

Design of a NULL Convention Self-Timed Divider

Scott C. Smith

*University of Missouri – Rolla, Department of Electrical and Computer Engineering
133 Emerson Electric Co. Hall, 1870 Miner Circle, Rolla, MO 65409
Phone: (573) 341-4232, Fax: (573) 341-4532, E-mail: smithsco@umr.edu*

Keywords: Asynchronous logic design, delay-insensitive circuits, computer arithmetic, iterative divider, NULL Convention Logic (NCL)

Abstract

An unsigned 8-bit \div 4-bit delay-insensitive iterative divider is developed using the NULL Convention Logic paradigm. The divider is simulated using a 0.5 μ m CMOS process with static cells. The simulation of the initial design yielded an average cycle time of 75.92 ns with a transistor count of 4,721. A subsequent design increased throughput by 18%, while only increasing area by 2.7%.

1. Introduction

For the last two decades, digital design has focused primarily on synchronous, clocked architectures. However, because clock rates have significantly increased while feature size has decreased, clock skew has become a major problem. To achieve acceptable skew, high-performance chips must dedicate increasingly larger portions of their area to clock drivers, thus dissipating increasingly higher power, especially at the clock edge, when switching is most prevalent.

As this trend continues, the clock is becoming more difficult to manage, causing renewed interest in asynchronous digital design. Researchers have demonstrated that correct-by-construction asynchronous paradigms, particularly NULL Convention Logic (NCL), require less power, generate less noise, produce less electromagnetic interference, and allow easier reuse of components than their synchronous counterparts, without compromising performance [1]. Furthermore, these paradigms should allow much greater flexibility in the design of complex circuits such as Systems-on-a-Chip (SoCs). Because these circuits are delay-insensitive, they should drastically reduce the effort required to ensure correct operation under all timing scenarios, compared to equivalent synchronous designs. Also, the self-timed nature of correct-by-construction SoCs should allow

designers to reuse previously designed and verified functional blocks in subsequent designs, without significant modifications or retiming effort within a reused functional block. Such SoCs may also provide for simpler interfacing between the digital core and nontraditional functional blocks.

One of the first tasks necessary to help integrate NCL into the semiconductor design industry is to develop and characterize the key components of a reusable-design library. Of fundamental importance are arithmetic circuits, including the divider described in this article, as well as the multipliers [2], ALUs [3], MACs [4], and counters [5] described elsewhere.

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 [6]. Like other delay-insensitive logic methods, the NCL paradigm assumes that forks in wires are isochronic [7]. 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 [8].

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 [9]. In particular, dual- 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. For NCL and other circuits to be purely delay-insensitive, assuming isochronic wire forks [7], they must meet the input-completeness and observability criteria [10].

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 [6], 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 [11]. 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 [7], 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. Furthermore, when circuits use the bit-wise completion strategy with selective input-incomplete components, they must also adhere to the completion-completeness criterion [10], 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 [6, 9, 12], 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 , 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 [6, 12], which use only one type of state-holding gate, the C-element [8]. 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 [13]. The primary type of threshold gate is the TH mn gate ($1 \leq m \leq n$). TH mn gates have n inputs. At least m of the n inputs must be asserted before the output becomes asserted. Because NCL threshold gates are designed with *hysteresis*, all asserted inputs must be deasserted before the output is deasserted. Hysteresis ensures a complete transition of inputs back to NULL before asserting the output associated with the next wavefront of input data. NCL threshold gates may also include a *reset* input to initialize the output. 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.

3. Previous Work

Previous work on asynchronous division has concentrated on design of self-timed circuits using precharged dynamic logic blocks [14, 15, 16, 17, 18]. Of these, most are implemented using differential cascode voltage switch logic (DCVSL), or variations thereof. While these circuits are self-timed, they are not delay-insensitive, as is the NCL divider developed herein. These self-timed circuits do contain timing assumptions, such as assuming that the skew between bits is less than the delay of the OR gate used to detect completion of each bit, for some circuits like a carry-save adder (CSA) for example [14]. This assumption allows for the completion tree to be omitted; thus decreasing area and potentially increasing throughput. Since NCL is delay-insensitive, this assumption is not allowed; however, the completion tree can still be eliminated in select NCL circuits, such as the CSA, without compromising delay-insensitivity, by utilizing the bit-wise completion strategy, where the completion signal for bit b in register $_i$ is only sent back to those bits in register $_{i-1}$ that took part in the generation of bit b [10]. Due to these fundamental differences, the NCL divider developed herein is not directly comparable to previous designs. However, it is expected that the NCL divider will consume less power because of its static nature, but in return will require more area and will not be as fast as its dynamic counterparts.

4. Unsigned Division

Figure 1 shows a hardware realization of the sequential division algorithm for unsigned integers. For an N -bit dividend, Z , and an M -bit divisor, D , the $(N+M)$ -bit Partial Remainder (PR) register is initially loaded with M zeros concatenated with Z . For each iteration, $j \in [1, N]$, the trial difference: $PR(N+M-1: N-1) - D$ is calculated to generate each successive bit of the quotient, Q_{N-j} . Therefore D is aligned with the upper $M+1$ bits of PR for the trial subtraction, and the lower part of PR is not affected. Q_{N-j} is logic 1 if the trial subtraction is greater than or equal to zero, which is true if either the *MSB* is logic 1 or if C_{out} is logic 1; otherwise Q_{N-j} is logic 0.

The Partial Remainder register is updated after each iteration. If Q_{N-j} is logic 0, then the Partial Remainder register is shifted one bit position to the left; otherwise the upper M bits of the Partial Remainder register are loaded with the result of the trial difference and the lower $N-1$ bits are shifted one bit position to the left. This division scheme is referred to as restoring division, since the partial remainder is restored to its correct value, and not loaded with the trial difference, if the trial difference is

Initially asserting the *Reset* signal initializes the NCL registration components to either NULL (N) or DATA0 (D0). The input multiplexers first load the dividend, Z , concatenated with four leading zeros, into the Partial Remainder register and the inverted divisor, D , into the Divisor register. In each of the subsequent 7 iterations, the newly generated partial remainder is fed back into the Partial Remainder register through a three-register feedback loop, as required for delay-insensitivity [12], while the divisor is fed back unchanged to the Divisor register. In the 8th iteration, the Partial Remainder register and Divisor register are loaded with the next set of input operands, while the quotient, Q , and remainder, R , are loaded into the Select registers to be output.

5.1 Sequencers

There are two Sequencers, an Input Sequencer that controls the loading of the input operands, D and Z , and an Output Sequencer that controls the loading of the quotient and remainder into their respective Select registers. The Input Sequencer, shown in Figure 4, is controlled by the input Multiplexer registers. The outputs of the Input Sequencer, $S0$ and SI , are used to select between the external inputs and the feedback inputs for the input Multiplexer registers.

The Output Sequencer, shown in Figure 5, is controlled by both the partial remainder feedback loop, as well as by the quotient and remainder Select registers. The outputs of the Output Sequencer, $S0$ and SI , are used to mask the Select registers' requests for iterations 1 through 7, and to output Q and R on the 8th iteration, respectively. This sequencer, along with the AND gate and subsequent completion component, are used to preserve the delay-insensitivity of the divider, despite the fact that the Select registers only accept DATA every eighth iteration. With the initial system reset, the Select registers are reset to NULL, such that they request DATA, and $S0$ is reset to logic 1. The output registers in the partial remainder feedback loop are initially reset to DATA0, such that the partial remainder Multiplexer registers, which are input-complete with respect to their feedback inputs, can load the initial dividend. After the initial dividend is loaded, the output registers in the partial remainder feedback loop will transition to NULL, causing them to request DATA, such that the output sequencer's K_i will become asserted, thus starting the sequencer's cycle. SI controls the loading of the Select registers, while $S0$ controls the masking of the registers' request signals, K_o , and mimics the requesting of

DATA/NULL wavefronts for the first seven iterations. SI is only asserted in cycle 15, which allows the quotient and remainder to be loaded into their respective Select registers to be output, only after the eighth iteration when they are the final, correct values. $S0$ is asserted in cycles 2, 4, 6, 8, 10, 12, and 14 in order to mimic the request for DATA and request for NULL from the Select registers, which is masked by the AND gate for the first seven iterations, since these registers are not receiving the DATA wavefronts, as they are only being fed back to the Partial Remainder register. Thus K_o from the Select registers is not changing. Instead, the Output Sequencer and internal iterations are only being controlled by the feedback loop. $S0$ is again asserted in cycle 15 to ensure that the DATA wavefront is received by the Select registers, which occurs when K_o becomes *rfn* (request for NULL), causing the AND gate to become de-asserted. $S0$ remains asserted in cycle 16, to ensure that the NULL wavefront is received by the Select registers, which occurs when K_o becomes *rfd* (request for DATA), causing the AND gate to once again become asserted, thus requesting the first iteration of the next division operation. Following this, K_o is once again masked, since the outputs of the next seven iterations will not be sent to the Select registers. Therefore this structure retains delay-insensitivity since it ensures that the sequencer and internal iterations are only controlled by the feedback loop when the intermediate results are not being sent to the Select registers, and are controlled by both the feedback loop and the Select registers during the eighth iteration when the quotient and remainder are both being sent to the Select registers and being used for input-completeness when loading the next dividend.

Both Sequencers are 16-stage single-rail ring structures with 7 tokens, where a token is defined as a DATA wavefront with corresponding NULL wavefront, and two bubbles, where a bubble is defined as either a DATA or NULL wavefront occupying more than one neighboring stage [12]. When K_i becomes *rfd* the DATA wavefront moves through the two NULL bubbles ahead of it, creating two DATA bubbles in its wake. Likewise, when K_i becomes *rfn* the NULL wavefront moves through the two DATA bubbles ahead of it, creating two NULL bubbles in its wake. The DATA/NULL wavefront restricts the forward propagation of the NULL/DATA wavefront, respectively, for each change of K_i , limiting the forward propagation to only the two bubbles. The cycle for the two outputs of the Input Sequencer is shown in boldface in Table 1, and the cycle for the two outputs of the Output Sequencer is shown in boldface in Table 2.

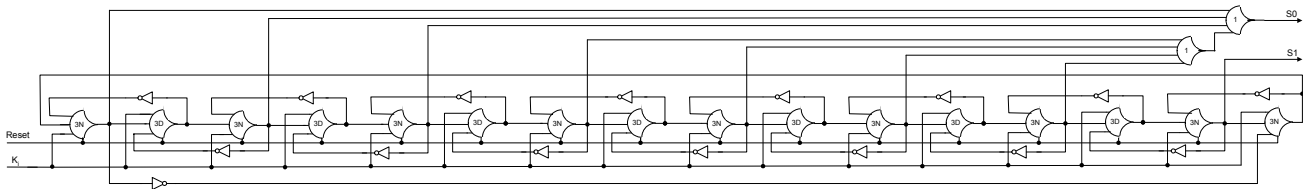


Figure 4. Input sequencer logic diagram.

Table 1: Input sequencer output table.

Cycle#	Initial State	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K_i	X	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
S_0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
S_1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

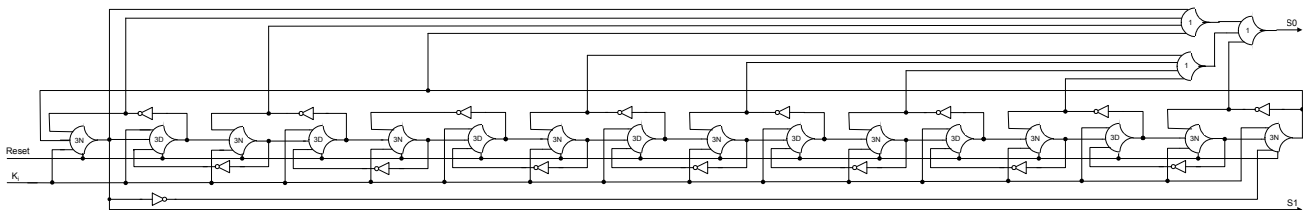


Figure 5. Output sequencer logic diagram.

Table 2: Output sequencer output table.

Cycle#	Initial State	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K_i	X	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
S_0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	1
S_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

5.2 Multiplexer Registers

There are three different types of Multiplexer Registers used in the divider design, differing in their input-completeness and data inputs. All three types utilize embedded registration, where the registration is embedded within the combinational logic in order to reduce delay and area. The following logic diagrams show a single bit slice of each Multiplexer Register type. Multi-bit Multiplexer Registers are created by simply replicating their respective bit slice.

The first, MUX Reg, is input-incomplete with respect to its data inputs, $D0$ and $D1$, as shown in Figure 6. This is allowable because the combinational circuitry, consisting of the RCA, OR function, multiplexer, and feedback of the divisor, is input-complete with respect to all inputs, thus allowing some components, such as the

multiplexer, to be input-incomplete. S selects between the $D0$ and $D1$ inputs, such that $F = D0$ when S is DATA0 and $F = D1$ when S is DATA1.

The second, MUX COMP0 Reg, is input-complete with respect to its $D0$ input, but not with respect to its $D1$ input, as shown in Figure 7. This is because $D0$ is DATA every iteration, but $D1$ is only DATA every 8th iteration, when a new dividend and divisor are being input. $S0$ and $S1$ select between the $D0$ and $D1$ inputs, such that $F = D0$ when $S0$ is asserted and $F = D1$ when $S1$ is asserted. Note that the output of the TH12 gate, which ORs $D0^0$ and $D0^1$ together, is an input to the gates that pass $D1$, as well as to those that pass $D0$. While it is not necessary for this signal to be an input to the gates that pass $D0$ in order for the circuit to be input-complete with respect to $D0$, this is necessary for the TH12 gate to be observable when selecting input $D0$.

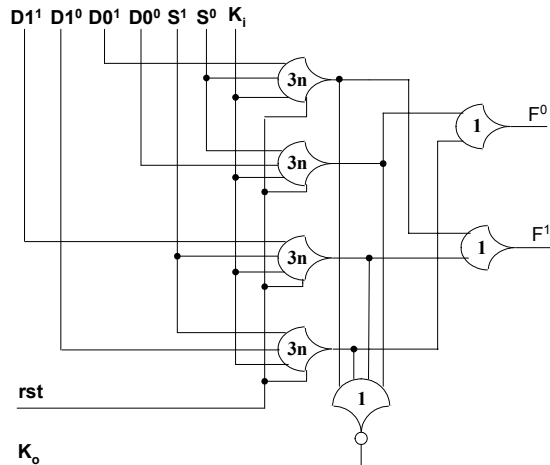


Figure 6. Bit slice of MUX Reg.

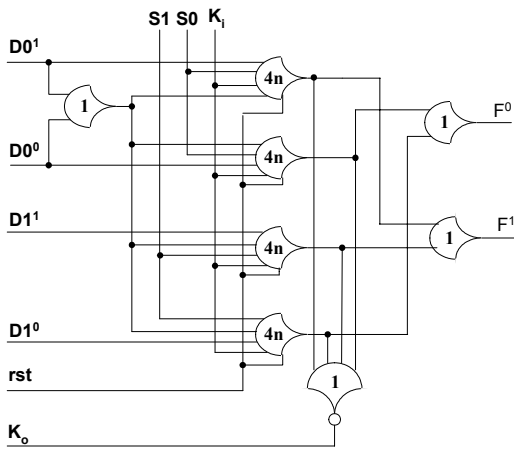


Figure 7. Bit slice of MUX Comp0 Reg.

The third, MUX0 COMP Reg, is similar to MUX COMP0 Reg; however it does not have a DI input, as shown in Figure 8. It is input-complete with respect to its $D0$ input; and $S0$ and $S1$ select between the $D0$ input and $DATA0$, such that $F = D0$ when $S0$ is asserted and $F = DATA0$ when $S1$ is asserted.

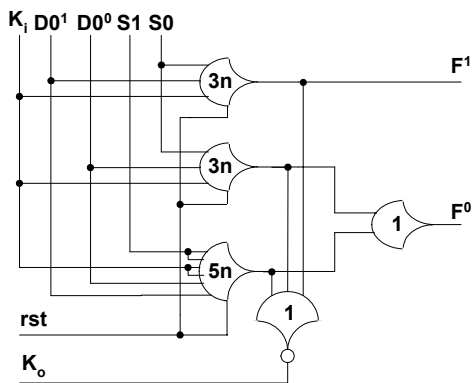


Figure 8. Bit slice of MUX0 Comp Reg.

5.3 Select Register

Both the Quotient and Remainder registers are Select registers, whose bit slice is shown in Figure 9. A Select register is similar to a standard NCL register [5] except that it has an additional input, S , which must be asserted in order for $DATA$ to pass through the register, and which must be deasserted in order for $NULL$ to pass through the register. Multi-bit Select registers are created by simply replicating the bit slice.

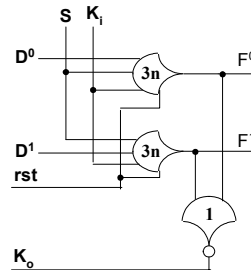


Figure 9. Bit slice of Select Reg.

6. Simulation Results

The divider was simulated using a $0.5\mu\text{m}$ CMOS process operating at 3.3V. Since NCL circuits are delay-insensitive, speed is data dependent; therefore average cycle time, T_{DD} , is calculated and used for comparison. T_{DD} is calculated as the arithmetic mean of the cycle times corresponding to all 3840 possible pairs of input operands for non-zero divisors. The testbench compares the outputs from the design and the values calculated using the standard VHDL functions. Two signals, *incorrect_q* and *incorrect_r*, are asserted whenever the design's quotient and remainder outputs don't match the calculated quotient and remainder values, respectively. This simulation was run for all 3840 combinations of the input operands and both *incorrect_q* and *incorrect_r* remained zero throughout, meaning that all results were correct.

Simulating the divider of Figure 3 yielded a T_{DD} of 75.92 ns with a transistor count of 4,721. To increase the throughput, a fourth register was added in the restrictive partial remainder feedback loop, such that the $DATA$ and $NULL$ wavefronts could move more independently [12]. This decreased T_{DD} to 64.33 ns, while increasing the transistor count to 4,849, resulting in an 18% increase in throughput with only a 2.7% increase in area. Transistor count is used to measure area instead of gate count, since NCL gates vary greatly in size (i.e. from 2 transistors for an inverter to 26 transistors for a TH24 gate); thus transistor count provides a better means of comparison. Note that a fourth register was not required in the divisor feedback loop, since this path was much faster than the partial remainder feedback loop. Adding a fourth register in the divisor feedback loop did not affect throughput.

7. Conclusions

This paper details the design of a 0.5 μ m unsigned 8-bit \div 4-bit delay-insensitive iterative divider, implemented using the NULL Convention Logic paradigm. This design differs from the previous work because its fundamental gates are implemented as static CMOS cells, whereas previous asynchronous dividers were implemented using dynamic logic cells. Furthermore, previous designs are self-timed, but not delay-insensitive; whereas the NCL divider is indeed delay-insensitive, making it even less susceptible to process variations and voltage fluctuations. Due to these fundamental differences, the NCL divider developed herein is not directly comparable to previous designs. However, it is expected that the NCL divider will consume less power because of its static nature, but in return will require more area and will not be as fast as its dynamic counterparts. This architecture can be readily utilized for larger width operands, such as for dividing the mantissas of IEEE floating-point numbers, by simply increasing the width of the registers, multiplexers, sequencers, and subtractor, accordingly.

The NCL divider does not assume that the most significant bit of the divisor is logic 1; hence, it takes eight iterations to compute the 8-bit quotient. If this bit was assumed to always be logic 1, the division could be performed in only four iterations and would be approximately twice as fast. Future NCL dividers could better exploit their self-timed nature by incorporating *early done* logic to detect when the partial remainder is less than the divisor and complete the division operation early in this case, without finishing the remaining iterations. Furthermore, the *early done* logic could also detect repeating quotient digits [14] and again terminate the remaining iterations. While both of these modifications would be expected to increase throughput, they would also increase area. Their effect on power usage is theoretically indeterminate and would depend on how much power was saved by finishing early and not performing the remaining iterations, when possible, versus how much extra power is used in the *early done* logic.

References

- [1] J. McCardle and D. Chester, "Measuring an Asynchronous Processor's Power and Noise," *Synopsys User Group Conference (SNUG)*, Boston, 2001.
- [2] 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.
- [3] S. K. Bandapati and S. C. Smith, "Design and Characterization of NULL Convention Arithmetic Logic Units," *The 2003 International Conference on VLSI*, pp. 178-184, June 2003.
- [4] S. C. Smith, R. F. DeMara, J. S. Yuan, M. Hagedorn, and D. Ferguson, "NULL Convention Multiply and Accumulate Unit with Conditional Rounding, Scaling, and Saturation," *Journal of Systems Architecture*, Vol. 47/12, pp. 977-998, June 2002.
- [5] 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*, accepted for publication, December 2003.
- [6] C. L. Seitz, "System Timing," in *Introduction to VLSI Systems*, Addison-Wesley, pp. 218-262, 1980.
- [7] C. H. (Kees) van Berkel, M. Rem, and R. Saeijs, "VLSI Programming," *1988 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 152-156, 1998.
- [8] D. E. Muller, "Asynchronous Logics and Application to Information Processing," in *Switching Theory in Space Technology*, Stanford University Press, pp. 289-297, 1963.
- [9] 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.
- [10] S. C. Smith, "Completion-Completeness for NULL Convention Digital Circuits Utilizing the Bit-wise Completion Strategy," *The 2003 International Conference on VLSI*, pp. 143-149, June 2003.
- [11] A. Kondratyev, L. Neukom, O. Roig, A. Taubin, and K. Fant, "Checking delay-insensitivity: 10^4 gates and beyond," *Eighth International Symposium on Asynchronous Circuits and Systems*, pp. 137-145, 2002.
- [12] J. Sparsø and J. Staunstrup, "Design and Performance Analysis of Delay Insensitive Multi-Ring Structures," *Twenty-Sixth Hawaii International Conference on System Sciences*, Vol. 1, pp. 349-358, 1993.
- [13] 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.
- [14] T. E. Williams and M. A. Horowitz, "A Zero-Overhead Self-Timed 160-ns 54-b CMOS Divider," *IEEE Journal of Solid State Circuits*, Vol. 26, No. 11, pp. 1651-1661, 1991.
- [15] M. Renaudin, B. E. Hassa, and A. Guyot, "A new Asynchronous Pipeline Scheme: Application to the Design of a Self-Timed Ring Divider," *IEEE Journal of Solid State Circuits*, Vol. 31, No. 7, pp. 1001-1013, 1996.
- [16] J. Chiang and J. Liao, "The Design and Implementation of an Asynchronous Radix-2 Non-Restoring 32-B/32-B Ring Divider," *IEEE International Symposium on Circuits and Systems*, Vol. 2, pp. 173-176, 1998.
- [17] J. Yang, C. Choy, and C. Chan, "A Self-Timed Divider Using a New Fast and Robust Pipeline Scheme," *IEEE Journal of Solid State Circuits*, Vol. 36, No. 6, pp. 917-923, 2001.
- [18] G. Cornetta and J. Cortadella, "A Multi-Radix Approach to Asynchronous Division," *Seventh International Symposium on Asynchronous Circuits and Systems*, pp. 25-34, 2001.
- [19] B. Parhami, *Computer Arithmetic Algorithms and Hardware Designs*, Oxford University Press, New York, 2000.