Self-Timed Design in GaAs—Case Study of a High-Speed, Parallel Multiplier

V. Chandramouli, Erik Brunvand, and Kent F. Smith

Abstract—**The problems with synchronous designs at high clock frequencies have been well documented. This makes an asynchronous approach attractive for high speed technologies like GaAs. We investigate the issues involved by describing the design of a parallel multiplier that can be part of a floating point multiplier. We first present a new architecture called the partial array of array (PAA) that is more regular than a partial tree approach while having the same latency. We then show how this architecture can be used in a self-timed implementation in the style of micropipelines. We next describe how we can design the final carry propagate adder using a new precharged logic family in GaAs that we developed as part of this project. We conclude with some general observations on doing asynchronous design in GaAs.**

Index Terms— **Self-timed systems, micropipelines, multipliers, GaAs, precharged circuits.**

I. Introduction

The speed of digital systems has increased dramatically over the past few years and this trend is likely to continue for long. However, increases in speed are also accompanied by problems that are unique to high-speed systems which often warrant changes in the design style. The most popular design technique is the synchronous approach, where a global timing signal coordinates the movement of data in the system. Problems with this technique at high clock frequencies are well-documented [1]–[4]. Almost all of the problems are associated with distributing a global clock signal(s). Self-timed designs, on the other hand, avoid these problems by doing away with a global clock signal. In addition, they offer other adavantages [1]–[3] as well, such as composability, incremental improvement, possible lower power, and an average case performance where applicable.

Self-timed designs are particularly attractive in GaAs for the following reasons. GaAs offers the potential for very high clock frequencies due to higher electron mobility and very small gate capacitances. However, as mentioned above, this makes global clock distribution difficult. In addition, the popular logic families in GaAs (such as direct coupled FET logic (DCFL) [5]) consume a lot of power and have poor noise margins. Thus, one could very well do without large clock buffers and additional sources of noise like spikes on power buses (caused when clock signals make transistions). Moreover, GaAs is a logical candidate for computation intensive applications, where arithmetic circuits are the major building blocks and highest speeds are desired. It is very likely that there will be a considerable difference between the average and worst-case times in these circuits (consider, for example, an n-bit addition) and self-timed systems, on average, can outperform synchronous designs. Since the integration levels are not as high in GaAs as CMOS, it is likely that complex systems will be implemented as MCMs/PCB's which exacerbate the problems of global clock distribution and make the self-timed approach an attractive option for GaAs.

From the preceding paragraphs, it is clear that there exists a synnergism between GaAs as the technology base and self-timed design as the architectural paradigm. However, other than the Caltech project [6] we have not found any large self-timed systems in GaAs in the literature. Therefore, we decided to explore the issues involved by designing a self-timed integer multiplier that can be part of a floating point multiplier. Parallel multipliers are large and complex but on the other hand can be easily implemented using few basic building blocks in a regular way. Besides, several applications like DSP, Graphics, etc. require fast area efficient multipliers.

The organization of this paper is as follows. In Section II, we introduce a new multiplier architecture called the Partial Array of Array. In Section III, the design details of the multiplier array are presented. In Section IV, we describe the design of the final addition of the partial products. In Section V, we make some general observations on doing self-timed design in GaAs and finally, we conclude the paper in Section VI.

II. Partial Array of Array: A New Architecture

Throughout this paper, we will be concerned with the IEEE single precision format for floating point numbers [7]. In floating point multiply, the multiplication of the mantissas is the critical step (24 b in this case). Any of the integer multiplication architectures can be used for this step. It is well known that a tree based architecture provides a very low latency but occupies a large area. See [8] for the area latency tradeoffs for several tree-based architectures. Array-based architectures, while being very compact, have a large latency. Several hybrid schemes have been proposed in the literature that have area and latency requirements in between the two extremes.

One such hybrid scheme has been proposed recently in [9]. This is an array of array based scheme and has area requirements close to that of an array multiplier and a latency of $O(\sqrt{N})$, for an $n \times n$ multiplier. Refer to Fig. 1(a) for a general layout of the architecture. It consists of a number of small array type submultipliers. Suppose we are multiplying $n \times m$ numbers. Then the m-bits of the multiplier are partitioned into $k+1$ subarrays of sizes $l_0, l_1, \cdots, l_k$ such that $\Sigma l_i = m$. All the sub-arrays work in parallel thus making efiicient use of hardware. Fig. 1(b) shows the scheme for an 8×8 multiplier. For the 24 $\times$ 24 case, which we are interested in, the array of array design would occupy an area a little larger than the regular array multiplier but the latency is improved to nine carry save adder (CSA or (3, 2) counter) delays which is more than twice the speed up compared to the array case. As compared with the tree based approach, which gives the fastest possible speed of 7 CSA delays at an area that is about three times as large, this scheme is highly competitive. Similar to Santoro's [10] modification of a tree based approach, we modified the array of array to an iterative, pipelined architecture. We can reduce eight partial products per iteration and iterate three times to get the full 48 b product. The resultant architecture called the partial array of array (PAA) is shown in Fig. 2. This has a latency of $(2+2+2)+2+2 = 10$ CSA delays. As compared to the full AA based approach, this occupies only one-third the area for a small penalty in latency. Each pipe stage has two CSA delays (ignoring latch delays) leading to a balanced pipeline.

We did an area-latency analysis of the various multiplier architectures [11] and found that both the partial tree and the PAA schemes are competing schemes. However, we chose the PAA scheme because it offered more regularity in the routing of the multiplier and multiplicand than the tree based approach. Moreover, it can be easily extended to handle Booth encoding for future extentions.

Manuscript received September 13, 1994; revised May 24, 1995.

V. Chandramouli is with the Advanced Computer Architecture Laboratory, the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA.

E. Brunvand and K. F. Smith are with the Department of Computer Science, University of Utah, Salt Lake City, UT 84112 USA.

Publisher Item Identifier S 1063-8210(96)01875-6.

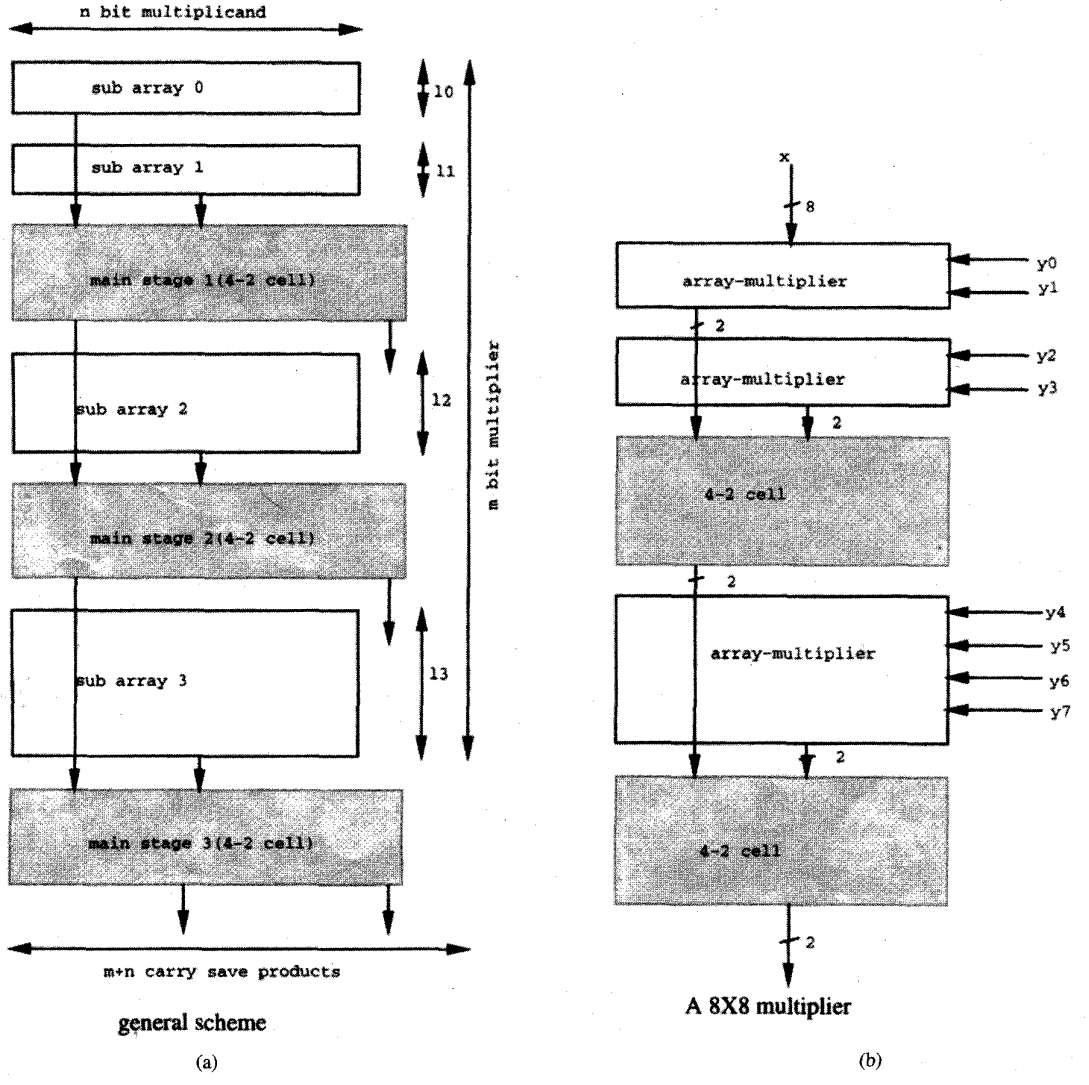

Fig. 1. An array of array scheme. (a) General scheme (b). An 8 × 8 multiplier.

III. Design of the Multiplier Array

We implemented the PAA architecture in the style of Micropipelines [3] using a bundled datapath[1] as shown in Fig. 3.

Pipelining is a natural choice for iterative structures like the PAA. While one could argue that the bundled datapath approach always entails waiting for the worst case where as a dual rail datapath could be done in a delay insensitive manner giving an average case performance, in this case it was not clear if the average case time was significantly better to warrant the extra investment in routing resources, entering new dual-rail logic cells into our CAD tool library etc. Moreover, since each pipestage has similar delays, the design of control for each stage is simpler. GaAs DCFL was chosen for circuit design. A restriction of using DCFL is that we are limited to using only NOR gates. Though it is functionally complete, this constrains the circuit design.

In order to minimize delays through latches, all latches were implemented as Earle latches by merging the latch with the logic. Thus, new custom laid out (3, 2) and (4, 2) [12] cells were added to the cell library, with and without latches. We also needed to modify the latch designs so that they can be used in a transition signaling environment. The modifications were similar to the ones in the asyncronous implementation of the ARM chip [13].

As can be seen in Fig. 3, we have used variable delay elements in our design. The use of variable delay lines gives us the flexibility of modifying the delays so that bundling constraints are met, once the design is fabricated. This is similar to adjusting the clock frequency in synchronous systems to meet timing constraints. An 8 × 24 (actually 8 × 8 slice, that can do 8 × 24 using three iterations) bit slice was laid out. It occupied an area of 0.7 sq. mm.

The whole circuit was extracted from the layout and simulated using spice. Buffers were added on the latch control lines since they have to drive across the whole array. It was expected that these lines would be buffered every eight stages if one were to build a 24 × 8 array. The capacitance values were suitably modified to mimic a 24 wide array. Based on these simulations (done at 25°C under typical-typical (tt) process corner), it was found that we could initiate a new multiply every 13 ns giving us a throughput of 76 Million Floating

[1] In this design style, both the data and the ready signal are "bundled" together. Care must be taken (by inserting additional delays if necessary) that the ready signal is not asserted before the data is ready.

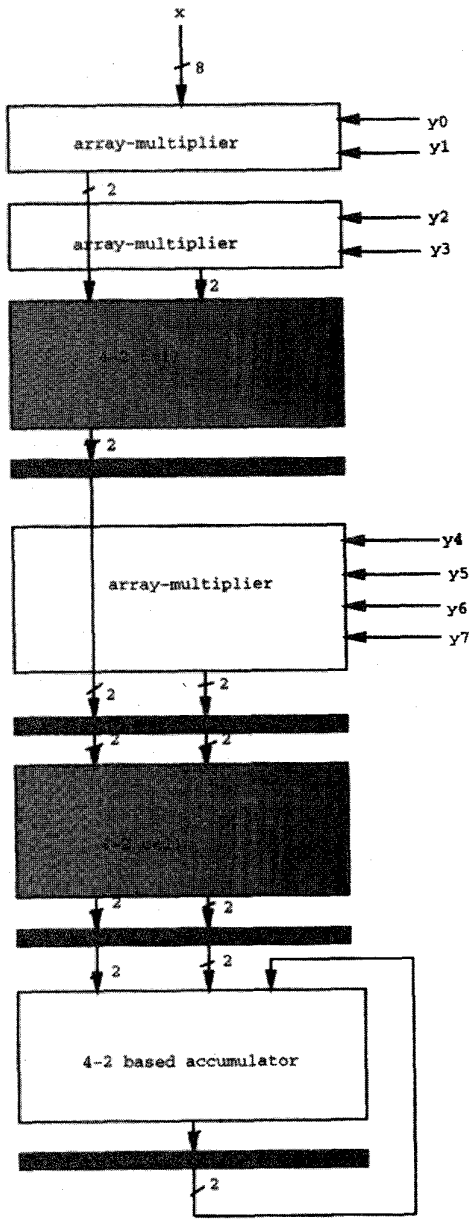

Fig. 2. A pipelined PAA scheme.

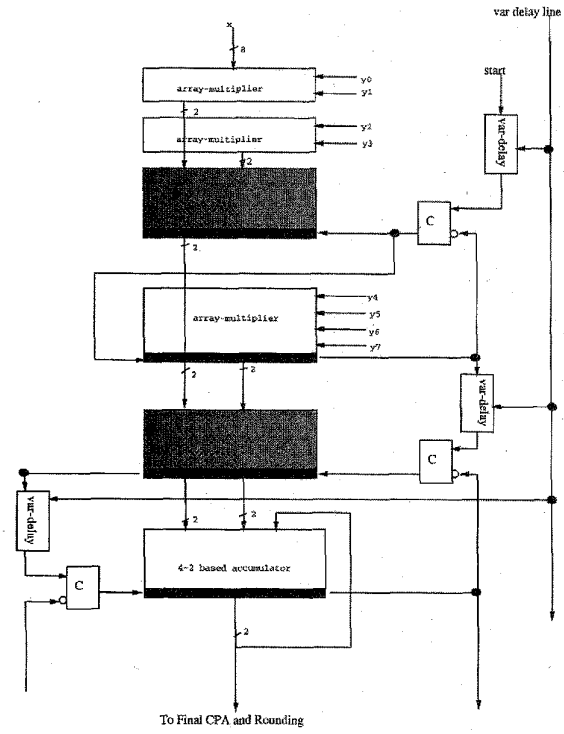

Fig. 3. A self-timed PAA.

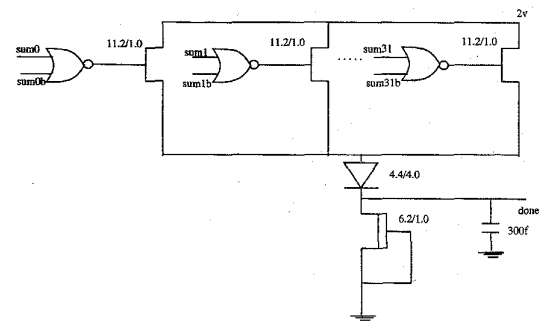

Fig. 4. "Done" generation.

Point Multiplies Per Second (MFLOMPS). The latency for a single multiply was measured to be 17 ns.

IV. Summing the Partial Products and Rounding of the Result

We need a fast carry propagate addition (CPA) stage where the partial products, which are in the carry save form, are converted back to the binary representation. At the same time, this stage should not be very area intensive. As a compromise, we decided to implement this stage using the carry completion sensing (CCS) [14] adder which uses delay-insensitive signalling. We can achieve an average case performance of $O(\log n)$ by using the CCS adder, with an area of a little more than that required by the carry-ripple adder.

However, there was no satisfactory way of building DCFL compatible delay-insensitive circuits. In order to address this problem, we developed a new dual rail logic family [15] that bears a superficial similarity to CMOS CVSL [16]. We designed precharged full adders using our new logic family. A comparison between the precharged full adder and an equivalent DCFL adder was made by doing a composite layout, extraction from the layout, and spice simulation of each circuit. The simulations used the parameters obtained from the Vitesse GaAsIII 1.0μ process [17]. These simulations showed that adders designed using our scheme consumed about 73% less power than the DCFL case, for comparable delays.

We can use the rounding algorithm developed in [10] along with the final CPA. The circuits implementing this algorithm can be implemented easily in GaAs DCFL with some modifications [11]. However, we need a fast, area efficient way of doing a carry propagate addition of the higher order 25 bits of the partial product. We designed a precharged carry skip CCS adder rather than a simple ripple CCS adder, to improve the speed. We also designed a novel circuit for detecting when the addition was completed which is shown in Fig. 4. The "sum" and "sumb" signals are generated from each bit of the adder.

TABLE I
A COMPARISON OF TWO MULTIPLIERS

Characteristics	AT&t/Sandia	proposed multiplier
Function	IEEE single prec.	same (only 1 rounding mode)
Fab. Process	HFET 1.0μ m	Vitesse 1.0μ m
Architecture	(4,2)based full tree	Partial Array of Array
Clocking scheme	Synchronous	Asynchronous
Logic family	SFFL	DCFL
Latency	9.25ns	24ns
Throughput	NA	13ns (76MFLOMPS)
Area	76 sq.mm	$\leq$25 sq.mm
Power	6.22W	3-3.5W

Table I shows the comparison between the proposed multiplier and an existing one, fabricated by AT&T/Sandia. The area and the power estimates for the 24×24 case were obtained by extrapolating the figures obtained from the 8×24 case and the characterized values of our CAD cell set. The area advantages of our approach is obvious. The difference in speed is both technological and architectural. The existing multiplier uses an experimental heterostructure process with a different logic family (SFFL) that is faster and more consumptive in power than DCFL. On the architectural front, their design uses a full tree approach which is known to have the smallest latency at a considerable investment in area whereas ours is an iterative array based design. Finally, we must point out that we did not have a sophisticated CAD tool set and instead had to rely on our own internally developed CAD tool.

V. OBSERVATIONS ON ASYNCHRONOUS DESIGN IN GaAs

We make the following observations based on our experience designing the multiplier.

- By using DCFL, we are constrained to using only NOR gates. This means three gate delays for important circuits like XOR (in the worst case) which are widely used in the control path of asynchronous cicrcuits. The added fan in and fanout restrictions makes circuit design harder.
- In two phase transition signalling, it is only the transition that is important and not the level. The nonavailability of the p device added circuit complexity to ensure uniform behavior when the signal is low.
- The asymmetrical rise and fall times in GaAs DCFL, which is characteristic of a ratioed logic family, means the bundling constraints must be met for both rising and falling edge of the signal.
- In spite of these problems, DCFL offers superior power-delay product than most other logic families and is widely used. More research needs to be done exploring other logic families for asynchronous designs.
- We have not come across a suitable logic family that can be used for building dual rail datapath and 4-phase signaling within the 0–2 V framework. We have proposed a new logic family to overcome this and the initial results have been very encouraging.
- The fact that each pipe stage had uniform delays and the use of variable delay lines ensured that not much time was lost in implementing system timing. Further, the use of a CCS adder using precharged circuits ensured that that the final CPA worked in an average sense. In a synchronous design, on the other hand, significant analysis would be required to ensure no skews and avoid the high frequency clock distribution problems. To make the final CPA fast, more complex schemes like carry lookahead would be used, increasing the area. Further, since each pipe stage in the array has a very small delay in comparison to the CPA unit, system timing would be more complicated to achieve maximum speed. In an asynchronous design, each unit can operate at its maximum speed.

VI. CONCLUSION

In this paper we investigated self-timed design in GaAs and described the design of a floating point multiplier. A new area efficient architecture called the partial array of array was presented. A new family of precharged circuits for doing delay insensitive asynchronous designs in GaAs was also mentioned. A test chip containing a precharged adder was fabricated and found to be functional. Based on spice simulations, we expect our multiplier to have a latency of 24 ns and a throughput of 76 MFLOMPS with a power dissipation of 3–3.5 W. The area is expected to be at most 25 sq. mm. Finally, we made some observations on doing asynchronous design in GaAs.

Our research thus far raises a number of interesting problems. Future projects in this direction could be to build the full multiplier as outlined here and test the chip, use Booth recoding with the PAA scheme for a IEEE double precision implementation and compare with commercial multipliers, do a synchronous design in GaAs and compare the issues, implement our design in submicron CMOS/BiCMOS and compare the differences beween an asynchronous MOS and GaAs implementations and finally use our precharged circuits to build complex delay insensitive circuits. We also did not address any testability issues here. An interesting problem would be to investigate the testability of an asynchronous multiplier.

ACKNOWLEDGMENT

The authors would like to thank N. Michell and G. Gopalakrishnan for their helpful discussions and pointers to the literature.

REFERENCES

[1] E. Brunwand, "A cell set for self-timed design using actel FPGA's," Dep. Comput. Sci., Univ. Utah, Tech. Rep. UUCS-91-013, Aug. 1991.
[2] C. L. Seitz, "System timing," in *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980, ch. 7.
[3] I. E. Sutherland, "Micropipelines," *Commun. ACM*, June 1989.
[4] V. Akella, "An integrated framework for high-level synthesis of self-timed circuits," Ph.D. dissertation, Dep. Comput. Sci., Univ. Utah, Dec. 1992.
[5] S. I. Long and S. E. Butner, *Gallium Arsenide Digital Integrated Circuit Design*. New York: McGraw-Hill, 1990.
[6] A. J. Martin *et al.*, "Asynchronous circuits in gallium arsenide," Dep. Comput. Sci., California Inst. Technol., Tech. Rep. Caltech-CS-TR-91-10, Nov. 1991.
[7] IEEE, *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std. 654-1985, 1985.
[8] P. J. Song and G. D. Michelli, "Circuit and architecture tradeoffs for high-speed multiplication," *IEEE J. Solid-State Circuits*, vol. 26, pp. 1184–1198, Sept. 1991.
[9] G. J. Hekstra and R. Nouta, "A fast multiplier architecture," in *Int. Symp. Circuits Syst.*, 1992.
[10] M. R. Santoro, "Design and clocking of VLSI multipliers," Ph.D. dissertation, Comput. Syst. Lab., Standford Univ., Oct. 1989.
[11] V. Chandramouli, "Design of a self-timed, pipelined, floating point multiplier in gallium arsenide," Master's thesis, Dep. Comput. Sci., Univ. Utah, Mar. 1994.
[12] D. T. Shen and A. Weinberger, "4-2 carry save adder implementation using send circuits," *IBM Tech. Disclosure Bulletin*, vol. 20, no. 9, Feb. 1978.
[13] D. Edwards, "A micropipelined microprocessor," private communication, 1993.
[14] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design*. New York: Wiley, 1979.
[15] V. Chandramouli, N. Michell, and K. Smith, "A new, precharged, low-power logic family for GaAs circuits," *IEEE J. Solid-State Circuits*, vol. 30, pp. 140–144, Feb. 1995.
[16] L. G. Heller *et al.*, "Cascode voltage switch logic: A differential CMOS logic family," in *IEEE Int. Solid-State Circuits Conf.*, Feb. 1984.
[17] G. Lee *et al.*, "A high density GaAs gate array architecture," *IEEE Custom Integr. Circuits Conf.*, 1991, pp. 14.7.1–14.7.4.