A High Throughput Multiplier Design Exploiting Input based Statistical Distribution in Completion Delays

Abstract—**Design methodologies such as Razor [3,4] minimize power dissipation by slowing down circuits so as to eliminate timing slacks to the point where occasional timing errors are observed. The main challenge here is the design of efficient mechanisms to detect and recover from these infrequent errors without loss of functionality. We present a design for widely used Wallace multipliers where, because of the highly skewed input based statistical distribution in completion delays, the potential for power and performance gains is significantly higher. Clock periods can be potentially reduced by a factor of 3 or more, with very rare timing errors for random input distributions. For error recovery we present a novel approach that latches and holds logic values at key internal circuit nodes during every clock cycle beyond the next clock edge. This allows generation of the correct outputs for that clock period one clock cycle later in case of a timing error. Meanwhile, very fast error evaluation, exploiting a unique characteristic of carry ripple addition, allows this hold to be quickly released if an error is not detected, ensuring no impact on the circuit timing in error free operation. While an additional area overhead of 10% was observed after implementing the design in a 32x32 Wallace Multiplier a 2.5x improvement in the average performance was achieved. Spice simulation results with varied clock period for 10000 vectors shows an optimum average performance improvement can be achieved at a reduced clock period of 3.75ns against the actual clock period of 9.5ns[8], the vectors which can trigger the critical path were obtained from [6]. Also, this design when deployed in a logic circuit would prove to be a Variation Tolerant Design.**

Index Terms - Wallace multiplier, delay distribution, completion sensing, clock period scaling.

I. Introduction

Achieving high performance within extremely limited power budgets is emerging to be one of the most difficult challenges in the design of current generation digital circuits and systems. Since most systems are synchronous, computations are typically allowed a fixed amount of time for completion, with the clock period selected to accommodate the worst-case completion time. Some margin is additionally allowed to allow for the process induced random performance variations that are being increasingly observed in scaled technologies. The circuit, however, still consumes significant static power during such timing margins, and other periods of inactivity, which is wasteful. The Razor approach, proposed in [4], aims at reducing this power consumption by minimizing this timing margin to zero and beyond (since timing critical paths are not activated in every cycle), by building in a system capability to detect occasional errors due to slow signal paths and recover from them. The timing margins are removed by reducing the supply voltage to slow down the circuit to a point where a small, acceptable number of errors are observed. As long as the power saving from the reduced voltage operation exceeds the extra power needed, on average, by the occasional error detection and recovery cycle, such a scheme can provide a net power saving. The challenge clearly is in designing an efficient low cost error detection and recovery capability to support this approach. The original Razor design [3] has changed and evolved [4,5,7] significantly over time in an attempt to achieve practicality. Even so the potential power savings from eliminating timing margins appear limited.

Instead of applying this approach of minimizing circuit timing slack with recovery from occasional errors to general purpose logic as attempted by the Razor team and others, we have been exploring its use in specific widely used computations such as addition and multiplication [1] where, because of the inherent characteristics in the applications, the potential for power savings can be much greater. Depending on the hardware implementation, these computations can display a wide range of completion delays depending on the applied inputs. For example a simple low cost (in hardware and power) 32-bit ripple carry adder (RCA) can generate a result with no carry propagation delays for some input cases, while requiring 32 stages of carry propagation in the worst case. Importantly, for random inputs, this delay distribution is highly skewed, which can allow a significant speedup in the operating clock period while still ensuring only a small error and recovery rate. This is seen in Figure 1, which simulates addition completion delays for a 32bit RCA designed at the gate level and, for simplicity, assumes fixed unit delay for every gate. Observe that the median addition delay is only 10 units when compared to the worst-case delay of 64 units. Note this does not mean that only 3-4 full adder stages generate a carry for the median case because strings of carries are generated and propagate in parallel at the same time. From the point of generation, a carry needs to see 01 or 10 inputs in the subsequent full adder stage for propagation, resulting in only a 50% chance that it will propagate through the next stage. The probability of a carry propagating n stages is 2^n , which dies out quickly with increasing n. Consequently, it if observed in Figure 1 that compared to a worst case delay of XXX less than 0.1% of random inputs result in an addition delay of greater than 25 units. This suggests that the addition can potentially be run at 2.5X the worst case speed, with only one in a thousand operations, on average, requiring correction and

recovery. Of course the overall system, utilizing such an adder must be designed to support and work with this occasional recovery operation. Fortunately, architectural level handing of exceptions through pipeline stalls and out of order instruction issue are now well understood and commonplace in processors to handle other forms of speculative execution; our design can be considered just another speculative implementation. Note the greatly reduced average clock period results in significant energy saving per computation because of the saving in the static power wasted during the mostly inactive part of the longer clock period in a traditional design. Moreover, this performance gain can be traded off for lower power consumption by reducing the supply voltage to slow down the circuitry instead of speeding up the clock as in the Razor[3] approach.

Our focus in this paper is low cost, high throughput integer multiplication. To portray the efficacy of our approach we present the design of a 32x32 Wallace Multiplier. Arithmetic circuits, particularly the multiplier, often have a decisive impact on overall timing in electronic systems. Moreover, multiplication being a very important and commonly used operation consumes a significant portion of the systems power in modern processors. Traditionally, state-of-the-art multiplier designs have focused on the performance over hardware cost and power, with numerous high-speed designs proposed in the literature. Many high speed multipliers employ a carry save approach, based on designs proposed by Wallace [10] and Dadda [11], that first quickly reduces the input based partial products to two numbers without any carry propagation delays, and then quickly adds these two numbers using high speed adders. Enhancements speed up each of these two steps at considerable expense in hardware and power. For example, the high speed multiplier described in [9] employs Booth encoding on the operands to reduce the number of partial products to be reduced by the carry save tree, specialized compressor circuits for carry save partial product reduction, and a mixed carry-select, carry look-ahead topology to add the final two accumulated partial products. Our goal is to achieve comparable speed for almost all the multiplication operations without many of these enhancements for partial product reduction, and the use of a much lower cost carry ripple adder to generate the final product; only rarely requiring recovery for a timing error due to the activation of a long path. Furthermore, it is critical for the timing error detection and correction circuitry to be fast, low cost and robust. Developing such a timing exception recovery capability is a major contribution of this paper.

To achieve this error recovery we present a novel low cost approach which involves latching and holding logic values at key internal circuit nodes during every clock cycle beyond the clock edge to allow recreation of the correct outputs for that clock period one clock cycle later in case of a timing error. Then very fast error evaluation, exploiting a unique characteristic of carry ripple addition, allows this hold to be quickly released without any impact on the circuit timing in the next period cycle in error free operation. The rare detection of a timing error (long carry propagation path) extends the hold signal for additional one or more clock cycles to give the computation time to complete. *Note that this approach is very different from those previously proposed for Razor or other work in the literature, and is particularly well suited to exploit the architecture of low cost Wallace multipliers.* CRISTA[6] is also a very different methodology which deliberately isolates long paths during design so that they can be deterministically allowed two cycles to complete.

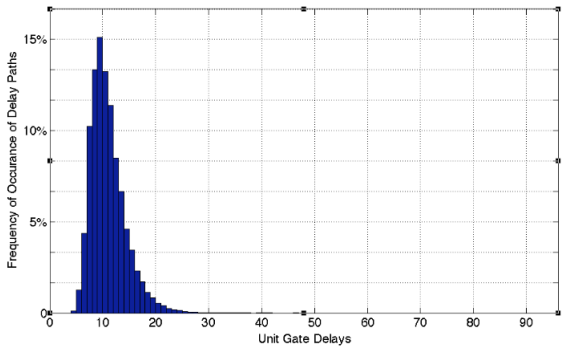

Fig. 1. Delay Distribution of 32 bit RCA

The rest of the paper is organized as follows. In section II, we review Wallace multipliers. In section III we present an overview of the proposed approach, and describe the internal signal hold until completion confirmed recovery scheme. The error detection mechanism and recovery timing is described in Section IV. Section V then presents the implementation of a 32x32 multiplier designed in 180nm technology. Simulation results are provided in section VI. Section VII concludes the paper.

II. Review of Wallace Multiplier

The parallel multiplier design with the lowest hardware cost is arguably the array multiplier[9]. Here the partial products generated from the inputs using AND gates are added using a regular NxN array of full adders (for n-bit multiplication). Unfortunately, the array design is slow, with a worst-case carry propagation path of length 2n-2 full adder stages, with additional slow paths that are successively shorter. Significantly speeding up all these slow paths with additional hardware is difficult. The popular Wallace multiplier overcomes this problem by performing the partial product addition in two steps. In the first step, the numerical values in the n partial products are accumulated into two binary numbers without any long carry propagation delays using a carry save tree for full adders. In the second step these two numbers are added together using a conventional carry look ahead, carry save, carry skip, or other high-speed adder[9].

The method behind the Wallace Multiplier is illustrated in Figure 2 for 8x8 multiplication. The initial 8x8 array of dots at the top of the figure represents the 8 (8 bit) partial products that have to be added together; each dot representing a bit position. These bits are generated from the appropriate multiplier and multiplicand positions using AND gates and are appropriately aligned by bit weight. In the first stage of a Wallace carry

save tree, partial product bits with the same bit-weight are grouped together in groups of three, and sometimes also two, as shown in Figure 1. These groups of bits are then added together using as many full adders (half adders for the groups of two) as necessary. Each full adders sum output results in a bit (dot) located in same column in the next layer of the carry save tree; the carry output results in a dot in the next more significant column (to the left). Note that the three dots from a column at the input of each full adder, result in two dots in the next layer, one in the same column and one in the next. This reduces the total number of dots in the next stage by 2/3 (3,2 compression). Since all full adders in each stage of the carry save tree operate in parallel, the delay for any stage in a single full adder delay. The process is repeated stage by stage until only two numbers remain to be added. A fast adder is typically used to perform this last step in completing the multiplication.

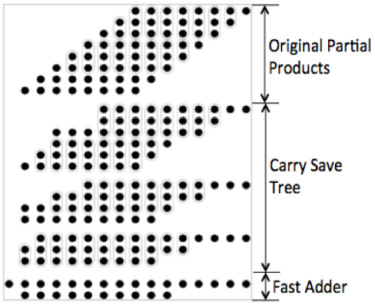

Fig. 2. Illustration of Wallace Multiplication Technique

Observe that the total multiplication time in such a scheme is the sum of the delays in generating the partial products, carry save reduction of the partial products, and the final fast addition. Partial product generation is relatively quick requiring only an AND operation. For the 8x8 bit example in Figure 2, the partial product reduction to two numbers required 4 full adder stages; this delay grows slowly as log of the size of the multiplier input word for larger multipliers. (As discussed below, in more recent implementations, the carry save tree employs 4:2 compressors instead of full adders to further reduce the delay of each stage.) Fast addition of the final two numbers with minimal carry propagation delays is critical to multiplier performance. In our example, the five least significant bits of the product are already available and need not be further added, thereby requiring an 11 bit final addition. However, the number of bits in the final two numbers to be added grows linearly with the multiplication word size, for example it is 59 for the 32x32 bit multiplication implemented later in this paper. This last step is usually achieved with hardware intensive state-of-the-art adder designs in high performance circuits. Other performance enhancement approaches aim at reducing the number of starting partial products by recoding the multiplier inputs, and reducing the number of carry save stages required by using larger counters or compressors. All these can add significantly to the area and power requirements of the multiplier.

The use of compressor circuits in place of regular full adders in the carry save tree is aimed at reducing the time required to compress the partial products down to two numbers. Observe in Figure 3 that the signal delays at the sum and carry outputs of a full adder are not the same. The carry out is a fast signal with two gate delays, while the sum function requires two EXOR delays, where each EXOR itself is realized from two levels of primitive gates. Therefore, in the traditional Wallace carry save tree, each partial product reduction stage is subject to the worst case sum delay of two EXOR gates (four primitive gates). 4,2 compressor circuits cleverly interleave the fast (carry) and slow (sum) signals through the full adders in two successive stages of the carry save tree to ensure that the fast signals encounter a slower path in the next stage, while the slow signals encounter faster paths. This results in a critical path delay through two stages of 3 EXOR gates instead a 4 as shown in Figure 3. The 32x32 design presented in this paper is optimized using such 4, 2 compressors.

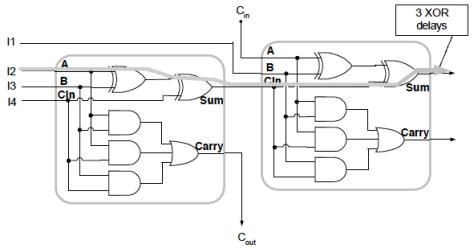

Fig. 3. 4:2 Compressor[12]

Our proposed approach to low power high throughput multiplication leverages the highly skewed distribution of carry propagation delays in ripple carry addition, and therefore dispenses with the need for power hungry high speed addition circuitry. A first sense of the potential of the proposed statistical approach can be seen from the unit delay gate level simulation results in Figure 3 for a basic 32x32 Wallace Multiplier employing a simple ripple carry adder to generate a 64 bit product (implemented as illustrated in Figure 2 for the smaller 8x8 example). The highly skewed distribution of the Wallace Multiplier is primarily due to the Ripple Carry Adder in the final stage. While the worst case delay was found to be 129 units [6], it can be seen that the median delay is only 37 units. Furthermore, it is observed that very few vectors trigger delays greater than 50-60 units. If the clock period is reduced from the worst-case completion delay of 129 units to about 55 units, it is possible to achieve an approximately 2.3X improvement in performance, with a rare timing error. Furthermore, if these occasional errors can be efficiently detected and recovered from within one or two additional clock cycles, the system can achieve robust multiplication with low energy per computation; the static power dissipation during the long, mostly inactive parts of a full length 129 unit clock period will be saved. (As discussed earlier, this $> 2X$ performance speed up can be easily traded-off for power savings by reducing the supply voltage which reduces power while increasing circuit delays, and/or using

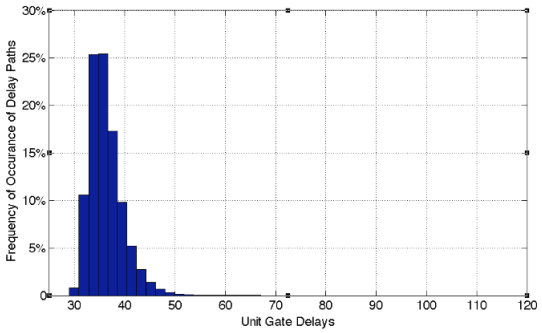

Fig. 4. Delay Distribution of 32x32 Wallace Multiplier

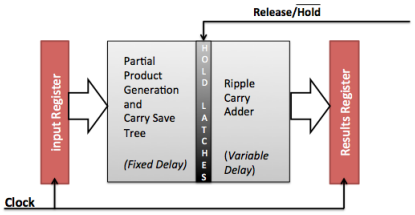

Fig. 5. Pipelined Schemtic for Wallace Multiplier Operation

slower, less leaky transistors.)

Clearly, designing an efficient and effective low cost timing error detection and recovery mechanism is the main challenge of such an approach. This is the main contribution of this paper.

III. The Internal Signal Hold Until Completion Confirmed Scheme

The schematic in Figure 4 illustrates pipelined operation of our Wallace multiplier design at a high level, where new inputs arrive and the current cycle results are captured and made available on each edge. The multiplier itself is broken up into two parts, the front-end logic that generates the partial products along with the carry save tree (these mostly display a fixed delay relatively independent of inputs), and the carry ripple adder with highly variable delay. For purposes of this initial discussion we assume that there exists a register of latches between these two parts that can hold the output signals of the carry save tree (the inputs to the ripple carry adder) for a desired duration. (In practice, as we shall see later, his capability can be more cost effectively implemented using tri-state gates that create dynamic latches.) This (low active) hold signal is activated on each active clock edge, but released (making the latches transparent again) shortly thereafter if no error is detected. *Observe that if the hold activated at the start of a clock cycle is released within a period less than the propagation delay of the first (carry save tree) part of the logic, i.e. before new logic values due to the updated inputs propagate to the latch register, the activation of the hold in each cycle does not in any way restrict the propagation of logic signals*. Therefore it has no impact on the overall multiplication delay, beyond the extra logic and loading delays introduced by the insertion of the latches.

As was observed in Section 2, because of the highly skewed completion delay distribution of the carry ripple adder, even with the system running at a clock 2-3X faster than the worst case delay, errors in the result of the multiplication due to timing violations on long paths can be expected to be extremely rare. Assume for the moment that a capability for the fast detection of such errors exists, i.e. it is possible to know quickly after the clock edge that captures a result in the results register whether or not that result is correct. This information can be used to control the hold/release signals for the latches, as shown in the timing waveforms in Figure 6. In case an error is detected, the hold on the latches is not released for the remainder of the cycle, and into the next cycle, until an error free result in the next cycle is confirmed. Observe that since the hold is activated on the same clock edge that updates the input registers, changes from the updated inputs are unable to immediately propagate through the carry save tree to influence the values in the latches. Thus in the hold mode the latches always hold signal values from the previous clock cycle. If not released in the event of an error, the hold latches continue to hold the values from the previous cycle at the inputs of the ripple carry adder, giving that computation an additional cycle to complete. The correct result is then available one cycle after the incorrect result for the same multiplication was captured in the results register. This can even extend to a third (or fourth) cycle for worst-case carry propagation until an error free result is confirmed. Meanwhile, the error flag can also be used as a signal to supervisory system for appropriate handling of the input and output values in each clock cycle. Erroneous cycles must be marked as stalls in pipelined operations, with the correct result following in the next cycle. Additionally, the flow of new inputs to the multiplier must be halted corresponding to each stall.

IV. Timing Error Detection

In the discussion in the previous Section we have assumed that a timing error, caused by a longer carry propagation path in the ripple carry adder than was allowed for in setting the clock period, can be quickly detected following the capture clock edge. The key to this capability lies in an important characteristic associated with the propagating carries. Observe from Figure 5 that the latches go into the hold mode and retain the signal values at the input of the ripple carry adder on the clock edge that captures a new multiplication result.

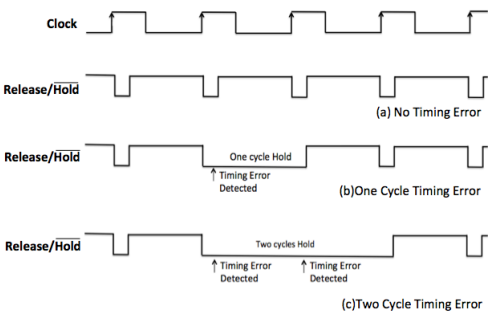

Fig. 6. Timing Model Waveforms for Hold Until Completion

This allows the ripple carry adder to continue working on the previous (just captured) result until the hold is released. In most cases, the outputs of the ripple carry adder stabilize before the clock edge, and the correct multiplication result is captured in the results register. Here the outputs of the ripple carry adder (inputs to the results register) do not change even as the adder is given additional time to operate into the next clock cycle. However, if the outputs of the ripple carry adder have not stabilized at clock time because of a long carry propagation path, then they will continue to change after the clock edge. This condition can be detected by comparing each of the adder output bits with the corresponding bit captured in the results register at the clock edge a mismatch indicates a timing error. A timing error detection circuit can therefore be constructed as shown in Figure 7.

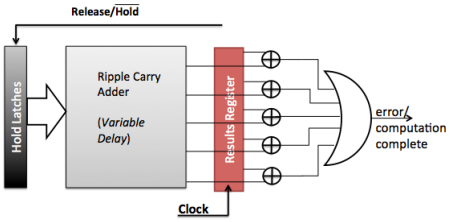

Fig. 7. Timing Error Detection Circuitry

Recall that a propagating carry in the carry ripple adder causes a change in the sum output of every full adder stage it propagates through. This is because the sum is is formed using EXORs on three inputs, two of which are stable as the carry propagates. *Thus if final stable outputs are not captured on a clock edge, at least on ripple carry adder output can be expected to change within a full adder delay of the clock edge.* (It is impossible for the adder outputs to show no change over any full adder delay window, but then change with a larger delay.) This allows the error/computation complete indication in Figure 7 to be evaluated within a few gate delays of the clock edge, achieving the important requirement of quick completion confirmation that can allow the hold in the latches to be released without any performance penalty.

V. Implementation of a 32x32 Multiplier with Statistical Timing

A 32x32 Wallace multiplier with statistical timing was implemented as described in the earlier Sections with a carry save tree built from 4, 2 compressors, and a ripple carry adder for the final stage. A schematic of the design is shown in Figure 8.

The partial products are generated by performing the AND operation on the appropriate bits from the multiplicand and the multiplier, resulting in 32 partial products, each 32 bits long, arranged as illustrated in Figure 2 for 8x8 multiplication. The carry save tree then compresses the partial products in 8 stages using 4:2 compressors to two numbers that are added by the ripple carry adder. As can be observed form the example in Figure 2, some of the least significant bits of the multiplication

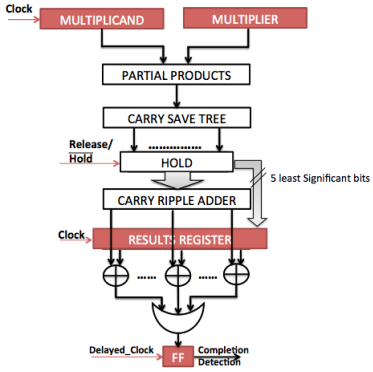

Fig. 8. Illustration of Wallace Multiplier with the Proposed Design

result are already available from the carry save compression and need not be added further. For the 32x32 multiplier, 5 bits are pre-generated in this manner, requiring a 59 bit ripple carry adder to add the remaining bits. (Note that the some of the 5 pre-generated bits least significant have very short logic paths through the carry save tree, and may need to be delayed through buffers to avoid a race condition at the hold latches.)

The hold block shown represent the latch and hold mechanism that is initiated on each clock edge, to continue to hold the inputs of the ripple carry adder until the error detection logic confirms that the correct multiplication result was captured at that clock edge. To minimize, performance and hardware overhead, this function is actually optimized in our design as a dynamic signal hold by tri-stating the driving output gates of the carry save tree. We thus embed the tri-state/hold logic within the final compression stage i.e. the EXOR and the NAND gates of the 4:2 compressors. This optimization is illustrated for a NAND output in figure 9.

As described in the previous section, detection whether or not the result captured on a clock edge is in error is achieved by comparing the captured result on a clock edge with the output of the ripple carry adder driving the results register. Since the inputs of the carry ripple adder are held stable past the active clock edge for all clock-cycles (until the hold is released), it can, if needed, continue working on the computation for which an initial result is captured on the clock edge. A subsequent mismatch between the ripple adder output and the captured result after the clock edge is an indication that the addition did not complete within the clock period. In our design an error signal is generated by latching this mismatch signal (from the EXOR-OR comparator circuit) using a clock appropriately delayed from the active clock edge. Since a propagating carry in the carry ripple adder causes a change in the sum output of every full adder stage it propagates through, this delayed clock need only incorporate the carry to sum output delay of a full adder, plus the comparator delay. In our design, this is about ten gate delays, allowing for the high fan in of the OR gate. Such fast error evaluation/completion detection, allows the tri-state/hold condition at the outputs of the carry save tree to be removed well before results for the new set of partial products for the next computation are

ready, thereby avoiding any performance penalty from the hold during an error free cycle.

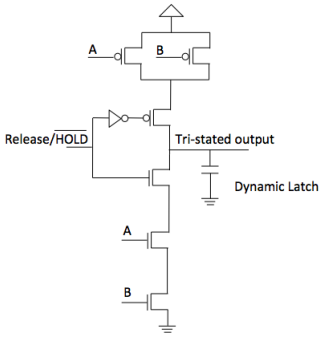

Fig. 9. Illustration of Logic Optimization at Hold Logic

A 32x32 Wallace multiplier with statistical timing incorporating the hold until completion detection logic described above was implemented at the RTL level and synthesized and optimized for timing using Mentor Graphics Spectrum in tsmc180nm technology. Parts of the design, such as the tri-state/hold logic in the 4:2 compressors in the final stage of the carry save tree, were embedded manually into at the transistor level. The entire design was then simulated for performance evaluation.

VI. Simulation Results

The timing simulations results shown in Table 1 were obtained using SPICE and tsmc 180nm technology files. The pessimistic worst-case delay of the multiplier without timing errors was found to be 9.5ns from the Mentor Graphics STA tool. Actual SPICE timing evaluation using know worst case inputs [8] yielded a delay of 9ns. This extremely rare worst case delay was not observed in the simulation of 10,000 random inputs in SPICE.

Table I shows the number of timing errors observed for the 10,000 random inputs when the allowed clock period for multiplication completion is reduced from the worst case 9ns to a range between 5ns and 3.5ns. It can be observed that as the clock period decreses the number of errors with a single clock period penalty increase very quickly. This is consistent with the delay distribution in Figure 4. For yet shorter clock periods, errors that require two additional clock cycle also begin to grow rapidly. Only 4 out of the 10,000 inputs triggered a timing error with the clock period set at 5ns, and understandably no case required more than two clock periods (10ns). 26 errors were triggered with the clock period set at 4ns, with one case requiring more than two cycles (greater than 8ns)to complete. Column 4 in the Table shows the effective average time required per multiplication at the corresponding single cycle clock period. This can be used to select an optimum clock period, which from the table is 3.75ns for our design. At this clock period the average multiplication time is 3.86ns. Finally, the last column shows the performance improvement over using a worst case clock of 9ns for all multiplications. This is obtained by dividing 9ns by the average multiplication time in column 3. At the optimum clock period of 3.75ns, the performance improvement is 2.36X.

TABLE I
Spice Simulation Results with Reduced Clock Periods

Reduced Clock period ns	Error Count 10,000 random inputs		Average Clock Period ns	Performance Improvement (actual clk/avg clk)
	(one cycle penalty)	(two cycle penalty)		
5	4	0	5.002	1.79x
4.5	10	0	4.5045	1.99x
4	26	1	4.015	2.24x
3.85	113	3	3.91	2.3x
3.75	257	13	3.856	2.36x
3.5	1876	47	4.2	2.14x

Direct comparisons of the performance of this new design with other designs in the literature are difficult to make, even at the same feature sizes, because of differences in the cell libraries and technology files, particularly the threshold and supply voltages used in the reported simulations. Nevertheless, the fastest multiplier described in the literature in comparable 180nm technology reported a simulated multiplication time of 2.85ns. This design employed a Kogge-Stone fast adder and other enhancements. obtain the error signal. While, the area and delay overhead associated with this kind of a comparator turns out to be significant, one can use fast comparators such as [15] and [16] where the output is precharged, and pulled down on a mismatch in any bit position during the evaluation phase, causing energy dissipation to flag an error/mismatch and thus ensuring that the total performance benefit is not affected.

VII. Conclusion

In this paper we have proposed a novel design approach for exploiting the highly skewed statistical variation in completion delays observed in low cost Wallace multipliers implemented with ripple carry adders in the final stage. In the vast majority of cases, such multipliers complete the computation in well under half the worst case delay, making it possible to operate them with a 50-60% shorter clock period with fewer than one timing error every thousand multiplications. To support such operation, we have developed a novel internal signal hold until completion detection recovery scheme that automatically allows such an ”overclocked” multiplier to steal one or more additional clock periods as needed to transparently complete the infrequent computation that needs additional time. Of course the overall system, utilizing such a multiplier, must be designed to support and work with this occasional recovery process. Fortunately, architectural level handing of exceptions through pipeline stalls and out of order instruction issue are now well understood and commonplace in processors to handle other forms of speculative execution; our design can be considered just another speculative implementation.

Achieving comparable deterministic multiplication speed requires very high speed, long word length, adders in the design which can require 3-4X more gates/area and consume significantly more power. By reducing the clock period to the point of encountering an acceptable number of timing errors,

our design, like the Razor[3] approach, eliminates the wasted static (leakage) power during the frequent quiescent intervals observed in traditional designs that reliably allow for rare worst case signal propagation. This greatly reduces the average power consumed per multiplication, which is critical in battery powered mobile applications.

REFERENCES

[1] Temporarily removed for blind review

[2] Borkar, S.; Karnik, T.; Vivek De; , "Design and reliability challenges in nanometer technologies," *Design Automation Conference, 2004. Proceedings. 41st* , vol., no., pp.75, 7-11 July 2004.

[3] Ernst, D.; Nam Sung Kim; Das, S.; Pant, S.; Rao, R.; Toan Pham; Ziesler, C.; Blaauw, D.; Austin, T.; Flautner, K.; Mudge, T.;"Razor: a low-power pipeline based on circuit-level timing speculation," *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, vol., no., pp. 7-18, 3-5 Dec. 2003.

[4] Blaauw, D.; Kalaiselvan, S.; Lai, K.; Wei-Hsiang Ma; Pant, S.; Tokunaga, C.; Das, S.; Bull, D.; , "Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance," *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International* , vol., no., pp.400-622, 3-7 Feb. 2008.

[5] Fojtik, M.; Fick, D.; Yejoong Kim; Pinckney, N.; Harris, D.; Blaauw, D.; Sylvester, D.; "Bubble Razor: An architecture-independent approach to timing-error detection and correction, *"Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, vol., no., pp.488-490, 19-23 Feb. 2012.

[6] Temporarily removed

[7] Choudhury, M.; Chandra, V.; Mohanram, K.; Aitken, R.; , "TIMBER: Time borrowing and error relaying for online timing error resilience," *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010* , vol., no., pp.1554-1559, 8-12 March 2010.

[8] Eriksson, H.; Larsson-Edefors, P.; Eckerbert, D.; , "Toward architecture-based test-vector generation for timing verification of fast parallel multipliers," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.14, no.4, pp.370-379, April 2006.

[9] J.M. Rabaey, *Digital Integrated Circuits*. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[10] Wallace, C. S.; , *"A Suggestion for a Fast Multiplier," Electronic Computers, IEEE Transactions on* , vol.EC-13, no.1, pp.14-17, Feb. 1964.

[11] L. Dadda. Some Schemes for Parallel Multipliers. *Alta Freq*., 34:349-356,1965.

[12] Vojin G. Oklobdzija High-Speed VLSI Arithmetic Units: Adders and Multipliers.

[13] Oklobdzija, V.G.; Villeger, D.; Liu, S.S.; , "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach ," *Computers, IEEE Transactions on* , vol.45, no.3, pp.294-306, Mar 1996.

[14] Vratonjic, M.; Zeydel, B.R.; Oklobdzija, V.G.; , "Low- and ultra low-power arithmetic units: design and comparison," *Computer Design: VLSI in Computers and Processors, 2005. ICCD 2005. Proceedings. 2005 IEEE International Conference on* , vol., no., pp. 249- 252, 2-5 Oct. 2005.

[15] Ergin, O.; Ghose, K.; Kucuk, G.; Ponomarev, D.; , "A circuit-level implementation of fast, energy-efficient CMOS comparators for high-performance microprocessors," *Computer Design: VLSI in Computers and Processors, 2002. Proceedings. 2002 IEEE International Conference on*, vol., no., pp. 118- 121, 2002.

[16] Chua-Chin Wang; Hsin-Long Wu; Chih-Feng Wu; , "A fast dynamic 64-bit comparator with small transistor count," *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva*. The 2000 IEEE International Symposium on , vol.5, no., pp.545-548 vol.5, 2000.

[17] Nagendra, C.; Irwin, M.J.; Owens, R.M.; , "Area-time-power tradeoffs in parallel adders," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions* on , vol.43, no.10, pp.689-702, Oct 1996.