-95 Advances in Artificial Intelligence*, Springer, p.35-48.

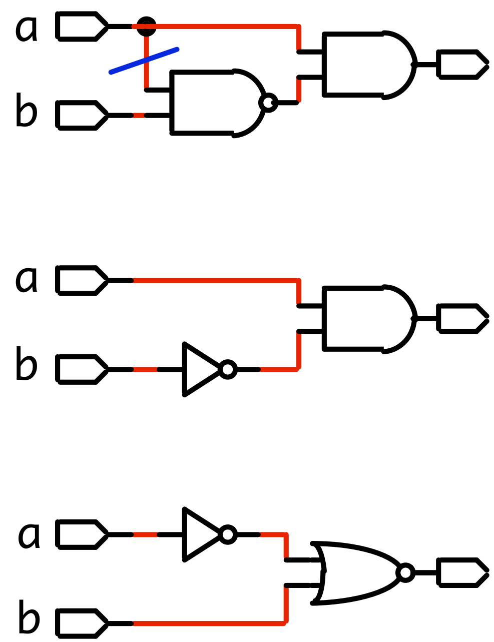
Copyright © 2019 William M. Bricken. All rights reserved.

Path Deletion by Pervasion



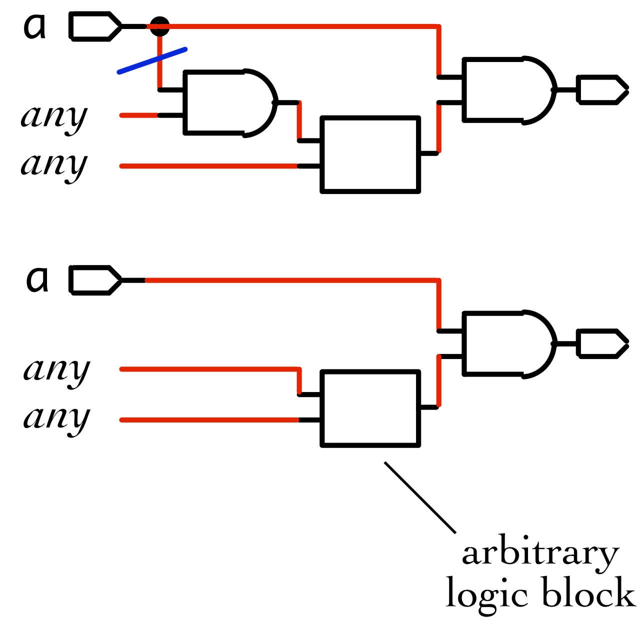
Example I

```
a ( a b )
a (   b )
```



Example II

```
((a) ((a)(b)))
((a) (   (b)))
((a)      b  )
```



Example III

```
((a) { … ((a)(…)) })
((a) { … (   (…)) })
((a) { …      …   })
```

Reducing *reconvergence* simplifies timing.

Copyright © 2019 William M. Bricken. All rights reserved.

Technical EDA Issues

Circuit design industry (Electronic Design Automation)

- 400,000 engineers in US
- $400 billion/year industry
- VLSI design: more than 1 million logic gates
- computational circuits over 50 billion transistors
- memory units over 1 trillion transistors

VLSI: Very Large Scale Integration of semiconductor chips

- delay minimization and global optimization
- verification and equivalence testing
- technology mapping to different libraries and architectures
- symmetry detection and abstraction
- timing and synchronization
- power consumption
- fault tolerance
- manufacturability and yield

Resistance to change

- existing tools are excellent
- *disruptive technologies can cost more in retraining than they gain in performance*

Copyright © 2019 William M. Bricken. All rights reserved.

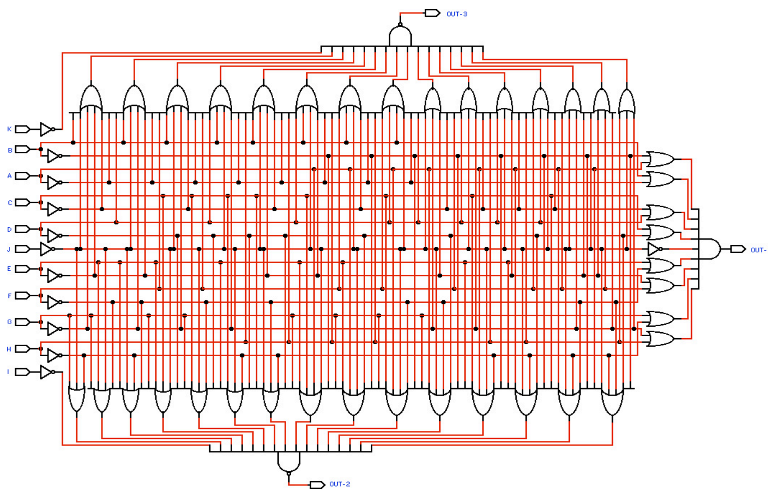
Circuit Design Generator

1999–2002: Interval Research Corporation & BTC

- logic synthesis (area and delay)
- technology mapping
- design exploration, abstraction, partitioning

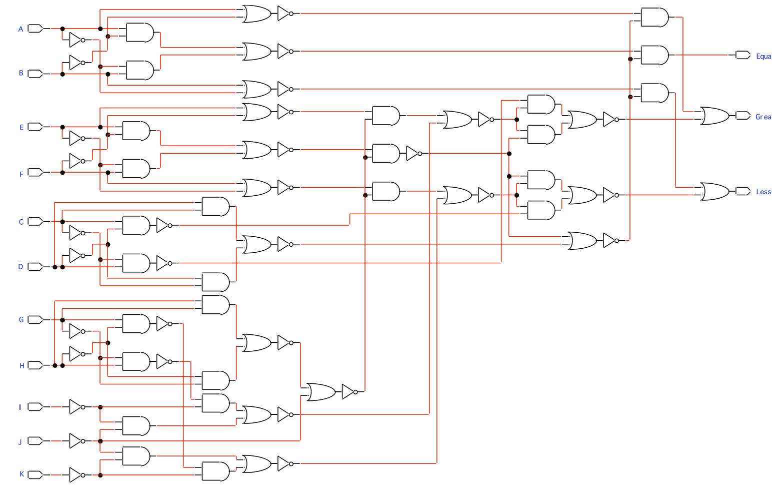
Copyright © 2019 William M. Bricken. All rights reserved.

Current Techniques (4-bit comparator)

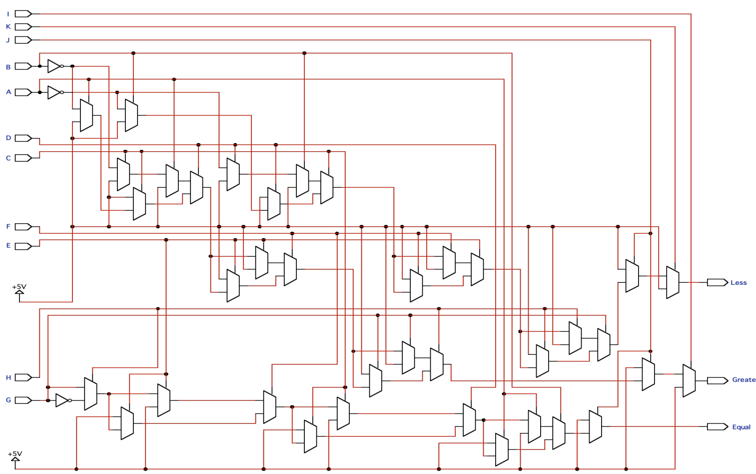


two-level logic (PLA)

Pattern: `A ((B)(C)) => ((A B)(A C))`

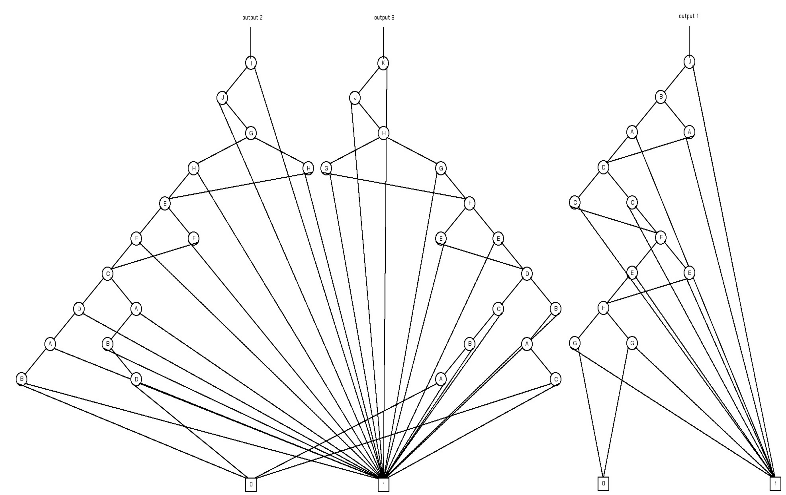


multilevel logic (ASIC)



multiplexor logic (MUX)

Pattern: `(((A) B) (A C))`

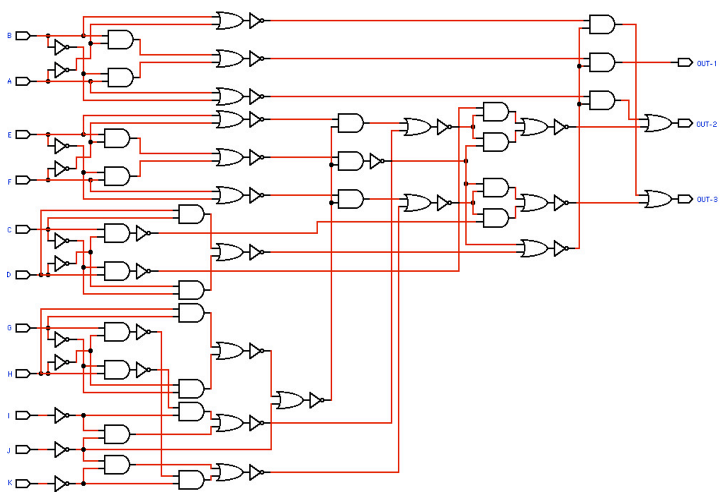


binary decision diagram (BDD)

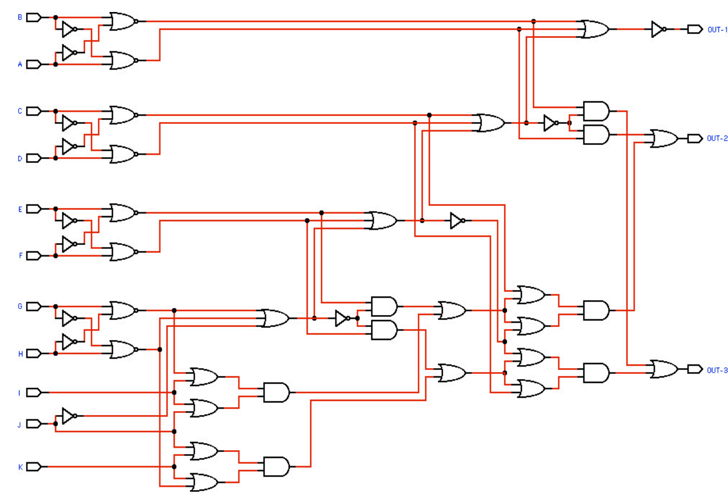
Pattern: `occlusion paths`

Copyright © 2019 William M. Bricken. All rights reserved.

Multilevel Structural Optimization

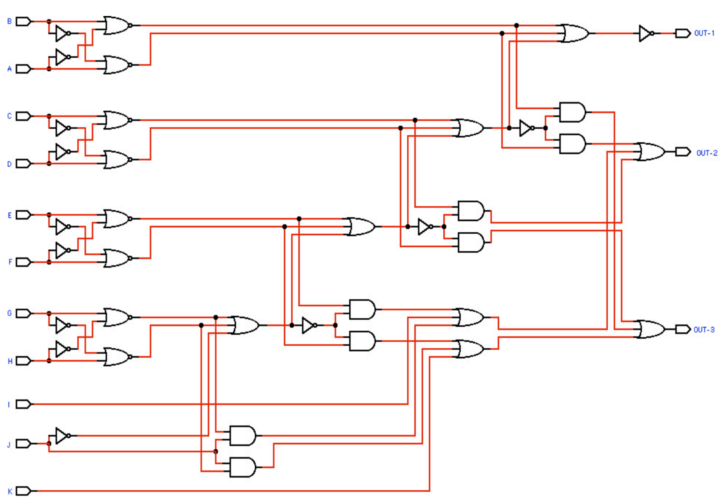


EDA software generated design



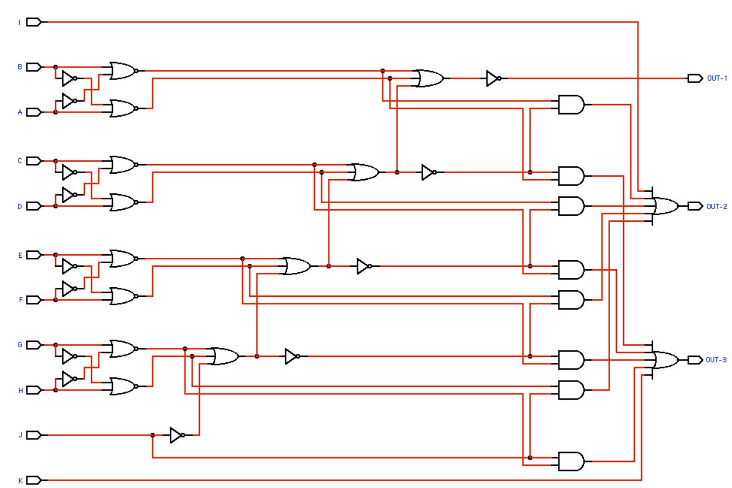
remove redundancy

Pattern: `A (A B) => A (B)`



reduce reconvergence

Pattern: `((A B)(A C)) => A ((B)(C))`

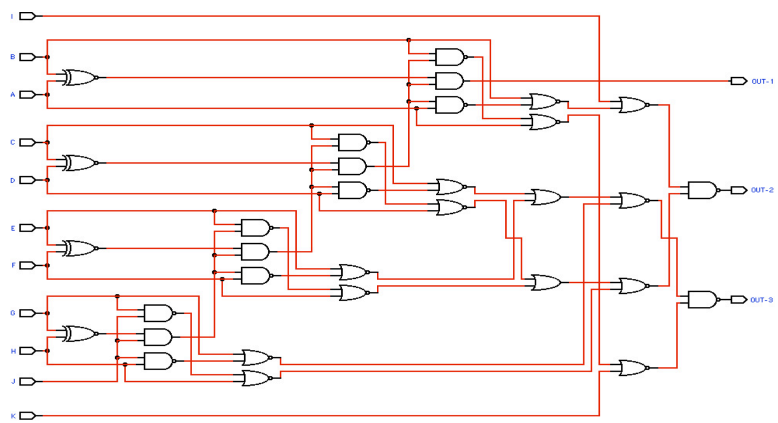


reduce fanout

boundary logic optimized design

Copyright © 2019 William M. Bricken. All rights reserved.

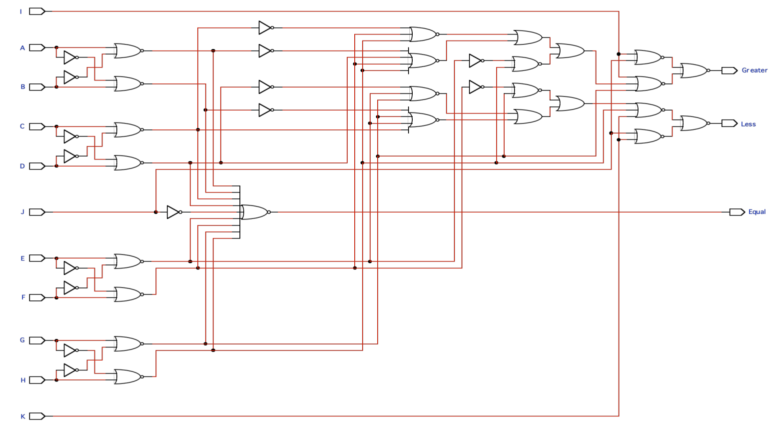
Technology Mapping



specific library, fanin = 2, fanout = 3

Library: { NOR2, OR2, NAND2, AND2, XOR2 }

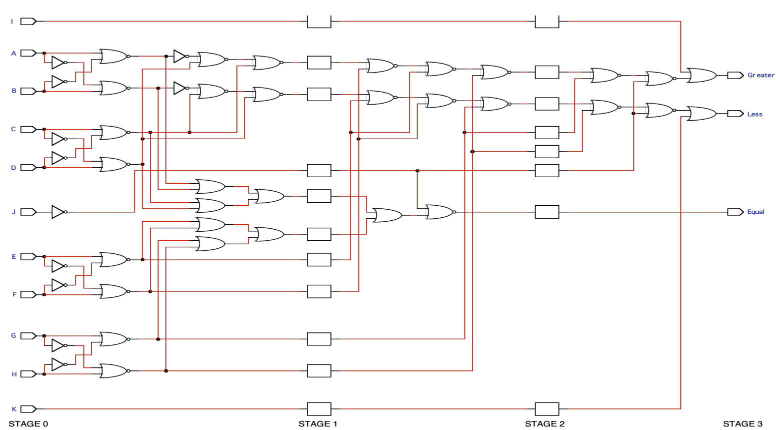
Pattern: (A B C D) ==> (((A B)) ((C D)))



reduced critical timing path, fanout = 4

Library: { INV, NOR2, NOR3, NOR4, NOR9 }

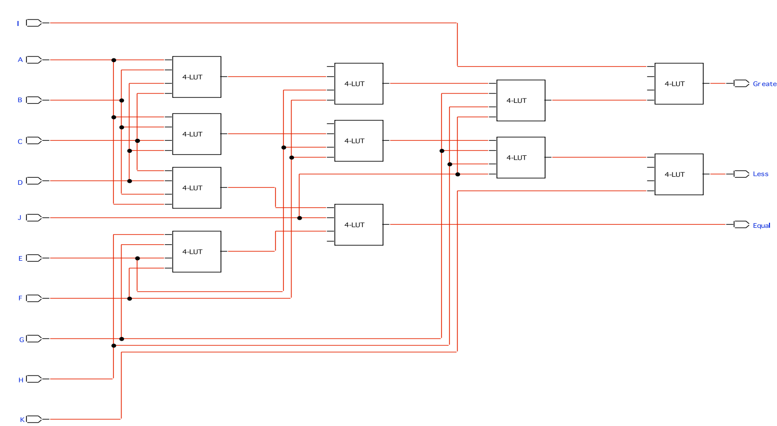
Pattern: (A ((B)(C))) ==> (A B)(A C)



3 gate pipeline, fanin = 2

Library: { INV, NOR2, OR2 }

Pattern: (A (B (C (D (E (F))))))



4-input look-up tables

Library: { 4LUT }

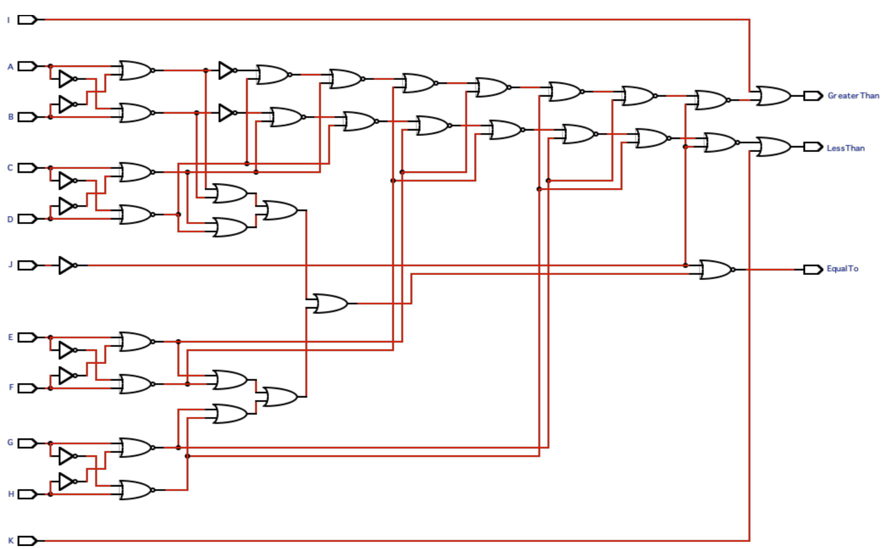
Pattern: (A (B (C (D (E)))))

Copyright © 2019 William M. Bricken. All rights reserved.

Design for Testability

Minimal structural variance

- maximum fanin = 2
- maximum fanout = 3
- fast transistor chain
- no reconvergence
- *enables* bypass functional logic



Technology library

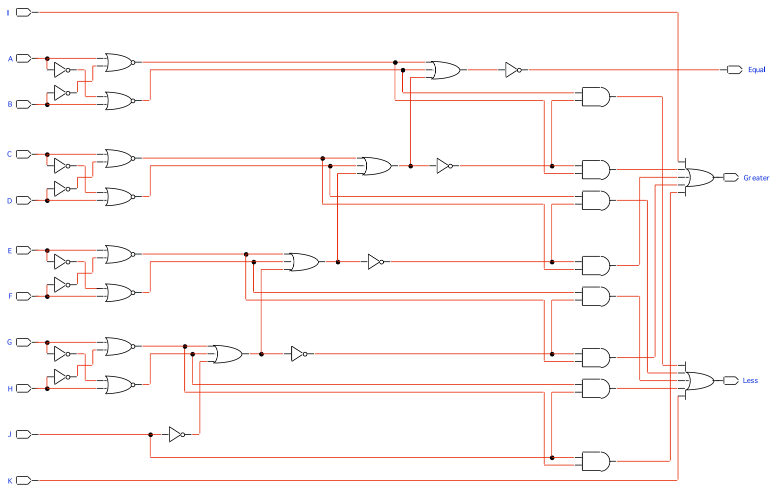
{ INV, NOR2, OR2 } fanout = 3

{(A),(A B),((A B))}

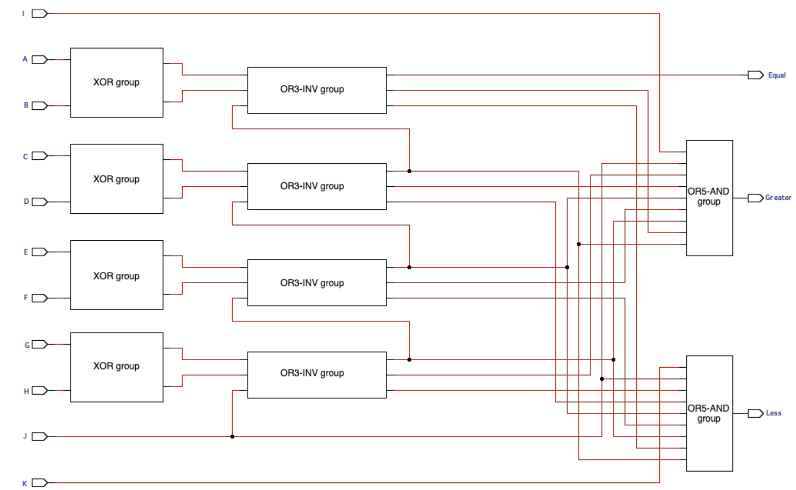
```
((eq 36) (gt 20) (lt 21))

 ((1   ( a)       )
  (2   ( b)       )
  (3   ( c)       )
  (4   ( d)       )       -- INVERTERS
  (5   ( e)       )
  (6   ( f)       )
  (7   ( g)       )
  (8   ( h)       )
  (9   ( j)       )
  (10  (12)       )
  (11  (13)       )
  (12  (a 2)      )
  (13  (b 1)      )
  (14  (c 4)      )       -- NOR2 gates
  (15  (d 3)      )
  (16  (e 6)      )
  (17  (f 5)      )
  (18  (g 8)      )
  (19  (h 7)      )
  (20  ((i 22))   )       -- OR2 gates
  (21  ((k 23))   )
  (22  ( 9 32)    )
  (23  ( 9 34)    )
  (24  (10 15)    )
  (25  (11 14)    )
  (26  (14 24)    )       -- NOR2 gates
  (27  (15 25)    )
  (28  (16 31)    )
  (29  (16 27)    )
  (30  (17 29)    )
  (31  (17 26)    )
  (32  (18 35)    )
  (33  (18 30)    )
  (34  (19 33)    )
  (35  (19 28)    )
  (36  ( 9 45)    )
  (39  ((12 13))  )
  (40  ((14 15))  )       -- OR2 gates
  (41  ((16 17))  )
  (42  ((18 19))  )
  (43  ((39 40))  )
  (44  ((41 42))  )
  (45  ((43 44))  )))
```

Copyright © 2019 William M. Bricken. All rights reserved.

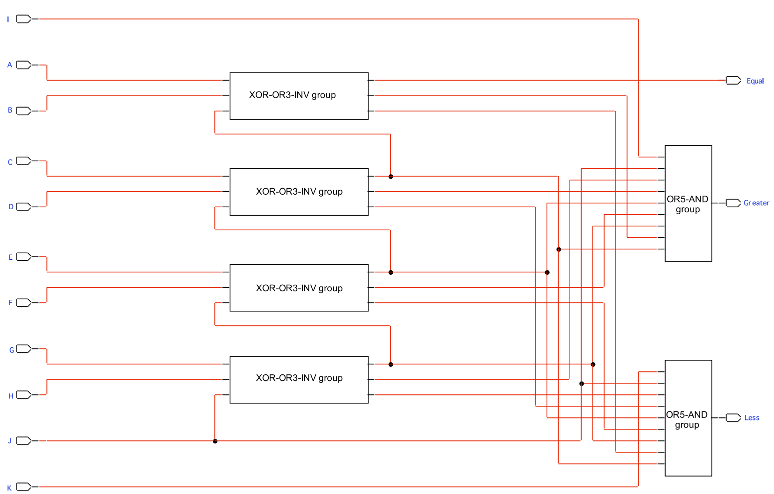
Behavioral Abstraction



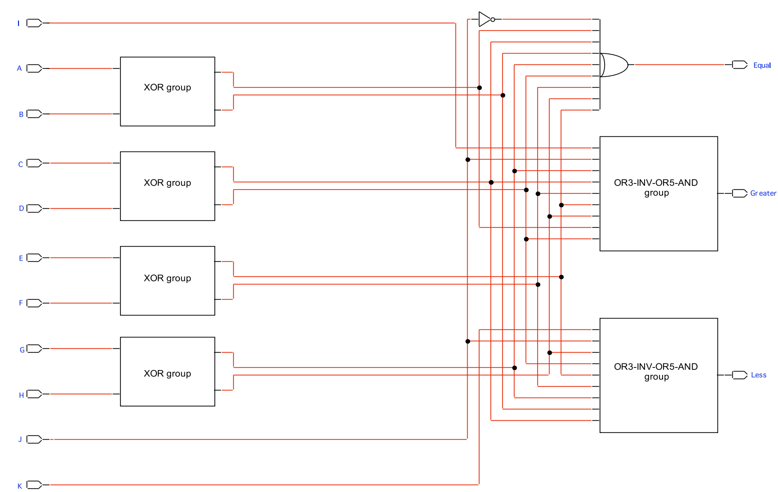
Minimized design



Functional abstraction



Sequential abstraction



Parallel abstraction

Copyright © 2019 William M. Bricken. All rights reserved.

Vector Abstraction

```
(((oa =17))
  ((!1   (@  35)  )     (35  (((18)( !1)) (36  (18  !1)))  )     (=66   (((=50  !1) ((=50)( !1))))  )
   (!2   (@ =66)  )     (36  (((19)( !2)) (37  (19  !2)))  )     (=67   (((=51  !2) ((=51)( !2))))  )
   (!3   (@ =67)  )     (37  (((20)( !3)) (38  (20  !3)))  )     (=68   (((=52  !3) ((=52)( !3))))  )
   (!4   (@ =68)  )     (38  (((21)( !4)) (39  (21  !4)))  )     (=69   (((=53  !4) ((=53)( !4))))  )
   (!5   (@ =69)  )     (39  (((22)( !5)) (40  (22  !5)))  )     (=70   (((=54  !5) ((=54)( !5))))  )
   (!6   (@ =70)  )     (40  (((23)( !6)) (41  (23  !6)))  )     (=71   (((=55  !6) ((=55)( !6))))  )
   (!7   (@ =71)  )     (41  (((24)( !7)) (42  (24  !7)))  )     (=72   (((=56  !7) ((=56)( !7))))  )
   (!8   (@ =72)  )     (42  (((25)( !8)) (43  (25  !8)))  )     (=73   (((=57  !8) ((=57)( !8))))  )
   (!9   (@ =73)  )     (43  (((26)( !9)) (44  (26  !9)))  )     (=74   (((=58  !9) ((=58)( !9))))  )
   (!10  (@ =74)  )     (44  (((27)(!10)) (45  (27 !10)))  )     (=75   (((=59 !10) ((=59)(!10))))  )
   (!11  (@ =75)  )     (45  (((28)(!11)) (46  (28 !11)))  )     (=76   (((=60 !11) ((=60)(!11))))  )
   (!12  (@ =76)  )     (46  (((29)(!12)) (47  (29 !12)))  )     (=77   (((=61 !12) ((=61)(!12))))  )
   (!13  (@ =77)  )     (47  (((30)(!13)) (48  (30 !13)))  )     (=78   (((=62 !13) ((=62)(!13))))  )
   (!14  (@ =78)  )     (48  (((31)(!14)) (49  (31 !14)))  )     (=79   (((=63 !14) ((=63)(!14))))  )
   (!15  (@ =79)  )     (49  (((32)(!15)) (82  (32 !15)))  )     (=80   (((=64 !15) ((=64)(!15))))  )
   (!16  (@ =80)  )     (82  (((33)(!16)) (( ) (33 !16)))  )     (=17   (((=65     ) ((=65)(    ))))  )))
   (!V1  (@ =V2)  )     (V5  (((V3)(!V1)) (V5s (V3 !V1)))  )     (=V2s  (((=V6 !V1) ((=V6)(!V1))))  )
```

common patterns in red

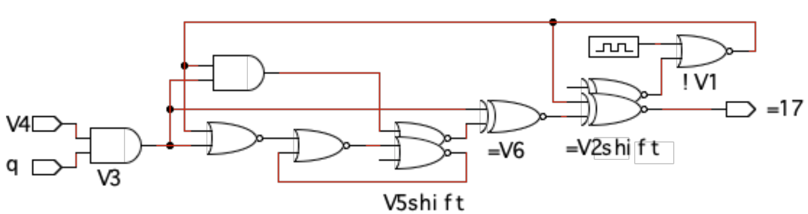
```
   (18  (( a)(q))  )     (=50  (((18 36) ((18)(36))))   )
   (19  (( b)(q))  )     (=51  (((19 37) ((19)(37))))   )
   (20  (( c)(q))  )     (=52  (((20 38) ((20)(38))))   )
   (21  (( d)(q))  )     (=53  (((21 39) ((21)(39))))   )
   (22  (( e)(q))  )     (=54  (((22 40) ((22)(40))))   )
   (23  (( f)(q))  )     (=55  (((23 41) ((23)(41))))   )
   (24  (( g)(q))  )     (=56  (((24 42) ((24)(42))))   )
   (25  (( h)(q))  )     (=57  (((25 43) ((25)(43))))   )
   (26  (( i)(q))  )     (=58  (((26 44) ((26)(44))))   )
   (27  (( j)(q))  )     (=59  (((27 45) ((27)(45))))   )
   (28  (( k)(q))  )     (=60  (((28 46) ((28)(46))))   )
   (29  (( l)(q))  )     (=61  (((29 47) ((29)(47))))   )
   (30  (( m)(q))  )     (=62  (((30 48) ((30)(48))))   )
   (31  (( n)(q))  )     (=63  (((31 49) ((31)(49))))   )
   (32  (( o)(q))  )     (=64  (((32 82) ((32)(82))))   )
   (33  (( p)(q))  )     (=65  (((33 !16)((33)(!16)))) )
   (V3  ((V4)(q))  )     (=V6  (((V3 V5s)((V3)(V5s)))) )
```

8-bit sequential multiplier

Vectorized form

```
((!V1   (@ =V2)                              )
  (V3   ((V4)(q))                            )
  (V5   (((V3)(!V1)) (V5s (V3 !V1)))         )
  (=V6  (((V3 V5s)((V3)(V5s))))              )
  (=V2s (((=V6 !V1)((=V6)(!V1))))            ))
```

Pseudo-circuit



Copyright © 2019 William M. Bricken. All rights reserved.

RECONFIGURABLE HARDWARE DESIGN

2002–2004: Unary Computers & BTC

- dynamically reconfigurable hardware design
- design abstraction, partitioning, place&route

Copyright © 2019 William M. Bricken. All rights reserved.

Logic Block Architecture

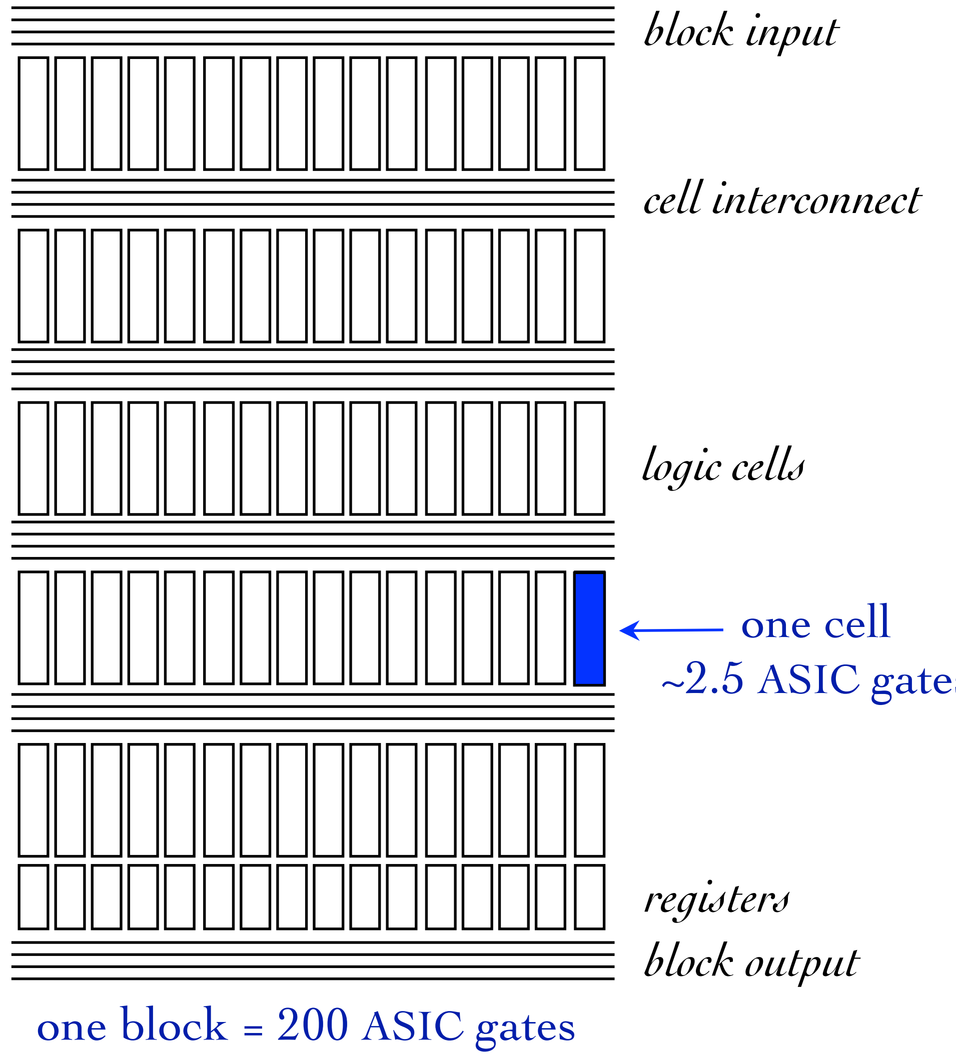
hierarchical pipelined multilevel logic

Hundreds of conventional circuits expressed as dnets were *statistically analyzed for common patterns of distinctions*.

Cells are designed to cover dnet patterns.

Each cell can be *dynamically reconfigured* to the functionality of about 2.5 conventional logic gates.

Each block coordinates 80 cells as *a single unified timed logic element*.



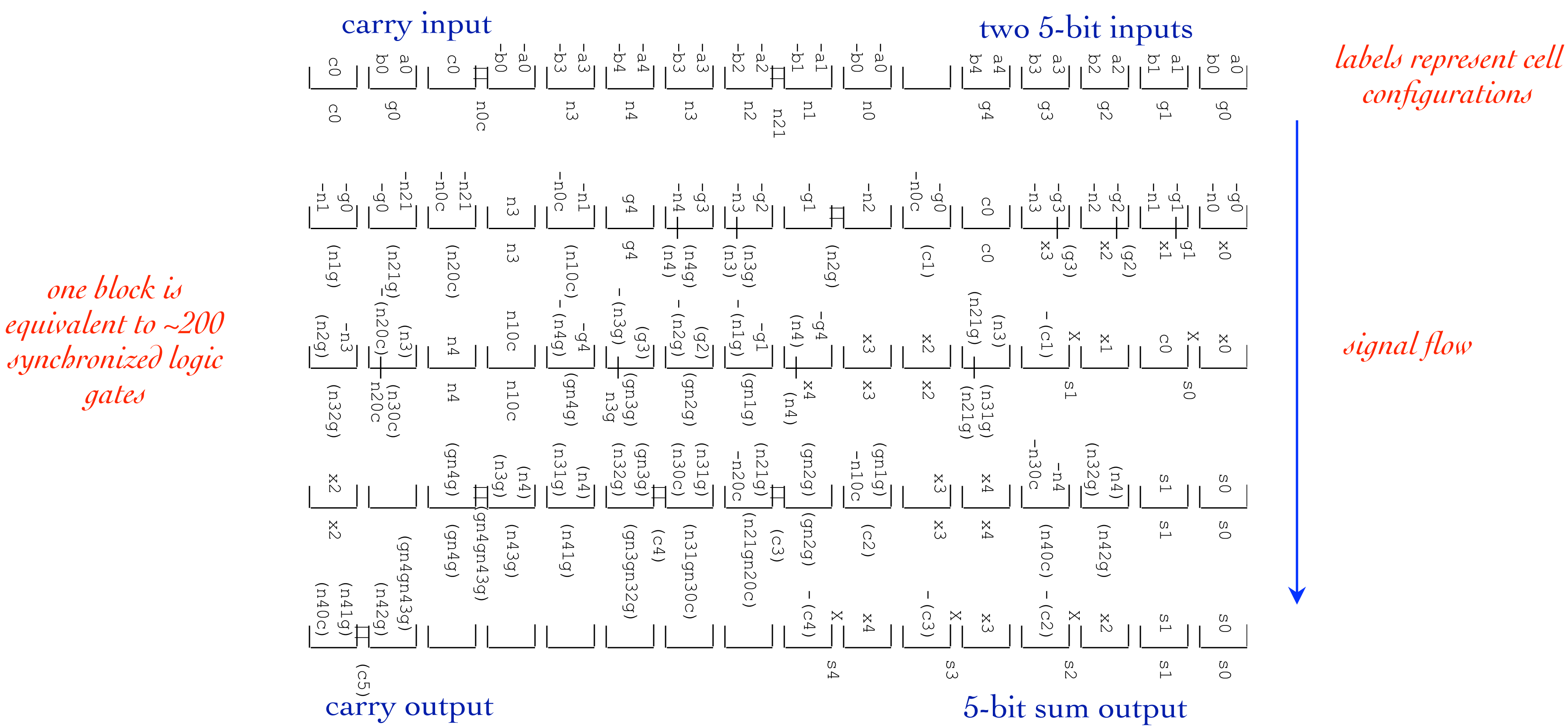
2004 wire technology: 130 µm
2004 block area: 16,000 µm^2 (.016 mm^2)
2004 gate density: 12,000 gates/mm^2

2019 wire technology: 7 µm
2019 block area: 30 µm^2
2019 gate density: 6,000,000 gates/mm^2

Copyright © 2019 William M. Bricken. All rights reserved.

Place and Route

5-bit adder



Optimization, layout and routing generated by applying simple boundary pattern transformations.

Copyright © 2019 William M. Bricken. All rights reserved.

Reconfigurable Chip Architecture

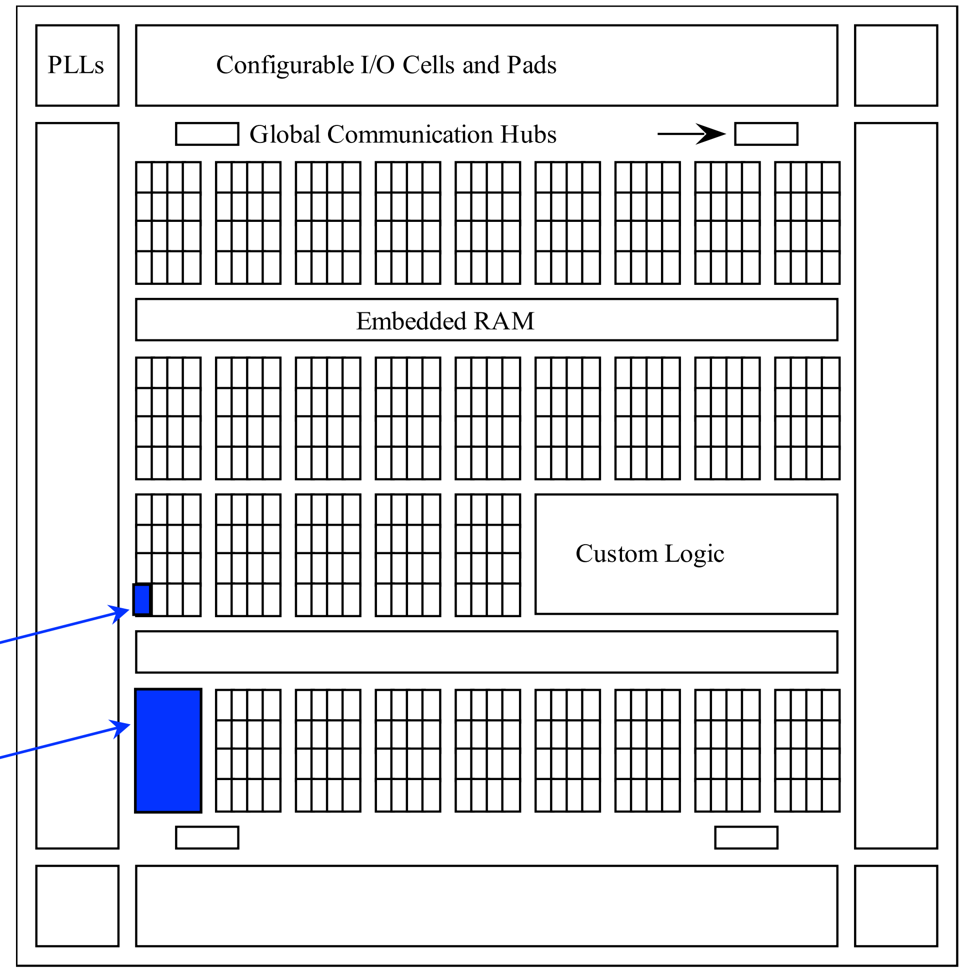
2019 technology is 300 times smaller

Co-designed software specification and hardware layout using *distinction patterns*

Fine-grain control of logic/routing trade-offs

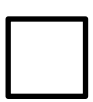
Synchronized timing eliminates timing analysis
2004 delay: 1.8 ns per block, *any logic*
2.7 ns across chip, *any location*

Regular cell structure for ease of fabrication



one block = 200 ASIC gates

one block-neighborhood = 3,200 ASIC gates



chip area: *7* x *7* mm = *49* mm^2

2004 chip: 32 block-neighborhoods provide 100,000 logic gates
2019 chip: 10,000 neighborhoods provide 30,000,000 logic gates

Copyright © 2019 William M. Bricken. All rights reserved.

INNOVATIVE HARDWARE DESIGN

(alternative dnet architectures)

1997-2005: Interval Research Corporation & Unary Computers

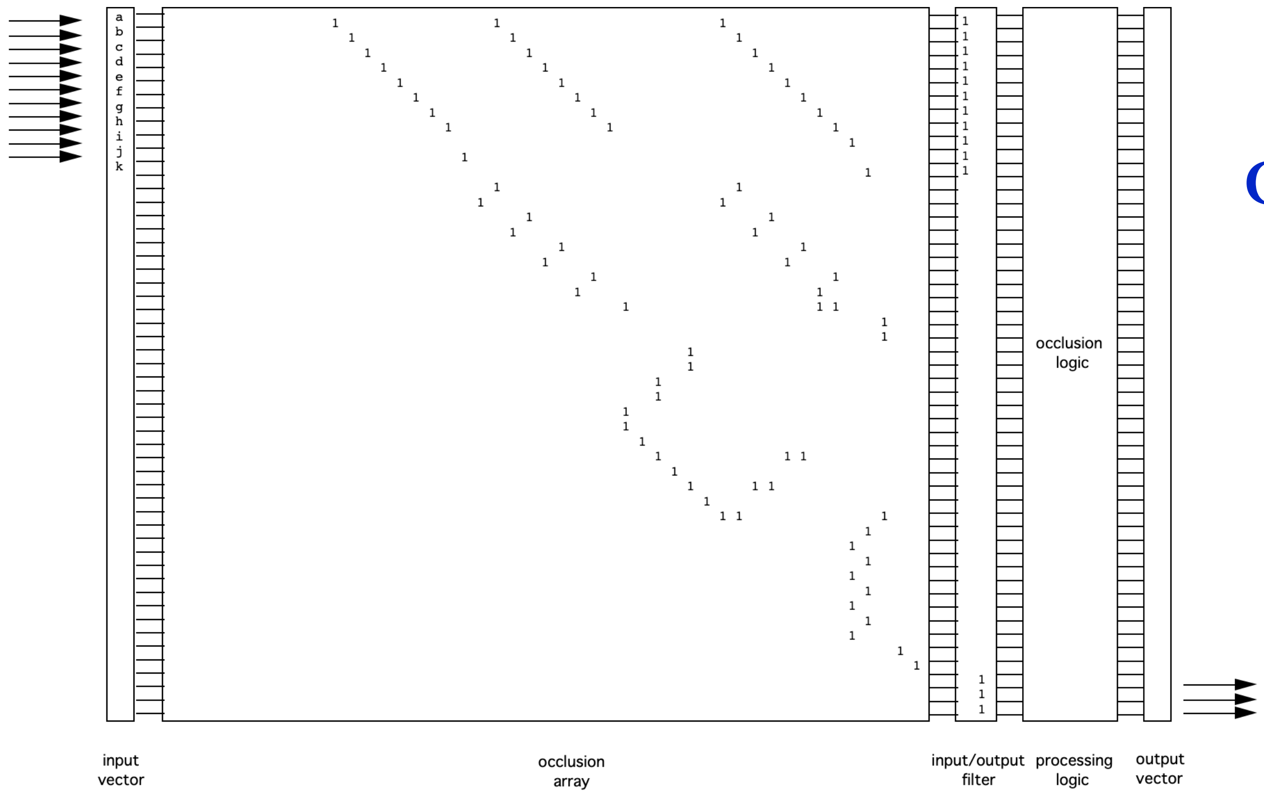
exotic architectures
- bit-stream circuit simulator
- boundary logic RISC instruction set
- inverting bar architecture
- reconfigurable occlusion array
- reconfigurable computation mesh

Copyright © 2019 William M. Bricken. All rights reserved.

Reconfigurable Occlusion Array

4-bit magnitude comparator

like a Pachinko machine



Occlusion

(A ( )) =

unary rather than binary

Dnets are implemented as a spatial array of distinctions.

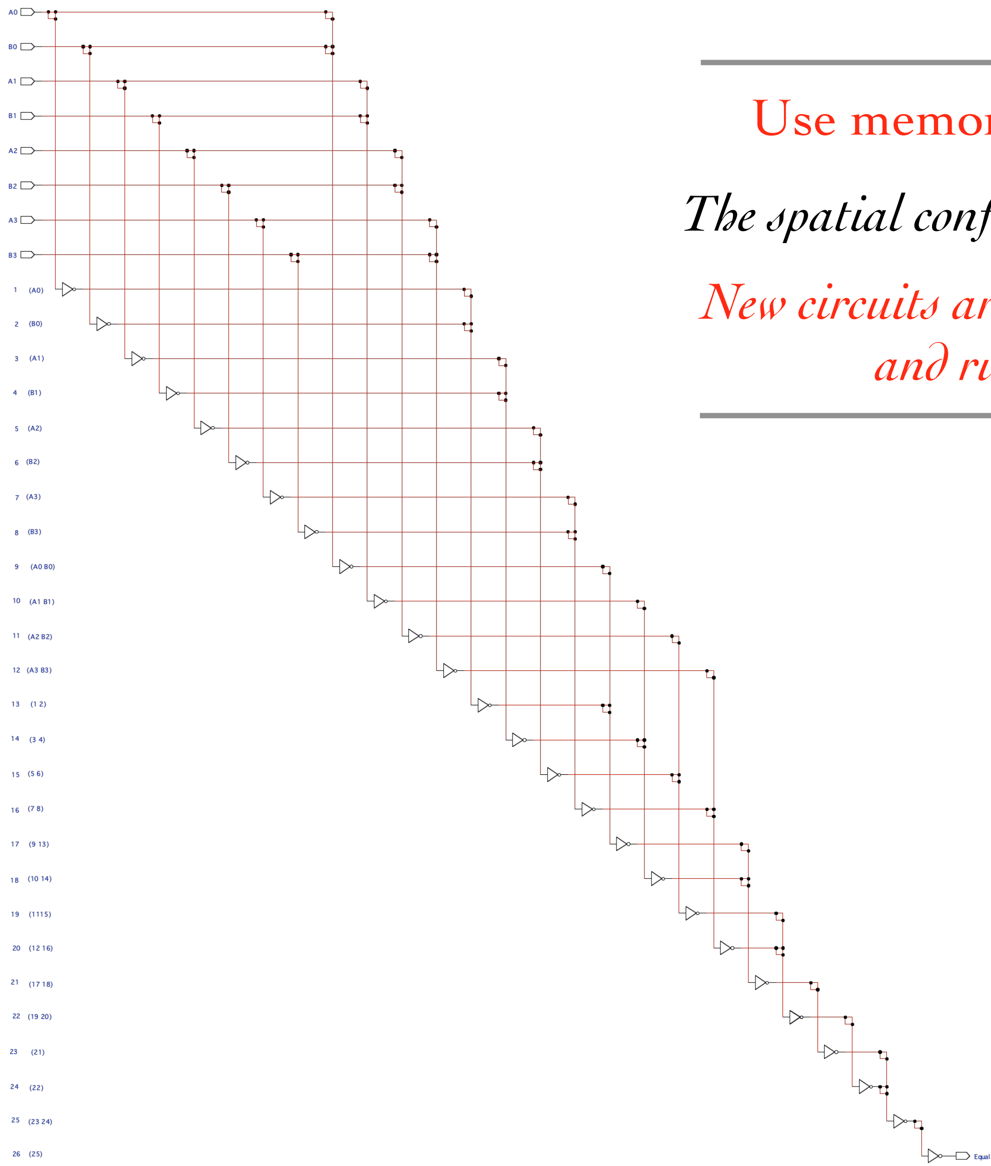
Wiring is virtual. Connectivity is a threaded array of disconnection locations.

Change is virtual. Disconnection is recorded by marking a memory cell.

Timing is virtual. Terminates when all output distinctions are marked.

Copyright © 2019 William M. Bricken. All rights reserved.

Reconfigurable Computational Mesh

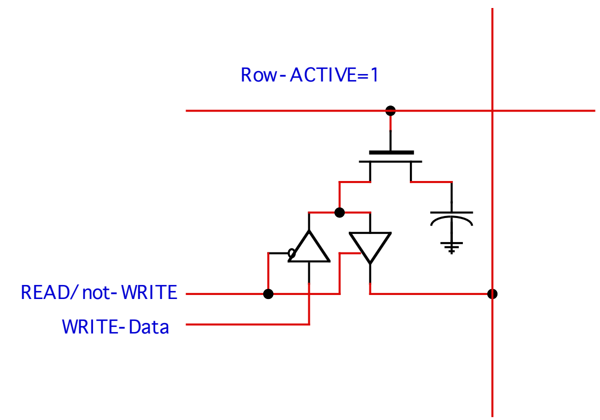


optimized 4-bit comparator

Use memory architecture for computation.

The spatial configuration of memory bits is the circuit.

New circuits are built as quickly as memory WRITE and run as fast as memory READ.



DRAM crosspoint

- standard memory cell
- WRITE to configure circuit
- READ to run circuit

Copyright © 2019 William M. Bricken. All rights reserved.

Conclusion

We have been developing the theory and application of boundary mathematics for two decades.

The extent to which boundary techniques differ from well known forms of mathematics is both a major political challenge and a significant technical advantage.

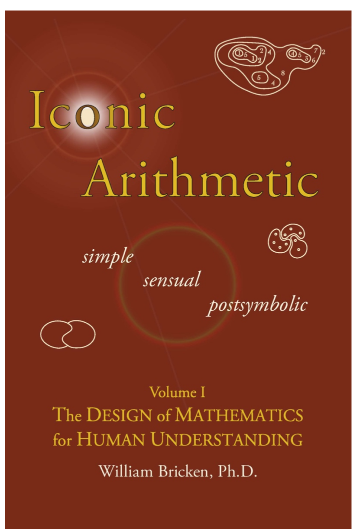
This presentation has emphasized boundary logic. There are equally interesting developments in imaginary and re-entrant boundary forms and in boundary numerics.

Copyright © 2019 William M. Bricken. All rights reserved.

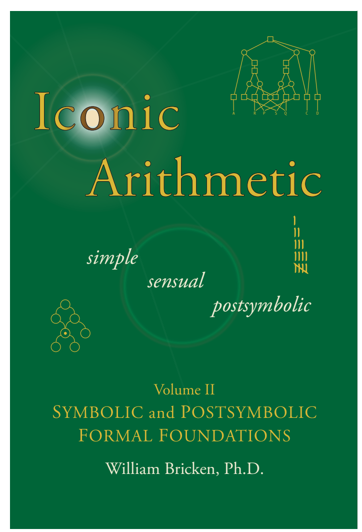
THANK YOU!

william@iconicmath.com

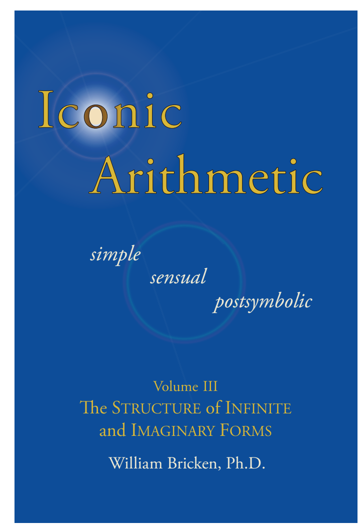
recent work: ICONIC ARITHMETIC



NEW 2018



NEW 2019



COMING 2020

Copyright © 2019 William M. Bricken. All rights reserved.