

Design of a Logic Element for Implementing an Asynchronous FPGA

Scott C. Smith

Department of Electrical & Computer Engineering, University of Missouri - Rolla
1870 Miner Circle
Rolla, MO 65462
573-341-4232

smithsco@umr.edu

ABSTRACT

A reconfigurable logic element (LE) is developed for use in constructing a NULL Convention Logic (NCL) FPGA. It can be configured as any of the 27 fundamental NCL gates, including resettable and inverting variations, and can utilize embedded registration for gates with three or fewer inputs. The developed LE is compared with a previous NCL LE, showing that the one developed herein yields a more area efficient NCL circuit implementation. The NCL FPGA logic element is simulated at the transistor level using the 1.8V, 180nm TSMC CMOS process.

Categories and Subject Descriptors

B.7.1 [Integrated Circuits]: Types and Design Styles – *gate arrays*.

General Terms

Design.

Keywords

Asynchronous logic design, delay-insensitive circuits, Field Programmable Gate Array (FPGA), NULL Convention Logic (NCL), reconfigurable logic.

1. 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 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 FPGAs without compromising delay-insensitivity [3].

NCL gates can be implemented with Boolean logic by using an 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 non-isochronic 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 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 non-pipelined 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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

FPGA'07, February 18–20, 2007, Monterey, CA, USA.

Copyright 2007 ACM 1-58113-000-0/00/0004...\$5.00.

The size of FPGAs is now more than 1 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 5 years, and 30% of chip area within the next 10 years.

This paper is partitioned into six sections. Section 2 provides a brief overview of NCL; Section 3 describes the previous work on asynchronous FPGA design; Section 4 presents the design of a reconfigurable NCL logic element (LE); Section 5 compares the LE developed herein with the previous work in [11]; and Section 6 provides conclusions and directions for future work.

2. 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].

2.1 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 {DATA0, DATA1, 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.

Most multi-rail 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.

2.2 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 Figure 1, is the $THmn$ gate, where $1 \leq m \leq n$. $THmn$ gates have n inputs, where at least m of the n inputs must be asserted before the 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.

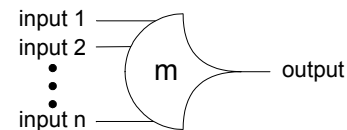


Figure 1. $THmn$ threshold gate.

Another type of threshold gate is referred to as a weighted threshold gate, denoted as $THmnWw_1w_2\dots w_R$. 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, \dots, w_R , each > 1 , are the integer weights of $input1, input2, \dots, inputR$, respectively. For example, consider the $TH34W2$ gate shown in Figure 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 .

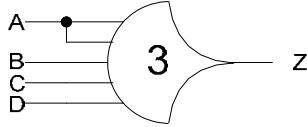


Figure 2. TH34w2 threshold gate: $Z = AB + AC + AD + BCD$.

Table 1. 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	$AC + BC + AD + BD$

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 a complete transition of inputs back to NULL before asserting the output associated with the next wavefront of input data. Therefore, a TH n n gate is equivalent to an n -input C-element and a TH1 n 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 1. 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* TH n n and *inverting* TH1 n 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].

3. PREVIOUS WORK

There have been a number of asynchronous FPGAs developed over the past 10+ years [20-28, 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 [29].

Two other asynchronous FPGAs [25-27] are both based on the precharge half-buffer (PCHB) logic family to implement quasi delay-insensitive circuits. Both use data-driven decomposition (DDD) [30] 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.

In an effort to design a delay-insensitive, reconfigurable logic device, Theseus Logic developed an FPGA based on the Atmel AT40K family [28]. The design involved replacing the D-type Flip Flop within each logic block with a threshold-configurable NCL TH m 4 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 bit-

stream 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 delay for realizing most non-trivial 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 [28]. A more efficient configurable logic element for an NCL FPGA was presented in [11]. The LE, shown in Figure 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 7 data inputs (I1-I7), a set and reset input, S and R , respectively, and 2 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 the one developed herein for implementing reconfigurable delay-insensitive NCL circuits, and will be compared with the LE developed herein in Section 5.

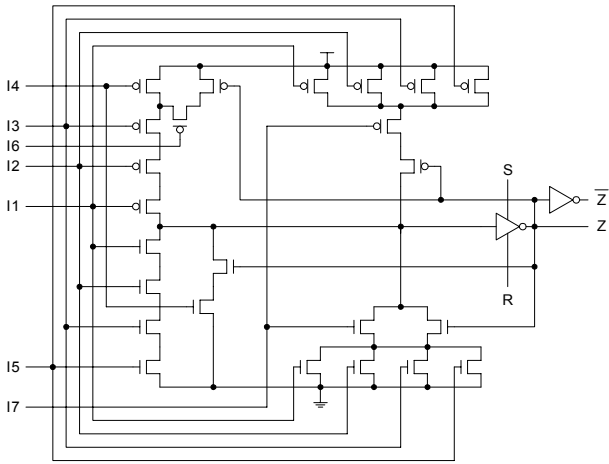


Figure 3. Reconfigurable NCL gate [11].

4. DESIGN OF A RECONFIGURABLE NCL LOGIC ELEMENT

Figure 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 $Dp(14:1)$; 5 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 next subsection. Initially, P is asserted to program the gate, then P is deasserted and the gate operates as configured.

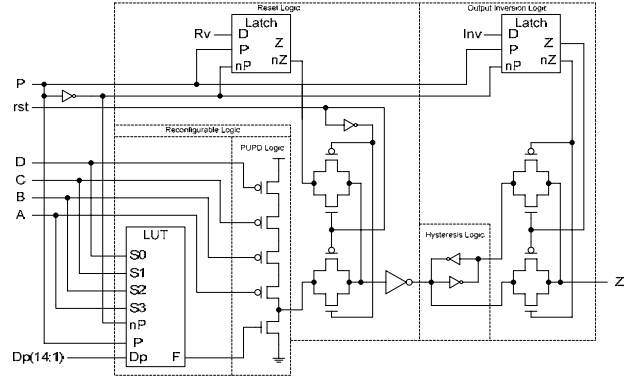


Figure 4. Reconfigurable NCL LE.

4.1 Reconfigurable and Hysteresis Logic

The reconfigurable logic portion consists of a 16-address LUT, shown in Figure 5, and a pull-up/pull-down (PUPD) function. The LUT contains 14 latches, shown in Figure 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 1. Thus, only 14 latches are required because address 0 is always logic 0 and address 15 is always logic 1, according to the 27 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 PMOS inverter to provide a full voltage swing at the output. Since the MUX output is inverted, its inputs are taken from the inverted latch outputs.

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 1, 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: $ABC = 011, 101, 110, \text{ and } 111$, which correspond to addresses 6, 10, 12, and 14. The other four combinations ($ABC = 000, 001, 010, \text{ and } 100$, 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.

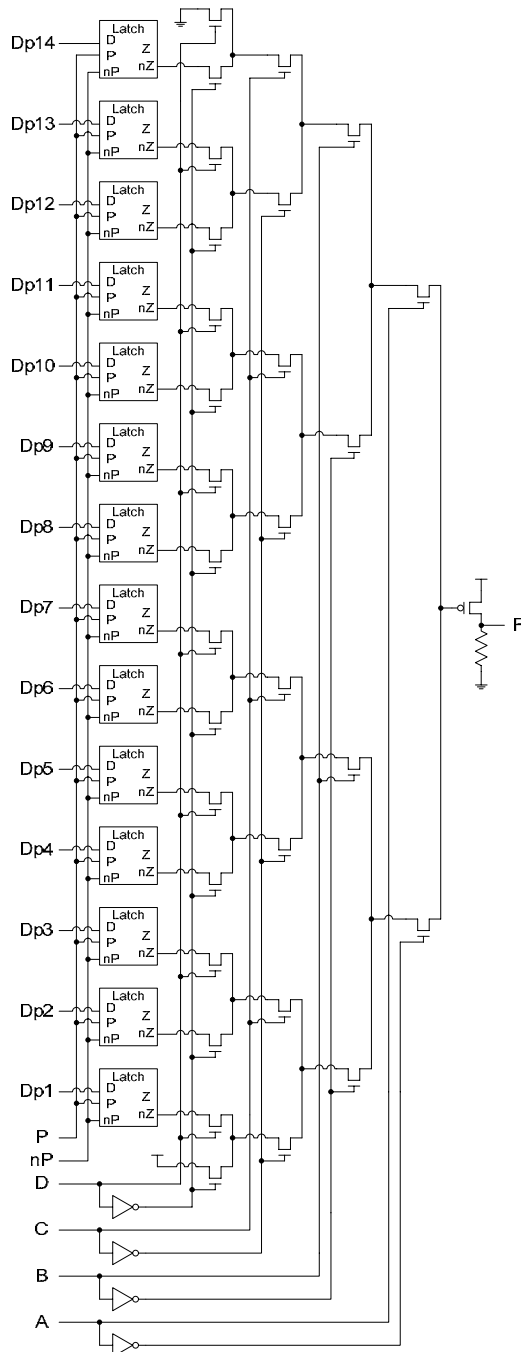


Figure 5. 16-bit LUT.

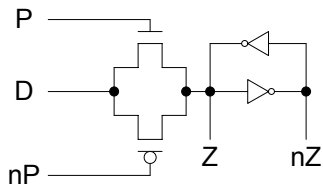


Figure 6. Programmable latch.

4.2 Reset Logic

The reset logic consists of a programmable latch and transmission gate MUX. During the programming phase when P is asserted (nP is deasserted), the latch stores the value, R_v , 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 R_v is passed through the MUX.

4.3 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 non-inverted value can be output, depending on the stored value of Inv , which is used as the MUX select input.

4.4 Simulation

The reconfigurable NCL LE in Figure 4 was simulated with Mentor Graphics Accusim II tool using a 1.8V, 180nm TSMC CMOS process. Figure 7 shows the simulation of this LE programmed as a non-inverting TH44d gate, which is equivalent to a 4-input C-element [13], resettable to logic 1. The first 5ns of the simulation is the programming phase, where $R_v = 1$ 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 4.1). At the end of the programming phase, P becomes logic 0, and the gate begins operation as a non-inverting 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 5ns intervals, until all four are logic 1, at which time Z becomes asserted. Next, the inputs are deasserted one at a time in 5ns 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.

The LE was also programmed as a non-inverting TH54w32 gate, as shown in Figure 8. The first 5ns of the simulation is again the programming phase, where $R_v = 0$ 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 non-inverting TH54w32 gate. The inputs, A , C , and D , are asserted one at a time in 5ns intervals, until all three are logic 1, at which time Z becomes asserted. Next, the inputs are deasserted one at a time in 5ns 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.

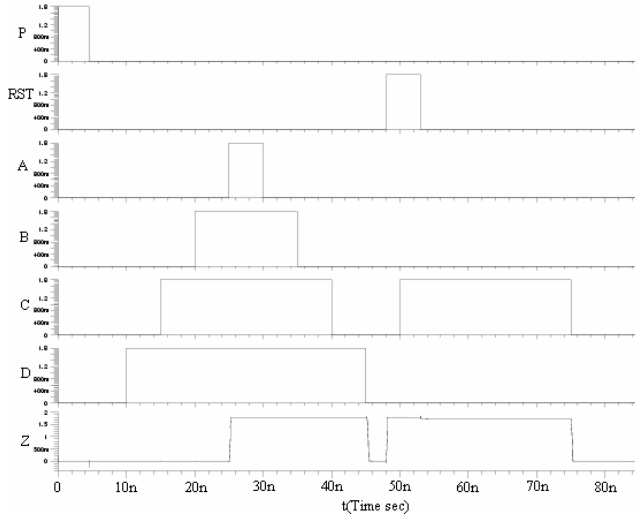


Figure 7. Simulation of Figure 4 programmed as a non-inverting TH44d gate.

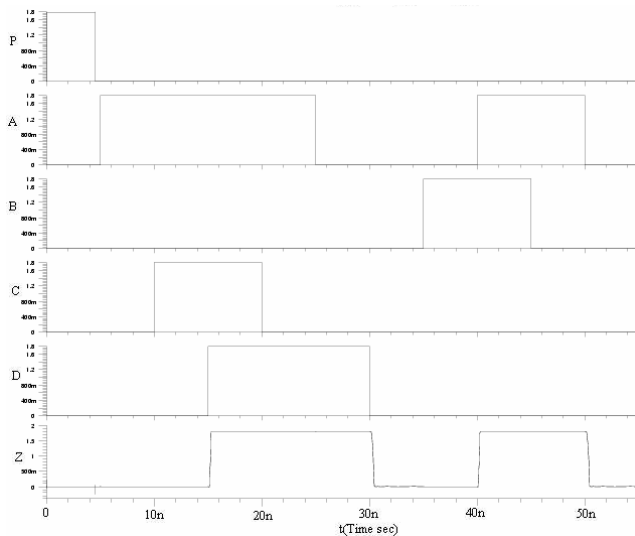


Figure 8. Simulation of Figure 4 programmed as a TH54w32 gate.

5. RECONFIGURABLE LOGIC ELEMENT ANALYSIS

Table 2 shows the 0→1 and 1→0 propagation delays for the reconfigurable LE developed herein, based on which input transition caused the output to transition, and lists the average propagation delay, T_p , during normal operation (i.e. excluding reset). Table 3 shows the average propagation delay of the LE when configured as different input sized gates, with and without embedded registration. Embedded registration [31] 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). Embedded registration can only be used for gates with three or fewer inputs.

Table 2. Propagation delay per input transition

	NCL LE (ps)	
	0→1	1→0
A	195	347
B	200	355
C	210	362
D	216	368
rst	141	115
	$T_p = 282$	

Table 3. Propagation delay per number of gate inputs

# inputs	T_p (ps)
1	271
2	274
3	278
4	282
1 w/ER	274
2 w/ER	278
3 w/ER	282
4 w/ER	N/A

5.1 Comparison to Previous Work

Comparing the LE developed herein to the reconfigurable NCL LE developed in [11], and described in Section 3, 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 Figure 9, where reset and set values, R_v and S_v , 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 LE designed herein, but it requires an additional 4 transistors and 2 resistors, so the reset logic developed herein is preferred.

Now consider the interconnects required for the LEs. The LE developed herein has 4 data inputs, A , B , C , and D , since it can be configured as a maximum 4-input gate. The cell from [11] can also be configured as a maximum 4-input gate; however, it has 7 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 4 data inputs (A , B , C , and D), the same as the LE designed herein. This allows for the switch to be significantly reduced by allowing for only the minimal connections to implement the 9 possible configurations by taking advantage of mutually exclusive connections. Figure 10 shows this internal switch logic, which routes the four external gate inputs (A , B , C , and D) to the 7 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 1 latch. $I7$ is always either logic 0, logic 1, or input C , so it requires two latches and one transmission gate MUX.

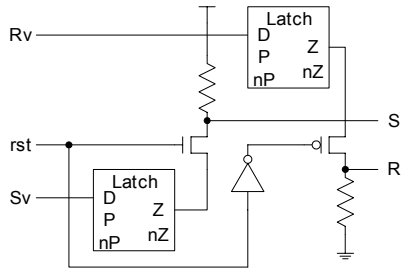


Figure 9. Additional reset logic for [11].

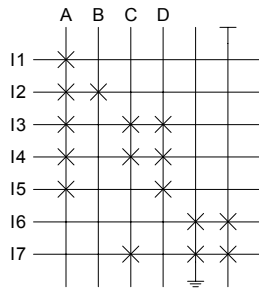


Figure 10. Additional configuration logic for [11].

Also consider the gate output. NCL circuits rarely require the output in both complemented and non-complemented 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 Figure 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 , 4 data inputs, A , B , C , and D , 12 configuration inputs, Sv , Rv , Inv , and 9 internal switch logic configuration inputs (i.e. for the circuit in Figure 10), and one input, P , to select between programming and operational mode.

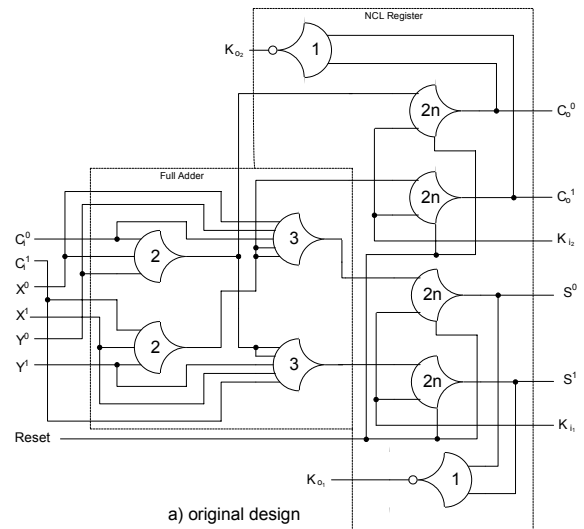
Now comparing the reconfigurable LE designed herein to the realistic version of [11] described above, shows that the one developed herein has 12% more transistors; however, it is much more versatile. The LE designed herein can be programmed as all

27 fundamental NCL gates, including both resetting and inverting variations, and can be programmed 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 one developed herein can utilize embedded registration with all seven 2-input and 3-input NCL gates, while 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 one 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.

5.2 Comparison of Small NCL Circuits

Consider the dual-rail NCL full-adder with output register shown in Figure 11. Implementing this circuit using the reconfigurable LE from [11] requires 10 gates and 1,420 transistors, with a worst-case delay of 2 gates and 3 gates for the carry, C_o , and sum, S , output, respectively, as shown in Figure 11a. Using the LE designed herein requires 8 gates and 1,272 transistors, with a worst-case delay of 1 gate and 3 gates for C_o and S , respectively, as shown in Figure 11b. Therefore, the registered full-adder is best implemented using the LE designed herein.

Now take for example the dual-rail input-complete NCL AND function [8] shown in Figure 12. Implementation using the LE designed herein requires 2 gates and a worst-case delay of 1 gate, with 318 transistors. Using the reconfigurable LE from [11] requires 5 gates and 710 transistors, with a worst-case delay of 2 gates, as shown in Figure 13, since the THand0 gate must be decomposed into a set of gates implementable with the reconfigurable LE from [11] (i.e. {TH12, TH13, TH14, TH22, TH33, TH44, TH23, and TH34w2}). Therefore, the dual-rail input-complete NCL AND function is again best implemented using the LE designed herein.



a) original design

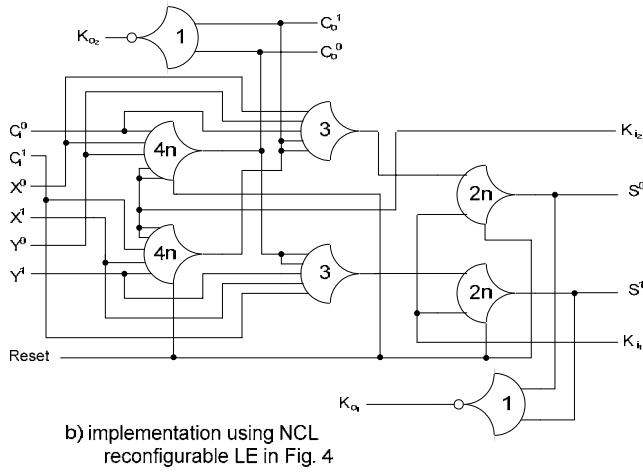


Figure 11. Embedded registration example.

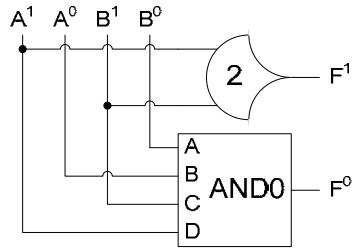


Figure 12. NCL AND function [8].

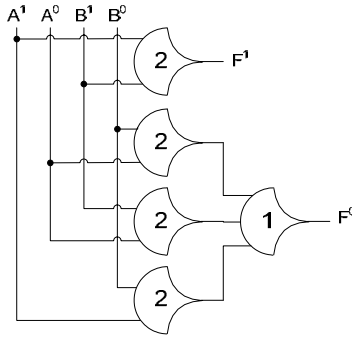


Figure 13. Decomposed NCL AND function [8].

5.3 Comparison of NCL Multipliers and ALUs

Table 4 shows a comparison of a variety of NCL multiplier architectures mapped to the LE developed herein and to the one designed in [11], demonstrating that the LE designed herein requires substantially less area compared to the one from [11]. Table 5 shows a comparison of a variety of NCL ALUs mapped to the two reconfigurable NCL LEs being compared, demonstrating again that the LE designed herein requires substantially less area compared to the one from [11].

6. CONCLUSIONS AND FUTURE WORK

This paper details the design of a reconfigurable NCL LE that can be programmed as any of the 27 fundamental NCL gates, shown in Table 1, and supports resetting and output inversion capability. The developed LE was compared with an alternative reconfigurable NCL LE described in [11] for a number of NCL circuits, showing that the one designed herein requires 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. Likewise, on average, using the LE in [11] requires 43% more gates and 28% more transistors for the ALUs.

Further analysis suggests that the LE developed herein is even more advantageous than previously explained when compared to the reconfigurable LE in [11]. Tables 4 and 5 show that using the reconfigurable LE from [11] results in substantially more gates being required than for the one developed herein, because many of the original circuit's gates need to be decomposed, as explained in Section 5.2. 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 are not considered. Doing so would further increase the area overhead when comparing the reconfigurable LE in [11] to the one 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 5.2, resulting in a slower circuit implementation. Therefore, using the LE developed herein should result in a faster implementation, requiring less area, compared to the reconfigurable LE in [11], for most non-trivial NCL circuits.

Table 4. Reconfigurable gate comparisons for NCL multipliers

Multiplier Architecture	Number of Gates		Number of Transistors	
	[11]	LE herein	[11]	LE herein
Dual-Rail Non-Pipelined [9]	176	141	24,992	22,419
Dual-Rail Full-Word Pipelined [9]	400	334	56,800	53,106
Dual-Rail Bit-Wise Pipelined [9]	365	275	51,830	43,725
Quad-Rail Non-Pipelined [32]	418	235	59,356	37,365
Quad-Rail Full-Word Pipelined [33]	523	268	74,266	42,612
Quad-Rail Bit-Wise Pipelined [33]	484	229	68,728	36,411

Table 5. Reconfigurable gate comparisons for NCL ALUs

ALU Architecture	Number of Gates		Number of Transistors	
	[11]	LE herein	[11]	LE herein
Dual-Rail Non-Pipelined [31, 34]	398	252	56,516	40,068
Dual-Rail Pipelined [31, 34]	696	513	98,832	81,567
Dual-Rail NULL Cycle Reduced [31, 34]	886	594	125,812	94,446
Quad-Rail Non-Pipelined [31, 34]	792	524	112,464	83,316
Quad-Rail Pipelined [31, 34]	1234	938	175,228	149,142
Quad-Rail NULL Cycle Reduced [31, 34]	1677	1141	238,134	181,419

An alternative reconfigurable NCL LE could also be designed that would allow for embedded registration to be utilized even for 4-input gates, which would be slightly larger and slower than the version presented herein. However, since fewer gates may be needed when using this 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.

The 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. TH_{nn} gates) and OR gates (i.e. TH_{1n} gates). Furthermore, these 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.

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 and connection of LEs 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 2-wire and 4-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.

7. REFERENCES

- [1] J. McCardle and D. Chester, "Measuring an Asynchronous Processor's Power and Noise," *Synopsys User Group Conference (SNUG)*, Boston, 2001.
- [2] G. E. Sobelman and K. M. Fant, "CMOS Circuit Design of Threshold Gates with Hysteresis," *IEEE International Symposium on Circuits and Systems (II)*, pp. 61-65, 1998.
- [3] Arun Balasubramanian, "An Asynchronous FPGA for NULL Convention Logic Circuits," Master's Thesis, Department of Electrical & Computer Engineering, University of Missouri – Rolla, January 2005.
- [4] K. M. Fant, *Logically Determined Design: Clockless System Design with NULL Convention Logic*, John Wiley and Sons, Inc., Hoboken, NJ, 2005.
- [5] A. J. Martin, "Programming in VLSI," in *Development in Concurrency and Communication*, Addison-Wesley, pp. 1-64, 1990.
- [6] K. Van Berkel, "Beware the Isochronic Fork," *Integration, The VLSI Journal*, Vol. 13/2, pp. 103-128, 1992.
- [7] A. Kondratyev, L. Neukom, O. Roig, A. Taubin, and K. Fant, "Checking delay-insensitivity: 104 gates and beyond," *Eighth International Symposium on Asynchronous Circuits and Systems*, pp. 137-145, 2002.
- [8] S. C. Smith, R. F. DeMara, J. S. Yuan, D. Ferguson, and D. Lamb, "Optimization of NULL Convention Self-Timed Circuits," *Integration, The VLSI Journal*, Vol. 37/3, pp. 135-165, 2004.
- [9] S. C. Smith, R. F. DeMara, J. S. Yuan, M. Hagedorn, and D. Ferguson, "Delay-Insensitive Gate-Level Pipelining," *Integration, The VLSI Journal*, Vol. 30/2, pp. 103-131, 2001.
- [10] E. Keller, "Building Asynchronous Circuits with JBits," *11th International Conference on Field Programmable Logic and Applications*, 2001.
- [11] D. R. Lamb, "Self-Timed Circuits for Adaptive Processing Systems," *Military and Aerospace Applications of Programmable Devices and Technologies Conference*, 1998.
- [12] C. L. Seitz, "System Timing," in *Introduction to VLSI Systems*, Addison-Wesley, pp. 218-262, 1980.
- [13] D. E. Muller, "Asynchronous Logics and Application to Information Processing," in *Switching Theory in Space Technology*, Stanford University Press, pp. 289-297, 1963.
- [14] K. M. Fant and S. A. Brandt, "NULL Convention Logic: A Complete and Consistent Logic for Asynchronous Digital Circuit Synthesis," *International Conference on Application Specific Systems, Architectures, and Processors*, pp. 261-273, 1996.
- [15] S. C. Smith, "Completion-Completeness for NULL Convention Digital Circuits Utilizing the Bit-wise Completion Strategy," *International Conference on VLSI*, pp. 143-149, 2003.

- [16] N. P. Singh, "A Design Methodology for Self-Timed Systems," Master's Thesis, MIT/LCS/TR-258, Laboratory for Computer Science, MIT, 1981.
- [17] T. S. Anantharaman, "A Delay Insensitive Regular Expression Recognizer," *IEEE VLSI Technology Bulletin*, Sept. 1986.
- [18] Ilana David, Ran Ginosar, and Michael Yoeli, "An Efficient Implementation of Boolean Functions as Self-Timed Circuits," *IEEE Transactions on Computers*, Vol. 41, No. 1, pp. 2-10, 1992.
- [19] J. Sparso, J. Staunstrup, M. Dantzer-Sorensen, "Design of Delay Insensitive Circuits using Multi-Ring Structures," *Proceedings of the European Design Automation Conference*, pp. 15-20, 1992.
- [20] S. Hauck, S. Burns, G. Borriello, and C. Ebeling, "An FPGA for Implementing Asynchronous Circuits," *IEEE Design & Test of Computers*, Vol. 11, No. 3, pp 60-69, 1994.
- [21] K. Maheswaran, "Implementing Self-Timed Circuits in Field Programmable Gate Arrays," Mater's Thesis, University of California – Davis, 1995.
- [22] R. E. Payne, "Self-Timed FPGA Systems," *5th International Workshop on Field Programmable Logic and Applications*, pp. 21-35, 1995.
- [23] C. Traver, R. B. Reese, and M. A. Thornton, "Cell Designs for Self-Timed FPGAs," *14th Annual IEEE International ASIC/SOC Conference*, pp. 175-179, 2001.
- [24] M. Aydin and C. Traver, "Implementation of a Programmable Phased Logic Cell," *45th Midwest Symposium on Circuits and Systems*, Vol. 2, pp. 21-24, 2002.
- [25] J. Teifel and R. Manohar, "An Asynchronous Dataflow FPGA Architecture," *IEEE Transactions on Computers*, Vol. 53, No. 11, pp. 1376-1392, 2004.
- [26] C. G. Wong, A. J. Martin, and P. Thomas, "An Architecture for Asynchronous FPGAs," *IEEE International Conference on Field-Programmable Technology*, pp. 170-177, 2003.
- [27] R. Manohar, "Asynchronous Reconfigurable Logic," *Custom Integrated Circuits Conference*, 2006.
- [28] K. Meekins, D. Ferguson, and M. Basta, "Delay Insensitive NCL Reconfigurable Logic," *IEEE Aerospace Conference*, Vol. 4, pp. 1961-1966, 2002.
- [29] D. H. Linder and J. H. Harden, "Phased logic: supporting the synchronous design paradigm with delay-insensitive circuitry," *IEEE Transactions on Computers*, Vol. 45/9, pp. 1031-1044, 1996.
- [30] C. G. Wong and A. J. Martin, "High-Level Synthesis of Asynchronous Systems by Data-Driven Decomposition," *Design Automation Conference*, pp. 508-513, 2003.
- [31] S. K. Bandapati and S. C. Smith, "Design and Characterization of NULL Convention Arithmetic Logic Units," *International Conference on VLSI*, pp. 178-184, 2003.
- [32] S. K. Bandapati, S. C. Smith, and M. Choi, "Design and Characterization of NULL Convention Self-Timed Multipliers," *IEEE Design and Test of Computers: Special Issue on Clockless VLSI Design*, Vol. 30/6, pp. 26-36, November-December 2003.
- [33] S. C. Smith, "Designing NULL Convention Combinational Circuits to Fully Utilize Gate-Level Pipelining for Maximum Throughput," *The 2004 International Conference on VLSI*, pp. 407-412, June 2004.
- [34] S. K. Bandapati and S. C. Smith, "Design and Characterization of NULL Convention Arithmetic Logic Units," to be published in *The Microelectronic Engineering Journal: Special Issue on VLSI Design and Test*.