Transactions Letters

High-Speed Parallel CRC Circuits in VLSI

Tong-Bi Pei and Charles Zukowski

Abstract—**In this paper, we investigate the use of VLSI technology to speed up cyclic redundancy checking (CRC) circuits used for error detection in telecommunications systems. By generalizing our analysis of a parallel prototype, we estimate performance over a wide range of external constraints and design choices. We show that parallel architectures fall somewhat short of ideal speedups in practice, but they should still enable current CMOS technologies to go well beyond 1 Gb/s data rates.**

I. Introduction

SENDING information over communication links involves both physical data transmission and information processing at each end. Transmission technology is in the midst of a revolution in speed capability, and thus the information processing functions are becoming more of a bottleneck. In this paper, we investigate using VLSI technology to speed up one key functional block: the cyclic redundancy checking (CRC) circuit.

CRC is a well-established method to allow detection of a wide range of data transmission errors. Roughly speaking, a small number of bits (say 32) is added to the end of a long message so that the entire bit stream, when treated as a polynomial (or number), is divisible by a particular key polynomial that is programmed into both the sender and receiver. The addition of any transmission errors is likely to result in a new message that is no longer divisible by the key. The conventional implementation of both CRC generation and detection circuits is a simple linear binary machine, i.e., a shift register connected with feedback through exclusive-or gates (see Fig. 1). While such a circuit is simple and can be run at fairly high clock speeds, it suffers from the limitation that the information stream must be bit-serial. If the clock frequencies of electronic finite state machines cannot keep up with optical transmission data rates, alternate architectures must be considered.

In the past it has been proposed that a parallel implementation, e.g., where the computation is done one byte at a time, would enable a large increase in bit rate [1], [2]. Such an approach appears to be a natural for modern VLSI technology, where the cost of additional circuitry can be kept quite small.

Paper approved by the Editor for VLSI in Communications of the IEEE Communications Society. Manuscript received April 15, 1989; revised November 9, 1990. This work was supported by the NSF Center for Telecommunications Research at Columbia University under Contract CDR-88-11111, and an NSF Presidential Young Investigator Award MIPS-86-58112.

The authors are with the Department of Electrical Engineering, Columbia University, New York, NY 10027.

IEEE Log Number 9107302

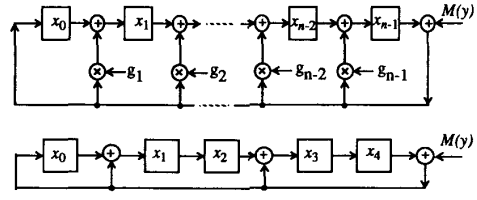

Fig. 1. A general serial CRC architecture and an example for $G(y) = 1 + y + y^3 + y^5$.

A reasonable question that must be answered, though, is how much speedup is really possible. Good estimates for the area and power penalties are also important. Through the design of a prototype and some generalized analysis, we answer these questions more fully than has been done in the past. We show how current CMOS technologies can be used to meet throughput requirements well above 1 Gb/s.

The second section contains a more detailed discussion of the CRC algorithm, both serial and parallel versions. In the third section, we discuss the design of a prototype parallel CRC chip using a standard two micron CMOS process. The fourth section contains a generalization of the experimental results for different keys and different degrees of parallelism.

II. CRC Algorithms

CRC algorithms are usually explained by viewing an m-bit binary message $(z_0, z_1, \cdots, z_{m-1}$ where z_{m-1} arrives first) as a polynomial $M(y) = z_0 y^0 + z_1 y^1 + \cdots + z_{m-1} y^{m-1}$. The basic CRC task in the sender is to calculate, using modulo-2 arithmetic, the remainder left over from dividing $y^n M(y)$ by a constant (and carefully chosen) nth-order generating (or key) polynomial $G(y) = \sum_{i=0}^{n} g_i y^i$ where $g_0 = g_n = 1$. This remainder $R(y)$ is then subtracted from $y^n M(y)$ (equivalent to addition in modulo-2 arithmetic, and thus concatenation with $M(y)$ since $y^n M(y)$ ends with n zeros) to create a new message that is guaranteed to be evenly divisible by $G(y)$. At the receiving end, a similar calculation can be used to verify that the remainder is zero after transmission [3].

Calculating a remainder is generally done with an algorithm that is similar to standard hand calculations; repeated shift and conditional subtract. Fig. 1 illustrates how the algorithm can be implemented with flip-flops and logic, both for the general case and for a particular $G(y)$. After the entire input $M(y)$ has been shifted in, the n-bit remainder is left in the flip-flops.

If the flip-flops are initialized to ones instead of zeros, as is often done, the only other change required in the system is in the remainder that the receiver is looking for. If we let $X(t) = [x_0 x_1 \cdots x_{n-1}]$ be the remainder after t conditional subtractions, the state transition equation for circuits using the general architecture of Fig. 1 can be written as (where $\oplus$ represents modulo-2 addition):

$$X(t+1) = [x_0 x_1 \cdots x_{n-2}][0|I_{n-1}] \oplus ([x_{n-1}] \oplus z_t)G \quad (1)$$

where $G = [g_0 g_1 \cdots g_{n-1}]$, z_t is the current bit in the message, and x_i is shorthand for $x_i(t)$.

The parallel approach to speed up the CRC algorithm is to merge a number of the shift and conditional subtract operations together within a single clock cycle. If k of the total m subtractions are combined, k message bits must arrive during each clock cycle and only $\frac{m}{k}$ clock cycles are required to compute the remainder. If we let $Z(t) = [z_t z_{t+1} \cdots z_{t+k-1}]$ be a vector containing a parallel group of k message bits as in [1], and combine k conditional subtractions, the state transition equation for the parallel CRC circuit becomes

$$X(t+k) = [x_0 x_1 \cdots x_{n-k-1}][0|I_{n-k}] \oplus ([x_{n-k} x_{n-k+1} \cdots x_{n-1}] \oplus Z(t))D \quad (2)$$

where

$$D = \begin{bmatrix} G \\ GT^1 \\ \cdot \\ \cdot \\ GT^{k-1} \end{bmatrix};$$

and T^j is the jth power of the matrix

$$T = \begin{bmatrix} 0 & 1 & 0 & \cdot & 0 \\ 0 & 0 & 1 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & 1 \\ g_0 & g_1 & g_2 & \cdot & g_{n-1} \end{bmatrix}.$$

By grouping together all terms that multiply D in (2), as with G in (1), an efficient assignment of intermediate variables results. Equation (2) jumps in a single step to the point where (1) jumps in k steps.

III. Parallel CRC Circuit

As an experiment to help determine implementation costs, we built a nonoptimized CRC generator chip using $k = 8$, $n = 32$, and an IEEE standard [4] generating polynomial is shown in (3) below. For this polynomial we have (4) shown below and the equation describing the state transition for this circuit becomes

$$X(t+8) = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ x_0 \cdots x_{23}] \oplus [u_0\ u_1 \cdots u_7]D \quad (5)$$

where $u_i = x_{i+24} \oplus z_{t+7-i}$, for $0 \le i \le 7$. Writing $X(t+8)$ as $[x'_0 x'_1 \cdots x'_{31}]$, the calculations described by (5) can be represented as in Fig. 2.

The technology used for our prototype was a two micron CMOS process with two levels of metal. The circuit followed the basic organization shown in Fig. 2 where each box represents a flip-flop and each circle represents an exclusive-or gate. The term $[x_0 x_1 \cdots x_{n-k-1}][0|I_{n-k}]$ from (2) is implemented simply by connecting the appropriate elements of $X(t)$ into the appropriate rows of the main array. Each row of the array implements a modulo-2 summation of up to nine binary variables, as determined by the D matrix. The eight exclusive-or gates along the top produce the intermediate variables u_i from (5) that are each distributed along a single column of the array.

In the implementation of the CRC cell, the array was separated into a number of parts as indicated on Fig. 2. The architecture of Fig. 2 was modified slightly, allowing compaction of the exclusive-or arrays by mixing nearby rows, allowing sharing of some intermediate variables among rows, and allowing tree structured connections along some rows. The exclusive-or gates used in the array shown in Fig. 2 were 2-input gates built with standard static circuits. SPICE simulations predict their delay with a single load to be about 3.5 ns. If they drive an entire column of the array with no buffers as done in the prototype, their predicted delay rises to about 37 ns. With simple buffers, this delay can be reduced to about 6 ns. The flip-flops are static edge-triggered D-type

$$G = [1\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0]. \quad (3)$$

$$D = \begin{bmatrix} 1&1&1&0&1&1&0&1&1&0&1&1&1&0&0&0&1&0&0&0&0&0&1&1&0&0&1&0&0&0&0&0 \\ 0&1&1&1&0&1&1&0&1&1&0&1&1&1&0&0&0&1&0&0&0&0&0&1&1&0&0&1&0&0&0&0 \\ 0&0&1&1&1&0&1&1&0&1&1&0&1&1&1&0&0&0&1&0&0&0&0&0&1&1&0&0&1&0&0&0 \\ 0&0&0&1&1&1&0&1&1&0&1&1&0&1&1&1&0&0&0&1&0&0&0&0&0&1&1&0&0&1&0&0 \\ 0&0&0&0&1&1&1&0&1&1&0&1&1&0&1&1&1&0&0&0&1&0&0&0&0&0&1&1&0&0&1&0 \\ 0&0&0&0&0&1&1&1&0&1&1&0&1&1&0&1&1&1&0&0&0&1&0&0&0&0&0&1&1&0&0&1 \\ 1&1&1&0&1&1&1&0&0&0&0&0&1&1&1&0&0&1&1&0&0&0&0&1&0&0&1&0&1&1&0&0 \\ 0&1&1&1&0&1&1&1&0&0&0&0&0&1&1&1&0&0&1&1&0&0&0&0&1&0&0&1&0&1&1&0 \end{bmatrix} \quad (4)$$

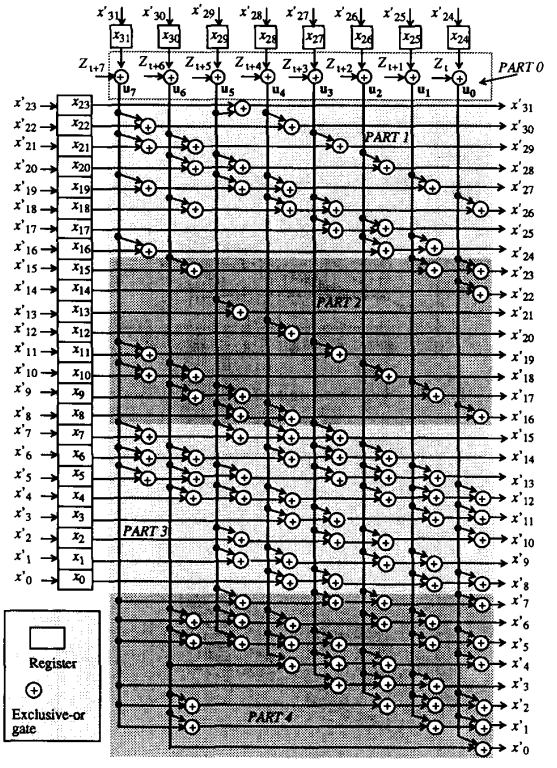

Fig. 2. The prototype CRC circuit.

Generating Polynomial	Degree	Notation
$1+y^2+y^{15}+y^{16}$	16	$G_{13}(y)$
$1+y^5+y^{12}+y^{16}$	16	$G_{14}(y)$
$1+y+y^4+y^6+y^8+y^{10}+y^{12}+y^{13}+y^{15}+y^{17}+y^{18}+y^{20}+y^{21}+y^{22}+y^{31}$	32	$G_{15_Y}(y)$
$1+y+y^2+y^{22}+y^{31}$	32	$G_{16}(y)$
$1+y+y^2+y^4+y^5+y^7+y^8+y^{10}+y^{11}+y^{12}+y^{16}+y^{22}+y^{23}+y^{26}+y^{31}$	32	$G_{17}(y)$
$1+y+y^5+y^6+y^9+y^{10}+y^{11}+y^{14}+y^{16}+y^{18}+y^{19}+y^{28}+y^{31}$	32	$G_{18}(y)$
$1+y+y^2+y^7+y^8+y^9+y^{10}+y^{11}+y^{13}+y^{17}+y^{22}+y^{23}+y^{24}+y^{25}+y^{26}+y^{27}+y^{31}$	32	$G_{19}(y)$

Fig. 3. Key for generating polynomials.

cells. SPICE simulations predict their delay with a single load to be about 3 ns.

The chip was fabricated through the MOSIS foundry and was verified to be functionally correct. It worked up to about 30 MHz (240 Mb/s) which was slightly faster than predicted by SPICE simulations. Since the circuit was not optimized for speed, there appears to be much room for improvement in the data rate. By adding features such as array column drivers (i.e., buffers), upgrading the technology beyond 2 micron, and increasing the parallelism beyond 8 bits, data rates well beyond 1 Gb/s should be possible.

IV. Analysis

The experience gained from the rough design of a single CRC cell can be extrapolated fairly well to a wide range of situations. Using the list of generating polynomials shown in Fig. 4 chosen from [1], [3], and [5], we would like to estimate the area, bandwidth, and power consumption for the circuit as a function of the generating polynomial G (including different values of n) and different choices for k ($k = 1$ representing a serial implementation). For the implementation of the exclusive-or array, we consider a sample of the many different possible approaches but assume that the most efficient IC designs would not stray too far from the regular array organization illustrated in Fig. 2.

The most important aspect of performance for high speed systems is the maximum circuit throughput. The minimum clock cycle time for a circuit based on (5) is given by

$$T = t_{\text{reg}} + t_{\text{load}}(G,k) + t_{\text{ex}}(h(G,k)) \tag{6}$$

where t_{reg} represents the total flip-flop delay (not a function of G and k), $h(G,k)$ is the maximum number of ones in any column of the D matrix (corresponding to the maximum depth of the exclusive-or array in the architecture of Fig. 2), $t_{\text{ex}}(l)$ represents the total delay of an $l+1$-input exclusive-or operation and $t_{\text{load}}(G,k)$ represents the time needed to generate and distribute the u_i variables. Since k bits are processed each clock cycle, the throughput is given by

$$\eta(G,k) = \frac{k}{t_{\text{reg}} + t_{\text{load}}(G,k) + t_{\text{ex}}(h(G,k))}. \tag{7}$$

Each of the terms in (7) can be analyzed separately. Since the maximum number of loads on the u_i drivers is on the order of n, and n is usually a fairly small and narrowly constrained number, we treat $t_{\text{load}}(G,k)$ as a constant. Using SPICE we estimated that the prototype, with buffers added, could drive the array in about 6 ns. In very large circuits this term could become significant, in which case the time required to drive up to n inputs, typically of order $\log_2 n$, would have to be modeled. The function $h(G,k)$, the maximum depth of the exclusive-or array, is illustrated in Fig. 4. A graph is included for each generating polynomial of Fig. 3. The depth tends to grow with increasing k and depends quite heavily on the polynomial used. In systems where a number of polynomials could be used, this function should be considered when the choice is made. We assume that a tree structure of exclusive-or gates is used, so the function $t_{\text{ex}}(l)$ grows logarithmically with l.

Using data obtained from the prototype design, we estimated the throughput speedup for different G and k as shown in Fig. 5. Equation (7) can be used to recompute Fig. 5 for a different technology or design style. While significant speedup can be achieved using parallel computation, simple multiplication by k is not quite achieved when $k > 2$. In fact, $\frac{k}{2}$ (or more precisely $[0.4k, 0.6k]$) appears to be a reasonable model over a wide range of situations. With $k = 8$, we estimate that the prototype polynomial achieved a speedup of about 4.9.

The area of the CRC circuit can be divided into two main components. The area of the registers is simply given by n

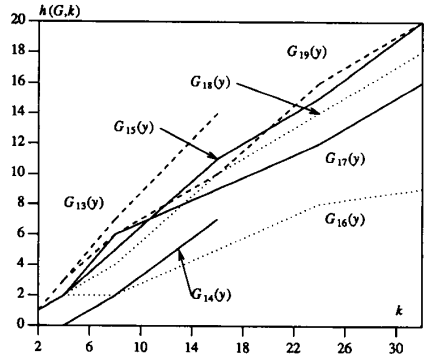

Fig. 4. Depth of the exclusive-or array for different G and k.

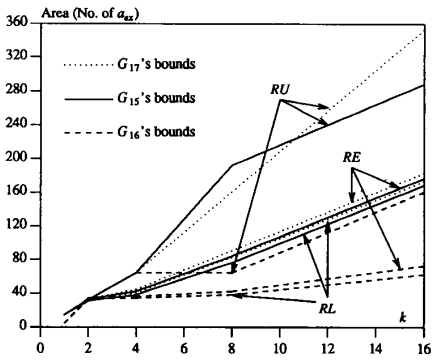

Fig. 6. Bounds on area of the exclusive-or array.

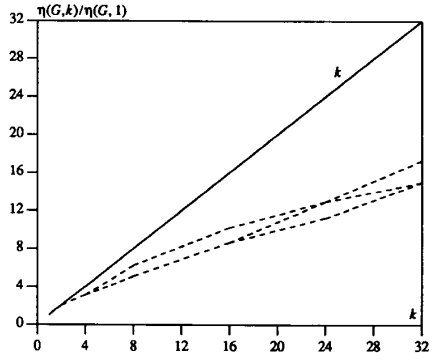

Fig. 5. Throughput speedup for different G and k.

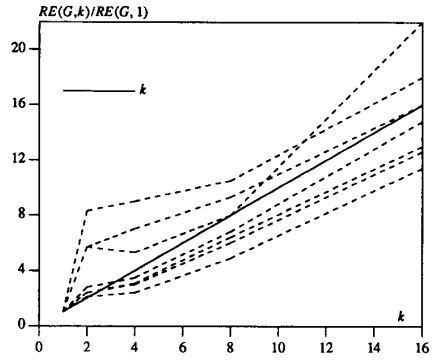

Fig. 7. Area comparison for different G and k.

times the area of a flip-flop a_{reg}, and it does not vary as a function of k. The area needed for exclusive-or gates varies depending on the partitioning and packing strategy used for the array, but can be roughly estimated as follows. We define the number of 2-input exclusive-or gates needed in the circuit, without sharing, as $RE(G,k) = s(G,k) + k$ where $s(G,k)$ is the total number of ones in the D matrix. Even when some sharing is combined with nearby packing in the array, it is hard to achieve an area of much less than $RE(G,k)$ times the area of a single 2-input gate a_{ex} due to the inevitable remaining gaps, so this provides a reasonable area estimate. If the array becomes large, and large drivers are required to drive the array, then the extra area devoted to these drivers must be considered. In addition, the extra area needed for wire routing, which is quite small in the prototype due to the use of second-level metal, could be included in a more detailed model.

Since the area estimate could be quite rough, we also computed some estimates that are conservative on the high and low side to bound the area. If rows are not mixed, there must be a significant number of gaps in the array structure, and we can define an equivalent number of gate locations as $RU(G,k) = nh(G,k) + k$. A reasonable upper bound for exclusive-or area is then given by $a_{\text{ex}}RU(G,k)$. Note that even for the upper bound, we assume that packing is used in each row. We found in the prototype design that the extra wiring area needed for row packing is quite small and the benefit is quite large. We also define $RL(G,k)$ as a rough measure (computed by hand) of the number of 2-input exclusive-or gates required if sharing of intermediate variables among nearby rows is allowed. A reasonable lower bound is then given by $a_{\text{ex}}RL(G,k)$, as this assumes that there are no gaps in the array.

The only factor in the total area estimate that is a function of G and k is thus the number of equivalent gate spaces needed in the array, bounded roughly between $RL(G,k)$ and $RU(G,k)$, and reasonably estimated by $RE(G,k)$. Fig. 6 illustrates how $RE(G,k)$, $RL(G,k)$, and $RU(G,k)$ grow as a function of k for a few representative polynomials. The size is clearly a fairly strong function of the generating polynomial used. Fig. 7 contains plots of the ratio $RE(G,k)/RE(G,1)$ as a function of k for a representative sample of polynomials. Tracking this estimate of equivalent gate count provides a rough indication of how the array size will grow across all array strategies discussed. This area ratio generally rises more quickly with k than does the throughput, and in some cases it surpasses even simple multiplication by k. In the prototype circuit where $k = 8$ was chosen, the array area expansion factor over the serial case was estimated at 6.

A final consideration in the design is the power dissipation. The analysis is sensitive to the design details and is highly dependent on the design constraints, e.g., if the clock frequency is fixed. Roughly speaking, the capacitive loads that must be

charged and discharged, accounting for much of the power dissipation in CMOS, are proportional to the circuit area. As a result, power will increase with greater k even if the clock frequency is unchanged. If a larger k enables the circuit to run at a lower frequency, the effects of these two changes with respect to power will tend to cancel. One interesting effect is that with larger k, there tends to be more glitching inside the array as the circuit settles so the power dissipation should rise faster with k than might be expected just from area considerations. If we let C_{ex}, C_{reg}, and C_{load} represent the total internal node capacitances in an exclusive-or gate cell, a register cell, and a u_i driver cell, respectively, a very rough approximation, based on the assumption that each node switches completely on an average of once per clock cycle, is given by

$$P = \frac{1}{2} f_{clk} V_{DD}^2 (s(G,k)C_{\text{ex}} + kC_{\text{load}} + nC_{\text{reg}}) \quad (8)$$

where f_{clk} is the clock frequency and V_{DD} is the power supply voltage.

We investigated two additional variations that influence the CRC performance. First, the exclusive-or array can be constructed using a cascade of 2-input gates, or multi-input gates that fit into an array structure [6]. Fig. 8 illustrates that, across a sample of generating polynomials, the simple cascade approach generally leads to lower speeds. Especially for large arrays, a combination of the approaches is likely to be optimal. Second, the array can be partitioned differently than indicated in Fig. 2, producing a relatively small impact on the area. Partitions with a small number of rows have a more limited capability for sharing among rows, but they do not suffer as much area loss from the need for an extra, mostly empty, column. The most important factor in choosing partition size, however, appears to be the desired aspect ratio.

V. Conclusion

We have investigated the use of VLSI to speed CRC computations in telecommunications systems. While we found that the parallel CRC algorithm falls somewhat short of its ideal speedup in real VLSI implementations, it still promises to help communications circuitry keep pace with even faster transmission technologies. Through extrapolation from a successful prototype design, we showed that data rates of many Gb/sec are possible using current CMOS technology. By modeling many of the tradeoffs involved in such a design, we have shown how an optimal circuit can be achieved for a wide range of applications and technologies. In fact, an automatic program for designing CRC circuits across such a range appears to be quite feasible, due to the regular nature of the circuit and the existence of these simple models.

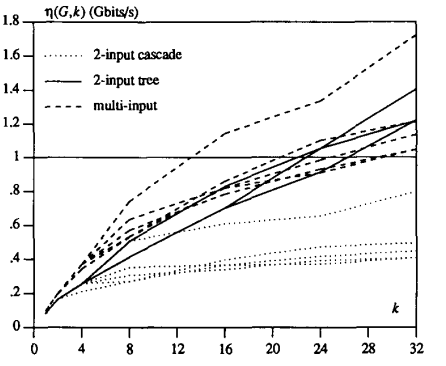

Fig. 8. Throughput estimates for different G, k, and exclusive-or design style.

References

[1] A. M. Patel, "A multichannel CRC register," in *Proc. AFIPS Conf.*, vol. 38, pp. 11–14, Spring 1971.
[2] A. W. Maholick and R. B. Freeman, "A universal cyclic division circuit," in *Proc. AFIPS Conf.*, vol. 39, pp. 1–8, Fall 1971.
[3] D. W. Davies, D. L. A. Barber, W. L. Price, and C. M. Solomonides, *Computer Networks and Their Protocols*. New York: Wiley, 1979, pp. 263–269.
[4] IEEE Computer Society Technical Committee on Computer Communications, *IEEE Standards for Local Area Networks: Carrier Sense Multiple Access with Collision Detection (CSMA/CD): Access Method & Physical Layout Specifications*. New York: Wiley-Interscience, 1985.
[5] W. W. Peterson, *Error Correcting and Error Detecting Codes*. Cambridge, MA: Technology Press, 1962, Appendix C.
[6] M. Shoji, *CMOS Digital Circuit Technology*. Englewood Cliffs, NJ: Prentice-Hall, 1988, p. 90.