01 Jan 2007

Implementation of Static and Semi-Static Versions of a Bit-wise Pipelined Dual-rail NCL 2S Complement Multiplier

R. Sankar

V. Kadiyala

Ravi Bonam

S. Kumar

et. al. For a complete list of authors, see https://scholarsmine.mst.edu/ele_comeng_facwork/1080

Recommended Citation

R. Sankar et al., "Implementation of Static and Semi-Static Versions of a Bit-wise Pipelined Dual-rail NCL 2S Complement Multiplier," *Proceedings of the IEEE Region 5 Technical Conference, 2007*, Institute of Electrical and Electronics Engineers (IEEE), Jan 2007.
The definitive version is available at https://doi.org/10.1109/TPSD.2007.4380386

Implementation of Static and Semi-Static Versions of a Bit-Wise Pipelined Dual-Rail NCL 2^S Complement Multiplier

R. Sankar, V. Kadiyala, R. Bonam, S. Kumar, S. Mohan, F. Kacani, W. K. Al-Assadi[1], and S. C. Smith[2]
University of Missouri - Rolla, Department of Electrical and Computer Engineering
1870 Miner Circle, Rolla, MO 65409
Email: waleed@umr.edu[1] and smithsco@umr.edu[2]

***Abstract*—This paper focuses on implementing a 2^s complement 8×8 dual-rail bit-wise pipelined multiplier using the asynchronous NULL Convention Logic (NCL) paradigm. The design utilizes a Wallace tree for partial product summation, and is implemented and simulated in VHDL, the transistor level, and the physical level, using a 1.8V 0.18µm TSMC CMOS process. The multiplier is realized using both static and semi-static versions of the NCL gates; and these two implementations are compared in terms of area, power, and speed.**

I. INTRODUCTION

For the past few decades, the development of synchronous circuits has dominated the semiconductor design industry. However, as we see a growing need for more power efficient, higher performance, and more noise tolerant chips, the advantages offered by asynchronous logic paradigms, such as NULL Convention Logic (NCL) [1], should not be ignored.

This paper addresses the use of industry-standard CAD tools for designing NCL circuits. The multiplier was first designed at the gate-level using standard NCL design techniques. VHDL simulation was then performed to ensure functional correctness. We utilized an NCL VHDL library, with delays based on physical-level simulations of static gates designed using TSMC's 1.8V, 0.18µm CMOS technology [2]. Next, we converted the static NCL transistor-level library into a semi-static version [3], and directly converted the semi-static transistor-level library to the physical level, using a Schematic Driven Layout tool. Finally, we implemented the multiplier components at the physical level using an HDL driven layout scheme, and simulated these to obtain power, speed, and area comparisons for the static versus semi-static implementations.

The paper is organized as follows: Section II provides a brief overview of NCL; Section III describes the multiplier design and implementation at the various levels of abstraction; Section IV includes the simulation results and comparisons; and Section V concludes the paper.

The authors gratefully acknowledge the support from the National Science Foundation under CCLI grant DUE-0536343.

II. NCL OVERVIEW

NCL is a delay-insensitive asynchronous paradigm, which means that NCL circuits will operate correctly regardless of when circuit inputs become available; therefore NCL circuits are said to be correct-by-construction (i.e., no timing analysis is necessary for correct operation). NCL circuits utilize dual-rail logic to achieve delay-insensitivity. A dual-rail signal, D, consists of two wires, D^0 and D^1, which may assume any value from the set {DATA0, DATA1, NULL}. The DATA0 state ($D^0 = 1$, $D^1 = 0$) corresponds to a Boolean logic 0, the DATA1 state ($D^0 = 0$, $D^1 = 1$) corresponds to a Boolean logic 1, and the NULL state ($D^0 = 0$, $D^1 = 0$) corresponds to the empty set meaning that the value of D is not yet available. The two rails are mutually exclusive, such that both rails can never be asserted simultaneously; this state is defined as an illegal state.

NCL uses threshold gates as its basic logic elements [3]. The primary type of threshold gate, shown in Fig. 1, is the *THmn gate*, where $1 \le m \le n$. 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.

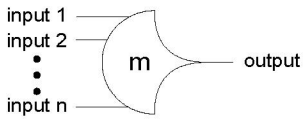

Fig. 1. THmn threshold gate.

Another type of threshold gate is referred to as a weighted threshold gate, denoted as THmnWw$_1$w$_2$…w$_R$. Weighted threshold gates have an integer value, $m \ge w_R > 1$, applied to *inputR*. Here $1 \le R < n$; where n is the number of inputs; m is the gate's threshold; and w_1, w_2, …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.

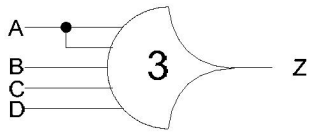

Fig. 2. TH34w2 threshold gate: Z = AB + AC + AD + BCD.

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 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 [3], 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.

TABLE I
27 FUNDAMENTAL NCL GATES

NCL Gate	Boolean Function	Transistors (static)	Transistors (semi-static)
TH12	A + B	6	6
TH22	AB	12	8
TH13	A + B + C	8	8
TH23	AB + AC + BC	18	12
TH33	ABC	16	10
TH23w2	A + BC	14	10
TH33w2	AB + AC	14	10
TH14	A + B + C + D	10	10
TH24	AB + AC + AD + BC + BD + CD	26	16
TH34	ABC + ABD + ACD + BCD	24	16
TH44	ABCD	20	12
TH24w2	A + BC + BD + CD	20	14
TH34w2	AB + AC + AD + BCD	22	15
TH44w2	ABC + ABD + ACD	23	15
TH34w3	A + BCD	18	12
TH44w3	AB + AC + AD	16	12
TH24w22	A + B + CD	16	12
TH34w22	AB + AC + AD + BC + BD	22	14
TH44w22	AB + ACD + BCD	22	14
TH54w22	ABC + ABD	18	12
TH34w32	A + BC + BD	17	12
TH54w32	AB + ACD	20	12
TH44w322	AB + AC + AD + BC	20	14
TH54w322	AB + AC + BCD	21	14
THxor0	AB + CD	20	12
THand0	AB + BC + AD	19	13
TH24comp	AC + BC + AD + BD	18	12

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 [1].

NCL systems contain at least two delay-insensitive registers, one at both the input and at the output. Two adjacent register stages interact through their request and acknowledge signals, K_i and K_o, respectively, to prevent the current DATA wavefront from overwriting the previous DATA wavefront, by ensuring that the two DATA wavefronts are always separated by a NULL wavefront. The acknowledge signals are combined in the Completion Detection circuitry to produce the request signal(s) to the previous register stage. NCL registration is realized through cascaded arrangements of single-bit dual-rail registers or single-signal quad-rail registers. These registers consist of TH22 gates that pass a DATA value at the input only when K_i is *request for data* (rfd) (i.e., logic 1) and likewise pass NULL only when K_i is *request for null* (rfn) (i.e., logic 0). They also contain a NOR gate to generate K_o, which is *rfn* when the register output is DATA and *rfd* when the register output is NULL.

An N-bit register stage, comprised of N single-bit dual-rail NCL registers, requires N completion signals, one for each register. The NCL completion component uses these K_o lines to detect complete DATA and NULL sets at the output of every register stage and request the next NULL and DATA set, respectively. In full-word completion, the single-bit output of the completion component is connected to all K_i lines of the previous register stage. Since the maximum input threshold gate is the TH44 gate, the number of logic levels in the completion component for an N-bit register is given by $\lceil \log_4 N \rceil$. On the other hand, bit-wise completion only sends the completion signal from bit *b* in register$_i$ back to the bits in register$_{i-1}$ that took part in the calculation of bit *b*. This method may therefore require fewer logic levels than that of full-word completion, thus increasing throughput.

III. DESIGN AND IMPLEMENTATION

The multiplier implementation utilized the Modified Baugh-Wooley algorithm [4] for partial product generation and a Wallace tree for partial product summation. The multiplier was then pipelined utilizing bit-wise completion in order to maximize throughput, reducing the average DATA-DATA cycle time, T_{DD}, from 46 gate delays for the original non-pipelined design to 4 gate delays. This required an additional 14 internal register stages of decreasing width. Note that at the last stage of the Wallace tree, a Ripple Carry Adder (RCA) was used instead of a Carry Look-Ahead Adder (CLA), since the performance of asynchronous circuits depends on average-case delay, not worst-case delay, as in synchronous circuits; and both CLAs and RCAs have the same average case delay of O(Log N) for an N-bit adder. Furthermore, an NCL RCA is much more area efficient than its equivalent CLA, and can be pipelined with less difficulty. Hence, a RCA is the preferred choice [5].

A block diagram of the multiplier is shown in Fig. 3, and its components are listed and described below:

1) HA: This component is an NCL half adder. It is inherently input-complete, has a delay of 2 gates, and utilizes 7 gates.
2) FA: This component is an NCL full adder. It is also inherently input-complete, has a delay of 2 gates, and utilizes 12 gates.

3) FA1: This is a specialized full adder component, where one of the inputs is always logic 1. It is inherently input-complete, has a delay of 2 gates, and utilizes 7 gates.
4) HA1: This is a specialized half adder component, where one of the inputs is always logic 1. Its *carry* output is equal to its input and its *sum* output is the inverse of its input; hence, it does not require any gates, since a dual-rail inverter is realized by simply swapping rails.
5) AND2c: This is an input-complete two-input AND function, used for partial product generation. It has a delay of 2 gates, and utilizes 5 gates.
6) NAND2c: This is an input-complete two-input NAND function, used for partial product generation. It has a delay of 2 gates, and utilizes 5 gates.

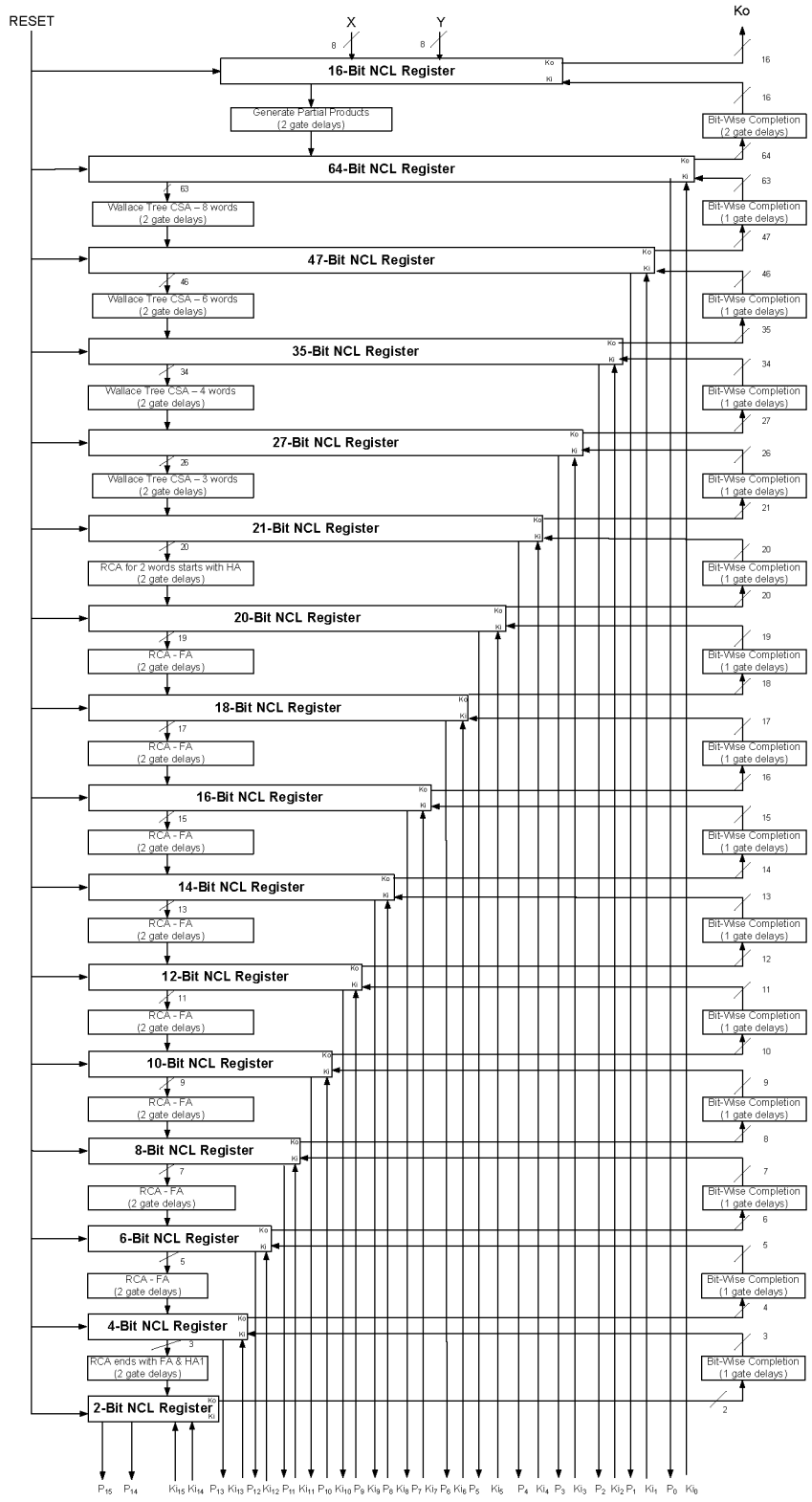

Fig. 3. Multiplier block diagram.

A. VHDL Implementation

Each of the above components was implemented as a gate-level structural design; and these were combined with generic registration and completion components [2] to form a hierarchical design of the entire 8×8 multiplier. The complete gate-level structural VHDL version of the multiplier was then simulated using an exhaustive VHDL testbench, and was verified to be functionally correct. The simulation yielded a T_{DD} of 2.6 ns, using gate delays based on physical-level simulations with TSMC's 1.8V, 0.18μm CMOS technology.

B. Transistor-Level Implementation

As explained in Section II, NCL threshold gates are designed with *hysteresis* state-holding capability, such that after the output is asserted, all inputs must be deasserted before the output will be deasserted. Therefore, NCL gates have both *set* and *hold* equations, where the *set* equation determines when the gate will become asserted and the *hold* equation determines when the gate will remain asserted once it has been asserted. The *set* equation determines the gate's functionality as one of the 27 NCL gates, as listed in Table I, whereas the *hold* equation is the same for all NCL gates, and is simply all inputs ORed together. The general equation for an NCL gate with output Z is: $Z = set + (Z^{-} \bullet hold)$, where Z^{-} is the previous output value and Z is the new value. Take the TH23 gate for example. The *set* equation is AB + AC + BC, as given in Table I, and the *hold* equation is A + B + C; therefore the gate is asserted when at least 2 inputs are asserted and it then remains asserted until all inputs are deasserted.

To implement an NCL gate using CMOS technology, an equation for the complement of Z is also required, which in general form is: $Z' = reset + (Z^{-\prime} \bullet set')$, where *reset* is the complement of *hold* (i.e., the complement of each input, ANDed together), such that the gate is deasserted when all inputs are deasserted and remains deasserted while the gate's *set* condition is false. For the TH23 gate, the *reset* equation is A'B'C' and the simplified *set'* equation is A'B' + B'C' + A'C'. Directly implementing these equations for Z and Z', after simplification, yields the static transistor-level implementation of an NCL gate, as shown in Fig. 4 for the TH23 gate. This requires the output, Z, to be fedback as an input to the NMOS and PMOS logic to achieve hysteresis behavior.

NCL gates can also be implemented in a semi-static fashion, where a weak feedback inverter is used to achieve hysteresis behavior, which only requires the *set* and *reset* equations to be implemented in the NMOS and PMOS logic, respectively. The semi-static TH23 gate is shown in Fig. 5. In general, the semi-static implementation requires fewer transistors, but is slightly slower because of the weak inverter. Note that TH1n gates are simply OR gates and do not require any feedback, such that their static and semi-static implementations are exactly the same.

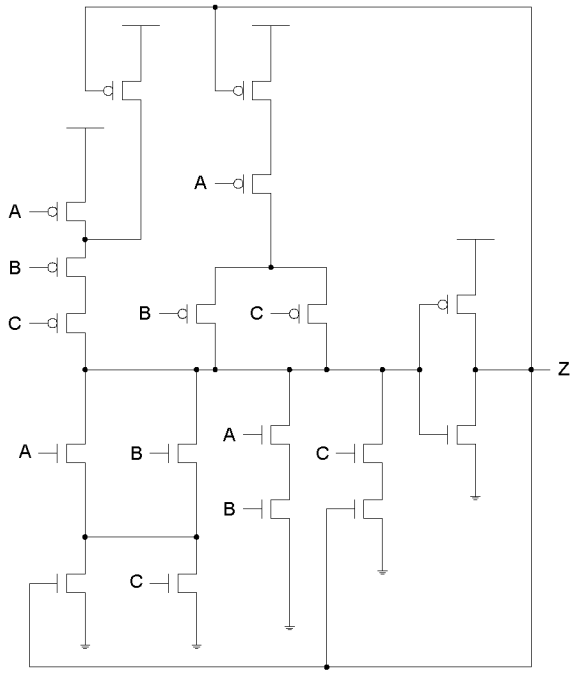

Fig. 4. Static CMOS implementation of a TH23 gate: Z = AB + AC + BC.

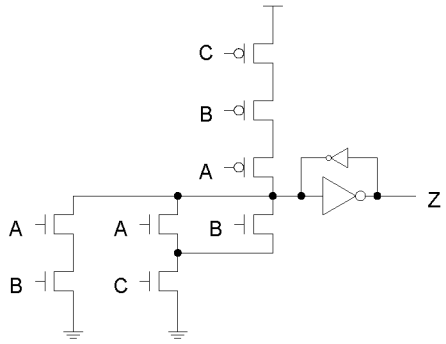

Fig. 5. Semi-static CMOS implementation of a TH23 gate: Z = AB + AC + BC.

Transistor-level libraries have been created for both the static and semi-static versions of all the NCL gates used in the design, utilizing Mentor Graphics Design Architect (DA) tool. All of the DA gates were then simulated with Accusim to verify the design by observing the I/O waveforms. For the static version, minimum widths were used for all transistors to maximize gate speed; however, for the semi-static version, larger transistors were required to overcome the weak feedback inverter to obtain proper gate functionality and reduce propagation delay.

According to the TSMC018 rules, the ratio of widths of the PUN and PDN networks should be around 2.42. Therefore, for the semi-static gates, the size of the strong inverter is set as Wp=7, Wn=3, L=2; and the weak inverter is sized as Wp=7, Wn=3, L=7. The rest of the gate's transistors are sized according to W_{PUN}/W_{PDN}=7/3, L=2.

C. Physical-Level Implementation

Using Mentor Graphics IC Station, a physical layout was

created for both static and semi-static versions of all NCL gates used in the design, using the Schematic Driven Layout (SDL) tool. All the NCL gate layouts were checked for coherency using Design Rules Check (DRC) and Layout Versus Schematic (LVS), and then made into standard cells. The gate layouts were then standardized so that they can be used in hierarchical layout generation for the multiplier components. Standardization means that a gate is defined at the block level, such that it is understood by the tool as a standard block. To do this, the cell is first set as a standard cell, and the site type is specified as '1' so that the blocks can be placed side by side. Next, a floor plan is created to enclose the standard cell with floor plan blocks; the cell is aligned at the origin of the sheet; and V_{DD} and GND are implemented as ports using Metal1, and all other ports implemented using Metal2, without using Metal#.port layers, and placing the ports close to the cell boundary.

Once the gates are standardized, they are utilized to implement the physical-level design of the basic multiplier components, HA, FA, HA1, FA1, AND2c, and NAND2c. SDL used to implement the NCL gates becomes more difficult to use as the complexity of the circuit increases; hence, a Verilog driven layout method is utilized instead to perform automated layout (i.e., using the $autoplace_standard_cells() command), where the Verilog netlist is generated from the gate-level structural VHDL file, using Leonardo Spectrum.

The most complex task is routing between instances, which show up as overflows in IC Station, and which increase with the complexity of the design. These can be routed using the Auto Route option as follows:

1) Set the peek-on-view option to eliminate shorts between metals.
2) Enable metal routing in both directions to provide the router with more flexibility.
3) Execute the RIP command to start routing. This might increase the number of overflows, but it helps yield better routing.
4) Execute the RUN command to reduce the number of overflows.

Steps 3 and 4 are repeated until the number of overflows is reduced to between 10 and 20. The remaining 10-20 overflows are then manually routed; and the V_{DD} and GND ports are made using Metal1, and the remaining ports made using Metal2.

Now, a netlist file is generated from the component layout, and is changed to a `CIR` file by including information for input waveform generation and plotting the results. Also, the TSMC018 library needs to be included using the `.lib` command.

After generating the corresponding `CIR` file, the component can then be simulated using the Eldo tool, which creates a `COU` file. Eldo simulation provides information about floating gates or improperly connected nets, and is therefore very useful for debugging the `CIR` file. Eldo also calculates the component's power dissipation. After the Eldo simulation is complete without any errors or warnings, the simulation results can be viewed using EZwave to check the I/Os for functional correctness, and measure a variety of parameters, such as Rise Time, Fall Time, Propagation Delay, etc.

D. Design Flow

The overall design flow is depicted in Fig. 6. Note that the transistor-level and physical-level NCL gate libraries were only created for the semi-static gates, because these libraries are available on the web for static NCL gates [6].

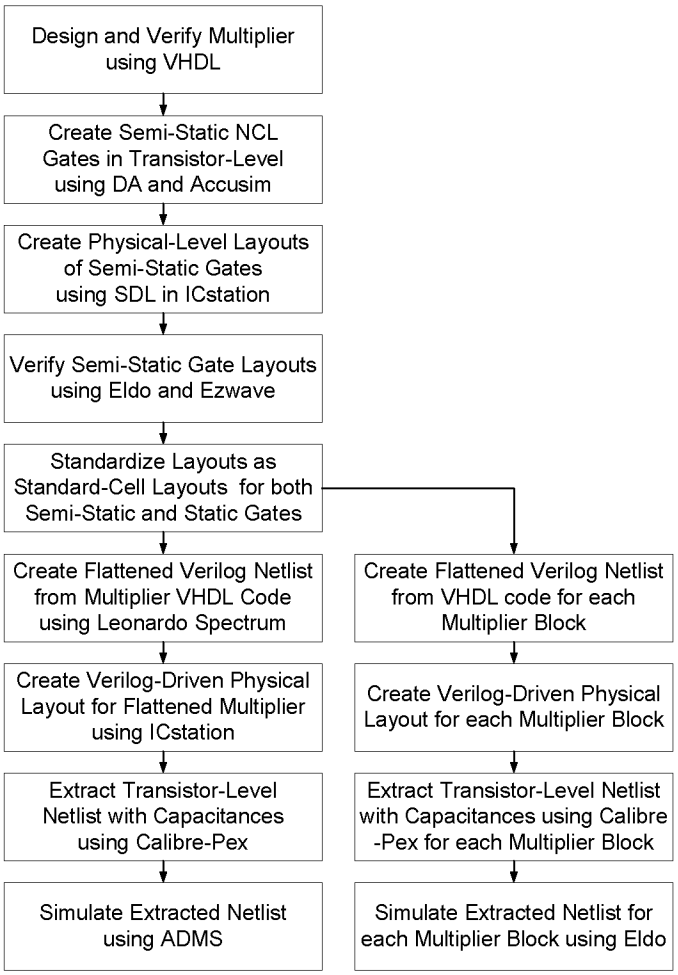

Fig. 6. NCL multiplier design flow.

IV. RESULTS AND COMPARISON

The physical-level simulation results for the static and semi-static multiplier components, designed using a 1.8V 0.18μm TSMC CMOS process, are shown below in Table II. Power is the average power reported by Eldo for an exhaustive test of all input combinations, using 10 ns for both the DATA and NULL wavefront widths. This shows that power dissipation and area is less for the semi-static implementation, as expected, since far fewer transistors are required because the gate feedback logic is replaced with a weak inverter for semi-static gate implementation. This reduces power dissipation because fewer transistors are switching and there are fewer nodes to contribute to charge-sharing. Propagation delay varies depending on the size of the

component. Small components, such as AND2c and NAND2c, are faster when implemented using semi-static gates than with static gates. However, for larger circuits, such as HA, FA1, and FA, the static implementation is faster.

TABLE II
ANALYSIS OF MULTIPLIER COMPONENTS

Component	Power (pW)		Delay (ps)		Area (μm^2)	
	Static	Semi-static	Static	Semi-static	Static	Semi-static
AND2c	201	184	497	376	1844	1543
Nand2c	201	184	497	376	1844	1458
HA	321	312	529	668	1941	1669
FA1	321	312	511	635	1941	1675
FA	682	630	645	755	3164	3006

The overall system-level layout of the multiplier has been completed, requiring 0.330836 mm^2 for the static version and 0.305261 mm^2 for the semi-static version. *ADVance MS* (ADMS), an extension of Mentor Graphics Eldo simulator, has been utilized to simulate the static and semi-static physical-level designs using a VHDL testbench, as described in [7], resulting in an average DATA-to-DATA cycle time (T_{DD}) of 5.642 ns for the semi-static design and 5.638 ns for the static version, averaged over 9 random input vectors, showing that at the system level, the two designs have almost the same speed. ADMS also automatically calculates the average energy per operation, which was 84.007 pJ for the static design and 159.30 pJ for the semi-static design, averaged over 10 random input vectors, showing that the static version required almost half the amount of energy compared to the semi-static version, which contradicts what we expected, based on the power results of the multiplier components shown in Table II.

This VHDL-controlled physical-level simulation method is absolutely necessary for asynchronous circuits because the inputs do not change relative to a periodic clock pulse, but instead change value at various times based on handshaking signals. However, it is very time-consuming, requiring approximately 15 hours for each simulation, running on a 900 MHz Sun machine.

V. CONCLUSION

We designed and implemented an 8×8 bit-wise pipelined dual-rail NCL 2^s complement multiplier at all levels of abstraction, from VHDL to layout, using both static and semi-static gates. The gate-level structural VHDL model of the entire system was successfully simulated and verified to be functionally correct. Furthermore, all of the major system components were implemented, simulated, and verified at the transistor-level and physical-level. Additionally, the full system-level implementation at the physical level was completed for both the static and semi-static versions; and these system-level designs were simulated using ADMS to calculate energy per operation and average propagation delay.

From the physical-level simulation results of the various multiplier components, the semi-static implementation was found to be better in terms of area and power dissipation, whereas the static implementation was faster for larger components, but slower for the small components. The overall system-level design also showed that the semi-static version required less area than the static version, but the system-level simulations showed that the two versions had approximately the same average cycle time, and that the static version required almost half the amount of energy compared to the semi-static version, which contradicts what we expected, based on the power results of the multiplier components.

Future work includes using ADMS to calculate the energy per operation for the static and semi-static multiplier components to help investigate why the system-level semi-static design requires almost twice the energy per operation compared to the static version. This may be due to the weak feedback inverter in the semi-static gates, which required larger N-network and P-network transistors to obtain proper gate switching. The design could also be optimized to increase throughput and reduce area. This can be achieved by designing the multiplier components using the Threshold Combinational Reduction method [8] for designing optimized NCL components. Doing so will substantially reduce the number of gates required to implement each component, and will also decrease the first stage's combinational delay (i.e., from 2 gates to 1 gate), thus increasing throughput for the entire pipeline [9] (i.e., T_{DD} will decrease from 4 gates to 3 gates), since the first stage is the throughput-limiting stage.

REFERENCES

[1] 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.

[2] http://web.umr.edu/~smithsco/VHDL.html (available April 2007).

[3] Gerald E. Sobelman and Karl M. Fant, "CMOS Circuit Design of Threshold Gates with Hysteresis," *IEEE International Symposium on Circuits and Systems (II)*, pp. 61-65, 1998.

[4] Behrooz Parhami, *Computer Arithmetic Algorithms and Hardware Designs*, Oxford University Press, New York, 2000.

[5] S. C. Smith, "Development of a Large Word-Width High-Speed Asynchronous Multiply and Accumulate Unit," *Elsevier's Integration, the VLSI Journal*, Vol. 39/1, pp. 12-28, September 2005.

[6] http://web.umr.edu/~smithsco/VLSI.html (available April 2007).

[7] A. Singh and S. C. Smith, "Using a VHDL Testbench for Transistor-Level Simulation and Energy Calculation," *The 2005 International Conference on Computer Design*, pp. 115-121, June 2005.

[8] S. C. Smith, R. F. DeMara, J. S. Yuan, D. Ferguson, and D. Lamb, "Optimization of NULL Convention Self-Timed Circuits," *Elsevier's Integration, the VLSI Journal*, Vol. 37/3, pp. 135-165, August 2004.

[9] S. C. Smith, R. F. DeMara, J. S. Yuan, M. Hagedorn, and D. Ferguson, "Delay-Insensitive Gate-Level Pipelining," *Elsevier's Integration, the VLSI Journal*, Vol. 30/2, pp. 103-131, October 2001.