논문 2006-43SC-6-6

파이프라인 구조를 적용한 병렬 CRC 회로 설계

(Design of Pipelined Parallel CRC Circuits)

이 현 빈*, 김 기 태*, 권 영 민**, 박 성 주***

(Hyunbean Yi, Kitae Kim, Youngmin Kwon, and Sungju Park)

요 약

본 논문은 CRC 회로의 성능을 향상시키기 위하여 파이프라인 구조를 적용한 병렬 CRC 회로 설계 방법을 제시한다. 입력 데이터의 폭이 CRC 다항식의 차수보다 큰 병렬 CRC 회로를 파이프라인 구조로 변형하기 위하여 로직을 분할하고, 파이프라인 단계의 길이를 결정하고, 각 파이프라인 단계에 레지스터를 삽입하는 알고리즘을 제시한다. 여러 가지 타입의 병렬 CRC 회로에 대해, 본 논문에서 제안한 방식이 현저하게 성능을 향상 시켰음을 알 수 있다.

Abstract

This paper introduces an efficient CRC logic partitioning algorithm to design pipelined parallel CRC circuits aimed at improving speed performance. Focusing on the cases that the input data width is greater than the polynomial degree, equations are derived to divide the parallel CRC logic and decide the length of the pipeline stage. Through design experiments on different types of parallel CRC circuits, we have found a significant reduction in delay by adopting our approach.

Keywords: Parallel CRC, pipeline, logic partitioning, logic-level

I. Introduction

Cyclic Redundancy Check (CRC) is an error-detection scheme used in communication systems. For data transmission on high speed medium and long distance links, most errors tend to be multiple-bit bursts. The CRC scheme that can detect the same number of error bits as the degree of the polynomial, no matter how big data sizes are. Accordingly, the scheme is most suitable for high speed communication systems. Most recent high speed serial interconnect technologies such as Gigabit Ethernet, PCI-Express, FiberChannel, and InfiniBand, etc. go for over 100 Gbps rather than 10 Gbps; therefore, it becomes crucial to design CRC circuits that run at that higher rate.

Traditionally, CRC circuits have been implemented in serial domain by using Linear Feedback Shift Registers (LFSRs). However, it becomes difficult to implement LFSRs operating at high frequency bit-clocks, so a number of studies have presented parallel CRC architectures and tried to speed them up by methods such as table look-up[1],[2], accumulating 8 bits in each clock cycle[3], pre-decoding and binary tree optimization[4], and polynomial division[5]. In

* 학생회원, 한양대학교 컴퓨터공학과
(Dept. of Computer Science & Engineering, Hanyang University)
** 정회원, 한국전자부품연구원 지능형 정보시스템 연구 센터
(Dept. of Intelligent IT System Research Center, Korea Electronics Technology Institute.)
*** 정회원, 한양대학교 전자컴퓨터공학과
(Dept. of Electronical Engineering Computer Science, Hanyang Univ.)
※ 본 논문은 유망전자 부품기술 개발 사업, 채널 기반형 입출력 부품 개발 과제로부터 지원을 받아 진행하였습니다.
접수일자: 2006년5월30일, 수정완료일: 2006년11월4일

addition, in [6] a design method based on the Galois field theory has been proposed. A generic VHDL code of parallel CRC circuits for any generator polynomial and data width has been presented, but there still remains the problem that delay can be increased pertaining to the number of parallel input bits[7],[8] claimed a better speed by ameliorating the approaches presented in [6] and [7], however, their optimized equations are restricted to cases where the input data-width is less than or equal to the polynomial degree.

This paper presents an effective method to improve the speed by partitioning CRC logic and using pipelined architecture for the case where the input data width is greater than or equal to the polynomial degree. A basic method to implement parallel CRC is introduced in section Ⅱ. Efficient partitioning techniques of the CRC logic and decision algorithm for the size of pipeline registers are described in section Ⅲ. Parallel CRC circuits are implemented and their delays are evaluated in section Ⅳ followed by conclusions in section Ⅴ.

Ⅱ. Parallel CRC Algorithm

In order to generate CRC codes, network and communication systems usually make use of several standardized polynomials and perform modulo2 binary divisions. Modulo2 binary divisions can be calculated by XOR (exclusive-or) operations and implemented by using an LFSR and XOR gates in serial domain. Fig. 1 shows a primitive polynomial and its hardware implementation, and it may be noted that the CRC circuits must be designed to afford high input data rate.

The basic ideas for designing parallel CRC circuits are explained in detail by using determinants based on the Galois Field theory in [8] and by precalculating XOR combinations to be loaded in LFSR after a number of shifts in [9]. As an LFSR is a discrete-time linear time-invariant (LTI) system, the current value in the register and the next value to be newly updated in each of w clock cycles can be defined as C = [cn-1 ⋯ c1 c0]T and C′ = [c′n-1 ⋯ c′1 c′0]T, respectively, for a data width w and a generating polynomial degree n. With these definitions, a recursive formula was identified in [8] as follows:

$$C' = F^{w} \otimes (C \oplus D) \quad (\otimes : multiplication, \oplus : XOR) \tag{1}$$

where parallel input data $D = [d_{w-1} \cdots d_1\ d_0 \mid 0 \cdots 0]^T$ for $w < n$ and $[d_{w-1} \cdots d_1\ d_0]^T$ for $w = n$ and $n \times n$ parallel CRC transformation matrix F:

$$F = \begin{bmatrix} p_{n-1} & 1 & 0 & \cdots & 0 \\ p_{n-2} & 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ p_1 & 0 & 0 & \cdots & 1 \\ p_0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

where, for $i = 0, 1, 2, \ldots, n-1$, p_i are the coefficients of generating polynomial.

Let R_i and x_i be the i^{th} register of LFSR and $c_i \oplus d_i$, respectively. We can obtain Table 1 from the equation (1) and Fig. 1. Finally, as shown in Fig. 2, a basic parallel CRC circuit can be easily implemented with a w-bit input data register (Din Reg), an n-bit CRC code register (CRCReg) and XOR combinational logic (XOR Logic) based on the Table 1.

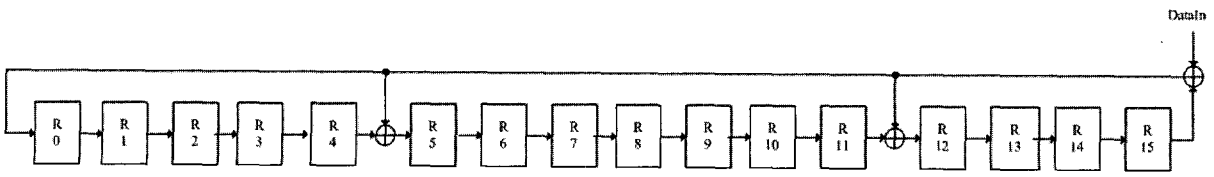


그림 1. P(y) = 1 + y^5 + y^{12} + y^{16} (ITU_TSS)에 의한 LFSR
Fig. 1. LFSR for P(y) = 1 + y^5 + y^{12} + y^{16} (ITU_TSS).

표 1. $(P(y) = 1 + y5 + y12 + y16,\ w = 16)$의 병렬 CRC XOR 조합 표

Table 1. Example XOR Combinations of $(P(y) = 1 + y5 + y12 + y16,\ w = 16)$ Parallel CRC.

R_{15}	$x_{15} \oplus x_4 \oplus x_{11} \oplus x_7 \oplus x_3$
R_{14}	$x_{14} \oplus x_3 \oplus x_{10} \oplus x_6 \oplus x_2$
R_{13}	$x_{13} \oplus x_2 \oplus x_9 \oplus x_5 \oplus x_1$
R_{12}	$x_{12} \oplus x_1 \oplus x_8 \oplus x_4 \oplus x_0$
R_{11}	$x_{11} \oplus x_0 \oplus x_7 \oplus x_3$
R_{10}	$x_{10} \oplus x_6 \oplus x_2 \oplus x_{15} \oplus x_4 \oplus x_{11} \oplus x_7 \oplus x_3$
R_9	$x_9 \oplus x_5 \oplus x_1 \oplus x_{14} \oplus x_3 \oplus x_{10} \oplus x_6 \oplus x_2$
R_8	$x_8 \oplus x_4 \oplus x_0 \oplus x_{13} \oplus x_2 \oplus x_9 \oplus x_5 \oplus x_1$
R_7	$x_7 \oplus x_3 \oplus x_{12} \oplus x_1 \oplus x_8 \oplus x_4 \oplus x_0$
R_6	$x_6 \oplus x_2 \oplus x_{11} \oplus x_0 \oplus x_7 \oplus x_3$
R_5	$x_5 \oplus x_1 \oplus x_{10} \oplus x_6 \oplus x_2$
R_4	$x_4 \oplus x_0 \oplus x_9 \oplus x_5 \oplus x_1$
R_3	$x_8 \oplus x_0 \oplus x_{15} \oplus x_{11} \oplus x_7$
R_2	$x_7 \oplus x_{14} \oplus x_{10} \oplus x_6$
R_1	$x_6 \oplus x_{13} \oplus x_9 \oplus x_5$
R_0	$x_5 \oplus x_{12} \oplus x_8 \oplus x_4$

III. Parallel CRC Logic Pipelining

If $w < n$, the logic-level of the critical path of the XOR Logic depends on the feedback logic part of the n-bit *CRC_Code*. However, if $w > n$, the logic-level will depend on the w-bit input data logic part. *The clock frequency of the CRC circuit is determined by the critical path of the feedback logic of the CRC_Code no matter how large w is. Thus, for w > n, we divide the XOR Logic into a feedback logic of the CRC_Code and an input data logic, and split the input data logic in several stages to make the logic-level of the critical path of each stage sublogic be smaller than that of the feedback logic of the CRC_Code.*

1. XOR Logic Partitioning

The XOR Logic in Fig. 2 can be divided into CRC Code XOR Logic (CX), Data XOR Logic (DX), and XORArray as shown in Fig. 3.

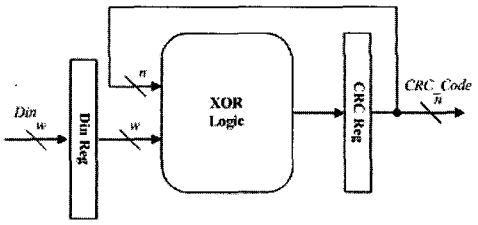


그림 2. 기본적인 병렬 CRC 회로

Fig. 2. Basic parallel CRC architecture.

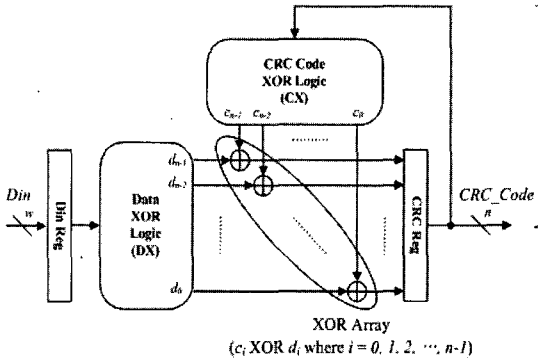


그림 3. 분할된 XOR 회로

Fig. 3. Partitioned XOR Logic.

Focusing on the cases where $w > n$, let us formulate the solutions for implementing CX and DX.

For $i = 0, 1, ..., n-1$, and time t, let parallel input data *Din*, output of CX *Cr*, output of DX *Dr* and *CRC_Code* be

$$Din = [din_{w-1}\ din_{w-2}\ ...\ din_0]^T,$$
$$Cr = [c_{n-1}\ c_{n-2}\ ...\ c_0]^T,$$
$$Dr = [d_{n-1}\ d_{n-2}\ ...\ d_0]^T$$

and

$$CRC\_Code = [crc_{n-1}\, crc_{n-2} \ldots crc_0]^T,$$

then we have

$$CRC\_Code(t+1) = Cr(t) \oplus Dr(t), (t > 0) \quad (2)$$

where *CRC_Code(1)* for $t = 0$ is the initial value.

If we assume that $Din = [0\ 0\ ...0]^T$, then $Dr = [0\ 0\ ...\ 0]^T$. As long as $Din = [0\ 0\ ...0]^T$, the data width constraints are meaningless. Therefore, from equations (1) and (2), we can derive an equation only for CX. D in equation (1) and $Dr(t)$ in the equation (2) can be removed, and C and C' in equation (1) can be replaced by $CRC\_Code(t)$ and $CRC\_Code(t+1)$, respectively. Thus, we can obtain an implementing

equation for CX:

$$Cr = F^w \otimes CRC\_Code \quad (3)$$

As noted above, the size of *Din* w is greater than n. The *Din* needs to be divided into groups of n-bit to make use of equation (1). If w is divisible by n, in other words, $w = n * k$ for a positive integer k, we can make k groups each of which includes n-bit as follows:

$$Din^1 = [din_{w-1}\ din_{w-2} \ldots din_{w-n}]^T$$
$$Din^2 = [din_{w-n-1}\ din_{w-n-2} \ldots din_{w-2n}]^T$$
$$\vdots \qquad\qquad \vdots$$
$$Din^k = [din_{w-(k-1)n-1}\ din_{w-(k-1)n-2} \ldots din_0]^T.$$

If w is not divisible by n, in other words, $w = n * (k-1) + j$ for a positive integer j, the number of elements in the last group Din^k becomes less than n, and we have to adjust it to be n by including j zeros as follows:

$$Din^k = [din_{w-(k-1)n-1}\ din_{w-(k-1)n-2} \ldots din_0 \mid 0 \ldots 0]^T.$$

To derive a formula for DX using Din^i for $i = 1, 2, \ldots, k$, we consider that a value loaded in an LFSR after n clock cycles becomes the initial value of the LFSR for Din^{i+1}. If we let $V = [v_{n-1}\ v_{n-2} \ldots v_0]^T$ be the n-bit value to be loaded in the LFSR, we can derive an equation $D'in^i = F^n \otimes (Din^i \oplus V)$ from equation (1), where the initial value of V is $[0 \ldots 0]^T$. Thus we have:

$$D'in^1 = F^n \otimes (Din^1 \oplus [0 \ldots 0]^T) = F^n \otimes Din^1,$$
$$D'in^2 = F^n \otimes (Din^2 \oplus D'in^1),$$
$$\vdots \qquad\qquad \vdots$$
$$D'in^k = F^n \otimes (Din^k \oplus D'in^{k-1}).$$

Now that $D'in^k$ is Dr, we can finally obtain an equation to implement the DX as follows:

$$Dr = F^n \otimes (Din^k \oplus (F^n \otimes (Din^{k-1} \ldots \ldots (F^n \otimes (Din^2 \oplus (F^n \otimes Din^1))) \ldots \ldots))). \quad (4)$$

2. Data XOR Logic Partitioning

If DX is partitioned, it will consist of Sub-Logics and registers as shown in Fig. 4. How many stages DX is divided into and how many logic-levels each DX Sub-Logic includes are the key challenges to speed up the circuits by pipelining the parallel CRC. We propose an efficient algorithm to determine the number of DX Sub-Logics, the logic-level, and the size of stage registers.

(1) Logic-level Partitioning

If the logic-level of the critical path of DX in Fig. 3 is less than that of CX, DX does not need to be partitioned. Therefore, a comparative analysis of the logic-levels of CX and DX must first be made.

If we suppose that logic is synthesized as a completed binary tree and let L be the logic-level of the critical path, L can be found by the equation (5):

$$L = round\ up\ (log_2 N) \quad (5)$$

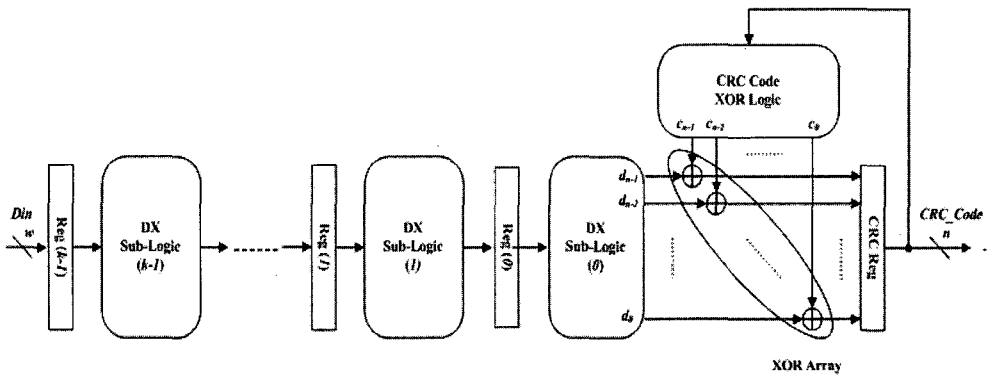


그림 4. 분할된 DXs
Fig. 4. Partitioned DXs.

표 2. w에 따른 CX와 DX의 로직 레벨
Table 2. Logic-Levels of CX and DX according to *w*.

Type of CRC		CRC16-A (ITU-TSS)[a]				CRC16-B (HDLC)[b]				CRC16-C (PCI-Express, InfiniBand, etc.)				CRC32 (Ethernet, ATM, PCI-Express, InfiniBand, etc.)			
Generating Polynomial		$1+x^5+x^{12}+x^{16}$				$1+x^2+x^{15}+x^{16}$				$1+x+x^3+x^{12}+x^{16}$				$1+x+x^2+x^4+x^5+x^7+x^8+x^{10}+x^{11}+x^{12}+x^{16}+x^{22}+x^{23}+x^{26}+x^{32}$			
XOR Logic		CX		DX		CX		DX		CX		DX		CX		DX	
Measure		N		L		N		L		N		L		N		L	
w (bit)	32	*11*	*4*	*18*	*5*	*14*	*4*	*29*	*5*	*15*	*4*	*24*	*5*	*17*	*5*	*17*	*5*
	64	*12*	*4*	*25*	*5*	*12*	*4*	*56*	*6*	*12*	*4*	*47*	*6*	*19*	*5*	*34*	*6*
	128	*9*	*4*	*70*	*7*	*8*	*3*	*100*	*7*	*10*	*4*	*77*	*7*	*20*	*5*	*69*	*7*
	256	*10*	*4*	*132*	*8*	*13*	*4*	*175*	*8*	*10*	*4*	*146*	*8*	*20*	*5*	*138*	*8*

표 3. 로직 레벨 조합 예
Table 3. Example combinations of Logic-Level.

CRC16-A & *w = 32*	Logic-level of DX Sub-Logic(0)	Logic-level of DX Sub-Logic(1)
Case 1	4	1
Case 2	3	2
Case 3	2	3
Case 4	1	4
Case 5	0	5

where N is the number of inputs which affects the output of the critical path [10]. We have obtained Ns from the equation (4) and analyzed Ls for 3 types of 16-bit CRCs and a 32-bit CRC as shown in Table 2.

As DX Sub-Logic(*0*), shown in Fig. 4, shares the XOR Array part with CX, the logic-level of DX Sub-Logic(*0*) must be less than the CX's logic-level and the logic-levels of other DX Sub-Logics must be less than the CX's logic-level plus '1'to allow the clock frequency to be determined by the CX.

Let us take an example. For CRC16-A with $w = 32$, we had CX's L = 4 and DX's L = 5 from Table 2. It is seen that DX must be split into two sub-logics because DX's L is larger than CX's L and is not over twice of CX's L. There are 5 possible cases as shown in Table 3. Although the logic-level of DX Sub-Logic(*0*) is the same as the logic-level of CX if *Case 1* is chosen, the critical path's delay of DX Sub-Logic(*0*) might be bigger than that of CX because of the differences in routing complexity and fan-outs by synthesis and placement & routing. Similarly, as DX Sub-Logic(*1*) and CX + XORArray are the same logic-level if *Case 5* is chosen, such delay problems can occur. Thus, we can obviate the delay problems by choosing one of *Case 2, Case 3* or *Case 4* .

(2) Register Size Decision

For the net-list with large number of gates, a systematic approach must be provided in inserting the registers to the proper places. In this subsection, considering that the number of nets in a logic block decreases monotonically from the inputs to an output, we introduce a simple technique to estimate the size of each stage register.

Let us define some parameters based on Fig. 4 as follows:

d_j : output of DX Sub-Logic(*0*),
DL(i) : logic-level of DX Sub-Logic(i),
$SBIn_{(j,i)}$: inputs affecting d_j of inputs of DX Sub-Logic(i),
$S_{(j,i)}$: size of $SBIn_{(j,i)}$,
$RS_{(i)}$: size of $Reg_{(i)}$,

where $i = 0, 1, \ldots, k-1$ and $j = 0, 1, \ldots, n-1$.

Then, from equation (5), we obtain

$$S_{(j,k-2)} = round\ up\ (S_{(j,k-1)} / 2^{DL(k-1)}),$$
$$S_{(j,k-3)} = round\ up\ (S_{(j,k-2)} / 2^{DL(k-2)}),$$
$$\vdots \qquad\qquad \vdots$$
$$S_{(j,0)} = round\ up\ (S_{(j,1)} / 2^{DL(1)}),$$

where DL(i) is given by the logic-level partitioning as mentioned in the subsection (1). and $S_{(j,k-1)}$ is obtained from equation (4). Thus, $S_{(j,i)}$ can be computed as:

$$S_{(j,i)} = round\ up\ (S_{(j,i+1)} / 2^{DL(i+1)}) \quad for\ i < k-1 \qquad (6)$$

For example, if DX is split into two DX Sub-Logics with DL(*1*) = 2 and DL(*0*) = 2 and $S_{(ex,1)}$ is 10 for an output d_{ex}, $SBIn_{(ex,1)}$ - to - d_{ex} the logic network in the DX is implemented as shown in Fig. 5 because $S_{(ex,0)}$ = *round up* ($S_{(ex,1)} / 2^{DL(1)}$) = 3. Applying equation (6) to all outputs, we can finally obtain RS(i) as follows:

$$RS(i) = S_{(0,i)} + S_{(1,i)} + \cdots + S_{(n-1,i)} \qquad (7)$$

As logic networks and stage registers are constructed for each of outputs, the area overhead is relatively high. Therefore, appropriate area optimization algorithm by finding registers shared is needed. Heuristics are required because the optimization is typical NP-hard problem, and in this paper we rely on the CAD tool by just plugging in all the output expressions.

IV. Performance Evaluation

We have implemented Basic and Pipelined parallel CRC circuits for w = 32, 64, 128 and 256, respectively. The circuits were mapped by Synopsys Design Analyzer with TSMC 0.25 ㎛ process logic cell library and performed timing analysis by Prime Time. Table 4 shows performance comparisons by Delay and Delay Reduction (Red.).

CRC16-A with w = 256 and CRC16-C with w = 256 have been structured with 3 pipeline stages, and all of the rest of the circuits have been structured with 2 pipeline stages. As shown in Fig. 6, the delays of the Basic parallel CRC circuits are increasing, since the critical path of DX becomes longer as w increases. The delays of the Pipelined parallel CRC circuits, however, vary only a little, because DX is partitioned and the critical path of the circuit is determined by CX affected slightly by w.

V. Conclusion

In this paper, an efficient method to design high performance pipelined parallel CRC circuits by partitioning logic circuits and deciding the number of the pipeline stages is proposed. After dividing CRC logic into the CRC code feedback logic and the input

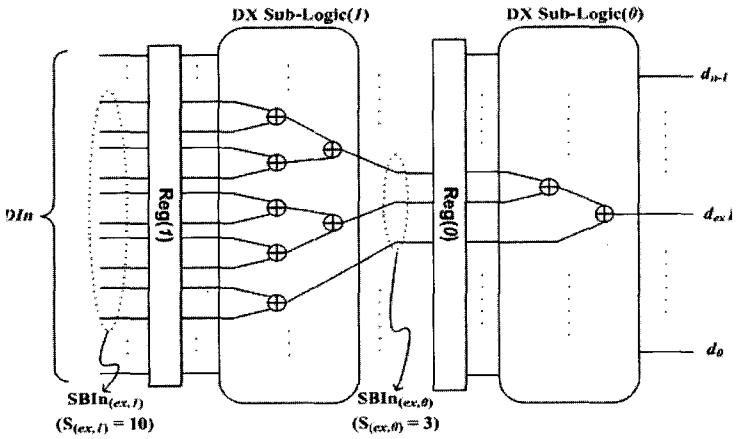


그림 5. 구현 예
Fig. 5. Implementation Example.

표 4. 결과 성능 비교
Table 4. Performance Comparisons.

Type of CRC		CRC16-A			CRC16-B			CRC16-C			CRC32		
Architecture		Basic[a]	Pipelined[b]		Basic	Pipelined		Basic	Pipelined		Basic	Pipelined	
Measure		Delay[c] (ns)	Delay (ns)	Red.[d] (%)	Delay (ns)	Delay (ns)	Red. (%)	Delay (ns)	Delay (ns)	Red. (%)	Delay (ns)	Delay (ns)	Red. (%)
w (bit)	32	*2.70*	*2.28*	*15.56*	*3.22*	*2.28*	*29.19*	*3.23*	*3.03*	*6.19*	*2.90*	*2.88*	*0.69*
	64	*3.06*	*2.34*	*23.53*	*3.32*	*2.43*	*26.81*	*3.05*	*2.25*	*26.23*	*3.24*	*2.55*	*21.30*
	128	*3.39*	*2.31*	*31.86*	*3.47*	*2.38*	*31.41*	*3.50*	*2.45*	*30.00*	*3.68*	*2.91*	*20.92*
	256	*3.84*	*2.18*	*43.23*	*3.99*	*2.63*	*34.09*	*3.92*	*2.18*	*44.39*	*4.28*	*2.94*	*31.31*

[a] *Basic: Basic Parallel CRC circuit (Ref. to Fig. 2)*
[b] *Pipelined: Pipelined Parallel CRC circuit (Ref. to Fig. 4)*
[c] *Delay: Data arrival time on critical path*
[d] *Red.: Delay Reduction*

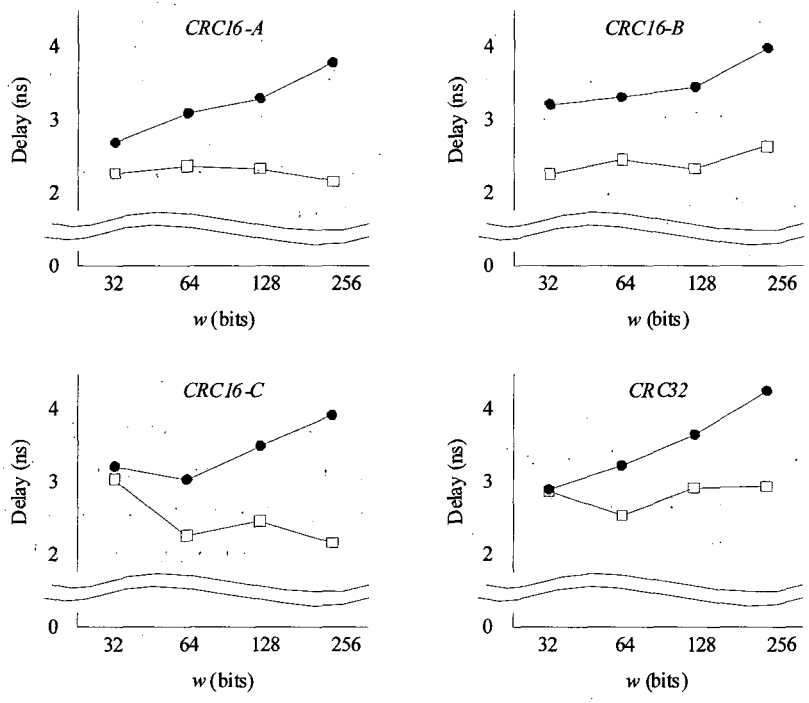


그림 6. 기본 회로 vs 파이프 라인 회로
Fig. 6. Basic vs Pipelined.

data logic, we split the input data logic in several pipeline stages, each with the shorter critical path, and then insert stage registers. Compared with basic parallel CRC circuits, the delay is reduced by up to 44%.

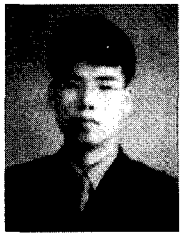
Reference

[1] D. V. Sarwate, "Computation of Cyclic Redundancy Checks via Table Look-Up," Comm. ACM, Aug. 1988.
[2] S.M. Joshi, P.K. Dubey and M.A. Kaplan, "A New Parallel Algorithm for CRC Generation," IEEE International Conference on Communications,

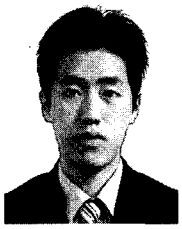
Vol. 3, pp. 18-22, Jun. 2000.
[3] T.B. Pei and C. Zukowski, "High-Speed Parallel CRC Circuits in VLSI," IEEE Transaction on Communications, Vol. 40, no. 4, pp. 653-657, 1992.
[4] R.F. Hobson and K.L. Cheng, "A High-Performance CMOS 32-Bit Parallel CRC Engine," IEEE Journal of Solid-State Circuits, Vol. 34, No. 2, pp. 233-235, Feb. 1999.
[5] F. Monteiro, A. Dandache, A. M'Sir and B. Lepley, "A Fast CRC Implementation on FPGA Using a Pipelined Architecture for the Polynomial Division," IEEE International Conference on Electronics, Circuits and Systems, Vol. 3, pp. 1231-1234, Sept. 2001.
[6] M.D. Shieh et al., "A Systematic Approach for Parallel CRC Computations," J. Information Science and Engineering, May 2001.
[7] M. Spachmann, "Automatic Generation of Parallel CRC Circuits," IEEE Design and Test of Computers, Vol. 18, pp. 108-114, May 2001.
[8] G. Campobello, G. Patane and M. Russo, "Parallel CRC Realization," IEEE Transactions on Computers, Vol. 52, pp. 63-71, Oct. 2003.
[9] Cypress Semiconductor Corporation, "Parallel Cyclic Redundancy Check (CRC) for HOTLINK™," Application note, Mar. 1999.
[10] G.D. Micheli, Synthesis and Optimization of Digital Circuits. McGRAW-HILL INTERNATIONAL EDITIONS, 1994.

저 자 소 개

이 현 빈(학생회원)
2001년 한양대학교 전자컴퓨터 공학과 학사 졸업.
2003년 한양대학교 컴퓨터공학과 석사 졸업.
2003년~현재 한양대학교 컴퓨터 공학과 박사과정.
<주관심분야 : SoC 테스팅, ASIC 설계, 네트워크 시스템 설계.>



권 영 민(정회원)
2002년 영남대학교 전자공학과 학사 졸업.
2004년 영남대학교 전자공학과 석사 졸업.
2004년~현재 한국 전자부품 연구원 전임 연구원.
<주관심분야: VLSI, SoC, high speed I/O interface, embedded system>



김 기 태(학생회원)
2005년 한양대학교 전자컴퓨터 공학과 학사 졸업.
2006년 한양대학교 컴퓨터공학과 석사 과정 중.
<주관심분야: 반도체, 테스팅, CAD/VLSI>

박 성 주(정회원)
1983년 한양대학교 전자공학과 학사 졸업.
1983년~1986년 금성사 소프트웨어 개발 연구원.
1992년 Univ. of Massachusetts 전기 및 컴퓨터공학과 박사졸업.
1992년~1994년 IBM Microelectronics 연구스텝.
1994년~현재 한양대학교 전자컴퓨터공학부 정교수.
<주관심분야 : 테스트 합성, Built-In Self Test, Scan Design, ATPG, ASIC설계, 고속 신호처리 시스템 설계, 그래프이론>