NCL Implementation of Dual-Rail 2^S Complement 8x8 Booth2 Multiplier using Static and Semi-Static Primitives

Mandar V. Joshi

Sagar R. Gosavi

V. Jegadeesan

A. Basu

et. al. For a complete list of authors, see https://scholarsmine.mst.edu/ele_comeng_facwork/754

NCL Implementation of Dual-Rail 2^s Complement 8×8 Booth2 Multiplier using Static and Semi-Static Primitives

M. V. Joshi, S. Gosavi, V. Jegadeesan, A. Basu, S. Jaiswal, W. K. Al-Assadi, and S. C. Smith
University of Missouri - Rolla, Department of Electrical and Computer Engineering
1870 Miner Circle, Rolla, MO 65409
Email: {mvjvx8, srggz3, vj7v7, abf6b, sj448, waleed, smithsco}@umr.edu

***Abstract*—In this work, we use static and semi-static versions of NULL Convention Logic (NCL) primitives (i.e., threshold gates) to implement a dual-rail 8×8 2^s complement multiplier using the Modified Booth2 algorithm for partial product generation and a Wallace tree for partial product summation. We establish the multiplier's functionality utilizing VHDL-based simulations of the gate-level structural design. The design is then implemented at the transistor-level and layout-level using both static and semi-static threshold gates, for a 1.8V 0.18µm TSMC CMOS process; and these two implementations are compared in terms of area, power, and speed.**

I. INTRODUCTION

The development of synchronous circuits currently dominates the semiconductor industry. However, major limiting factors to the synchronous, clocked approach include the increasing difficulty of clock distribution, increasing clock rates, decreasing feature size, increasing power consumption, timing closure effort, and difficulty with design reuse. Asynchronous, clockless circuits require less power, generate less noise, and produce less electro-magnetic interference (EMI), compared to their synchronous counterparts, without degrading performance. Furthermore, delay-insensitive asynchronous paradigms, like NULL Convention Logic (NCL) [1], have a number of additional advantages, especially when designing complex circuits, like Systems-on-a-Chip (SoCs), including substantially reduced crosstalk between analog and digital circuits, ease of integrating multi-rate circuits, and facilitation of component reuse. As demand increases for designs with higher performance, greater complexity, and decreased feature size, asynchronous paradigms will become more prevalent in the multi-billion dollar semiconductor industry, as predicted by the International Technology Roadmap for Semiconductors, which states 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 [2].

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

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 or quad-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 $1 \leq m \leq n$. 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.

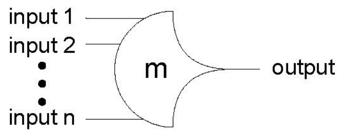

Fig. 1. THmn threshold gate.

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, $m \geq w_R > 1$, 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, …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.

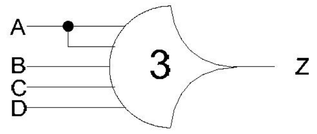

Fig. 2. TH34w2 threshold gate: Z = AB + AC + AD + BCD.

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 I. Since each rail of an NCL signal is considered a separate variable or value, a four variable function is not the same as a function of four literals, which would normally consist of eight variables or values, assuming dual-rail signals.

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 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 $\lceil \log_4 N \rceil$.

This paper concentrates on implementation of an NCL Booth2 multiplier [4] at all levels of abstraction (i.e., from VHDL to layout). Section II describes the multiplier design; Section III presents the implementation at the various levels of abstraction; Section IV includes the results and compares the static and semi-static versions in terms of power, area, and delay; and Section V concludes the paper.

II. MULTIPLIER DESIGN

One of the most widely used multiplication algorithms is Booth's algorithm, in its various forms, for both unsigned and 2^s complement numbers. Booth's algorithm reduces the number of partial products (PPs) by recoding the multiplier (MR), such that one PP is generated per multiple MR bits, instead of one PP for every MR bit, as is the case for standard binary multiplication. Since multiplication latency depends on the number of PPs, Booth's algorithm aims to speedup multiplication.

The most common implementation of Booth's algorithm is the Booth2 version, where the MR is divided into groups of 3 bits overlapping by one bit, and each group corresponds to one PP, selected based on Table II. The first group consists of the least significant two bits of MR concatenated with a logic 0, while the most significant group may require the MR to be sign extended by one bit, if the number of bits in MR is odd. Each subsequent PP is shifted 2 bit positions to the left, with respect to the previous PP. Overall, the number of PPs is reduced from *N* to $\lceil N/2 \rceil$, where *N* is the length of the multiplicand (MD).

TABLE II
PARTIAL PRODUCT SELECTION TABLE

MR_2	MR_1	MR_0	PP
0	0	0	0
0	0	1	+MD
0	1	0	+MD
0	1	1	+2MD
1	0	0	-2MD
1	0	1	-MD
1	1	0	-MD
1	1	1	-0

A Wallace tree [5], consisting of Carry Save Adders (CSAs), is then used to sum the PPs in order to reduce the number of PPs by a factor of 3/2 at each level, resulting in O(Log N) delay and O(N) area, where *N* is the number of PPs.

III. DESIGN IMPLEMENTATION

The Mentor Graphics design flow utilized in this paper is illustrated in Fig. 3.

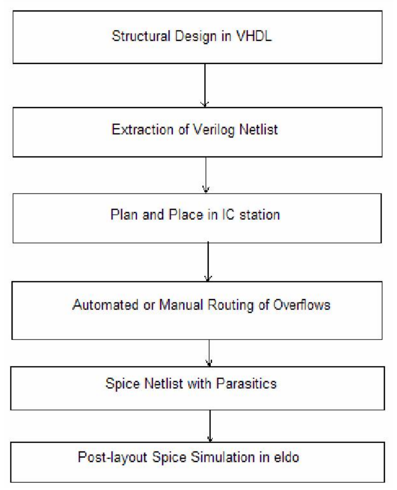
Structural Design in VHDL
Extraction of Verilog Netlist
Plan and Place in IC station
Automated or Manual Routing of Overflows
Spice Netlist with Parasitics
Post-layout Spice Simulation in eldo

Fig. 3. Mentor Graphics design flow for NCL circuits.

A. Multiplier Functional Blocks

Paper [4] details the general PP generation circuitry for the Boolean domain. In this work, we translate the Boolean PP generation circuits into NCL, as described below.

1) Decoder: This component recodes a group of 3 MR bits.
2) Block1: This component generates the LSB of each PP4.
3) Block2: This component generates all PP bits, excluding the PP LSBs, which are generated by Block1, as described above.

B. Partial Product Summation

After generation, the PPs are summed using a Wallace tree, as shown in Fig. 4, consisting of NCL full adders and half adders [6]. The multiplier's 8-bit operands produce 4 PP rows (i.e. 5 PP levels), which are summed using 3 CSAs and a Ripple Carry Adder (RCA) to produce the 16-bit product. Note that a 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; hence, a RCA is the preferred choice [7].

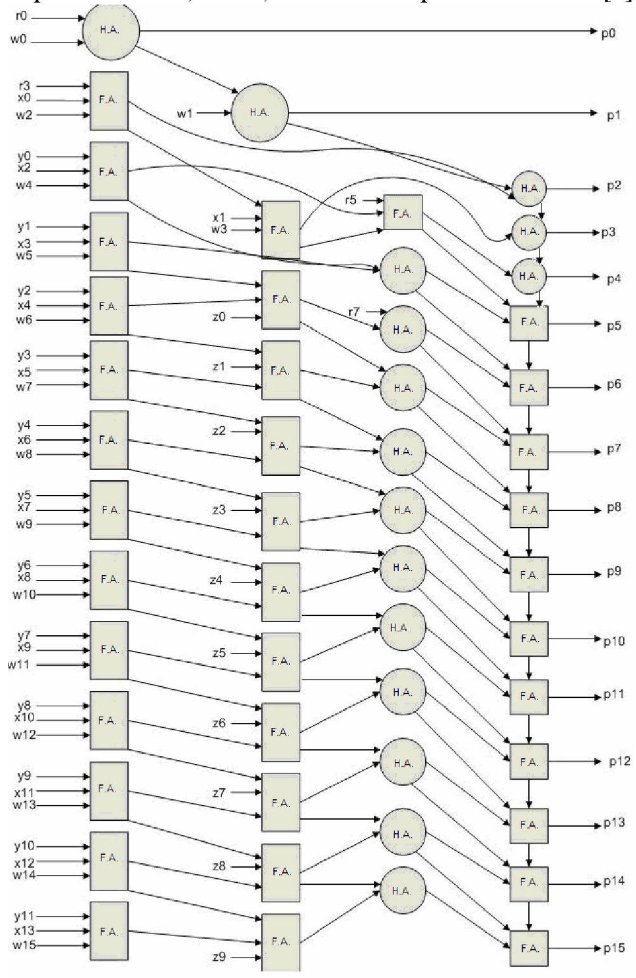

Fig. 4. PP summation: Wallace tree and RCA.

C. VHDL Implementation

The PP generation components and adders were implemented as gate-level structural designs, and were combined with generic registration and completion components [8] to form a hierarchical design of the entire 8×8 multiplier. The complete gate-level structural VHDL version of the multiplier was simulated using an exhaustive VHDL testbench, and was verified to be functionally correct.

D. Transistor-Level Implementation

As explained in Section I, 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: $Z = \text{set} + (Z^{-} \bullet \text{hold})$, 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: $Z' = \text{reset} + (Z^{-\prime} \bullet \text{set}')$, 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. 5 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.

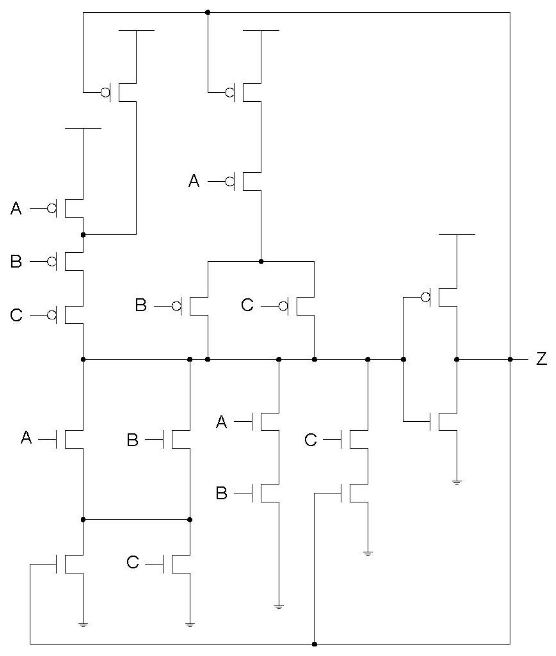

Fig. 5. Static CMOS implementation of a TH23 gate: Z = AB + AC + BC.

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. 6. 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.

Transistor-level libraries have been created for both the static and semi-static versions of all the NCL gates used in the design. 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.

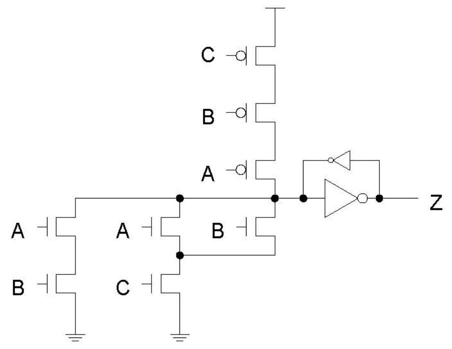

Fig. 6. Semi-static CMOS implementation of a TH23 gate: Z = AB + AC + BC.

E. Physical-Level Implementation

After the transistor-level schematic is drawn, Schematic Driven Layout (SDL) is run to generate the physical-level layout for each primitive (i.e., NCL threshold gate) that NCL "Cell Library" for both static and semi-static implementations. The layouts of the gates are created using Mentor Graphics IC Station, and are declared as Standard Cells so that they can be detected and used by IC Station's Plan, Place, and Route tool to develop the Multiplier Functional Blocks. Post-layout simulation of each gate is performed using Mentor Graphics Eldo tool, to verify the functionality of the gate with extracted parasitic capacitances. The physical-level layout for both static and semi-static versions of the TH23 gate is shown in Fig. 7.

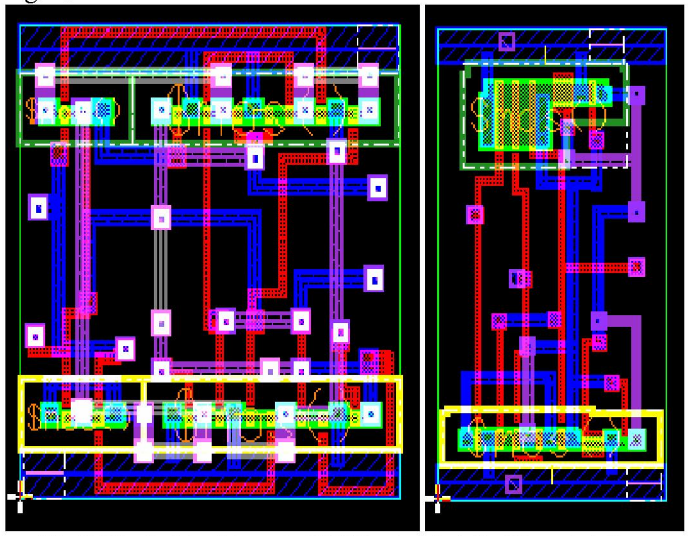

Fig. 7. Physical-level layout of TH23 gate (a) static (b) semi-static.

During the physical-level design phase, some routing problems were encountered that resulted in overflows in the design that needed to be routed manually. This was due to the complexity of the circuit at the component and system levels.

In some cases, the location of the I/O ports were changed to lessen the routing complexity.

IV. Results and Comparison

The number of components and gates utilized in the overall multiplier design are listed in Tables III and IV, respectively; and 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 in Tables V and VI, respectively.

Propagation delay is a function of number and size of transistors, and routing complexity of the gates. It can be clearly seen from Tables V and VI that semi-static components are slower than their static counterparts, because of the weak feedback inverter. However, this observation does have a few isolated exceptions, e.g., PP.rail1 in Block 1 shows a slightly greater delay for the static design. Power dissipation and area is less for the semi-static implementations, as expected, since far fewer transistors are required. 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.

TABLE III
Summary of Component Instances

Component	Occurrences
FA	16
HA	11
FA without carry	10
HA without carry	6
Decoder	13
Block 1	6
Block 2	29
Total	**91**

TABLE IV
Summary of Gate Instances

NCL Gate	Occurrences
TH12	226
TH13	59
TH14	163
TH23	325
TH33w2	313
TH22	16
TH44	476
TH33w2	25
TH34w3	19
TH54w22	29
Total	**1653**

TABLE V
Delay and Power Summary for Static Components

	Max Rise Time (ps)	MAX Fall Time (ps)	MAX T_{PHL} (ps)	MAX T_{PLH} (ps)	Power Dissipation (W)
Block I					2.819E-9
PP.rail 0	80.864	74.882	755.69	510.22	
PP.rail 1	160.22	158.71	489.33	312.43	
Block 2					1.399E-9
PP0.rail 0	89.867	83.13	1945.1	1346.1	
PP0.rail 1	127.65	91.837	1950.4	1352.8	
Decoder					5.31E-10
M.rail 0	93.996	71.72	413.47	301.62	
M.rail 1	76.66	68.421	379.06	257.38	
2M.rail 0	60.584	85.19	476.15	282.79	
2M.rail 1	81.064	67.032	514.78	334.62	
Half Adder					4.3815E-10
S.rail 0	65.929	65.55	395.01	267.81	
S.rail 1	67.591	60.79	410.05	306.45	
C.rail 0	57.331	86.496	504.5	304.97	
C.rail 1	108.54	132.16	224.59	150.79	
Full Adder					7.39749E-10
S.rail 0	91.665	128.78	682.42	411.54	
S.rail 1	93.47	134.74	665.72	383.45	
C0.rail 0	158.44	193.33	384.87	195.64	
C0.rail 1	75.329	119.27	648.66	386.86	

TABLE VI
Delay and Power Summary for Semi-Static Components

	Max Rise Time (ps)	MAX Fall Time (ps)	MAX T_{PHL} (ps)	MAX T_{PLH} (ps)	Power Dissipation (W)
Block I					2.5793E-10
PP0.rail 0	119.93	121.26	728.04	585.97	
PP0.rail 1	166.37	157.57	337.86	262.56	
Block 2					1.2025E-9
PP.rail 0	120.4	128.38	1828	1650.6	
PP.rail 1	124.85	130.86	1802.4	1626.5	
Decoder					4.9272E-10
M.rail 0	111.69	114.57	400.14	373.47	
M.rail 1	128.14	25.78	400.81	366.95	
2M.rail 0	138.93	126.26	469.25	467.04	
2M.rail 1	119.26	121.74	484.85	445.93	
Half Adder					4.0769E-10
S.rail 0	111.34	108.48	383.44	355.87	
S.rail 1	126.38	124.22	391.97	368.39	
C.rail 0	124.92	123.43	433.55	418.05	
C.rail 1	102.48	110.41	157.24	134.42	
Full Adder					6.4356E-10
S.rail 0	161.82	164.35	564.38	541.23	
S.rail 1	156.85	140.04	494.08	448.61	
C0.rail 0	148.88	153.15	248.2	137.43	
C0.rail 1	142.42	149.6	556.48	517.15	

The overall system-level layout of the multiplier has been completed, requiring 0.2894 mm^2 for the static version and 0.2550 mm^2 for the semi-static version. *ADVance MS* (ADMS), an extension of Mentor Graphics Eldo simulator, is currently being utilized to simulate the static and semi-static physical-level designs using a VHDL testbench, as described in [9]. 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. Power, energy per operation, and average propagation delay are automatically calculated using ADMS, but the system-level results have not been obtained in time for inclusion in this paper.

V. CONCLUSION

We designed and implemented a non-pipelined 2^s complement 8×8 dual-rail Booth2 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. The full system-level implementation at the physical level was completed for both the static and semi-static versions; however, the system-level power, energy per operation, and average propagation delay results were not obtained in time for inclusion in this paper.

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. The overall system-level design also showed that the semi-static version required less area than the static version.

Future work includes completing the simulation of the system-level layout, and optimizing the design to decrease area and increase throughput. Area and delay can be reduced by designing the multiplier components using the Threshold Combinational Reduction method [6] for designing optimized NCL components. Doing so will substantially decrease the number of gates required to implement each component. Furthermore, throughput can be substantially increased by pipelining the design [10].

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://www.itrs.net/Links/2005ITRS/Design2005.pdf (available March 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] M Nicolaidis, R Duarte, "Fault Secure Parity Prediction Booth Multipliers," *IEEE Design & Test of Computers*, pp. 90-101, 1999.

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

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.