



Optimization of NULL convention self-timed circuits

S.C. Smith^{a,*}, R.F. DeMara^b, J.S. Yuan^b, D. Ferguson^c, D. Lamb^c

^a *Department of Electrical and Computer Engineering, University of Missouri–Rolla, 133 Emerson Electric Co. Hall, 1870 Miner Circle, Rolla, MO 65409, USA*

^b *School of Electrical Engineering and Computer Science, University of Central Florida, Box 162450, Orlando, FL 32816-2450, USA*

^c *Theseus Logic, Inc., 485 North Keller Road, Suite 140, Maitland, FL 32751, USA*

Received 1 September 2001; accepted 8 December 2003

Abstract

Self-timed logic design methods are developed using *Threshold Combinational Reduction (TCR)* within the NULL Convention Logic (NCL) paradigm. NCL logic functions are realized using 27 distinct transistor networks implementing the set of all functions of four or fewer variables, thus facilitating a variety of gate-level optimizations. TCR optimizations are formalized for NCL and then assessed by comparing levels of gate delays, gate counts, transistor counts, and power utilization of the resulting designs. The methods are illustrated to produce (1) fundamental logic functions that are 2.2–2.3 times faster and require 40–45% fewer transistors than conventional canonical designs, (2) a Full Adder with reduced critical path delay and transistor count over various alternative gate-level synthesis approaches, resulting in a circuit with at least 48% fewer transistors, half as many gate delays to generate the *carry* output, and the same number of gate delays to generate the *sum* output, as its nearest competitors, and (3) time, space, and power optimized increment circuits for a 4-bit up-counter, resulting in a throughput-optimized design that is 14% and 82% faster than area- and power-optimized designs, respectively, an area-optimized design that requires 22% and 42% fewer transistors than the speed- and power-optimized designs, respectively, and a power-optimized design that dissipates 63% and 42% less power than the speed- and area-optimized designs, respectively. Results demonstrate support for a variety of optimizations utilizing conventional Boolean minimization followed by table-driven gate substitutions, providing for an NCL design method that is readily automatable.

© 2004 Elsevier B.V. All rights reserved.

Keywords: Asynchronous logic design; Self-timed circuits; Dual-rail encoding; Quad-rail encoding; Threshold gates; Full adder; Up-counter; NULL convention logic (NCL)

*Corresponding author. Tel.: +1-573-341-4232; fax: +1-573-341-4532.

E-mail addresses: smithsco@umr.edu (S.C. Smith), demara@mail.ucf.edu (R.F. DeMara), dlamb@theseus.com (D. Lamb).

1. Introduction

NULL Convention Logic (NCL) [1] offers a self-timed logic paradigm where control is inherent with each datum. NCL follows the so-called “weak conditions” of Seitz’s delay-insensitive signaling scheme [2]. As with other self-timed logic methods discussed herein, the NCL paradigm assumes that forks in wires within basic components are isochronic [3,4]. The origins of various aspects of the paradigm, including the NULL (or spacer) logic state from which NCL derives its name, can be traced back to Muller’s work on speed-independent circuits in the 1950s and 1960s [5].

Earlier work by Seitz presents an extensive discussion of self-timed logic, illustrating its advantages over traditional clocked logic, including reduced power, noise, EMI, and easier component reuse, and includes one approach to designing such circuits [2]. Some other methods of designing self-timed circuits are detailed in [6–9]. These techniques concentrate on developing circuits from a standardized set of gates, while other techniques [10,11] emphasize formal logic methods that directly yield designs at the transistor-level. In the application of CMOS technology, processors implemented with this type of signaling scheme include the MIPS R3000 [12] and another at Caltech [13], the FLYSIG processor at the University of Paderborn [14], the MSL16A at the Chinese University of Hong Kong [15], and the TITAC processor at the Toyko Institute of Technology [16].

NCL differs from the above mentioned methods in that they only utilize one type of state-holding gate, the *C-element* [5]. On the other hand, all NCL gates are state-holding. Thus, NCL optimization methods can be considered as a subclass of the techniques for developing self-timed circuits using a pre-defined set of more complex components with built-in hysteresis behavior. In functions that do not require full minterm generation, such attributes may allow optimizations that produce smaller, faster self-timed combinational circuits. To demonstrate NCL’s viability, the NCL08 microcontroller as described in [17] requires 40% less power and generates 10 dB less peak EMI noise than its corresponding synchronous counterpart, the Motorola HCS08, while achieving similar performance. Within the NCL paradigm, this paper illustrates circuit minimization techniques to specifically target speed, area, or power, the application of these techniques to design various NCL circuits, and their associated tradeoffs.

1.1. Previous work

Asynchronous circuits fall into two main categories: *self-timed* and *bounded-delay* models. Paradigms, like NCL, assume delays in both logic elements and interconnects to be unbounded, although they assume that wire forks within basic components, such as a full adder, are isochronic [3,4]. Wires connecting components do not have to adhere to this assumption. This implies the ability to operate in the presence of indefinite arrival times for the reception of inputs. Completion detection of the output signals allows for handshaking to control input wavefronts. On the other hand, bounded-delay models such as *Huffman circuits* [18], *burst-mode circuits* [19], and *micropipelines* [20] assume that delays in both gates and wires are bounded. Delays are added based on worst-case scenarios to avoid hazard conditions. This leads to extensive timing analysis of worst-case behavior to ensure correct circuit operation. Since NCL exhibits neither of these characteristics, bounded-delay models are not addressed further in this paper.

Most self-timed methods combine *C-elements* [5] with Boolean gates for circuit construction. A C-element behaves as follows: when all inputs assume the same value then the output assumes this value, otherwise the output does not change. Seitz's method [2] employs a sum of products (SOP) network using AND and OR gates, combined with a network to OR both rails of all inputs together. The output of the OR network is then combined with the SOP outputs, using C-elements, to produce the circuit outputs. DIMS [9] and Anantharaman's approach [7] are similar to each other in that each produces an SOP circuit using OR gates and C-elements, instead of AND gates. Singh's method [8] combines small self-timed logic functions to produce the desired functionality. While David's method [6] produces self-timed circuits with n inputs and m outputs, composed of four subnets, *ORN*, *CEN*, *DRN*, and *OUTN*. *ORN* consists of n 2-input OR gates, which OR together both rails of each dual-rail input. *CEN* is an n -input C-structure, which is equivalent to an n -input C-element, whose inputs are the n outputs from *ORN*. *DRN* is a monotonic implementation of each rail of the dual-rail output(s). *OUTN* produces the circuit output and consists of $2m$ 2-input C-elements, each with the output of *CEN* as 1-input, and an output from *DRN* as the other input. Seitz's method, Anantharaman's approach, and DIMS require the generation of all minterms to implement a function, where a minterm is defined as the logical AND, or product, containing all input signals in either complemented or non-complemented form. While Singh's and David's methods do not require full minterm generation, they rely solely on C-elements for speed-independence.

Since Seitz's and Anantharaman's approaches, along with DIMS, require the generation of all minterms, no optimization is possible. However, Singh's and David's approaches allow for some Boolean optimization to be performed, but they may not facilitate the same potential for optimization provided by NCL's many state-holding gates, as will be shown in Section 4.

1.2. Overview of NCL

NCL threshold gates are a special case of the logical operators or gates available in digital VLSI circuit design [21]. Such an operator consists of a set condition and a reset condition, which the environment must ensure are not both satisfied at the same time. If neither condition is satisfied then the operator maintains its current state. A number of NCL-based designs have been commercially developed by Theseus Logic, Inc., which has formed strategic alliances with Motorola for microcontroller design and Synopsys for NCL-based design tool development.

NCL uses symbolic completeness of expression [1] to achieve self-timed behavior. A symbolically complete expression is defined as an expression that only depends on the relationships of the symbols present in the expression without a reference to the time of evaluation. In particular, dual-rail signals, quad-rail signals, or other *Mutually Exclusive Assertion Groups* (MEAGs) can be used to incorporate data and control information into one mixed signal path to eliminate time reference [22]. A dual-rail signal, D , consists of two wires, D^0 and D^1 , which may assume any value from the set {DATA0, DATA1, NULL}. The DATA0 state ($D^0 = 1, D^1 = 0$) corresponds to a Boolean logic 0, the DATA1 state ($D^0 = 0, D^1 = 1$) corresponds to a Boolean logic 1, and the NULL state ($D^0 = 0, D^1 = 0$) corresponds to the empty set meaning that the value of D is not yet available. The two rails are mutually exclusive, so that both rails can never be asserted simultaneously; this state is defined as an illegal state. A quad-rail signal, Q , consists of four wires, Q^0, Q^1, Q^2 , and Q^3 , which may assume any value from the set {DATA0, DATA1, DATA2,

DATA3, NULL}. The DATA0 state ($Q^0 = 1, Q^1 = 0, Q^2 = 0, Q^3 = 0$) corresponds to two Boolean logic signals, X and Y , where $X = 0$ and $Y = 0$. The DATA1 state ($Q^0 = 0, Q^1 = 1, Q^2 = 0, Q^3 = 0$) corresponds to $X = 0$ and $Y = 1$. The DATA2 state ($Q^0 = 0, Q^1 = 0, Q^2 = 1, Q^3 = 0$) corresponds to $X = 1$ and $Y = 0$. The DATA3 state ($Q^0 = 0, Q^1 = 0, Q^2 = 0, Q^3 = 1$) corresponds to $X = 1$ and $Y = 1$, and the NULL state ($Q^0 = 0, Q^1 = 0, Q^2 = 0, Q^3 = 0$) corresponds to the empty set meaning that the result is not yet available. The four rails of a quad-rail NCL signal are mutually exclusive, so no two rails can ever be asserted simultaneously; these states are defined as illegal states. Both dual- and quad-rail signals are space optimal 1-out-of- N delay-insensitive codes, requiring two wires per bit. Other higher order MEAGs are not typically wire count optimal, however they can be more power efficient due to the decreased number of transitions per cycle.

NCL uses *threshold gates with hysteresis* [23] for its composable logic elements. One type of threshold gate is the *TH m n gate*, where $1 \leq m \leq n$, as depicted in Fig. 1. A TH m n gate corresponds to an operator with at least m signals asserted as its set condition and all signals de-asserted as its reset condition. TH m n gates have n inputs. At least m of the n inputs must be asserted before the output will become asserted. Because threshold gates are designed with hysteresis, all asserted inputs must be de-asserted before the output will be de-asserted. Hysteresis is used to provide a means for monotonic transitions and a complete transition of multi-rail inputs back to a NULL state before asserting the output associated with the next wavefront of input data. In the symbol for a TH m n gate, each of the n inputs is connected to the rounded portion of the gate; the output emanates from the pointed end of the gate; and the gate's threshold value, m , is written inside of the gate. Fig. 2 shows a static CMOS implementation of a TH23 gate, with inputs A, B , and C , and output Z . Sobelman and Fant [23] detail various design implementations (static, semi-static, and dynamic) of TH m n gates.

Another type of threshold gate is referred to as a weighted threshold gate, denoted as TH m nW $w_1w_2\dots w_R$. Weighted threshold gates have an integer value, $m \geq w_R > 1$, applied to *input R* . Here $1 \leq R < n$; where n is the number of inputs; m is the gate's threshold; and w_1, w_2, \dots, w_R , each > 1 , are the integer weights of *input1, input2, ..., input R* , respectively. For example, consider a TH34W2 gate, whose $n = 4$ inputs are labeled A, B, C , and D . The weight of input A , $W(A)$, is therefore 2. Since the gate's threshold, m , is 3, this implies that in order for the output to be asserted, either inputs B, C , and D must all be asserted, or input A must be asserted and any other input, B, C , or D must also be asserted.

Inputs are partitioned into two separate wavefronts, the NULL wavefront and the DATA wavefront. The NULL wavefront consists of all inputs to a circuit being NULL, while the DATA wavefront refers to all inputs being DATA, some combination of DATA0 and DATA1. Initially all circuit elements are reset to the NULL state. First, a DATA wavefront is presented to the circuit. Once all of the outputs of the circuit transition to DATA, the NULL wavefront is

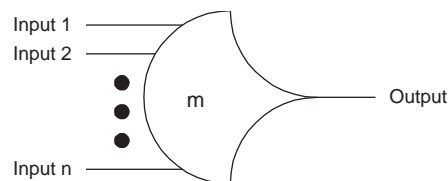


Fig. 1. TH m n threshold gate [1].

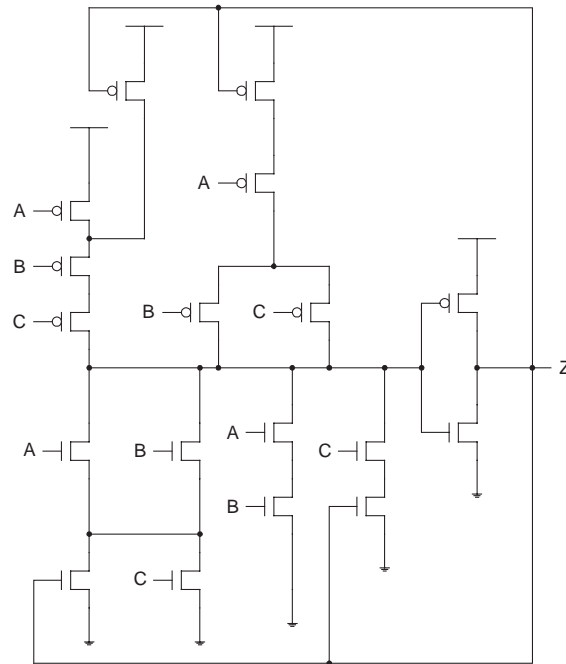


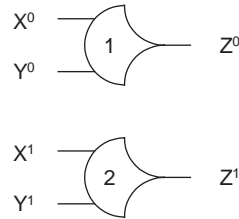
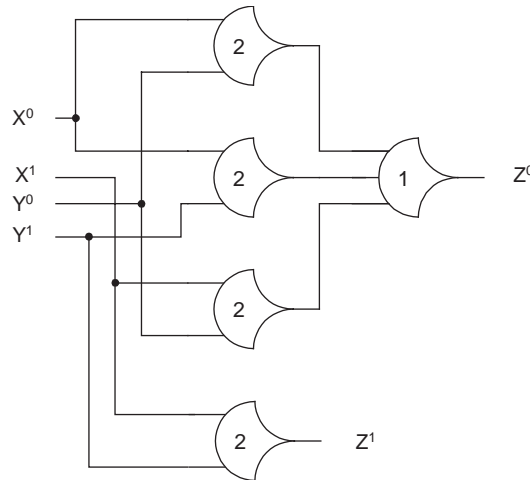
Fig. 2. Static CMOS implementation of a TH23 gate.

presented to the circuit. Once all of the outputs of the circuit transition to NULL, the next DATA wavefront is presented to the circuit. This DATA/NULL cycle continues repeatedly. As soon as all outputs of the circuit are DATA, the circuit's result is valid. The NULL wavefront then transitions all of these DATA outputs back to NULL. When they transition back to DATA again, the next output is available. This period is referred to as the DATA-to-DATA cycle time, denoted as T_{DD} and has an analogous role to the clock period in a synchronous circuit.

The *input-completeness* criterion [1], which NCL circuits must maintain in order to be self-timed, requires that:

1. the outputs of a circuit may not transition from NULL to DATA until all inputs have transitioned from NULL to DATA, and
2. the outputs of a circuit may not transition from DATA to NULL until all inputs have transitioned from DATA to NULL.

In circuits with multiple outputs, it is acceptable for some of the outputs to transition without having a complete input set present, as long as all outputs cannot transition before all inputs arrive. This signaling scheme is equivalent to the “weak conditions” of delay-insensitive signaling defined by Seitz [2]. Consider the input-incomplete NCL AND function shown in Fig. 3. The output can change from NULL to DATA0 without both inputs first transitioning to DATA. For instance, if $A = \text{DATA0}$ and $B = \text{NULL}$ then $Z = \text{DATA0}$, which breaks the completeness of input criterion. Fig. 4 shows an input-complete NCL AND function since the output cannot transition until both inputs have transitioned. Completeness of DATA can be ensured for an N input function as shown in Algorithm 1.

Fig. 3. Input-incomplete AND function: $Z = X \cdot Y$.Fig. 4. Conventional input-complete AND function: $Z = X \cdot Y$.

Algorithm 1. Input-completeness pseudocode.

```

for (i = 1 to N) loop
  INPUTi = NULL
  group INPUTSj (1 ≤ j ≤ N, j ≠ i) such that they form an N - 1 bit word called REMAINDER
  for (k = 0 to 2N-1 - 1) loop
    REMAINDER = k
    if (all output bits are DATA) then
      return (INCOMPLETE)
    end loop
  end loop
end loop
return (COMPLETE)

```

If a function is complete with respect to DATA, it is also complete with respect to NULL due to the hysteresis functionality of every NCL gate. This completeness check takes $O(N \cdot 2^{N-1})$; however, this is unnecessary for many functions due to their inherent input-completeness. For example, the XOR function, the full adder, and the increment circuitry, all are inherently complete such that it is impossible to know the output without all of the inputs being known.

There is one more condition that must be met in order for NCL to retain its self-timed nature. No *orphans* may propagate through a gate within a basic component. An orphan is defined as a wire that transitions during the current DATA wavefront, but is not used in the determination of the output. Orphans are caused by wire forks and can be neglected through the isochronic fork assumption [3,4], as long as they are not allowed to cross a gate boundary. This *observability* condition ensures that every gate transition is observable at the output. Consider an incorrect version of an XOR function shown in Fig. 5, where an orphan is allowed to pass through the TH12 gate. For instance, when $X = \text{DATA0}$ and $Y = \text{DATA0}$, the TH12 gate is asserted, but does not take part in the determination of the output, $Z = \text{DATA0}$. This orphan path is shown in boldface in Fig. 5. A correct, fully observable version of the XOR function is shown in Fig. 6, where no orphans propagate through any gate. Whereas previous work on optimization of circuits constructed from logical operators has concentrated on transistor-sizing [24] and decomposition of high fan-in operators [25], this paper will emphasize composable circuit construction utilizing a predefined set of complex state-holding gates.

1.3. Paper outline

This paper is organized into six sections. In Section 2, the *Threshold Combinational Reduction* (TCR) method for optimizing combinational NCL circuits is developed. The method is demonstrated in Sections 3–5. Section 3 presents optimal input-complete AND/NAND,

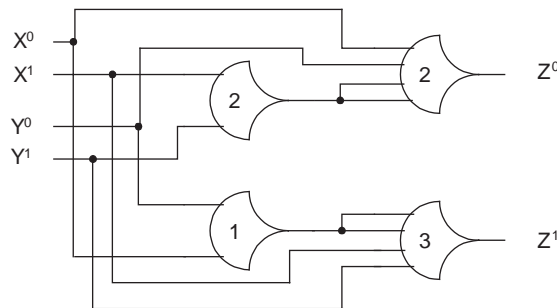


Fig. 5. Unobservable XOR function: $Z = X \oplus Y$ (orphans may propagate through a gate).

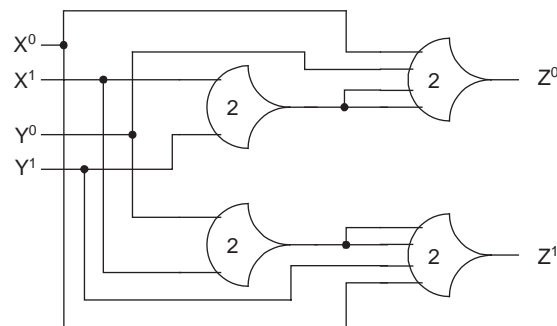


Fig. 6. Observable XOR function: $Z = X \oplus Y$ (orphans may not propagate through any gate).

OR/NOR, and XOR/NXOR logic functions, designed using TCR. Section 4 applies TCR to produce a self-timed Full Adder that significantly reduces critical path delay and transistor count over previous gate-level self-timed approaches. Section 5 illustrates the use of TCR to derive a variety of time, space, and power optimized NCL increment circuitries for an up-counter with a feedback circuit. Section 6 provides conclusions and outlines directions for future work.

2. TCR method definition

As depicted in Fig. 7, the design process begins with a specification of the circuit functional behavior and desired optimization criteria. Circuit behavior is specified as Boolean logic expressions, truth tables, and/or narrative descriptions. The optimization criteria include parameters such as critical path delay, area, or power consumption, that are to be minimized in the target design. Several alternate designs are generated which are then assessed against the optimization criteria, allowing for the preferred design to be selected for implementation. The TCR method is currently in the process of being automated through incorporation into the Mentor Graphics design tool suite.

First, a logic encoding scheme is selected such as dual-rail, quad-rail, or other MEAG representations, as depicted in Fig. 7. Typically either dual-rail or quad-rail is chosen since these encodings yield the minimum of two wires per bit. If a dual-rail encoding is used, the next step is to select the optimization space in which minimization will be performed. The proposed TCR design methods have been numbered “1”, “2”, and “3”, each with design steps labeled “A”, “B”, or “C”, appropriately.

2.1. Method 1: input-incomplete functions

As depicted in Fig. 7, Method 1 corresponds to Boolean optimization. Maximal use of input-incomplete NCL logic functions generates the individual outputs, while maintaining the completeness of input criterion for the circuit as a whole. For example, gates in Boolean designs that target the basic logical operators (AND, OR, XOR, NAND, NOR, NXOR, NOT) are directly mapped to an NCL design by using as many input-incomplete NCL functions as possible. As described in Step 1A of Fig. 7, each Boolean gate is replaced with its NCL equivalent function, using input-incomplete versions whenever possible. Step 1B ensures input-completeness for the circuit as a whole by employing input-complete functions only for selected gates in the data path, so that the computation of an entire output set implies that the complete input set has arrived.

2.2. Method 2: dual-rail optimizations

Method 2 is based on dual-rail optimization. In Step 2A, the NCL circuit is optimized by using reduced sum-of-product (SOP) expressions for both rails of the output. These expressions are then mapped to TH1n and THnn gates. As in Boolean circuits, a Karnaugh map can be constructed for each output. The $0s$ in the Karnaugh map refer to a signal's $rail^0$ line and the $1s$ refer to a signal's $rail^1$ line. Reduced SOP expressions for both the $1s$ and $0s$ in the Karnaugh map are derived. After these expressions for the outputs have been obtained, an assessment must be made to ensure that

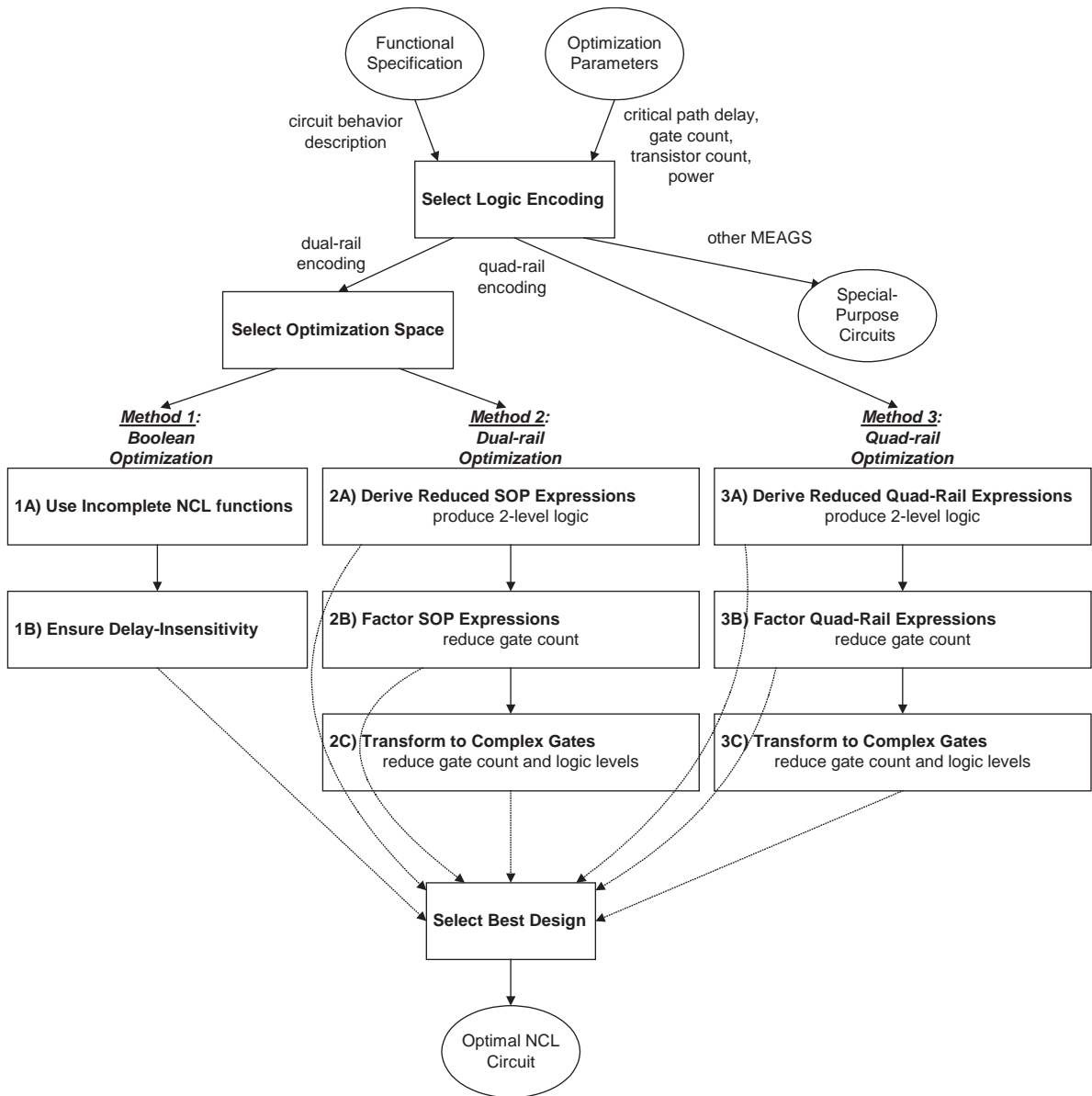


Fig. 7. TCR design flow.

the complete output set cannot be generated without all of the inputs being present. If under any timing scenario, a complete output set can be generated without all of the inputs being present, the missing logic terms must be added to the reduced expressions to guarantee that the completeness of input criterion holds. This method will always generate two-level logic, given that threshold gates with a sufficiently large number of inputs are available. The first level will consist of THnn gates, to produce the required product terms; and the second level will

consist of TH1n gates, which act to OR the product terms together to produce the desired outputs. Step 2A is similar to Anantharaman's approach [7] and DIMS [9]. In Step 2B, the common sub-expressions are factored and consolidated to reduce the gate count. Finally, the factored expressions for each rail are manipulated in Step 2C to obtain equations of the forms contained in Table 1. The observability criterion must be ensured for every circuit output from Steps 2A, 2B, and 2C.

Table 1 lists the 27 transistor networks, along with their corresponding Boolean equations, used to construct NCL circuits. These 27 transistor networks, implemented as macros, constitute the set of all functions consisting of four or fewer variables. Since each rail of an NCL signal is considered a separate variable, a four variable function is not the same as a function of four literals, which would normally consist of eight variables. Twenty-four of these macros can be realized using complex threshold gates, identical to the standard threshold gate forms for functions of four or fewer variables [26–28]. The other three macros could be constructed from threshold gate networks, but have been implemented as transistor networks to reduce area and delay. Table 1 also contains the transistor count for these 27 macros.

Table 1
27 NCL macros

NCL macro	Boolean function	Transistor count
TH12	$A+B$	6
TH22	AB	12
TH13	$A+B+C$	8
TH23	$AB+AC+BC$	18
TH33	ABC	16
TH23w2	$A+BC$	14
TH33w2	$AB+AC$	14
TH14	$A+B+C+D$	10
TH24	$AB+AC+AD+BC+BD+CD$	26
TH34	$ABC+ABD+ACD+BCD$	24
TH44	$ABCD$	20
TH24w2	$A+BC+BD+CD$	20
TH34w2	$AB+AC+AD+BCD$	22
TH44w2	$ABC+ABD+ACD$	23
TH34w3	$A+BCD$	18
TH44w3	$AB+AC+AD$	16
TH24w22	$A+B+CD$	16
TH34w22	$AB+AC+AD+BC+BD$	22
TH44w22	$AB+ACD+BCD$	22
TH54w22	$ABC+ABD$	18
TH34w32	$A+BC+BD$	17
TH54w32	$AB+ACD$	20
TH44w322	$AB+AC+AD+BC$	20
TH54w322	$AB+AC+BCD$	21
THxor0	$AB+CD$	20
THand0	$AB+BC+AD$	19
TH24comp	$AC+BC+AD+BD$	18

2.3. Method 3: quad-rail optimizations

For some circuits, it may be advantageous to use quad-rail optimization, referred to as Method 3 in Fig. 7. Two dual-rail signals yield the same five logic states as one quad-rail signal, however using quad-rail logic signals may lead to a more efficient design. Quad-rail optimization follows the same steps as does dual-rail optimization. In Step 3A, the NCL circuit is optimized by using reduced SOP expressions for all four rails of the output. These expressions are then mapped to TH1n and THnn gates. As in dual-rail optimization, a Karnaugh map can be constructed for each output, but instead of only $0s$ and $1s$, corresponding to a signal's $rail^0$ and $rail^1$, respectively, the K-map also contains $2s$ and $3s$, which correspond to a signal's $rail^2$ and $rail^3$, respectively. Reduced SOP expressions for the $0s$, $1s$, $2s$, and $3s$ in the Karnaugh map are derived. After these expressions for the outputs have been obtained, an assessment must be made to ensure that the complete output set cannot be generated without all of the inputs being present. If under any timing scenario, a complete output set can be generated without all of the inputs being present, the missing logic terms must be added to the reduced expressions to guarantee that the completeness of input criterion holds. This method will always generate two-level logic, given that threshold gates with a sufficiently large number of inputs are available. The first level will consist of THnn gates, to produce the required product terms; and the second level will consist of TH1n gates, which act to OR the product terms together to produce the desired outputs. In Step 3B, the common sub-expressions are factored and consolidated to reduce the gate count. Finally, the factored expressions for each rail are manipulated in Step 3C to obtain equations of the forms contained in Table 1. The observability criterion must be ensured for every circuit output from Steps 3A, 3B, and 3C.

2.4. Performance assessment

To assess the performance of alternate designs, Mentor Graphics' ModelSim tool was used to simulate VHDL implementations of the circuits to generate their timing characteristics and Mentor Graphics' AccuSim tool was used to calculate each circuit's power dissipation. All NCL circuits presented herein have been exhaustively tested and their average cycle time, T_{DD} , has been reported. Furthermore, to highlight TCR's effect on power dissipation, the average power per operation, P_{DD} , was calculated for the best dual-rail, quad-rail, and MEAG counter designs. The technology library for the 27 macros is based on Spice simulations of static $0.5\ \mu\text{m}$ CMOS gates, operating at 3.3 V. Along with T_{DD} and P_{DD} , the number of gates and transistors has also been tabulated for comparison for each test circuit, as shown in Sections 3, 4 and 5.5. The design that best meets the desired criteria can then be selected for implementation.

3. Application to input-complete basic logic functions

Several optimizations can be used to generate designs that are very competitive in terms of speed and area as compared to other self-timed approaches. For example, Figs. 4, 8 and 9 show the conventional implementations of the logic functions: AND, OR, and XOR, respectively. Each of these may be obtained directly from their canonical form. Method 2 is readily applicable.

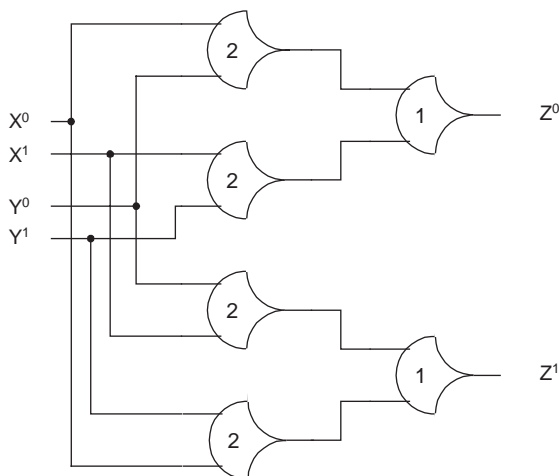


Fig. 8. Conventional input-complete OR function: $Z = X + Y$.

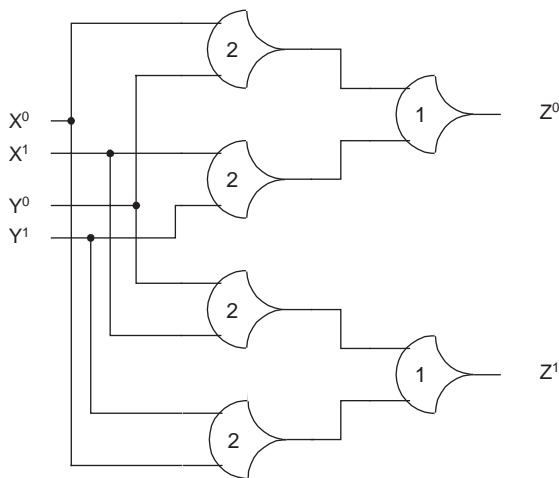


Fig. 9. Conventional input-complete XOR function: $Z = X \oplus Y$.

Dual-Rail Encoding Optimization achieves significant reduction in both area and speed. TCR Step 2C can be applied directly from the canonical form to reduce the circuit complexity and improve performance. Specifically, consider the objective of realizing an optimized input-complete 2-input OR function: $Z = X + Y$. The canonical expression for Z^0 is given by: $Z^0 = X^0 Y^0$, which

Table 2
Performance characteristics of input-complete NCL logic functions

	Component list	Gate delays	Gate count	Transistor count	T_{DD} (ns)
<i>Complete AND</i>					
Conventional	4 × TH22, 1 × TH13	2	5	56	1.58
TCR Method 2	1 × THand0, 1 × TH22	1	2	31	0.71
<i>Complete OR</i>					
Conventional	4 × TH22, 1 × TH13	2	5	56	1.58
TCR Method 2	1 × THand0, 1 × TH22	1	2	31	0.71
<i>XOR</i>					
Conventional	4 × TH22, 2 × TH12	2	6	60	1.70
TCR Method 2	2 × TH24comp	1	2	36	0.75

directly maps to a TH22 gate in Table 1. The canonical expression for Z^1 is given by: $Z^1 = X^1 Y^1 + X^0 Y^1 + X^1 Y^0$, which directly maps to a THand0 gate. Similarly, an optimized input-complete 2-input AND function: $Z = X \cdot Y$ can be realized. The canonical expression for Z^0 is given by: $Z^0 = X^0 Y^0 + X^0 Y^1 + X^1 Y^0$, which directly maps to a THand0 gate. The canonical expression for Z^1 is given by: $Z^1 = X^1 \cdot Y^1$, which directly maps to a TH22 gate. The derivation of an optimized 2-input XOR function: $Z = X \oplus Y$ is a bit more complex. The canonical expression for Z^0 is given by: $Z^0 = X^0 Y^0 + X^1 Y^1$, which directly maps to a THxor0 gate. The canonical expression for Z^1 is given by: $Z^1 = X^1 Y^0 + X^0 Y^1$, which also directly maps to a THxor0 gate. However, two transistors can be eliminated for each rail of Z by adding the two *don't care* terms, representing the cases when both rails of either X or Y are simultaneously asserted. The new equations for Z^0 and Z^1 are as follows: $Z^0 = X^0 Y^0 + X^1 Y^1 + X^0 X^1 + Y^0 Y^1$ and $Z^1 = X^1 Y^0 + X^0 Y^1 + X^0 X^1 + Y^0 Y^1$, both of which now map to TH24comp gates.

As shown in Table 2, the AND, OR, and XOR functions produced using TCR outperform the conventional canonical designs in terms of both area and throughput. In particular, the TCR optimized AND and OR functions are 2.2-fold faster and require 45% fewer transistors than the conventional canonical designs. Furthermore, the optimized XOR function is 2.3-fold faster and requires 40% fewer transistors than the conventional canonical design. The inverse logic functions, NAND, NOR, and NXOR, can easily be attained by exchanging the output rails of the AND, OR, and XOR functions, respectively.

4. Application to full adder

The truth table for a full adder circuit is shown in Fig. 10, where X and Y denote the input addends and C_i denotes the carry input. S and C_o denote the *sum* and *carry* outputs, respectively. This circuit can be extensively optimized using TCR Method 2. Applying TCR Step 2A, the K-map for the C_o output is obtained as shown in Fig. 11, yielding: $C_o^0 = X^0 Y^0 + C_i^0 X^0 + C_i^0 Y^0$ and $C_o^1 = X^1 Y^1 + C_i^1 X^1 + C_i^1 Y^1$. Both functions directly map to a TH23 gate, so factoring in Step 2B is not necessary. Since a TH23 gate does not produce an output which is complete with respect to

X	Y	C _i	C _o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Fig. 10. Truth table for full adder.

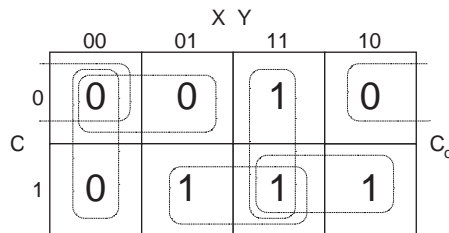


Fig. 11. K-map for C_o output of full adder.

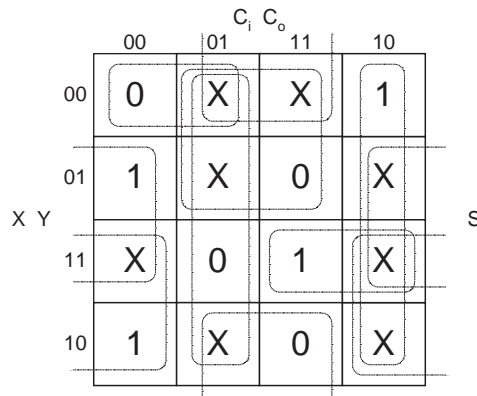


Fig. 12. K-map for S output of full adder.

any of its inputs, there must be another output or set of outputs that enforce the completeness of input criterion. As explained below, the *sum* output, S, will enforce the completeness of input criterion for the circuit as a whole, thus allowing the *carry* output to be input-incomplete.

The K-map for S, based on X, Y, C_i, and the intermediate output C_o, is shown in Fig. 12, with essential prime implicants covered. This covering yields: $S^0 = C_o^1 X^0 + C_o^1 Y^0 + C_o^1 C_i^1 + X^0 Y^0 C_i^0$ and $S^1 = C_o^0 X^1 + C_o^0 Y^1 + C_o^0 C_i^1 + X^1 Y^1 C_i^1$, both of which directly map to TH34W2 gates, so factoring in Step 2B is not necessary. C_o is taken as the A input such that $W(C_o) = 2$, as shown in Fig. 13. Checking input-completeness, the *carry* output requires at least two inputs to be

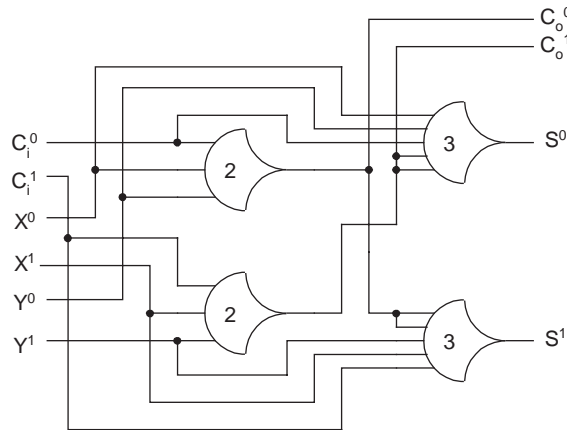


Fig. 13. Optimized NCL full adder [1].

Table 3
Full adder using various self-timed methodologies

Method	Design level	Gate delays for C_o	Gate delays for S	Gate count	Transistor count
Seitz [2]	Gate	2	3	18	154
Anantharaman [7]	Gate	2	2	12	168
DIMS [9]	Gate	2	2	12	168
David [6]	Gate	3	3	20	186
Singh [8]	Gate	6	4	19	192
TCR (Method 2)	Gate	1	2	4	80
Martin [29]	Transistor	1	1	3	42 or 34

generated and the *sum* output requires either the carry output and one more input, or all three inputs to be generated; so all three inputs are needed to generate the *sum* output. Therefore, the completeness of input criterion holds for the circuit as a whole.

As shown in Table 3, the NCL design of the full adder produced using TCR optimizations can outperform those of other self-timed methods, such as Anantharaman's and DIMS, Seitz's, David's, and Singh's approaches, shown in Figs. 14–17, respectively. Here n -input C-elements are drawn as THnn gates since their functionality is identical. The NCL design has far fewer gates and transistors, while requiring fewer logic levels to produce the *carry* output, C_o . NCL also requires fewer logic levels to produce the *sum* output, S , than three of the five other methods, and has the same number of logic levels for S as the other two. Notice that the NCL full adder uses the *carry* output as an input to compute the *sum* output, whereas the other methods compute the *sum* and *carry* outputs independently. This is because for the other methods it is not practical to use the *carry* output to help generate the *sum* output. For the other methods the *carry* output is generated in the same number of logic levels, or more, as the *sum* output. Therefore, to use the *carry* output as an input for calculating the *sum* output would require more logic levels, as well as more gates. Besides NCL, only Seitz's full adder can be designed such that C_o can be computed before the C_i

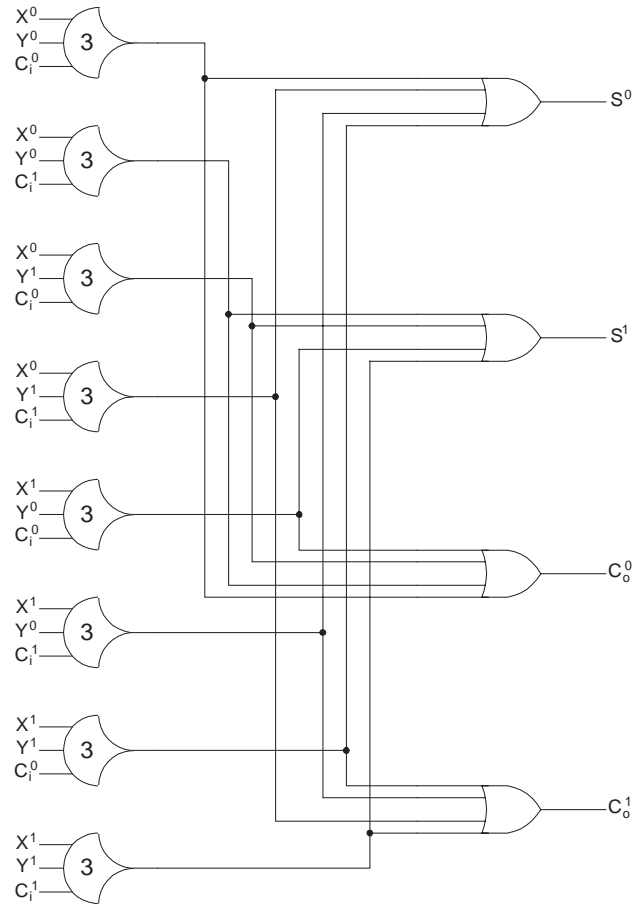


Fig. 14. Full adder using Anantharaman's approach or DIMS [9].

input is known for the cases $A=DATA0$, $B=DATA0$ and $A=DATA1$, $B=DATA1$. This optimization is important if the full adder component is to be used to construct an N -bit ripple-carry adder; since it allows the addition to be performed in $O(\log_2 N)$ on average instead of $O(N)$. This optimization could be applied to DIMS, Anantharaman's approach, and David's method, if their signaling scheme was slightly changed such that it coincided with the "weak conditions" of delay-insensitive signaling defined by Seitz [2] and used by NCL.

NCL circuits are often able to outperform other self-timed methods since NCL targets a wider range of logical operators whereas other methods target a more standard, restricted set. However, the full adder can be further optimized by design methods at the transistor level as demonstrated by Martin [29]. His full adder requires three complex transistor networks: the first computes both rails of the *sum* output, while the second and third each compute one rail of the *carry* output. The resulting design consists of only 42 transistors, when the input and output inverters are included, or 34 transistors otherwise. However, this method is not directly comparable to the

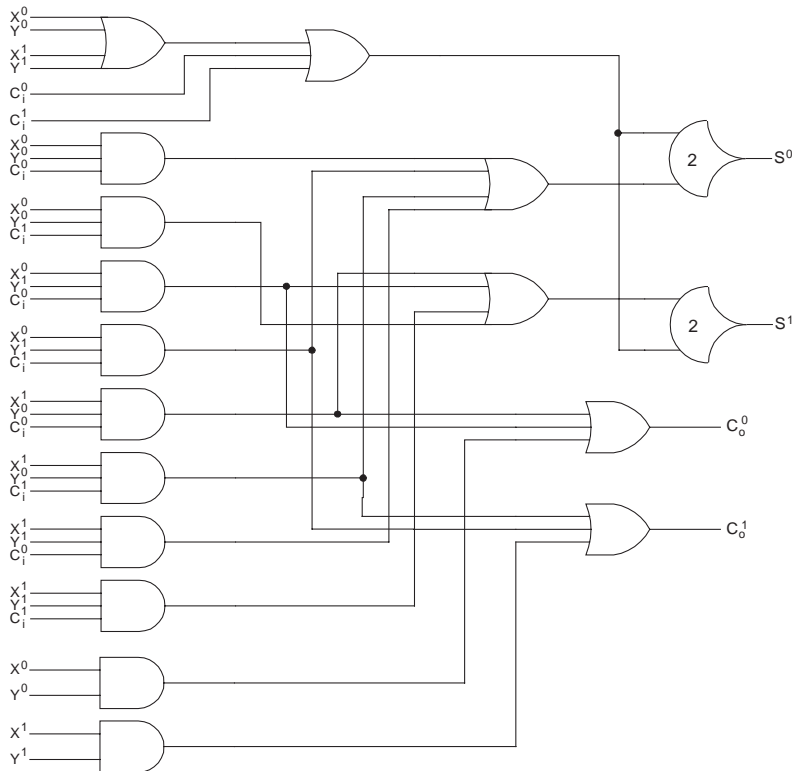


Fig. 15. Full adder using Seitz's approach [2].

other above-mentioned methods since it optimizes designs at the transistor level instead of targeting a predefined set of gates.

As for general-purpose methods, DIMS, Seitz's method, and Anantharaman's approach require functions in canonical form, so that no simplification is possible. DIMS and Anantharaman's approach cannot outperform NCL, and at best will be identical only if the NCL design cannot be simplified beyond its canonical form. Seitz's approach can outperform NCL in terms of area, but not speed, for a limited class of circuits. These include functions with 4 or more inputs, with one or few outputs, that contain almost all 1s or 0s in their truth table. These are the types of circuits that will receive little benefit from TCR optimizations. David's and Singh's approaches also favor these same classes of functions, and typically produce more efficient circuits than those obtainable by Seitz's approach. Singh's approach will require less area, but more delay than TCR for these types of functions, whereas David's approach will provide the same speed with significantly less area. For example, consider the function: $f(a, b, c, d) = ab'cd'$ [6]. Table 4 shows that Seitz's, David's, and Singh's circuits are all better than those obtainable by TCR, in terms of area for this function and that Anantharaman's approach is the same. However, only David's approach outperforms TCR in both area and speed. David's approach is better because this function, and others like it, cannot be simplified beyond their canonical form in NCL to ensure input-completeness, so no simplification is possible by TCR methods.

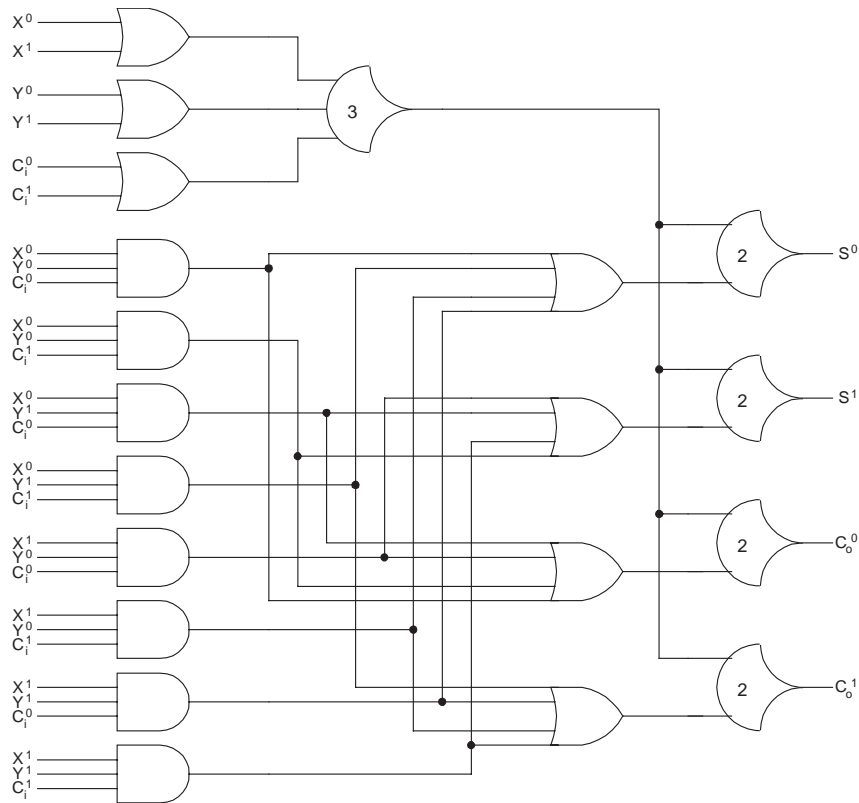


Fig. 16. Full adder using David's approach.

5. Application to up-counter

A number of experiments based on the 4-bit counter shown in Fig. 18 have been conducted. The specifications for this counter included a full NCL interface with request and acknowledge signals labeled K_i and K_o , respectively. Functionality was specified to reset *count* to 0000_2 when the *reset* signal is applied, to increment *count* by 1 when *inc* = 1, and to keep *count* the same when *inc* = 0. The counter will rollover to 0000_2 when *count* = 1111_2 and *inc* = 1.

The functional design of the 4-bit counter, shown in Fig. 19, will be the same for all counter models considered herein. However, the *Increment Circuitry* will differ based on the particular TCR optimization method that is used. Fig. 19 shows that there are three NCL registers to feedback the current count to the increment circuitry. These *Registration Stages* act to control the DATA/NULL wavefronts, through their request line, K_i , and their acknowledge line, K_o . The completion logic (*COMP*) detects complete DATA and NULL sets, where all outputs are DATA or all outputs or NULL, respectively, at the output of NCL registration. Three registers are used to prevent handshaking lockup scenarios [1]. This technique of organizing registers into a ring is fully discussed in [9,30].

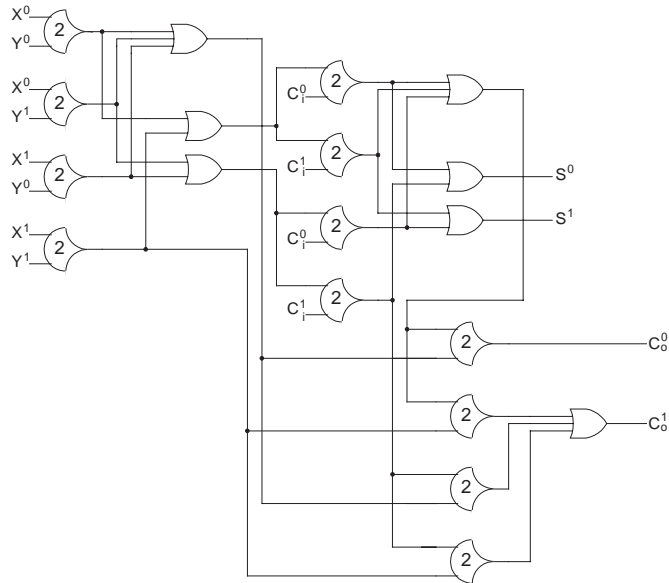


Fig. 17. Full adder using Singh's approach.

Table 4
Self-timed method comparison for $f(a, b, c, d) = ab'cd'$

Method	Gate delays	Gate count	Transistor count
Seitz [2]	4	25	250
Anantharaman [7]	3	21	368
DIMS [9]	3	21	368
David [6]	3	9	88
Singh [8]	4	15	168
NCL	3	21	368

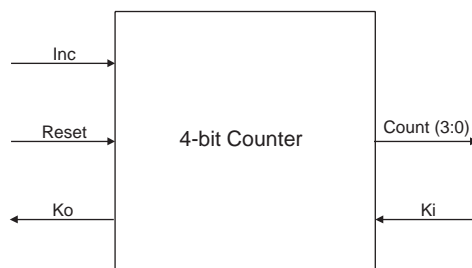


Fig. 18. A 4-bit up-counter block diagram.

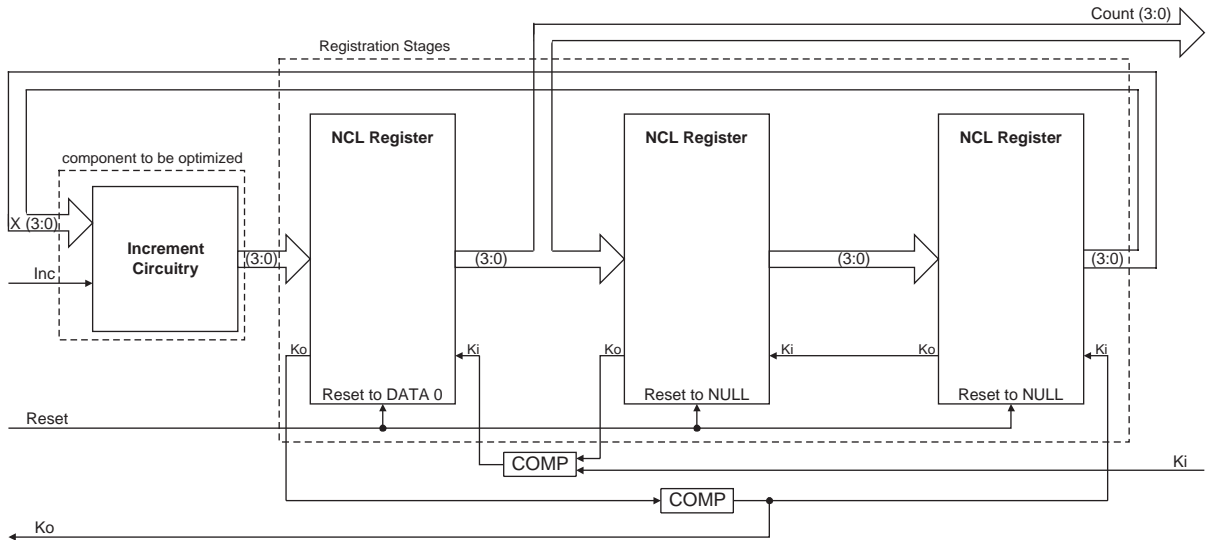


Fig. 19. Up-counter with three-register feedback.

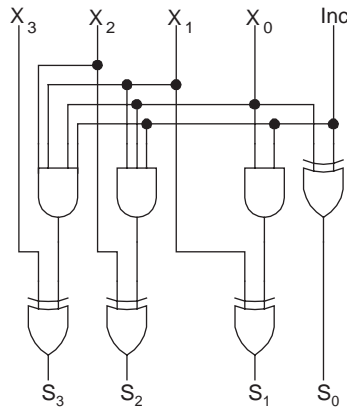


Fig. 20. Boolean increment circuit.

5.1. Method 1: input-incomplete functions

This technique was applied to the optimized Boolean increment circuitry of the 4-bit counter shown in Fig. 20, which is based on a carry look-ahead adder. The Boolean XOR gates were replaced with the XOR function described in Section 3, and the Boolean AND gates were replaced with input-incomplete versions of the AND function shown in Fig. 3. The resulting logic diagram is depicted in Fig. 21. The completeness of input criterion for the circuit as a whole is satisfied since all of the inputs are needed to produce a complete output set, due to the inherent completeness of input of an XOR function. This model has a worse-case path delay of two NCL gates in the increment circuitry. It consists of 14 NCL gates and T_{DD} was determined to be 4.81 ns via simulation in the Mentor Graphics toolset.

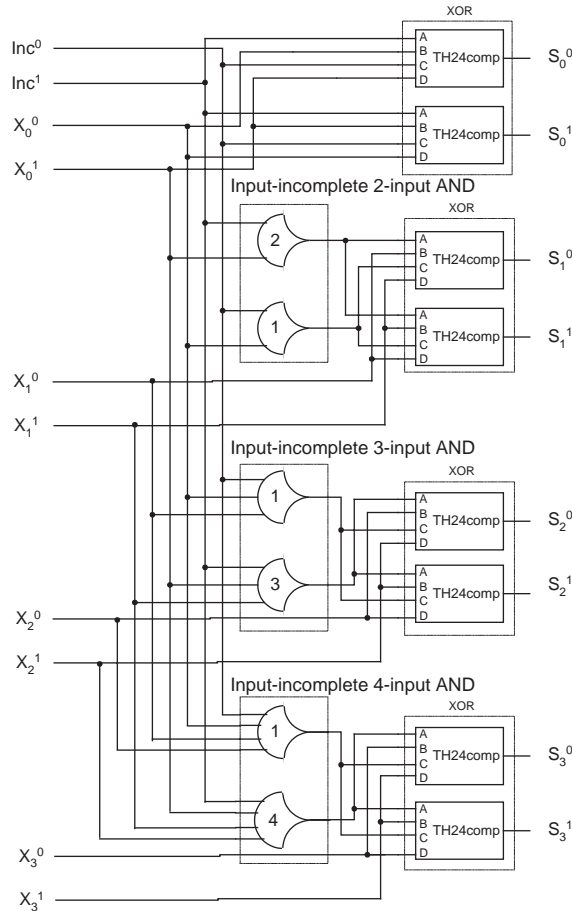


Fig. 21. Increment circuit using input-incomplete AND functions.

5.2. Method 2: dual-rail encoding optimizations

The resulting logic diagram after deriving reduced SOP expressions from Step 2A is shown in Fig. 22. This model has a theoretical worst-case path delay of 2 threshold gates in the increment circuitry. However, TH15 and TH55 gates are not supported in the 27 NCL macros, since they require 5 inputs. Therefore, the TH15 gate was realized by connecting a TH14 gate in series with a TH12 gate. However, this technique could not be applied to the TH55 gate, since this decomposition would violate the observability criterion. Instead the two TH55 gates were decomposed into one TH44 gate and two TH22 gates, in order to maintain observability of every gate transition at the output. This decomposition is valid since every transition of the TH44 gate will result in exactly one of the two TH22 gates also transitioning. The decompositions caused the worst-case path delay to be three NCL gates, instead of two. The reduced SOP model consists of 39 gates, but only 36 gates are necessary if TH55 and TH15 gates are used. From Synopsys simulation, T_{DD} was determined to be 5.34 ns.

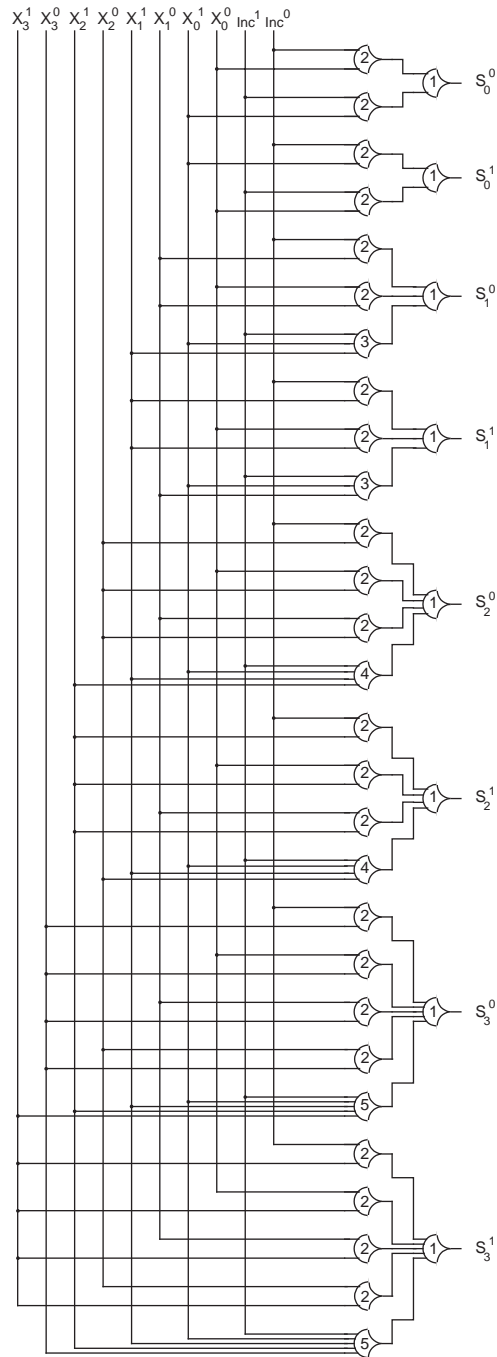


Fig. 22. Increment circuit using dual-rail reduced SOP expressions.

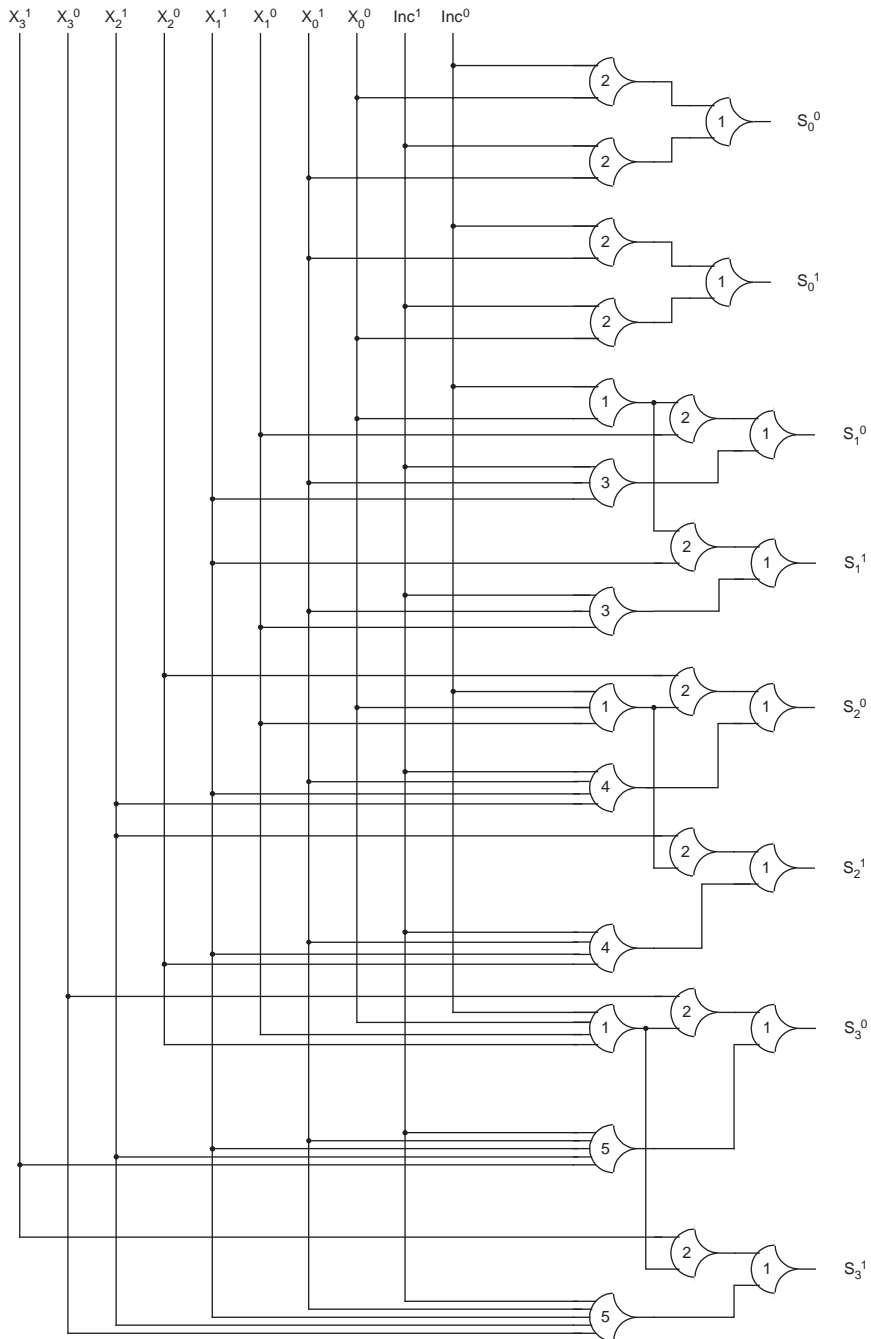


Fig. 23. Increment circuit using dual-rail factored SOP expressions.

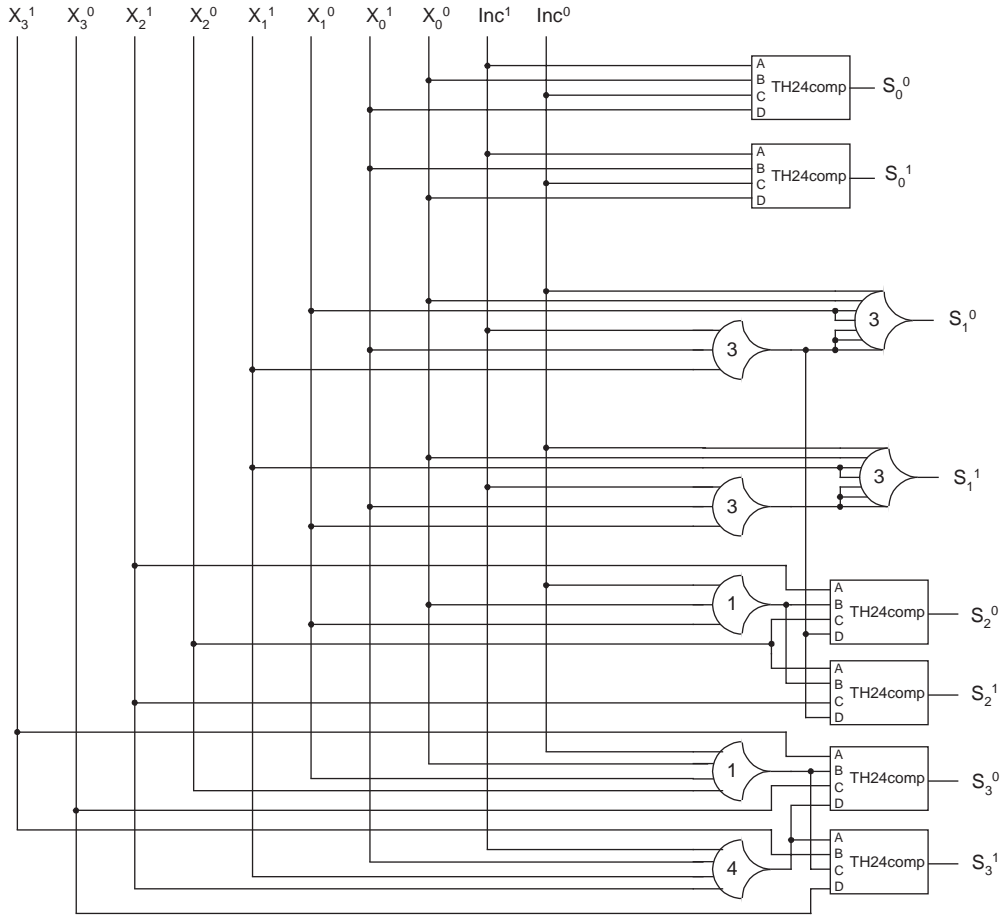


Fig. 24. Dual-rail increment circuit using complex gates.

To further reduce the gate count, the expressions for S_1 , S_2 , and S_3 can be factored using Step 2B. This factoring increases the worst-case path delay from two NCL gates to three NCL gates. Since constructing TH55 and TH15 gates for the reduced SOP model from smaller gates caused a worst-case path delay of 3 threshold gates, factoring did not increase the depth of the critical path. The logic diagram for the increment circuitry factored form is shown in Fig. 23. The factored SOP model consists of 28 gates, but only 27 gates are necessary if TH55 gates are used. From Mentor Graphics simulation, T_{DD} was determined to be 5.28 ns.

Step 2C maps the factored expressions to the full 27 macros in Table 1, reducing both the number of gates and the number of logic levels. Note that the expressions for S_0 , S_2 , and S_3 can be mapped to TH24comp gates by adding two *don't care* terms as for the XOR function explained in Section 3. The logic diagram for the increment circuitry using complex gates is shown in Fig. 24. It has a worst-case path delay of two NCL gates in the increment circuitry. The complex dual-rail model consists of 13 gates, and from Mentor Graphics simulation T_{DD} was determined to be 4.81 ns and P_{DD} was calculated as 14.44 μ W.

5.3. Method 3: quad-rail encoding optimizations

Quad-rail optimizations proceed in a similar fashion to dual-rail optimizations. In Step 3A, the NCL circuit is optimized by using reduced SOP expressions for all four rails of both outputs, S_0 and S_1 , the low order two bits and the high order two bits, respectively, derived from the Karnaugh map shown in Fig. 25. Note that not all of the coverings that eliminate *Inc* are shown, so as not to clutter the drawing. The reduced SOP expressions derived from these K-maps are as follows: $S_0^0 = Inc^0 X_0^0 + Inc^1 X_0^3$, $S_0^1 = Inc^0 X_0^1 + Inc^1 X_0^0$, $S_0^2 = Inc^0 X_0^2 + Inc^1 X_0^1$, $S_0^3 = Inc^0 X_0^3 + Inc^1 X_0^2$, $S_1^0 = Inc^0 X_1^0 + X_0^0 X_1^0 + X_0^1 X_1^0 + X_0^2 X_1^0 + Inc^1 X_0^3 X_1^3$, $S_1^1 = Inc^0 X_1^1 + X_0^0 X_1^1 + X_0^1 X_1^1 + X_0^2 X_1^1 + Inc^1 X_0^3 X_1^0$, $S_1^2 = Inc^0 X_1^2 + X_0^0 X_1^2 + X_0^1 X_1^2 + X_0^2 X_1^2 + Inc^1 X_0^3 X_1^1$, $S_1^3 = Inc^0 X_1^3 + X_0^0 X_1^3 + X_0^1 X_1^3 + X_0^2 X_1^3 + Inc^1 X_0^3 X_1^2$. These equations can now be directly mapped to TH1n and THnn gates to produce the reduced SOP model. This model has a theoretical worst-case path delay of two NCL gates in the increment circuitry. However, TH15 gates are not supported in the 27 NCL macros, since they require 5 inputs. Therefore, the actual worst-case path delay is three NCL gates. The reduced SOP model consists of 40 gates, but only 36 gates are necessary if TH15 gates are used. From Mentor Graphics simulation, T_{DD} was determined to be 5.59 ns. A logic diagram of this model has not been included in order to conserve space.

To further reduce the gate count, the expressions for S_1 can be factored using Step 3B. This factoring increases the worst-case path delay from two NCL gates to three NCL gates. Since constructing TH15 gates for the reduced SOP model from smaller gates caused a worst-case path delay of 3 gates, factoring did not increase the depth of the critical path. The factored equations for S_1 are as follows: $S_1^0 = X_1^0(Inc^0 + X_0^0 + X_0^1 + X_0^2) + Inc^1 X_0^3 X_1^3$, $S_1^1 = X_1^1(Inc^0 + X_0^0 + X_0^1 + X_0^2) + Inc^1 X_0^3 X_1^0$, $S_1^2 = X_1^2(Inc^0 + X_0^0 + X_0^1 + X_0^2) + Inc^1 X_0^3 X_1^1$, $S_1^3 = X_1^3(Inc^0 + X_0^0 + X_0^1 + X_0^2) + Inc^1 X_0^3 X_1^2$. The factored SOP model reduced the gate count to only 25 gates, and from Mentor Graphics simulation, T_{DD} was determined to be 5.57 ns. A logic diagram of this model has not been included in order to conserve space.

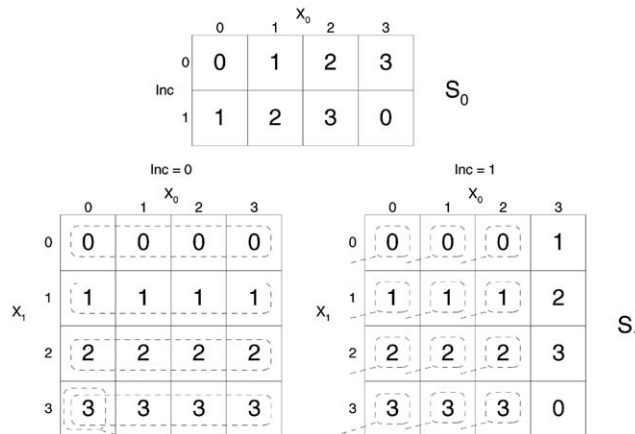


Fig. 25. Karnaugh map for quad-rail counter.

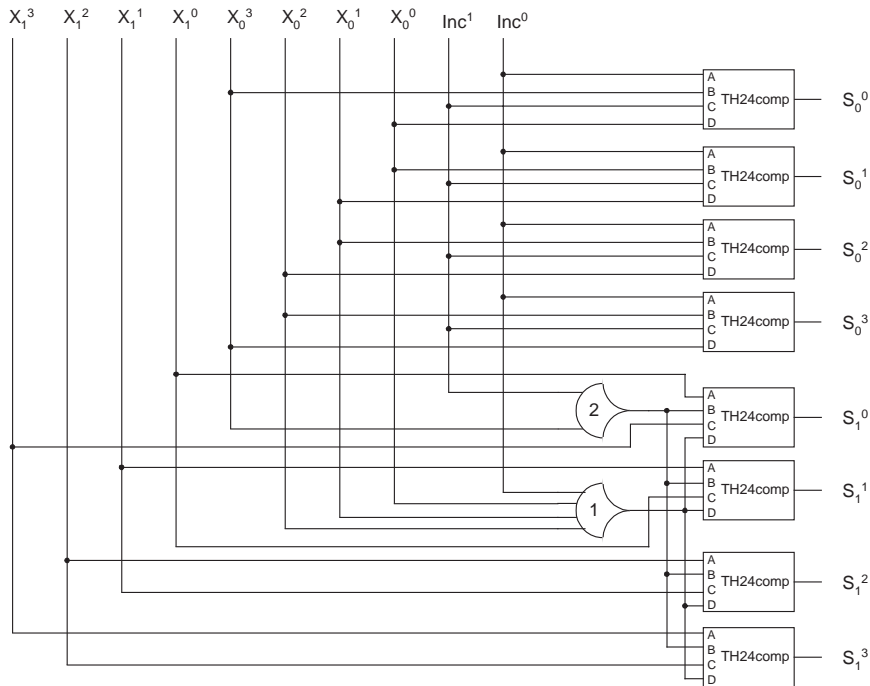


Fig. 26. Quad-rail increment circuit using complex gates.

Step 3C maps the factored expressions to the full 27 macros in Table 1, reducing both the number of gates and the number of logic levels. Note that the expressions for S_0 and S_1 can be mapped to TH24comp gates by adding two *don't care* terms as for the XOR function explained in Section 3. The logic diagram for the increment circuitry using complex gates is shown in Fig. 26. It has a worse case path delay of two NCL gates in the increment circuitry. The complex quad-rail model consists of 10 gates and from Mentor Graphics simulation T_{DD} was determined to be 5.47 ns and P_{DD} was calculated as 9.30 μW .

5.4. MEAG implementation

For comparison purposes a 16-rail MEAG implementation of the increment circuitry was also designed, as shown in Fig. 27. Other MEAG optimization follows the same steps as does quad-rail optimization, where the Karnaugh map(s) contain numbers in the range of 0 to $N - 1$, for an N -rail MEAG, yielding reduced SOP expressions for each rail, which can then be factored, and finally mapped to the 27 NCL macros in Table 1. This design has a worse-case path delay of one NCL gate in the increment circuitry; it consists of 16 gates; and from Mentor Graphics simulation T_{DD} was determined to be 8.77 ns and P_{DD} was calculated as 5.37 μW .

Since this increment circuitry has a worse-case delay of one gate, while both the dual- and quad-rail versions require two gate delays, it would seem that the 16-rail MEAG counter

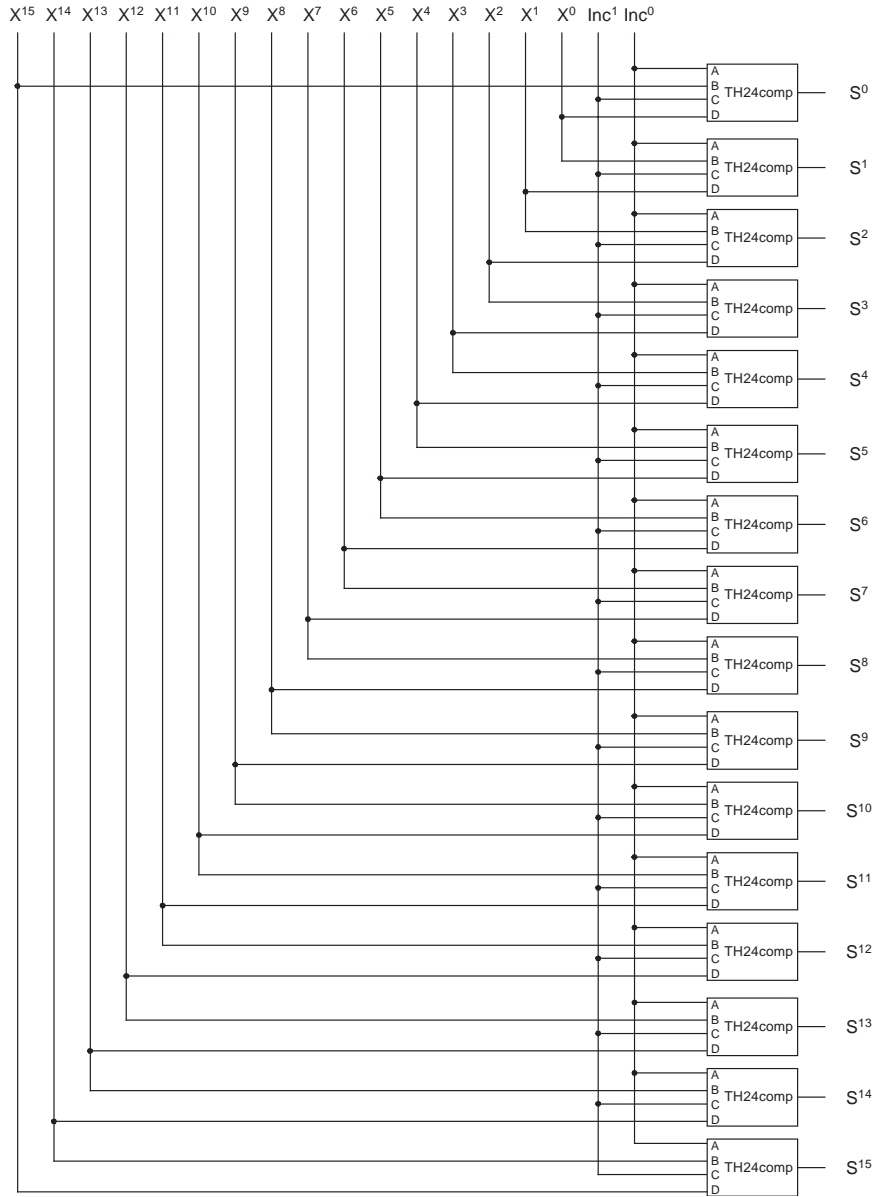


Fig. 27. A 16-rail MEAG increment circuit using complex gates.

should be faster. However, this is not the case because the 16-rail registers used in the feedback loop require two gate delays to generate K_0 , whereas both the dual- and quad-rail registers only require one gate delay for K_0 generation. Thus the larger register delay outweighs the decreased increment circuitry delay, resulting in a greater average cycle time for the 16-rail MEAG counter.

Table 5
Alternate designs for NCL up-counter increment circuit

Model type	Theoretical gate count	Actual gate count	Transistor count	T_{DD} (ns)	P_{DD} (μ W)
(1) Incomplete AND	14	14	216	4.81	
(2a) Reduced dual-rail	36	39	460	5.34	
(2b) Factored dual-rail	27	28	308	5.28	
(2c) Complex dual-rail	13	13	212	4.81	14.44
(3a) Reduced quad-rail	36	40	440	5.59	
(3b) Factored quad-rail	25	25	266	5.57	
(3c) Complex quad-rail	10	10	166	5.47	9.30
16-rail MEAG	16	16	288	8.77	5.37

5.5. Up-counter performance summary

Table 5 lists the timing and gate counts for each of the eight counter models. The *theoretical gate count* is the number of gates that would be required if TH55 and/or TH15 gates were used. Since these gates are not part of the 27 NCL macros, they have been constructed from existing gates, as discussed in Section 5.2, to yield the *actual gate count*. Table 5 indicates that the factored forms of both the dual- and quad-rail circuits yield fewer gates and transistors, as well as smaller cycle times, compared to their original reduced SOP forms. However, the complex gate models yield the best time and space performance for Methods 2 and 3, as expected. The optimal design in terms of speed is generated from both Methods 1 and 2C, although the design from Method 2C is preferred since it contains fewer gates and transistors. This design is 14% and 82% faster than the area- and power-optimized designs, respectively. The most area efficient design is generated from Method 3C, requiring 22% and 42% fewer transistors than the speed- and power-optimized designs, respectively. The 16-rail MEAG counter optimizes power consumption, dissipating 63% and 42% less power than the speed- and area-optimized designs, respectively.

6. Conclusion

When functions do not require canonical form for input-completeness, TCR can produce self-timed circuits that require less area and fewer logic levels than alternative gate-level approaches. TCR is applicable when composing logic functions where each gate is a state-holding element. The TCR method combines techniques such as input-incomplete functions, dual-rail, quad-rail, and other MEAG encodings, reduced SOP expressions, and factored SOP expressions for reducing gate count. It then employs a mapping of the factored SOP equations to a set of 27 macros, which constitute the set of all functions consisting of four or fewer variables. The TCR method has been used to successfully design various delay-insensitive MACs [31], multipliers [32], and ALUs [33]. The results can also be extended to a gate-level pipelining strategy for circuits composed of state-holding elements to maximize throughput of combinational circuits produced by TCR methods as

described in [34]. Furthermore, throughput of NCL circuits can be increased by applying the NULL Cycle Reduction technique to reduce the NULL cycle time without compromising delay-insensitivity [35].

References

- [1] K.M. Fant, S.A. Brandt, NULL convention logic: a complete and consistent logic for asynchronous digital circuit synthesis, Proceedings of the International Conference on Application Specific Systems, Architectures, and Processors, 1996, pp. 261–273.
- [2] C.L. Seitz, System timing, in: C. Mead, L. Conway (Eds.), Introduction to VLSI Systems, Addison-Wesley, Reading, MA, 1980, pp. 218–262.
- [3] A.J. Martin, Programming in VLSI, in: C.A.R. Hoare (Ed.), Development in Concurrency and Communication, Addison-Wesley, Reading, MA, 1990, pp. 1–64.
- [4] K. Van Berkel, Beware the isochronic fork, integration, VLSI J. 13/2 (1992) 103–128.
- [5] D.E. Muller, Asynchronous logics and application to information processing, in: H. Aiken, W.F. Main (Eds.), Switching Theory in Space Technology, Stanford University Press, Stanford, CA, 1963, pp. 289–297.
- [6] I. David, R. Ginosar, M. Yoeli, An efficient implementation of Boolean functions as self-timed circuits, IEEE Trans. Comput. 41/1 (1992) 2–10.
- [7] T.S. Anantharaman, A delay insensitive regular expression recognizer, IEEE VLSI Techn. Bull. September (1986).
- [8] N.P. Singh, A design methodology for self-timed systems, Master's Thesis, MIT/LCS/TR-258, Laboratory for Computer Science, MIT, 1981.
- [9] J. Sparso, J. Staunstrup, M. Dantzer-Sorensen, Design of delay insensitive circuits using multi-ring structures, in: Proceedings of the European Design Automation Conference, 1992, pp. 15–20.
- [10] A.J. Martin, Compiling communicating processes into delay-insensitive VLSI circuits, Distributed Comput. 1/4 (1986) 226–234.
- [11] C.H. (Kees) van Berkel, Handshake circuits: an intermediary between communicating processes and VLSI, Ph.D. Thesis, Eindhoven University of Technology, 1992.
- [12] A.J. Martin, A. Lines, R. Manohar, M. Nystrom, P. Penzes, R. Southworth, U. Cummings, T.K. Lee, The design of an asynchronous MIPS R3000 microprocessor, in: H. Aiken, W.F. Main (Eds.), Proceedings of the 17th Conference on Advanced Research in VLSI, Ann Arbor, MI, 1997, pp. 164–181.
- [13] A.J. Martin, S.M. Burns, T.K. Lee, D. Borkovic, P.J. Hazewindus, The design of an asynchronous microprocessor, advanced research in VLSI, in: Proceedings of the Decennial Caltech Conference on VLSI, Pasadena, CA, 1989, pp. 351–373.
- [14] W. Hardt, B. Kleinjohann, FLYSIG: dataflow oriented delay-insensitive processor for rapid prototyping of signal processing, in: Proceedings of the Ninth International Workshop on Rapid System Prototyping, Leuven, 1998, pp. 136–141.
- [15] P.K. Tsang, C.C. Cheung, K.H. Leung, T.K. Lee, P.H.W. Leong, MSL16A: an asynchronous forth microprocessor, in: Proceedings of the IEEE Region 10 Conference, Vol. 2, 1999, pp. 1079–1082.
- [16] T. Nanya, Y. Ueno, H. Kagotani, M. Kuwako, A. Takamura, TITAC: design of a quasi-delay-insensitive microprocessor, IEEE Des. Test Comput. 11/2 (1994) 50–63.
- [17] J. McCardle, D. Chester, Measuring an asynchronous processor's power and noise, Proceedings of the Synopsys User Group Conference (SNUG), Boston, 2001.
- [18] S.H. Unger, Asynchronous Sequential Switching Circuits, Wiley, New York, 1969.
- [19] S.M. Nowick, D.L. Dill, Synthesis of asynchronous state machines using a local clock, in: Proceedings of the ICCAD, 1991, pp. 192–197.
- [20] I.E. Sutherland, Micropipelines, Commun. ACM 32/6 (1989) 720–738.
- [21] A. Martin, The limitations to delay-insensitivity in asynchronous circuits, advanced research in VLSI, in: H. Aiken, W.F. Main (Eds.), Proceedings of the Sixth MIT Conference, Cambridge, MA, 1990, pp. 263–278.

- [22] T. Verhoeff, Delay-insensitive codes—an overview, *Distributed Comput.* 3 (1988) 1–8.
- [23] G.E. Sobelman, K.M. Fant, CMOS circuit design of threshold gates with hysteresis, in: *Proceedings of the IEEE International Symposium on Circuits and Systems (II)*, 1998, pp. 61–65.
- [24] S.M. Burns, Performance analysis and optimization of asynchronous circuits, Ph.D. Thesis, CS-TR-91-1, Caltech, 1991.
- [25] S.M. Burns, General conditions for the decomposition of state holding elements, in: *Proceedings of the Second International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1996, pp. 48–57.
- [26] M.L. Dertouzos, *Threshold Logic: a synthesis approach*, MIT Press, Cambridge, MA, 1965.
- [27] Lewis & Coates, *Threshold Logic*, Wiley, New York, 1967.
- [28] C. Sheng, *Threshold Logic*, Ryerson Press, New York, 1969.
- [29] A.J. Martin, Asynchronous data paths and the design of an asynchronous adder, *Formal Methods Syst. Des.* 1/1 (1992) 117–137.
- [30] T.E. Williams, Self-timed rings and their application to division, Ph.D. Thesis, CSL-TR-91-482, Department of Electrical Engineering and Computer Science, Stanford University, 1991.
- [31] S.C. Smith, R.F. DeMara, J.S. Yuan, M. Hagedorn, D. Ferguson, NULL convention multiply and accumulate unit with conditional rounding, scaling, and saturation, *J. Syst. Architecture* 47/12 (2003) 977–998.
- [32] S.K. Bandapati, S.C. Smith, M. Choi, Design and characterization of NULL convention self-timed multipliers, *IEEE Des. Test Comput.: Special Issue on Clockless VLSI Design* 30/6 (2003) 26–36.
- [33] S.K. Bandapati, S.C. Smith, Design and characterization of NULL convention arithmetic logic units, in: *Proceedings of the 2003 International Conference on VLSI*, 2003, pp. 178–184.
- [34] S.C. Smith, R.F. DeMara, J.S. Yuan, M. Hagedorn, D. Ferguson, Delay-insensitive gate-level pipelining, integration, *VLSI J.* 30/2 (2001) 103–131.
- [35] S.C. Smith, R.F. DeMara, J.S. Yuan, M. Hagedorn, D. Ferguson, speedup of delay-insensitive digital systems using NULL cycle reduction, in: *Proceedings of the 10th International Workshop on Logic and Synthesis*, Lake Tahoe, CA, 2001, pp. 185–189.



Scott C. Smith is an assistant professor in the Department of Electrical and Computer Engineering at the University of Missouri—Rolla. His research interests include computer architecture, logic design, embedded-system design, and VLSI. Dr. Smith received BS degrees in electrical engineering and computer engineering from the University of Missouri—Columbia in May of 1996. He then received an MS in electrical engineering from the University of Missouri—Columbia in May of 1998, and a Ph.D. in computer engineering from the University of Central Florida, Orlando in May of 2001. He is a member of Sigma Xi, Eta Kappa Nu, Tau Beta Pi, the IEEE, and the American Society of Engineering Education.



Dr. Ronald DeMara received the B.S.E.E. degree from Lehigh University in 1987, the M.S.E.E. degree from the University of Maryland, College Park in 1989, and the Ph.D. degree in Computer Engineering from the University of Southern California in 1992. Since 1993, he has been a full-time faculty at the University of Central Florida and is currently an Associate Professor. He served as Associate Chair and Program Coordinator of the Computer Engineering program, has taught 11 different courses, and developed four courses. He has supervised a total of 19 Master students and 4 Ph.D. students. He has completed over 60 publications in conference proceedings, journals and book chapters.



J.S. Yuan received the Ph.D. degree in 1988 from the University of Florida. Currently, he is a professor and director of the Chip Design and Reliability Laboratory at the School of Electrical Engineering and Computer Science at University of Central Florida. Dr. Yuan has published more than 140 papers in referred journals and conference proceedings in the area of semiconductor devices and circuits. He has authored the book *Semiconductor Device Physics and Simulation*, published by Plenum in 1998 and the book *SiGe, GaAs, and InP Heterojunction Bipolar Transistors*, published by Wiley in 1999.



Dr. David R. Lamb is President and Chief Executive Officer of Theseus Logic, Inc. Prior to joining Theseus Logic in late 1996, David Lamb had over 30 years experience in the semiconductor field and held senior industry and university positions in both the USA and UK. Most recently, he was Associate Director at the Honeywell Technology Center where he managed a broad portfolio of technology areas including data acquisition systems, wireless communications, photonics, information processing systems, and flat panel/head mounted displays. He received the Lund Award, Honeywell's highest award for managerial excellence. David has Ph.D. and M.Sc. degrees from London University, as well as B.A. and M.A. degrees from Oxford University.