High-Speed Parallel CRC Implementation Based on Unfolding, Pipelining, and Retiming

Chao Cheng, *Student Member, IEEE*, and Keshab K. Parhi, *Fellow, IEEE*

***Abstract*—This brief presents a high-speed parallel cyclic redundancy check (CRC) implementation based on unfolding, pipelining, and retiming algorithms. CRC architectures are first pipelined to reduce the iteration bound by using novel look-ahead pipelining methods and then unfolded and retimed to design high-speed parallel circuits. A comparison on commonly used generator polynomials between the proposed design and previously proposed parallel CRC algorithms shows that the proposed design can increase the speed by up to 25% and control or even reduce hardware cost.**

***Index Terms*—Cyclic redundancy check (CRC), linear feedback shift register (LFSR), pipelining, retiming, unfolding.**

I. Introduction

Cyclic redundancy check (CRC) is widely used to detect errors in data communication and storage devices. When high-speed data transmission is required, the general serial implementation cannot meet the speed requirement. Since parallel processing is a very efficient way to increase the throughput rate, parallel CRC implementations have been discussed extensively in the past decade [1], [2].

Although parallel processing increases the number of message bits that can be processed in one clock cycle, it can also lead to a long critical path (CP); thus, the increase of throughput rate that is achieved by parallel processing will be reduced by the decrease of circuit speed. Another issue is the increase of hardware cost caused by parallel processing, which needs to be controlled. This brief addresses these two issues of parallel CRC implementations.

In [1] and [2], recursive formulas have been developed for parallel CRC hardware computation based on mathematical deduction.

They have identical CPs. The parallel CRC algorithm in [2] processes an m-bit message in $(m+k)/L$ clock cycles, where k is the order of the generator polynomial and L is the level of parallelism. However, in [1], m message bits can be processed in m/L clock cycles.

High-speed architectures for parallel long Bose–Chaudhuri–Hocquenghen (BCH) encoders in [3] and [4], which are based on the multiplication and division computations on generator polynomial, are efficient in terms of speeding up the parallel linear feedback shift register (LFSR) structures. They can also be generally used for the LFSR of any generator polynomial. However, their hardware cost is high.

We will show that our proposed design can achieve shorter CP, which leads to a parallel CRC circuit with higher processing speed, and can control or reduce the hardware cost of parallel processing, with regard to the most commonly used CRC generator polynomials.

The proposed design starts from LFSR, which is generally used for serial CRC. An unfolding algorithm [5] is used to realize parallel processing. However, direct application of unfolding may lead to a parallel CRC circuit with long iteration bound, which is the lowest achievable CP [5]. Two novel look-ahead pipelining methods are developed to reduce the iteration bound of the original serial LFSR CRC structures; then, the unfolding algorithm is applied to obtain a parallel CRC structure with low iteration bound. The retiming algorithm is then applied to obtain the achievable lowest CP.

This brief is organized as follows. Section II analyzes the effect of unfolding on the iteration bound of a parallel CRC circuit. Our proposed pipelining algorithms are discussed in Section III. Retimed and unfolded LFSR CRC circuits are compared in Section IV.

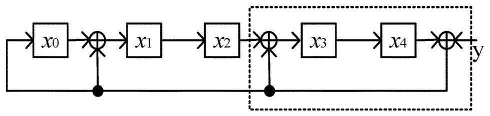

Fig. 1. CRC architecture for $G(y) = 1 + y + y^3 + y^5$.

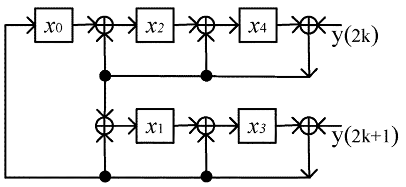

Fig. 2. Two-parallel CRC architecture for $G(y) = 1 + y + y^3 + y^5$.

II. Unfolding CRC Circuit

We start with the simple example in [1]. Consider the CRC architecture with the generator polynomial $G(y) = 1 + y + y^3 + y^5$, which is shown in Fig. 1. After applying an unfolding algorithm [5] with unfolding factors $J = 2$ and $J = 3$, we obtain the two-parallel and three-parallel architectures shown in Figs. 2 and 3, respectively.

Iteration bound is defined as the maximum of all the loop bounds. Loop bound is defined as t/w, where t is the computation time of the loop and w is the number of delay elements in the loop [5].

Manuscript received May 31, 2005; revised November 28, 2005 and March 9, 2006. This paper was recommended by Associate Editor C.-T. Lin.

The authors are with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: chao@ece.umn.edu; parhi@ece.umn.edu).

Digital Object Identifier 10.1109/TCSII.2006.882213

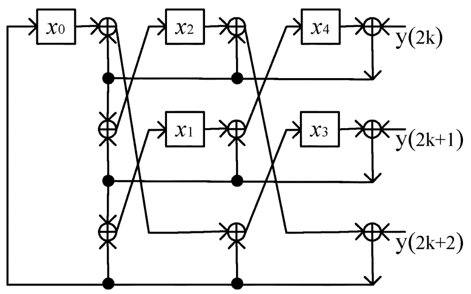

Fig. 3. Three-parallel CRC architecture for $G(y) = 1 + y + y^3 + y^5$.

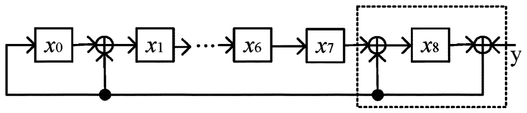

Fig. 4. CRC architecture for $G(y) = 1 + y + y^8 + y^9$.

It is obvious that the iteration bounds for the CRC architectures in Figs. 1–3 are $T_{\rm XOR}$, $2T_{\rm XOR}$, and $3T_{\rm XOR}$, respectively, where $T_{\rm XOR}$ is the computation time of an XOR gate. In this case, the iteration bound of a J-parallel CRC architecture for $G(y) = 1 + y + y^3 + y^5$ is $J \cdot T_{\rm XOR}$. Although we can use retiming to reduce the CP of a circuit, we cannot achieve a CP with a computation time that is less than the iteration bound of this circuit. In other words, the CP of a parallel CRC architecture cannot be less than $J \cdot T_\infty$, where J is the level of parallelism and T_∞ is the iteration period bound of the original data flow graph (DFG) [5]. Therefore, it is very important to reduce the iteration bound before the unfolding algorithm is applied.

III. Pipelining to Reduce Iteration Bound

A. *Proposed Look-Ahead Pipelining (Method 1)*

In this section, we propose a look-ahead pipelining algorithm to reduce the iteration bound of the CRC architecture.

The CRC architecture with the generator polynomial $G(y) = 1+y+y^8+y^9$ is shown in Fig. 4. Its iteration bound is $2T_{\rm XOR}$, which corresponds to the section in the dashed square and the term $y^8 + y^9$ in the generator polynomial. The largest iteration bound of a general serial CRC architecture is also $2T_{\rm XOR}$. For example, the serial architectures of commonly used generator polynomials CRC-16 and CRC-12 have the iteration bound of $2T_{\rm XOR}$ because they have terms $y^{15}+y^{16}$ and $y^{11}+y^{12}$ in their generator polynomials, respectively.

The critical loop with the delayed input redrawn in Fig. 5 is described by

$$a(n+1) = a(n) + y(n) + b(n+1). \tag{1}$$

If we apply look-ahead pipelining to (1), we can obtain a two-level pipelined version given by

$$\begin{aligned} a(n+2) &= a(n+1) + y(n+1) + b(n+2) \\ &= a(n) + b(n+1) + b(n+2) \\ &\quad + y(n) + y(n+1). \end{aligned} \tag{2}$$

The architecture for (2) is shown in Fig. 6. In Fig. 6, we can see that the loop bound in Fig. 5 has been reduced from $2T_{\rm XOR}$ to $T_{\rm XOR}$ at the cost of two XOR gates and two flip-flops.

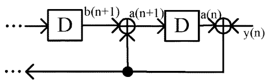

Fig. 5. Loop bound of $2T_{\rm XOR}$.

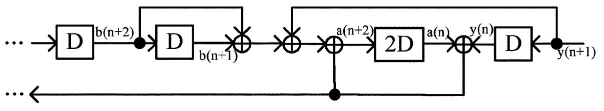

Fig. 6. Pipelined loop with a loop bound of $T_{\rm XOR}$.

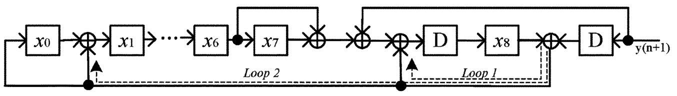

Fig. 7. Two-level pipelined CRC architecture for $G(y) = 1 + y + y^8 + y^9$.

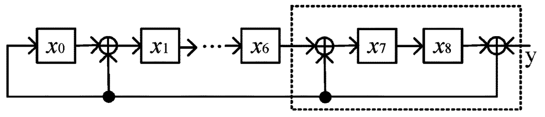

Fig. 8. CRC architecture for $G(y) = 1 + y + y^7 + y^9$.

The CRC architecture in Fig. 4 can now be pipelined as shown in Fig. 7. In Fig. 7, we can see that the loop bounds of loop 1 and loop 2 are $T_{\rm XOR}$ and $(5/8)T_{\rm XOR}$, respectively. So, the iteration bound of the two-level pipelined CRC architecture is $T_{\rm XOR}$.

If we apply higher levels of pipelining, as shown in (2), the loop bound of loop 1 will decrease; however, that of loop 2 will increase. For example, for three-level and four-level pipelining, the loop bound of loop 1 is $(2/3)T_{\rm XOR}$ and $(1/2)T_{\rm XOR}$, respectively; however, that of loop 2 is $(7/8)T_{\rm XOR}$ and $(9/8)T_{\rm XOR}$, respectively. Therefore, the iteration bounds of three-level and four-level pipelined CRC architectures are $(7/8)T_{\rm XOR}$ and $(9/8)T_{\rm XOR}$, respectively.

We can also see from this example that increasing the pipelining level will not necessarily reduce the iteration bound of the CRC architecture. For this example, the lowest iteration bound we can achieve is $(7/8)T_{\rm XOR}$, when the pipelining level is three.

B. *Improved Look-Ahead Pipelining (Method 2)*

Consider the inner loop marked by the dashed square in Fig. 8. This loop can be redrawn as shown in Fig. 9 and can be represented by

$$a(n+2) = a(n) + y(n) + b(n+2). \tag{3}$$

If we apply look-ahead pipelining to (3), we can obtain a four-level pipelined realization of (3) as

$$\begin{aligned} a(n+4) &= a(n+2) + y(n+2) + b(n+4) \\ &= a(n) + b(n+2) + b(n+4) \\ &\quad + y(n) + y(n+2). \end{aligned} \tag{4}$$

The architecture for (4) is shown in Fig. 10. In Fig. 10, we can see that four-level pipelining can be achieved at the cost of two XOR gates and two flip-flops if the there are two delay elements in the initial loop. However, in the proposed pipelining method

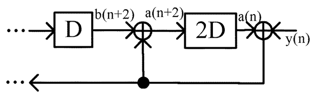

Fig. 9. Loop bound of T_{XOR}.

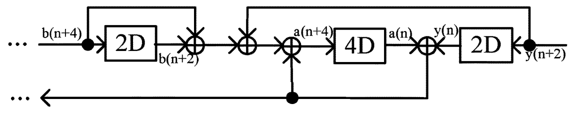

Fig. 10. Pipelined loop with a loop bound of $(1/2)T_{\text{XOR}}$.

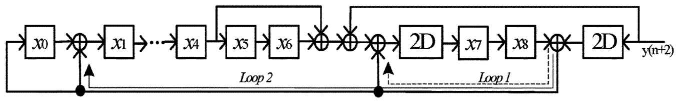

Fig. 11. Four-level pipelined CRC architecture for $G(y) = 1 + y + y^7 + y^9$.

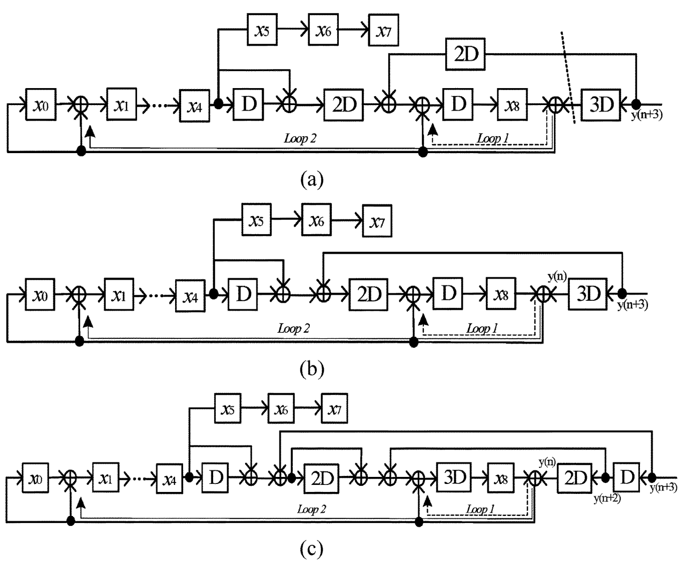

Fig. 12. (a) and (b) Retiming and pipelining process. (c) Improved four-level pipelined CRC architecture for $G(y) = 1 + y + y^8 + y^9$.

1, four-level pipelining can be achieved at the cost of four XOR gates and four flip-flops.

The CRC architecture in Fig. 8 can be pipelined as shown in Fig. 11. In Fig. 11, we can see that the loop bounds of loop 1 and loop 2 are $(1/2)T_{\text{XOR}}$ and $(5/8)T_{\text{XOR}}$, respectively. So, the iteration bound of the four-level pipelined CRC architecture is $(5/8)T_{\text{XOR}}$. Use of four-level pipelining reduces the iteration bound of Fig. 8 from T_{XOR} to $(5/8)T_{\text{XOR}}$ at the cost of two XOR gates and two delay elements. The efficiency of the improved pipelining method will be demonstrated in the succeeding discussions of this brief.

Loop 1 in Fig. 7 can be represented as shown in Fig. 9 and transformed as shown in Fig. 12, based on retiming and pipelining [5].

If we perform four-level pipelining to Fig. 4 by the proposed pipelining method 2, as shown in Fig. 12, we need four extra XOR gates in loop 2, instead of six, as needed by method 1. Thus, the loop bound of loop 2 is reduced from $(9/8)T_{\text{XOR}}$ to $(7/8)T_{\text{XOR}}$, and the iteration bound is also reduced from $(9/8)T_{\text{XOR}}$ to $(7/8)T_{\text{XOR}}$; two XOR gates are saved.

TABLE I
DATA FLOW OF FIG. 12(c) WHEN THE INPUT MESSAGE IS 101011010

Clock #	y(n+3)	y(n+2)	y(n)	x0	x1	x2	x3	x4	x5	x6	x7	x8
1	1	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0	0
4	0	1	1	1	1	0	0	0	0	0	0	1
5	1	0	0	1	0	1	0	0	0	0	0	1
6	1	1	1	0	1	0	1	0	0	0	0	0
7	0	1	0	0	0	1	0	1	0	0	0	0
8	1	0	1	1	1	0	1	0	1	0	0	1
9	0	1	1	0	1	1	0	1	0	1	0	0
10	0	0	0	0	0	1	1	0	1	0	1	0
11	0	0	1	1	1	0	1	1	0	1	0	0
12	0	0	0	0	1	1	0	1	1	0	1	0

The following example illustrates the correctness of the proposed pipelining method 2: For the CRC architecture in Fig. 12(c), let the message sequence be 101011010; Table I shows the data flow at the marked points of this architecture at different time slots.

In Table I, we can see that the achieved four-level pipeline introduces a latency of three clock cycles. However, this is not a drawback, when the message bits are long.

IV. APPLYING RETIMING FOR MINIMUM CP

After we apply pipelining to the original serial CRC architecture, the minimum achievable CP (iteration bound) of the unfolded CRC architecture is reduced. In this section, we carry out retiming for minimum CP to obtain fast parallel CRC architectures by using the example in Fig. 12(c).

Applying three-parallel unfolding to Fig. 12(c), we obtain the design in Fig. 13, where all the numbered nodes ①, ②, ... represent XOR gates.

If the input sequence is 101011010, we can get the data flow of Fig. 13, as shown in Table II. We can see that four clock cycles are needed to encode a 9-bit message.

It is obvious that the CP of Fig. 13 is $5T_{\text{XOR}}$. After we apply retiming to it, its CP can be reduced to $3T_{\text{XOR}}$, which is shown in Fig. 14.

If the input sequence is still 101011010, we can get the data flow of Fig. 14, as shown in Table III.

Compared with Table II, Table III shows a latency of one more clock cycle caused by retiming. One may be led to believe that three-parallel design does not process 3 bits of the message efficiently because encoding a 9-bit message requires five clock cycles. This is because we are just giving a simple example to illustrate our design. In real applications, the message length will be much longer than 9 bits. For example, if the message is 90 bits long, the CRC architecture in Fig. 14 will take $30 + 2 = 32$ clock cycles to encode it. It is obvious that 90/32 is very close to 3.

The example we used in Sections III and IV is not the best solution for a three-parallel implementation of $G(y) = 1 + y + y^8 + y^9$. In Fig. 14, we can see that the given solution requires 21 XOR gates to achieve a CP of $3T_{\text{XOR}}$ by four-level pipelining. However, two-level pipelining would be enough to get a CP of $3T_{\text{XOR}}$, which leads to a three-parallel solution requiring only 15 XOR gates. The preceding example was only used to illustrate

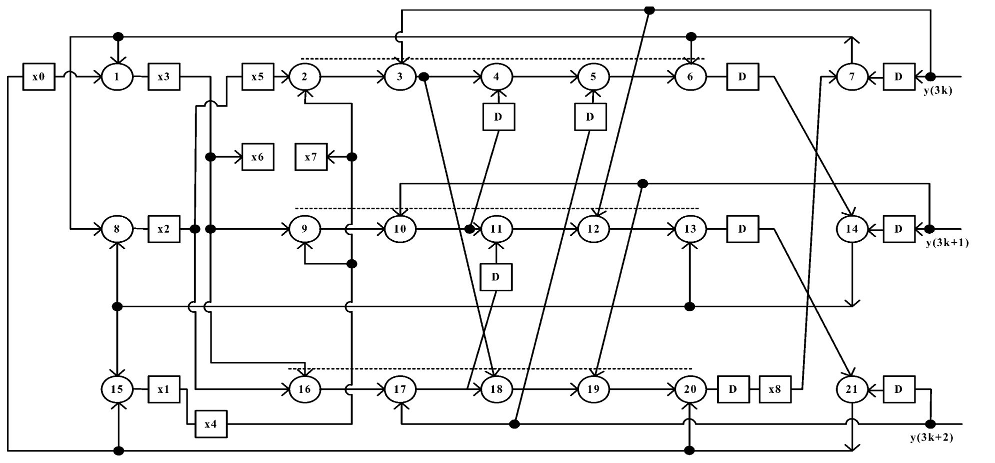

Fig. 13. Three-parallel CRC architecture for Fig. 12(c) after unfolding.

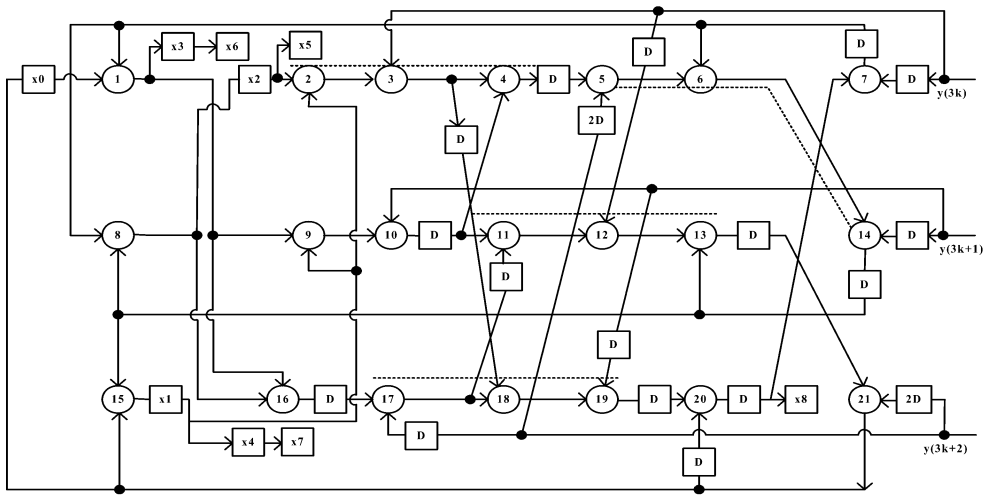

Fig. 14. Retimed three-parallel CRC architecture for Fig. 12(c).

TABLE II
DATA FLOW OF FIG. 13 WHEN THE INPUT MESSAGE IS 101011010

Clock #	y(3k)	y(3k+1)	y(3k+2)	x0	x1	x2	x3	x4	x5	x6	x7	x8
1	1	0	1	0	0	0	0	0	0	0	0	0
2	0	1	1	0	1	0	1	0	0	0	0	0
3	0	1	0	0	1	1	0	1	0	1	0	0
4	0	0	0	0	1	1	0	1	1	0	1	0

TABLE III
DATA FLOW OF FIG. 14 WHEN THE INPUT MESSAGE IS 101011010

Clock #	y(3k)	y(3k+1)	y(3k+2)	x0	x1	x2	x3	x4	x5	x6	x7	x8
1	1	0	1	0	0	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0	0	0
3	0	1	0	0	1	0	1	0	0	0	0	0
4	0	0	0	0	1	1	0	1	0	1	0	0
5	0	0	0	0	1	1	0	1	1	0	1	0

the complete procedure of how to apply our proposed design based on method 2.

V. COMPARISON AND ANALYSIS

We focus our discussion on the commonly used generator polynomials [6] that are shown in Table IV. By observing these

TABLE IV
COMMONLY USED GENERATOR POLYNOMIALS

CRC-12	$y^{12}+y^{11}+y^{3}+y^{2}+y+1$
CRC-16	$y^{16}+y^{15}+y^{2}+1$
SDLC (CCITT)	$y^{16}+y^{12}+y^{5}+1$
CRC-16 REVERSE	$y^{16}+y^{14}+y^{1}+1$
SDLC REVERSE	$y^{16}+y^{11}+y^{4}+1$
CRC-32	$y^{32}+y^{26}+y^{23}+y^{22}+y^{16}+y^{12}+y^{11}$ $+y^{10}+y^{8}+y^{7}+y^{5}+y^{4}+y^{2}+y+1$

polynomials, we can see that they all have many zero coefficients between the second and third highest order nonzero coefficients. This property is also applicable to the polynomial example we used in Sections III and IV for $G(y)=1+y+y^8+y^9$.

A comparison between previous high-speed CRC architectures and the proposed ones is shown in Table V for different parallelism levels of commonly used CRC generator polynomials. Tree structure is applied to further reduce the CP to $O(\log(\text{CP}))$ at the cost of some additional XOR gates.

In Table V, we can see that for all the listed commonly used generator polynomials, our design can maintain or reduce the

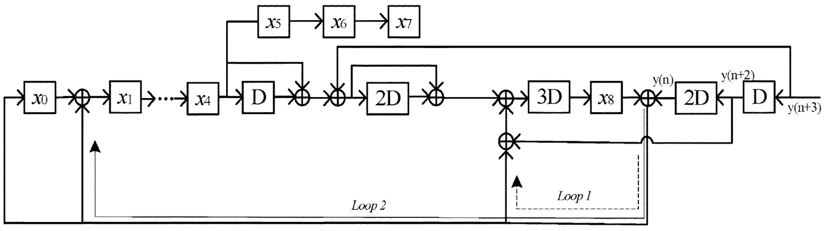

Fig. 15. Further improved four-level pipelined CRC architecture for $G(y) = 1 + y + y^8 + y^9$.

TABLE V

COMPARISON BETWEEN THE PREVIOUS HIGH-SPEED CRC ARCHITECTURES AND THE PROPOSED ONES IN TERMS OF REQUIRED TOTAL NUMBER OF XOR GATES (RXOR), REQUIRED NUMBER OF CYCLES FOR THE M-BIT MESSAGE (RC), REQUIRED DELAY ELEMENTS (RD), AND CRITICAL PATH (CP) (REPRESENTED AS THE NUMBER OF XOR GATES BEFORE AND AFTER THE APPLICATION OF THE TREE STRUCTURE) FOR DIFFERENT PARALLELLISM LEVELS (PL)

CRC (P.L.)	Algorithms	R.D	R.C. Per m-bits	Before Tree Stru.		After Tree Stru.	
				R.XOR	C.P. (# of XOR's)	R.XOR	C.P. (# of XOR's)
CRC-12* (12)	[1]	12	m/12	52	10	52	4
	[2]	12	(m+12)/12	52	10	52	4
	Proposed Pipe_L=4	20	(m+3)/12	108	9	131	4
CRC-16 (16)	[1]	16	m/16	72	15	72	4
	[2]	16	(m+16)/16	72	15	72	4
	Proposed Pipe_L=4	28	(m+3)/16	112	8	137	4
SDLC* (CCITT) (16)	[1]	16	m/16	88	8	88	4
	[2]	16	(m+16)/16	88	8	88	4
	Proposed Pipe_L=8	26	(m+4)/16	80	6	87	3
CRC-16 REVERSE (16)	[1]	16	m/16	154	15	154	4
	[2]	16	(m+16)/16	154	15	154	4
	Proposed Pipe_L=4	23	(m+2)/16	80	8	91	4
SDLC* REVERSE (16)	[1]	16	m/16	84	8	84	4
	[2]	16	(m+16)/16	84	8	84	4
	Proposed Pipe_L=10	26	(m+5)/16	80	6	83	3
CRC-32 (32)	[1]	32	m/32	452	17	452	5
	[2]	32	(m+32)/32	452	17	452	5
	Proposed Pipe_L=6	35	m/32	448	15	467	4

CP. In addition to the fact that our design can process an m-bit message within almost the same number of clock cycles as previous designs, our design can speed up the throughput rate of CRC architectures and control or even reduce the hardware cost. Our design requires more XOR gates for some generator polynomials and saves XOR gates for others. Our design also requires a small number of additional delay elements. In Table V, no optimization techniques such as identifying common subexpressions [2] are applied.

Note that as shown in Fig. 12, when the pipelining level increases, the loop bound of loop 2 will become larger than that of loop 1. Increasing the pipelining level will increase the loop bound of loop 2 and thus increase the iteration bound. The iteration bound of Fig. 12(c) can be further reduced if we move the XOR gate of $y(n+2)$ from loop 2 to loop 1, which is shown in Fig. 15.

In Fig. 15, we can see that although moving the addition operation of $y(n+2)$ increases the loop bound of loop 1 from $0.5T_{\text{XOR}}$ to $(3/4)T_{\text{XOR}}$, that of loop 2 is reduced to $(3/4)T_{\text{XOR}}$, and the iteration bound is also reduced from $(7/8)T_{\text{XOR}}$ to $(3/4)T_{\text{XOR}}$. This iteration bound reduction method is also applied to the parallel implementation of the commonly used generator polynomials shown in Table V and marked with $*$.

Our design can also be applied to other LFSR architectures, especially those with a generator polynomial that has many zero coefficients between the second and third highest order nonzero coefficients.

VI. CONCLUSION

This brief has proposed two pipelining methods for high-speed parallel CRC hardware implementation. Method 1 has a simpler structure, while method 2 has better performance. Furthermore, method 1 is useful as a preprocessing step before method 2 can be applied. Our parallel CRC design can efficiently reduce the CP and control or decrease the required hardware at the same time. Although the proposed design is not efficiently applicable for the LFSR architecture of any generator polynomial, it is very efficient for the generator polynomials with many zero coefficients between the second and third highest order nonzero coefficients, as shown in the commonly used generator polynomials.

REFERENCES

[1] T.-B. Pei and C. Zukowski, "High-speed parallel CRC circuits in VLSI," *IEEE Trans. Commun.*, vol. 40, no. 4, pp. 653–657, Apr. 1992.

[2] G. Campobello, G. Patané, and M. Russo, "Parallel CRC realization," *IEEE Trans. Comput.*, vol. 52, no. 10, pp. 1312–1319, Oct. 2003.

[3] K. K. Parhi, "Eliminating the fanout bottleneck in parallel long BCH encoders," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 3, pp. 512–516, Mar. 2004.

[4] X. Zhang and K. K. Parhi, "High-speed architectures for parallel long BCH encoders," in *Proc. ACM Great Lakes Symp. VLSI*, Boston, MA, Apr. 2004, pp. 1–6.

[5] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. Hoboken, NJ: Wiley, 1999.

[6] T. V. Ramabadran and S. S Gaitonde, "A tutorial on CRC computations," *IEEE Micro*, vol. 8, no. 4, pp. 62–75, Aug. 1988.