A Standard-Cell Self-timed Multiplier for Energy and Area Critical Synchronous Systems *

Kip C. Killpack Eric Mercer Chris J. Myers
Electrical Engineering Department
University of Utah
Salt Lake City, UT 84112

Abstract

This paper describes the design of a standard-cell self-timed multiplier for use in energy and area critical synchronous systems. The area of this multiplier is bounded by N rather than N^2 as seen in more traditional combinational parallel array designs, where N is the word size. Energy has a polynomial growth with word size, but has a coefficient that is much smaller than that seen in a combinational array design. Although the multiplier is self-timed, it can be embedded in a synchronous system appearing as a combinational element. This paper presents latency, area, and energy estimates for the multiplier implemented at various word sizes, and compares these numbers with a traditional combinational array multiplier. The self-timed multiplier uses $\frac{1}{3}$ the energy and $\frac{1}{7}$ the area of the combinational design for a 24-bit word size.

1: Introduction

SONIC Innovations produces a digital hearing aid that presents an interesting optimization challenge. The hearing aid filters incoming sound into specific frequency bands that can be amplified according to a customer's hearing loss profile. The filter algorithm requires 40 multiplies and 120 additions to be completed on each incoming sample that arrives at a rate of 20kHz. Key considerations in this hearing aid design are chip energy to maximize battery life and chip size to minimize production costs. As the filter algorithm operates on real numbers, a number representation must be chosen that provides a sufficient dynamic range for sound processing without compromising the energy and area budget. The first generation hearing aid required 3 floating point adders and 1 floating point multiplier running on an internal 1.25MHz clock to complete the required number of operations. The floating point representation was chosen over a fixed point representation to reduce the impact of the multiplier on the area budget. If the area of the multiplier itself is further reduced, then the use of a fixed point representation can be considered. This would mitigate the complexity of the most common operation, which is addition.

Another important point to consider is the impact of the circuit architecture on future generations of the hearing aid. As the hearing aid design evolves through fabrication processes, it is important to consider the impact of routing and scheduling of resources on

*Supported in part by NSF CAREER award MIP-9625014, SRC contract 97-TJ-487, a grant from Intel Corporation, and a Wayne Brown Fellowship.

the energy, area, and timing budget. It is possible to meet a resource budget by designing an architecture that uses a single fast multiplier. However, having to share and schedule a single multiplier creates a routing challenge that decreases the area savings of using a single multiplier. Moreover, in deep sub-micron processes, routing is expensive because the delays in longer wires dominate gate delays. Thus, routing information over large distances in a single clock cycle is problematic. For these reasons, the multiplier presented in this work targets area as its main cost criteria. If the multiplier is small enough, it can be duplicated numerous times on chip to address scheduling and routing issues without overstepping the area budget. In addition, because the multiplier is meant to be used in a battery operated device, it is important to consider energy consumption. If the multiplier design is not low energy, then the cost of replication on the energy budget would make its use prohibitive.

This paper focuses on the energy and area impact of the multiplier by presenting the design of a standard-cell self-timed multiplier that can be embedded in synchronous applications. It is self-timed in that it generates its own timing reference for iterations. The multiplier bounds area growth by N, where N is the word size of the operands. This results in a significant area savings when compared to the N^2 growth seen in combinational implementations. The self-timed nature of the multiplier frees the environment from distributing a high speed external clock to perform the multiplier iterations, and it allows the multiplier to be used like a standard combinational component in a synchronous framework. To show the potential of the multiplier design, this paper presents energy, area, and latency estimates for the multiplier implemented on various word sizes. It then compares and contrasts these results to a fully combinational parallel array multiplier.

The remainder of this paper is organized as follows. Section 2 discusses related work in multiplier design and points out contributions in this work. Section 3 discusses the multiplier's architecture and implementation. Section 4 demonstrates the self-timed control for the multiplier. Section 5 gives some results for various word sizes and compares them against a traditional combinational implementation. And finally, Section 6 draws conclusions and discusses future work for the self-timed design.

2: Related work

Self-timed multiplier designs can be broadly grouped into 2 categories: parallel array and serial-parallel (i.e., iterative). A parallel array design uses on the order of N^2 full adders in a $N \times N$ configuration, and it is often pipelined for increased throughput [1, 3, 5, 9, 10, 13, 17]. The area impact of parallel array designs can be reduced using radix-4 Booth recoding [2], but Booth recoding does not affect the $O(N^2)$ area bound seen in full parallel array designs. In [4, 7], self-timed and synchronous parallel array designs are compared to show the power savings found in self-timed design methodologies. Moreover, [7] shows power to be polynomial in both design styles. Although self-timed parallel array designs are fast, they require a considerable amount of area and are thus, not appropriate for the hearing aid application.

Iterative or serial-parallel multipliers generally use on the order of N full adders N times to complete a given multiplication. This increases the latency of the multiply while substantially reducing the area. In [12], a *delay insensitive* design style is employed to remove internal glitches on all wires to save energy. However, it requires 2 wires to represent each bit in the multiplication, causing an increase in area. In [15], several self-timed designs

are compared, but all have considerable area penalty to achieve self-timing.

To remove the self-timed area overhead, work in [8], employs a *bundled-data* design style to implement an iterative multiplier. Bundled-data design replaces the traditional fixed frequency clock in synchronous design with individual delay elements that are matched to the latency of each stage of the design. Thus, each stage has its own unique delay line to signal the completion of the stage to the controller. Furthermore, [8] builds each stage's delay line using elements from the stage's critical path, and it matches load to the critical path at each gate. This allows the delay lines to scale evenly with the critical path delay under voltage, temperature, and process variations. Work in [8] further optimizes the design by skipping iterations in the algorithm according to the operands (i.e., a 0-bit in the multiplicand). Although [8] achieves better area than [12], the control for the multiplier still falls directly on the critical path. This is because a control communication is required at each stage of the design.

Work in [14] addresses control and latency issues in an iterative multiplier design using a stoppable clock with a partial parallel array. In a complete parallel array, area is bounded by $O(N^2)$ because for each bit of the multiplicand there is a row of full adders. However, [14] implements several but not all rows of the parallel array and then reuses those rows to complete the multiply. This creates a small area multiplier that still has acceptable latency for a 64-bit word size. In the extreme, the design in [14] would use a single row of full-adders $\frac{N}{2}$ times. The $\frac{N}{2}$ comes from the radix-4 Booth recoding that retires two bits at a time. The frequency of the stoppable clock is matched to the worst-case critical path delay in the design. When operands are ready to be multiplied, the clock is started, it runs a fixed number of iterations, and then stops to wait for the next set of multiplicands. The stoppable clock removes control from the critical path because communications are no longer required between each stage of the design. When the clock pulse arrives, it is assumed that all stages are ready for the next input.

The multiplier presented in this paper uses a similar bundled data design style to that of [8], however it removes control from the critical path through the use of a stoppable clock as in [14]. This paper presents an evaluation of the extreme case of the class of multipliers proposed in [14]. However, with the hearing aid application as the guiding practical example, this work strives to further optimize area and energy above latency and throughput. Therefore, in this multiplier design, the extreme case critical path is not limited to a row of carry save adders as in [14]. Instead this multiplier design includes the Booth recode with the carry save adders on the critical path. This eliminates a set of pipeline latches and reduces the number of generated clock cycles needed to complete the multiplication. As a byproduct of creating a very shallow pipeline and reducing the overall gate count, energy is reduced.

3: Architecture

The micro architecture for the multiplier is shown in Figure 1. This multiplier encodes the operands using Booth recoding so that it can handle *two's complement* numbers without any pre or post calculations [2]. Radix-4 Booth recoding is used because it requires only half the number of iterations required by radix-2 Booth recoding, while the multiples of B needed are still easily computed during iterations.

The *Selector B* block outputs multiples of B to the *Carry Save Adder Shift Register*

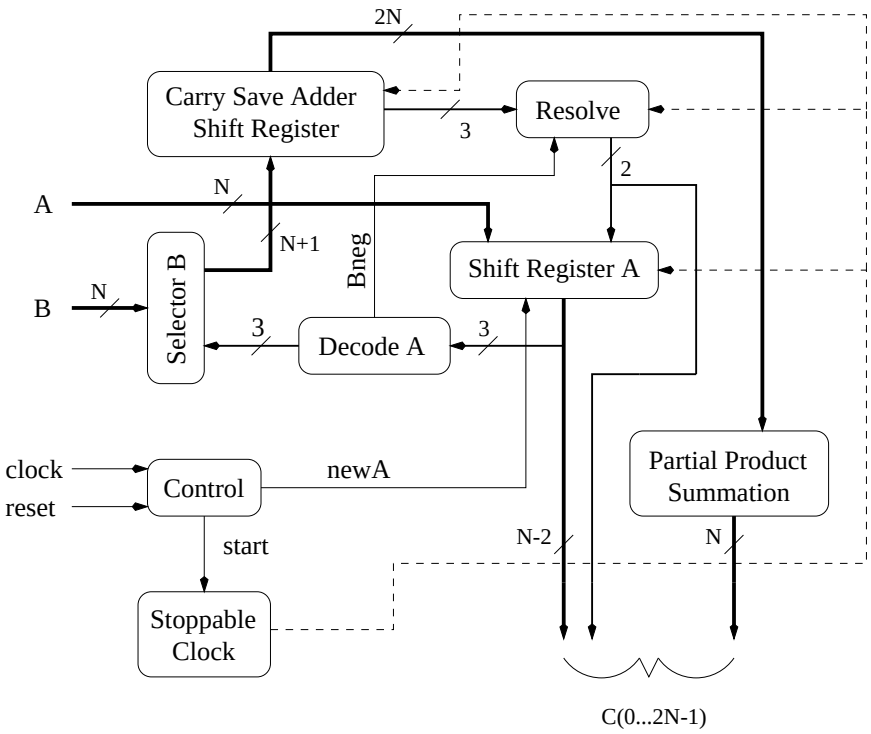

Figure 1. Architecture for the multiplier.

block according to input from *Decode A*. Figure 2 shows how the *Selector B* module works. In this figure, the B operand comes in on the left, is recoded according to *Decode A* signals, and goes out to the *Carry Save Adder Shift Register* on the right. For radix-4 Booth recoding, the *Selector B* block must produce the following multiples of the B operand: B, $-B$, $2B$, and $-2B$. The multiplier uses a *two's complement* representation, so generating the multiples of B is a matter of inverting and shifting as appropriate. In order to complete the *two's complement* conversion for $-B$ and $-2B$, a 1 must be added into the low-order bit of the running sum of the multiply. The portion of the multiplier which handles this carry injection is the *Resolve* circuit.

Because radix-4 Booth recoding retires two bits on each iteration, there are situations when unresolved carries are shifted out of the *Carry Save Adder Shift Register*. These carries must be resolved before the two bits are retired to the final answer. The *Resolve* block takes a shifted out carry bit with two shifted out sum bits and adds them together to resolve the shifted out carry. The architecture for this block is shown in Figure 3. The carry out of this circuit is fed back around to the same circuit to be added in on the next cycle. This circuit generates two final answer bits on each cycle which are passed to *Shift Register A* for storage. As was mentioned, the *Resolve* circuit also injects the *two's complement* carry when the Bneg signal from the *Decode A* block is high.

The *Shift Register A* block, shown in Figure 4, either loads a new value of A when beginning a multiply, or shifts the current value of A by two bits towards the least significant bit. In the figure, the *Shift Register A* is shifting two by bits from left to right. The *Shift Register A* passes its two low order bits along with the last shifted out bit to the *Decode*

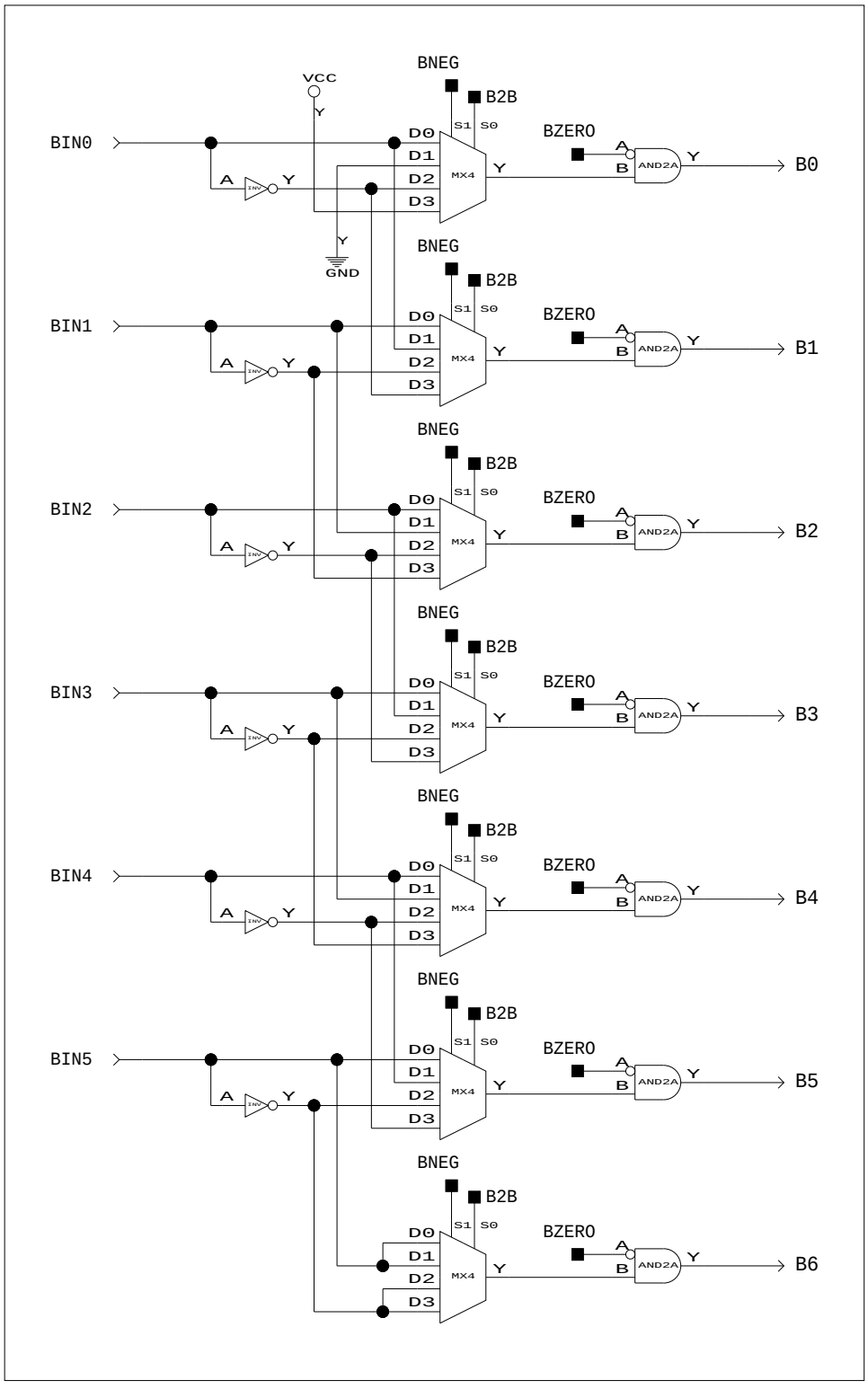

Figure 2. Selector B.

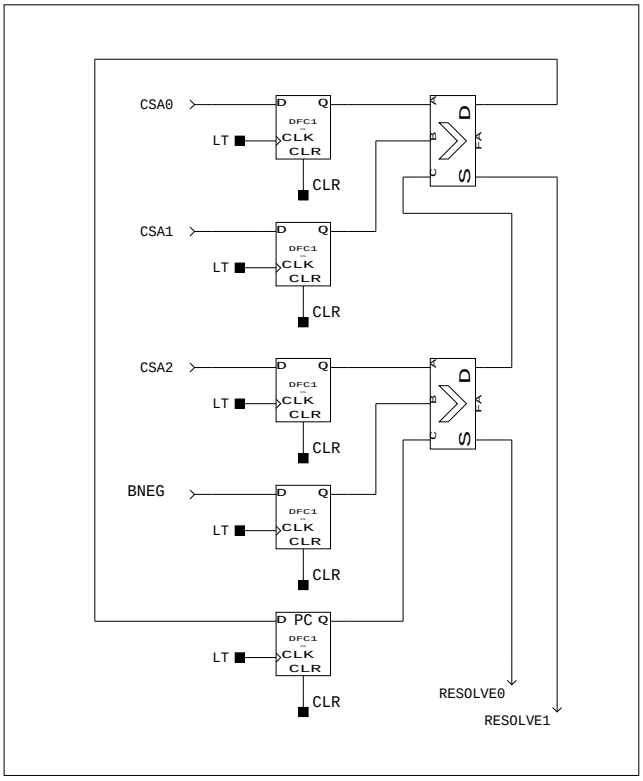

Figure 3. Resolve circuit.

A block. The *Decode A* block uses these three bits to determine the multiple of B to be added to the running sum of the multiply. *Shift Register A* is also used to store the final answer bits retired from the *Resolve* block while iterating. The answer bits are shifted in two at a time on each cycle. When the iterations are completed, *Shift Register A* holds the bottom $N - 2$ bits of the final answer.

The *Carry Save Adder Shift Register* is a row of full adders feeding flip-flops. The interconnection between the full adders and flip-flops can be seen in Figure 5. Rather than having a carry ripple chain, the adders are used in a carry save fashion [6]. This means that carries are passed to flip-flops to be resolved on the next iteration. Therefore each iteration has a time cost that is independent of the width of the operands. Shifting is performed on each iteration by passing outputs of the full adders to flip-flops that are two bits towards the least significant bit. The high order bit starts in the top right of the figure. Bits are passed from right to left, until the chain wraps to the bottom row. Then the bits are passed from left to right.

The *Partial Products Summation* is a basic ripple adder used to resolve all remaining carries when the iterations are done. Since less than one third of the total latency is in the ripple adder, it is not necessary to optimize it.

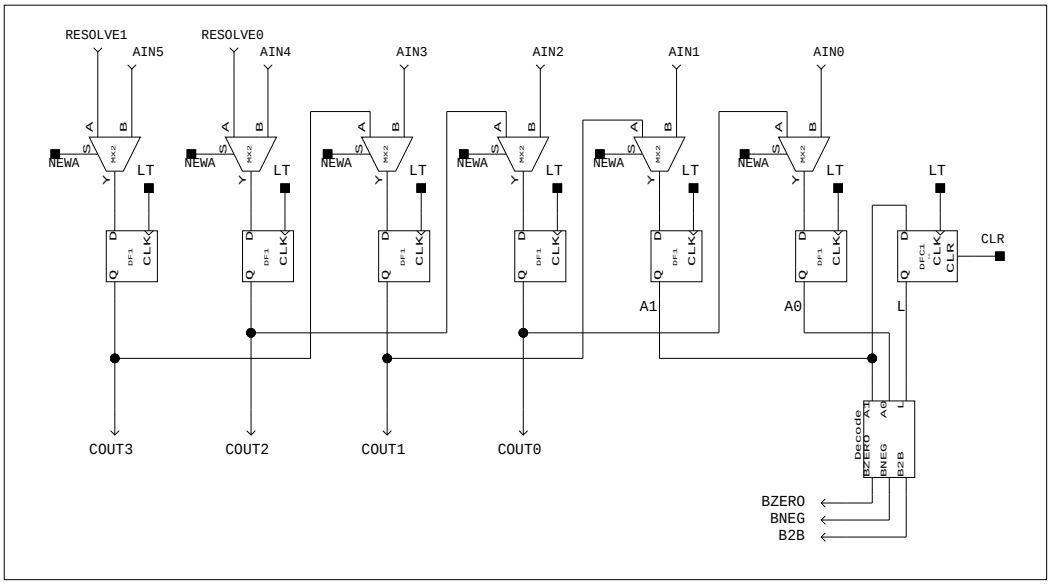

Figure 4. Shift register A and Decode A.

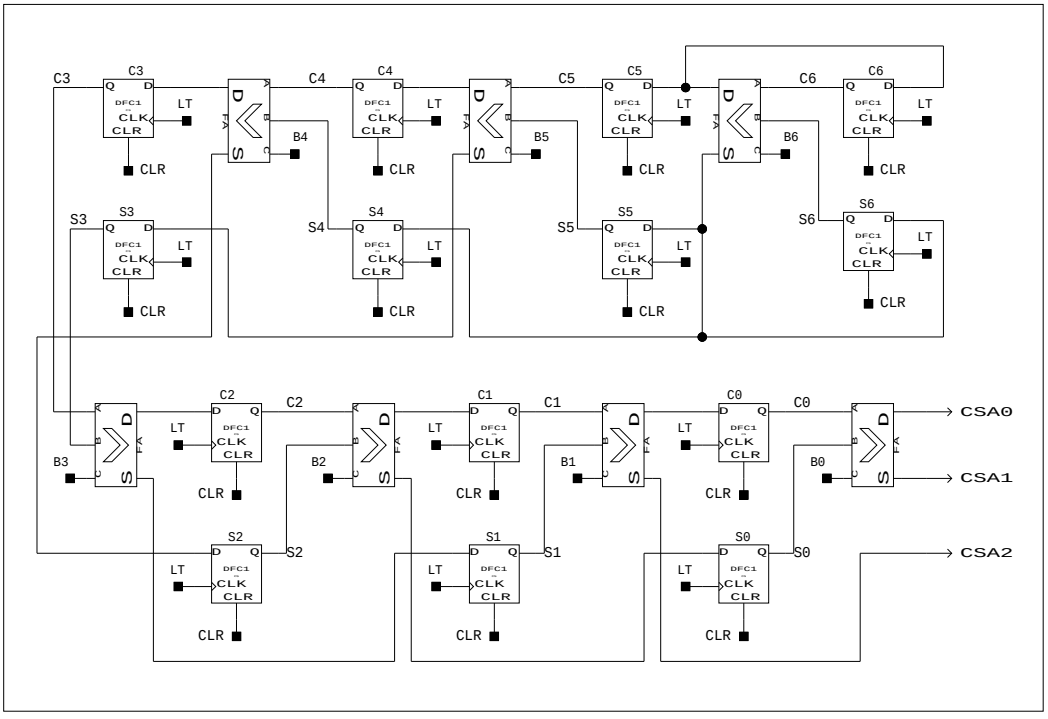

Figure 5. Carry save adder shift register.

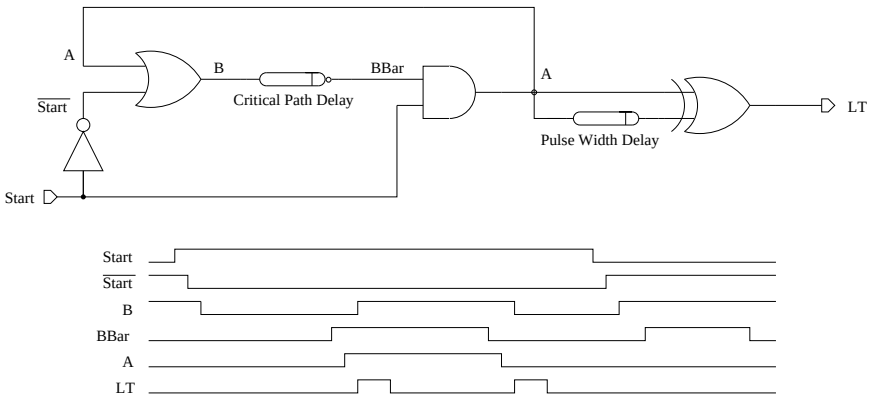

Figure 6. Stoppable clock design.

4: Control and delay

The control circuit consists of a stoppable clock shown in Figure 6 along with a synchronous state machine. One advantage of having a stoppable clock is that it allows for local high frequency computation without the need for global distribution of a high frequency clock. This is of critical importance in the hearing aid application as the energy consumed by a high frequency clock is unacceptable. The clock is made from a ring oscillator, a one-shot, and stop and start logic. The ring oscillator delay is matched to the worst case critical path delay in the self-timed circuit. The core of a ring oscillator is a delay element. A chain of inverters can be used for the delay element; however, inverters do not scale the same as the components on the critical path with process variation, temperature, and voltage swings. For a typical $0.6\mu m$ process with every gate running at the slow corner, it may take only 18 inverters to match the worst case critical path of this design. In that same process, if everything is running at the fast corner, 44 inverters are needed to match the worst case critical path delay. One solution to the mismatch in gate scaling is to make the delay out of gates that match the critical path [8]. The fanout capacitance for each gate is also duplicated to help ensure that the scaling of the ring oscillator matches that of the critical path throughout all conditions. Additional buffers are often added to delay elements to make them conservative. It is possible to make the additional buffer delay a dynamically programmable delay so the period of oscillation can be adjusted after fabrication, and therefore, it does not need to be overly conservative [14].

The state machine starts the ring oscillator by asserting the *Start* signal high. The ring oscillator then continues to create pulses out of the one-shot until the oscillator is cut-off by lowering *Start*. The state machine stops the ring oscillator when $(\frac{N}{2} + 2)$ iterations have been performed. Although the state machine for the control logic is designed using synchronous techniques, it has some timing assumptions that must be guarded to avoid erroneously firing the latch trigger when stopping and starting the clock. These timing assumptions can be verified using `ATACS`, a tool for the synthesis and verification of timed circuits [11], using layout timing information from the gates used in the implementation. Because the *Start* signal is passed to a portion of the stoppable clock that is sensitive to glitches, the output logic for this signal must be hazard-free. This is done by encoding the states of the state machine with a *Gray* code. Thus, at each step of the state machine, a

Table 1. Latency normalized by N=12 combinational fast.

	Self-Timed			Combinational		
N	Fast	Typ	Slow	Fast	Typ	Slow
12	3.36	7.12	20.10	1.00	2.10	5.97
16	4.22	8.95	25.18	1.35	2.85	8.10
20	5.08	10.77	30.26	1.71	3.60	10.20
24	5.95	12.60	35.35	2.06	4.35	12.32
28	6.81	14.42	40.43	2.42	5.09	14.44
32	7.67	16.25	45.52	2.77	5.84	16.56

single bit of the state vector will monotonically change. This enables hazard free output logic implementation.

5: Comparisons

This section compares the self-timed design to a fully parallel combinational multiplier. This is simply a baseline comparison to understand how the self-timed multiplier performs in latency, energy, and area. The only difference in operation of the parallel array and the self-timed multiplier is that the parallel array does multiplications on unsigned positive numbers rather than signed *two's complement* numbers. The array can be used to multiply *two's complement* numbers if the operands are converted to unsigned numbers first and the answer is converted back to a *two's complement* number. These sign-magnitude computations are not included in the comparisons. Table 1 shows the normalized latency of the self-timed design compared to the combinational design. The normalization constant is 45.71ns. This is the latency of the combinational design on the fast corner with $N = 12$. The latency calculation includes capacitive loading on large fanout gates. Figure 7 shows latency in nanoseconds for various sizes of N. The latency for both designs scales linearly with N, but the self-timed design has a larger constant. It is important to note that the latency of both designs can be reduced. If is is known that 1 of the 2 operands requires less dynamic range, then the self-timed and combinational designs can be changed to take advantage of that fact. In the hearing aid application, the filter coefficients are constrained to be in the range of $(1, -1)$. Let N_c represent the the number of bits for the coefficients. If $N_c = 10$ then a radix-4 Booth recoded multiplier only needs to perform $\frac{N_c}{2} = 5$ iterations. At $N = 24$ and $N_c = 10$, the latency at the slow corner of the self-timed design can be reduced to roughly half that shown in Table 1 at $N = 24$. A similar improvement, of course, is also possible in the combinational design. With this optimization, the self-timed multiplier runs at 1.1MHz. An additional optimization for the self-timed multiplier would be to pipeline the critical path. Currently, the Booth recode, B select, and carry save add are all on the critical path. These sections could be pipelined to decrease iteration latency at the expense of adding a few more cycles to fill the pipeline stages. Another approach to improve latency would be to apply more aggressive circuit styles. In the current design, the multiplier is restricted to use only basic standard-cells. However, moving to more aggressive circuit styles would increase design time and complexity. Since one of our goals is simplicity and ease of design, these alternative circuit styles are not considered here.

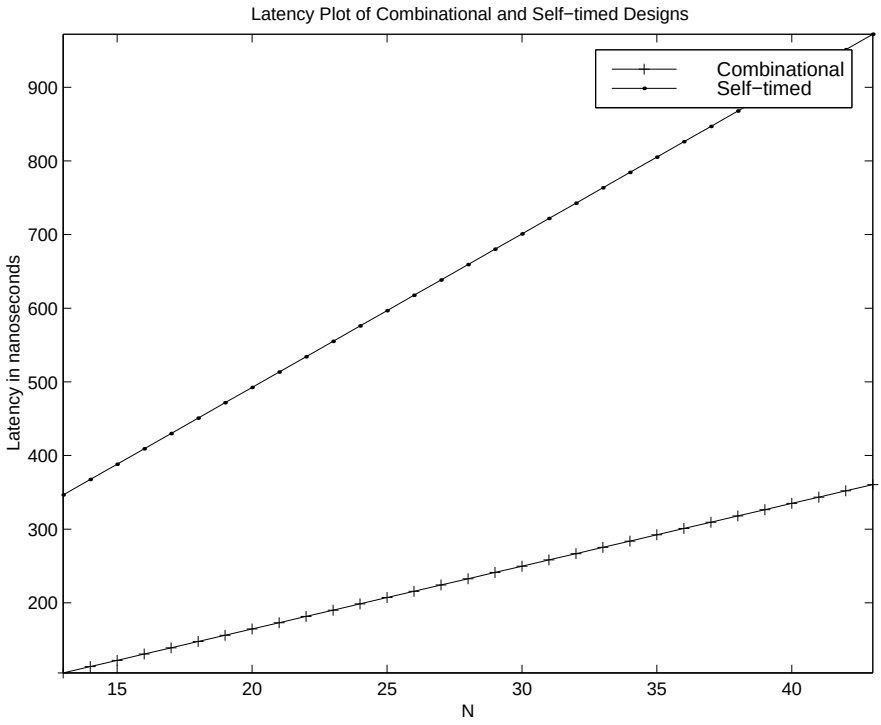

Figure 7. Plot of latency as word size increases.

The energy per operation in this design is compared to that of the synchronous design. Energy is chosen over power because power depends on the average time it takes to perform an operation [16]. If the parallel array is run at maximum speed, its power would be far above that of the self-timed design due to its higher frequency. Therefore, such a comparison is misleading. The energy per operation is independent of the amount of time each design takes to multiply and is akin to running each multiplier at the same global clock frequency. An estimate of energy consumed per operation is obtained using simulation over a large set of random data. Energy is consumed on a node when it switches either from high-to-low or low-to-high. The amount of energy consumed in either case is

$$E = \frac{1}{2}CV^2,$$

where C is the output capacitance on that node and V is the voltage swing of the transition. Energy consumed by an entire design is calculated as

$$E_{\text{total}} = \frac{1}{2}V^2 \sum_i n_i C_i,$$

where i ranges over all nodes in the design. V is the voltage swing that varies depending on the process corner. C_i is the capacitive load seen at node i. As transistor sizing is not considered here, input capacitances on gates are assumed to be equal for all gates. This reduces the capacitive load calculation at node i to $C_i = FO_i C_g$, where FO_i is the fanout of node i and C_g is a constant gate capacitance. The n_i term is the activity factor of node

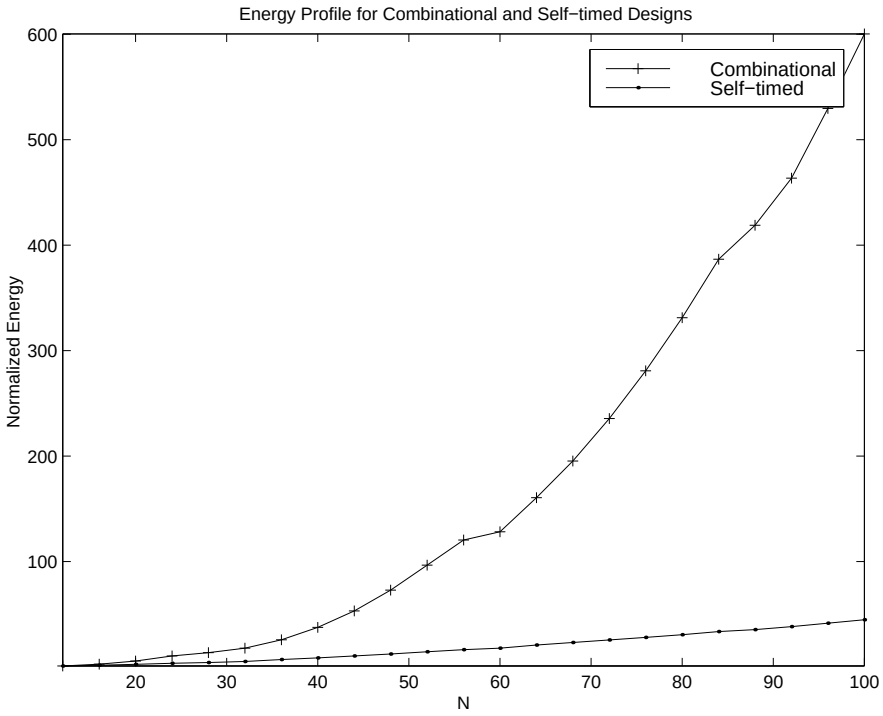

Figure 8. Plot of energy as word size increases.

i. It is the average number of times node i switches (either high-to-low or low-to-high) during each multiply. Simulation of a large set of random multiplies is used to find the n_i term for each node.

A plot of normalized energy estimates for various sizes of N is in Figure 8. Energy has a polynomial growth with word size for both designs, but the coefficient for the self-timed design is much smaller than the combinational design. Table 2 shows normalized energy estimates for various sizes of N at different process corners. The polynomial growth of energy for the self-timed design is most easily seen in the typical corner column of Table 2. The self-timed design has polynomial growth for energy because it has hardware on the order of $O(N)$ which is used on the order of $O(N)$ times. The combinational design has hardware on the order of $O(N^2)$ that switches an average of $O(N)$ times per multiply. It is important to note that the energy estimates assume no correlation between the incoming operands, so the probability of any input bit making a transition is 50 percent. In the hearing aid application, there would be less switching frequency since there is correlation between the filter coefficients and the data samples. Thus, these estimates are conservative.

Figure 9 shows the area plot of the self-timed and combinational multiplier designs. Area is measured in terms of inverter count. The value is calculated by first counting each type of gate and flip-flop in each design. Each individual gate count is multiplied by the number of transistors required to implement the gate or flip-flop, and that number is then divided by 2 to get the total inverter count. The area calculation includes electrical and fanout considerations by conservatively adding buffer trees where appropriate. Not captured in this area calculation is the actual size of the transistors. In addition, for this comparison, the area due to routing is neglected and is assumed to be of equal cost in both designs.

Table 2. Energy normalized by N=12 combinational slow.

	Self-Timed			Combinational		
N	Fast	Typ	Slow	Fast	Typ	Slow
12	1.54	1.24	1.03	1.45	1.19	1.00
16	2.53	2.03	1.67	3.92	3.19	2.65
20	3.74	3.01	2.50	8.47	6.80	5.59
24	5.19	4.18	3.47	15.82	12.55	10.24
28	6.41	5.14	4.25	20.92	16.24	13.04
32	7.85	6.32	5.21	27.97	21.37	16.96

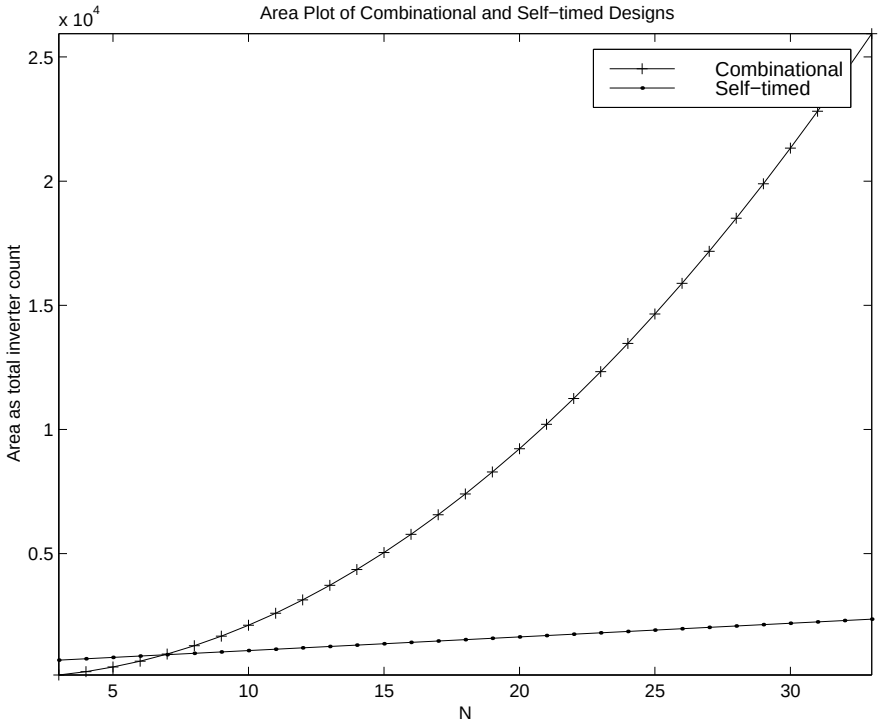

Figure 9. Plot of area as word size increases.

Table 3. Area as number of inverters and normalized.

	Self-Timed		Combinational	
N	Area	Scale	Area	Scale
12	1200	1.00	3132	2.61
16	1420	1.18	5776	4.81
20	1640	1.37	9220	7.68
24	1860	1.55	13464	11.22
28	2080	1.73	18508	15.42
32	2300	1.92	24352	20.29

This benefits the parallel array which clearly has a larger routing area. The two curves in Figure 9 have approximately equal area at $N = 7$. The area growth of the self-timed design is linear in N, where the combinational design follows N^2. Table 3 shows the inverter count for various sizes of N. The scale factor is the normalized area using the area of the self-timed design at $N = 12$ as the normalization constant.

6: Conclusion

This paper details the standard-cell design of a self-timed multiplier. This multiplier can be embedded in current synchronous systems as a combinational unit without any interfacing issues. This paper presents a comparison between the self-timed design and a traditional parallel array design and shows that the area of the self-timed design grows linearly with word size N. This is a marked improvement over the combinational design that has N^2 area growth. Both designs have a polynomial energy growth, but the coefficient for the self-timed design is much smaller than the combinational design yielding a significant energy savings. The self-timed multiplier uses $\frac{1}{3}$ the energy and $\frac{1}{7}$ the area of that of the combinational design at $N = 24$, the target size for the hearing aid application.

As the multiplier is intended to be embedded in a synchronous system, latency cannot be ignored completely. This paper shows the latency of the self-timed design to be linear in N, but with a higher constant than that of the combinational design. To address this shortcoming, this paper suggests two simple optimizations. The iterative portion of the algorithm has complexity $O(N)$. Thus, reducing N improves latency. N can be reduced without compromising dynamic range by considering the hearing aid application the multiplier is intended for. As the coefficients only require $N_c = 10$ bits for a fixed point representation, the speed of the multiplier is increased to 1.1MHz where the data sample has $N = 24$ bits of representation. In addition to this, the critical path can be pipelined to speed up the latency of iterations at the expense of iterating a few more times.

Future work for the self-timed multiplier includes an actual physical implementation optimized for the hearing aid application. This will facilitate the calculation of *measured* latency, energy, and area numbers that include transistor sizing and more detailed electrical considerations. The physical implementation will also allow for characterization of the stoppable clock. If modules with stoppable clocks are to be used, it is necessary to know how reliable stoppable clocks can be. Other work includes the application of more aggressive design styles to optimize the critical path in the multiplier. These design styles do not lend

themselves to standard-cell implementation, but may help to further decrease the latency of the self-timed design.

7: Acknowledgements

We would like to thank Gerald Wilson and Keith Davis from SONIC Innovations for guiding us through their hearing aid application and for pointing us towards this research idea.

References

[1] A. J. Acosta, R. Jiménez, A. Barriga, M. J. Bellido, M. Valencia, and J. L. Huertas. Design and characterisation of a CMOS VLSI self-timed multiplier architecture based on a bit-level pipelined-array structure. *IEE Proceedings, Circuits, Devices and Systems*, 145(4):247–253, August 1998.

[2] A. D. Booth. A signed binary multiplication technique. *Quarterly Journal of Mechanical Applied Mathmatics*, 4(2), 1951.

[3] R. G. Burford, X. Fan, and N. W. Bergmann. An 180 MHz 16 bit multiplier using asynchronous logic design techniques. In *Proc. IEEE Custom Integrated Circuits Conference*, pages 215–218, 1994.

[4] V. Chandramouli, Erik Brunvand, and Kent F. Smith. Self-timed design in GaAs—case study of a high-speed, parallel multiplier. *IEEE Transactions on VLSI Systems*, 4(1):146–149, March 1996.

[5] Jen-Shiun Chiang and Jun-Yao Liao. A novel asynchronous control unit and the application to a pipelined multiplier. In *Proc. International Symposium on Circuits and Systems*, volume 2, pages 169–172, June 1998.

[6] L. Dadda. Some schemes for parallel multipliers. *Alta Frequency*, 34(5):349–356, March 1965.

[7] Jaco Haans, Kees van Berkel, Ad Peeters, and Frits Schalij. Asynchronous multipliers as combinational handshake circuits. In S. Furber and M. Edwards, editors, *Asynchronous Design Methodologies*, volume A-28 of *IFIP Transactions*, pages 149–163. Elsevier Science Publishers, 1993.

[8] David Kearney and Neil W. Bergmann. Bundled data asynchronous multipliers with data dependant computation times. In *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 186–197. IEEE Computer Society Press, April 1997.

[9] C. H. Lau, D. Renshaw, and J. Mavor. A self-timed wavefront array multiplier. In *Proc. International Symposium on Circuits and Systems*, pages 138–141, 1989.

[10] J. B. Lipsher and K. Maheswaran. A 4-bit asynchronous pipelined multiplier in the Xilinx 4000 series FPGA. Technical report, University of California, Davis, 1994.

[11] Chris J. Myers. *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits.* PhD thesis, Dept. of Elec. Eng., Stanford University, October 1995.

[12] Christian D. Nielsen and Alain J. Martin. Design of a delay-insensitive multiply-accumulate unit. *Integration, the VLSI journal*, 15(3):291–311, October 1993.

[13] O. Salomon and H. Klar. Self-timed fully pipelined multipliers. In S. Furber and M. Edwards, editors, *Asynchronous Design Methodologies*, volume A-28 of *IFIP Transactions*, pages 45–55. Elsevier Science Publishers, 1993.

[14] Mark Santoro and Mark A. Horowitz. SPIM: A pipelined 64x64-bit iterative multiplier. *IEEE Journal of Solid-State Circuits*, 24(2):487–493, April 1989.

[15] J. Sparsø, C. D. Nielsen, L. S. Nielsen, and J. Staunstrup. Design of self-timed multipliers: A comparison. In S. Furber and M. Edwards, editors, *Asynchronous Design Methodologies*, volume A-28 of *IFIP Transactions*, pages 165–179. Elsevier Science Publishers, 1993.

[16] José A. Tierno and Alain J. Martin. Low-energy asynchronous memory design. In *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 176–185, November 1994.

[17] Kenneth Y. Yun, Peter A. Beerel, Vida Vakilotojar, Ayoob E. Dooply, and Julio Arceo. The design and verification of a high-performance low-control-overhead asynchronous differential equation solver. In *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 140–153. IEEE Computer Society Press, April 1997.