# A Systematic Approach for Parallel CRC Computations

MING-DER SHIEH, MING-HWA SHEU, CHUNG-HO CHEN[*]
AND HSIN-FU LO
*Department of Electronic Engineering*
*National Yunlin University of Science and Technology*
*Yunlin, Taiwan 640, R.O.C.*
*E-mail: shiehm@cad.el.yuntech.edu.tw*
[*]*Department of Electrical Engineering*
*National Cheng-Kung University*
*Tainan, Taiwan 701, R.O.C.*

Cyclic redundancy codes (CRCs) form a powerful class of codes suited especially for the detection of burst errors in data storage and communication applications. In the traditional hardware implementation, a simple shift-register-based circuit performs the computation by handling the data one bit at a time. Parallel implementation can perform the necessary logic operations much faster than the serial implementation, therefore, it is very suitable to be applied in today's high-speed systems employing CRC checking. In this paper, we describe the ways toward accomplishing two types of circuit design for parallel CRC computations. Our approach is to systematically decompose the original input message into a set of subsequences based on the theory of Galois field. Parallel CRC computations can then be achieved by inputting those subsequences at the same time and employing the lookahead technique for those subsequences to speedup computation. The resulting hardware implementations are very flexible with the characteristics of modular design and scalable structure to fulfill different levels of parallelism in a single circuit for parallel CRC computation.

*Keywords:* parallel cyclic redundancy code (CRC) computation, Galois field, linear feedback shift register (LFSR), VLSI design, error control code

## 1. INTRODUCTION

Cyclic redundancy codes (CRCs) [1, 2], widely used in data communications and storage devices, provide some attractive properties to handle errors, especially for the detection of burst errors. In those applications employing CRC checking, a frame check sequence (FCS), generated by using CRC, is appended at the end of each message during transmission and the integrity of the transmitted message with the associated FCS is checked by the receiver for proper transmission. Generally, CRC implementations can use either hardware or software methods [3, 4]. Software implementations of CRC encoding/decoding do not resort to dedicated hardware requirements; however, their applicability is limited to lower encoding rates. In traditional hardware implementations, a simple shift register associated with a specific exclusive-OR (XOR) circuit, which is also known as a linear feedback shift register (LFSR), performs the computations by handling

Received July 26, 1999; revised October 25, 1999; accepted January 4, 2000.
Communicated by Youn-Long Lin.

the data one bit at a time. As the demands of high transmission rate requirement in various applications, the need of a simple and systematic method to rapidly calculate the CRC becomes a practical issue from both industrial and academic points of views.

Parallel CRC implementation can perform the necessary logic operations much faster than conventional serial implementation. In [4-6], the problem of parallel implementations is solved by empirically emulating the state transitions in the shift register during a fixed length *r* of incoming serial data. In other words, they investigated how the content of a particular linear feedback shift register is modified after *r* clock pulses. Then, either a lookup ROM table or a XOR tree is constructed for the *r* input data by assuming that they are accessed in parallel. Albertengo [7] designed the parallel CRC encoders based on digital system theory and *z*-transforms, which allows designers to derive the logic equations of the parallel encoder circuit for any generator polynomial. Glaise [8] used the theory of Galois Fields $GF(2^m)$ and employed a *GF* multiplier for parallel CRC calculation. By inspecting the operations of the single-input LFSR, Pei [9] derived the state transition equation for the parallel CRC circuit by merging a number of the shift and modulo-2 (XOR) operations together within a single clock cycle. Basically, the main concept is also to emulate the state transitions in the shift register during a fixed length of serial data. Based on the presented prototype CRC circuit, the authors analyzed the potential speedup and hardware overhead for seven polynomial generators with different *r*-values. In [10], a 32-bit parallel CRC engine coping with the layout routing and optimization of the XOR tree is presented assuming that the state transition equations with 32 parallel inputs are given for the CRC polynomial [11].

In this paper, we present a simple but systematic approach for parallel CRC computation based on the theory of Galois field and use the lookahead technique to accomplish circuit design. First, the original input message systematically decomposed into a set of subsequences based on properties of the finite field. Parallel CRC computations are then achieved by inputting those subsequences at the same time and employing the lookahead technique to skip the introduced consecutive zeros within those subsequences to speedup computation. Compared with the previous works, our approach is very flexible in terms of design methodology and applications, and is suitable for modular design in VLSI implementation. In particular, the developed architecture and the resulting implementation can be easily extended to fulfill different levels of parallelism in a single circuit for parallel CRC computation. And, these properties are very useful when applied in the intellectual property (IP) design paradigm.

The paper is organized as follows. In Section 2, we briefly review the theory of finite field and CRC computation related to this work. Then, the ways toward accomplishing two types of circuit design for parallel CRC computations based on the theory of Galois field and the lookahead technique are derived in Section 3. Section 4 describes the corresponding hardware implementation and evaluation of our development. Finally, we conclude this paper in Section 5.

## 2. FINITE FIELD AND CRC BACKGROUND

Let the original message $\boldsymbol{M} = (m_0, m_1, \ldots, m_{k-1})$ contain *k* digits and *n* represent the length of a linear block code $\boldsymbol{C} = (c_0, c_1, \ldots, c_{n-1}) \in \mathbf{C^*}$. An $(n, k)$ linear code $\mathbf{C^*}$ is called a cyclic code if every cyclic shift of a code vector in **C*** is also a code vector of itself [1, 2]. In general, the basic properties of cyclic codes can be derived through manipulations of the polynomial representations of ***M*** and ***C***. Specifically, if $M(x) = m_0 + m_1x + \dots + m_{k-1}x^{k-1}$ is the message polynomial associated with ***M*** and the code polynomial $C(x) = c_0 + c_1x + c_2x^2 + \dots + c_{n-1}x^{n-1}$, then the properties of a cyclic code are characterized by the associated generator polynomial $G(x)$ of degree $n$-$k$, i.e., $G(x) = 1 + g_1x + \dots + g_{n-k-1}x^{n-k-1} + x^{n-k}$. It follows that every code polynomial $C(x)$ in an ($n$, $k$) cyclic code can be expressed as $C(x) = M(x)\,G(x)$.

Cyclic redundancy codes (CRCs), also known as shortened cyclic codes, take advantage of the considerable burst-error detection capability provided by cyclic codes. In practical applications, the code word is usually arranged in the systematic form that is achieved based on the equation $C(x) = x^{n-k}M(x) + S(x)$, where $S(x)$ is the remainder or syndrome obtained through dividing $x^{n-k}M(x)$ by $G(x)$. In this way, the rightmost $k$ digits of each code vector are the unaltered information digits and the leftmost $n$-$k$ digits are parity-check digits. A systematic code thus corresponds to the code vector ($s_0$, $s_1$, …, $s_{n-k-1}$, $m_0$, $m_1$, …, $m_{k-1}$), which can be accomplished with a division circuit. The division circuit is a linear ($n$-$k$)-stage shift register with feedback connections based on $G(x)$.

The general classes of cyclic codes can be implemented using shift-register-based circuits. The only difference between the CRCs and cyclic codes is that CRCs can have arbitrary length up to a limit $N$, the length of the original cyclic code defined by $G(x)$. Fig. 1 conceptually depicts the encoding of an ($n$, $k$) cyclic code or CRC in the systematic form, where the storage elements (D flip-flops) are connected in the form of a shift register and the symbol ⊕ denotes a modulo-2 adder (XOR gate). A feedback connection exists if the corresponding coefficient $g_i$ of $G(x)$ is one. Shifting the message $M(x)$ into the circuit from the right end is equivalent to premultiplying $M(x)$ by $x^{n-k}$. As soon as the complete message has entered the circuit with switches $S_1$ and $S_3$ closed and $S_2$ connected to point $a$, the $n - k$ digits in the register form the remainder and thus they are the parity-check digits. The contents in the register are then shifted out with switches $S_1$ and $S_3$ opened and $S_2$ connected to point $b$. If the flip-flops are initialized to other values instead of zeros, the only change required in the system is in the remainder that the receiver is looking for.

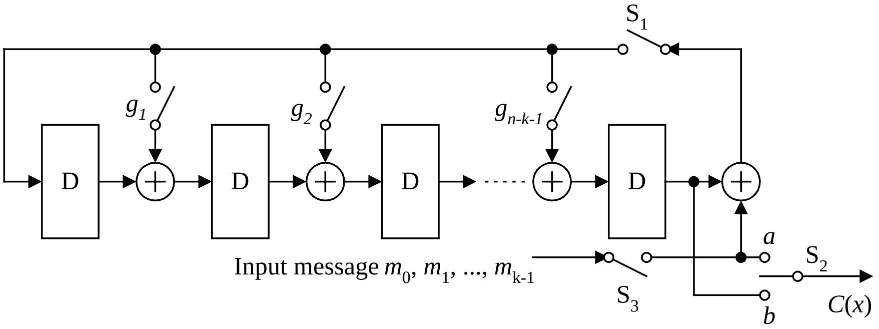


Fig. 1. Encoding of an ($n$, $k$) cyclic code based on the generator polynomial $G(x)$.

It is also possible to design a systematic encoder based on the coefficients of the code's parity polynomial $H(x) = 1 + h_1x + \ldots + h_{k-1}x^{k-1} + x^k$, where $H(x)$ is derived based on the equation $G(x)H(x) = x^n + 1$. As shown in [1], multiplying $C(x)$ by $H(x)$, we obtain

$$C(x)H(x) = M(x)G(x)H(x) = M(x)(x^n + 1) = M(x) + x^nM(x). \tag{1}$$

Because the degree of $M(x)$ is $k - 1$ or less, the coefficients of $x^k, x^{k+1}, \ldots, x^{n-1}$ in $C(x)H(x)$ should be equal to zero. As a result, the following $n$-$k$ equalities can be derived:

$$\sum_{i=0}^{k} h_i c_{n-i-j} = 0\text{, for } 1 \le j \le n\text{-}k \text{ and } h_0 = h_k = 1. \tag{2}$$

Since $h_k$ =1, the equalities of (2) can be rewritten as

$$c_{n-k-j} = \sum_{i=0}^{k-1} h_i c_{n-i-j}\text{, for } 1 \le j \le n\text{-}k \text{ and } h_0 = 1. \tag{3}$$

For a systematic code, the high-order codeword coordinates $c_i$ are provided by the message $M(x)$, i.e., $c_i = m_{k+i-n}$ for $n$-$k \le i \le n$-1, and the remaining coordinates are then computed recursively using equation (3). Fig. 2 shows the encoding of an $(n, k)$ cyclic code in systematic form using the parity polynomial $H(x)$, where the feedback connection is determined by the coefficient $h_i$ of $H(x)$. The operation consists of two major steps. Step I: The meassage $M(x)$ is shifted into the $k$ storage elements with switches $S_2$ opened and $S_1$ closed. Step II: The remaining ($n$-$k$) coordinates are then generated by using equation (3) with switches $S_2$ closed and $S_1$ opened.

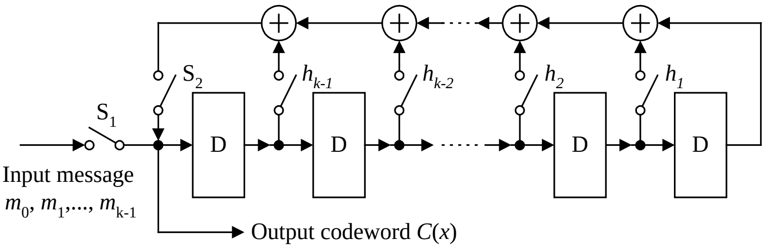


Fig. 2. Encoding of an $(n, k)$ cyclic code based on the parity polynomial $H(x)$.

Comparing the two encoding circuits, it can be concluded that for codes with more parity-check digits than the message digits, the $k$-stage encoding circuit (Fig. 2) is more economical; otherwise the ($n$-$k$)-stage encoding circuit (Fig. 1) is preferable. For current CRC applications, the ($n$-$k$)-stage encoding circuit is commonly applied because that the number of parity-check digits is much smaller than the number of message digits. For completeness, we investigate these two types of implementation for parallel CRC calculation in this paper. And, Figs. 1 and 2, respectively, are referred to as the Type-I and Type-II implementations in our development.

## 3. DEVELOPED METHODS FOR PARALLEL CRC COMPUTATIONS

In this section, we present two types of design methodologies for parallel CRC computations based on the theory of Galois field and the lookahead technique. It is assumed that (1) the length of the input message ***M*** is *k* bits, (2) the generator polynomial $G(x)$ is of degree ($n$-$k$), and (3) *r* bits of ***M*** are to be encoded/decoded simultaneously. On the communication channel, the first message bit is assumed to be the coefficient of the highest power of *x*, and successive bits correspond to decreasing powers of *x* (the last bit goes with the zero power of *x*). In practice, the least significant data bit is transmitted first such that it becomes the most significant bit for CRC computations. Without loss of generality, it is assumed that the bit with the highest power of *x* is transmitted first in this paper, i.e., the *k* message bits are fed into the encoder in order of decreasing index.

### 3.1 Parallel CRC Computation Based on the Polynomial Generator *G*(*x*)

For parallel CRC computation based on the Type-I implementation, it is further assumed that (1) the length of the original message is divisible by the specified value *r*, i.e., $k \bmod r = 0$, and (2) the value *r* is less than the degree ($n$-$k$) of the generator polynomial. In practical applications, the first assumption is usually satisfied because multiple bits, e.g. a byte, of the message are received in parallel and encoded/decoded to generate the CRC at the same time. In general, if $k \bmod r \neq 0$, the final CRC (syndrome) can also be derived by setting a proper initial state to eliminate the effect of adding extra *p* zeros such that $(k + p) \bmod r = 0$ when the value *p* is fixed and known in advance. The same concept is also applied in different applications like CRC-32 [11], in which the transmitter and the receiver initialize the remainder value to all 1's before CRC accumulation starts. The second assumption is to take into account the hardware implementation discussed in this paper.

Based on the property of linear code, the original input message $M(x)$ can be decomposed into a linear combination of *r* subsequences and we use the symbol $M_i(x)$ to represent the subsequence *i*. In this way, we can derive the following equation.

$$M(x) = m_0 + m_1x + \ldots + m_{k-1}x^{k-1} = M_0(x) + M_1(x) + \ldots + M_{r-1}(x),$$

$$\text{where } M_i(x) = \sum_{j=0}^{(k/r)-1} m_{j\times r+i}\, x^{j\times r+i} = m_i\, x^i + m_{r+i}x^{r+i} + m_{2r+i}x^{2r+i} + \ldots + m_{k-r+i}x^{k-r+i}. \quad (4)$$

The missing bit $m_q$, $q \notin \{i, r+i, \ldots, k-r+i\}$, can be treated as a zero coefficient in $M_i(x)$. When $M(x)$ is multiplied by $x^{n-k}$, equation (4) becomes

$$\begin{aligned} x^{n-k}M(x) &= x^{n-k}(M_0(x) + M_1(x) + \ldots + M_{r-1}(x)) \\ &= \sum_{i=0}^{r-1} x^{n-k} \sum_{j=0}^{(k/r)-1} m_{j\times r+i}\, x^{j\times r+i} \\ &= \sum_{i=0}^{r-1} x^{n-k-(r-1)+i} \sum_{j=0}^{(k/r)-1} m_{j\times r+i}\, x^{(j+1)\times r-1} \end{aligned}$$

$$= \sum_{i=0}^{r-1} x^{n-k-(r-1)+i}\overline{M}_i(x)$$

where $\overline{M}_i(x) = \sum_{j=0}^{(k/r)-1} m_{j\times r+i}x^{(j+1)\times r-1} = m_i\, x^{r-1} + m_{r+i}\, x^{2r-1} + m_{2r+i}\, x^{3r-1} + \ldots + m_{k-r+i}\, x^{k-1}$. (5)

In other words, the $k$ bits in the original message $M(x)$ is first uniformly distributed into $r$ subsequences, $M_i(x)$ for $0 \leq i \leq (r\text{-}1)$, each consisting of $k/r$ bits. By properly manipulating $x^{\text{n-k}}M(x)$ based on equation (5), the $\overline{M}(x)$ subsequences can then be derived. The salient features of the $\overline{M}(x)$ subsequences are that (a) they have the same polynomial representations, i.e. the combination of the power of $x$ for the possible nonzero bits is the same, and (b) there exists ($r$-1) consecutive zero coefficients between two adjacent message bits as shown in equation (5). Table 1 illustrates the relationship among $M(x)$, $M_i(x)$ and $\overline{M}(x)$ for $k = 16$ and $r = 4$. With the relationship in mind, it is recognized that if $r$ possible nonzero bits with the same power of $x$ in $r$ subsequences of $\overline{M}(x)$ are encoded/decoded in parallel at the same time, then exactly ($r$-1) consecutive zeros exist in each $\overline{M}(x)$ subsequence before the next possible nonzero bit comes in.

**Table 1. The relationship among $M(x)$, $M_i(x)$ and $\overline{M}_i(x)$ for $k$ = 16 and $r$ = 4.**

| degree | $x^{15}$ | $x^{14}$ | $x^{13}$ | $x^{12}$ | $x^{11}$ | $x^{10}$ | $x^{9}$ | $x^{8}$ | $x^{7}$ | $x^{6}$ | $x^{5}$ | $x^{4}$ | $x^{3}$ | $x^{2}$ | $x^{1}$ | $x^{0}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $M(x)$ | $m_{15}$ | $m_{14}$ | $m_{13}$ | $m_{12}$ | $m_{11}$ | $m_{10}$ | $m_9$ | $m_8$ | $m_7$ | $m_6$ | $m_5$ | $m_4$ | $m_3$ | $m_2$ | $m_1$ | $m_0$ |
| $M_3(x)$ | $m_{15}$ | 0 | 0 | 0 | $m_{11}$ | 0 | 0 | 0 | $m_7$ | 0 | 0 | 0 | $m_3$ | 0 | 0 | 0 |
| $M_2(x)$ | 0 | $m_{14}$ | 0 | 0 | 0 | $m_{10}$ | 0 | 0 | 0 | $m_6$ | 0 | 0 | 0 | $m_2$ | 0 | 0 |
| $M_1(x)$ | 0 | 0 | $m_{13}$ | 0 | 0 | 0 | $m_9$ | 0 | 0 | 0 | $m_5$ | 0 | 0 | 0 | $m_1$ | 0 |
| $M_0(x)$ | 0 | 0 | 0 | $m_{12}$ | 0 | 0 | 0 | $m_8$ | 0 | 0 | 0 | $m_4$ | 0 | 0 | 0 | $m_0$ |
| $\overline{M}_3(x)$ | $m_{15}$ | 0 | 0 | 0 | $m_{11}$ | 0 | 0 | 0 | $m_7$ | 0 | 0 | 0 | $m_3$ | 0 | 0 | 0 |
| $\overline{M}_2(x)$ | $m_{14}$ | 0 | 0 | 0 | $m_{10}$ | 0 | 0 | 0 | $m_6$ | 0 | 0 | 0 | $m_2$ | 0 | 0 | 0 |
| $\overline{M}_1(x)$ | $m_{13}$ | 0 | 0 | 0 | $m_9$ | 0 | 0 | 0 | $m_5$ | 0 | 0 | 0 | $m_1$ | 0 | 0 | 0 |
| $\overline{M}_0(x)$ | $m_{12}$ | 0 | 0 | 0 | $m_8$ | 0 | 0 | 0 | $m_4$ | 0 | 0 | 0 | $m_0$ | 0 | 0 | 0 |

Based on the derivation, the syndrome $S(x)$, also referred to as the remainder or parity-check bits, of the CRC can be expressed as

$$\begin{aligned} S(x) &= x^{\text{n-k}}M(x) \bmod G(x) \\ &= \sum_{i=0}^{r-1} x^{n-k-(r-1)+i}\overline{M}_i(x) \bmod G(x) \\ &= (x^{n-k-r+1}\overline{M}_0(x) + x^{n-k-r+2}\overline{M}_1(x) + \ldots + x^{n-k}\overline{M}_{r-1}(x)) \bmod G(x). \qquad (6) \end{aligned}$$

As shown in [1], premultiplying $\overline{M}_\iota(x)$ by $x^{n\text{-}k\text{-}(r\text{-}1)+i}$ can be accomplished by shifting $\overline{M}_\iota(x)$ into the ($r$-1-$i$)$th$ position of the circuit counting from the right end. Fig. 3 shows a possible implementation of the parallel CRC calculation. For clarity, the controlling switches are not shown in the Fig. and they are operated in the same manner as those of the single-input case in Fig. 1. However, the implementation of Fig. 3 does not achieve speed advantage over the one depicted in Fig. 1. In the following, we show how to take advantage of (*r*-1) consecutive zeros in $\overline{M}_t(x)$ and apply the lookahead technique to speedup the parallel CRC computation. The basic concept of the lookahead technique can be stated as follows. When *r* possible nonzero bits with the same power of *x* in *r* subsequences of $\overline{M}_t(x)$ are to be processed in parallel, effects of the aligned (*r*-1) consecutive zeros in each $\overline{M}_t(x)$ subsequence in the following (*r*-1) time steps can be taken into considerations at the same time. As a result, we can look ahead (*r*-1) time steps, merge the operations of *r* consecutive time steps in a single clock cycle and then proceed to the next *r* possible nonzero bits. Theoretically, a potential speedup ratio of *r* can be achieved by employing the lookahead techniques on the *r* subsequences of $\overline{M}_t(x)$.

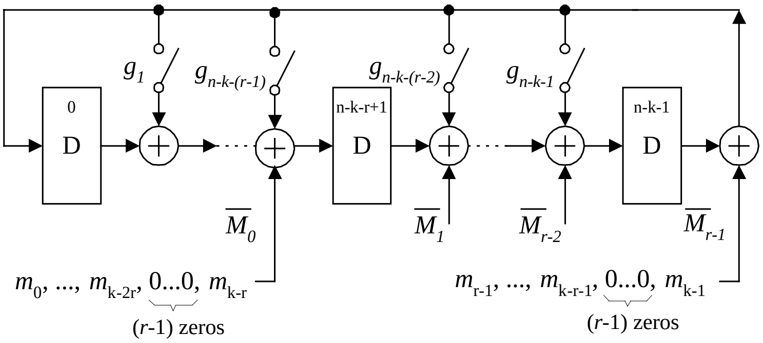


Fig. 3. A possible implementation for parallel CRC computation (Type I).

Let the D flip-flops (in Fig. 3) be labeled from 0 to *n*-*k*-1 starting from the left end and $s_i(t)$ represent the current stored value (state) in the D flip-flop *i* at time step *t*. For all zero inputs from $\overline{M}_i(x)$ at the present time, the LFSR becomes autonomous, i.e., it is equivalent to have no inputs except for clocks, and the next states of the flip-flops can be expressed in matrix form as

$$S(t+1) = \begin{bmatrix} s_0(t+1) \\ s_1(t+1) \\ s_2(t+1) \\ \vdots \\ s_{n-k-2}(t+1) \\ s_{n-k-1}(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & 0 & \cdots & 0 & g_1 \\ 0 & 1 & 0 & \cdots & 0 & g_2 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & g_{n-k-2} \\ 0 & 0 & 0 & \cdots & 1 & g_{n-k-1} \end{bmatrix} \times \begin{bmatrix} s_0(t) \\ s_1(t) \\ s_2(t) \\ \vdots \\ s_{n-k-2}(t) \\ s_{n-k-1}(t) \end{bmatrix} = T_g \times S(t). \qquad (7)$$

The transformation matrix $T_g$ is characterized by the selected generator polynomial $G(x)$ and can be treated as a one-order lookahead matrix for a set of zero inputs coming in at time step *t*. When considering the *r* possible nonzero inputs from $\overline{M}_t(x)$ at the present time step and applying the lookahead technique for the following (*r*-1) time steps, the internal states are updated based on the following equation.

$$S(t+1)=T_g{}^{r-1}(T_g(S(t)\oplus I(t)))=T_g{}^{r}(S(t)\oplus I(t))$$
$$\text{where}\quad I(t)=[0\,|\,m_{k-(t+1)r},m_{k-(t+1)r+1},\cdots,m_{k-(t+1)r+r-1}]^T_{1\times(n-k)}\quad\text{and } 0\le t\le k/r-1.\quad(8)$$

In equation (8), the next state $S(t+1)$ denotes the updated internal states in a single clock cycle because the $r$ message bits with the same weight, i.e. the power of $x$, taking from each $\overline{M}_i(x)$ and the following ($r$-1) consecutive zero vectors are actually processed in a single time step. Throughout this paper, we will use the notation $S(t+1)$ to represent the next state after triggering the clock signal once. The operation $T_g(S(t)\oplus I(t))$ is to take into account the $r$ message bits, while the operation $T_g{}^{r-1}$ is to deal with the foreseen ($r$-1) consecutive zero vectors. It can be seen that starting from time step $t = 0$, the final remainder is derived after $k/r$ clock cycles. A potential speedup factor $r$ can then be achieved in parallel implementation compared with the serial-input counterpart. Based on equation (8), we know exactly the changes of internal states of the flip-flops in the course of parallel CRC computation. Note that similar results can also be found in [9, 10] because the final value of CRC computation should be the same. But, in the paper we show the detailed procedures of parallel CRC computation from a different point of view. The procedure is simpler and extendable for a variety of applications as shown in the following section. Fig. 4 shows the conceptual implementation of parallel CRC computation with the applied lookahead technique. The detailed implementation of the lookahead function block will be described in next section.

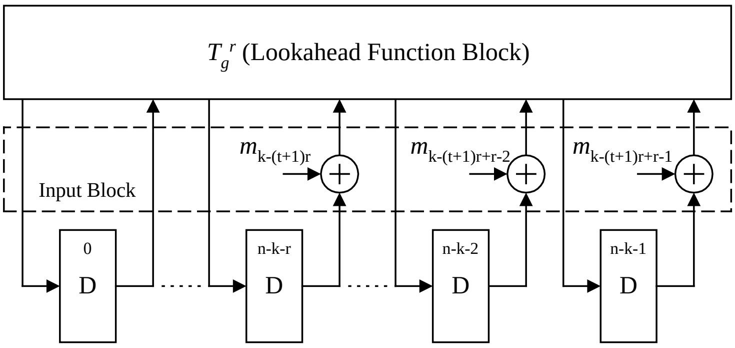


Fig. 4. Implementation of parallel CRC computation (Type I).

### 3.2 Parallel CRC Computation Based on the Parity Generator *H*(*x*)

This subsection presents the parallel CRC computation based on the Type-II implementation. Fig. 5 depicts the simplified block diagram of encoding cyclic code based on the parity polynomial $H(x)$, in which the storage elements are labeled from 0 to $k$-1 starting from the left end and the parameters $s_i$, $0 \le i \le k$-1, represent the internal states of the storage elements. The functional block $T_h$ is characterized by the parity polynomial as described below. For accessing multiple message bits and generating parity-check bits in parallel, the operations of the encoding circuit must accomplish the two major steps as described in the serial-input counterpart. For simplicity of explanation, it is also assumed that the conditions, $k$ mod $r$ = 0 and ($n$-$k$) mod $r$ = 0, are satisfied for parallel CRC computation.

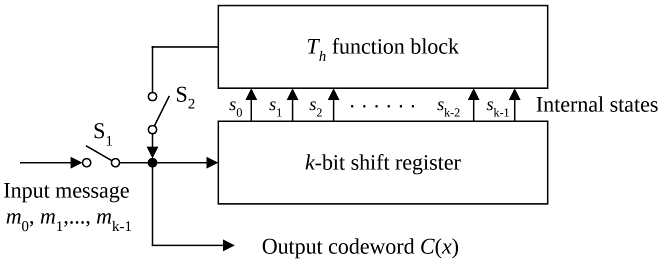


Fig. 5. The simplified block diagram of Type II encoding circuit.

From hardware implementation points of view, if the length of the original message is divisible by $r$, i.e., $k$ mod $r$ = 0, the serial-input shift register of length $k$ can be reconstructed as $r$ single-input shift registers of length $k/r$ for inputting $r$ message bits at the same time. Let the input message $M(x)$ be divided into $r$ subsequences $I_i$ for $0 \le i \le r$-1. The resulting subsequence $I_i$ is similar to $\overline{M}_i(x)$ defined in equation (5) except that we ignore the ($r$-1) consecutive zeros between two adjacent message bits. In other words, the subsequence $I_i$ is represented as ($m_i$, $m_{r+i}$, $m_{2r+i}$, …, $m_{k-r+i}$) and the $r$ bits are taken from $r$ subsequences, one bit at a time from the higher-order to lower-order bits of each subsequence $I_i$.

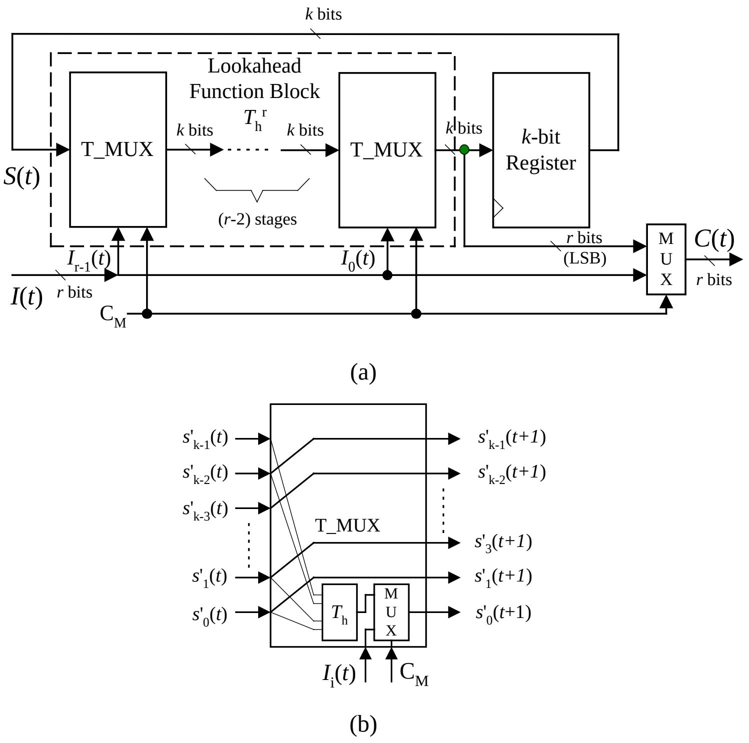
After the registers are fully loaded with the $k$ message bits, the controlling switches change states and the parity-check bits are then generated based on the parity polynomial $H(x)$ of the autonomous LFSR, i.e., it has no inputs except for clocks. If the parity-check bits are generated one at a time, then the next states of the flip-flops can be updated based on the following equation

$$S(t+1) = \begin{bmatrix} s_0(t+1) \\ s_1(t+1) \\ s_2(t+1) \\ \vdots \\ s_{k-2}(t+1) \\ s_{k-1}(t+1) \end{bmatrix} = \begin{bmatrix} h_{k-1} & h_{k-2} & h_{k-3} & \cdots & h_1 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \times \begin{bmatrix} s_0(t) \\ s_1(t) \\ s_2(t) \\ \vdots \\ s_{k-2}(t) \\ s_{k-1}(t) \end{bmatrix} = T_h \times S(t). \tag{9}$$

Similar to the Type I implementation for parallel CRC computation, if $r$ parity-check bits are to be computed at each time step, then the lookahead technique is applied to update the internal states of the flip-flops, and the lookahead equation can be expressed as

$$S(t+1) = T_h^{r-1}(T_h \times S(t)) = T_h^r \times S(t). \tag{10}$$

Fig. 6(a) shows the implementation of the Type II encoding circuit, which has the properties of modular design and easily applied for a variety of applications, e.g., the lookahead stages can be arranged in a similar manner as the Type-I encoding circuit shown in the next section. The T_MUX module depicted in Fig. 6(b) is designed to take into accounts both the access of input message and the autonomous behavior of the LFSR. Initially, the $C_M$ is set to select the input $I_i(t)$ such that the circuit will receive $r$ input message bits at the same time until the whole input message bits are stored in the register. After that, the value of $C_M$ is changed and $r$ bits of the parity-check bits are generated at each time step, thus completing the process of parallel CRC generation. It should be noted that if $k$ mod $r \neq 0$ or $(n-k)$ mod $r \neq 0$, a similar circuit could be derived in the same way at the expense of adding extra control signals for data multiplexing and selection.



(a)

(b)

Fig. 6. Parallel CRC computation for Type II encoding circuit; (a) modular design of the circuit, and (b) implementation of the T_MUX module.

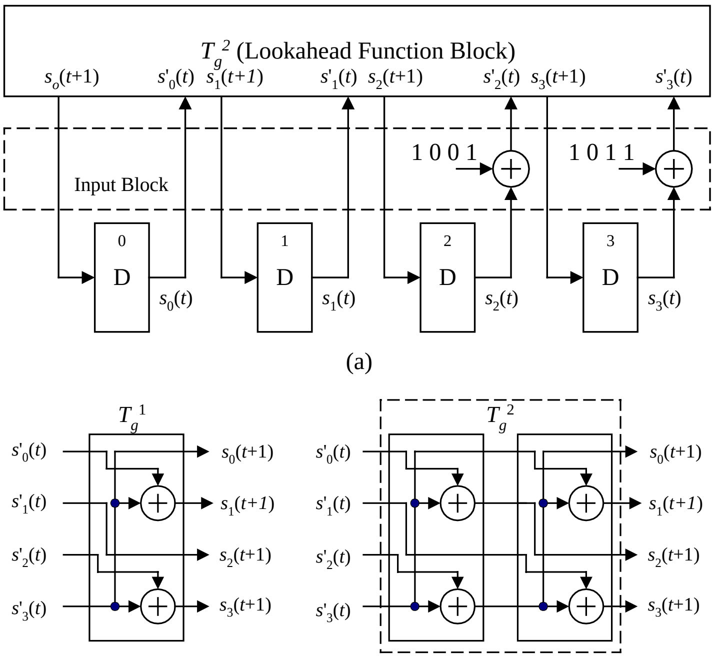
## 4. HARDWARE IMPLEMENTATION AND PERFORMANCE EVALUATION

Because the two types of parallel CRC computations have similar behavior and the Type-I encoding/decoding circuit is commonly used in practical applications, without loss of generality we focus on the Type-I implementation and its evaluation in this section.

### 4.1 Hardware Implementation Based on the Polynomial Generator *G*(*x*)

As an example, assume that the circuit is to be designed with the following characteristics: $k = 8$, $r = 2$, $G(x) = 1 + x + x^3 + x^4$ and $M(x) = 1 + x + x^5 + x^6 + x^7$. Fig. 7(a) depicts the resulting implementation of parallel CRC computation. The design of the lookahead function block and input block is described as follows. First, the one-order lookahead function $T_g$ is derived by assuming all zero inputs such that $s_i(t) = s_i'(t)$ and the corresponding next-state equations can be expressed as

$$\begin{bmatrix} s_0(t+1) \\ s_1(t+1) \\ s_2(t+1) \\ s_3(t+1) \end{bmatrix} = T_g \times s(t) = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} s_0(t) \\ s_1(t) \\ s_2(t) \\ s_3(t) \end{bmatrix} \tag{11}$$



(a)

(b)

Fig. 7. Example I (a) circuit implementation, and (b) implementations of the one-order and two-order lookahead functions.

For the circuit implementation based on $T_g$, the following equations can be derived: $s_0(t+1) = s_3'(t)$, $s_1(t+1) = s_0'(t) + s_3'(t)$, $s_2(t+1) = s_1'(t)$, and $s_3(t+1) = s_2'(t) + s_3'(t)$. In finite field, the addition operation is performed in modulo-2 addition, therefore it can be implemented in XOR gates. Since the target implementation is for $r = 2$, the next-state functions based on the two-order lookahead function $T_g^2$ are expressed as $s_0(t+1) = s_2'(t) + s_3'(t)$, $s_1(t+1) = s_2'(t)$, $s_2(t+1) = s_0'(t) + s_3'(t)$, and $s_3(t+1) = s_1'(t) + s_2'(t) + s_3'(t)$.

Those logic equations can be either designed independently or constructed by two cascaded one-order lookahead implementations as shown in Fig. 7(b). The input block is to consider the operation of $S(t) \oplus I(t)$. Based on the equation (8), we have $I(0) = [00m_6m_7]^T$, $I(1) = [00m_4m_5]^T$, $I(2) = [00m_2m_3]^T$ and , $I(3) = [00m_0m_1]^T$. Therefore, the two input subsequences $(m_0\, m_2\, m_4\, m_6) = (1001)$ and $(m_1\, m_3\, m_5\, m_7) = (1011)$ should be added to the internal states $s_2$ and $s_3$, respectively, as shown in Fig. 7(a). After four clock cycles, the final state of the storage elements is $(s_0\, s_1\, s_2\, s_3) = (0001)$ assuming the initial state is (0000). It implies that the output code word $C(x)$ is $x^3 + x^4M(x) = x^3 + x^4 + x^5 + x^9 + x^{10} + x^{11}$.

In general, the $r$-order lookahead function $T_g^r$ can be implemented as $r$ cascaded one-order lookahead function $T_g^1$. From implementation points of view, we can have two choices based on the dedicated applications. *Choice I*: The $r$-order lookahead function is optimized to minimize the timing requirement for achieving maximum operation rate. In this case, the $r$-order lookahead function can be flattened and optimized to reduce the critical path delay including the routing consideration. And, a careful design of XOR gate like the one in [12] can be applied to minimize the propagation delay. *Choice II*: We can first optimize the $r/p$-order lookahead function $T_g^{r/p}$ as long as $r$ is divisible by $p$, where $p$ is an integer. Then, the $r$-order lookahead function is implemented as $p$ cascaded $r/p$-order lookahead function blocks. In such a situation, it becomes the modular design of a lookahead function and can be flexible for designing a circuit that the value $r$ is changeable, i.e., for different values of $r$ as long as $r$ is a factor of the order of the generator polynomial. The flexibility can be easily achieved by multiplexing the intermediate values as shown in Fig. 8. In Fig. 8, the same circuit can be applied for four different numbers of parallel-input bits that are going to be processed at the same time by setting the values of controlling signals $cs_0$ and $cs_1$. Note that extra inputs should be set to zero in such an application and, for simplicity, the length of $M(x)$ is assumed to be divisible by the selected number of input bits.

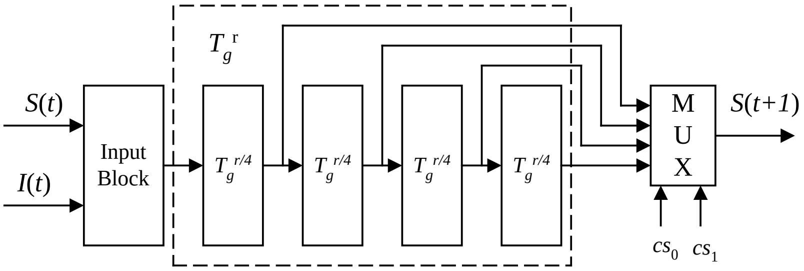


Fig. 8. A flexible design of the lookahead function block.

*Example II:* The CRC-32 [11] uses the polynomial generator $G(x) = 1 + x + x^2 + x^4 + x^5 + x^7 + x^8 + x^{10} + x^{11} + x^{12} + x^{16} + x^{22} + x^{23} + x^{26} + x^{32}$. For the application using the Media Independent Interface (MII) associated with the data link layer, a nibble-based input bits ($r = 4$) is used to generate the CRC code. Based on our development, the 4-order lookahead function is listed in Table 2. Fig. 9 shows the conceptual block diagram for the implementation of the CRC-32.

**Table 2. The 4-order lookahead function for CRC-32.**

| $s_i(t+1)$ | $\Sigma s_j(t)$ | $s_i(t+1)$ | $\Sigma s_j(t)$ | $s_i(t+1)$ | $\Sigma s_j(t)$ | $s_i(t+1)$ | $\Sigma s_j(t)$ |
|---|---|---|---|---|---|---|---|
| $i = 31$ | $j = 27$ | 23 | 19, 28, 29 | 15 | 11, 31 | 7 | 3, 28, 30, 31 |
| 30 | 26 | 22 | 18, 28 | 14 | 10, 30, 31 | 6 | 2, 29, 30 |
| 29 | 25, 31 | 21 | 17 | 13 | 9, 29, 30, 31 | 5 | 1, 28, 29, 31 |
| 28 | 24, 30* | 20 | 16 | 12 | 8, 28, 29, 30 | 4 | 0, 28, 30, 31 |
| 27 | 23, 29 | 19 | 15, 31 | 11 | 7, 28, 29, 31 | 3 | 29, 30, 31 |
| 26 | 22, 28, 31 | 18 | 14, 30 | 10 | 6, 28, 30, 31 | 2 | 28, 29, 30 |
| 25 | 21, 30, 31 | 17 | 13, 29 | 9 | 5, 29, 30 | 1 | 28, 29 |
| 24 | 20, 29, 30 | 16 | 12, 28 | 8 | 4, 28, 29, 31 | 0 | 28 |

* The *j* values (24, 30) represent the operation $s_{24}(t) + s_{30}(t)$.

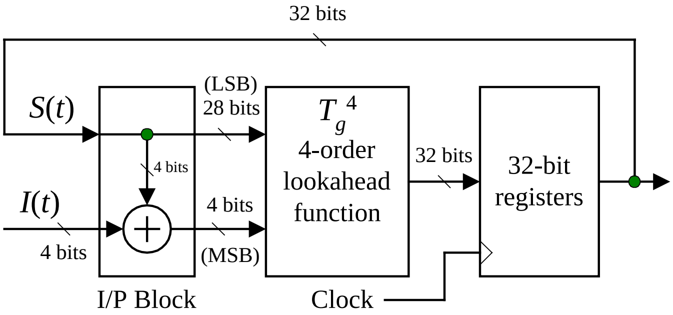


Fig. 9. Implementation of CRC-32 for $r = 4$.

The presented methodologies for parallel CRC computation have been successfully verified through both software simulation and hardware implementation. The software simulation is performed based on the automatic generation of synthesizable Verilog hardware description language [13] and the Cadence and Synopsys CAD tools. Because the presented methodologies have the characteristics of systematic derivation and modular design, it becomes very easy to develop the automatic synthesis environment, for which users can specify both the generator polynomial and the number of parallel-input bits. For the hardware implementation, a VLSI test chip shown in Fig. 10 is designed for the CRC-32 [11], which can be applied for $r = 1, 2, 4$ or 8, and is fabricated by the TSMC 0.6 *um* single-polysilicon-triple-metal process. The core size is 1072 × 1070$um^2$. When operating at 25 MHz with $r = 8$, the average power dissipation of the chip is 4.35 *m*W and the thoughput rate corresponds to 200Mbps. Note that because the test chip is targeted for 100 Mbps with $r = 4$, the goal can be easily achieved by using our approach; therefore we did not optimize the operating speed as well as the hardware requirement. Simulation result shows that the clock rate can operate at about 100 MHz with $r = 8$.

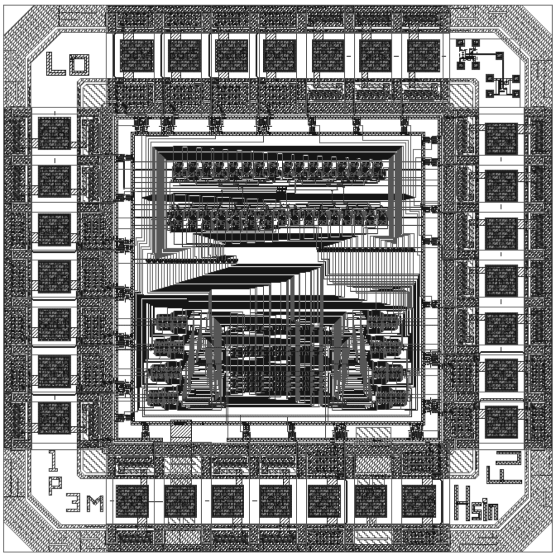


Fig. 10. The corresponding VLSI test chip for $r$ = 1, 2, 4 or 8.

### 4.2 Performance Evaluation

The evaluation of hardware requirements and speed comparisons for a variety of parallel-input combinations for Type-I implementation based the CRC-32 generator polynomial [11] is shown in Table 3, in which the area of controlling switches is ignored in the comparison. The total gate counts (GC) are calculated based on the COMPASS cell library [14] in terms of the 2-input NAND gates and the hardware overhead (HO) is normalized with respect to the serial-input implementation. The critical path delay (CPD) is estimated in terms of the number of 2-input XOR gates in the path. And, we use the Synopsys software to derive the corresponding values in the Table 3.

Two types of implementation are distinguished in our comparison: the cascaded lookahead function block and the flattened lookahead function block. In the cascaded implementation, it is assumes that the one-order lookahead function block $T_g^1$ is used as the basic building cell to implement $r$-order lookahead function block $T_g^r$. Specifically, the $T_g^r$ block is constructed from $r$ cascaded $T_g^1$ blocks only. In such a case, the number of required 2-input XOR gates is the summation of those in the lookahead function and the input block and can be simply expressed as (Nz – 2) × $r$ + $r$, where Nz represents the number of nonzero coefficients of the generator polynomial. For a flexible design, e.g, the circuit can be applied for both $r$ = 4 and 8, they are all constructed from the one-order lookahead function block; therefore the only difference between a fixed $r$-value and the flexible designs is the required multiplexiers. On the contrary, in the flattened implementation the $T_g^r$ block is flattened and then optimized by choosing both the area and timing constraints simultaneously in Synopsys options. In terms of a flexible design for $r$ = 4 and 8, the 8-order lookahead function block is derived by cascading two flattened 4-order blocks. It implies that the flattened block of the smallest $r$-value function is used to construct the other $r$-value functions in the flexible design.

**Table 3. Performance evaluation for a variety of parallel-input combinations.**

| Inputs | D-FF | MUX | Cascaded Lookahead Function Block | | | | Flattened Lookahead Function Block | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | XOR | GC* | HO | CPD | XOR | GC | HO | CPD |
| 1 | 32 | 0 | 14 | 298 | 1 | 1 | 14 | 298 | 1 | 1 |
| 2 | 32 | 0 | 28 | 340 | 1.14 | 2 | 25 | 331 | 1.11 | 2 |
| 4 | 32 | 0 | 56 | 424 | 1.42 | 3 | 48 | 400 | 1.34 | 2 |
| 8 | 32 | 0 | 112 | 592 | 1.99 | 6 | 89 | 523 | 1.76 | 3 |
| 16 | 32 | 0 | 224 | 928 | 3.11 | 8 | 177 | 787 | 2.64 | 4 |
| 32 | 32 | 0 | 448 | 1600 | 5.37 | 15 | 408 | 1480 | 4.97 | 7 |
| 1, 2 | 32 | 32 $_{(2\text{ to }1)}$ | 28 | 436 | 1.46 | 1, 2 | 28 | 436 | 1.46 | 1, 2 |
| 4, 8 ** | 32 | 32 $_{(2\text{ to }1)}$ | 112 | 688 | 2.31 | 3, 6*** | 96 | 640 | 2.15 | 2, 4 |
| 1, 2, 4, 8 | 32 | 32 $_{(4\text{ to }1)}$ | 112 | 816 | 2.74 | 1, 2, 3, 6 | 112 | 816 | 2.74 | 1, 2, 3, 6 |
| 16, 32 | 32 | 32 $_{(2\text{ to }1)}$ | 448 | 1696 | 5.69 | 8, 15 | 354 | 1414 | 4.74 | 4, 8 |
| 4, 8, 16, 32 | 32 | 32 $_{(4\text{ to }1)}$ | 448 | 1824 | 6.12 | 3, 6, 8, 15 | 384 | 1632 | 5.48 | 2, 4, 7, 13 |

* The relative gate count in terms of 2-input NAND gate is NAND : XOR : D-FF : MUX$_{2\text{-to-}1}$ : MUX$_{4\text{-to-}1}$ = 1 : 3 : 8 : 3 : 7.

** It means that the circuit can be applied for either r = 4 or r = 8.

*** The number of 2-input XOR gates in the critical path is dependent of the selected *r*-value.

It might be criticized that when $r = 32$ in the cascaded case, only a speedup ratio of 2 is achieved at the expense of 5.37 times the hardware requirement of the serial-input implementation. This conclusion is valid only when we assume that the applied clock cycle time is approaching the critical path delay of the implementation in two distinct cases, respectively. In practice, the clock rate might be dominated by another blocks in many applications such that the critical path delay of the lookahead function block might be neglected without sacrificing the overall operating speed. For example, for $r = 32$ in the cascaded case, the total delay time of the 15 cascaded 2-input XOR gates is 12.03 *ns*. Therefore, if the clock is operated less than 83 MHz, then a potential speedup ratio of 32 can be achieved at the expense of 5.37 times the hardware requirement of the serial-input implementation assuming that the same clock rate is applied for both cases. Note that the interconnection overhead and delay in the lookahead function should be taken into considerations in the final VLSI implementation. Nevertheless, the data listed in Table 3 provide an useful information for the designers to choose the best structures for a dedicated application. The best structure might be based on trade-offs among area, speed, regularity, and flexibility. It is evident that speed advantage can be achieved by using parallel CRC computation and a versatile design to deal with a different number of parallel-input bits can also be derived in the same circuit with small amounts of extra hardware requirement.

## 5. CONCLUSIONS

This paper shows a systematic method to calculate CRC in parallel. We take advantage of both Galois Field property and the lookahead technique to derive equations of two types of encoding/decoding schemes and their associated hardware implementations. Compared to [9], the derivation of state updating equations is easier based on our methodology. The method can be easily expanded to all feasible *r* parallel-input bits for fast CRC calculation without added complexity in the developing process or practical limitation like the size of look-up tables needed in other approaches. With little hardware overheads to control the number of lookahead stages, the same circuit can then be used for other applications using the same polynomial generator but processing a different number of input bits at a time. Therefore, from hardware implementation points of view, our design is more flexible than the result shown in [10]. Our development is valuable from both industrial and academic points of view and all the results have been verified through both software simulation and hardware implementation.

## REFERENCES

1. S. Lin and D. J. Costello, Jr, *Error Control Coding: Fundamentals and Applications,* Prentice-Hall, Inc., 1983.
2. S. B. Wicker, *Error Control Systems for Digital Communication and Storage,* Prentice-Hall, Inc., 1995.
3. T. V. Ramabadran and S. S. Gaitonde, "A tutorial on CRC computations," *IEEE Micro*, Vol. 8, No. 4, 1988, pp. 62-75.
4. R. Lee, "Cyclic code redundancy," *Digital Design*, Vol. 11, No. 7, 1977, pp. 77-85.
5. A. Perez, "Byte-wise CRC calculation," *IEEE Micro*, Vol. 3, No. 3, 1983, pp. 40-50.
6. A. K. Pandeya and T. J. Cassa, "Parallel CRC lets many lines use one circuit," *Computer Design*, Vol. 14, No. 9, 1975, pp. 87-91.
7. G. Albertengo and R. Sisto, "Parallel CRC generation," *IEEE Micro*, 1990, pp. 63-71.
8. R. J. Glaise and X. Jacquart, "Fast CRC calculation," in *Proceedings of IEEE International Conference on Computer Design*, 1993, pp. 602-605.
9. T. B. Pei and C. Zukowski, "High-speed parallel CRC circuits in VLSI," *IEEE Transactions on Communications*, Vol. 40, No. 4, 1992, pp. 653-657.
10. R. F. Hobson and K. L. Cheung, "A high-performance CMOS 32-bit parallel CRC engine," *IEEE Journal of Solid-State Circuits*, Vol. 34, No. 2, 1999, pp. 233-235.
11. ANSI/IEEE std 802.3, *Carrier Sense Multiple Access with Collision Detection* (*CSMA/CD*) *Access Method and Physical Layer Specifications*, fourth edition 1993.
12. J. M. Wang, S. C. Fang, and W. S. Feng, "New efficient designs for XOR and XNOR functions on the transistor level," *IEEE Journal of Solid-State Circuits*, Vol. 29, No. 7, 1994, pp. 780-786.
13. H. F. Lo, "Automatic generation of synthesizable Verliog codes for parallel CRC computations," Technical Report, NYUST-EE-88-1, Institute of Electronic and Information Engineering, National Yunlin University of Science & Technology, 1999.
14. COMPASS, *0.6 um, 5-V High-Performance Standard Cell Library,* June 1994.

**Ming-Der Shieh (謝明得)** received the B.S. degree in electrical engineering from National Cheng Kung University, Taiwan, in 1984, the M.S. degree in electronic engineering from National Chiao Tung University, Taiwan, in 1986, and the Ph.D. degree in electrical engineering from Michigan State University, East Lansing, in 1993. From 1988 to 1989, he was an engineer at United Microelectronic Corporation, Taiwan. He is currently an associate professor at National Yunlin University of Science & Technology, Taiwan. His research interests include computer-aided design, VLSI design and testing, VLSI for signal processing and digital communication.

**Ming-Hwa Sheu (許明華)** graduated from National Taiwan Institute of Technology, Taiwan, in 1986 and received the M.S. and Ph.D. degrees in electrical engineering from National Cheng Kung University, Tainan, Taiwan, in 1989 and 1993 respectively. He is presently an associate professor in Department of Electronic Engineering, National Yunlin University of Science & Technology, Taiwan. His research interests include CAD/VLSI, digital signal processing, algorithm analysis, and data communication.

**Chung-Ho Chen (陳中和)** received his MSEE degree from the University of Missouri-Rolla in 1989 and the Ph.D. degree in electrical engineering from the University of Washington, Seattle, in 1993. Since 1993, he was the faculty member of the Department of Electronic Engineering, National Yunlin University of Science & Technology. In 1999, he joined the Department of Electrical Engineering, National Cheng Kung University as an associate professor. His research interests include computer architecture, VLSI systems, and network processors. Dr. Chen is a member of the IEEE Computer Society.

**Hsin-Fu Lo (羅信富)** received the B.S. degree in electronic engineering from National Yunlin University of Science & Technology in 1998. He is currently a master student in Institute of Electronic and Information Engineering at National Yunlin University of Science & Technology, Taiwan. His research interests include VLSI design and VLSI architecture in digital signal processing.