Exact Combinational Logic Synthesis and Non-Standard Circuit Design

Paul Tarau
Dept. of Computer Science and Engineering
University of North Texas, Denton, Texas
paul.tarau@gmail.com

Brenda Luderman
ACES CAD
Lewisville, Texas
brenda.luderman@gmail.com

ABSTRACT

Using a new exact synthesizer that automatically induces minimal universal boolean function libraries, we introduce two indicators for comparing their expressiveness: the first based on how many gates are used to synthesize all binary operators, the second based on how many N-variable truth table values are covered by combining up to M gates from the library. By applying the indicators to an exhaustive enumeration of minimal universal libraries, two dual asymmetrical operations, Logic Implication "$\Rightarrow$" and Half XOR "$<$" are found to consistently outperform their symmetrical counterparts, NAND and NOR. Our expressiveness metrics bring support to the conjecture that asymmetrical operators are significantly more expressive that their well studied symmetric counterparts, omnipresent in various circuit design tools.

ACM Categories and Subject Descriptors: B.6 **[Logic Design]:** B.6.1 **[Design Styles]: Combinational logic**

General Terms: *Algorithms, Design, Theory, Measurement, Performance*

Keywords: *exact combinational circuit synthesis, asymmetrical logic operators, minimal universal boolean logic libraries, minimal transistor-count circuits*

1. INTRODUCTION

Exact circuit synthesis has been a recurring topic of interest in circuit design, complexity theory, boolean logic, combinatorics and graph theory for more than half a century [11, 25, 20, 5, 4, 12]. While its extreme intractability (typically, single digit number of gates for most problems) could be an explanation for having attracted some of the best minds in the aforementioned research fields, our main reason for revisiting it in this paper is more practically driven.

Traditional silicon CMOS manufacturing relies often on hundreds of hand-made library cells covering most of the gate combinations and transistor sizes used in typical circuits. As quantum effects leading to increased transistor leakage and noise are getting harder and harder to circumvent for deep submicron designs (and often involve using new materials and processes), porting such large libraries to the new manufacturing technologies is labor intensive and costly.

Polymorphic or multi-functional circuits [24], often evolved using genetic programming have emerged [23, 28]. With such circuits, that overlap digital logic with unconventional control parameters ranging from voltage to temperature [24, 29] reuse of traditional libraries can be particularly difficult.

Further down the road, beyond the next decade, the advent of alternative circuit implementations might involve radical departures from traditional CMOS processes, ranging from optical and quantum computing to biological or molecular techniques [16]. In some of these fields, manufacturability is likely to limit the variety of gate-level building blocks. This also implies that the resulting libraries might have to use as few as possible gates, involving unconventional, yet unknown processes.

Given a library of universal gates, the exact synthesis of boolean circuits consists of finding minimal representation using only gates of the library.

Fairly efficient exact synthesis programs using symmetric operators have been described as early as in [5] and [4]. Knuth in [11], section 7.1.2 mentions asymmetric operators like $A < B$ as forming one of the 5 (out of 16) boolean functions used as part of a *boolean chain* (sequence of connected 2-argument boolean functions) needed for synthesis by exhaustive enumeration. Interestingly, the other 4 are: $>, *, +, \oplus$. Note that $>$ is the symmetric of $<$, and that with its exception, $*, +, \oplus$ have been heavily used in various synthesis algorithms. Knuth also computes minimal representations of all 5-argument functions using a clever reduction to equivalence classes.

Rewriting/simplification has been used in various forms in recent work on multi-level synthesis [17, 18] using non-SOP encodings ranging from And-Inverter Gates (AIGs) and XOR-AND nets to graph-based representations in the tradition of [2]. While not explicitly implying the use of the asymmetrical operators $<$ and $\Rightarrow$, AIGs can be used to implicitly express them, given that $A < B$ is equivalent to $\sim A * B$ and $A \Rightarrow B$ is equivalent to $\sim (A* \sim B))$.

Interestingly, new synthesis targets, ranging from AIGs to cyclic combinational circuits [21], turned out to be competitive with more traditional minimization based synthesis techniques. Synthesis of reversible circuits with possible uses in low-power adiabatic computing and quantum computing [26, 13, 14, 13] have emerged. Despite its super-exponential

complexity, exact circuit synthesis efforts have been reported successful for increasingly large circuits [8, 11].

This paper extends our recent work on the use of a logic programming framework for circuit synthesis described in [30, 31] by replacing the fixed library/Leaf DAG based algorithm with a more general DAG-based algorithm, configurable to support libraries given as run-time parameters. The new algorithm is also faster, as it optimizes execution time through aggressive constraint propagation.

We implement full automation of exact synthesis tasks, covering automated discovery of universal libraries and a priori estimation of tractability of a given problem, with special focus on circuit types likely to be relevant for nanoscale processes.

While the quantitative expressiveness comparison of two libraries can be solved through exact synthesis for the small cases when it is tractable, the challenge is to extend this to the intractable cases that appear in practical design. We achieve this with two indicators for comparing their expressiveness: the first based on how many gates are used to synthesize all binary operators, the second based on how many N-variable truth table values are covered by combining up to M gates from the library.

By applying the indicators to an exhaustive enumeration of minimal universal libraries, two dual asymmetrical operations, Logic Implication "⇒" and Half XOR "<" are found to consistently outperform their symmetrical counterparts, NAND and NOR, leading to the conjecture that *asymmetrical operators are significantly more expressive that their well studied symmetric counterparts*, omnipresent in various circuit design tools.

2. EXACT COMBINATIONAL CIRCUIT SYNTHESIS AS COMBINATORIAL GENERATION + CONSTRAINT PROPAGATION

Our exact synthesis algorithm uses depth-first backtracking and constraint propagation to find minimal N-input, M-output circuits representing boolean functions, based on a given library of operators and constants.

Synthesis Algorithm

1. First, obtain an output specification from a symbolic formula and compute a conservative upper limit (in terms of a cost function, for instance the number of gates) on the size of the synthesized expression.

2. Next, enumerate candidate circuits (represented as directed acyclic graphs) *in increasing cost order*, to ensure that minimal circuits are generated first. This involves the following steps:

   (a) Encode constants 0,1 and N primary input variables as bitstrings of size 2^N representing truth tables as described at the end of this section

   (b) Initialize the list of available gates as being the set of primary inputs.

   (c) Until a maximum number of gates is reached, connect a new gate's inputs to the previously constructed gates' outputs. The nondeterministic choice of a new gate's connections is a combinatorial step that is implemented efficiently using backtracking and constraint propagation. The computation of the outputs is also combinatorial, with respect to using any of the possible gates in the library that match the constraints. At each step constraints are generated and checked as follows:

      i. propagate back the known values of the primary outputs to candidate gates

      ii. compute the values of the outputs as bitstrings encoding all bit combinations, as soon as their inputs are known

      iii. ensure that all gate outputs are distinct

3. On success, the resulting circuit is decoded into a symbolic expression consisting of a list of primary input variables, a list of gates describing the operators and their input and output arguments, and a list of primary output variables.

4. At the end, the following assertions hold:

   (a) The symbolic expression is guaranteed to evaluate to the list of truth tables provided initially or obtained by evaluation of a given boolean expression.

   (b) The list of gates is guaranteed to be minimal, given that circuits are generated in increasing gate order.

Synthesizing Minimal Universal Libraries.

Most minimal universal boolean function libraries have been discovered and documented in fields ranging from symbolic logic and complexity theory to circuit design. However, as they were needed as input for evaluating expressiveness of various libraries, we decided to induce them and prove their minimality automatically by adapting our synthesis algorithm as follows:

1. Encode the 2^N binary operators as integers from 0 to $2^N - 1$ based on the value columns of their truth tables (seen as bitstrings)

2. Generate candidate libraries as subsets of K operators for increasing values of K (extended progressively with 0 to 2 constant functions in the set `{0,1}`)

3. Try out each candidate library if it can synthesize any of the well known universal functions NAND,NOR

4. for each library of size N that passes the universality test, ensure that it is minimal, i.e.

   (a) generate each of its sub-libraries of size N-1

   (b) discard the candidate library if any of its sub-libraries has been already found as being universal

   (c) otherwise accumulate the newly found universal library in the result set and use it to reject libraries having it as a sub-library later

The AutoLib Exact Synthesizer. The algorithms described in this section are planned to be the basis of the open source AutoLib Exact Synthesizer. A proof of concept implementation is available from `http://logic.csci.unt.edu/tarau/research/2007/autolib.zip`.

The tool accepts two input syntaxes:

```
?-syn([nand,nor],[],[A^B,A*B]).

?-syn([nand,nor],[],2:[6,1]).
```

The first two arguments of the query are the list of library operators and the list of constants. The third argument is either a symbolic expression or a descriptor of the form `NV:POs` where `NV` indicates the number of primary input variables and `POs` is a list of specifications of the primary outputs, encoded as bitstrings representing the value column in their truth table. Output is simply a printout of the truth table specifications followed by a symbolic expression describing the list of input variables, the list of gates, and the list of output variables paired with the integer specification of their truth table value columns.

```
  A^B       A*B       [A,B] : [
[0,0]:0   [0,0]:0         nand(A,B,C),
[0,1]:1   [0,1]:0          nor(A,B,D),
[1,0]:1   [1,0]:0          nor(D,C,E),
[1,1]:0   [1,1]:1          nor(E,D,F)
                      ] = [F,E]:[6,1].
```

Delay-Constrained Minimal Circuit Synthesis. Given the uniform gate structure of the circuits, we can ensure that delays are within acceptable margins by simply constraining the maximum length of the longest path from the primary inputs to the primary outputs.

Fanout restrictions. Given that fanout restrictions can add only constant depth and size increases [10] the tool currently focuses on exact synthesis assuming unbound fanout and binary operator libraries.

Boolean Operations with Bitstring Truth Table Encodings. We have used an adaptation of the efficient bitstring-integer encoding described in the Boolean Evaluation section of [11] of n variables as bitstring representations of truth table columns. Let x_k be a variable for $0 \leq k < n$. Then $x_k = (2^{2^n} - 1)/(2^{2^{n-k-1}} + 1)$, where the number of distinct variables in a boolean expression, n, is the number of bits for the encoding. Variables representing such bitstring-truth tables can be combined with the usual bitwise integer operators, to obtain new bitstring truth tables, encoding all possible value combinations of their arguments. For instance, if $n = 2$, the formula computes $x_0 = 3 = [0, 0, 1, 1]$ and $x_1 = 5 = [0, 1, 0, 1]$.

3. COMPARING THE EXPRESSIVENESS OF UNIVERSAL BOOLEAN FUNCTION LIBRARIES

DEFINITION 1. *A set of boolean functions F is universal if any boolean function can be written as a composition of functions in F.*

A well known universal set is (conjunction, negation) i.e. $(*, \sim)$ - this follows immediately from the rewriting of a truth table in terms of conjunction, disjunction and negation followed by elimination of disjunctions using De Morgan's laws. Universality of a library is usually proven by expressing, with primitives in the library, conjunction and negation or universal single operators like NAND,NOR.

3.1 Automated Induction of Minimal Universal Libraries

The table 2 lists the complete set of 40 minimal universal libraries induced by our synthesizer from the 16 binary operators listed in table 1 together with constant functions `{0,1}`.

zero	0	*	1	>	2	head	3
<	4	tail	5	^	6	+	7
nor	8	=	9	ntail	10	<=	11
nhead	12	=>	13	nand	14	one	15

Figure 1: The 16 Operators and their Truth Tables

nand	nor	<,1	>,1
=>,0	<=,0	<,=>	<,<=
<,=	<,nhead	<,ntail	<,one
>,=>	>,<=	>,=	>,nhead
>,ntail	>,one	=>,^	=>,nhead
=>,ntail	=>,zero	<=,^	<=,nhead
<=,ntail	<=,zero	*,=,0	*,^,1
*,nhead	*,ntail	+,=,0	+,^,1
+,nhead	+,ntail	*,=,^	*,=,zero
*,^,one	+,=,^	+,=,zero	+,^,one

Figure 2: The 40 Minimal Universal Libraries

It is known that minimal universal libraries differ up to (multiplicative) constant factors with respect to the number of gates needed to express a given circuit. While, in general, minimality is not preserved by rewriting a given minimal representation in terms of a different minimal library, upper bounds (useful in limiting the search space) can be generated by using known minimal representations in terms of an alternate library.

3.2 Measuring Expressiveness as Performance on Exact Synthesis Tasks

The table in Fig. 3 compares a few libraries (with obvious equivalences removed) used in synthesis with respect to the total gates needed to express all the 16 2-argument boolean operations (themselves included). The table in Fig. 4 provides the same data for a few selected non-minimal universal libraries.

This comparison provides our first indicator for the relative expressiveness of libraries.

By including operations like "$\oplus$" and "=", that are known to require a relatively high number of other gates (or a high transistor count) to express, one can minimize the number of operators (and circuit size) required. Using only gates known to have low transistor-count implementations like `nand` and `nor`, the expressiveness drops significantly (36 required). Surprisingly, $(\Rightarrow, 0)$ and its dual $(<, 1)$ do clearly

Library	Total	Library	Total	Library	Total
*,=,0	23	+,^,1	23	<,=>	24
*,^,1	25	+,=,0	25	*,=,^	26
+,=,^	26	<,=	28	=>,^	28
<,1	28	=>,0	28	<,nhead	30
=>,nhead	30	nand	36	nor	36

Figure 3: Total gates for minimal libraries

Library	Total	Library	Total
<,=>,0,1	20	>,<=,0,1	23
*,=,0,1	22	*,^,0,1	24
nand,nor,0,1	24	nand,nor	28
nand,0	32	nand,1	32
nor,0	32	nor,1	32

Figure 4: Total gates for some interesting non-minimal libraries

better than nand and nor: they can express all 16 operators with only 28 gates. As section 4.2.1 will show, they turn out to also have low transistor count implementations.

Interestingly enough, the libraries like $(*, =, 0)$ that provide, arguably, some the most human readable expressions when expressing other operators, has a relatively small gate count, 23. The same applies to $(*, \oplus, 0)$ known to provide a boolean ring structure.

Note also, that besides spotting out the most expressive 1-operator minimal universal libraries $(<, 1)$ and $(\Rightarrow, 0)$, the comparison also identifies $(<, \Rightarrow)$ as highly expressive two operator library (26 gates), with potential for practical design uses, given that $<$ and $\Rightarrow$ have both low transistor-count implementations (see section 4.2.1).

Finally, the overall "winner" of the comparison, expressing the 16 operators with only 20 gates is the library $<, \Rightarrow, 0, 1$. Given that both of its operators have small transistor count implementations (see section 4.2.1) this turns out to be an unexpectedly practical overall winner. Note also that $<$ and $\Rightarrow$ are dual operators - which makes symbolic reasoning on their properties easier.

3.3 Measuring Expressiveness through Search Space Covering with a Given Number of Gates

A dual method for evaluating expressiveness is to count the total number of distinct truth tables covered using a given number of gates. One can observe that this can be easily implemented by reusing the synthesizer's circuit generator to enumerate and count all possible outputs of circuits of a given size.

DEFINITION 2. *We call K-gate covering of N-variable functions with library L, the total number of distinct results obtained by evaluating all single output N-variable circuits containing up to K-gates from L.*

The tables in figures 5 to 10 show coverings for 2-4 variable functions using 2-5 gates, as well as the percentage of the search space covered (2^{2^N} distinct values for N variables).

Figures 11 and 12 visualize the coverings for 5-gate functions from various libraries on 3 and 4 variable truth tables.

Library	Variables	Gates	Covered	% covered
nor	2	2	9	56
nand	2	2	9	56
nor,0	2	2	10	62
nand,1	2	2	10	62
<,nhead	2	2	12	75
=>,0	2	2	12	75
<,1	2	2	12	75
nand,nor	2	2	14	87
*,^,1	2	2	12	75
<,=>	2	2	14	87
=>,^	2	2	12	75
<,=	2	2	12	75

Figure 5: Coverings for 2-variables, 2 gates

Library	Variables	Gates	Covered	% covered
nor	3	4	91	35
nand	3	4	91	35
nor,0	3	4	91	35
nand,1	3	4	91	35
<,nhead	3	4	104	40
=>,0	3	4	104	40
<,1	3	4	104	40
nand,nor	3	4	132	51
*,^,1	3	4	170	66
<,=>	3	4	204	79
=>,^	3	4	244	95
<,=	3	4	244	95

Figure 6: Coverings for 3-variables, 4 gates

Library	Variables	Gates	Covered	% covered
nor	3	5	139	54
nand	3	5	139	54
nor,0	3	5	139	54
nand,1	3	5	139	54
<,nhead	3	5	156	60
=>,0	3	5	156	60
<,1	3	5	156	60
nand,nor	3	5	211	82
*,^,1	3	5	238	92
<,=>	3	5	236	92
=>,^	3	5	256	100
<,=	3	5	256	100

Figure 7: Coverings for 3-variables, 5 gates

Library	Variables	Gates	Covered	% covered
nor	4	3	143	0
nand	4	3	143	0
nor,0	4	3	143	0
nand,1	4	3	143	0
<,nhead	4	3	258	0
=>,0	4	3	258	0
<,1	4	3	258	0
nand,nor	4	3	366	0
*,^,1	4	3	294	0
<,=>	4	3	810	1
=>,^	4	3	749	1
<,=	4	3	749	1

Figure 8: Coverings for 4-variables, 3 gates

Library	Variables	Gates	Covered	% covered
nor	4	4	436	0
nand	4	4	436	0
nor,0	4	4	436	0
nand,1	4	4	436	0
<,nhead	4	4	671	1
=>,0	4	4	671	1
<,1	4	4	671	1
nand,nor	4	4	1142	1
*,^,1	4	4	1126	1
<,=>	4	4	2286	3
=>,^	4	4	3760	5
<,=	4	4	3760	5

Figure 9: Coverings for 4-variables, 4 gates

Library	Variables	Gates	Covered	% covered
nor	4	5	1243	1
nand	4	5	1243	1
nor,0	4	5	1243	1
nand,1	4	5	1243	1
<,nhead	4	5	1616	2
=>,0	4	5	1616	2
<,1	4	5	1616	2
nand,nor	4	5	3394	5
*,^,1	4	5	4265	6
<,=>	4	5	7166	10
=>,^	4	5	16654	25
<,=	4	5	16654	25

Figure 10: Coverings for 4-variables, 5 gates

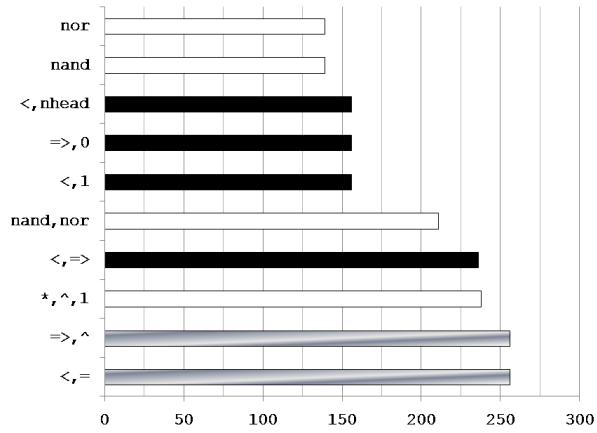

Figure 11: 5-gate coverings of the 2^{2^3} 3-variable functions (more = better)

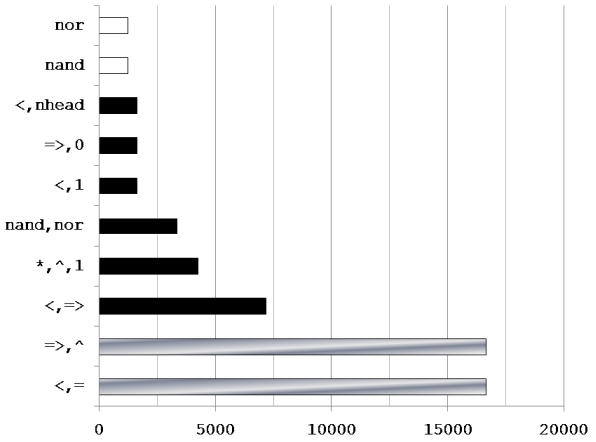

Figure 12: 5-gate coverings of the 2^{2^4} 4-variable functions (more = better)

Interestingly, one can see that while these results typically parallel those described in subsection 3.2, an amplifying effect can be observed, especially in the case of the larger truth tables in Fig. 12. While this time the "winners" are mixed libraries containing one asymmetrical operator and one in the $\oplus, =$ family, the gap between $(nand, nor)$ and $(<, \Rightarrow)$ shows that the expressiveness gap is likely to favor circuits built exclusively or containing asymmetrical operators. This observation is the main motivation of the next section that investigates some properties of $<$ and $\Rightarrow$ based libraries, relevant for synthesis tasks.

4. USING ASYMMETRICAL OPERATORS FOR COMBINATIONAL CIRCUIT SYNTHESIS

Surprisingly, Half XOR $(<)$ has been neglected by logicians, complexity theorists and circuit designers, to the point where there are relatively few references to it in the literature. To some extent, the same is true in the field of circuit design about its *dual*, $(\Rightarrow, 0)$, Logical Implication, which, on the other hand, has been extensively studied as an axiomatic basis for both classical and intuitionistic propositional logic.

4.1 Some Minimal Representations with Asymmetric Operators

Figure 13 shows minimal $(<, 1)$-representations for 0, negation, some 2-input boolean functions and the 3-argument IF-THEN-ELSE (ITE), as produced by our synthesizer.

Function	" $<$ " *Representation*
0	$1 < 1$
$\sim A$	$A < 1$
$A * B$	$(A < 1) < B$
$A + B$	$(A < (B < 1)) < 1$
$A \Rightarrow B$	$(B < A) < 1$
$A \Leftarrow B$	$(A < B) < 1$
$A \oplus B$	$((A < B) < ((B < A) < 1)) < 1$
$A = B$	$(A < B) < ((B < A) < 1)$
A NAND B	$((A < 1) < B) < 1$
A NOR B	$A < (B < 1)$
ITE A B C	$(A < (C < 1)) < ((B < A) < 1)$

Figure 13: $(<, 1)$-Representations

Figure 14, shows minimal $(\Rightarrow, 0)$-representations for 1, negation, some 2-input boolean functions and the 3-argument ITE, as produced by our synthesizer.

As expected, the library $(<, \Rightarrow, 0, 1)$ provides more elegant representations, especially for larger circuits (Fig. 15).

Given that $(A < 1)$ is equivalent to $(A \Rightarrow 0)$, the constant 1 can be dropped from the library without reducing expressiveness. Note that while the two constants could be dropped as $(<, \Rightarrow)$ forms a minimal universal library, in practice, their use provides not only better circuits but also simpler routing constraints as they can be implemented at 0-cost as connections to VSS and VDD. Moreover, one can keep in mind that in practice, their occurrences in a synthesized minimal circuit can always be replaced with a 2-transistor inverter as both $A \Rightarrow 0$ and $A < 1$ are equivalent to logical negation. For this reason, our synthesis algorithm will try to use the constant functions, when available, instead of primary input variables.

Function	" $\Rightarrow$ " *Representation*
1	$0 \Rightarrow 0$
$\sim A$	$A \Rightarrow 0$
$A + B$	$(A \Rightarrow 0) \Rightarrow B$
$A * B$	$(A \Rightarrow (B \Rightarrow 0)) \Rightarrow 0$
$A < B$	$(B \Rightarrow A) \Rightarrow 0$
$A > B$	$(A \Rightarrow B) \Rightarrow 0$
$A = B$	$((A \Rightarrow B) \Rightarrow ((B \Rightarrow A) \Rightarrow 0)) \Rightarrow 0$
$A \oplus B$	$(A \Rightarrow B) \Rightarrow ((B \Rightarrow A) \Rightarrow 0)$
A NOR B	$((A \Rightarrow 0) \Rightarrow B) \Rightarrow 0$
A NAND B	$A \Rightarrow (B \Rightarrow 0)$
ITE A B C	$(A \Rightarrow (B \Rightarrow 0)) \Rightarrow ((C \Rightarrow B) \Rightarrow 0)$

Figure 14: $(\Rightarrow, 0)$-Representations

$A + B$	$(A \Rightarrow 0) \Rightarrow B$
$A * B$	$((A \Rightarrow 0) < B)$
$A \oplus B$	$((A \Rightarrow B) \Rightarrow (A < B))$
$A * B * C$	$(B \Rightarrow (A \Rightarrow 0)) < C$
$A + B + C$	$(B < (A \Rightarrow 0)) \Rightarrow C$
ITE A B C	$((A \Rightarrow (B \Rightarrow 0)) \Rightarrow (B < C)$
$A * B * C * D$	$(C \Rightarrow (B \Rightarrow (A \Rightarrow 0))) < D$
$A + B + C + D$	$(C < (B < (A \Rightarrow 0))) \Rightarrow D$

Figure 15: $(<, \Rightarrow, 0, 1)$-Representations

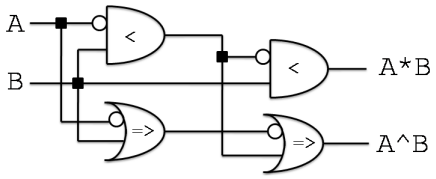

Figure 16: Half Adder synthesized with $(<, \Rightarrow)$

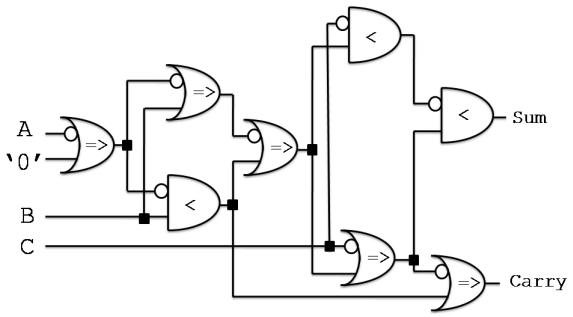

Figure 17: Full Adder synthesized with $(<, \Rightarrow, 0, 1)$

Figure 16 shows a Half Adder synthesized using library $(<, \Rightarrow)$ and figure 17 shows a Full Adder synthesized with library $(<, \Rightarrow, 0, 1)$. As an example of practical expressiveness of asymmetric operators, note that the full adder in Fig. 17, $[A * B + B * C + C * A, A \oplus B \oplus C]$ is built using only 8 gates from the library $(<, \Rightarrow, 0)$ which is the original gate count + 1, using $(+, *, \oplus)$.

4.2 Synthesis from CNF, DNF and NNF forms

As Disjunctive Normal Forms (DNF, also called sum-of-products), Conjunctive Normal Forms (CNF , also called product-of-sums) and Negation Normal Forms (NNF) are the result of repeated conjunctions and disjunctions (except for negation at their leaf nodes), we first focus on optimal $(<, \Rightarrow, 0, 1)$-representations of these. The following propositions are proved by induction on the number of primary input variables.

PROPOSITION 1. *A sequence of disjunctions of N variables has a minimal $(<, 1)$-representation with 2 occurrences of constant 1 and exactly one occurrence of each input variable, provided by the formula:*
$A_1 + A_2 + \cdots + A_N = (A_1 < (A_2 < \ldots (A_N < 1) \ldots)) < 1$

PROPOSITION 2. *A sequence of conjunctions of N variables has a minimal $(<, 1)$-representation with $N-1$ occurrences of constant 1 and exactly one occurrence of each input variable, provided by the formula:*
$A_1 * A_2 * \ldots A_{N-1} * A_N = ((A_1 < 1) < ((A_2 < 1) < \ldots ((A_{N-1} < 1) < A_N) \ldots)$

An optimal $(\Rightarrow, 0)$-representation of conjunctions and disjunctions is provided as follows.

PROPOSITION 3. *A sequence of conjunctions of N variables has a minimal $(\Rightarrow, 0)$-representation with 2 occurrences of constant 0 and exactly one occurrence of each input variable, provided by the formula:*
$A_1 * A_2 * \cdots * A_N = (A_1 \Rightarrow (A_2 \Rightarrow \ldots (A_N \Rightarrow 0) \ldots)) \Rightarrow 0$

PROPOSITION 4. *A sequence of disjunctions of N variables has a minimal $(\Rightarrow, 0)$-representation with $N-1$ occurrences of constant 0 and exactly one occurrence of each input variable, provided by the formula:*
$A_1 + A_2 + \ldots A_{N-1} + A_N = ((A_1 \Rightarrow 0) \Rightarrow ((A_2 \Rightarrow 0) \Rightarrow \ldots ((A_{N-1} \Rightarrow 0) \Rightarrow A_N) \ldots)$

Synthesis from CNF and DNF formulae (that can be obtained directly from truth table descriptions of circuits) proceeds by applying the encodings provided by the previous propositions recursively, followed by (and interleaved with) simplification steps.

Negation normal forms (NNF) representations can benefit from combining the smaller (N+1 gates) representations for conjunctions using $\Rightarrow$ with the smaller (N+1 gates) representation for disjunctions using $<$. This property provides an intuitive explanation for the expressiveness of libraries based on $(<, \Rightarrow)$.

Note that to avoid the large delays induced by the linear chains of operators, balanced trees can be used instead without changing the number of gates significantly.

PROPOSITION 5. *A formula in NNF form is convertible to a $(<, \Rightarrow)$ equivalent of the same size up to a constant factor. Balancing a $(<, \Rightarrow)$-formula yields and equivalent NNF-formula of the same size up to a constant factor.*

The proposition follows from the fact (Props. 1, 3) that disjunctions and conjunctions can be rewritten with $(<, \Rightarrow)$-expressions that do not duplicate variable occurrences.

4.2.1 Transistor Implementations for $(<, \Rightarrow)$-circuits

Clearly as $A < B$ is equivalent to $nor(A, \sim B)$, an obvious 6-transistor implementation is obtained when input B drives a 2-transistor inverter while its output and input A drive a 4-transistor NOR gate.

This logic circuit is shown in Fig. 18. The output node, $A < B$, has a direct path to the power nodes VDD and VSS through the source connections of the transistors connected to it. As a result, the output is called "buffered" and the logic circuit type is "powered".

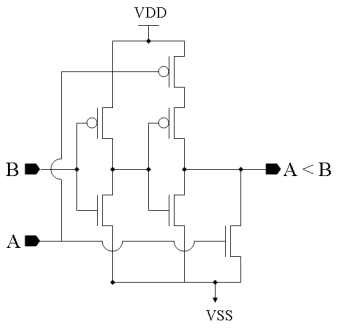

Figure 18: Powered 6-Transistor $A < B$

Semi-Powered 4-Transistor

A	B	A < B
0	0	0
0	1	unbuffered '1'
1	0	0
1	1	0

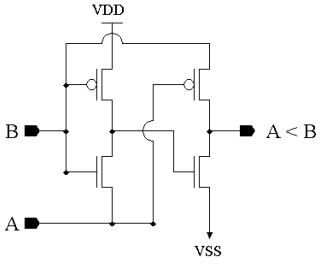

Figure 19: Semi-Powered 4-Trans. $A < B$

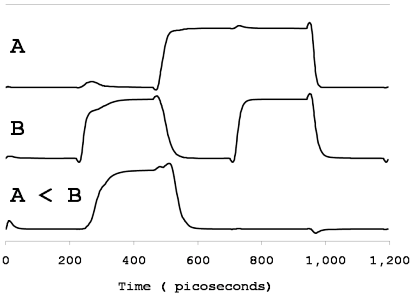

Figure 20: Spice Simulation of 4-Trans $A < B$

Semi-Powered 4-Transistor

A	B	A => B
0	0	1
0	1	1
1	0	unbuffered '0'
1	1	1

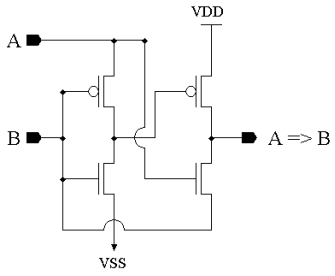

Figure 21: Semi-Powered 4-Trans. $A \Rightarrow B$

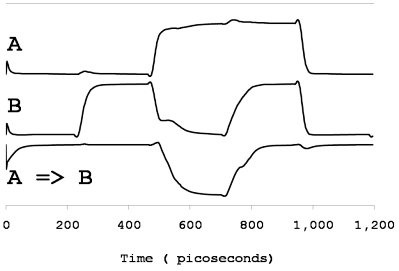

Figure 22: Spice Simulation of 4-Trans $A \Rightarrow B$

To reduce transistor count, a *pass transistor logic* (PTL) circuit for $A < B$ can be implemented using 4 transistors. In this circuit, the output node, $A < B$, in Fig. 19 has a direct path to the power net VSS while input B provides the VDD power. Therefore, the logic circuit type is "semi-powered" and the output level for VDD is called "unbuffered".

Dually, a 4-transistor PTL circuit for $A \Rightarrow B$ is implemented as shown in Fig. 21.

Using 50nm CMOS models [1] and VDD = 1 Volt, SPICE netlists were simulated for both the 4 transistor $A < B$ circuit (Fig. 20) and the 4-transistor $\Rightarrow$ circuit (Fig. 22), showing that they are functionally correct. With both inputs buffered and the output loaded with 4fF, the maximum input-to-output 0.5V/0.5V propagation delay was 48ps.

The constant function 1 can be implemented by direct routing to the VDD power grid. Similarly, the constant function 0 can be implemented by direct routing to the VSS power grid. Buffering of the unbuffered signals can be handled by adding to the synthesis algorithm an additional constraint to force alternation of the $<$ and $\Rightarrow$ gates.

In conclusion, assuming a design using PTL-logic, the transistor count for an implementation of the $<$ and $\Rightarrow$ functions is 4*, while constant functions* 1 *and* 0 *are essentially free, with transistor count 0.*

5. CONCLUSION

We have described two quantitative methods for measuring the relative expressiveness of boolean function libraries using exact synthesis. Both indicators suggest that libraries based on asymmetrical operators $(<, \Rightarrow)$ are a practical alternative to NAND and NOR-based libraries. Their relative expressiveness challenges, to some extent, the widely believed statement [6, 3] that symmetric functions are genuinely more interesting for circuit synthesis. While we have provided low cost transistor models for $<$ and $\Rightarrow$ gates and tested their signal correctness with SPICE, the validation of their use in various context requires more extensive SPICE simulations as well as precise area, delay and power estimates.

On the general synthesis algorithm side, we plan to add tabling of sub-circuits to avoid recomputation. It has been pointed out in recent papers like [9, 7] that SAT-solver and circuit synthesis algorithms are synergistically related. Adapting intelligent backtracking mechanisms like those used in modern SAT-solvers will be implemented to improve performance. Using properties like NPN-equivalence [32, 19, 15] will provide library specific constraints likely to speed up search.

As $(\Rightarrow, 0)$ has been used as a foundation of various *implicative* formalizations of classic and intuitionistic logics, we plan to use the powerful rewriting mechanisms available for it (that can be transposed to $(<, 1)$ using duality) to extend exact synthesis with symbolic rewriting based heuristics.

Given that $A < B$ and $A \Rightarrow B$ are order relations, suggests their use in novel analog or non-silicon designs, provided that one can measure that signal A is in a given sense weaker than B.

Polymorphic or multi-functional [24] NAND/NOR gates have been recently synthesised at transistor level [22, 27]. Given the significantly higher expressiveness of the library $(<, \Rightarrow)$ we plan to try out a similar experiment using it.

Given that exact synthesis of reversible circuits using Fredkin and Toffoli gates [13, 14, 9] is important for future quan-

tum computing and adiabatic computing research, we plan to extend the synthesizer to support such libraries optimally.

6. REFERENCES

[1] Baker, R. J. *CMOS Circuit Design, Layout and Simulation*. John Wiley and Sons, Hoboken, NJ, USA, 2008.

[2] Bryant, R. E. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers 35*, 8 (1986), 677–691.

[3] Canteaut, A., and Videau, M. Symmetric Boolean Functions. *IEEE Transactions on Information Theory 51*, 8 (2005), 2791–2811.

[4] Culliney, J. N., Young, M. H., Nakagawa, T., and Muroga, S. Results of the Synthesis of Optimal Networks of AND and OR Gates for Four-Variable Switching Functions. *IEEE Trans. Computers 28*, 1 (1979), 76–85.

[5] Davies, D. W. Switching functions of three variables. *Trans. Inst. Radio Engineers 6*, 4 (1957), 265–275.

[6] Dietmeyer, D. L. *Logic Design of Digital Systems*. Allyn and Bacon, 1971.

[7] Drechsler, R. Using Synthesis Techniques in SAT Solvers. citeseer.ist.psu.edu/drechsler04using.html.

[8] Drechsler, R., and Gunther, W. Exact Circuit Synthesis. In *International Workshop on Logic Synthesis* (1998).

[9] Grosse, D., Chen, X., Dueck, G. W., and Drechsler, R. Exact sat-based toffoli network synthesis. In *ACM Great Lakes Symposium on VLSI* (2007), H. Zhou, E. Macii, Z. Yan, and Y. Massoud, Eds., ACM, pp. 96–101.

[10] Hoover, H. J., Klawe, M. M., and Pippenger, N. Bounding Fan-out in Logical Networks. *J. ACM 31*, 1 (1984), 13–18.

[11] Knuth, D. The Art of Computer Programming, Volume 4, draft, 2006. http://www-cs-faculty.stanford.edu/∼/knuth/taocp.html.

[12] Lai, H. C., and Muroga, S. Logic Networks with a Minimum Number of NOR(NAND) Gates for Parity Functions of Variables. *IEEE Trans. Computers 36*, 2 (1987), 157–166.

[13] Maslov, D., Dueck, G. W., and Miller, D. M. Synthesis of Fredkin-Toffoli reversible networks. *IEEE Trans. VLSI Syst. 13*, 6 (2005), 765–769.

[14] Maslov, D., Dueck, G. W., and Miller, D. M. Techniques for the synthesis of reversible Toffoli networks. *ACM Trans. Design Autom. Electr. Syst. 12*, 4 (2007).

[15] Meyer, J., and Kocan, F. Sharing of SRAM Tables Among NPN-Equivalent LUTs in SRAM-Based FPGAs. *IEEE Trans. VLSI Syst. 15*, 2 (2007), 182–195.

[16] Mira, J., and Álvarez, J. R., Eds. *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach: First International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2005, Las Palmas, Canary Islands, Spain, June 15-18, 2005, Proceedings, Part II* (2005), vol. 3562 of *Lecture Notes in Computer Science*, Springer.

[17] Mishchenko, A., and Brayton, R. A boolean paradigm for multivalued logic synthesis. In *Proc. IgVLS'02, June, 2002, pp. 173-177.* (2002).

[18] Mishchenko, A., and Sasao, T. Encoding of Boolean functions and its application to LUT cascade synthesis. In *International Workshop on Logic Synthesis* (2002).

[19] Muroga, S., and Lai, H. C. Minimization of Logic Networks Under a Generalized Cost Function. *IEEE Trans. Computers 25*, 9 (1976), 893–907.

[20] Oettinger, A. G., and Aiken, H. H. Retiring computer pioneer. *Commun. ACM 5*, 6 (1962), 298–299.

[21] Riedel, M. Cyclic Combinational Circuits. In *Ph.D. Dissertation, Caltech.* (2004).

[22] Sekanina, L. Evolutionary design of gate-level polymorphic digital circuits. In *EvoWorkshops* (2005), F. Rothlauf, J. Branke, S. Cagnoni, D. W. Corne, R. Drechsler, Y. Jin, P. Machado, E. Marchiori, J. Romero, G. D. Smith, and G. Squillero, Eds., vol. 3449 of *Lecture Notes in Computer Science*, Springer, pp. 185–194.

[23] Sekanina, L. Evolutionary design of digital circuits: Where are current limits? In Stoica et al. [27], pp. 171–178.

[24] Sekanina, L., Starecek, L., Gajda, Z., and Kotásek, Z. Evolution of multifunctional combinational modules controlled by the power supply voltage. In Stoica et al. [27], pp. 186–193.

[25] Shannon, C. E. *Claude Elwood Shannon: collected papers*. IEEE Press, Piscataway, NJ, USA, 1993.

[26] Shende, V., Prasad, A., Markov, I., and Hayes, J. Synthesis of reversible logic circuits. In *IEEE Trans. on CAD 22, pp. 710–722* (June 2003).

[27] Stoica, A., Arslan, T., Suess, M., Yalçin, S., Keymeulen, D., Higuchi, T., Zebulum, R. S., and Aydin, N., Eds. *First NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2006), 15-18 June 2006, Istanbul, Turkey* (2006), IEEE Computer Society.

[28] Stoica, A., Zebulum, R. S., Keymeulen, D., and Daud, T. Transistor-level circuit experiments using evolvable hardware. In Mira and Álvarez [16], pp. 366–375.

[29] Stoica, A., Zebulum, R. S., Keymeulen, D., Ramesham, R., Neff, J., and Katkoori, S. Temperature-adaptive circuits on reconfigurable analog arrays. In Stoica et al. [27], pp. 28–31.

[30] Tarau, P., and Luderman, B. A Logic Programming Framework for Combinational Circuit Synthesis. In *23rd International Conference on Logic Programming (ICLP), LNCS 4670* (Porto, Portugal, Sept. 2007), Springer, pp. 180–194.

[31] Tarau, P., and Luderman, B. Revisiting Exact Combinational Circuit Synthesis. In *Proceedings of ACM SAC'08* (Fortalezza, Brazil, Mar. 2008). to appear.

[32] Young, M. H., and Muroga, S. Symmetric Minimal Covering Problem and Minimal PLA's with Symmetric Variables. *IEEE Trans. Computers 34*, 6 (1985), 523–541.