Accumulative Parallel Counters

Behrooz Parhami and Chi-Hsiang Yeh

Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106-9560, USA

Abstract

An accumulative parallel counter represents a true generalization of a sequential counter in that it incorporates the memory feature of an ordinary counter; i.e., it adds the sum of its n binary inputs to a stored value. We examine the design of accumulative parallel counters and show that direct synthesis of such a counter, as opposed to building it from a combinational parallel counter and a fast adder, leads to significant reduction in complexity and delay. While the mere fact that savings can be achieved comes as no surprise to seasoned arithmetic designers, its extent and consequences in designing large-scale (systolic) associative processors, modular multi-operand adders, serial-parallel multipliers, and digital neural networks merits detailed examination. Both simple accumulative parallel counters and their modular versions, that keep the accumulated count modulo an arbitrary constant p, are dealt with.

1. Introduction

A parallel counter [SWAR73] has been defined as a combinational digital logic circuit having n binary inputs and $m = \lfloor \log_2 n \rfloor + 1$ binary outputs, or alternatively, having between 2^{m-1} and $2^m - 1$ binary inputs for m binary outputs, where the outputs correspond to the base-2 representation of the sum of the n input bits $x_1, x_2, \ldots, x_n$; i.e., a number between 0 and n. Such parallel counters have been studied extensively in connection with counting of multiple responders in associative devices and finding the sum of a column of 1s in fast parallel multipliers, with numerous implementation alternatives investigated.

However, the above definition does not represent a true generalization of serial counters in the sense that the memory feature of serial counters is not carried over. We define an accumulative parallel counter (APC) as a sequential circuit with a single q-bit word of memory holding, at the end of each counting cycle, the sum of its previous content and its n single-bit inputs. Thus, in each cycle, the stored or output count will be incremented by a value between 0 and n. The sum can be defined to be modulo 2^q or modulo some other arbitrary number p, with or without a wraparound (overflow) indication output.

An APC can of course be designed by combining an ordinary combinational parallel counter (CPC) with a fast adder. To design an n-input q-output modulo-p APC, we need an n-input CPC (actually $n - 1$ inputs will do) and a fast modulo-p carry-propagate adder with inputs of length $\max(q, \lfloor \log_2 n \rfloor + 1)$. If modulo-$p$ operation is desired for p not equal to 2^q or $2^q - 1$, then the fast adder must be followed by a modulo-p reduction circuit. However, following the idea of merged arithmetic [SWAR80], a much more efficient implementation that combines the two or three needed operations is possible.

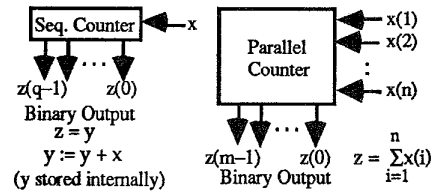

Fig. 1. Sequential and parallel counters.

We discuss accumulative modulo-p parallel counters, concentrating initially on providing the accumulative property for the simpler modulo-2^q or modulo-$(2^q - 1)$ parallel counters. We demonstrate that such APCs are only slightly more complex than their combinational versions. The optimal design and added complexity is dependent on the relationships between various parameters (e.g., on whether $\log_2 n$ is much smaller than or comparable to q or $\log_2 p$) and on the level of tolerance to additional delay in reading out the correct accumulated count. Design options are presented and compared with regard to speed and cost in each case.

The results are then extended to the case of an arbitrary modulus p. We show that by adopting a delayed reduction policy, whereby reading out the correct (modulo-p) count may involve a small additional delay, such counters can also be implemented with negligible overhead. Designs are presented and evaluated for a wide range of the parameters q, n, and p. All of our designs compare favorably with n-input, ordinary and modulo-p CPCs and are considerably more efficient than those obtained by incorporating the accumulative property after the fact.

We conclude by discussing the implications of our results in several application areas and pointing to possible directions for future research.

1058-6393/96 $5.00 © 1996 IEEE
Proceedings of ASILOMAR-29

2. Simple Parallel Incrementers

In this section, we discuss the design of (n, q) parallel incrementers with n single-bit inputs $x_1, x_2, \ldots, x_n$ (increment signals), a q-bit binary input $y_{q-1} \ldots y_1 y_0$, and a q-bit binary output $z_{q-1} \ldots z_1 z_0$ which is the sum of y and the n single-bit inputs modulo 2^q or $2^q - 1$. Since mod-(2^q-1) addition can be performed by using end-around carries in intermediate computation steps with no added complexity, the modular reduction aspect of the design will be implicit in our discussion. Clearly, an APC is nothing but a parallel incrementer suitably connected to a q-bit register. Since this q-bit register is the same in all designs, we can safely ignore it and concentrate primarily on the design of parallel incrementers.

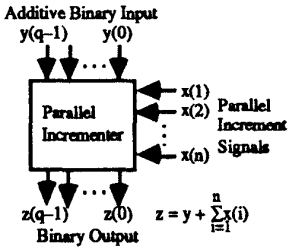

Fig. 2. An (n, q) parallel incrementer.

A parallel incrementer can of course be designed by combining an ordinary CPC with a fast adder. The delay and cost of such an (n, q) parallel incrementer are increased by the delay and cost of the q-bit fast adder, compared to the respective parameters of an $(n - 1)$-input CPC (one of the n increment signals can be accommodated by using the carry-in input of the fast adder). Overhead of such designs is quite high for moderate q. Direct synthesis of a parallel incrementer leads to more efficient designs.

We present a synthesis procedure based on full and half adders (FAs, HAs) when q is not much larger than $\log_2 n$. An (n, q) parallel incrementer of this type consists of an $(n - 1)$-input CPC and a q-bit ripple-carry adder, as shown for a $(16, q)$ parallel incrementer in Fig. 3. The CPC part of the design is based on a divide-and-conquer strategy: given two CPCs each with l inputs, a $(2l + 1)$-input CPC is obtained by the use of a ripple-carry adder of length $\lfloor \log_2 l \rfloor + 1$. Similarly, two such $(2l + 1)$-input CPCs can be combined with a $(\lfloor \log_2 l \rfloor + 2)$-bit ripple-carry adder to produce a $(4l + 3)$-input CPC. This procedure is repeated until a CPC of size $\geq n - 1$ is obtained.

The delay and cost of an $(n - 1)$-input parallel incrementer of this type (in terms of FA/HA levels and their multiplicity) can be easily derived. The CPC part has delay and cost [SWAR73]:

$$D_{CPC}(n - 1) = 2\lfloor \log_2(n - 1) \rfloor - 1$$
$$C_{CPC}(n - 1) \approx n - 1$$

$C_{CPC}(n - 1)$ is actually between $n - \log_2 n - 1$ and $n - 1$, but we take it to be $n - 1$ for simplicity. Since the delay in the computation of the least significant CPC output bit is $\lfloor \log_2(n - 1) \rfloor$ and the time required for the carry to propagate in the q-bit ripple-carry adder is q, the delay and cost of an (n, q) parallel incrementer with simple carry ripple in the final stage are given by:

$$D_{PIR}(n, q) \leq \lfloor \log_2(n - 1) \rfloor + q$$
$$C_{PIR}(n, q) \leq n + q - 1$$

Thus, if q is not much larger than $\log_2 n$, as assumed, the delay overhead of parallel incrementers constructed in this manner is considerably reduced compared with those of the naive design. More importantly, these advantages are achieved with a simpler, more regular design.

For example, when n is a power of 2 (as is usually the case in practice), the delay of an (n, q) parallel incrementer is at most only $q - \log_2 n$ FA levels more than that of an n-input CPC and its cost is about the same because we can take advantage of unused FA inputs in the CPC to enter the bits of y. In particular, for $q = \log_2 n$, no speed penalty is incurred. In the worst case, when $n - 1$ is a power of 2, the cost increases by q FAs.

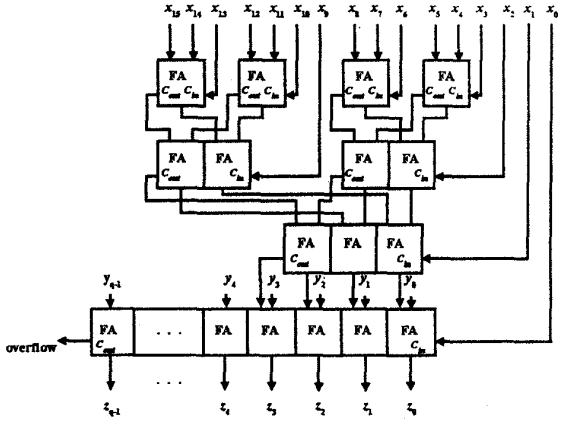

Fig. 3. Design of a $(16, q)$ parallel incrementer.

When q is large and high speed is desired, computation of the sum, except for its least-significant $\lfloor \log_2(n - 1) \rfloor + 1$ bits, can be accelerated using standard speedup techniques (carry-select, carry-skip, etc.), alone or in combinations. Use of carry-select is particularly attractive in terms of both speed and cost. The high-order $q - \lfloor \log_2(n - 1) \rfloor - 1$ bits of y are input to an incrementer, with a multiplexer used to select the $(q - \lfloor \log_2(n - 1) \rfloor)$-bit incrementer result (including overflow indication) or corresponding bits of y, based on the carry produced by the position indexed $\lfloor \log_2(n - 1) \rfloor$, for output. If the delay of the incrementer, which operates concurrently with the CPC and the low-order bits of the final ripple-carry adder, does not exceed $2\lfloor \log_2(n - 1) \rfloor + 1$, then we have:

$$D_{PIS}(n, q) \leq 2\lfloor \log_2(n-1) \rfloor + 2$$
$$C_{PIS}(n, q) \leq n + (1+\alpha)q - 1$$

In deriving the above, we have used the assumption that a multiplexer delay is ≤ 1 FA delay. The parameter $\alpha << 1$ is the product of two factors: the ratio (cost of a 1-bit 2-way mux)/(cost of FA) and $1 - \lfloor \log_2(n-1) \rfloor / q$, the latter factor accounting for the fact that the number of multiplexers is less than q.

The carry-select approach is faster than the ripple-carry scheme for $q > \lfloor \log_2(n-1) \rfloor + 2$. Even a ripple-carry incrementer used with the carry-select approach provides this speed advantage for q up to $3\lfloor \log_2(n-1) \rfloor + 2$; viz, $6 \leq q \leq 11$ with $n = 16$, or $13 \leq q \leq 32$ with $n = 2048$.

For even larger values of q, our design options include use of carry-skip to speed up the incrementer (thus allowing its length to grow even further with the same single-level carry-select structure), use of 2-level carry-select, or a combination of these methods. It is doubtful that more than 2 levels of carry-select will ever be needed in practice. A carry-skip structure which is allowed a total delay of $2\lfloor \log_2(n-1) \rfloor + 1$ units can lead to designs with q as large as $(\log_2 n)^2$ bits with no increase in delay and minimal effect on cost.

3. Modular Parallel Incrementers

In this section, we extend our results for simple parallel incrementers to the case of modulo-p parallel incrementers, where the modulus p satisfies $2^{q-1} < p \leq 2^q$. An $(n, q; p)$ modular parallel incrementer computes the q-bit modulo-p sum of the n single-bit increment signals and a q-bit binary input. An $(n, q; p)$ modular parallel incrementer can be realized by using an (n, q) parallel incrementer followed by a modulo-p reduction circuit [PARH94]. However, as before, a merged design significantly reduces the delay and cost overheads.

Assuming $q \geq \log_2 n$, the modular reduction is essentially equivalent to detecting if the sum has exceeded $p - 1$ and to subtract p from it (add $2^q - p$ to it) if it has. Overflow is detected by observing the carry-out.

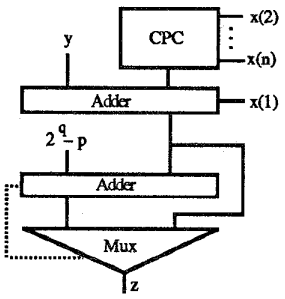

Fig. 4. Modulo-p parallel incrementer.

A straightforward implementation of a modular parallel incrementer consists of an $(n-1)$-input CPC followed by two q-bit ripple-carry adders and a q-bit multiplexer: the first q-bit adder adds the output of the CPC and a single-bit increment signal (only needed if n is a power of 2) to the additive binary input; the second q-bit adder adds the constant $2^q - p$ to the result of the first one; the multiplexer selects the result of one of the q-bit adders based on the sign of the second one.

The added cost for an $(n, q; p)$ modular parallel incrementer with respect to an (n, q) parallel incrementer is the cost of a q-bit ripple-carry adder and a q-bit multiplexer; the delay overhead is less than 2 units (1 FA delay plus a multiplexer delay). The speedup methods discussed in Section 2, while applicable to this design, make it somewhat more complex. We are thus motivated to seek other performance improvement methods for both simple and modular APCs.

4. Improving the Performance

The designs of simple or modular parallel incrementers presented so far are efficient for moderate n and q in that they involve negligible cost overhead. However, when n or q becomes larger, their delay or complexity will increase. In this section, we introduce a technique called *delayed readout* and present several APC designs, with small overhead, that are suitable for applications that can tolerate a small additional delay in reading out the correct accumulated count. We also deal with pipelined designs. The two techniques essentially yield efficient designs for the cases of large q and large n, respectively.

4.1. APCs with Delayed Readout

We first present a simple modulo-2^q APC with delayed readout. The accumulated count is kept in carry-save form, which we call *internal count*, rather than fully computed within each cycle. The output of the CPC part and the stored internal count are added by a carry-save adder (CSA) to get the new internal count within each cycle. So, only 1 FA delay, plus the register setup time, is added to the delay of the CPC to achieve the accumulative property. When there is need for reading out the accumulated count, the correct external count is obtained by adding the two components of the carry-save internal count using a carry-propagate adder of length q. The added complexity compared with the simple APC design is a register and a carry-save adder, both of length q. In return, the internal delay (not counting the readout operation) is reduced to $2\lfloor \log_2(n-1) \rfloor$ FA delays, which is only one FA delay larger than that of an n-input CPC and is independent of q.

It is also possible to take advantage of the fact that the high-order bits of the count are updated relatively

infrequently. Thus, one can keep these bits in a counter of the type proposed by Vuillemin [VUIL91]. When a readout is needed, carries in the carry-save part are assimilated, leading to the determination of the carry going into the high-order section. This carry then selects either the original count of the incremented count for output. Both delay and complexity are thus reduced. The optimal division of the word-length q into the carry-save part and the upper (counter) part is technology-dependent.

Modular operation increases the complexity somewhat. With either operation mode discussed above, the correct modular count is impossible to maintain without propagating all carries. The carry-out can be handled by ignoring it and instead adding $2^q - p$ to the internal count (this requires an additional CSA level and a multiplexer). With a carry-save internal count added to the CPC output, the maximum possible sum of $(2^q - 1) + (2^q - 2) + n$ may exceed $4p$ in the worst case. Thus, modular reduction at readout can add significantly to the implementation cost and may be unacceptable in some applications.

4.2. Pipelining Considerations

A parallel incrementer or APC may be used to accumulate the count for multiple sets of n inputs. The multiple sets could be independent, resulting from a computation in which things to be counted are obtained in multiple stages, or due to input partitioning in order to reduce the implementation cost of the APC. Pipelining the multiple inputs is a natural way to reduce the total APC delay.

The pipelined version of our modulo-2^q APC uses the delayed readout method. Full-adder outputs are connected to latches. Each sum bit is connected to a single latch, whereas each carry output (except in the last level) is connected to two latches. An appropriate number of latches are inserted on each line between the CPC part and the CSA in order to equalize the delay from input to the CSA for all bits. In this way, once the two CSA registers (the bottom two rows of latches) are read out and cleared, the next computation can proceed with no intermediate delay. Since the feedback loop has merely one level of FA delay, such APCs can be pipelined efficiently. The construction of a pipelined (12, 6) APC is illustrated in Fig. 5. This design can operate at a higher throughput than a pipelined partial CSA tree [KORE93] which has a feedback loop with 2 FA delays.

The pipelined design of two-level accumulative parallel counter can be realized by using a pipelined internal accumulative parallel counter, an external register of length q, two buffers of length q', and a reduction circuit consist of several carry-propagate adders of length q. One carry-propagate adder suffices when modulus $p = 2^q$. Three or four (five) adders are required for the reduction circuit when $2^q + 2^{q'+1} + n - 3 \geq 2p$ or $3p$ ($4p$), while in most cases, two adders suffice unless 2^q, $2^{q'}$, n, and p are comparable.

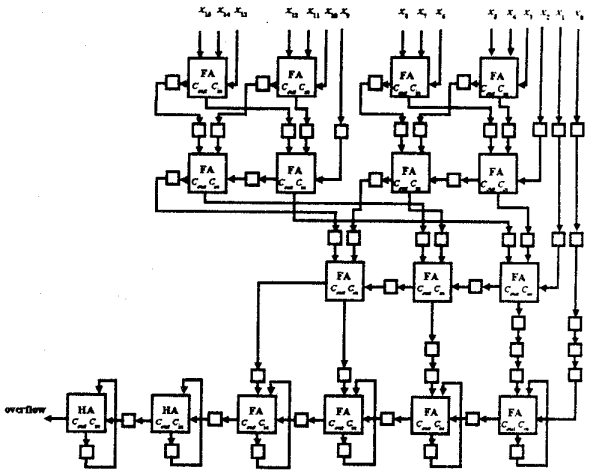

Fig. 5. Design of a pipelined (16, 6) APC.

The buffers are used to keep the values of internal registers whenever a read-out or reduction operation is needed. When the values of the internal registers are written into the buffer, the two internal registers are cleared. Then the reduction is performed and the result is stored in the external register of length q and/or read-out. The operation of counting does not need to be stopped when the accumulated count is read out. However, the last two rows of latches in Fig. 5 must be provided with a reset line so that the counter can be cleared when needed.

5. Some Applications

Parallel incrementers and APCs find applications in many areas, including modular or serial-parallel multipliers, (modular) multi-operand adders [PIES94], digital neural networks [ZHAN91], and large-scale systolic associative processors [PARH92].

In designing high-speed multipliers based on column compression (finding the sum of bits in one column of the partial products matrix), the role of APCs and parallel incrementers is quite similar to that of additive multiply modules, used to design highly regular multiplication networks with skewed layout [HWAN79], in that they allow uniform synthesis with identical building blocks, regular connections, and little need for external *glue* logic.

In digital neural networks and threshold logic applications, a parallel incrementer with its additive input set to the negative of the threshold can be used to evaluate the *firing* condition, since the sign of the output would directly indicate if the number of active inputs exceeds the threshold. Unequal input weights can be handled either by generalizing our designs to include multiple input *columns*

or, in the case of weights that are small integers, using fan-out or multiple sequential passes.

Finally, systolic associative processors (APs) can make use of our designs for counting the multiplicity of responders following a search operation. Systolic APs are built from small building-block APs that are connected into a linear array, a 2-D mesh, a binary tree, etc. To determine the number of responders, a *count* instruction is pipelined through the system. As the instruction is forwarded from one module to the next along the pipeline path, it carries the partial count up to that stage (point in linear array, diagonal cut in 2-D mesh, or level in tree). The receiving module must then combine information about its own responders with this partial count before forwarding the result.

6. Conclusion

We have defined parallel incrementers and APCs as true generalizations of ordinary incrementers and sequential counters. We use the term *accumulative parallel counter*, rather than the more appropriate *parallel counter*, for such devices in order to avoid confusion and to honor the widespread use of the latter to designate an $(n, \lfloor \log_2 n \rfloor + 1)$ parallel incrementer, with its y input fixed at 0. Our design strategies and examples are not meant to exhaust the space of possible design approaches but rather as a beginning in the discussion of these novel devices.

Further research may deal with many issues, including: refinement, extension, and augmentation of the techniques presented; detailed analyses of delay, cost, and associated tradeoffs; investigation of usefulness and impact in other application areas; and possible generalization to accumulative parallel compressors.

References

[DADD80] Dadda, L., "Composite Parallel Counters", *IEEE Trans. Computers*, Vol. 29, pp. 940-946, Oct. 1980.

[DORM81] Dormido, S. and M.A. Canto, "Synthesis of Generalized Parallel Counters", *IEEE Trans. Computers*, Vol. 30, pp. 699-703, Sep. 1981.

[DORM82] Dormido, S. and M.A. Canto, "An Upper Bound for The Synthesis of Generalized Parallel Counters", *IEEE Trans. Computers*, Vol. 31, pp. 802-805, Aug. 1982.

[FOST71] Foster, C.C. and F.D. Stockton, "Counting Responders in an Associative Memory", *IEEE Trans. Computers*, Vol. 20, pp. 1580-1583, Dec. 1971.

[GAJS80] Gajski, D.D., "Parallel Compressors", *IEEE Trans. Computers*, Vol. 29, May 1980.

[HWAN79] Hwang, K., *Computer Arithmetic: Principles, Architecture, and Design*, Wiley, 1979.

[JONE92] Jones, R.F. Jr. and E.E. Swartzlander Jr., "Parallel Counter Implementation", *Asilomar Conf.*, pp. 381-385, 1992.

[KOBA78] Kobayashi, H. and H. O'Hara, "A Synthesizing Method for Large Parallel Counters with a Network of Smaller Ones", *IEEE Trans. Computers*, Vol. 27, pp. 753-757, Aug. 1978.

[KORE93] Koren, I., *Computer Arithmetic Algorithms*, Prentice-Hall, 1993.

[LAIH82] Lai, H.C. and S. Muroga, "Logic Networks of Carry-Save Adders", *IEEE Trans. Computers*, Vol. 31, pp. 870-882, Sep. 1982.

[LINR95] Lin, R. and S. Olariu, "An Improved Shift Switching Based Parallel Counter Scheme", *Proc. Workshop Reconfigurable Architectures*, Santa Barbara, CA, Apr. 1995.

[MEOA75] Meo, A.R., "Arithmetic Networks and Their Minimization Using a New Line of Elementary Units", *IEEE Trans. Computers*, Vol. 24, pp. 258-280, Mar. 1975.

[OBER81] Oberman, R.M.M., *Counting and Counters*, Macmillan, 1981.

[PARH89] Parhami, B., "Parallel Counters for Signed Binary Signals", *Proc. Asilomar Conf. Signals, Systems, and Computers*, Pacific Grove, CA, Oct./Nov. 1989, pp. 513-516.

[PARH92] Parhami, B., "Architectural Tradeoffs in the Design of VLSI-Based Associative Memories", *Microprocessing and Microprogramming*, Vol. 36, pp. 27-41, 1992/93.

[PARH94] Parhami, B., "Analysis of Tabular Methods for Modular Reduction", *Proc. Asilomar Conf. Signals, Systems, and Computers*, Pacific Grove, CA, Oct./Nov. 1994, pp. 526-530.

[PIES94] Piestrak, S.J., "Design of Residue Generators and Multioperand Modular Adders Using Carry-Save Adders", *IEEE Trans. Computers*, pp. 68-77, Jan. 1994.

[SING73] Singh, S. and R. Waxman, "Multiple Operand Addition and Multiplication", *IEEE Trans. Computers*, Vol. 22, No. 2, pp. 113-119, Feb. 1973.

[SVOB70] Svoboda, A., "Adder with Distributed Control", *IEEE Trans. Computers*, Vol. 19, pp. 749-751, Aug. 1970.

[SWAR73] Swartzlander, E.E., Jr., "Parallel Counters", *IEEE Trans. Computers*, Vol. 22, pp. 1021-1024, Nov. 1973.

[SWAR80] Swartzlander, E.E., Jr., "Merged Arithmetic", *IEEE Trans. Computers*, Vol. 29, pp. 946-950, Oct. 1980.

[VUIL91] Vuillemin, J.E., "Constant Time Arbitrary Length Synchronous Binary Counters", *Proc. Symp. Computer Arithmetic*, Grenoble, France, June 1991, pp. 180-183.

[ZHAN91] Zhand, D., G.A. Jullien, W.C. Miller, and E. Swartzlander Jr., "Arithmetic for Digital Neural Networks", *Proc. Int'l Conf. Computer Architecture*, 1991, pp. 58-63.