1-1-2007

Design of an FPGA Logic Element for Implementing Asynchronous NULL Convention Logic Circuits

Scott C. Smith
Missouri University of Science and Technology

Design of an FPGA Logic Element for Implementing Asynchronous NULL Convention Logic Circuits

Scott C. Smith, *Senior Member, IEEE*

***Abstract*—Two versions of a reconfigurable logic element are developed for use in constructing a NULL convention logic (NCL) field-programmable gate array (FPGA): one with extra embedded registration capability, which requires additional area, and one without. Both versions can be configured as any of the 27 fundamental NCL gates, including resettable and inverting variations, and both can utilize embedded registration for gates with three or fewer inputs; however, only the version with the additional embedded registration capability can utilize embedded registration with four-input gates. These two approaches are compared with each other and with an existing approach, showing that both versions developed herein yield a more area efficient NCL circuit implementation, compared to the previous work. The two FPGA logic elements are simulated at the transistor level using the 1.8-V, 180-nm TSMC CMOS process.**

***Index Terms*—Asynchronous logic design, delay-insensitive circuits, field-programmable gate array (FPGA), NULL convention logic (NCL), reconfigurable logic.**

I. Introduction

THOUGH synchronous circuit design presently dominates the semiconductor design industry, there are major limiting factors to this design approach, including clock distribution, increasing clock rates, decreasing feature size, and excessive power consumption. Correct-by-construction asynchronous circuits, such as NULL convention logic (NCL), have been demonstrated to require less power, generate less noise, produce less electromagnetic interference (EMI), and provide for easier component reuse compared to their synchronous counterparts, without compromising performance [1]. However, NCL circuits are not composed of the same fundamental gates as Boolean circuits; all NCL gates are designed with hysteresis state-holding behavior, such that once a gate's output is asserted, it remains asserted until all gate inputs are deasserted [2]. Therefore, NCL circuits must be designed as application-specific integrated circuits (ASICs), and cannot easily be implemented on standard field-programmable gate arrays (FPGAs) without compromising delay insensitivity [3].

NCL gates can be implemented with Boolean logic by using an set-reset (SR) latch to achieve the hysteresis behavior [4]; however, this implementation yields potential race conditions, which may cause the SR latch to temporarily enter the metastable state or produce the incorrect output all together [3]. This becomes even more of a problem when an NCL circuit is implemented on a standard FPGA, since a single NCL gate may be comprised of multiple Boolean gates, such that the single NCL gate could be distributed over many configurable logic blocks (CLBs), yielding nonisochronic forks [5], [6], or problematic orphans [7], and thus further compromising delay insensitivity. To illustrate this point, a few NCL circuits were synthesized to a Xilinx Spartan 2 FPGA using the Mentor Graphics Leonardo Spectrum tool. For small circuits, like a full-adder [8], the FPGA design worked correctly; however, for larger circuits, such as a nonpipelined 4-bit $\times$ 4-bit dual-rail unsigned multiplier [9], the FPGA design malfunctioned. Specifically, 20 out of the 256 possible operations generated an incorrect output, due to both rails of at least one dual-rail output signal being simultaneously asserted, which is an illegal state. So, as expected, the race conditions caused the standard FPGA implementation to function incorrectly for some input combinations; hence, clocked Boolean FPGAs are unsuitable for implementing NCL designs, thus justifying the need for a completely asynchronous NCL FPGA.

Through extensive timing analysis and careful FPGA placement, large NCL circuits could be successfully implemented on standard FPGAs [10]; however, there would be no advantage to this approach. A major advantage of NCL is its delay insensitivity, making timing analysis unnecessary. However, this approach would require even more timing analysis than for synchronous design. Furthermore, since an NCL gate consists of many Boolean gates, this approach would require substantially more area than an equivalent synchronous design and would be much slower. Hence, this approach is purely academic and has no practical application.

The size of FPGAs is now more than one million equivalent gates, making them a viable alternative to custom design for all but the most complex processors. FPGAs are relatively low cost and are reconfigurable, making them perfect for prototyping, as well as for implementing the final design, especially for low volume production. To compete with this cheap, reconfigurable synchronous implementation, an NCL-specific FPGA is needed, such that NCL circuits can be efficiently implemented without necessitating a prohibitively expensive full custom design. This will become increasingly important as asynchronous paradigms become more widely used in the industry to increase circuit robustness, decrease power, and alleviate many clock-related issues, as predicted by the International Technology Roadmap for Semiconductors (ITRS). The 2005 ITRS estimates that asynchronous circuits will account for 19% of chip area within the next five years, and 30% of chip area within the next ten years.

Manuscript received August 10, 2006.

The author is with the Department of Electrical and Computer Engineering, University of Missouri-Rolla, Rolla, MO 65409 USA (e-mail: smithsco@umr.edu).

Digital Object Identifier 10.1109/TVLSI.2007.898726

This paper is partitioned into seven sections. Section II provides a brief overview of NCL. Section III describes the previous work on asynchronous FPGA design. Section IV presents the design of a reconfigurable NCL logic element (LE). Section V presents an alternative design of a reconfigurable NCL LE that includes extra embedded registration capability. Section VI compares the two LEs developed herein with each other and with the previous work in [11] and Section VII provides conclusions and directions for future work.

II. NCL Overview

NCL is a self-timed logic paradigm in which control is inherent in each datum. NCL follows the so-called weak conditions of Seitz's delay-insensitive signaling scheme [12]. Like other delay-insensitive logic methods, the NCL paradigm assumes that forks in wires are isochronic [5], [6]. Various aspects of the paradigm, including the NULL (or spacer) logic state from which NCL derives its name, have origins in Muller's work on speed-independent circuits in the 1950s and 1960s [13].

A. Delay Insensitivity

NCL uses symbolic completeness of expression to achieve delay insensitive behavior. A symbolically complete expression depends only on the relationships of the symbols present in the expression without reference to their time of evaluation [14]. In particular, dual-rail and quad-rail signals, or other mutually exclusive assertion groups (MEAGs), can incorporate data and control information into one mixed-signal path to eliminate time reference. A dual-rail signal D consists of two mutually exclusive wires D^0 and D^1 which may assume any value from the set $\{\mathrm{DATA0}, \mathrm{DATA1}, \mathrm{NULL}\}$; likewise, a quad-rail signal consists of four mutually exclusive wires that represent two bits. For NCL and other circuits to be delay insensitive, assuming isochronic wire forks [5], [6], they must meet the input completeness and observability criteria [8].

Completeness of input requires that all the outputs of a combinational circuit may not transition from NULL to DATA until all inputs have transitioned from NULL to DATA, and that all the outputs of a combinational 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, according to Seitz's weak conditions [12], 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. Observability requires that no *orphans* may propagate through a gate [7]. 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 [5], [6], as long as they are not allowed to cross a gate boundary. This observability condition, also referred to as indicatability or stability, ensures that every gate transition is observable at the output, which means that every gate that transitions is necessary to transition at least one of the outputs. The observability condition can be relaxed through orphan analysis and still achieve self-timed behavior; however, this requires some delay analysis [7]. Furthermore, when circuits use the bit-wise completion strategy with selective input incomplete components, they must also adhere to the completion completeness criterion [15], which requires that completion signals only be generated such that no two adjacent DATA wavefronts can interact within any combinational component.

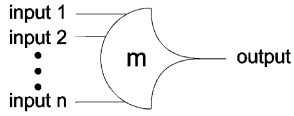

Fig. 1. THmn threshold gate.

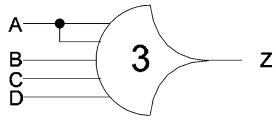

Fig. 2. TH34w2 threshold gate: $Z = AB + AC + AD + BCD$.

Most multirail delay insensitive systems [12], [16]–[19], including NCL, have at least two register stages, one at both the input and the output. Two adjacent register stages interact through request and acknowledge lines K_i and K_o, respectively, to prevent the current DATA wavefront from overwriting the previous DATA wavefront by ensuring that the two are always separated by a NULL wavefront.

B. Logic Gates

NCL differs from other delay insensitive paradigms [12], [16]–[19], which use only one type of state holding gate, the C-element [13]. A C-element behaves as follows: when all inputs assume the same value, the output assumes this value; otherwise, the output does not change. On the other hand, all NCL gates are state holding. NCL uses threshold gates as its basic logic elements [2]. The primary type of threshold gate, shown in Fig. 1, is the *THmn gate*, where $1 \leq m \leq n$. THmn gates have n single-wire inputs, where at least m of the n inputs must be asserted before the single wire output will become asserted. In a THmn 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. NCL circuits are designed using a threshold gate network for each output rail (i.e., two threshold gate networks would be required for a dual-rail signal D, one for D^0, and another for D^1).

Another type of threshold gate is referred to as a weighted threshold gate, denoted as $THmnW w_1 w_2, \ldots, w_R$. Weighted threshold gates have an integer value $m \geq w_R > 1$ applied to *inputR*. Here $1 \leq R < n$; where n is the number of inputs; m is the gate's threshold; and $w_1, w_2, \ldots, w_R$, each > 1, are the integer weights of *input1, input2*, ..., *inputR*, respectively. For example, consider the TH34W2 gate shown in Fig. 2, 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 along with any other input B, C, or D.

NCL threshold gates are designed with hysteresis state holding capability, such that all asserted inputs must be deasserted before the output will be deasserted. Hysteresis ensures

TABLE I
27 FUNDAMENTAL NCL GATES

NCL Gate	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

a complete transition of inputs back to NULL before asserting the output associated with the next wavefront of input data. Therefore, a THnn gate is equivalent to an n-input C-element and a TH1n gate is equivalent to an n-input OR gate. There are 27 fundamental NCL gates, constituting the set of all functions consisting of four or fewer variables [8], as shown in Table I. Since each rail of an NCL signal is considered a separate variable or value, a four variable function is not the same as a function of four literals, which would normally consist of eight variables or values, assuming dual-rail signals.

NCL threshold gate variations include *resetting* THnn and *inverting* TH1n gates. Circuit diagrams designate resettable gates by either a d or an n appearing inside the gate, along with the gate's threshold. d denotes the gate as being reset to logic 1; n, to logic 0. Both resettable and inverting gates are used in the design of delay insensitive registers [14].

III. PREVIOUS WORK

There have been a number of asynchronous FPGAs developed over the past 10+ years [20]–[27], [11]. MONTAGE [20] was developed to support both synchronous and asynchronous circuits. PGA-STC [21] and STACC [22] both target bundled data systems in which there are separate data and control paths where the delay in the data path must be matched in the control path. To implement this delay matching, both architectures include some sort of programmable delay element. MONTAGE [20] and PGA-STC [21] both use a lookup table (LUT)-based design, where the output is fed back as one of the inputs, to implement the C-element's state holding capability. STACC [22] is based on fine grain FPGA architectures where the global clock is replaced by an array of timing cells that generate local register control signals. These systems rely heavily on the placement and routing tools to yield a functional FPGA circuit, where all delays are correctly matched.

Another type of asynchronous FPGA uses a programmable phased logic cell [23], [24]. Phased logic converts a traditional synchronous gate-level circuit into a delay insensitive circuit by replacing each conventional synchronous gate with its corresponding phased logic gate, and then augmenting the new network with additional control signals. Since phased logic circuitry is derived directly from its equivalent synchronous design, and not created independently, it does not have the same potential for optimization as does NCL. Furthermore, the phased logic paradigm has been developed mainly for easing the timing constraints of synchronous designs, not for obtaining speed and power benefits [28], whereas these are main concerns of other asynchronous paradigms.

Two other asynchronous FPGAs [25], [26] are both based on the precharge half-buffer (PCHB) logic family to implement quasi delay insensitive circuits. Both use data-driven decomposition (DDD) [29] to convert a high-level circuit description into PCHB circuits. The basic template for the individual logic blocks of these two architectures is the same; however, the choice of cells and clustered architectures are different. Reference [25] targets arithmetic or DSP computations, whereas [26] targets general purpose circuits and microprocessors. Using [25] to implement general purpose circuits or [26] to implement arithmetic circuits may lead to inefficient resource utilization, where large parts of clustered logic blocks are not utilized. Another drawback to the PCHB approach is that communication consumes significantly more energy than does computation [26]. This is not the case with NCL. The FPGA in [25] utilizes a very fine grained pipelined architecture consisting of clustered logic blocks, which contain a 4-dual-rail input LUT along with other asynchronous components, in an island-style interconnect topology with connection boxes and switch boxes, where each interconnect consists of three wires and a dual-rail signal with single wire acknowledge. The design in [25] is the basis for Achronix Semiconductor's ULTRA line of FPGAs, which yield performance in the 1.6–2.2 GHz range, while consuming significantly less power than today's leading FPGAs. Achronix FPGAs are packaged with software tools to convert synchronous designs to asynchronous logic, such that the end user need not be familiar with asynchronous circuit design.

In an effort to design a delay insensitive, reconfigurable logic device, Theseus Logic developed an FPGA-based on the Atmel AT40K family [27]. The design involved replacing the D-type flip-flop within each logic block with a threshold configurable NCL THm4 gate, and removing the associated clock trees from the original design. Atmel's routing algorithm for this chip was then modified to convert an NCL gate-level schematic to a bitstream to program the FPGA. This method is advantageous in that it reuses a proven architecture, but the design only utilizes a fraction of the NCL threshold gates, thus increasing area and

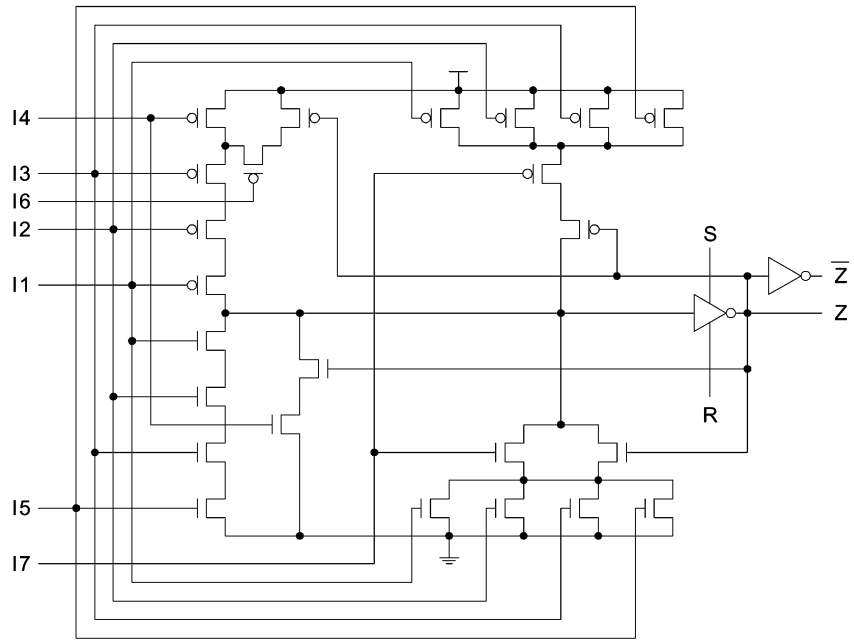

Fig. 3. Reconfigurable NCL gate [11].

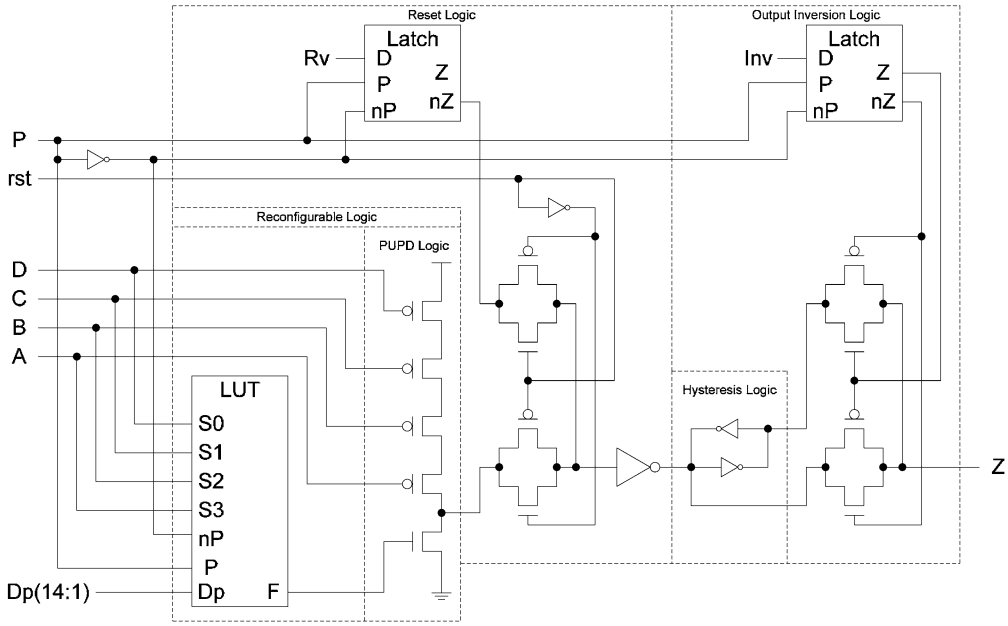

Fig. 4. Reconfigurable NCL LE without extra embedded registration.

delay for realizing most nontrivial NCL circuits. It also has the disadvantage of being unable to use all of the LUTs in the FPGA, thus resulting in inefficient resource utilization [27]. A more efficient configurable logic element for an NCL FPGA was presented in [11]. The LE, shown in Fig. 3, consists of 32 transistors, and is capable of being programmed as 8 of the 27 fundamental NCL gates (TH12, TH13, TH14, TH22, TH33, TH44, TH23, and TH34w2), including both resetting and inverting variations, and can be programmed as an inverter. The gate has seven data inputs (I1–I7), a set and reset input S and R, respectively, and two outputs Z and its complement. The following configuration would implement a TH23 gate (i.e., $Z = AB + AC + BC) : I1 = A, I2 = B, I3 = A, I4 = C, I5 = A, I6 = 0$, and $I7 = C$. However, this method still utilizes less than 1/3 of the available 27 fundamental NCL gates, which can lead to increased area and delay for implementing arbitrary NCL circuits. This LE is an alternative to those developed herein for implementing reconfigurable delay insensitive NCL circuits, and will be compared to those developed herein in Section VI.

IV. Design of a Reconfigurable NCL Logic Element

Fig. 4 shows a hardware realization of a reconfigurable NCL LE, consisting of reconfigurable logic, hysteresis logic, reset

logic, and output inversion logic. There are 16 inputs used specifically for programming the gate: *Rv*, *Inv*, and *Dp(14:1)*; five inputs are only used during gate operation: A, B, C, D, and *rst*; and P is used to select between programming and operational mode. Z is the gate output; *Rv* is the value Z will be reset to when *rst* is asserted during operational mode; and *Inv* determines if the gate output is inverted or not. During programming mode, *Dp(14:1)* is used to program the LUT's 14 latches in order to configure the LE as a specific NCL gate; addresses 15 and 0 are constant values and therefore do not need to be programmed, as explained in the Section IV-A. Initially, P is asserted to program the gate, then P is deasserted and the gate operates as configured.

A. Reconfigurable and Hysteresis Logic

The reconfigurable logic portion consists of a 16-address LUT, shown in Fig. 5, and a pull-up/pull-down (PUPD) function. The LUT contains 14 latches, shown in Fig. 6, and a pass transistor multiplexer (MUX). When P is asserted (*nP* is deasserted), the *Dp* values are stored in their respective latch to configure the LUT output to one of the 27 equations in Table I. Thus, only 14 latches are required because address 0 is always logic 0 and address 15 is always logic 1, according to the 27 NCL gate equations. The gate inputs A, B, C, and D are connected to the MUX select signals to pass the selected latch output to the LUT output. The MUX consists of N-type transistors and a CMOS inverter to provide a full voltage swing at the output. Since the MUX output is inverted, its input is taken from the inverted latch output.

The LUT output is then connected to the N-type transistor of the PUPD function, such that the output of this function will be logic 0 only when F is logic 1. Since all gate inputs (i.e., A, B, C, and D) are connected to a series of P-type transistors, the PUPD function output will be logic 1 only when all gate inputs are logic 0. The rest of the time when the LUT is outputting logic 0 and at least one gate input is asserted, the output of the PUPD logic will be floating, and the reconfigurable gate's output will be supplied through the weak inverter loop in the hysteresis logic. Hence, when the LUT outputs logic 1, Z becomes asserted. Z then remains asserted because of the hysteresis logic until all gate inputs become deasserted, at which time Z becomes deasserted. Z then remains deasserted because of the hysteresis logic until the LUT outputs another logic 1, causing Z to become asserted again.

To configure this LE as a specific NCL gate, the LUT should be programmed with logic 1 for any set of inputs corresponding to the gate's set condition, shown in Table I, and should be programmed with a logic 0 for the remaining input combinations. Take for example a TH23 gate, whose equation is $AB + AC + BC$. The LUT should be programmed with logic 1 for the following four input patterns: $ABC = 011, 101, 110$, and 111, which correspond to addresses 6, 10, 12, and 14. The other four combinations ($ABC = 000, 001, 010$, and 100, corresponding to addresses 0, 2, 4, and 8) should be programmed with a logic 0. For gates with less than four inputs, the unused input(s) should be set to logic 0. Hence, for the TH23 gate, D would be connected to logic 0.

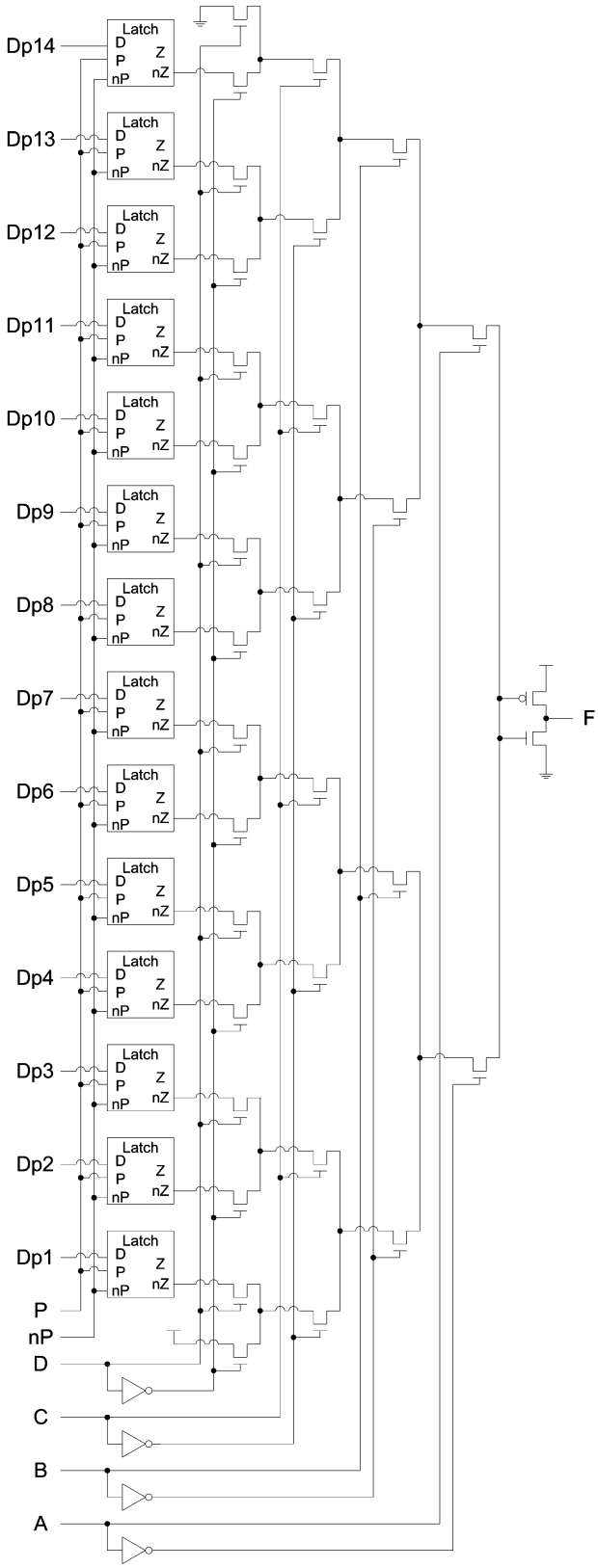

Fig. 5. 16-bit LUT.

B. Reset Logic

The reset logic consists of a programmable latch and transmission gate MUX. During the programming phase when P is

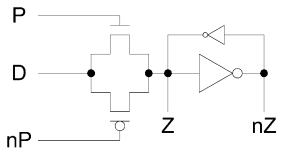

Fig. 6. Programmable latch.

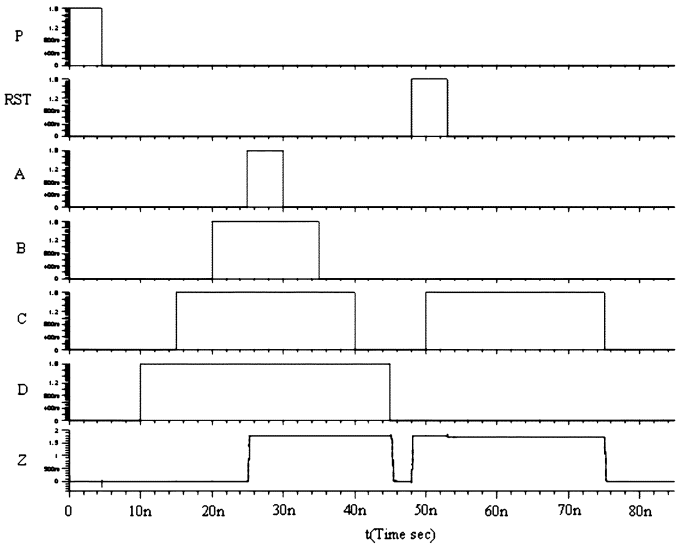

Fig. 7. Simulation of Fig. 4 programmed as a noninverting TH44d gate.

asserted (*nP* is deasserted), the latch stores the value *Rv*, that the gate will be reset to when *rst* is asserted. *rst* is the MUX select input, such that when it is logic 0, the output of the PUPD function passes through the MUX to be inverted and output on Z; and when *rst* is logic 1, the inverse of *Rv* is passed through the MUX.

C. Output Inversion Logic

The output inversion logic also consists of a programmable latch and transmission gate MUX. The programmable latch stores *Inv* during the programming phase, which determines if the gate is inverting or not. The input and output of the hysteresis logic are both fed as data inputs to the MUX, so that either the inverted or noninverted value can be output, depending on the stored value of *Inv*, which is used as the MUX select input.

D. Simulation

The reconfigurable NCL LE in Fig. 4 was simulated with Mentor Graphics Accusim II tool using a 1.8-V, 180-nm TSMC CMOS process. Fig. 7 shows the simulation of this LE programmed as a noninverting TH44d gate, which is equivalent to a four-input C-element [13], resettable to logic 1. The first 5 ns of the simulation is the programming phase, where $Rv = 1$ and $Inv = 0$ are stored, while the LUT is programmed as a TH44 gate by setting Dp(14:1) to "00000000000000" (remember that address 15 is hardwired to logic 1 and address 0 to logic 0, as explained in Section IV-A). At the end of the programming phase, P becomes logic 0, and the gate begins operation as a noninverting TH44d gate. The first input is 0000, such that Z is logic 0. The inputs A, B, C, and D are then asserted one at a time in 5 ns intervals, until all four are logic 1, at which time Z becomes asserted. Next, the inputs are deasserted one at a time in 5-ns intervals, showing that Z remains asserted due to the hysteresis logic, until all inputs are logic 0, at which time Z becomes logic 0. Following this, the gate is reset by asserting *rst*, which causes Z to be asserted. When *rst* is deasserted, Z remains asserted due to the hysteresis logic, since C is logic 1; it does so until C is deasserted, at which time all inputs are logic 0, causing Z to return to logic 0.

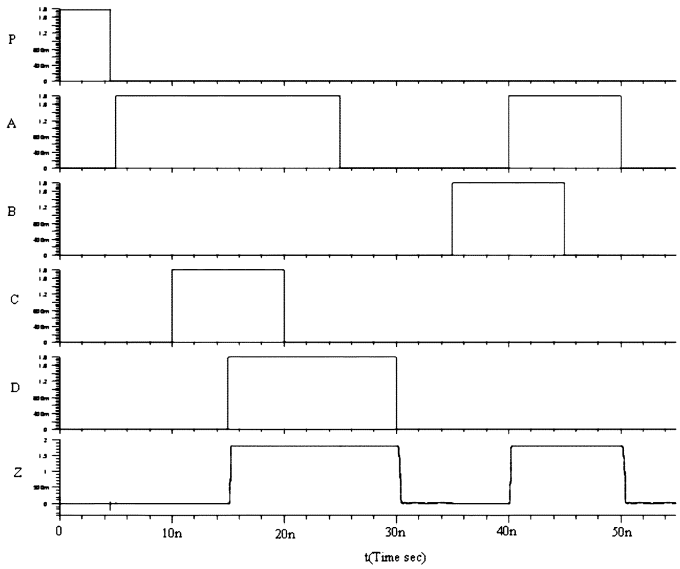

Fig. 8. Simulation of Fig. 4 programmed as a TH54w32 gate.

The LE was also programmed as a noninverting TH54w32 gate, as shown in Fig. 8. The first 5 ns of the simulation is again the programming phase, where $Rv = 0$ and $Inv = 0$ are stored, while the LUT is programmed as a TH54w32 gate (i.e., $Z = AB + ACD$) by setting Dp(14:1) to "11110000000000." At the end of the programming phase, P becomes logic 0, and the gate begins operation as a noninverting TH54w32 gate. The inputs A, C, and D are asserted one at a time in 5-ns intervals, until all three are logic 1, at which time Z becomes asserted. Next, the inputs are deasserted one at a time in 5-ns intervals, showing that Z remains asserted due to the hysteresis logic until all inputs are logic 0, at which time Z becomes logic 0. The inputs A and B are then asserted, causing Z to again become asserted, and are subsequently deasserted, causing Z to return to logic 0.

V. Alternative Reconfigurable NCL Logic Element with Extra Embedded Registration Capability

An alternative to the reconfigurable NCL LE described in Section IV is shown in Fig. 9. This design is very similar to the previous version; however, it contains an additional latch, and input *ER* for selecting when embedded registration is used, and additional embedded registration logic within the reconfigurable logic's PUPD logic, along with an additional registration request input, K_i. The remaining portions of the design, hysteresis logic, reset logic, and output inversion logic, function the same as in the previous version, explained in Sections IV-A, IV-B, and IV-C, respectively.

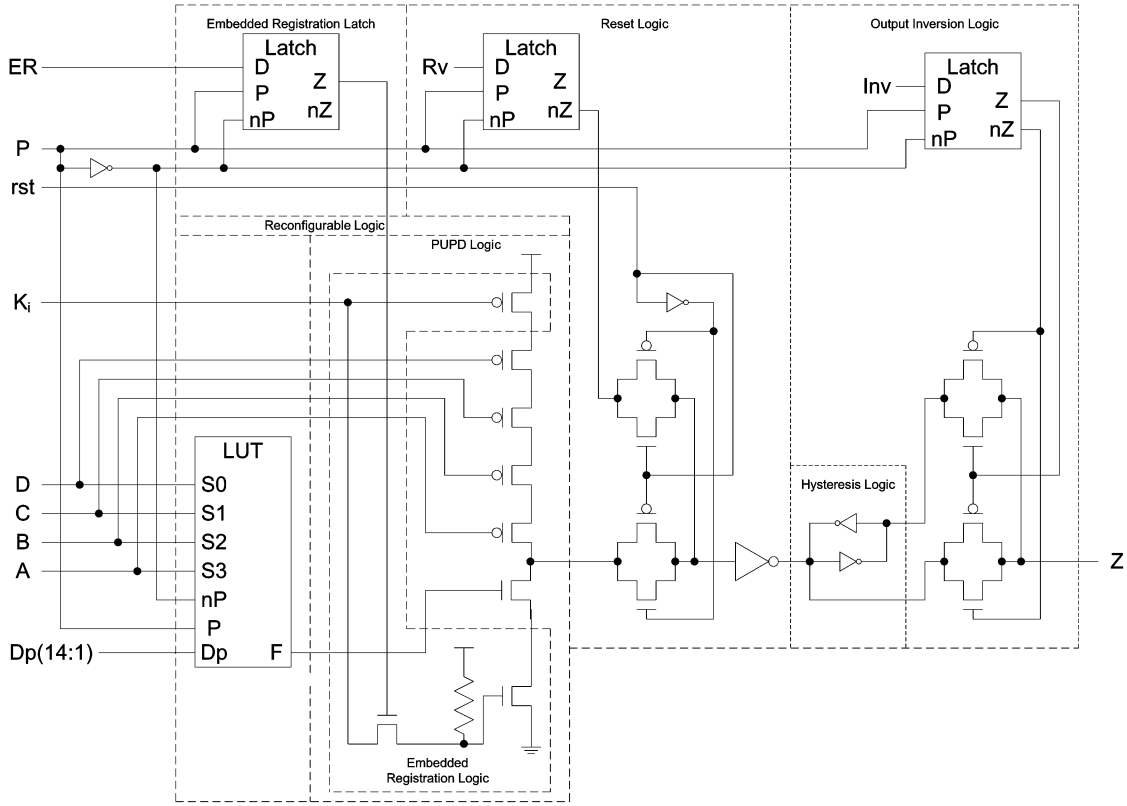

Fig. 9. Reconfigurable NCL LE with extra embedded registration.

A. Reconfigurable Logic

The reconfigurable logic portion consists of the same 16-address LUT used in the previous version and explained in Section IV-A, and a revised PUPD function that includes additional embedded registration logic. When embedded registration is disabled (i.e., $\mathrm{ER} = \mathrm{logic}\,0$ during the programming phase), K_i should be connected to logic 0, and the PUPD logic functions the same as explained in Section IV-A. However, when embedded registration is enabled, the output of the PUPD function will only be logic 0 when both F and K_i are logic 1, and will only be logic 1 when all gate inputs (i.e., A, B, C, and D) and K_i are logic 0. The rest of the time the output of the PUPD logic will be floating, and the reconfigurable gate's output will be supplied through the weak inverter loop in the hysteresis logic, as explained in Section IV-A.

B. Embedded Registration

Embedded registration [30] merges delay insensitive registers into the combinational logic, when possible, which increases circuit performance and substantially decreases the FPGA area required to implement most designs, especially high throughput circuits (i.e., circuits containing many registers). Fig. 10 shows an example of embedded registration applied to an NCL full-adder, where (a) shows the original design consisting of a full-adder and 2-bit NCL register [8], [14], (b) shows the design utilizing embedded registration when implemented using the reconfigurable NCL LE without extra embedded registration capability, and (c) shows the design utilizing embedded registration when implemented using the reconfigurable NCL LE with extra embedded registration capability. Both reconfigurable gates can be used to embed the registration with the full-adder's carry output C_o since it is generated by three-input gates (i.e., two TH23 gates), such that adding the K_i input to these gates changes them to four-input gates, which map to one of the 27 fundamental NCL gates in Table I, since these 27 gates constitute all functions of four or fewer variables. The equation for rail 1 of the registered carry output C_o^1 in Fig. 10(a) is: $K_i \bullet (X^1Y^1 + X^1C_i^1 + Y^1C_i^1)$, which maps to a TH44w2 gate, as shown in Fig. 10(b). The same transformation can also be applied to rail 0 of the carry output C_o^0, as shown in Fig. 10(b).

Embedded registration cannot be utilized for the sum output S when using the reconfigurable NCL LE without extra embedded registration capability, because S is generated by four-input gates (i.e., two TH34w2 gates), such that adding the K_i input to these gates changes them to five-input gates, which are not included in the 27 fundamental NCL gates. However, utilizing the extra embedded registration capability of the reconfigurable NCL LE shown in Fig. 9, allows for the registration to also be embedded with the full-adder's sum output, S, as shown in Fig. 10(c).

C. Simulation

The reconfigurable NCL LE in Fig. 9 was simulated with Mentor Graphics Accusim II tool using a 1.8-V, 180-nm TSMC CMOS process. This gate was programmed as a noninverting TH44d gate and a noninverting TH54w32 gate, without embedded registration, yielding the same waveforms as the

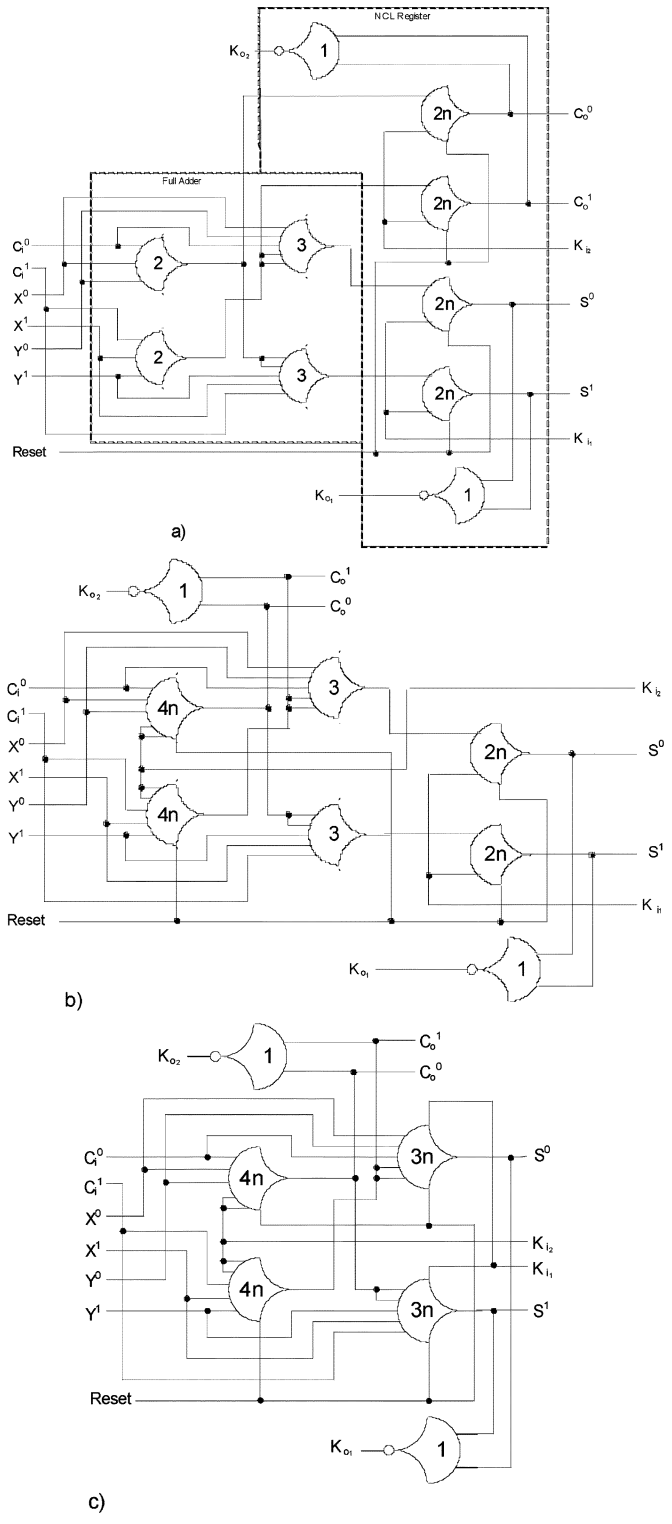

Fig. 10. Embedded registration example. (a) Original design. (b) implemenation using NCL reconfigurable LE in Fig. 4. (b) Implemenation using NCL reconfigurable LE in Fig. 9.

previous gate's simulations, as explained in Section IV-D and shown in Figs. 7 and 8, respectively, but with slightly different timing. Fig. 11 shows the LE programmed as a TH54w32 gate with embedded registration. The first 5 ns of the simulation is the programming phase, where $ER = 1, Rv = 0$,

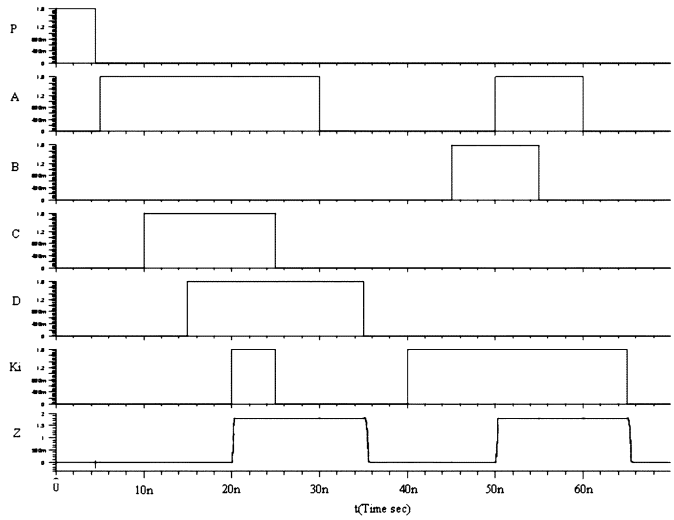

Fig. 11. Simulation of Fig. 9 programmed as a TH54w32 gate with embedded registration.

TABLE II
PROPAGATION DELAY COMPARISON BASED ON INPUT TRANSITION

	without ER (ps)		with ER (ps)	
	0→1	1→0	0→1	1→0
A	195	347	222	415
B	200	355	228	434
C	210	362	239	450
D	216	368	243	462
Ki	N/A	N/A	213	466
rst	141	115	141	115
	Tp = 282		Tp = 337	

and $Inv = 0$ are stored, while the LUT is programmed as a TH54w32 gate (i.e., $Z = AB + ACD$) by setting Dp(14:1) to "11110000000000." At the end of the programming phase, P becomes logic 0, and the gate begins operation as a noninverting TH54w32 gate with embedded registration. Comparing Fig. 11 with Fig. 8 shows that the gate operates the same as explained in Section IV-D; however, Z will not become asserted until K_i is also asserted, as shown at 20 ns, and Z will not become deasserted until K_i is also deasserted, as shown at 65 ns.

VI. RECONFIGURABLE LOGIC ELEMENT COMPARISON

Table II compares the $0 \rightarrow 1$ and $1 \rightarrow 0$ propagation delays for the two reconfigurable LEs developed herein, based on which input transition caused the output to transition, and shows the average propagation delay, T_p, during normal operation (i.e., excluding reset). Table III compares the average propagation delay of the two LEs when configured as different input sized gates, with and without embedded registration. Remember that embedded registration can be used with the reconfigurable LE without extra embedded registration capability, as explained in Section V-C, but only for gates with three or fewer inputs. Comparing the two reconfigurable LEs developed herein shows that the version without extra embedded registration is 6% smaller (i.e., 159 versus 169 transistors) and 20% faster; however, since fewer gates may be required when using the version with extra embedded registration, the extra embedded registration version

TABLE III
PROPAGATION DELAY COMPARISON BASED ON NUMBER OF GATE INPUTS

	Tp (ps)	
# inputs	without ER	with ER
1	271	319
2	274	325
3	278	331
4	282	337
1 w/ER	274	329
2 w/ER	278	330
3 w/ER	282	333
4 w/ER	N/A	337

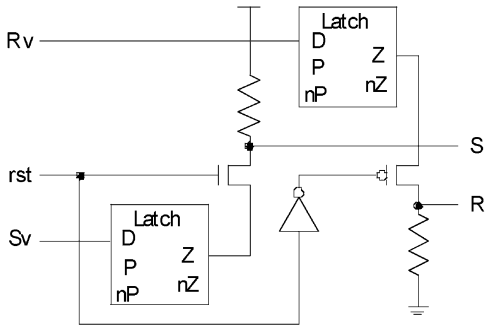

Fig. 12. Additional reset logic for [11].

may produce a smaller, faster circuit, depending on the amount of additional embedded registration that can be utilized.

A. Comparison to Previous Work

Comparing the LEs developed herein to the reconfigurable NCL LE developed in [11], and described in Section III, shows that the LE in [11] is much smaller; however, this is not a fair comparison. In order to compare this architecture, one has to consider reset operation and interconnect switches. First consider the reset operation. The LE from [11] uses an SR inverter for resetting capability, which acts as follows: when S and R are both logic 1, the output is reset to logic 0; when S and R are both logic 0, the output is reset to logic 1; when $S = 1$ and $R = 0$, the output is the complement of the input; and $S = 0$ and $R = 1$ is illegal. The 32-transistor version of [11] includes both an S and R input. However, direct implementation would yield a race condition between the S and R inputs when resetting. Hence, the LE from [11] needs to include additional logic for proper resetting. One way to do this is shown in Fig. 12, where reset and set values *Rv* and *Sv*, respectively, are stored in latches, such that the SR inverter's R and S inputs transition to these programmed values when *rst* is asserted to reset the gate, otherwise $S = 1$ and $R = 0$, and the gate acts as a regular inverter. Note that this mitigates the previously mentioned race condition by allowing at most one of an SR inverter's S or R inputs to change during reset. This method of reset could have been used for the reconfigurable LEs designed herein, but it requires an additional four transistors and two resisters, so the reset logic developed herein is preferred.

Now consider the interconnects required for the LEs. The LEs developed herein have four data inputs A, B, C, and D since they can be configured as maximum four-input gates. The cell

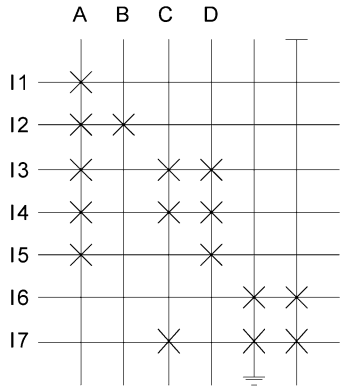

Fig. 13. Additional configuration logic for [11].

from [11] can also be configured as a maximum four-input gate; however, it has seven data inputs $I1$–$I7$. This would require a much larger general purpose interconnect switch to select the LE inputs. However, a portion of this interconnect switch can be moved inside of the reconfigurable LE, such that the external interface only consists of four data inputs (A, B, C, and D), the same as the LEs designed herein. This allows for the switch to be significantly reduced by allowing for only the minimal connections to implement the nine possible configurations by taking advantage of mutually exclusive connections. Fig. 13 shows this internal switch logic, which routes the four external gate inputs (A, B, C, and D) to the seven internal gate inputs ($I1$–$I7$). Note that input C is the weighted input when implementing the TH34w2 gate. $I1$ is always connected to input A, so no programmable interconnect is needed. $I2$ is always connected to either input A or B, so it requires one latch to store the configuration and one transmission gate MUX to form the selected connection. $I3$ and $I4$ are always connected to either A, C, or D, so they each require two latches and two transmission gate MUXs. Like $I2, I5$ is always connected to either of two inputs, so it requires one latch and one transmission gate MUX. $I6$ is always either logic 0 or logic 1, so it only requires one latch. $I7$ is always either logic 0, logic 1, or input C, so it requires two latches and one transmission gate MUX.

Also consider the gate output. NCL circuits rarely require the output in both complemented and noncomplemented form, so it is beneficial to move the output selection into the LE as well, in order to significantly reduce the interconnect switch size at the LE output. This would include the addition of a latch and transmission gate MUX, as shown in Fig. 4 as the output inversion logic. Including this necessary functionality, a realistic version of the LE designed in [11] requires 142 transistors, one reset input, *rst*, four data inputs, A, B, C, and D, 12 configuration inputs, *Sv, Rv, Inv*, and nine internal switch logic configuration inputs (i.e., for the circuit in Fig. 13), and one input, P, to select between programming and operational mode.

Now comparing the reconfigurable LEs designed herein to the realistic version of [11] described previously, shows that the version with and without extra embedded registration capability has 19% and 12% more transistors, respectively; however, both are much more versatile. Both LEs designed herein can be programmed as all 27 fundamental NCL gates, including both resetting and inverting variations, and can be programmed

TABLE IV
RECONFIGURABLE GATE COMPARISON FOR NCL MULTIPLIERS

	Number of Gates			Number of Transistors		
Multiplier Architecture	[11]	without ER	with ER	[11]	without ER	with ER
Dual-Rail Non-Pipelined [9]	176	141	128	24,992	22,419	21,632
Dual-Rail Full-Word Pipelined [9]	400	334	302	56,800	53,106	51,038
Dual-Rail Bit-Wise Pipelined [9]	365	275	231	51,830	43,725	39,039
Quad-Rail Non-Pipelined [31]	418	235	229	59,356	37,365	38,701
Quad-Rail Full-Word Pipelined [32]	523	268	233	74,266	42,612	39,377
Quad-Rail Bit-Wise Pipelined [32]	484	229	194	68,728	36,411	32,786

as an inverter. On the other hand, the reconfigurable LE designed in [11] can only be programmed as 8 of the 27 fundamental gates (TH12, TH13, TH14, TH22, TH33, TH44, TH23, and TH34w2), including both resetting and inverting variations, and can be programmed as an inverter. Furthermore, the reconfigurable LE without extra embedded registration capability can utilize embedded registration with all seven two-input and three-input NCL gates; and the version with extra embedded registration capability can utilize embedded registration with all 27 fundamental NCL gates. However, the reconfigurable LE in [11] can only utilize embedded registration with TH22 and TH33 gates. Therefore, an arbitrary NCL design may require more area and delay when using the reconfigurable LE developed in [11], versus the versions developed herein, because more LEs may be required to implement the design since the reconfigurable LE from [11] can only be programmed as a small subset of the 27 fundamental NCL gates and can only utilize minimal embedded registration.

B. Comparison of Small NCL Circuits

Consider the dual-rail NCL full-adder with output register shown in Fig. 10. Implementing this circuit using the reconfigurable LE from [11] requires 10 gates and 1420 transistors, with a worse-case delay of two and three gates for the carry C_o and sum S output, respectively, as shown in Fig. 10(a). Using the LE designed herein without extra embedded registration requires 8 gates and 1272 transistors, with a worse-case delay of one and three gates for C_o and S, respectively, as shown in Fig. 10(b). Implementation with the LE designed herein with extra embedded registration requires 6 gates and 1014 transistors, with a worse-case delay of one and two gates for C_o and S, respectively, as shown in Fig. 10(c). Therefore, the registered full-adder is best implemented using the LE designed herein with extra embedded registration, when considering both area and speed.

Now take for example the dual-rail input-complete NCL AND function [8] shown in Fig. 14. Implementation using either LE designed herein requires two gates and a worse-case delay of one gate, with 338 and 318 transistors for the version with and without extra embedded registration, respectively. Using the reconfigurable LE from [11] requires 5 gates and 710 transistors, with a worse-case delay of two gates, as shown in Fig. 15, since the THand0 gate must be decomposed into a set of gates that are implementable using the reconfigurable LE from [11] (i.e., $\{\mathrm{TH12}, \mathrm{TH13}, \mathrm{TH14}, \mathrm{TH22}, \mathrm{TH33}, \mathrm{TH44}, \mathrm{TH23},$ and $\mathrm{TH34w2}\}$). Therefore, the dual-rail input complete NCL AND function is best implemented using the LE designed herein without extra embedded registration, since embedded registration was not able to be utilized.

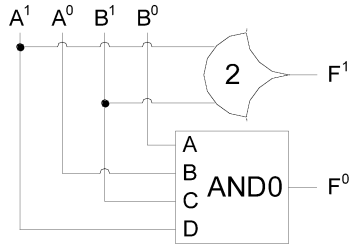

Fig. 14. NCL AND function [8].

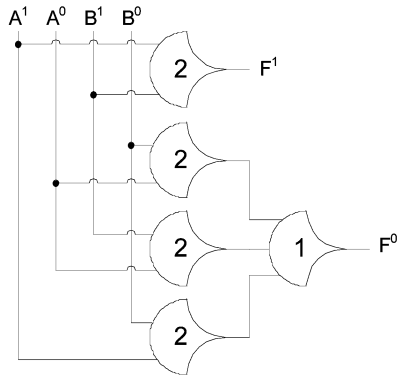

Fig. 15. Decomposed NCL AND function [8].

C. Comparison of NCL Multipliers and ALUs

Table IV shows a comparison of a variety of unsigned 4-bit × 4-bit NCL multiplier architectures mapped to the two reconfigurable LEs developed herein and to the one designed in [11], demonstrating that both LEs designed herein require substantially less area compared to the one from [11]. Furthermore, the LE developed herein with extra embedded registration requires less area than the version without extra embedded registration for all of the multiplier designs except for the quad-rail nonpipelined version, since these designs contain a substantial number of four-input gates where embedded registration can only be applied using the extra embedded registration capability.

Table V shows a comparison of a variety of eight-operation, 4-bit operand NCL ALUs mapped to the three reconfigurable NCL LEs being compared, demonstrating again that both LEs designed herein require substantially less area compared to the one from [11]. However, for the ALUs, the LE developed

TABLE V
RECONFIGURABLE GATE COMPARISON FOR NCL ALUS

	Number of Gates			Number of Transistors		
ALU Architecture	[11]	without ER	with ER	[11]	without ER	with ER
Dual-Rail Non-Pipelined [30, 33]	398	252	250	56,516	40,068	42,250
Dual-Rail Pipelined [30, 33]	696	513	505	98,832	81,567	85,345
Dual-Rail NULL Cycle Reduced [30, 33]	886	594	590	125,812	94,446	99,710
Quad-Rail Non-Pipelined [30, 33]	792	524	522	112,464	83,316	88,218
Quad-Rail Pipelined [30, 33]	1234	938	910	175,228	149,142	153,790
Quad-Rail NULL Cycle Reduced [30, 33]	1677	1141	1137	238,134	181,419	192,153

herein without extra embedded registration requires less area than the version with extra embedded registration for all ALU designs, since these designs contain few four-input gates where embedded registration can be applied. Hence, the extra embedded registration capability is underutilized.

VII. CONCLUSIONS AND FUTURE WORK

This paper details the design of two reconfigurable NCL LEs, one with extra embedded registration capability and one without, both of which can be programmed as any of the 27 fundamental NCL gates, shown in Table I, and support resetting and output inversion capability. These gate designs were compared with each other and an alternative reconfigurable NCL LE described in [11] for a number of NCL circuits, showing that both versions designed herein require substantially less area than the one in [11]. For the multiplier circuits, using the LE in [11] requires 56% more gates and 42% more transistors compared to the version without extra embedded registration, and 79% more gates and 51% more transistors compared to the version with extra embedded registration, on average. Likewise, on average, using the LE in [11] requires 43% more gates and 28% more transistors compared to the version without extra embedded registration, and 45% more gates and 22% more transistors compared to the version with extra embedded registration, for the ALUs. The comparison further shows that the version without extra embedded registration is 6% smaller and 20% faster compared to the one with extra embedded registration; however, since fewer gates may be needed when using the LE with extra embedded registration, the extra embedded registration version may produce a smaller, faster circuit, depending on the amount of additional embedded registration that can be utilized. Comparison of the multiplier circuits demonstrate this point, showing that using the LE with extra embedded registration results in 6% fewer transistors on average, since these circuits contain a substantial number of four-input gates where embedded registration can only be applied using the extra embedded registration capability. However, the opposite was true for the ALUs, where using the version with extra embedded registration resulted in 5% more transistors on average, since these designs contain few four-input gates where embedded registration can be applied, such that the extra embedded registration capability was underutilized.

Further analysis suggests that the LEs developed herein are even more advantageous than previously explained when compared to the reconfigurable LE in [11]. Tables IV and V show that using the reconfigurable LE from [11] results in substantially more gates being required than for either of the two versions developed herein, because many of the original circuit's gates need to be decomposed, as explained in Section VI-B. This is partially accounted for in the comparison of number of transistors; however, the increased number of interconnect switches required because of these additional gates is not considered. Doing so would further increase the area overhead when comparing the reconfigurable LE in [11] to the versions herein. Furthermore, gate decomposition is likely to increase the number of gates and interconnect switches on the circuit's critical path(s), as demonstrated in Section VI-B, resulting in a slower circuit implementation. Therefore, using the LEs developed herein should result in a faster implementation, requiring less area, compared to the reconfigurable LE in [11], for most nontrivial NCL circuits.

Either reconfigurable LE developed herein or the one in [11] could be used to implement delay insensitive circuits designed using DIMS [19], Anantharaman's [17], and Singh's [16] approaches, as well as NCL, since these other approaches only require C-elements (i.e., THnn gates) and OR gates (i.e., TH1n gates). Furthermore, these three reconfigurable LEs could also be used to implement delay insensitive circuits using Seitz's [12] and David's [18] methods, by replacing the AND gates with C-elements. Doing so will not change the circuit functionality, but may increase delay for these circuits. Additionally, all three reconfigurable LEs could also be configured to implement Boolean circuits using NOR-NOR logic (i.e., inverting TH1n gates); although, this implementation would be very inefficient.

Additional topics that need further investigation, but are beyond the scope of this paper, include the overall FPGA architecture, the configurable logic block (CLB) architecture, and the FPGA interconnect strategy. Possible choices for overall architecture include island-style or hierarchical. Alternative numbers of LEs, connection of LEs, and additional circuitry within a CLB need to be studied. While most busses can be implemented using a multiplexed bus structure, constructed from the 27 fundamental NCL gates, some designs require an arbitrated bus structure, which would require a gate like the MUTEX [4]. Hence, the overall FPGA design would also need to include bus arbiter elements. Finally, interconnect grouping needs to be researched. Interconnects can be routed as single wires, dual-rail signals, or quad-rail signals. The advantage to dual-rail and quad-rail interconnects is that only one latch is required to configure a two-wire and four-wire connection, respectively; whereas one latch is required to configure each wire connection when using single-wire routing, as is the case for standard synchronous FPGAs.

Compared to the high-throughput, fine-grain Achronix Semiconductor FPGA overviewed in Section III, an NCL FPGA utilizing either LE developed herein would be even more fine grained, such that the resulting NCL FPGA would likely yield significant speed and power benefits compared to synchronous FPGAs, as does the Achronix FPGA.

REFERENCES

[1] J. McCardle and D. Chester, "Measuring an asynchronous Processor's power and noise," in *Proc. Synopsys User Group Conf. (SNUG)*, 2001, pp. 66–70.

[2] G. E. Sobelman and K. M. Fant, "CMOS circuit design of threshold gates with hysteresis," in *Proc. IEEE Int. Symp. Circuits Syst. (II)*, 1998, pp. 61–65.

[3] A. Balasubramanian, "An asynchronous FPGA for NULL convention logic circuits," Master's thesis, Dept. Electr. Comput. Eng., Univ. Missouri–Rolla, Rolla, 2005.

[4] K. M. Fant, *Logically Determined Design: Clockless System Design with NULL Convention Logic*. New York: Wiley, 2005.

[5] A. J. Martin, "Programming in VLSI," in *Development in Concurrency and Communication*. Reading, MA: Addison-Wesley, 1990, pp. 1–64.

[6] K. Van Berkel, "Beware the isochronic fork," *Integr., VLSI J.*, vol. 13, no. 2, pp. 103–128, 1992.

[7] A. Kondratyev, L. Neukom, O. Roig, A. Taubin, and K. Fant, "Checking delay-insensitivity: 104 gates and beyond," in *Proc. 8th Int. Symp. Asynchronous Circuits Syst.*, 2002, pp. 137–145.

[8] S. C. Smith, R. F. DeMara, J. S. Yuan, D. Ferguson, and D. Lamb, "Optimization of NULL convention self-timed circuits," *Integr., VLSI J.*, vol. 37, no. 3, pp. 135–165, 2004.

[9] S. C. Smith, R. F. DeMara, J. S. Yuan, M. Hagedorn, and D. Ferguson, "Delay-insensitive gate-level pipelining," *Integr., VLSI J.*, vol. 30, no. 2, pp. 103–131, 2001.

[10] E. Keller, "Building asynchronous circuits with JBits," in *Proc. 11th Int. Conf. Field Program. Logic Appl.*, 2001, pp. 628–632.

[11] D. R. Lamb, "Self-timed circuits for adaptive processing systems," in *Proc. Military Aerosp. Appl. Program. Devices Technol. Conf.*, 1998, Paper No. B2.

[12] C. L. Seitz, "System timing," in *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980, pp. 218–262.

[13] D. E. Muller, "Asynchronous logics and application to information processing," in *Switching Theory in Space Technology*. Stanford, CA: Stanford Univ. Press, 1963, pp. 289–297.

[14] K. M. Fant and S. A. Brandt, "NULL convention logic: A complete and consistent logic for asynchronous digital circuit synthesis," in *Proc. Int. Conf. Appl. Specific Syst., Arch., Process.*, 1996, pp. 261–273.

[15] S. C. Smith, "Completion-completeness for NULL convention digital circuits utilizing the bit-wise completion strategy," in *Proc. Int. Conf. VLSI*, 2003, pp. 143–149.

[16] N. P. Singh, "A design methodology for self-timed systems," Master's thesis, Lab. Comput. Sci., MIT, Cambridge, 1981, MIT/LCS/TR-258.

[17] T. S. Anantharaman, "A delay insensitive regular expression recognizer," *IEEE VLSI Technol. Bulletin*, Sep. 1986.

[18] I. David, R. Ginosar, and M. Yoeli, "An efficient implementation of boolean functions as self-timed circuits," *IEEE Trans. Comput.*, vol. 41, no. 1, pp. 2–10, Jan. 1992.

[19] J. Sparso, J. Staunstrup, and M. Dantzer-Sorensen, "Design of delay insensitive circuits using multi-ring structures," in *Proc. Eur. Design Autom. Conf.*, 1992, pp. 15–20.

[20] S. Hauck, S. Burns, G. Borriello, and C. Ebeling, "An FPGA for implementing asynchronous circuits," *IEEE Design Test Comput.*, vol. 11, no. 3, pp. 60–69, 1994.

[21] K. Maheswaran, "Implementing self-timed circuits in field programmable gate arrays," Master's thesis, Electr. Comput. Eng. Dept., Univ. California, Davis, 1995.

[22] R. E. Payne, "Self-timed FPGA systems," in *Proc. 5th Int. Workshop Field Program. Logic Appl.*, 1995, pp. 21–35.

[23] C. Traver, R. B. Reese, and M. A. Thornton, "Cell designs for self-timed FPGAs," in *Proc. 14th Annu. IEEE Int. ASIC/SOC Conf.*, 2001, pp. 175–179.

[24] M. Aydin and C. Traver, "Implementation of a programmable phased logic cell," in *Proc. 45th Midw. Symp. Circuits Syst.*, 2002, pp. 21–24.

[25] J. Teifel and R. Manohar, "An asynchronous dataflow FPGA architecture," *IEEE Trans. Comput.*, vol. 53, no. 11, pp. 1376–1392, Nov. 2004.

[26] C. G. Wong, A. J. Martin, and P. Thomas, "An architecture for asynchronous FPGAs," in *Proc. IEEE Int. Conf. Field-Program. Technol.*, 2003, pp. 170–177.

[27] K. Meekins, D. Ferguson, and M. Basta, "Delay insensitive NCL reconfigurable logic," in *Proc. IEEE Aerosp. Conf.*, 2002, pp. 1961–1966.

[28] D. H. Linder and J. H. Harden, "Phased logic: Supporting the synchronous design paradigm with delay-insensitive circuitry," *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 1031–1044, Sep. 1996.

[29] C. G. Wong and A. J. Martin, "High-level synthesis of asynchronous systems by data-driven decomposition," in *Proc. Design Autom. Conf.*, 2003, pp. 508–513.

[30] S. K. Bandapati and S. C. Smith, "Design and characterization of NULL convention arithmetic logic units," in *Proc. Int. Conf. VLSI*, 2003, pp. 178–184.

[31] S. K. Bandapati, S. C. Smith, and M. Choi, "Design and characterization of NULL convention self-timed multipliers," *IEEE Design Test Comput.: Special Issue Clockless VLSI Design*, vol. 30, no. 6, pp. 26–36, Nov./Dec. 2003.

[32] S. C. Smith, "Designing NULL convention combinational circuits to fully utilize gate-level pipelining for maximum throughput," in *Proc. Int. Conf. VLSI*, 2004, pp. 407–412.

[33] S. K. Bandapati and S. C. Smith, "Design and characterization of NULL convention arithmetic logic units," *Microelectron. Eng. J.: Special Issue VLSI Design Test*, vol. 84, no. 2, pp. 280–287, Feb. 2007.

Scott C. Smith (M'95–SM'06) received the B.S. degrees in electrical engineering and computer engineering and the M.S. degree in electrical engineering from the University of Missouri, Columbia, in 1996 and 1998, respectively, and the Ph.D. degree in computer engineering from the University of Central Florida, Orlando, in 2001.

Currently, he is an Associate Professor with the Department of Electrical and Computer Engineering, the University of Missouri, Rolla. He has authored 10 journal publications, 23 conference papers, 3 U.S./international patents, and 2 additional international patents, all of which can be viewed from his website: web.umr.edu/~smithsco. His research interests include computer architecture, asynchronous logic design, CAD tool development, embedded system design, VLSI, trustable hardware, and self-reconfigurable logic.

Dr. Smith is a member of Sigma Xi, Eta Kappa Nu, Tau Beta Pi, and ASEE.