CMOS Implementation of Threshold Gates with Hysteresis

Farhad Parsan, Scott Smith

CMOS Implementation of Threshold Gates with Hysteresis

Farhad A. Parsan[1], and Scott C. Smith[1]

University of Arkansas, Fayetteville AR 72701, USA,
{fparsan,smithsco}@uark.edu

Abstract. NULL Convention Logic (NCL) is one of the mainstream asynchronous logic design paradigms. NCL circuits use threshold gates with hysteresis. In this chapter, the transistor-level CMOS design of NCL gates is investigated, and various gate styles are introduced and compared to each other. In addition, a novel approach to design static NCL gates is introduced. The new approach is based on integrating each pair of pull-up and pull-down transistor networks into one composite transistor network. The new static gates are then compared to the original ones in terms of delay, area, and energy consumption. It will be shown that the new gate style is significantly faster with negligible area and energy overhead.

Keywords: NULL convention logic, NCL, C-element, threshold gate

1 Introduction

Delay-insensitive asynchronous circuits have been the target of a renewed research effort because of the advantages they offer over traditional synchronous circuits. Minimal timing analysis, inherent robustness against power-supply, temperature, and process variations, reduced energy consumption, less noise and EMI emission, and easy design reuse are some of the benefits of these circuits [1]. NULL Convention Logic (NCL) is one of the mainstream asynchronous logic design paradigms that has been shown to be a promising method for designing delay-insensitive asynchronous circuits [2–4]. NCL circuits are correct-by-construction [3], requiring very little timing analysis, if any. In today's nanometer processes where meeting timing closure is becoming increasingly more difficult due to increasing clock rates and process variation, this quality is very attractive. NCL has been used for a number of industrial designs [4, 5], and is becoming more popular as design automation tools and techniques are being developed to automate the design process [6].

NCL circuits utilize threshold gates with hysteresis to maintain delay insensitivity. The general form of an NCL gate is very similar to a C-element [7]. Several CMOS implementation schemes have been introduced for NCL gates, including: dynamic, static, semi-static, and differential [8–10]. Each implementation offers some advantages and has some drawbacks in terms of delay, area, and power consumption. It is important for an NCL circuit designer to choose the CMOS

Fig. 1. Symbolically complete logic concept

implementation that best fits an application. In this chapter, we introduce different CMOS implementations of NCL gates and discuss their tradeoffs. In addition, a new approach to design static NCL gates is proposed and compared with the traditional approach in terms of delay, area, and energy consumption. It will be shown that the new static gates offer faster operation, with a small increase in area, and consume almost the same amount of energy. It will be also shown that when the NCL static gates are sized for improved switching speed, the slight area disadvantage is eliminated, resulting in better speed and area.

This chapter is organized as follows: an overview of NCL logic is presented in Section 2; Section 3 discusses different CMOS implementations of NCL gates and compares them against each other; the new static gate design is then introduced in Section 4 and compared to the traditional static gate design; sizing both versions of static gates is discussed in Section 5; and these static gate styles (sized and unsized) are used to implement NCL multipliers in Section 6 to compare transistor-level simulations; and finally, Section 7 presents conclusions.

2 NCL Overview

NCL is a delay-insensitive asynchronous logic design paradigm in which control is inherent within each datum. It follows the so-called "weak conditions" of Seitz's delay-insensitive signaling scheme [11]. Similar to other delay-insensitive logic design paradigms, NCL assumes that wire forks are isochronic [12]. NCL is a "symbolically complete" logic meaning that the output validity is unambiguously determined regardless of time reference [3]. Fig. 1 shows an unknown circuit inside a black box. Assuming that the unknown circuit is a traditional Boolean combinational circuit, once the inputs are asserted, it is impossible to determine when the outputs become valid unless the circuit's delay is known. However, if the unknown circuit is using a symbolically complete logic, such as NCL, one can determine the output validity without needing to know the circuit's delay. This is because NCL uses delay-insensitive codes for data communication, alternating between set and reset phases. In the set phase, data changes from spacer (called NULL) to a proper codeword (called DATA); and in the reset phase it changes back to NULL. NCL combines DATA and NULL into a single path presented by dual-rail, quad-rail, or in general, any Mutually Exclusive Assertion Group (MEAG) signals [13].

In practice, dual-rail signal encoding is more popular, since it is most similar to traditional Boolean logic. Table 1 shows the dual-rail signal encoding. A dual-rail signal, D, consists of two wires, D^0 and D^1. D is logic 0 (DATA0) when D^0

Table 1. Dual-rail encoding

	DATA0	**DATA1**	**NULL**	**Illegal**
$\mathbf{D^0}$	1	0	0	1
$\mathbf{D^1}$	0	1	0	1

= 1 and $D^1 = 0$; it is logic 1 (DATA1) when $D^0 = 0$ and $D^1 = 1$; and it is NULL when $D^0 = 0$ and $D^1 = 0$. D^0 and D^1 are mutually exclusive, such that they are never asserted at the same time; doing so would produce an illegal codeword.

Fig. 2 shows a simple Boolean AND gate versus a dual-rail NCL circuit that performs the same AND operation. For the Boolean AND gate, inputs X, and Y, and output Z use only one wire, but the dual-rail NCL AND circuit uses two wires for each input and output. For the Boolean AND gate, initially X = 1 and Y = 0, so output Z is 0. For the NCL AND circuit, initially X is DATA1 ($X^1 = 1$, $X^0 = 0$) and Y is DATA0 ($Y^0 = 1$, $Y^1 = 0$); therefore, output Z is DATA0 ($Z^0 = 1$, $Z^1 = 0$). For the Boolean AND gate, once input Y is asserted, output Z becomes invalid until the signal propagates through the AND gate and asserts the output (in this example after 1 ns). For the NCL AND circuit, however, before input Y changes to its next DATA value, all inputs must first transition to the NULL state (i.e., all input rails must go to 0) and we must wait until the output then transitions to NULL. At this point, the circuit is ready to accept a new DATA set, so X and Y can both change from NULL to DATA1. Consequently, output Z then changes from NULL to DATA1 after some time (1 ns in this example). An NCL circuit always cycles through NULL and DATA phases so the validity of the output can always be unambiguously determined by merely looking at the output. A NULL at the output means that the output is not valid and a DATA at the output means that the output is valid. For a Boolean circuit, on the other hand, the output validity can only be determined if we know when the inputs change and the worst-case propagation delay of the circuit.

NCL circuits are comprised of 27 threshold gates with hysteresis [2]. Each gate is denoted as THmnW$w_1 w_2 \ldots w_r$ in which m is the threshold of the gate, n is the number of inputs, and w_r is the weight of input r if its weight is greater than 1. Fig. 3(a) shows the symbol of an NCL gate. For an NCL gate with no weighted inputs, the output is asserted when at least m out of n inputs are asserted. As an example, the TH23 gate asserts its output when at least two out of three inputs are asserted; therefore, assuming the inputs are A, B, and C, the set function of a TH23 gate can be expressed as F = AB + AC + BC. Fig. 3(b) shows a TH23w2 gate, where input A has a weight of two. Therefore, asserting A alone asserts the gate output. The set function of the TH23w2 gate can then be expressed as F = A + BC.

The standard NCL gate library is shown in Table 2. Since NCL gates have hysteresis, once the output is asserted, it remains asserted until all the inputs

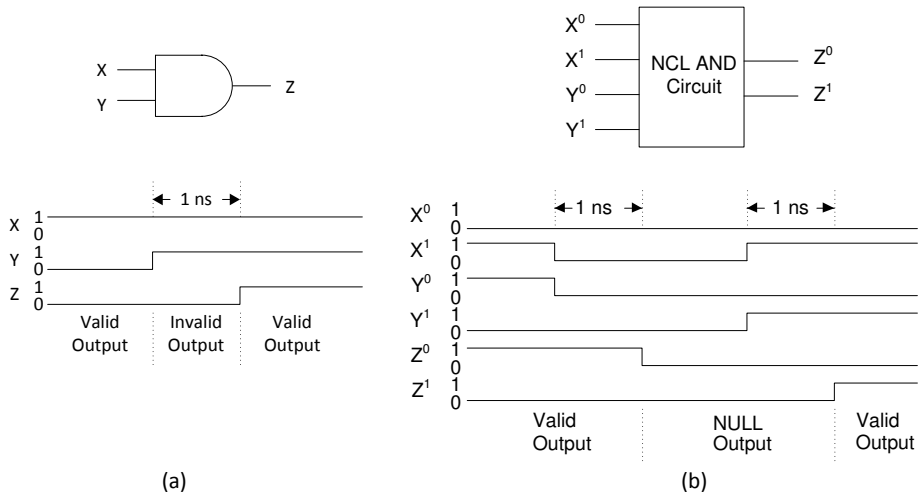


Fig. 2. (a) Boolean AND gate versus (b) NCL AND circuit

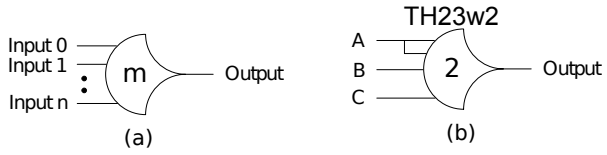


Fig. 3. (a) NCL threshold gate symbol (b) a weighted NCL threshold gate

are deasserted. Hysteresis behavior is required to ensure the delay-insensitivity of NCL circuits [2]. A non-weighted NCL gate with m = n (i.e., THnn) is a special case of NCL gates that is equivalent to an n-input C-element [14]. C-elements are well-known gates used in many other asynchronous logic design styles. A non-weighted NCL gate with m = 1 (i.e., TH1n) is another special case of NCL gates that is equivalent to an n-input Boolean OR gate. Among the 27 NCL gates, there are 3 gates (TH24comp, Thand0, THxor0) that are not actually threshold gates, but can be made by combining other threshold gates. These gates are included in the standard NCL gate library so that any function of 4 or fewer variables directly maps to one of these 27 NCL gates. Due to hysteresis, NCL gates act as memory elements; therefore, like any other memory element they have to be initialized. Initialization can be performed implicitly by asserting/deasserting all the gate inputs, or it can be done explicitly by adding a reset input to the gate. Depending on whether the reset signal asserts or deasserts the gate output, resettable gates are denoted with an 'n' (output deasserted) or a 'd' (output asserted) at the end of their name. Additionally, the output of an NCL gate can be provided in its inverted form; this is denoted by a small

Table 2. Standard NCL gate library

NCL Gate	Set Function
TH12	A + B
TH22	AB
TH13	A + B + C
TH23	AB + AC + BC
TH33	ABC
TH23w2	A + BC
TH33w2	AB + AC
TH14	A + B + C + D
TH24	AB + AC + AD + BC + BD + CD
TH34	ABC + ABD + ACD + BCD
TH44	ABCD
TH24w2	A + BC + BD + CD
TH34w2	AB + AC + AD + BCD
TH44w2	ABC + ABD + ACD
TH34w3	A + BCD
TH44w3	AB + AC + AD
TH24w22	A + B + CD
TH34w22	AB + AC + AD + BC + BD
TH44w22	AB + ACD + BCD
TH54w22	ABC + ABD
TH34w32	A + BC + BD
TH54w32	AB + ACD
TH44w322	AB + AC + AD + BC
TH54w322	AB + AC + BCD
THxor0	AB + CD
THand0	AB + BC + AD
TH24comp	AC + BC + AD + BD

circle at the output of the gate symbol and a 'b' at the end of the gate name. Fig. 4 shows how the NCL AND circuit in Fig. 2 can be built using two NCL gates, based on the canonical SOP equations for both the rail1 and rail0 outputs, shown in equations 1 and 2, respectively, and mapping these to the set function of the gates shown in Table 2.

$$Z^1 = X^1 Y^1 \tag{1}$$

$$Z^0 = X^0 Y^0 + X^0 Y^1 + X^1 Y^0 \tag{2}$$

Therefore, output Z becomes DATA1 when both X and Y are DATA1 and it becomes DATA0 when either input is DATA0 and the other input is DATA (i.e., DATA0 or DATA1). Reference [2] elaborates on how to design more complex combinational logic circuits using NCL.

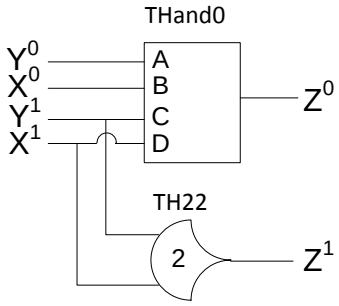


Fig. 4. NCL AND circuit

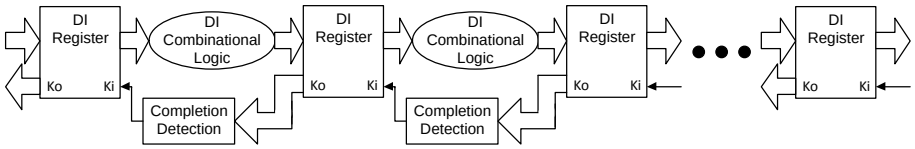


Fig. 5. NCL design framework

The NCL design framework consists of delay-insensitive (DI) Combinational Logic blocks sandwiched between DI Registers. This design framework, shown in Fig. 5, is very similar to the traditional synchronous design framework, except that Completion Detection blocks are used to synchronize data communication instead of a global clock. Completion Detection checks the output of a register to see if the previous DATA (NULL) has successfully propagated through the Combinational Logic; if so, it then allows the next NULL (DATA) to start propagating through the Combinational Logic. Ki and Ko are the handshaking signals used for requesting and acknowledging DATA and NULL. A typical DATA/NULL cycle is shown in Fig. 6. It starts with DATA propagating through a combinational block; once DATA passes the following register, the completion detection block acknowledges that DATA evaluation is finished and that NULL can now propagate. Then NULL propagates through the combinational block and clears the previous DATA; once NULL passes the register, the completion detection block acknowledges that NULL propagation is complete and allows the next DATA to start propagating through the combination block. The time period between two consequent DATA phases is called the DATA-to-DATA Cycle Time (T_{DD}), and is a measure of an NCL pipeline's throughput.

A single-bit dual-rail NCL register is shown in Fig. 7, where I^0 and I^1 are the input rails and O^0 and O^1 are the output rails. A single-bit NCL register is comprised of two TH22n gates and one inverting TH12 gate. When a combinational block is ready for DATA, Ki is asserted, allowing DATA to pass through the register; and once DATA is evaluated by the combinational block, Ki is

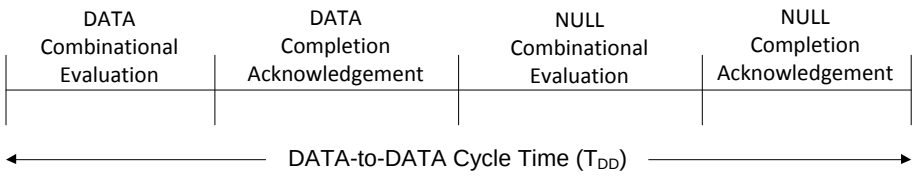


Fig. 6. DATA/NULL cycle

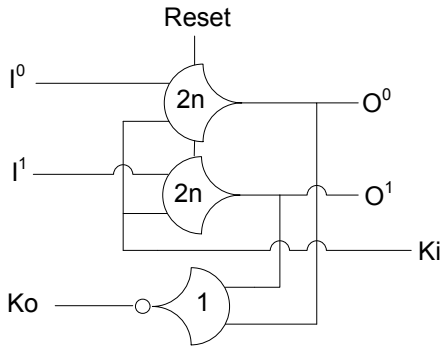


Fig. 7. A single-bit dual-rail NCL register

deasserted, allowing NULL to pass through. The *Ki* signals of a multi-bit register are all connected together and connected to the output of the completion detection block of the next register.

The completion detection block detects whether there is a complete DATA/ NULL set at the output of a register. When a register's output is NULL (i.e., both output rails in Fig. 7 are deasserted), the inverting TH12 gate is asserted to request the next DATA (rfd). When a register's output is DATA (i.e., either of the output rails in Fig. 7 is asserted), the inverting TH12 gate is deasserted to request NULL (rfn). All *Ko* outputs of a multi-bit register are input to a completion detection block that asserts its output when all *Ko* signals are rfd, and deasserts its output when all *Ko* signals are rfn. An n-bit completion detection block, shown in Fig. 8, is equivalent to an n-input C-element, comprised of THnn gates. The minimum number of levels required for a completion detection block is $\lceil \log_4 n \rceil$, where n is the number of *Ko* signals [2].

3 CMOS NCL Gate Design

3.1 Dynamic Gates

The dynamic implementation of NCL gates can be used in real-time computing applications where a minimum data rate is guaranteed so that the state informa-

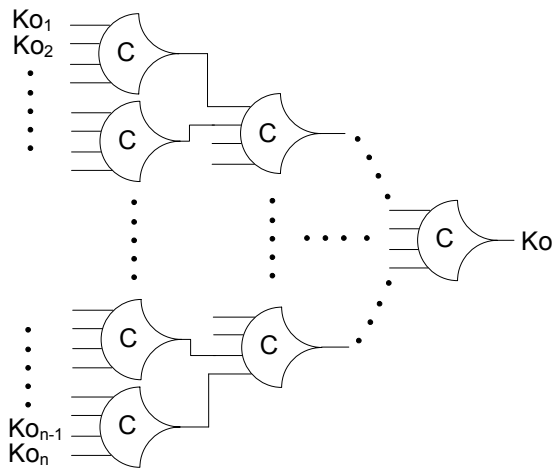


Fig. 8. NCL completion detection block

tion can be maintained on an isolated node. The structure of an NCL dynamic gate is shown in Fig. 9(a).

The set block realizes the set function of an NCL gate, such that when the set function becomes true, the set block becomes active and discharges the internal node Y, causing output Z to be asserted. Similarly, when all inputs are deasserted, the reset block becomes active and charges the internal node Y to V_{DD}, causing output Z to be deasserted. In a CMOS implementation of NCL dynamic gates, the set block is a pull-down network of NMOS transistors, derived from the equations in Table 2 for each of the 27 NCL gates. On the other hand, the reset block is always a series chain of PMOS transistors consisting of one transistor per input; therefore, NCL gates that have the same number of inputs have the same reset block. The reset function of an NCL gate with n inputs can be expressed as:

$$reset = I'_1 \bullet I'_2 \bullet \ldots \bullet I'_n \tag{3}$$

where I_n represents input n. For most NCL gates, the set and reset functions are not complements of each other, so there are times when neither the set nor reset block is active. In a dynamic implementation, when neither is active, the internal node Y will be floating, so its value will be preserved on its parasitic capacitance, $C_{parasitic}$, for a few milliseconds before its charge leaks away, enabling the NCL gate to maintain its state, but only for a finite amount of time. Therefore, once the set function becomes true and the output is asserted, it remains asserted until the reset function becomes true and deasserts the output (hysteresis behavior). Fig. 9(b) shows the dynamic implementation of a TH23 gate, whose set function is:

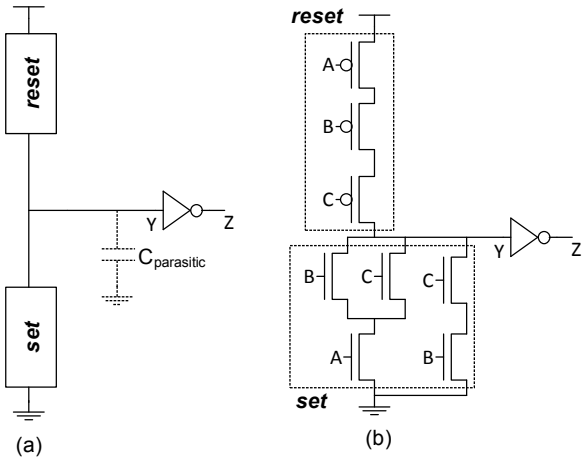


Fig. 9. (a) Structure of NCL dynamic gates (b) TH23 dynamic gate

$$F = AB + AC + BC \tag{4}$$

The set function can then be factored to reduce the number of transistors:

$$F = A\,(B + C) + BC \tag{5}$$

The NCL dynamic implementation is the smallest and fastest NCL gate style, and consumes the least amount of energy; however, since its output cannot be held indefinitely when neither set nor reset is active, it is not considered a delay-insensitive solution. Moreover, since the state information is stored on a small parasitic capacitance, it is very vulnerable to noise and charge sharing effects, although the latter can be alleviated by transistor reordering in the pull-down network [8], careful transistor sizing, and post-layout simulations. For these reasons, dynamic NCL gates are rarely used in real applications.

3.2 Semi-Static Gates

The semi-static (or pseudo-static) implementation of NCL gates utilizes feedback to maintain state information, and therefore, does not require a minimum input data rate, since it can hold the output state indefinitely. The structure of an NCL semi-static gate is shown in Fig. 10(a). In a semi-static implementation, the state information is maintained via a staticizer, in the form of a weak feedback inverter. The weak feedback inverter compensates for the leakage current that discharges the internal node Y when both set and reset blocks are inactive. This implementation is also more robust to noise and charge sharing effects because the weak feedback inverter, if carefully sized, can restore the value on the internal

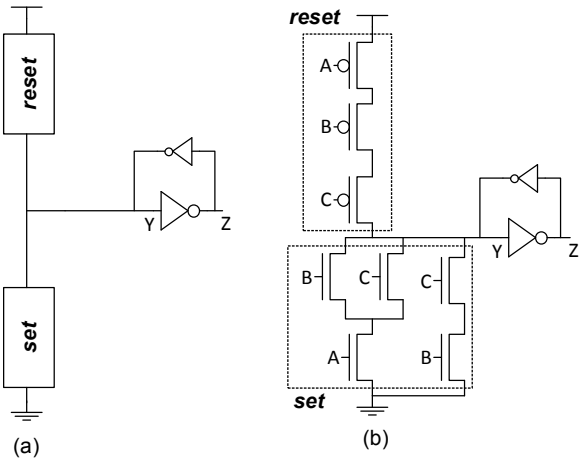


Fig. 10. (a) Structure of NCL semi-static gates (b) TH23 semi-static gate

node Y in a reasonably short time. The semi-static implementation of a TH23 gate is shown in Fig. 10(b).

Appropriate weak feedback inverter sizing is essential for correct operation of a semi-static gate. If a feedback inverter is made very weak, it will not be able to compensate for the leakage current on the internal node, and consequently, the charge on internal node Y will leak away and the gate output Z may become invalid or switch value altogether. On the other hand, a feedback inverter that is not weak enough will require a large contention current from the pull-down network (set block) or pull-up network (reset block) to switch the output value, in which case the gate's output may get stuck at a high or low value. The appropriate feedback inverter sizing also determines the performance of the gate. The weaker the feedback inverter, the more similar the semi-static implementation is to the dynamic implementation; therefore, it would be faster and would consume less energy. But, similar to the dynamic implementation, making the feedback inverter very weak makes the gate more vulnerable to noise and charge sharing effects. A more analytical discussion of semi-static C-elements, which are a special case of semi-static NCL gates, can be found in [7].

There are different ways of weakening the feedback inverter. In the standard way, shown in Fig. 11(a), usually the length of the NMOS transistor in the feedback inverter is increased. This makes the feedback inverter weak enough to be overpowered by the reset block PMOS transistor chain. The length of the PMOS transistor in the feedback inverter can also be increased or left minimum-sized since the set block pull-down network (PDN) is made of NMOS transistors and, due to the higher mobility of NMOS transistors compared to PMOS transistors, the PDN is usually able to overpower the weak inverter's PMOS transistor. Besides increasing the length of the feedback inverter's NMOS transistor, some-

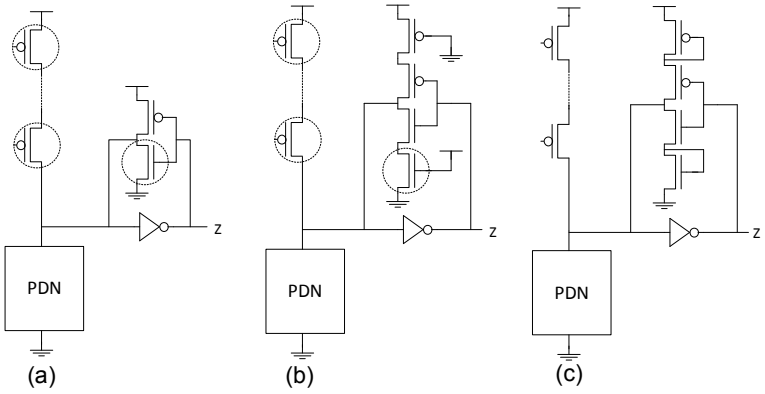


Fig. 11. Different feedback inverter weakening methods (a) standard method (b) using current limiters (c) using diode-connected current limiters

times it is better to increase the width of the reset block PMOS transistor chain. The minimum set of transistors that usually need to be sized in a standard semi-static gate is shown with dashed circles in Fig. 11(a). In order to save area, sometimes it is better to add series transistors with the feedback inverter [15]. This weakening method is shown in Fig. 11(b). Here, the added series transistors limit the current available to the feedback inverter, making it weaker. The minimum set of transistors that usually need to be sized is shown with dashed circles. Finally, one can save even more area by using diode-connected transistors in series with the feedback inverter, as shown in Fig. 11(c) [16]. Using this method, the feedback inverter becomes weak enough even with minimum-sized transistors; therefore, no sizing is usually required. Again, weakening the feedback inverter makes the gate faster and less energy hungry, but the gate becomes more vulnerable to noise and charge sharing effects, so a trade-off is involved. In practice, optimal sizing of the feedback inverter is not trivial; a more analytic sizing approach is described in [17].

Among the other implementations of the NCL gates (except dynamic implementation), semi-static gates are usually considered to be small (i.e., having minimal number of transistors) and low-energy; however, this image of semi-static NCL gates significantly depends on the weak feedback inverter sizing. The relative sizing requirements for semi-static gates makes this implementation less robust to PVT variations. Also, due to the inherent contention between the set/reset blocks and the weak feedback inverter for switching the output, this implementation is usually slower than the other implementations. This contention can be minimized by appropriate weak feedback inverter sizing, but it can never be removed. A comparison of various semi-static implementations with the other implementations can be found in [16].

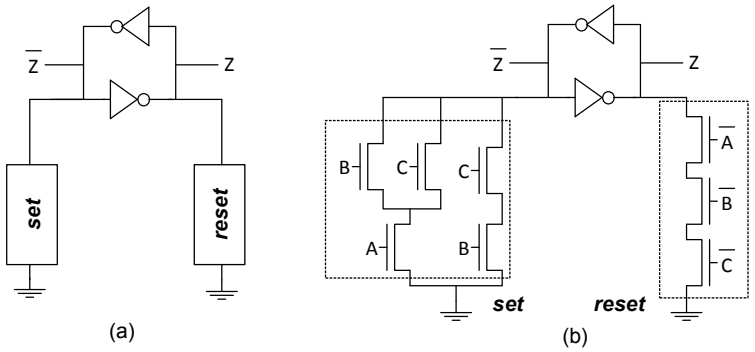


Fig. 12. (a) Structure of NCL differential gates (b) TH23 differential gate

3.3 Differential Gates

The differential implementation of NCL gates [9] [15] is most similar to a Differential Cascode Voltage-Switch Logic (DCVSL) implementation of Boolean gates [18], with the exception of using cross-coupled inverters instead of cross-coupled PMOS transistors. A differential NCL gate is shown in Fig. 12(a). The major difference between the semi-static implementation of NCL gates and the differential implementation is that the reset block is now connecting output Z to ground through a pull-down network. Due to this change in the circuit structure, the reset block should use NMOS transistors instead of PMOS transistors, and therefore requires the input complements instead. Since each differential NCL gate provides both output Z and its complement, $\overline{Z}$, no extra logic is necessary to invert inputs. Fig. 12(b) shows the differential implementation of a TH23 gate. In a differential NCL gate, asserting an output requires pulling the other output low through a pull-down network (either set or reset block); therefore, before outputs switch value, there is always a short time when both outputs become low. Since in a circuit realized with differential NCL gates, the inputs of each differential gate come from the outputs of other differential gates, this ensures that before a pull-down block becomes active, the other pull-down block becomes inactive first, therefore, no contention between pull-down blocks will ever happen.

Enabling the reset block to use higher-mobility NMOS transistors instead of PMOS transistors improves the differential implementation in several ways. These improvements are mainly because of the reset block being stronger than before so it can switch the state of the cross-coupled inverters with less effort. The immediate result being that the differential implementation is usually faster than the semi-static implementation. Also, less aggressive sizing is now required, so the differential implementation is usually smaller than the semi-static implementation. In fact, a differential NCL gate can usually use all minimum-sized transistors and still function correctly. In addition, due to the symmetry of the differential implementation, the cross-coupled inverters are usually sized equally

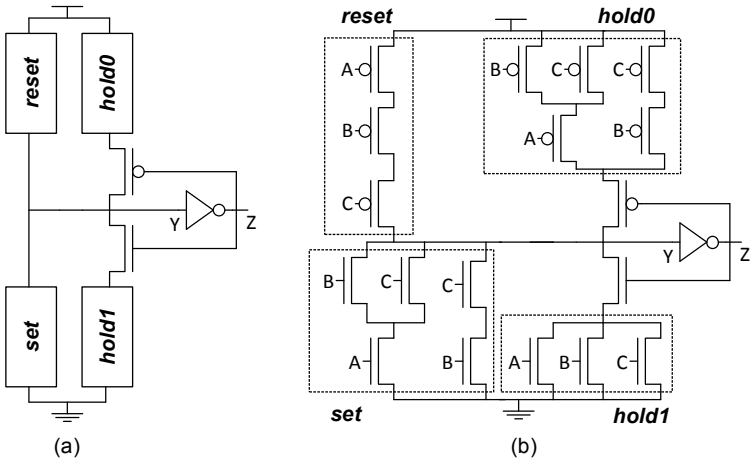


Fig. 13. (a) Structure of NCL static gates (b) TH23 static gate

and the whole structure is therefore less sensitive to sizing, and consequently, more robust to PVT variations.

3.4 Static Gates

All the CMOS NCL gate implementations discussed so far rely on either a parasitic capacitance to maintain state information, such as in the dynamic implementation, or rely on a simple feedback mechanism via an inverter, such as in the semi-static and differential implementations. As discussed, relying on the parasitic capacitance makes NCL gates vulnerable to leakage, noise, and charge sharing problems, and eliminates their delay-insensitivity, while a feedback inverter slows down the gates due to the intrinsic switching contention involved. A static NCL gate implementation removes all these drawbacks, offering faster and more reliable operation.

As depicted in Fig. 13(a), static NCL gates are comprised of 4 transistor networks: set, reset, hold1, and hold0. Similar to other implementations, the set block determines the gate's functionality as one of the 27 NCL gates. Once the set function becomes true, the output is asserted. The output then remains asserted through the hold1 block until all inputs are deasserted. The hold1 block is simply made by ORing all inputs together; therefore, it is the same for gates having the same number of inputs. The hold1 function of a static NCL gate with n inputs can be expressed as:

$$hold1 = I_1 + I_2 + \ldots + I_n \tag{6}$$

where I_n represents input n. Since both set and hold1 blocks contribute to asserting Z and maintaining its assertion, the set equation of a static NCL gate can be described as:

$$Z = set + \left(Z^{-} \bullet hold1\right) \tag{7}$$

Where Z^{-} is the previous output value of the gate and Z is the new output value. As an example, as depicted in Fig. 13(b), the TH23 gate has the following set and hold1 functions: $set = A\,(B + C) + BC$; $hold1 = A + B + C$.

In order to implement a static NCL gate in CMOS technology, the complement of Z is also required. The complement of Z, denoted as Z', is realized with reset and hold0 blocks. The reset block, similar to the previous implementations, consists of all complemented inputs ANDed together. Once all inputs are deasserted, the reset block becomes active and deasserts the output. The output then stays deasserted through the hold0 block until new input values activate the set block to assert the output again. The reset equation of a static NCL gate can therefore be described as:

$$Z' = reset + \left(Z^{-'} \bullet hold0\right) \tag{8}$$

It can be proven that the following relations exist between set, reset, hold1, and hold0 functions:

$$set = hold0' \tag{9}$$

$$reset = hold1' \tag{10}$$

Equation 10 can be directly inferred from the definition of reset and hold1 functions and DeMorgan's law; and equation 9 is the logical consequence of the fact that in a static implementation, the pull-up and pull-down networks must be complements of each other to avoid a short-circuit path or a floating node. According to the above equations, the equations for a static TH23 gate are: $hold0 = A^{'}(B^{'} + C^{'}) + B'C'$ and $reset = A'B'C'$. The CMOS implementation of the static TH23 gate is shown in Fig. 13(b).

In contrast to the semi-static implementation, the static implementation of NCL gates is faster since output switching does not involve contention. It is also very robust to leakage, noise, and charge sharing since for any input combination the internal node Y is connected to either V_{DD} through the pull-up network, or GND through the pull-down network. Moreover, the switching threshold of static gates being typically around $V_{DD}/2$ adds to their noise immunity. Additionally, transistor sizing in a static implementation only impacts its performance, not its functionality; therefore, the static implementation is very robust to PVT variations. Its main drawback is the area overhead from adding hold0 and hold1 blocks. For example, in the case of the TH23 gate, the static implementation shown in Fig. 13(b) requires 20 transistors, while the semi-static and differential implementations only require 12 transistors. A more analytical discussion of static C-elements, that are a special case of static NCL gates, can be found in [7].

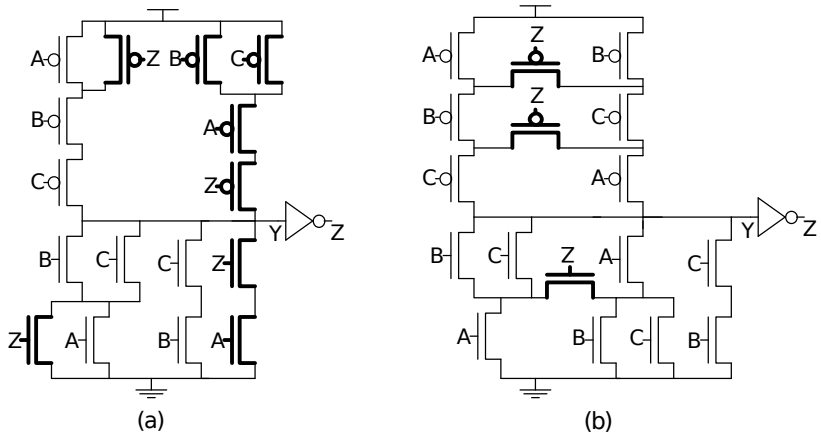


Fig. 14. (a) Original TH23 static gate (b) Proposed TH23 static gate

4 New Static Gates

In the previous section, area overhead was mentioned to be the main drawback of static NCL gates; however, sometimes it is possible to share transistors between each pair of pull-up (reset and hold0) or pull-down (set and hold1) networks to reduce area. For example, the direct static implementation of the TH23 gate, shown in Fig. 13(b), consists of 20 transistors; but after sharing transistors, the optimized implementation only requires 18 transistors, as shown in Fig. 14(a). There are two types of transistors in a static NCL gate: switchers, which contribute to switching the gate's output, and keepers, which only contribute to retaining the gate's state when neither set nor reset blocks are active. In Fig. 14(a) the keepers are shown in boldface.

The development of the new static NCL gates is inspired by the observation that in a traditional static NCL gate, the hold0 and hold1 transistor networks are only used for retaining the gate's state when neither set nor reset functions are true. In other words, the hold0 and hold1 transistor networks only contribute to holding the output state but not switching it. The idea behind the new static NCL gates is to integrate the set and hold1 transistor networks as well as the reset and hold0 transistor networks into a single composite transistor network such that it involves more transistors in output switching. Fig. 14(b) shows the application of this idea to the TH23 gate. The new gate structure differs from the original one in two ways. First, the reset network has been duplicated and rearranged, and then some extra PMOS transistors are added to realize the hold0 function by connecting appropriate nodes of the two PMOS transistor chains. Second, the hold1 function is realized by duplicating and flipping a portion of the set network and then connecting the middle nodes with an NMOS transistor. The new gate consists of 19 transistors, which is one transistor more than the original one; however, compared to the original gate, the number of keepers has

been reduced from 8 to 3 (shown in boldface), while the number of switchers has increased from 8 to 14, resulting in faster switching compared to the original gate.

The correctness of the new gate structure can be easily proved using Boolean algebra. For the pull-up network, when $Z = 1$ both PMOS keepers are off so the function of the pull-up network can be expressed as:

$$A'B'C' + B'C'A' = A'B'C' \tag{11}$$

which is the same as the function of the reset block, and when $Z = 0$ both PMOS keepers turn on so the function of the pull-up network can be expressed as:

$$(A' + B')(B' + C')(C' + A') = A'(B' + C') + B'C' \tag{12}$$

which is the same as the function of the hold0 block. Similarly, for the pull-down network, when $Z = 0$ the NMOS keeper is off so the function of the pull-down network can be expressed as:

$$(B + C)A + A(B + C) + BC = A(B + C) + BC \tag{13}$$

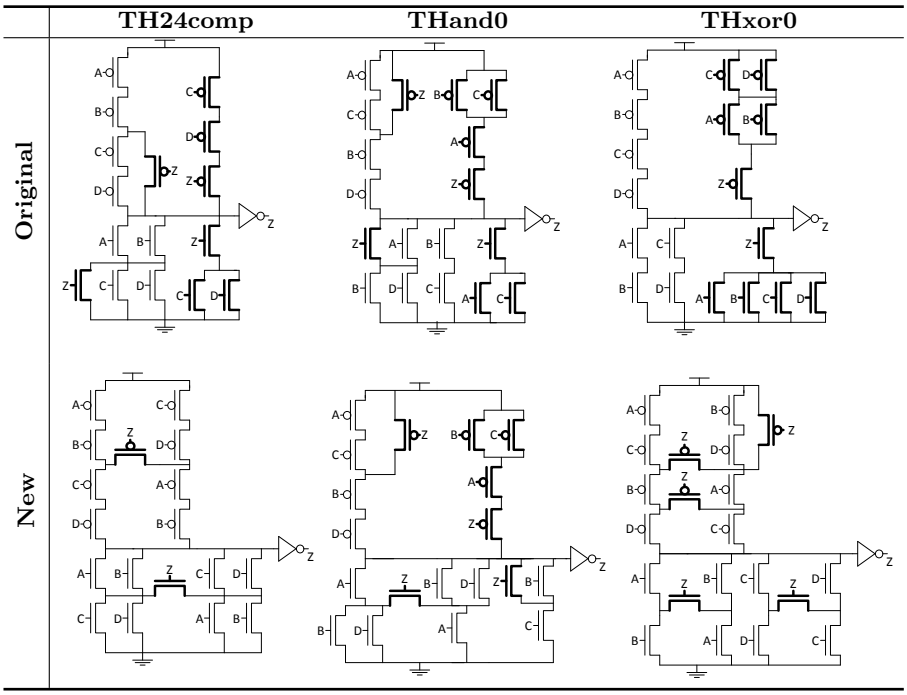
which is the same as the function of the set block, and when $Z = 1$ the NMOS keeper turns on so the function of the pull-down network can be expressed as:

$$(B + C + A)(A + B + C) + BC = A + B + C \tag{14}$$

which is the same as the function of the hold1 block.

The new gate structure also speeds-up output switching in one additional way. Careful investigation shows that the number of transistors in a series chain for holding the gate's state when neither set nor reset functions are true has increased. For example, the hold0 path that was originally going through Z→B→C is now going through B→Z→B→C, which is one transistor longer than the original path. Similarly, the hold1 path that was originally going through B→Z is now going through B→Z→B. Hence, the new gate structure's transistor chain length for hold0 and hold1 paths has increased by one transistor. This is equivalent to weaker hold0 and hold1 networks (i.e., the paths have higher resistance); therefore, the set and reset networks can switch the gate's output faster. This might look confusing since, as mentioned before, the set and hold0 (and similarly reset and hold1) networks are complements of each other such that they are never asserted simultaneously; therefore, the set network never needs to overpower the hold0 network (or reset network overpower hold1). However, since at the time of switching there is a short moment when both pull-up and pull-down networks turn on and create a short-circuit path from V_{DD} to GND (similar to static Boolean gates), a pull-up (pull-down) network with higher resistance, and consequently less current flow, helps the pull-down (pull-up) network pull the internal node to GND (V_{DD}) with less effort, resulting in faster switching.

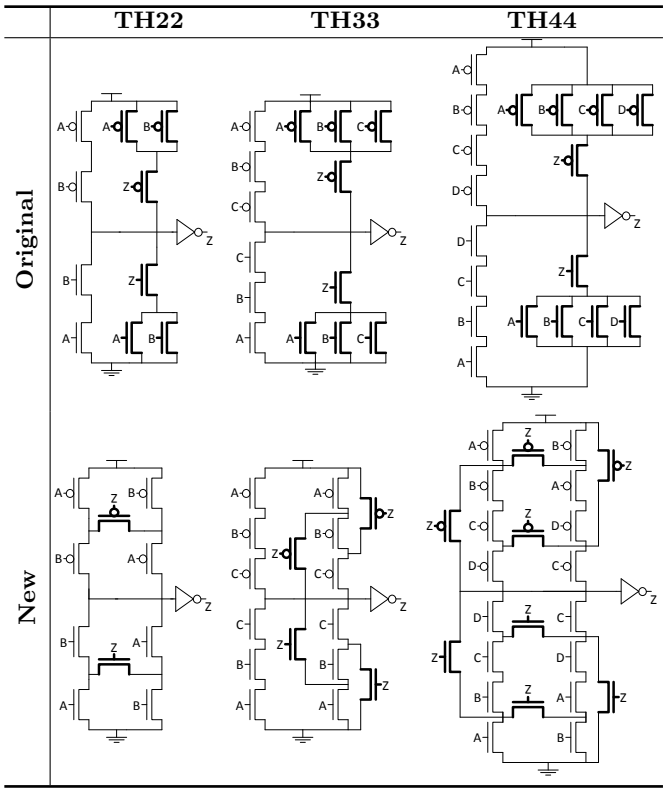
Table 3. Original complex static gates versus the new versions



The last interesting feature of the new static gate structure is that it is more symmetric than the original structure, resulting in closer output rise/fall times.

Converting traditional static gates to the new ones is not always easy and straightforward, especially for more complex gates. Additionally, although in the case of the TH23 gate there was only one transistor overhead for the new gate style, sometimes area overhead is more than a few transistors, resulting in an area versus delay tradeoff. Based on how complex the gate is, sometimes it is possible to partially apply this technique (e.g., to only the pull-up or pull-down network, or even just a portion of them). Table 3 shows the design of a few complex NCL gates using both the original and the new method, with keeper transistors shown in boldface. The first row of gates pertains to the original design, while the second row shows the new designs. Comparing the new versions with the original ones shows that the number of keepers has been reduced in all the new versions. For the THand0 gate, the pull-up network could not be converted to utilize fewer keepers, so it is not changed. Table 4 compares the original and the new static C-elements. For the TH22 gate, the new version is equivalent to the symmetric C-element design in [12]. As mentioned before, converting the original static design to the new one is not always easy and does not follow strict rules. However, the following guidelines are helpful:

Table 4. Original C-elements versus the new versions



1. Remove the hold1/hold0 networks from the original design
2. Duplicate the set/reset networks
3. Rearrange/flip the duplicated networks and connect their internal nodes to the original network by adding keepers such that the hold1/hold0 functionality is ensured
4. If the new structure requires more keepers in the pull-up or pull-down networks then try to apply this technique partially or just use the original design

Table 5 shows a comparison between the new gates and the original ones in terms of delay, area, and energy consumption. The gates are implemented and simulated using the IBM CMOS9SF 90nm CMOS process. All simulations are performed under the following conditions: typical process corner, nominal power supply voltage of 1.2 V, temperature of 27 °C, and capacitive load of 10 fF. Both high-to-low (TPHL) and low-to-high (TPLH) propagation delays are included in this table. The simulation results show that on average the new gates offer 9%

Table 5. Comparison between original and new static gate styles

	TPLH [ps]			TPHL [ps]			Energy [fJ]			Transistors		
Gate	**New**	**Original**	**Improve**	**New**	**Original**	**Improve**	**New**	**Original**	**Improve**	**New**	**Original**	**Overhead**
TH22	155	168	7.70%	83	123	32.20%	18.4	18.5	0.60%	12	12	0
TH33	174	197	11.40%	128	193	33.90%	20.2	19.4	-4.50%	18	16	2
TH44	198	226	12.00%	183	262	30.20%	23.1	20.2	-14.30%	26	20	6
TH44w2	200	214	6.40%	179	198	9.70%	22.3	20.6	-7.80%	25	22	3
TH23	172	180	4.50%	115	207	44.30%	20	20.3	1.40%	19	18	1
TH34w2	191	194	1.70%	150	222	32.10%	21.6	20.3	-6.30%	27	22	5
TH24comp	160	188	14.80%	134	217	38.20%	19.8	20.4	3.00%	20	18	2
THxor0	167	189	12.00%	142	255	44.20%	20.4	20.7	1.80%	23	20	3
TH22n	167	189	11.50%	86	138	37.80%	18.7	18.9	1.30%	16	16	0
THand0	180	195	7.50%	195	252	22.80%	20.6	21.2	2.80%	21	20	1
Average	177	194	9.00%	139	207	32.50%	20.5	20.1	-2.20%	20.7	18.4	2.3

improvement in TPLH and 32.5% improvement in TPHL, with a 2.2% increase in energy consumption and an average of 2.3 additional transistors per gate. For the results in Table 5, all transistors are minimum-sized and the results are averaged over all possible input combinations.

5 Sizing New Static Gates

The new static gate design speeds up switching with a reasonable area overhead (2.3 transistors per gate). However, the new static gates have the potential to be smaller than the original static gates when both gate styles are properly sized for faster switching. For example, assume that the TH23 gates in Fig. 14 need to be sized. Since only the switchers are responsible for output switching, one can double their width while allowing the keepers to stay minimum-sized. This is shown in Fig. 15. The keepers are all minimum-sized (1X) in this figure, so their size is not shown. The size of the switchers in the original static gate, however, has doubled, even for the parallel switchers, in order to account for when only one of them contributes to output switching. The switchers in the new static gate are then sized such that they provide the same pull-up/pull-down resistance as the original static gate on the switching paths. Finally, the output inverter for each gate can be sized such that it offers a balanced output rise/fall time targeting a certain output load. Assuming that the output inverters would have almost similar (or comparable) sizes, the new static gate would be smaller than the original one, shown by adding up the size of transistors for each gate. In the case of the TH23 gate, the original gate size is 24X while the new gate size is 19X.

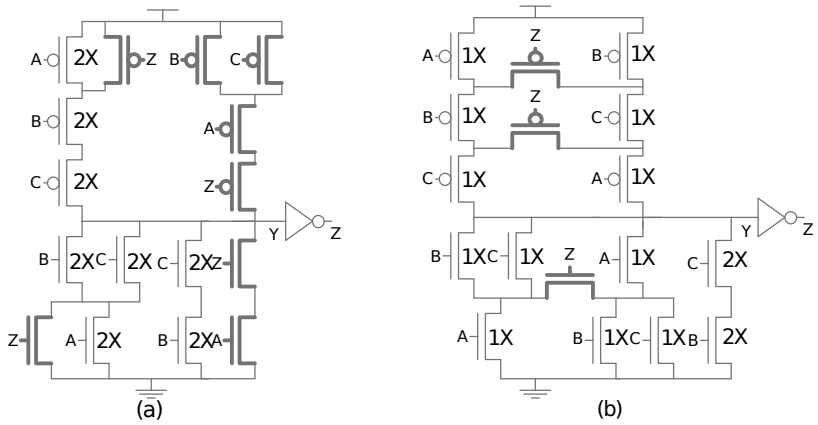


Fig. 15. A sizing example for (a) original (b) new static TH23 gates

Table 6. Comparison of NCL multipliers realized with different static gate styles

Gate Style	Original Minimum	New Minimum	Original Sized	New Sized
Delay per operation [ns]	1.45	1.05	1.29	0.98
Energy per operation [pJ]	1.29	1.28	2.62	2.34
Area [μm^2]	59.4	62.6	122.2	111.3
Minimum VDD [V]	0.25	0.26	0.22	0.27

6 Simulation Results

In order to measure the performance of the new static gate style at the circuit level and compare it to the original static gate style, a delay-insensitive NCL 4×4 pipelined multiplier [19] was simulated at the transistor level using each gate style. The results, averaged over all 256 input combinations, are shown in Table 6. All simulations are performed under the following conditions: typical process corner, nominal power supply voltage of 1.2 V, and temperature of 27 °C. In order to measure the minimum power supply voltage for each variation of multiplier, V_{DD} is dropped to the point where the NCL multiplier outputs wrong data or completely stalls due to deadlock [2].

For the minimum-sized gates, the multiplier using the new gate style is 27% faster and requires 5% more area, with approximately the same energy per operation and the same low-voltage operation capability. Table 6 also shows the comparison between the multipliers utilizing sized static gates. The multiplier realized with the new sized gates is now both faster (24%) and smaller (8%) than the multiplier using the original sized gates. In addition, the energy per op-

eration is now 10% lower, but the minimum power supply voltage has increased by 22%.

7 Conclusion

In this chapter, different CMOS implementations of NCL gates were introduced and their trade-offs were discussed. It was shown that each implementation offers some advantages for designing NCL circuits. Omitting the dynamic implementation, since it is not delay-insensitive, comparison of the other implementations shows that static gates tend to be faster and more robust to noise and PVT variations, while semi-static gates are more energy efficient, and differential gates are more area efficient.

Additionally, a new approach to designing static NCL gates was introduced. The new gate style was compared to the original style in terms of delay, energy, and area, showing that the new gate style is significantly faster, while requiring slightly more area and energy for minimum sized gates. After sizing the gates, it was shown that the new gate style is faster, and requires less area and energy. These conclusions are supported by transistor-level simulation of a delay-insensitive NCL pipelined multiplier, to compare the different gate styles on a larger scale.

References

1. Beerel, P.A., Ozdag, R.O., Ferretti, M.: A designer's guide to asynchronous VLSI. Cambridge University Press (2010)
2. Smith, S.C., Di, J.: Designing asynchronous circuits using NULL Convention Logic (NCL). Synthesis Lectures on Digital Circuits and Systems, Vol. 4/1. Morgan & Claypool Publishers (2009)
3. Fant, K.M.: Logically Determined Design: Clockless System Design with NULL Convention Logic. Wiley-Interscience (2005)
4. Ligthart, M., Fant, K., Smith, R., Taubin, A., Kondratyev, A.: Asynchronous design using commercial HDL synthesis tools. Advanced Research in Asynchronous Circ. and Syst., Proc. Sixth Int. Symp. on (Apr. 2000) 114–125
5. McCardle, J., Chester, D.: Measuring an asynchronous processor's power and noise. Proc. Synopsys Users Group Conf. (SNUG). Synopsys, Mountain View, Calif. (2001) 66–70
6. Parsan, F.A., Al-Assadi, W.K., Smith, S.C.: Gate Mapping Automation for Asynchronous NULL Convention Logic Circuits. IEEE Trans. Very Large Scale Integr. (VLSI) Syst., to be published, available: http://dx.doi.org/10.1109/TVLSI.2012.2231889
7. Shams, M., Ebergen, J.C., Elmasry, M.I.: Modeling and comparing CMOS implementations of the C-element. Very Large Scale Integ. (VLSI) Syst., IEEE Trans. on 6 (Dec. 1998) 563–567
8. Sobelman, G.E., Fant, K.: CMOS circuit design of threshold gates with hysteresis. Circ. and Syst., Proc. of the IEEE Int. Symp. on, Vol. 2 (Jun. 1998) 61–64 vol.62
9. Yancey, S., Smith, S.C.: A differential design for C-elements and NCL gates. Circ. and Syst. , 53rd IEEE Int. Midwest Symp. on (Aug. 2010) 632–635

10. Parsan, F.A., Smith, S.C.: CMOS implementation of static threshold gates with hysteresis: A new approach. VLSI and System-on-Chip (VLSI-SoC), 2012 IEEE/IFIP 20th International Conference on (Oct. 2012) 41-45
11. Seitz, C.L.: System timing. Introduction to VLSI Systems. MA: Addison-Wesley (1980) 218–262
12. Berkel, K.V.: Beware the isochronic fork. Integr. VLSI J. 13 (Jun. 1992) 103–128
13. Verhoeff, T.: Delay-insensitive codes – an overview. Distributed Computing 3 (1988) 1–8
14. Muller, D.E.: Asynchronous logics and application to information processing. Stanford, CA: Stanford Univ. Press (1963)
15. Shams, M., Ebergen, J.C., Elmasry, M.I.: Optimizing CMOS implementations of the C-element. Comp. Design, Proc. of IEEE Int. Conf. on (Oct. 1997) 700-705
16. Parsan, F.A., Smith, S.C.: CMOS implementation comparison of NCL gates. Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on (Aug. 2012) 394-397
17. Li, D., Mazumder, P.: On circuit techniques to improve noise immunity of CMOS dynamic logic. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 12 (2004) 910-925
18. Heller, L., Griffin, W., Davis, J., Thoma, N.: Cascode voltage switch logic: A differential CMOS logic family. Solid-State Cir. Conf. Digest of Tech. Papers. IEEE Int., Vol. XXVII (Feb. 1984) 16-17
19. Smith, S.C., DeMara, R.F., Yuan, J.S., Hagedorn, M., Ferguson, D.: Delay-insensitive gate-level pipelining. Integ., the VLSI Journal 30 (Oct. 2001) 103–131