Efficient Designs for Multi-Input Counters

Chi-Hsiang Yeh
Dept. of Electrical & Computer Engineering,
Queen's University
Kingston, Ontario, Canada, K7K 3N6

Behrooz Parhami
Dept. of Electrical & Computer Engineering,
University of California,
Santa Barbara, CA 93106-9560, USA

Abstract

A multi-input counter, or 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 inputs to a stored value. In this paper, we present efficient designs for simple multi-input counters and their modular versions, which keep the accumulated count modulo an arbitrary constant.

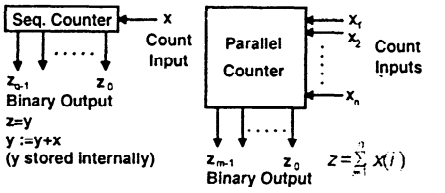

Figure 1. Sequential and parallel counters.

1 Introduction

In its simplest form, a counter is a sequential circuit that stores an integer value and can increment (and, in the case of up/down counters, also decrement) it by 1 upon the receipt of a special "enable" or "count" signal. Counters are among the most widely used components in digital systems, with applications in computer systems, communication equipment, scientific instruments, and industrial process control, to name a few. A vast variety of counter designs have been proposed in the literature [3, 6, 12], patented [1, 2, 4], and/or used in practice.

Although techniques are available for designing high-speed counters with conventional number representations [5], the speeds that can be achieved are limited by the requirement for carry propagation. Even with advanced designs based on redundant number representations [6] or hierarchical incrementation using small blocks at the right end and increasingly wider blocks toward the left [11, 15], speeds will be limited by the basic switching characteristics of components and, perhaps, by the need for final conversion of the redundant count into a conventional binary number. Even when no conversion is needed, to count the number of 1s among many thousands of bits by feeding them sequentially to a high-speed counter would imply delays well in excess of several microseconds and such a delay is unacceptable in certain applications. Some form of parallelism in handling the large number of input bits is needed to achieve higher speeds.

In fact, there are indications that the reverse of the process discussed above may be viable in counter design: Instead of sequentializing naturally parallel inputs to cut down on hardware complexity and cost, we might want to parallelize high-frequency sequential inputs to allow the use of relatively slow, and thus low-cost, compact, and/or power-efficient, circuits. For example, a 400 MHz incoming signal, when demultiplexed 32 ways, is transformed into a set of 12.5 MHz signals that can be handled by non-speed-critical components. In this way, a complex circuit that would have to be designed by paying meticulous attention to speed optimization at every juncture (with the attendant overheads in design time, testing effort, VLSI area, and power consumption) is replaced by a simple high-speed front end feeding a lower-grade main part. With the unyielding pursuit of high-throughput and low-power digital systems, this type of "demultiplexed" computation has become the norm in certain areas (notably data communication) that, as recently as a decade ago, used to rely on "multiplexed" hardware for reasons of economy.

A parallel counter [13] has been defined as a combinational digital 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 $x_1, x_2, ..., x_n$; 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.

0-7803-5700-0/99/$10.00©1999 IEEE

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. In effect, the corresponding sequential counter is one whose count is reset to zero after every n cycles. We define a *multi-input counter (MIC)*, also called *an accumulative parallel counter (APC)* [8], 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 or more generally, the sum of its previous content and its weighted inputs. Thus, in each cycle, the stored or output count of the former 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. MICs can be used to replace parallel counters in many applications in order to reduce the hardware cost and/or store the output values for future use.

In this paper, we focus on designs for simple MICs whose inputs are single-bit numbers. Most of the proposed techniques can also be applied to the general case. We discuss modulo-p multi-input 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 MICs are only slightly more complex than their combinational versions. Our merged designs can considerably reduce the delay for readout of the naive designs that combine an ordinary combinational parallel counter (CPC) with a register and a fast adder in a two-stage arrangement. while offering lower cost at the same time. The results are then extended to the case of an arbitrary modulus 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 that fact.

2 Parallel Incrementers/Decrementers and MICs

In this section, we discuss the design of (n,m) parallel incrementers with n single-bit inputs $x_1,x_2,...,x_n$ (increment signals), an m-bit binary input $y_{m-1}...y_1y_0$, and an m-bit binary output $z_{m-1}...z_1z_0$ which is the sum of y and the n single-bit inputs modulo 2^m or 2^m-1 (Fig. 2a). Since mod-(2^m-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. A parallel decrementer is similarly defined to compute $y-\sum x_i$, where the x_i are now called the decrement signals and y is either an unsigned value that wraps around (underflows) when it reaches zero or else it is a suitably encoded signed integer. We will not discuss parallel decrementers explicitly, as they can be easily derived from the designs offered for parallel incrementers.

Clearly, an (n,q) MIC can be implemented by an (n,q) parallel incrementer connected to a q-bit register. When q is large, however, the delay of such a design would be significant and can be considerably improved by suitably connecting an $(n,\lceil\log_2 n\rceil)$ parallel incrementer to a $\lceil\log_2 n\rceil$-bit register and a $(q-\lceil\log_2 n\rceil)$-bit sequential counter (Fig. 2b), which can be any design with small delay [11, 15]. Since the design of fast counters has been studied extensively, in what follows, we concentrate primarily on the synthesizing of parallel incrementers in order to complete the MIC design of Fig. 2b.

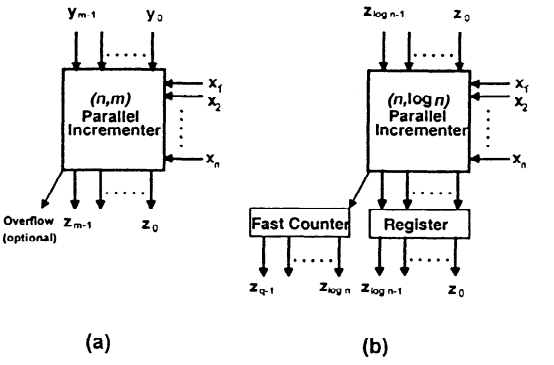

Figure 2. (a) An (n,m) parallel incrementer and (b) an (n,q) MIC.

Clearly, a parallel incrementer can be designed by combining an ordinary CPC with a fast adder. The delay and cost of such an (n,m) parallel incrementer are increased by the delay and cost of the m-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). The overhead of such a design is quite high for moderate m. 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 m is not much larger than $\log_2 n$; this is clearly the case for the intended use in Fig. 2b. An (n,m) parallel incrementer of this type consists of an $(n-1)$-input CPC and an m-bit ripple-carry adder, as shown for a (32,7) 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 $\lfloor\log_2 l\rfloor+1=\lceil\log_2(2l+1)\rceil$. 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 preceding designs are easily extended to the case of modulo-p parallel incrementers, where the modulus p satisfies $2^{m-1}<p\leq 2^m$. An $(n,m;p)$ modular parallel incrementer computes the m-bit modulo-p sum of the n single-bit increment signals and an m-bit binary input. An $(n,m;p)$

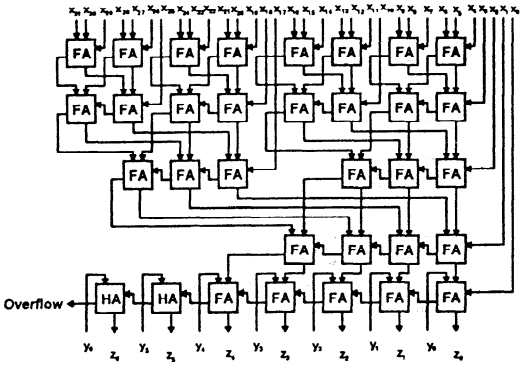

Figure 3. Design of a $(32,q)$ parallel incrementer.

modular parallel incrementer can be realized by using an (n,m) parallel incrementer followed by a modulo-p reduction circuit [7]. However, as before, a merged design significantly reduces the delay and cost overhead.

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

A straightforward implementation of a modular parallel incrementer consists of an $(n-1)$-input CPC followed by two m-bit ripple-carry adders and an m-bit multiplexer: the first m-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 m-bit adder adds the constant $2^m - p$ to the result of the first one; the multiplexer selects the result of one of the m-bit adders based on the sign of the second one (Fig. 4).

The added cost for an $(n,m;p)$ modular parallel incrementer with respect to an (n,m) parallel incrementer is the cost of an m-bit ripple-carry adder and an m-bit multiplexer; the delay overhead is less than 2 units (1 FA delay plus a multiplexer delay). The cost and delay can both be somewhat reduced if one uses custom FA cells incorporating the selection function in their sum logic in lieu of standard FA and multiplexer cells at the bottom of Fig. 4.

When m is large and high speed is desired, computation of the sums, except for their 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. The use of carry-select is particularly attractive in terms of both speed and cost.

In what follows we derive the layout area of the preceding parallel incrementer design under the Thompson model using two layers of wires (see [14] for details). If an FA can be laid out within an $a \times b$ rectangle, and the width of a wire is 1, then an (n,m) parallel incrementer can be laid out within

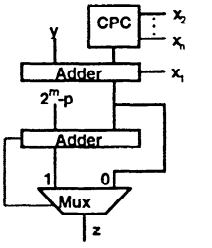

Figure 4. Modulo-p parallel incrementer.

an area of about

$$\frac{b+2}{4}(a\log_2 n + \frac{1}{2}\log_2^2 n)n + mab.$$

More precisely, we need at most 2 tracks for wires connecting the first and the second level of FAs, 3 tracks for wires connecting the second and the third level of FAs, i tracks for wires connecting the $(i-1)^{th}$ and the i^{th} level of FAs, and so on (see Fig. 3). Since there are $\lceil \log_2 n \rceil - 1$ levels of FAs other than the ripple-carry adder at the last stage, the total number of tracks required for these wires is

$$2+3+\cdots+(\lceil \log_2 n \rceil - 1) \approx \frac{\log_2^2 n}{2}.$$

Therefore, the height of the layout is about

$$a\log_2 n + \frac{1}{2}\log_2^2 n.$$

The width of the layout is easily seen to be about

$$\frac{n-n/4}{3}(b+1) + \frac{n}{4} = \frac{n}{4}(b+2).$$

Note that we have implicitly assumed $a \geq 2$ and $b \geq 2$; conditions that are certain to hold in practice. When m is large, we can utilize the empty space at the lower left of the preceding layout to accommodate the FAs of the final ripple-carry adder or even wrap the ripple-carry adder around other parts of the layout. The increase in area is at most about mab, so the result follows.

Note that the preceding is only an upper bound on the layout area of our design and the actual layout may be even more compact if done using a software router. Note also that the layout of an (n,m) parallel incrementer requires n input lines and m output lines so the circumference of the layout must be at least $n+m$. Moreover, the n inputs have to be combined to obtain the output, which requires $\Theta(\log n)$ levels when using constant fan-in devices such as FAs. For this, we need a minimum of about n FAs, as argued above. Therefore, it is impossible for the layout of any parallel incrementer design to have significantly smaller layout than that of our design.

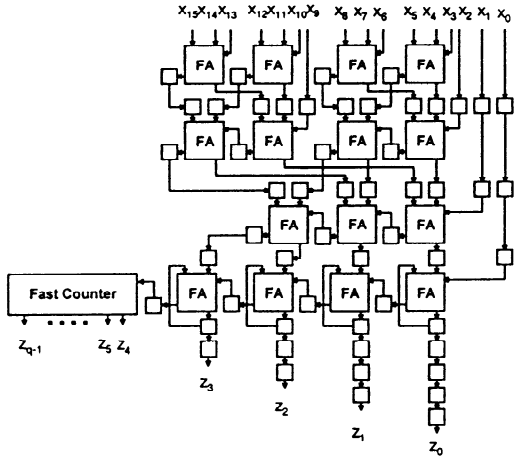

Figure 5. Design of a pipelined $(16, q)$ MIC.

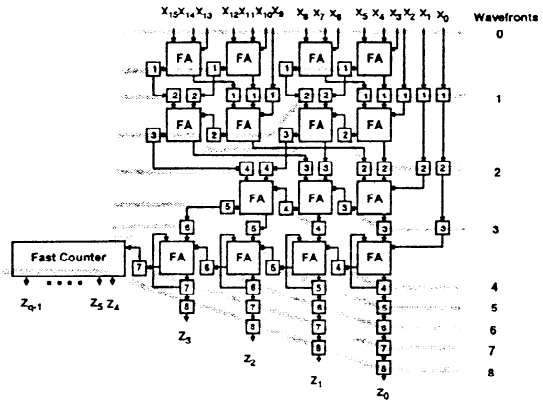

Figure 6. Computation wavefronts in the pipelined MIC of Fig. 5.

3 Pipelined Multi-Input Counters

A parallel incrementer or MIC 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 MIC. Pipelining the multiple inputs is a natural way to reduce the total MIC delay.

In the pipelined version of our modulo-2^q MIC (Fig. 5), full-adder sum and carry outputs are connected to latches. The carry out of the leftmost FA in each shaded group (forming a ripple-carry adder) is connected to a second latch before going to the next level. In Fig. 6, the integer shown inside each latch indicates the number of clock cycles from the application of inputs to the first loading of the latch. Latches with identical integer labels define a computational wavefront that incorporates all the partial parallel counting results for one set of inputs. By removing every other layer of latches (say those with odd labels), a more economical, and correspondingly slower, pipelined implementation can be obtained.

If the latency of the serial counter part is d clock cycles, then we add $d+1$ latches to the most significant sum bit of the pipelined ripple-carry adder in the last level, $d+2$ latches to the second most significant sum bit, $d+3$ latches to the third, and so on. Then the outputs of the serial counter and those of the last latches of the various sum bits collectively represent the current count of the pipelined MIC. For a 2^l-input MIC, the count will be available for read out after $2l+d-1$ cycles. For example, in the particular MIC given in Fig. 5, the number of inputs is 16 and the latency of the serial counter is 1, so the count will appear at the lowermost sum latches and counter outputs after 8 cycles, no matter how large q is. Since serial counters with $d=1$ are available, the readout delay is only $2l$ for 2^l inputs, which is very close to the minimum.

The preceding pipelined design for simple MICs can be extended to obtain pipelined modular MICs with high counting and sampling rate and low latency. To obtain a pipelined $(n, q; p)$ MIC, we need two additional q-bit accumulative adders with fixed input $-2p$, the left one with initial value $A=0$ and the right one with initial value $B=-p$ (Fig. 7). The outputs of a pipelined (n, q) MIC are fed to two pipelined (fast) adders, one adding A, and the other adding B, to the intermediate count of the pipelined (n, q) MIC. The outputs of the two adders are then fed to a multiplexer with a simple control logic whose inputs consist of the overflow signals from the pipelined (n, q) MIC and the two pipelined adders at the final stage.

Initially, the output of the fast adder on the left holds the correct count. After at least p/n cycles, the adder on the left may overflow and the outputs of the other become the correct count; an event that can be detected by examining the three overflow signals in Fig. 7. Then the content of the left accumulative adder has to be reduced to $A=-2p$. Since the output of the right adder will continue to be the correct count for at least another p/n cycles, the accumulative adders do not need to be very fast. Note that a fast q-bit adder can have latency as small as $O(\log q) = O(\log\log p)$, which is considerably smaller than $p/n > p/\log_2 p$. In usual cases, a ripple-carry adder, which has latency $\log_2 p = o(p/\log p)$, is sufficient so the complexity introduced by the accumulative adders is small. Moreover, the inputs of the accumulative adders are fixed, which may further reduce the implementation cost. When the right adder overflows, the left adder will again supply the correct count and the the content of the right accumulative adder has to be reduced to $B=-3p$. The computation thus flip-flops between the left and right paths in Fig. 7.

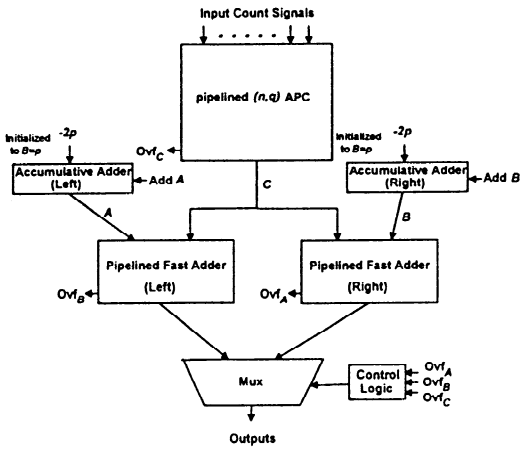

Figure 7. Design of a pipelined $(n, q; p)$ MIC.

Note that the most significant bit of the pipelined (n, q) MIC is discarded so that the intermediate count is a multiple of 2^q smaller than the true sum of the input signals since the MIC was initialized. Although 2^q is not a multiple of p, this does not pose any problem since at least one of the two values $\dot{-}A$ or $-B$ will be truncated by exactly the same amount, so that the effect is always canceled. This is the reason we need to use accumulative adders of width q, the number of bits in the output of our (n, q) MIC. Note also that the latches at the final stages of the (n, q) MIC part can be removed, if so desired, while an appropriate number of latches must be added to each output bit of the two adders at the final stage of the modular MIC. Clearly, both the counting and sampling periods are dictated by only the delay of a FA and a latch. The latency of a modular $(n, q; p)$ parallel incrementer is only increased by $O(\log q) = O(\log \log p)$ compared to a simple (n, q) parallel incrementer when fast adders are used at the final stage.

4 Conclusion

In this paper, we have proposed efficient pipelined designs of simple and modular MICs. They improve previous designs significantly in terms of sampling rate and read-out latency, while requiring smaller cost at the same time. Further research is warranted on several aspects of MICs. While the designs presented in this paper are area- and time-efficient, they are not strictly optimal in either respect. Therefore, improving the latency or VLSI layout area of our designs might be attempted, although we conjecture that the much greater complexity of faster designs will likely make them highly cost-ineffective. Actual hardware realization and VLSI layout may also provide additional insights and optimizations at a finer level. Finally, identification and evaluation of new application areas for parallel counters, and their generalized forms suggested here, might lead to accelerated development as well as additional domain-specific designs and optimizations.

References

[1] D. Chu and M. Ward, "Data capture in an uninterrupted counter," *U.S. Patent No. 4,519,091,* May 1985.

[2] D.H. Eby, "Synchronous programmable two-stage serial/ parallel counter," *U.S. Patent No. 4,905,262,* Feb. 1990.

[3] M. Ercegovac and T. Lang, "Binary counter with counting period of one half adder independent of counter size," *IEEE Trans. Circuits and Systems,* Vol. 36, No. 6, pp. 924-926, June 1989.

[4] M.W. Evans, "Minimal logic synchronous up/down counter implementations 'for CMOS," *U.S. Patent No. 4,611,337,* Sep. 1986.

[5] R.M.M. Oberman, *Counting and Counters,* Wiley, New York, 1981.

[6] B. Parhami, "Systolic up/down counters with zero and sign detection," *Proc. Symp. Computer Arithmetic,* pp. 174-178, 1987.

[7] B. Parhami, "Analysis of tabular methods for modular reduction," *Proc. Asilomar Conf. Signals, Systems, and Computers,* pp. 526-530, 1994.

[8] B. Parhami and C.-H. Yeh, "Accumulative parallel counters," *Proc. Asilomar Conf. Signals, Systems, and Computers,* pp. 966-970, 1995.

[9] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs,* Oxford University Press, 2000.

[10] N. Pippenger, "The complexity of computations by networks," IBM *J. Res. Develop.,* Vol. 31, No. 2, pp. 235-243, Mar. 1987.

[11] M.R. Stan, "Synchronous up/down counter with clock period independent of counter size," *Proc. IEEE Symp. Computer Arithmetic,* pp. 274-281, 1997.

[12] M.R. Stan, A.F. Tenca, and M.D. Ercegovac, "Long and fast up/down counters," *IEEE Transactions on Computers,* Vol. 47, No. 7, pp. 722-735, July 1998.

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

[14] Thompson, C.D., "Area-time complexity for VLSI," *Proc. ACM Symp. Theory of Computing,* 1979, pp. 81-88.

[15] J.E. Vuillemin, "Constant time arbitrary length synchronous binary counters," *Proc. Int'l Symp. Computer Arithmetic,* pp. 180-183, 1991.

[16] C.-H. Yeh and B. Parhami, "Efficient pipelined multioperand adders with high throughput and low latency: designs and applications," *Proc. Asilomar Conf. Signals, Systems, and Computers,* pp. 894-898, 1996.