NULL Convention Multiply and Accumulate Unit with Conditional Rounding, Scaling, and Saturation

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

R. F. DeMara and J. S. Yuan
School of Electrical Engineering and Computer Science
Box 162450
University of Central Florida
Orlando, FL 32816-2450
Phone: (407) 823-5916
Fax: (407) 823-5385
E-mail: demara@mail.ucf.edu

M. Hagedorn and D. Ferguson
Theseus Logic, Inc.
485 North Keller Road, Suite 140
Maitland, FL 32751
Phone: (407) 551-4697
Fax: (407) 551-4705
E-mail: dferguson@theseus.com

KEYWORDS: Asynchronous circuit design, multiply and accumulate unit, Array multiplication, Modified Baugh-Wooley Algorithm, Booth's Algorithm, gate-level pipelining, NULL Convention Logic.

Abstract

Approaches for maximizing throughput of self-timed multiply-accumulate units (MACs) are developed and assessed using the NULL Convention Logic (NCL) paradigm. In this class of self-timed circuits, the functional correctness is independent of any delays in circuit elements, through circuit construction, and independent of any wire delays, through the isochronic fork assumption [1, 2], where wire delays are assumed to be much less than gate delays. Therefore self-timed circuits provide distinct advantages for System-on-a-Chip applications.

First, a number of alternative MAC algorithms are compared and contrasted in terms of throughput and area to determine which approach will yield the maximum throughput with the least area. It was determined that two algorithms that meet these criteria well are the Modified Baugh-Wooley and Modified Booth2 algorithms. Dual-rail non-pipelined versions of these algorithms were first designed using the *Threshold Combinational Reduction (TCR)* method [3]. The non-pipelined designs were then optimized for throughput using the *Gate-Level Pipelining (GLP)* method [4]. Finally, each design was simulated using Synopsys to quantify the advantage of the dual-rail pipelined *Modified Baugh-Wooley* MAC, which yielded a speedup of 2.5 over its initial non-pipelined version. This design also required 20% fewer gates than the dual-rail

pipelined *Modified Booth2* MAC that had the same throughput. The resulting design employs a three-stage feed-forward multiply pipeline connected to a four-stage feedback multifunctional loop to perform a 72+32×32 MAC in 12.7 ns on average using a 0.25 μm CMOS process at 3.3 V, thus outperforming other delay-insensitive/self-timed MACs in the literature.

1.0 Introduction

This paper evaluates a number of both bitwise and digitwise multiplication algorithms suitable for self-timed MAC design. The bitwise algorithms include *Array* multiplication [5] and multiplication using the *Modified Baugh-Wooley* algorithm [5]. Digitwise algorithms include *Modified Booth* multiplication [5] as well as combinational *N-Bit* × *M-Bit* multiplication. These algorithms are compared in terms of throughput and area to first maximize steady-state throughput, where inputs/outputs are being supplied/consumed as fast as requested, and then minimize total gate count within the NCL multi-rail paradigm, outlined in Section 1.1. This article considers 2^s-complement operands with rounding, scaling, and saturation of the output.

1.1 Overview

Self-timed designs of multipliers and Multiply-Accumulate units (MACs) have been of recent interest in the literature [6, 7, 8]. In this paper the objective of end-to-end pipeline optimization of MACs with large word widths is addressed. Currently, one of the more mature self-timed logic design paradigms that readily supports pipelining optimizations is NULL Convention Logic (NCL) [4]. Other self-timed approaches include Seitz's method [9], Anantharaman's approach [10], Singh's method [11], David's approach [12], and Delay-Insensitive-Minterm-Synthesis (DIMS) [13]. By being self-timed, designs of these paradigms operate without global clock signals such that their functions produce outputs only when all input operands are present. Thus, this class of self-timed design implemented in CMOS technology offers reduced power consumption and noise margin compared to corresponding clocked Boolean designs. The potential for improved power characteristics has motivated recent interest in self-timed designs for applications in mobile electronics involving MACs.

The NCL self-timed design approach considered here follows the so-called "weak conditions" of Seitz's delay-insensitive signaling scheme [9] and includes an assumption that

forks in wires are isochronic [1, 2]. NCL threshold gates are a special case of the logical operators, or gates, available in digital VLSI circuit design [14]. One type of NCL threshold gate, the *THmn* gate, has N inputs, 1 output, and a threshold value, $M \leq N$. The output becomes asserted only when M or more inputs are asserted. Once the output becomes asserted, it does not become de-asserted until all inputs are de-asserted. This state-holding functionality is referred to as *hysteresis*. NCL gates are discussed in detail in [3].

A number of NCL-based designs have been commercially developed by Theseus Logic, Inc., which has formed strategic alliances with Motorola for microcontroller design and Synopsys for NCL-based design tool development. An NCL-version of the 8-bit Motorola Star-8 microcontroller was fabricated in 2000 and met or exceeded its performance targets [15]. Several other NCL-based designs have been completed with Lockheed Sanders for military applications while smartcards and other applications are currently under development.

1.2 Previous Work

Approaches to self-timed MAC design are an area of recent interest [6, 7, 8]. The *Modified Baugh-Wooley* algorithm, the *Array* algorithm, and the *Modified Booth* algorithm for multiplication are all described in [5]. The *Modified Baugh-Wooley* algorithm removes the need for negatively weighted bits present in the traditional 2^s-complement multiplication algorithm by modifying the most significant bit of each partial product and the last row of partial products, and by adding two extra bits to the partial product matrix. This allows for summation of the partial products without using special adders equipped to handle negative inputs and without increasing the height of a tree of 3-input, 2-output carry-save adders.

Array multiplication of 2^s-complement numbers also begins with each partial product bit generated according to the Modified Baugh-Wooley algorithm. Array multiplication's distinguishing characteristic is the technique for partial product summation. In the Modified Baugh-Wooley algorithm the partial products are summed using a Wallace tree [5], which reduces the number of partial products by a factor of $\frac{2}{3}$ after each level of the tree and requires $O(\log_2 N)$ time and $O(N)$ space, where N denotes the number of partial products [16]. On the other hand, Array multiplication reduces the number of partial products by one at each level, and therefore requires both $O(N)$ time and space [16].

The *Modified Booth* algorithms reduce the number of partial products to be summed by partitioning the multiplier into groups of overlapping bits, which are then used to select multiples of the multiplicand for each partial product. Consider, for example an N-bit by N-bit 2^s-complement multiply. In the Modified Booth2 algorithm the multiplier is partitioned into overlapping groups of three bits, each of which selects a partial product from the following list: +0, +M, +2M, -2M, -M, and -0, where M represents the multiplicand. This recoding reduces the number of partial products from N to $\lfloor \frac{N+2}{2} \rfloor$. The tradeoff is more logic in the recoding portion of the multiplier in exchange for fewer partial products to sum.

1.3 Paper Outline

This paper is organized into five sections. An overview of NCL is given in Section 2. In Section 3, the non-pipelined and pipelined versions of both the Modified Baugh-Wooley and Modified Booth2 MACs are developed; then their throughputs are estimated analytically and also through simulation. Section 4 then compares these designs, along with a variety of alternate designs, in terms of their gate count. Section 5 provides conclusions and compares the NCL MAC developed herein to other delay-insensitive/self-timed MACs.

2.0 Overview of NCL

2.1 Self-Timed Operation

NCL uses symbolic completeness of expression [17] to achieve self-timed behavior. A symbolically complete expression is defined as an expression that only depends on the relationships of the symbols present in the expression without a reference to the time of evaluation. Traditional Boolean logic is not symbolically complete; the output of a Boolean gate is only valid when referenced with time. For example, assume it takes 1 ns for output Z of an AND gate to become valid once its inputs X and Y have arrived. Suppose $X = 1$, $Y = 0$, and $Z = 0$, initially. If Y changes to 1, Z will change to 1 after 1 ns; therefore Z is not valid from the time Y changes until 1 ns later. Hence, output Z not only depends on the inputs X and Y, but also time must be referenced in order to determine the validity of Z. An NCL AND function on the other hand only depends on the X and Y inputs and does not reference time. When X and Y are both DATA, Z becomes DATA (logic 0 or logic 1), which is a valid output; next when both X and Y become NULL (absence of DATA), Z becomes NULL, and the next valid output will occur only

after *X* and *Y* once again become DATA, causing *Z* to transition to DATA. Therefore, the validity of the output is determined by only considering the output itself. When the output is DATA, the output is valid, and when it is NULL, the output must first transition to DATA before it is again valid. No time reference is required.

In particular, dual-rail signals, quad-rail signals, or other *Mutually Exclusive Assertion Groups* (MEAGs) can be used to incorporate data and control information into one mixed signal path to eliminate time reference [18], as described below. Time reference refers to the propagation delays of the logic components, necessary to determine the output validity in a traditional Boolean design. 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, so that both rails can never be asserted simultaneously; this state is defined as an illegal state. A quad-rail signal, *Q*, consists of four wires, Q^0, Q^1, Q^2, and Q^3, which may assume any value from the set {DATA0, DATA1, DATA2, DATA3, NULL}. The DATA0 state ($Q^0 = 1$, $Q^1 = 0$, $Q^2 = 0$, $Q^3 = 0$) corresponds to two Boolean logic signals, *X* and *Y*, where X = 0 and Y = 0. The DATA1 state ($Q^0 = 0$, $Q^1 = 1$, $Q^2 = 0$, $Q^3 = 0$) corresponds to X = 0 and Y = 1. The DATA2 state ($Q^0 = 0$, $Q^1 = 0$, $Q^2 = 1$, $Q^3 = 0$) corresponds to X = 1 and Y = 0. The DATA3 state ($Q^0 = 0$, $Q^1 = 0$, $Q^2 = 0$, $Q^3 = 1$) corresponds to X = 1 and Y = 1, and the NULL state ($Q^0 = 0$, $Q^1 = 0$, $Q^2 = 0$, $Q^3 = 0$) corresponds to the empty set meaning that the result is not yet available. The four rails of a quad-rail NCL signal are mutually exclusive, so that no two rails can ever be asserted simultaneously; these states are defined as illegal states. Both dual-rail and quad-rail signals are space optimal delay-insensitive codes, requiring two wires per bit. Other higher order MEAGs are not typically wire count optimal, however they can be more power efficient due to the decreased number of transitions per cycle. Take for example a four-bit signal. This signal can be represented as four dual-rail signals, two quad-rail signals, or one 16-rail MEAG. Since each MEAG requires one wire to transition per DATA to NULL transition and NULL to DATA transition, the total number of wire transitions required for one complete cycle, DATA to NULL to DATA, is 8 for the dual rail design, 4 for the quad-rail design, and only 2 for the 16-rail MEAG. Each wire transition requires some power dissipation, therefore higher order MEAGs are more power

efficient for data representation; however, the overall power also depends on the combinational logic, therefore the use of higher order MEAGs may or may not be more power efficient.

Logic elements are realized using 27 distinct transistor networks implementing the set of all functions of four or fewer variables [3]. Because NCL gates are designed with *hysteresis*, all asserted inputs must be de-asserted before the output will be de-asserted. Hysteresis ensures a complete transition of inputs back to the NULL state before the output associated with the next set of input data is asserted.

Inputs are partitioned into two separate wavefronts, the NULL wavefront and the DATA wavefront. The NULL wavefront consists of all inputs to a circuit being NULL, while the DATA wavefront refers to all inputs being DATA (i.e., some combination of DATA0 and DATA1 in the case of the dual-rail encoding). Initially all circuit elements are reset to the NULL state by a global reset signal to each NCL register. First, a DATA wavefront is presented to the circuit, by either another circuit or through external inputs. Once all of the outputs of the circuit transition to DATA, the NULL wavefront is presented to the circuit. Once all of the outputs of the circuit transition to NULL, the next DATA wavefront is presented to the circuit. This DATA/NULL cycle continues repeatedly, as controlled by the request and acknowledge lines, K_i and K_o, respectively. As soon as all outputs of the circuit are DATA, the circuit's result is valid and the K_o line is set to request for NULL (*rfn*), by completion detection circuitry at the output of NCL registers [4]. The NULL wavefront then transitions all of these DATA outputs back to NULL and the *Ko* line is set to request for DATA (*rfd*), by the completion detection circuitry. When the outputs transition back to DATA again, the next output is then available. This period is referred to as the DATA-to-DATA cycle time, denoted as T_{DD}, and has an analogous role to the clock period in a synchronous circuit. Figure 1 shows an NCL pipeline in a static state, such that no transitions can occur until the request input line, K_i, of the downstream register transitions to *rfd*, signifying that the NULL wavefront at the output of the downstream register has been received by the next register after the downstream register.

The *input-completeness* criterion [17], which NCL circuits must maintain in order to be self-timed, requires that:

1. the outputs of a circuit may not transition from NULL to DATA until all inputs have transitioned from NULL to DATA, and

2. the outputs of a 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 for some of the outputs to transition without having a complete input wavefront present, as long as all outputs cannot transition before all inputs arrive. There is one more condition that must be met in order for NCL to retain its self-timed nature. No *orphans* may propagate through a gate. 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, as long as they are not allowed to cross a gate boundary. This *observability* condition ensures that every gate transition is observable at the output, meaning that every gate that transitions causes at least one of the outputs to transition.

2.2 Pipelining in NCL

NCL systems consist of cascaded arrangements of three main functional blocks, *Registration*, *Completion*, and *Combinational* circuits [17]. The NCL *Registration* controls the DATA/NULL wavefronts. NCL *Completion* detects complete DATA and NULL sets, where all outputs are DATA or all outputs or NULL, respectively, at the output of every register stage. NCL *Combinational* circuits provide the desired input/output processing behavior, as detailed in [3].

An NCL registration stage is similar in concept to a clocked Boolean register and can be inserted between two combinational logic blocks in order to increase the throughput of the design. An N-bit registration stage is comprised of N single-bit NCL registers and requires N completion signals, K_o, one for each bit. The NCL *Completion* component uses these N K_o lines to detect complete DATA and NULL sets at the output of every register stage and request the next NULL and DATA sets, respectively, from either the upstream registration stage or from the circuit input. Since the maximum input threshold gate currently supported is the TH44 gate, which is the same as a 4-input C-element [19], the number of logic levels in the completion component for an N-bit register is given by $\lceil \log_4 N \rceil$.

All NCL systems have at least two register stages, one at both the input and output. Additionally, all NCL systems with feedback have at least three registration stages in the feedback loop [17]. This technique of organizing registration stages into a ring is fully discussed in [20, 13]. These registration stages interact through handshaking to prevent the next DATA

wavefront from overwriting the current DATA wavefront by ensuring that the two DATA wavefronts are always separated by a NULL wavefront. Even though these systems are self-timed, it is possible to take advantage of pipelining techniques when interconnecting NCL registration, completion, and combinational circuits [4].

3.0 MAC Designs

A block diagram for the MACs developed in this paper is shown in Figure 2. Each MAC unit performs a 32-bit by 32-bit fixed-point fractional multiply, accepting (signed × signed), (signed × unsigned), and (unsigned × unsigned) 2^s-complement operands. The product may be added to or subtracted from the 72-bit accumulator. The MAC also supports 2^s-complement and convergent rounding, up-scaling and down-scaling, output saturation, and includes a multiply only option. The output is the 72-bit 2^s-complement result along with a bit to detect overflow. The taxonomy in Figure 3 is useful to illustrate relationships between some possible multiplication algorithms applicable when designing a MAC. These include bitwise algorithms such as *Array* multiplication and the *Modified Baugh-Wooley* algorithm; and digitwise algorithms such as *Modified Booth* as well as combinational *N-Bit* × *M-Bit* multiplication. The Modified Booth algorithms [5] considered were Booth2, Booth3, and Booth4, as higher radix Booth recodings incur an excessive number of gates, as discussed in Section 4.5. The *N*-Bit × *M*-Bit algorithms considered were 2-Bit × 2-Bit, 2-Bit × 3-Bit, 2-Bit × 4-Bit, and 3-Bit × 3-Bit combinational multiplication, since larger operand implementations are not competitive in terms of gate count, as discussed in Section 4.9. For all of these algorithms both dual-rail and quad-rail encodings were assessed and compared in terms of throughput and area to determine that the dual-rail pipelined Modified Baugh-Wooley MAC achieves highest throughput with the fewest number of gates. The next best performing approach is pipelined dual-rail Modified Booth2, which was also implemented as a non-pipelined design for comparison. For each design in Section 3, the circuit operation, optimization, and performance are discussed in that order. Unless otherwise stated, designs are implemented in dual-rail logic.

3.1 Non-Pipelined Modified Baugh-Wooley MAC

3.1.1 Operation

The structure of the non-pipelined Modified Baugh-Wooley MAC is shown in Figure 4. NCL enables several optimizations as discussed in Section 3.1.2. In Phase 1, the multiplication begins by generating all of the partial products that can be generated in one gate delay. Next, these partial products are used in the first level of the Wallace tree, while the last row of partial products and most significant bit of each partial product, requiring two gate delays, are generated. Concurrently, the previous value in the accumulator is shifted, if necessary, depending on the *sign* bits, to account for the type of multiplication being performed. It is complemented if the result is to be subtracted from the accumulator, or is zeroed if multiply only is specified. Next, the modified accumulator and the uncombined partial products are used, along with the output from the first level of the Wallace tree, as the input to the second level of the Wallace tree. After this, there are six more Wallace tree levels before the partial products are reduced to two 65-bit words, where a ripple-carry addition is performed. The rationale for selecting a ripple-carry adder is detailed in Section 3.3.2.

During the summation of the partial products in Phase 1, Phase 2 begins with the multiply sign and the accumulate sign being generated as inputs to overflow detection. Also, the control signals are ensured for input-completeness in order for the MAC to remain self-timed, as described in Section 2.1. After the ripple-carry addition, the result is again shifted if necessary to account for the type of multiplication being performed and is complemented if the result is to be subtracted from the accumulator.

In Phase 3, the result can then be rounded and saturated if required. To round the result it is determined if the lower portion (LSB) is greater than or equal to 0.5, greater than 0.5, or less than 0.5. The LSB is contained in either the lower 31, 32, or 33 bits, depending on whether up-scaling, no scaling, or down-scaling is selected, respectively, as shown in Figure 5. After the LSB is compared to 0.5, a rounding bit is generated to be added to the upper portion of the result (MSB), based on the LSB and the selected rounding algorithm, either 2^s-complement or convergent rounding, described in Algorithm 3.1 and Algorithm 3.2, respectively. Next, this bit, selected from among RND31, RND32, or RND33, is added to the MSB of the result using a carry-lookahead adder. After the carry-lookahead addition, the result can then be saturated as shown in Table 1, by checking bits 71, 64, and 63. While the result is processed by the saturation

logic, the overflow bit is generated from bit 71 and the multiply and accumulate signs calculated earlier. The result is then output and fed back to the input register through an additional asynchronous register such that there are three registers in the feedback loop to prevent a lockup scenario as explained in Section 2.2.

3.1.2 Design Optimizations

There are two optimizations considered: the first is architectural and the second is NCL-specific. The first optimization deals with accumulation. The accumulator is shifted and complemented at the beginning and added to the second level of the Wallace tree, without increasing the overall delay of the Wallace tree. The result is then shifted and complemented again, following the ripple-carry addition, to reduce the critical path delay. The shifting accounts for the various multiply types: (signed × signed), (signed × unsigned), and (unsigned × unsigned), while the complementing is used for subtraction from the accumulator. The alternative is to shift and 2^s-complement the two outputs of the Wallace tree and then accumulate. This approach results in four words to be summed before the ripple-carry addition: the accumulator, the two shifted and complemented partial products, and the extra bit to be added to the least significant bit of each partial product due to their required 2^s-complementing. In the second approach, the four extra words that need to be summed before the ripple-carry addition can begin require two carry-save adders, which have a worse-case delay of two times the delay of a full adder. Therefore, this optimization will always reduce the critical path by twice the worst-case propagation delay of a full adder. In this design four gate delays were eliminated from the critical path.

Other optimizations include partial product generation facilitated through completeness optimizations in NCL, as discussed in Section 2.1. All partial products, except for the most significant bits and the last partial product, are directly generated by AND functions. To ensure completeness of the X and Y inputs only the X_iY_j partial products, where $i = j$ and $30 \geq i, j \geq 0$, require the use of *complete AND functions* [3], where both the X and Y input must be present to produce the output. The rest of the partial products, X_iY_j, where $i \neq j$, can be generated using *incomplete AND functions* [3], where the output, Z = DATA0, can be generated if either X or Y is DATA0, even if the other input is NULL. Since the incomplete AND functions require 14 fewer

transistors than the complete AND functions, and can be used for 930 of the 961 AND functions required for partial product generation, a net total of 13,020 transistors were saved in this design.

3.1.3 Average Cycle Time Determination

To determine the average cycle time for the MAC, the average cycle time for a ripple-carry adder was required. A C-language program was written that calculates the number of occurrences of each possible number of gate delays for an *N*-bit ripple-carry adder, from the minimum number of three gate delays for no carries, to the maximum number of *N*+1 gate delays for a carry occurring at each adder. The program then calculates the weighted average of the number of occurrences of each scenario to determine the expected average number of gate delays for the *N*-bit ripple-carry adder, assuming that all inputs to the ripple-carry adder are equiprobable. With $N = 65$, as in this design, the program calculates $T_{DD} = 8.33$ gate delays. With the average number of gate delays for the ripple-carry adder known, the calculation of T_{DD} follows Algorithm 3.2 in [4], as the average number of gate delays through the combinational logic for both DATA and NULL plus the number of gate delays through the completion circuitry for both DATA and NULL. Since the delay in the completion logic is 4 gates ($\lceil \log_4 73 \rceil$, where 73 is the width of the output register) and the number of gate delays through the combinational circuitry is 34, as shown in Figure 4, plus the average delay of the ripple-carry adder, determined to be 8.33 from the program, $T_{DD} = (2 \times 4) + (2 \times (34 + 8.33)) = 92.66$ gate delays, accounting for both the DATA and NULL cycle, which introduces the factor of 2. Simulation results are presented in Section 3.5. Experience with the program for a range of values of parameter *N* indicates logarithmic behavior for the ripple-carry addition as corroborated by [16].

3.2 Non-Pipelined Modified Booth2 MAC

3.2.1 Operation

The structure of the non-pipelined Modified Booth2 MAC is shown in Figure 6. In Phase 1, the multiplication begins by generating all of the partial products and the shifted and complemented, or zeroed, accumulator value, since both of these operations require three gate delays, as depicted in the figure. Next, the partial products and the modified accumulator are combined through the first of six levels of the Wallace tree. The two partial products output from the Wallace tree are used in a 67-bit ripple-carry addition. The Modified Booth2 MAC requires a

67-bit ripple-carry addition, versus the 65-bit ripple-carry addition required in the Modified Baugh-Wooley MAC, since the Modified Booth2 MAC has two less Wallace tree levels, each of which reduces the length of the ripple-carry addition by one.

During the summation of the partial products in Phase 1, Phase 2 begins with the multiply sign and the accumulate sign being generated as inputs to overflow detection. Also, the control signals and the multiplier and multiplicand, *X* and *Y*, respectively, are ensured for completeness in order to maintain delay-insensitivity. Both *X* and *Y* must be ensured here because they are not implicitly complete in the partial product generation circuitry, as they are in the Modified Baugh-Wooley design, ensured by selectively complete AND functions. After the ripple-carry addition, the result is again shifted, if necessary, to account for the type of multiplication being performed and is complemented if the result is to be subtracted from the accumulator.

In Phase 3, the result can then be rounded and saturated if required and the overflow bit generated in exactly the same manner as for the Modified Baugh-Wooley MAC. The result is then output and fed back to the input register through an additional asynchronous register such that the required three registers are present in the feedback loop.

3.2.2 Design Optimizations

Optimizations for selecting multiplication type and adding/subtracting the partial products to/from the accumulator, applicable to the Modified Baugh-Wooley design and here, were implemented, as described in Section 3.1.2.

3.2.3 Average Cycle Time Determination

T_{DD} can be calculated in the same way as described in Section 3.1.3. Since the delay in the completion logic is 4 gates ($\lceil \log_4 73 \rceil$, where 73 is the width of the output register) and the number of gate delays through the combinational circuitry is 32, as depicted on the left-hand side of Figure 6, plus the average of the 67-bit ripple-carry adder, determined to be 8.38 from the C-language program, $T_{DD} = (2 \times 4) + (2 \times (32 + 8.38)) = 88.76$ gate delays, accounting for both the DATA and NULL cycle. Subsequent simulations also indicate the Modified Booth2 design outperforms the Modified Baugh-Wooley design for the non-pipelined configuration only.

3.3 Pipelined Modified Baugh-Wooley MAC

3.3.1 Operation

The structure of the pipelined Modified Baugh-Wooley MAC is shown in Figure 7. The first stage begins by generating all of the partial products that can be generated in one gate delay. Next, these partial products are used in the first level of the Wallace tree, while the remaining partial products that require two gate delays are generated. The remaining partial products, along with the output from the first level of the Wallace tree, are then used as the input to the second level of the Wallace tree. Stage 1 also contains the third level of the Wallace tree along with the multiply sign generation. The second stage consists of four more levels of the Wallace tree. Stage 3 begins with the final level of the Wallace tree, followed by the shifting and 2^s-complementing of the Wallace tree output, if necessary, to account for the type of multiplication being performed and for subtraction from the accumulator. The third stage also contains another carry-save adder, required because of the 2^s-complement operation. Stage 4 begins the feedback loop and contains the circuitry to zero *Ain* for the multiply only function and the final carry-save adder to add *Ain* to the Wallace tree output. The fourth stage also generates the accumulate sign. The fifth stage consists solely of a 71-bit ripple-carry adder. Stage 6 contains the first part of the rounding logic, while Stage 7 contains the remaining rounding logic along with the saturation circuitry, control signal completeness logic, and overflow detection circuitry, as explained in Section 3.1.1.

3.3.2 Throughput Maximization

An effective approach for pipelining a self-timed MAC begins with minimization of the feedback loop. This is in part because the feed-forward portion of the MAC can be pipelined to a fine granularity as long as completeness is ensured at each stage boundary. This enables the throughput of the feed-forward path to be at least equal to that of the feedback loop, if not greater. To do this, it is preferable to postpone the addition of *Ain* with the partial products until absolutely necessary. Moreover, the subtraction and multiply mode selection method can be revised such that it reduces the number of operations required in the feedback loop. To increase throughput in the non-pipelined design, *Ain* was complemented and shifted, or zeroed, and the result from the ripple-carry adder was complemented and shifted. However, for the pipelined design, the two outputs of the Wallace tree can be 2^s-complemented and shifted, allowing the

shifting and complementing of *Ain* followed by the shifting and complementing of the result to be removed from the feedback loop. This is replaced instead by the 2^s-complementing and shifting of the final two partial products, followed by an extra carry-save adder in the feed-forward portion of the design. The zeroing of *Ain* for the multiply only function is still required to be performed within the feedback loop. In the pipelined implementation, this change eliminates five gate delays from the feedback path with no additional latency in the pipeline. The corresponding logic is relocated to the feed-forward portion of the design. Partitioning the feed-forward portion into three stages with a maximum of 8 gate delays per stage allows the inclusion of the additional logic without decreasing overall throughput.

After the feedback logic of the MAC is minimized, it can be pipelined by inserting asynchronous registers [4]. It was shown in [21] that a feedback loop containing N tokens requires $2N$ bubbles for maximum throughput. A token is defined as a DATA wavefront with corresponding NULL wavefront; and a bubble is defined as either a DATA or NULL wavefront that occupies more than one neighboring stage (i.e. for the case of two adjacent DATA stages, $stage_i$ is the DATA wavefront and $stage_{i-1}$ is a bubble). This allows for each DATA and NULL wavefront to move through the feedback loop independently. Since the feedback loop in the MAC design only contains one token, two bubbles are necessary to maximize throughput. A token requires two stages, one stage for the DATA portion and one stage for the NULL portion, while each bubble requires one stage. Therefore, the feedback loop was partitioned into four stages for maximum throughput.

The front end of the feedback loop was partitioned as shown in Figure 7. Partitioning of the ripple-carry adder is not advisable since this would incur extra gate delays on the critical path. Inserting a register in the middle of the ripple-carry addition would tend to lessen the benefits of its asynchronous behavior by increasing the $O(\log_2 N)$ average time for an N-bit ripple-carry addition, since $\log_2 N1 + \log_2 N2 > \log_2 N$, where $N = N1 + N2$, $N \geq 6$, and $N1, N2 \geq 3$. The last two stages were divided to minimize the worst-case delay of each stage. The Upper Rounding logic for the most significant 41 bits of the result can be partitioned into a 5 gate delay circuit followed by a 1 gate delay circuit, without violating the input-completeness criteria. Alternately, inserting a register between this partition would result in Stage 6 having 10 gate delays and Stage 7 having 4 gate delays. The 10 gate delays of Stage 6 in this alternate

design would exceed the 9 gate delays of Stage 7 in the current design. Furthermore, simulation shows both finer and coarser partitionings decrease throughput.

Throughput can be further increased using partial bitwise completion [4] where the feed-forward output joins the feedback input. Two separate completion logic blocks are appropriate. The first, whose input is Ko_1, only acknowledges the inputs from the feed-forward circuit; the second, whose input is Ko_2, only acknowledges the feedback inputs. This optimization can decrease the inter-dependencies between the feedback loop and the feed-forward path to boost throughput an additional 2%.

Finally, the feed-forward portion is pipelined such that its throughput is at least as great as that of the feedback loop. In other words, the output from the feed-forward portion of the design must always be available when the feedback input is ready. Therefore, the minimum forward path through the feedback loop must be determined. Since the minimum delay through a ripple-carry adder is 3 gates and the delay for each register is 1 gate, the minimum forward path through the feedback loop is $3 + 3 + 5 + 9 + (5 \times 1) = 25$ gate delays, as calculated by the delays in the logic components of the feedback path in Figure 7, and indicated on the right side of the figure. In order to ensure that the feedback loop will never wait on input from the feed-forward portion, the maximum cycle time of the feed-forward pipeline must not exceed 25 gate delays. Decreasing the cycle time of the feed-forward portion to less than 25 gate delays will not increase the throughput as a whole. Therefore, this MAC optimization problem is transformed to ensuring a maximum cycle time of 25 gate delays for the feed-forward portion of the design, while adding as few asynchronous registers as possible. Following the method described in [4] for pipelining NCL circuits, it was determined that the addition of two asynchronous registers, as shown in Figure 7, would result in a maximum cycle time of 24 gate delays for the feed-forward circuitry. Furthermore, our simulations showed that finer partitioning does not increase throughput, while coarser partitioning decreases throughput.

3.4 Pipelined Modified Booth2 MAC

3.4.1 Operation

The structure of the pipelined Modified Booth2 MAC is shown in Figure 8. The first stage begins by generating all of the partial products, which are then input to the first of two levels of the Wallace tree. Stage 1 also contains the multiply sign generation and the

completeness generation for the multiplier and multiplicand, *X* and *Y*, respectively, since they are not implicitly complete in the partial product generation circuitry. The second stage consists of three more levels of the Wallace tree. Stage 3 begins with the final level of the Wallace tree, followed by the shifting and 2^s-complementing of the Wallace tree output, if necessary, to account for the type of multiplication being performed and for subtraction from the accumulator. The third stage also contains another carry-save adder, required because of the 2^s-complement operation. Stage 4 begins the feedback loop and contains the circuitry to zero *Ain* for the multiply only function and the final carry-save adder to add *Ain* to the Wallace tree output. The fourth stage also generates the accumulate sign. The fifth stage consists solely of a 71-bit ripple-carry adder. Stage 6 contains the first part of the rounding logic, while Stage 7 contains the remaining rounding logic along with the saturation circuitry, control signal completeness logic, and overflow detection circuitry, as detailed in Section 3.1.1.

3.4.2 Throughput Maximization

The throughput maximization procedure for the feedback loop follows that of the pipelined Modified Baugh-Wooley design, explained in Section 3.3.2. The minimum forward path through the feedback loop is also 25 gate delays, and is independent of the selected multiplication algorithm. Addition of as few as two asynchronous registers, as shown in Figure 8, results in a maximum cycle time of 24 gate delays for the feed-forward portion. Since the feedback loop for the pipelined Modified Booth2 and Baugh-Wooley designs are the same, and the feedback loop is the limiting factor of throughput maximization for each, the two designs should have the same throughput.

3.5 Simulation Results

Representative MAC operations need to be selected to provide a basis for comparison of throughputs. A candidate operation is $Aout = \sum_{i=0}^{N}(X_i \times Y_i)$, where $X_i = X_0 + (2^{-21} \times i)$ and $Y_i = Y_0 + (2^{-11} \times i)$ with *N* chosen to be 255. This allows a variety of computations to be performed such that any unusually short or long operations will not significantly skew the average cycle time. For instance, in our testbench X_0 and Y_0 were randomly selected such that X_0 = A61C039Dh = -0.702270077076 and Y_0 = F0046718h = -0.124865639955. Also, (signed × signed) multiplication was selected and rounding, scaling, and saturation were

disabled. The same operation was also performed in a C-language program and the result from this program agreed with the results from each of the simulated designs:
Aout = 05A0B13C0E04A37000h = 11.2554087704.

Both the non-pipelined and pipelined Modified Baugh-Wooley and Booth2 MAC designs were simulated using Synopsys in order to compare their throughputs to ensure that the relative values were consistent with the predicted results. The Synopsys technology library for the NCL gates is based on static 3.3 V, 0.25 μm CMOS implementations. The average cycle time, T_{DD}, for the non-pipelined Modified Baugh-Wooley MAC was determined to be 31.8 ns; while T_{DD} for the non-pipelined Modified Booth2 MAC was determined to be 31.2 ns. Therefore, the non-pipelined Modified Booth2 MAC is faster than the non-pipelined Modified Baugh-Wooley MAC, as anticipated in Section 3.2.3. As for the pipelined designs, the Modified Baugh-Wooley and Booth2 MACs were anticipated to run at the same speed due to the fact that the feedback path was the same in both designs. The simulations of the two pipelined designs indicate that they both have an average cycle time of 12.7 ns.

4.0 Gate Requirements for the Proposed Designs

In Section 3.3.2 and Section 3.4.2 it was shown that the throughput of a pipelined self-timed MAC design is limited by the feedback loop, independent of the feed-forward portion. This is due to the fact that the feed-forward portion can be readily pipelined to a fine granularity to match or exceed the throughput of the feedback loop. Since the feedback loop performs accumulation independent of the selected multiplication algorithm, the throughput of the MAC as a whole is independent of the multiplication algorithm. This is demonstrated by the pipelined versions of the Modified Baugh-Wooley and Booth2 MACs operating with the same cycle time.

The design objective stated in the abstract is to obtain the highest throughput MAC using the fewest gates. Since the throughput of the pipelined MAC does not depend on the multiplication algorithm, the MAC throughput optimization problem can be transformed into the selection of the multiplication algorithm that requires the least amount of area to implement. The following sections will compare various algorithms to determine which requires the least gate count.

4.1 Modified Baugh-Wooley MAC

Since both the non-pipelined and pipelined designs were implemented in VHDL, the actual number of gates can be tabulated. The non-pipelined design requires 10,703 gates, while the pipelined design uses 13,613 gates, as shown in Figure 3. For both of these designs approximately 2,048 gates were from partial product generation with 32 complete AND functions and 961 incomplete AND functions.

4.2 Modified Booth2 MAC

Since both the non-pipelined and pipelined versions of this design were also implemented in VHDL, the actual number of gates can again be tabulated. The non-pipelined design used 14,101 gates, while the pipelined design used 17,015 gates, as shown in Figure 3. For both of these designs approximately 7,854 gates were from the partial product generation. Even though the Booth2 recoding eliminates two levels in the Wallace tree, the additional gates required in the partial product generation outpace the savings. This causes the pipelined Modified Booth2 design to contain 3,402 more gates than the pipelined Modified Baugh-Wooley design. The Modified Booth2 MAC requires 405 fewer adders, which is 1,620 fewer gates, since each adder contains four gates. However, it requires approximately 5,806 additional gates for partial product generation. Since both designs operate with the same cycle time, the preferred design is the pipelined Modified Baugh-Wooley MAC, since it requires less area. This is even more evident when the number of transistors for partial product generation is compared. Since the number of transistors for the Modified Baugh-Wooley partial product generation can be greatly reduced as explained in Section 3.1.2, even though the number of gates remain the same, the transistor requirement for partial product generation of the two designs magnifies this differential, as shown in Figure 3. The partial product generation for the Modified Booth2 design requires 3.8-fold more gates than for the Modified Baugh-Wooley design, but 6.8-fold more transistors, due to the more sophisticated gates required in the recoding logic.

4.3 Array MAC

Both the Array MAC and the Modified Baugh-Wooley MAC use the same logic to generate the partial products and both require O(N) area for the partial product summation, as explained in Section 1.1. However, the Modified Baugh-Wooley MAC only requires O($\log_2 N$)

gate delays for the partial product summation, while the Array MAC requires O(N) gate delays. Therefore, many more asynchronous registers would be required to partition the feed-forward circuitry of the Array MAC than the two required for the Modified Baugh-Wooley MAC, in order to achieve the same throughput. Hence, the Array MAC would require approximately the same number of adders as the Modified Baugh-Wooley MAC, but would require many more asynchronous registers, causing it to contain many more gates than the Modified Baugh-Wooley MAC. However, the structure of the Array MAC is very regular compared to the irregular structure of the Modified Baugh-Wooley MAC, which could make it more desirable when layout is taken into consideration, despite its larger size.

4.4 Modified Booth3 MAC

The Modified Booth3 multiplication algorithm partitions the multiplier into overlapping groups of four bits, each of which selects a partial product from the following list: +0, +M, +2M, +3M, +4M, -4M, -3M, -2M, -M, and -0, where M represents the multiplicand. For the 32-bit × 32-bit multiplication, this decoding theoretically reduces the number of partial products from 17 for the Modified Booth2 algorithm to only 11. However, the +3M and -3M partial products cannot be obtained by simple shifting and/or complementing, like the others. These partial products are referred to as hard multiples. Therefore, two actual partial products must be used to represent each theoretical partial product to avoid the ripple-carry addition that would be required to compute both the +3M and -3M partial products. Any +3M partial product is represented by a +2M and a +M partial product, while any -3M partial product is represented by a -2M and a -M partial product. Since each theoretical partial product must be represented by two partial products, the actual number of partial products for the Modified Booth3 MAC is 22, and the number of Wallace tree levels required to sum these partial products is 7. This is more than the 17 partial products required for the Modified Booth2 design, which can be summed using only 6 Wallace tree levels. Therefore, a Modified Booth3 MAC requires more adders to sum the partial products than would the Modified Booth2 MAC. Furthermore, the partial product generation requires scanning four multiplier bits at a time for the Modified Booth3 algorithm, versus only three bits which are simultaneously scanned in the Modified Booth2 algorithm. This requires more complex recoding logic for the Modified Booth3 algorithm. Since the Booth3

algorithm requires more adders and more recoding logic than the Booth2 algorithm, and increases the depth of the Wallace tree, it requires more gates than the Modified Booth2 design.

4.5 Modified Booth4 MAC

The Modified Booth4 multiplication algorithm also suffers from the problem of hard multiples. It partitions the multiplier into overlapping groups of five bits, each of which selects a partial product from the following list: +0, +M, +2M, +3M, +4M, +5M, +6M, +7M, +8M, -8M, -7M, -6M, -5M, -4M, -3M, -2M, -M, and -0, where M represents the multiplicand. The hard multiples are +3M, +5M, +6M, +7M, -7M, -6M, -5M, and -3M. However, if the hard multiples were to be generated through ripple-carry addition, the +6M and -6M multiples could be obtained simply by shifting the +3M and -3M multiples, respectively. For the 32-bit × 32-bit multiplication, this decoding theoretically reduces the number of partial products from 17 for the Modified Booth2 algorithm to only 9. However, since the hard multiples require two partial products to represent each theoretical partial product, the actual number of partial products required is 17. The most significant partial product cannot be a hard multiple and therefore only requires one partial product for its representation. The actual number of partial products for the Modified Booth4 MAC is the same as for the Modified Booth2 MAC. The only difference is the partial product generation, which requires scanning five multiplier bits at a time for the Modified Booth4 algorithm, versus only three bits that are simultaneously scanned in the Modified Booth2 algorithm. This requires more complex recoding logic for the Modified Booth4 algorithm. Therefore, the Modified Booth4 MAC requires more gates than the Modified Booth2 MAC. Furthermore, higher radix Modified Booth algorithms can be expected to exhibit similar characteristics.

4.6 Combinational 2-Bit × 2-Bit MAC

The 2-Bit × 2-Bit partial product generation partitions both the multiplier and multiplicand into 16 groups of two bits that do not overlap. Each 2-bit multiplier, 2-bit multiplicand pair generates 4 bits of partial product. Every 2-bit multiplier group generates two rows of partial products since each 2-bit multiplier, 2-bit multiplicand pair generates 4 bits and each consecutive group of 4 bits is shifted two places due to the 2-bit partitioning of the multiplicand. This results in consecutive groups of 4 bits generated from one 2-bit multiplier

group to be overlapped by two bits. Since there are sixteen 2-bit multiplier groups and each group generates two partial products, there are a total of 32 partial products. Since this number of partial products is the same as for the Modified Baugh-Wooley design, both designs will require the same amount of gates to sum the partial products. Therefore, the only difference between the two designs is the partial product generation. The 2-Bit × 2-Bit partial product generation requires approximately 2,816 gates, while the Modified Baugh-Wooley partial product generation only requires approximately 2,048 gates, as shown in Figure 3. Hence, the 2-Bit × 2-Bit algorithm requires approximately 768 more gates than does the Modified Baugh-Wooley algorithm, making it less area efficient. This is even more evident when the transistor count for the partial product generation is compared. The Modified Baugh-Wooley partial product generation requires approximately 18,880 transistors, while the 2-Bit × 2-Bit partial product generation requires approximately 38,400 transistors, more than twice as many.

4.7 Combinational 2-Bit × 3-Bit MAC

The 2-Bit × 3-Bit partial product generation partitions the multiplier into 16 groups of two bits, and the multiplicand into 10 groups of three bits with 1 group of two bits, such that no groups overlap. Each 2-bit multiplier, 3-bit multiplicand pair generates 5 bits of partial product. Every 2-bit multiplier group generates two rows of partial products since each 2-bit multiplier, 3-bit multiplicand pair generates 5 bits and each consecutive group of 5 bits is shifted three places due to the 3-bit partitioning of the multiplicand. All two-row partial products generated from one 2-bit multiplier group contain an unused slot every third bit position, such that every third bit position in a two-row partial product only contains one bit rather than two bits, as in the other bit positions. Since there are sixteen 2-bit multiplier groups and each group generates two partial products, 32 partial products are anticipated. However, because of the unused slots, there are actually only 26 rows of partial products, which can be summed in 7 Wallace tree levels. The multiplier could also be partitioned into 10 groups of three bits with 1 group of two bits, while the multiplicand was partitioned into 16 groups of two bits, such that no groups overlap. This alternate partitioning also produces 26 rows of partial products. Recall that the Booth2 design, which has 17 rows of partial products that can be summed in 6 levels of Wallace tree, saved 405 adders or 1,620 gates in the partial product summation, as discussed in Section 4.2. Since the 2-Bit × 3-Bit algorithm requires 26 rows of partial products, which can be summed in 7 Wallace

tree levels, this algorithm cannot utilize fewer adders than the Booth2 algorithm. Therefore, the number of gates saved by the reduced Wallace tree of the 2-Bit × 3-Bit algorithm is no more than 1,620. The number of gates required to generate the partial products for the 2-Bit × 3-Bit algorithm is approximately 4,768, a difference of approximately 2,720 additional gates than for the Modified Baugh-Wooley partial product generation. Therefore, the 2-Bit × 3-Bit algorithm would require at least 1,100 more gates than the Modified Baugh-Wooley design since it can save no more than 1,620 gates in the Wallace tree and requires an additional 2,720 gates for partial product generation.

4.8 Combinational 2-Bit × 4-Bit MAC

The 2-Bit × 4-Bit partial product generation partitions the multiplier into 16 groups of two bits, and the multiplicand into 8 groups of four bits, such that no groups overlap. Each 2-bit multiplier, 4-bit multiplicand pair generates 6 bits of partial product. Every 2-bit multiplier group generates two rows of partial products since each 2-bit multiplier, 4-bit multiplicand pair generates 6 bits and each consecutive group of 6 bits is shifted four places due to the 4-bit partitioning of the multiplicand. All two-row partial products generated from one 2-bit multiplier group contain two unused slots every fourth bit position, such that for every four bit positions in a two-row partial product only two contain two bits while the other two contain only one bit. Since there are sixteen 2-bit multiplier groups and each group generates 2 partial products, 32 partial products are anticipated. However, because of the unused slots, there are actually only 23 rows of partial products, which can be summed in 7 Wallace tree levels. The multiplier and multiplicand could also be partitioned vise-versa, resulting in the same number of partial product rows. Since this design also requires 7 Wallace tree levels, as did the 2-Bit × 3-Bit design, it could not possibly save more than 1,620 gates in the Wallace tree, as explained in Section 4.7. The partial product generation is also more complicated than for the 2-Bit × 3-Bit partial product generation since more inputs are required. Therefore, partial product generation for this design requires at least as many gates as for the 2-Bit × 3-Bit design. Hence, this design must require more gates than the Modified Baugh-Wooley MAC, following the logic of Section 4.7.

4.9 Combinational 3-Bit × 3-Bit MAC

The 3-Bit × 3-Bit partial product generation partitions both the multiplier and multiplicand into 10 groups of three bits, with one group of two bits, such that no groups overlap. Each 3-bit multiplier, 3-bit multiplicand pair generates 6 bits of partial product. Every 3-bit multiplier group generates two rows of partial products since each 3-bit multiplier, 3-bit multiplicand pair generates 6 bits and each consecutive group of 6 bits is shifted three places due to the 3-bit partitioning of the multiplicand, such that all consecutive groups of 6 bits generated from one 3-bit multiplier group overlap by three bits. The last multiplier group is only two bits, so for each 2-bit multiplier, 3-bit multiplicand pair, 5 bits of partial product are generated. This 2-bit multiplier group generates two rows of partial products since each 2-bit multiplier, 3-bit multiplicand pair generates 5 bits and each consecutive group of 5 bits is shifted three places due to the 3-bit partitioning of the multiplicand. These last two rows of partial products contain an unused slot every third bit position, such that every third bit position in the last two-row partial product only contains one bit rather than two bits, as in the other bit positions. Since there are 10 3-bit multiplier groups and one 2-bit multiplier group, each of which generates 2 partial products, 22 partial products are anticipated. However, because of the unused slots generated by the 2-bit multiplier group, there are actually only 21 rows of partial products, which can be summed in 7 Wallace tree levels. Since this design also requires 7 Wallace tree levels, as did the 2-Bit × 3-Bit design, it could not possibly save more than 1,620 gates in the Wallace tree, as explained in Section 4.7. The partial product generation is also more complicated than for the 2-Bit × 3-Bit partial product generation since more inputs are required. Therefore, partial product generation for this design requires at least as many gates as for the 2-Bit × 3-Bit design. Hence, this design must require more gates than the Modified Baugh-Wooley MAC, following the logic of Section 4.7. Furthermore, any larger sized *N*-Bit × *M*-Bit algorithms would not be likely to reduce the number of gates due to their increasing complexity.

4.10 Quad-Rail MACs

To test the feasibility of quad-rail multiplication, a quad-rail 4-bit × 4-bit unsigned multiplier was designed, implemented, and tested. The resulting design operated with the same throughput as its dual-rail counterpart but required slightly more than twice as many gates, showing that a quad-rail encoding is not as efficient for realizing multiplication. Furthermore,

quad-rail partial product generation circuitry was designed for each of the algorithm types shown in Figure 3; and the resulting quad-rail designs required at least 2% more gates and 10% more transistors than their dual-rail counterparts.

5.0 Conclusion

5.1 Overview of the NCL MAC Design Process

In Section 3 it was shown how to design and then pipeline both a self-timed Modified Baugh-Wooley MAC and a Modified Booth2 MAC in order to achieve maximum throughput. Throughput maximization was accomplished by first minimizing the feedback loop and then partitioning the feed-forward path such that its throughput was at least equal to that of the feedback loop; since the feedback loop was determined to be the factor that limits increasing throughput. Section 3 also showed that the feedback loop did not depend on the chosen multiplication algorithm, and therefore the throughput also did not depend on the multiplication algorithm, although a faster multiplication algorithm would decrease latency of an isolated multiply. This was substantiated through simulations of both the pipelined Modified Baugh-Wooley MAC and the pipelined Modified Booth2 MAC, which both had the same throughput.

Since it was shown that the throughput of the MAC did not depend on the multiplication algorithm, the self-timed MAC throughput optimization problem was transformed into selecting the multiplication algorithm requiring the fewest gates. Section 4 compared the area of multiple MAC designs using various multiplication algorithms. The best design is therefore the one that requires the fewest gates to implement. It was also shown in Section 4 that the pipelined Modified Baugh-Wooley design required the least area, and was therefore the best design based on the criteria of the highest throughput with the least area. The dual-rail pipelined Modified Baugh-Wooley MAC yielded a speedup of 2.5 over its initial non-pipelined version and required 20% fewer gates than the dual-rail pipelined Modified Booth2 MAC, which had the same throughput.

5.2 Comparison with Related Work

Table 2 compares this optimized NCL MAC to other delay-insensitive/self-timed MACs in the literature, showing that the 3.3 V, 0.25 μm CMOS NCL MAC outperforms the other designs. A serial-parallel MAC using the methods and tools developed at Caltech [22] for design

of delay-insensitive circuits is described in [6]. Using the Caltech approach an 8+4×4 MAC was fabricated with 5 V, 2 μm CMOS technology that operated at 37 ns; an extrapolation to larger word sizes was presented in [6]. Using this extrapolation it was determined that a 64+32×32 MAC would operate at 901 ns, much slower than the NCL MAC, as expected, since the implemented algorithm is not fully parallel. A self-timed 16+8×8 MAC designed using SCCVSL (single-rail CMOS cascode voltage switch logic) and fabricated in 0.6 μm technology is described in [7]. This MAC employs the parallel Booth2 algorithm, and has an average cycle time of about 90 ns. A third self-timed MAC described in [8] was designed in single-ended dynamic logic [23], utilizing conditional evaluation along with the traditional Array multiplication algorithm. Conditional evaluation allows for rows with a zero bit product to be multiplexed around, to reduce energy and delay. In [8] a 16+8×8 MAC was simulated using 3.3 V, 0.35 μm CMOS technology, to determine the average cycle time of 7.8 ns. This delay information was then used in [8] to estimate the average cycle time for a 32+16×16 MAC as approximately 24 ns. These comparisons indicate that the NCL-based dual-rail pipelined Modified Baugh-Wooley MAC developed herein outperforms the three above-mentioned methods, even after technology adjustments.

Furthermore, the NCL MAC supports conditional rounding, scaling, and saturation, whereas the other MACs discussed herein do not. Without the conditional rounding, scaling, and saturation the NCL MAC's performance could be substantially increased, since the feedback path, which limits the MAC's throughput, is dominated by this logic. Removing the conditional rounding, scaling, and saturation logic would result in a feedback path consisting only of the Zero Ain function, the CSA, the Calculate Accumulate Sign function, the 71-bit RCA, the Control Signal Completeness function, and the Overflow calculation, along with the five asynchronous registers to repartition the feedback path of Figure 7 as follows: Stage 4 and Stage 5 remain unchanged with a minimum of 3 gate delays each, Stage 6 now consists of the Control Signal Completeness function that has two gate delays, and Stage 7 consists solely of the Overflow calculation that is three gate delays. Therefore the minimum forward path of the feedback loop is reduced to: $3 + 3 + 2 + 3 + (5 \times 1) = 16$ gate delays. The feed-forward pipeline could then be repartitioned such that its maximum cycle time does not exceed 16 gate delays, as explained in Section 3.3.2. Hence, the throughput of the MAC without conditional rounding, scaling, and saturation is limited by the feedback path of 16 gate delays, versus the feedback path

of 25 gate delays for the MAC with conditional rounding, scaling, and saturation. Removal of the conditional rounding, scaling, and saturation logic provides a potential means to eliminate further gate delays in the feed-forward path.

References

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

[2] K. Van Berkel, "Beware the Isochronic Fork," *Integration, The VLSI Journal,* Vol. 13, No. 2, pp. 103-128, 1992.

[3] S. C. Smith, *Gate and Throughput Optimizations for NULL Convention Self-Timed Digital Circuits*, Ph.D. Dissertation, School of Electrical Engineering and Computer Science, University of Central Florida, 2001.

[4] 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.

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

[6] C. D. Nielsen and A.J. Martin, "Design of a Delay-Insensitive Multiply and Accumulate Unit," *Twenty-Sixth Hawaii International Conference on System Sciences*, Vol. 1, pp. 379-388, 1993.

[7] T. Tang, C. Choy, P. Siu, and C. Chan, "Design of Self-Timed Asynchronous Booth's Multiplier," *Asian South-Pacific Design Automation Conference*, pp. 15-16, 2000.

[8] V. A. Bartlett and E. Grass, "A Low-Power Concurrent Multiplier-Accumulator Using Conditional Evaluation," *6th IEEE International Conference on Proceedings of ICECS*, Vol. 2, pp. 629-633, 1999.

[9] C. L. Seitz, "System Timing," in *Introduction to VLSI Systems*, Addison-Wesley, pp. 218-262, 1980.

[10] T. S. Anantharaman, "A Delay Insensitive Regular Expression Recognizer," *IEEE VLSI Technology Bulletin*, Sept. 1986.

[11] N. P. Singh, *A Design Methodology for Self-Timed Systems*, Master's Thesis, MIT/LCS/TR-258, Laboratory for Computer Science, MIT, 1981.

[12] 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.

[13] J. Sparso, J. Staunstrup, M. Dantzer-Sorensen, "Design of Delay Insensitive Circuits using Multi-Ring Structures," *European Design Automation Conference*, pp. 15-20, 1992.

[14] A. J. Martin, "The Limitations to Delay-lnsensitivity in Asynchronous Circuits," *Advanced Research in VLSI: Sixth MIT Conference*, pp. 263-278, 1990.

[15] John McCardle and David Chester, "Measuring an Asynchronous Processor's Power and Noise," Synopsys User Group Conference (SNUG), Boston, 2001.

[16] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, McGraw-Hill Book Company, New York, 1995.

[17] Karl M. Fant and Scott 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.

[18] T. Verhoff, "Delay-Insensitive Codes–An Overview," *Distributed Computing,* Vol. 3, pp. 1-8, 1988.

[19] D. E. Muller, "Asynchronous Logics and Application to Information Processing," in *Switching Theory in Space Technology*, Stanford University Press, pp. 289-297, 1963.

[20] T. E. Williams, *Self-Timed Rings and Their Application to Division*, Ph.D. Thesis, CSL-TR-91-482, Department of Electrical Engineering and Computer Science, Stanford University, 1991.

[21] Jens Sparso and Jorgen 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.

[22] A. J. Martin, "Synthesis of Asynchronous VLSI Circuits," *Formal Methods for VLSI Design*, pp. 237-283, 1990.

[23] G. E. Sobelman and D. Raatz, "Low-power Multiplier Design using Delayed Evaluation," *International Symposium on Circuits and Systems*, pp. 1564-1567, 1995.

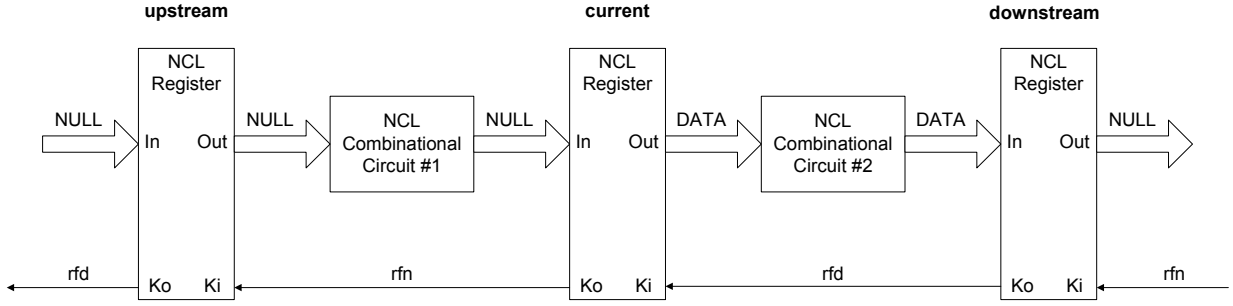

Figure 1: NCL Pipeline.

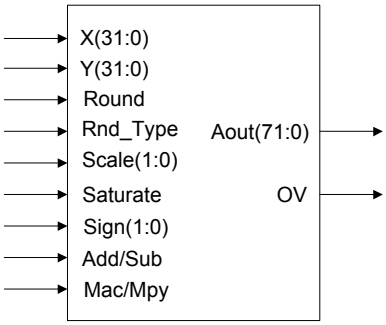

Figure 2: MAC block diagram.

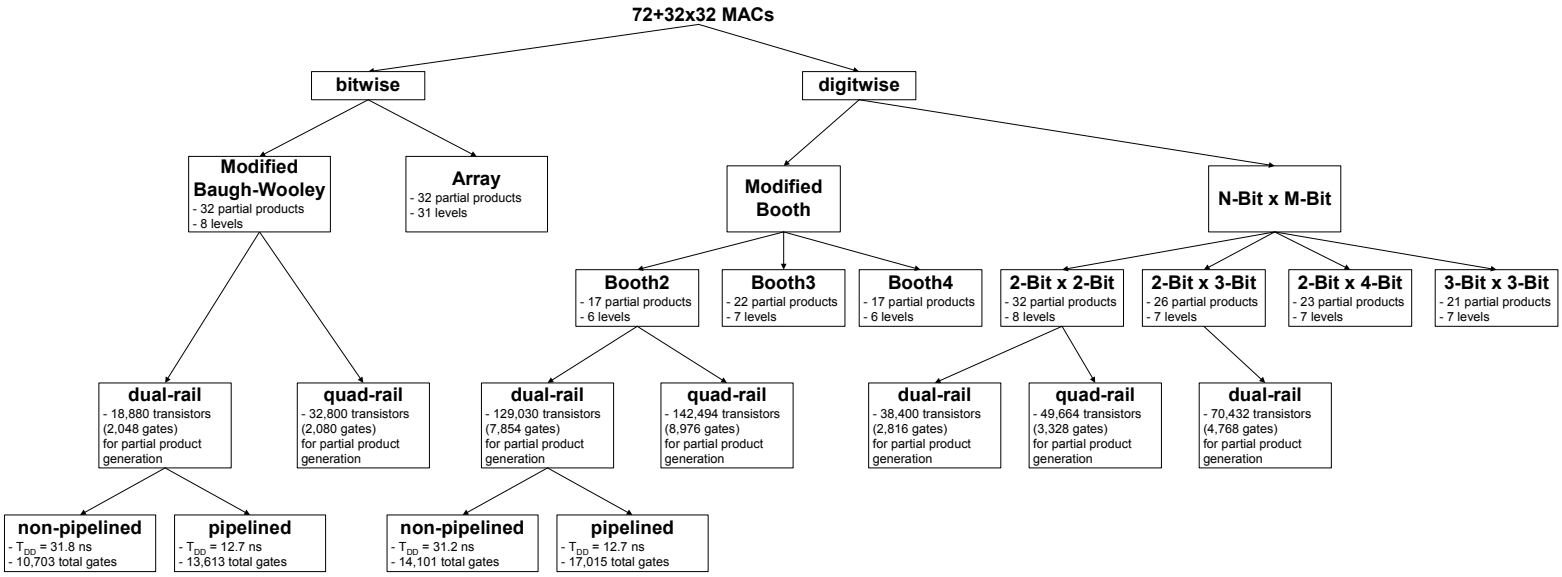

Figure 3. Taxonomy of 72+32×32 MACs.

```
if (LSB >= 0.5) then
        MSB = MSB + 1
else if (LSB < 0.5) then
        MSB = MSB
end if
LSB = 0
```

Algorithm 3.1: 2^s-complement rounding.

```
if (LSB > 0.5) then
        MSB = MSB + 1
else if (LSB < 0.5) then
        MSB = MSB
else if (LSB = 0.5) and (the least significant bit of MSB = 0) then
        MSB = MSB
else if (LSB = 0.5) and (the least significant bit of MSB = 1) then
        MSB = MSB + 1
end if
LSB = 0
```

Algorithm 3.2: Convergent rounding.

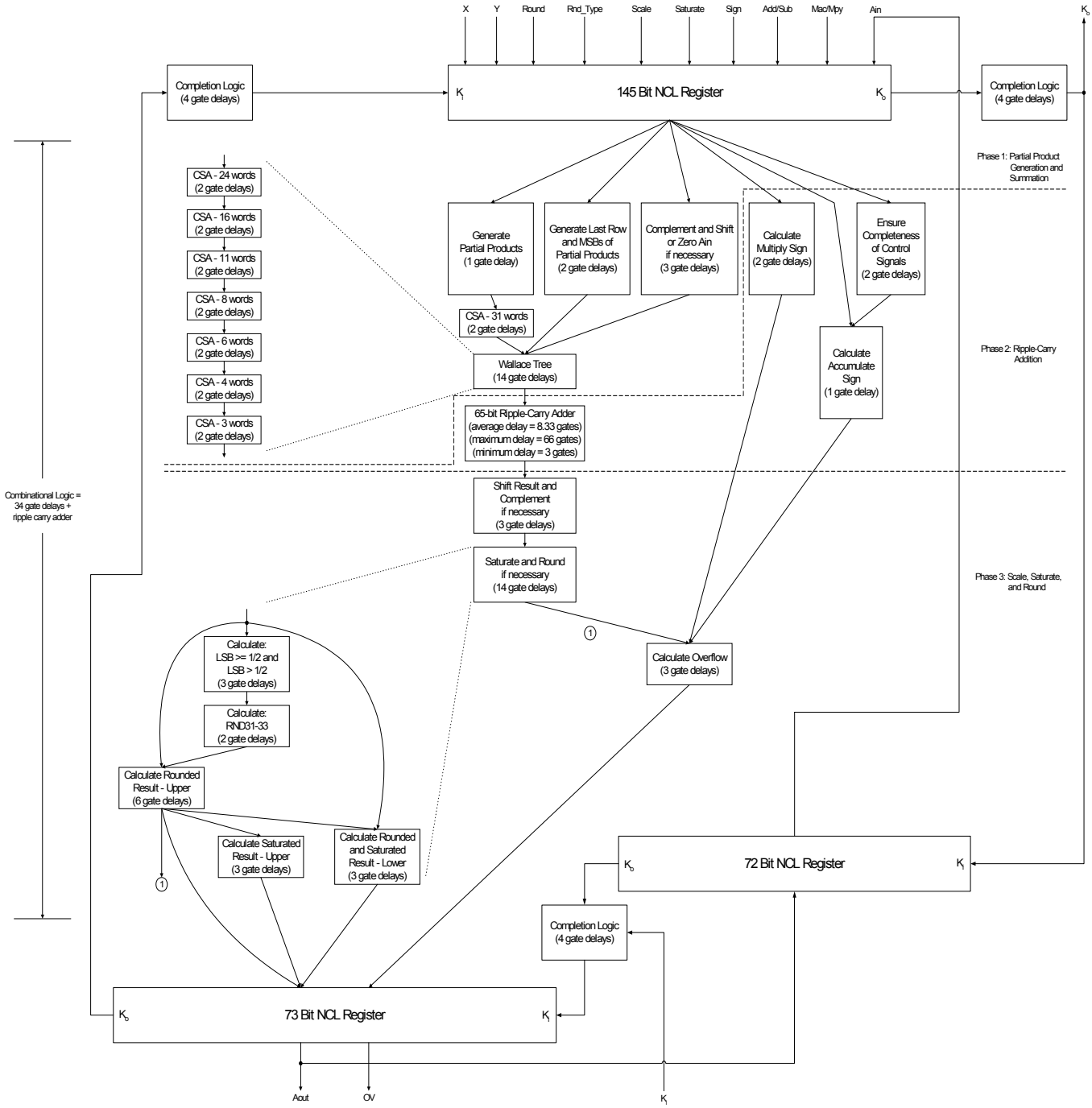

Figure 4. Non-pipelined Modified Baugh-Wooley MAC.

a)

71 64	63 31	30 0
Extension	MSB	LSB

b)

71 64	63 32	31 0
Extension	MSB	LSB

c)

71 64	63 33	32 0
Extension	MSB	LSB

Figure 5. Partitioning of the Output for a) up-scaling, b) no scaling, and c) down-scaling.

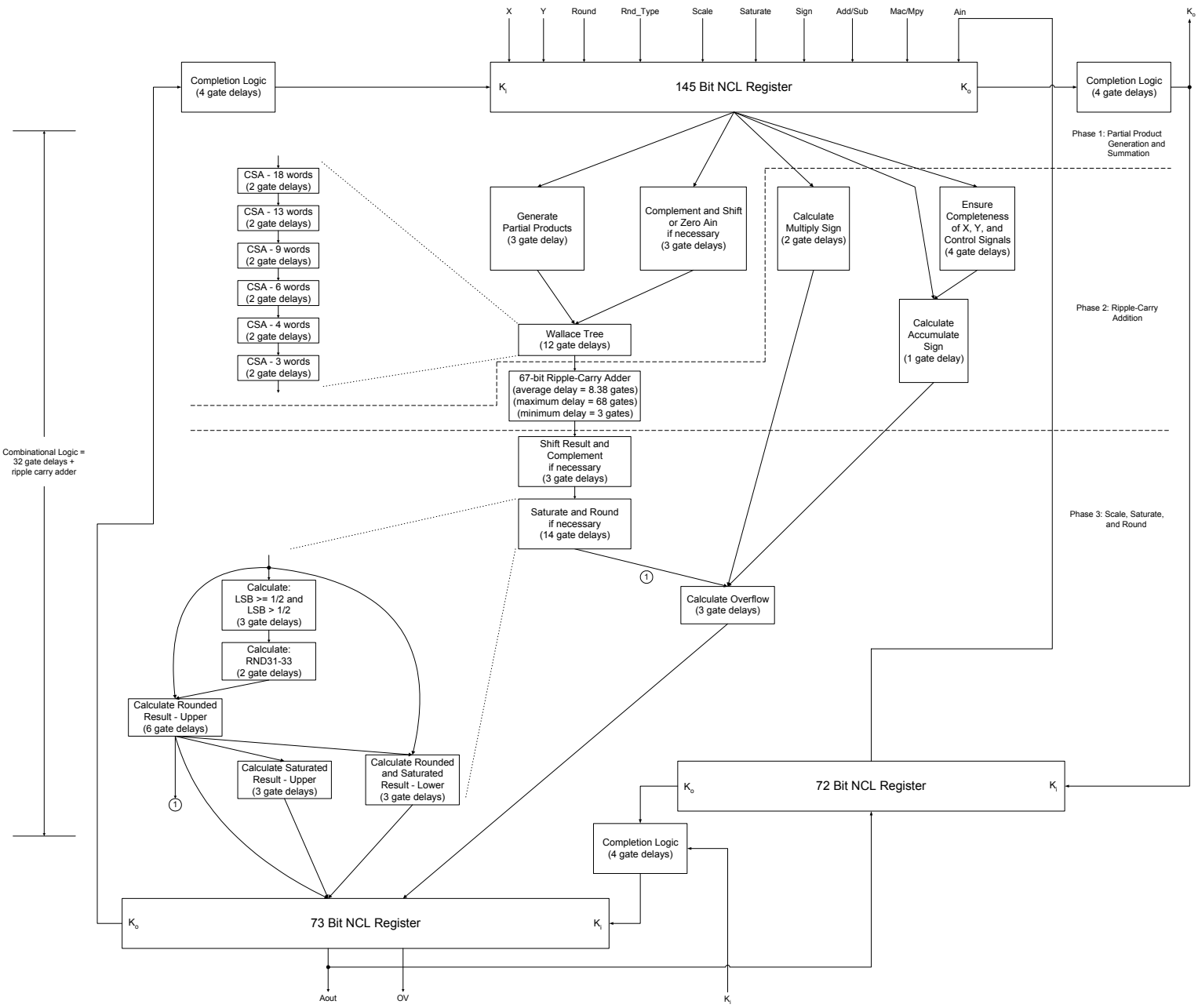

Figure 6. Non-pipelined Modified Booth2 MAC.

Table 1. Saturation Table.

B_{71}	B_{64}	B_{63}	**Saturated Result**	**Saturated and Rounded Result**
0	0	0	No Change	Result of Rounding Algorithm
0	0	1	00 7FFF FFFF	00 7FFF 0000
0	1	0	00 7FFF FFFF	00 7FFF 0000
0	1	1	00 7FFF FFFF	00 7FFF 0000
1	0	0	FF 8000 0000	FF 8000 0000
1	0	1	FF 8000 0000	FF 8000 0000
1	1	0	FF 8000 0000	FF 8000 0000
1	1	1	No Change	Result of Rounding Algorithm

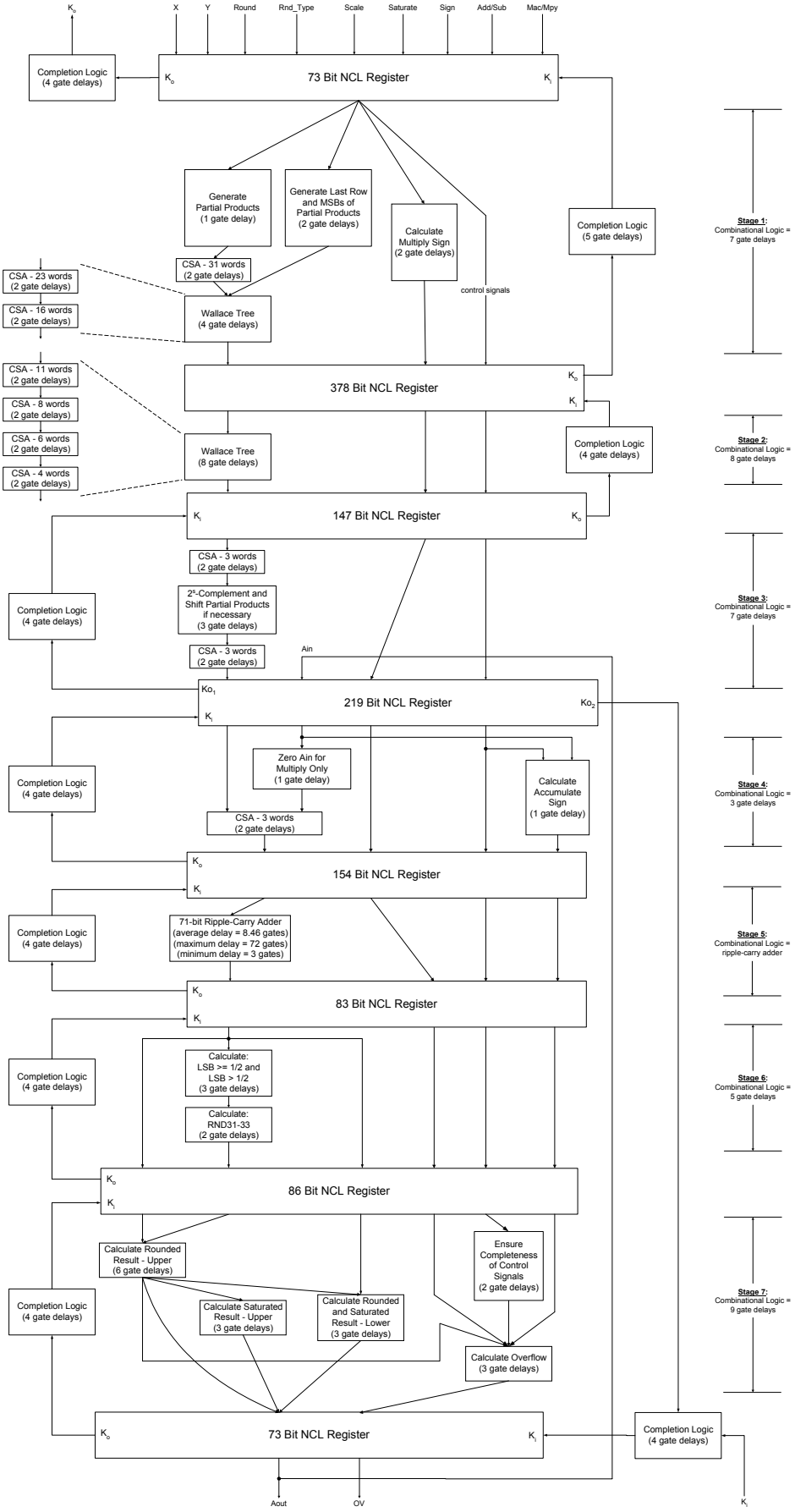

Figure 7. Pipelined Modified Baugh-Wooley MAC.

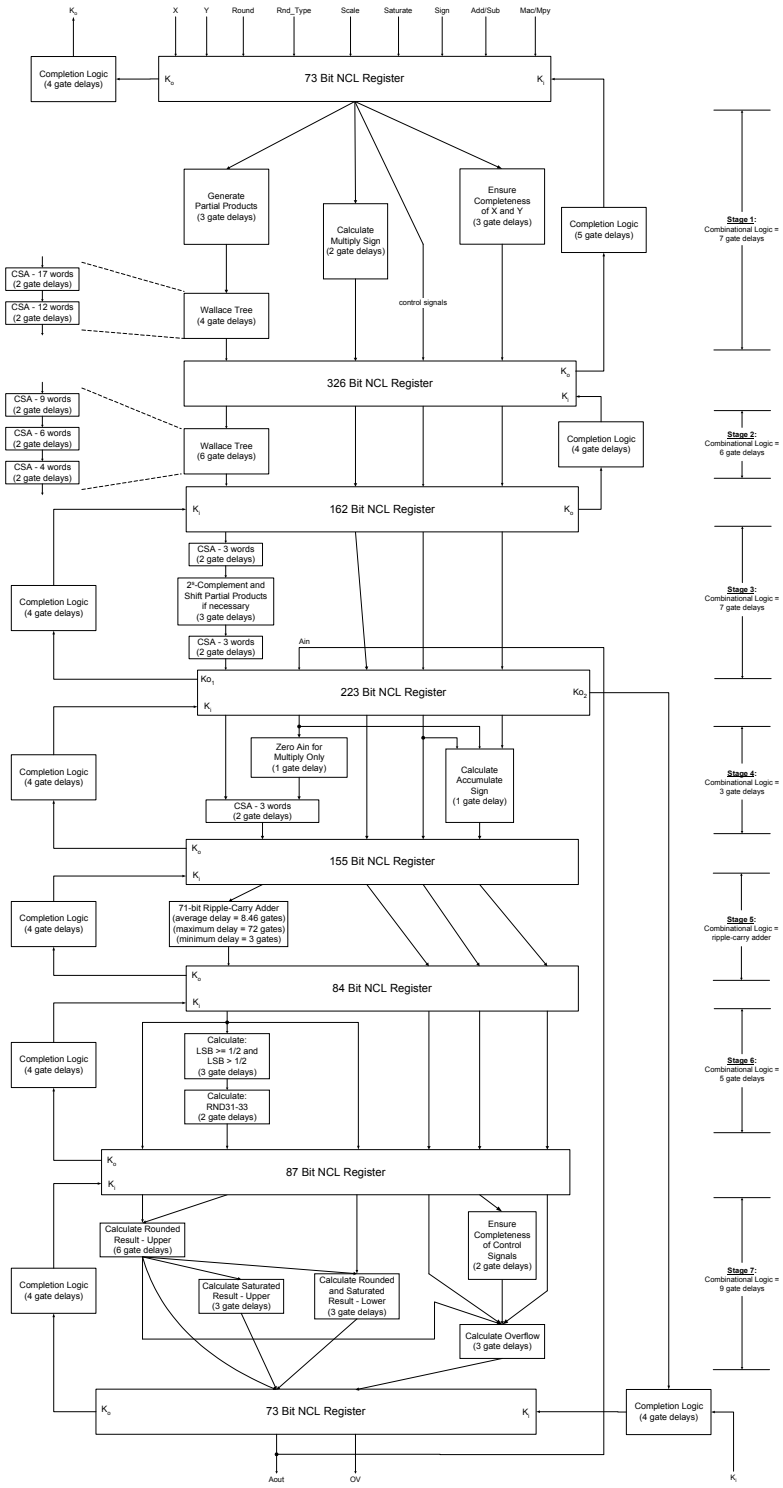

Figure 8. Pipelined Modified Booth2 MAC.

Table 2. MAC comparisons.

MAC Type	Algorithm	Technology	Avg. Cycle Time
72+32×32	Modified Baugh-Wooley	3.3 V, 0.25 μm CMOS	12.7 ns
64+32×32	Serial-Parallel [6]	5 V, 2 μm CMOS	901.0 ns
16+8×8	Modified Booth2 [7]	0.6 μm CMOS	90.0 ns
16+8×8	Conditional Evaluation [8]	3.3 V, 0.35 μm CMOS	7.8 ns
32+16×16	Conditional Evaluation [8]	3.3 V, 0.35 μm CMOS	24.0 ns