파이프라인 구조를 적용한 병렬 CRC 회로 설계

김기태*, 이현빈*, 박성주*, 박창원**
*한양대학교 컴퓨터공학과
**전자부품연구원 지능형정보시스템연구센터

Pipelined Parallel CRC

Kitae Kim*, Hyunbean Yi*, Sungju Park* and Changwon Park**
E-mail : *{kkt79,bean,parksj}@mslab.hanyang.ac.kr
**parkcw@keti.re.kr

Abstract

In this paper, we propose a method that applies pipeline architecture to parallel CRC circuits. We developed a logic partitioning algorithm for applying pipeline architecture. Our algorithm can be used for the polynomial and the input data width, both of arbitrary length and minimize the logic level. Design experiments show the superiority of our approach in reducing the delay in comparison with previous works.

1. Introduction

Cyclic Redundancy Check (CRC)는 통신 시스템에서 가장 널리 사용되는 데이터 오류 검출 방법이다. 고속의 중장거리 데이터 송수신시에 발생하는 대부분의 에러는 연속적인 여러 비트에 걸쳐 발생하는 특성이 있으며, 데이터의 크기에 상관 없이 코드 생성 다항식 (폴리노미얼: Polynomial)의 차수만큼의 비트 열로 여러 비트의 오류를 검출 할 수 있는 CRC 는 고속 통신 시스템에 매우 적합하다. 통신 시스템 기술의 발달로 네트워크 및 데이터 I/O 송수신 표준들은 1 Gb/s 를 넘어 10 Gb/s 이상의 속도를 목표로 하고 있으며, 그에 따라 고속으로 동작 할 수 있는 CRC 회로 설계 기술이 필요하다. CRC 는 기본적으로, Linear Feedback Shift Register (LFSR)를 이용한 직렬 회로로 구성되어, 높은 주파수의 비트 클럭을 사용해야 하므로, 병렬 CRC 구현에 관한 연구가 이루어져왔다[1-8].

고속으로 갈수로 클럭의 주파수를 계속 증가시키는 것은 기술 및 비용 면에 있어서 한계가 있으므로, 저비용으로 시스템 성능을 향상시키기 위해서 데이터 폭의 확장을 고려할 수 있다. 그러나, 데이터 폭이 커지면 커질수록 로직의 크기가 커져 클럭 주기 또한 늘어나게 된다. 게이트 수 및 로직 레벨 최적화를 통하여 성능을 향상시킬 수 있지만, 이 또한 NP-Hard 문제이다.

본 논문은 전자부품연구원 유망전자부품기술개발사업 (Electro-0580)의 지원하에 이루어졌음.

본 논문에서는, 데이터 폭이 생성 다항식의 차수보다 큰 경우, CRC 로직을 분리 및 파이프 라인구조로 구현하여 성능을 향상시키기 위한 방법을 제시한다. 2 장에서 기본적인 병렬 CRC 회로 구현 방법을 간단하게 설명하고, 3 장에서 파이프 라인 구조로 구현하기 위한 로직 분할 알고리즘을 설명한다. 4 장에서 데이터 폭과 생성 다항식이 서로 다른 여러 병렬 CRC 회로를 구현하여 성능을 평가하고, 5 장에서 결론을 맺는다.

2. Parallel CRC Algorithm

여러 통신 시스템에서는, 코드 생성 다항식을 이용하여, 이진 모듈로 2 연산을 수행함으로써 나온 결과를 CRC 코드로 사용한다. 이와 같은 연산을, 직렬 데이터 송수신 시스템에서는 LFSR 과 XOR 게이트를 이용하여 하드웨어로 구현 할 수 있다. 그림 1 은 ITU-TSS 표준 16 비트 CRC 생성을 위한 LFSR 이다.

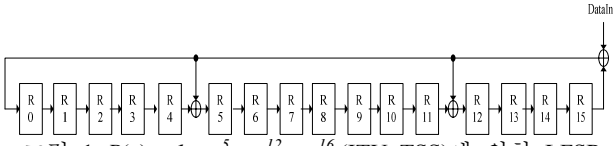

그림 1. $P(y) = 1 + y^5 + y^{12} + y^{16}$ (ITU_TSS)에 의한 LFSR

병렬 CRC 코드 생성의 기본적인 아이디어는, 이와 같은 LFSR 에서 여러 번의 쉬프트와 XOR 연산 후에 각 플립플롭에 저장되는 결과를 병렬 데이터 입력의 XOR 조합회로로 한 클럭 사이클에 생성될 수 있도록 하는 것이다. 논문 [8]등에서는 Galois Field 를 사용한 수식으로써 병렬 CRC 구현 방법을 자세히 설명하고 있다.

그림 1 과 같은 LFSR 은, 이산 시간 (discrete-time), 시불변 선형 시스템 (time-invariant linear system)이므로, 생성 다항식의 차수를 'n', 병렬 입력 데이터의 폭을 'w', 생성 다항식 $P(y) = p_0 y^0 + p_1 y^1 + \ldots + p_{n-1} y^{n-1}$ 라고 하고, 레지스터의 현재 상태 값 $C = [c_{n-1} \ldots c_1 \; c_0]^T$, 레지스터의 다음 상태 값 $C' = [c'_{n-1} \ldots c'_1 \; c'_0]^T$ 라고

정의 할 수 있다. 이를 바탕으로, 논문 [8]에서, n x n CRC 변환 행렬 F

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

를 이용하여, $w < n$ 일 때, 병렬 입력 데이터 $D = [d_{w-1} \ldots d_1\ d_0 \mid 0 \ldots 0]^T$ 이고, $w = n$ 일 때, $D = [d_{w-1} \ldots d_1\ d_0]^T$ 라 놓으면,

$$C' = F^w \otimes (C \oplus D) \quad (2)$$
($\otimes$: multiplication, $\oplus$: XOR)

임을 증명하였다.

i 를 0 이상 $n-1$ 이하인 양의 정수로 놓고, R_i 는 그림 1 과 같이 LFSR 의 i 번째 레지스터라고 하고, $x_i = c_i \oplus d_i$ 로 놓으면, 병렬 CRC 회로의 XOR 로직의 입출력 함수를 구할 수 있다.

최종적으로, 그림 2 와 같이, w 비트 레지스터와, n 비트 레지스터, 그리고 XOR 로직 입출력 표를 바탕으로 **XOR Logic** 을 구성함으로써 쉽게 병렬 CRC 회로를 구현 할 수 있다.

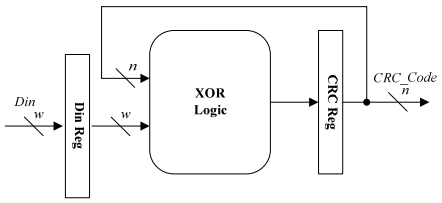

그림 2. 기본적인 병렬 CRC 회로

3. Pipelining of Parallel CRC Logic

$w <= n$ 인 경우에는, 식 (2)에 의해 XOR Logic 의 임계경로 (critical path)의 로직 레벨이 C 에 의해 결정된다. 그러나, $w > n$ 인 경우는, D 의 크기가 C 보다 크므로 XOR Logic 의 임계경로의 로직 레벨이 D 에 의해 좌우된다. 따라서, 본 논문에서는, n 비트 CRC 회로에 대해서, w 가 n 보다 클 경우, 그림 2 의 XOR Logic 을 피드백 되는 CRC_Code 를 입력으로 하는 로직과 병렬 데이터 Din 을 입력으로 하는 로직으로 분리하고, Din 을 입력으로 하는 로직을, 한 단계의 로직 레벨이 CRC_Code 를 입력으로 하는 로직의 최대 로직 레벨의 이하가 되도록 파이프 라인으로 구성하여 성능을 향상시키는 방법을 제시한다.

3.1 XOR Logic 분할 (Partitioning)

그림 2 의 XOR Logic 의 각 출력은 CRC_Code n 비트 전체 또는 그 중 몇 비트와, Din w 비트 전체 또는 그 중 몇 비트와의 XOR 로 구성된다. 따라서, 그림 3 과같이 CRC Code XOR Logic, Data XOR Logic, XOR Array, 이와 같이 크게 세 로직으로 분리될 수 있다. CRC Code XOR Logic 은 매 클럭마다 업데이트 되는 값이 피드백되기 때문에 파이프 라인으로 구현하기 어렵다. 따라서, Data XOR Logic 을 여러 파이프 라인 단계로 나누되, 각 단계의 로직의 로직 레벨이 CRC Code XOR Logic 의 로직 레벨의 이하가 되도록 함으로써, w 의 사이즈가 증가하더라도 보다 높은 주파수의 클럭을 사용 할 수 있다.

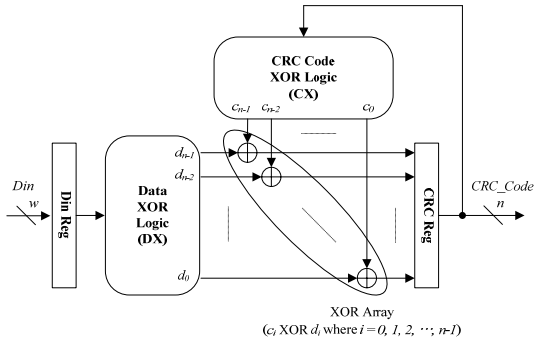

그림 3. Partitioned XOR Logic

그림 3 의 분리된 두 로직 CRC Code XOR Logic 과 Data XOR Logic 을 각각 CX 와 DX 로 놓고, 식 (2)를 $w > n$ 인 경우로 확장하여, 분할된 두 XOR Logic 구현을 위한 식을 유도한다. 우선, 정수 $i = 0, 1, \ldots, n-1$ 에 대해서, 병렬 입력 데이터 Din, CX 의 결과 Cr, DX 의 결과 Dr, CRC 결과 CRC_Code 를 각각,

$$\begin{aligned} &Din = [din_{w-1}\ din_{w-2} \ldots din_0]^T, \\ &Cr = [c_{n-1}\ c_{n-2} \ldots c_0]^T, \\ &Dr = [d_{n-1}\ d_{n-2} \ldots d_0]^T, \\ &CRC\_Code = [crc_{n-1} crc_{n-} \ldots crc_0]^T \end{aligned} \quad (3)$$

라고 정의하면, $crc_i(t+1) = c_i(t) \oplus d_i(t)$가 된다. 즉, 다음 상태에 CRC Reg 에 저장될 $CRC\_Code(t+1)$ 은 현재 $Cr(t)$와 $Dr(t)$의 XOR 에 의해 생성되므로 다음과 같은 식을 얻을 수 있다.

$$CRC\_Code(t+1) = Cr(t) \oplus Dr(t) \quad (4)$$

CX 는, Din 의 원소가 모두 0 인 경우에 해당되므로 Dr 또한 모두 0 이 되어 식 (2)와 (4)로부터 $Cr(t)$는 다음과 같이 쉽게 유도된다.

$$Cr(t) = F^w \otimes CRC\_Code(t), (CRC\_Code(t+1) = Cr(t)) \quad (5)$$

DX 는 w 가 n 이상이므로, 식 (2)에 적용하기 위해서는 w-비트의 입력 데이터를 n-비트 단위로 그룹화 할 필요가 있다. 만일, w 가 정확하게 n 으로 나누어 떨어진다면, 즉, k 가 양의 정수라 할 때, $w = n * k$ 이면, 다음과 같이 Din 을, 원소의 개수가 n-비트인 k 개의 그룹으로 나눈다.

$$\begin{aligned} &Din^1 = [din_{w-1}\ din_{w-2} \ldots din_{w-n}]^T \\ &Din^2 = [din_{w-n-1}\ din_{w-n-2} \ldots din_{w-2n}]^T \\ &\quad\vdots \qquad\qquad\qquad\qquad \vdots \\ &Din^k = [din_{w-(k-1)n-1}\ din_{w-(k-1)n-2} \ldots din_0]^T \end{aligned} \quad (6)$$

만일, w 가 n 으로 나누어 떨어지지 않는다면 즉, j 를 n 이하의 양의 정수라고 할 때, $(w = n * (k-1) + j)$ 이면, Din^k 에 아래와 같이 j 개의 0 을 포함시킴으로써 Din^k 를 n 비트의 한 그룹으로 구성한다.

표 1. w 에 따른 CX 와 DX 의 로직 레벨

Type of CRC		CRC16-A (ITU-TSS)				CRC16-B (HDLC)				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	N	L	N	L	N	L	N	L
w (*bits*)	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*

-. ITU-TSS = International Telecommunications Union - Telecommunication Standardization Sector, HDLC = High-Level Data Link Control, CX = CRC Code XOR Logic, DX = Data XOR Logic, N = number of input bits affecting the critical path, L = logic level of the critical path, w = Parallel input data width.

$$Din^k = [din_{w-(k-1)n-1}\ din_{w-(k-1)n-2}\ \dots\ din_0 \mid 0\ \dots\ 0]^T \quad (7)$$

이와 같이 나눈 입력 데이터 그룹을 이용하여 DX 를 구현하기 위하여, LFSR 에 데이터를 쉬프트 입력을 할 때, 매 사이클 후 LFSR 에 저장되어 있는 결과가 다음 입력 될 데이터에 대해서 LFSR 의 초기값이 됨을 이용한다. LFSR 에 저장되는 변수값 $V = [v_{n-1}\ v_{n-2}\ \dots\ v_0]^T$ 라 하고, $D'in^k = F^n \otimes (V \oplus Din^k)$ 라고 정의하자. DX 에서는, Cr 이 0 이므로, 초기값은 $V = [0\ \dots\ 0]^T$ 이 된다. 따라서, Din^1 에 대해서 $V = [0\ \dots\ 0]^T$ 이 되고, Din^2 에 대해서 $V = D'in^1$ 이 되어, Din^k 에 대해서 $V = D'in^{k-1}$ 이 됨을 알 수 있다. 이와 같은 과정을 식 (2)에 적용하면 다음과 같다.

$$\begin{aligned} D'in^1 &= F^n \otimes ([0\ \dots 0]^T \oplus Din^1) = F^n \otimes Din^1, \\ D'in^2 &= F^n \otimes (D'in^1 \oplus Din^2), \\ &\vdots \qquad\qquad \vdots \\ D'in^k &= F^n \otimes (D'in^{k-1} \oplus Din^k). \end{aligned} \quad (8)$$

여기에서, 최종적으로 얻은 $D'in^k$ 가 곧 $Dr(t)$가 된다. 다시 정리하면, $Dr(t)$는 다음과 같이 나타낼 수 있다.

$$\begin{aligned} Dr(t) = F^n \otimes (F^n \otimes \dots (F^n \otimes ((F^n \otimes Din^1(t)) \\ \oplus Din^2(t))) \dots \oplus Din^k(t)). \end{aligned} \quad (9)$$

3.2 Data XOR Logic 분할

그림 3 의 Data XOR Logic 을 분할하면, 그림 4 와 같이, 분할된 작은 로직과 레지스터의 연속으로 구현될 것이다. 각 로직의 로직 레벨과 레지스터 크기를 어떻게 결정하느냐가 파이프 라인 병렬 CRC 회로 구현의 주요 핵심이 된다.

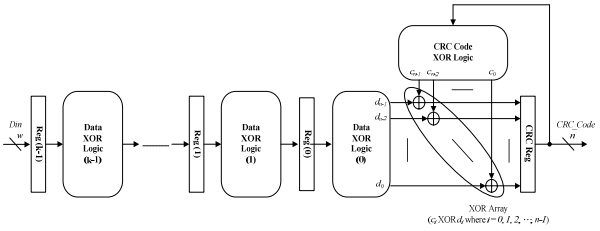

그림 4. Partitioned DXs

로직 게이트 연결정보 (ex) net list)를 분석하여 적절한 위치에 레지스터를 삽입하는 방법을 고려할 수 있지만, 회로가 크고 복잡해지면 체계적인 방법이 없다면 매우 어려울 것이다. 따라서, 이 장에서, 각 로직의 로직 레벨 과 레지스터의 크기를 결정하기 위한 체계적이면서도 간단한 방법을 제시한다.

3.2.1 Logic Level Partitioning

그림 3 에서, DX (Data XOR Logic)의 로직 레벨이 CX (CRC Code XOR Logic)의 로직 레벨 이하라면, DX 분할이 필요하지 않을 수도 있다. 그러므로, 우선, 구현하고자 하는 CRC 회로의 CX 와 DX 의 로직 레벨을 분석하여 DX 분할의 필요성 여부를 파악하여야 한다. 표 1 에, 많이 사용되고 있는 세 가지 16 비트 CRC 와 하나의 32 비트 CRC 각각에 대하여, 입력 데이터 폭 w 가 32, 64, 128, 256 비트인 경우에 대해서, CX 와 DX 의 임계경로를 구성하는 입력 수 N 과, 로직 레벨 L 을 정리하였다. 단, 모든 로직이 전 이진 트리 (complete binary tree)로 합성된다고 가정하면, L 은 다음과 같이 계산된다.

$$L = \lceil \log_2 N \rceil \quad (10)$$
($\lceil\ \rceil$: 소수점 첫째 자리에서 올림)

Case1 의 경우에, DX(0)의 로직 레벨을 4 로 하면, CX 과 로직 레벨이 같지만 라우팅 복잡도나 fan-out 등의 차이로 인해 CX 보다 큰 딜레이가 발생할 가능성이 있다. *Case5* 의 경우에도, DX(1)을 5 으로 하면 CX + XOR Array 와 로직 레벨이 같아 마찬가지 문제가 발생할 가능성이 있으므로, *Case2* 와 *3* 중 하나를 선택함으로써 딜레이 문제를 미연에 방지 할 수 있다. CRC32, w = 32 인 경우는, CX 의 L 과 DX 의 L 이 같으므로, 로직 레벨의 관점에서만 볼 때는 파이프 라인으로 구현 할 필요가 없을 수도 있다. CRC16-B, w = 128 인 경우에는, 가능한 조합이 하나밖에 없으므로, 반드시, DX(0)의 로직 레벨은 3, DX(1)의 로직 레벨은 4 가 되도록 분리해야 할 것이다.

3.3.3 Register Size Decision

$i = 0, 1, \dots, k-1$ 이라고 하고, 각 로직의 로직 레벨을 DL(*0*), DL(*1*), …, DL(*k-1*)라고 정의하자. DL(*i*)는 결국, DX 의 임계경로의 로직 레벨이 분할된 결과 이므로, 우선, DX 의 임계경로에 해당되는 한 출력을 기준으로

표 2. Performance Comparisons

Type of CRC		CRC16-A			CRC16-B			CRC16-C			CRC32		
Architecture		Basic	Pipeline		Basic	Pipeline		Basic	Pipeline		Basic	Pipeline	
Measure		Delay (ns)	Delay (ns)	Red. (%)	Delay (ns)	Delay (ns)	Red. (%)	Delay (ns)	Delay (ns)	Red. (%)	Delay (ns)	Delay (ns)	Red. (%)
w (bits)	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*

-. Basic: Basic Parallel CRC circuit; Pipeline: Pipelined Parallel CRC circuit; Delay: Data arrival time on critical path; Red.: Delay Reduction

하여 각 로직의 입출력에 필요한 레지스터의 크기를 결정하고, 나머지 출력에 대해서도 각각 그와 같은 방법을 적용하도록 한다.

출력 *Dr* 중 임계경로의 출력을 *dc* 라 하고 DX(*i*)의 입력 중에서 *dc* 에 영향을 미치는 입력 수를 *wc*(*i*)라고 하자. *wc*(*k-1*)은 *Din* 중에서 *dc* 에 영향을 주는 입력 수가 된다. 예를 들어, CRC16-A, *w* = 32 인 경우, *wc*(*k-1*)은 18 이다. *wc*(*i*) 중에서 DX(*i*)의 임계경로에 영향을 주는 입력 수를 *wdc*(*i*)라고 하면, 식 (10)에 의해 *wdc*(*i*)는 다음과 같이 나타낼 수 있다.

$$2^{DL(i)-1}+1 \le wdc(i) \le 2^{DL(i)} \qquad (11)$$

각 DX 의 입력 레지스터를 그림 4 와 같이, Reg(*0*), Reg(*1*), …, Reg(*k-1*)이라 할 때, Reg(*i*) 중에서 *dc* 에 영향을 주는 부분을 Reg*dc*(*i*)라고 하자. DL(*i*)는 Reg*dc*(*i-1*) 중 임계경로에 해당되는 한 레지스터에 대한 로직레벨이 된다. 따라서, Reg*dc*(*i*)는 다음과 같이 구할 수 있다.

$$\text{If } i = k\text{-}1,\ \text{Reg}dc(i) = wc(i).$$
$$\text{Otherwise, Reg}dc(i) = \lceil wc(i+1) / wdc(i+1) \rceil. \qquad (12)$$

단, *wdc*(*i*)의 범위가 식 (11)과 같으므로, *wdc*(*i*) = $2^{DL(i)}$로 계산함으로써 Reg*dc*(*i*)의 크기를 최소화 할 수 있다.

나머지 다른 출력에 대해서도 각각 이와 같은 과정을 적용하되, 여러 단계의 로직 레벨 분할이 필요하지 않을 경우에는, 임계 경로와 단계를 맞추기 위한 더미 레지스터를 삽입해 주어야 한다. 각각의 출력에 대해 별도로 로직을 구성하고 레지스터를 삽입하므로 면적 오버헤드가 커질 것이다. 따라서, 공유되는 게이트 및 레지스터를 찾아 면적 오버헤드를 줄이는 연구가 필요하다. 그러나, 이에 관한 알고리즘은 NP-Hard 문제로써 휴리스틱이 필요하며, 이 문제를 이 논문의 향후 연구로 둔다.

4. Performance Evaluation

w = 32, 64, 128, 256 인 경우에 대해서 각각 그림 2 와 같이 하나의 로직으로 구현하고, 그림 4 와 같이 파이프라인 구조를 사용하여 구현하여 성능을 분석 및 비교하였다.

TMS 0.25 um 공정 라이브러리를 사용하여 Synopsys Design Analyzer 로 매핑하였으며, Prime Time 을 사용하여 표 2 와 같이 임계경로의 지연시간을 분석을 하였다. 매핑 결과, CRC16-A, w=256 과 CRC16-C, w=256 은 3 단계의 Pipeline 으로 구성 되었으며, 나머지 다른 회로는 모두 2 단계로 구성 되었다. Basic 에서는, *w* 가 증가함에 따라 DX 의 임계경로가 길어져 지연시간이 증가하지만, Pipeline 에서는 DX 를 각 단계의 로직 레벨이 CX 로직의 이하가 되도록 분할 했기 때문에, 임계경로는 CX 에 의해 결정되며 *w* 가 증가하더라도 큰 차이가 없음을 알 수 있다. CRC16-C, w=32 인 경우는 CX 에서 발생하는 팬 아웃 (fan-out)이 많아 w=64 인 경우에 비해 큰 지연시간을 나타냈다.

5. 결론

본 논문은 병렬 CRC 회로를 파이프 라인으로 구현하는 방법에 대해서 설명하였다. 데이터 폭이 폴리노미얼의 차수에 비해 같거나 큰 경우에 대하여, 파이프 라인으로 구현하기 위한 로직 분할 및 레지스터 크기를 결정 알고리즘을 제시하였다. 우선, CRC 전체 로직을 CRC 결과 피드백 부분과 입력 데이터 부분으로 분리한 후, 입력 데이터 부분을 파이프 라인 구조를 사용하여 분할 하되, 각 단계의 로직 레벨이 피드백 부분의 로직 레벨 이하가 되도록 함으로써 속도를 향상시켰다. 간단한 알고리즘을 사용하여 각 단계마다 필요한 레지스터의 크기를 결정하였다. 구현 결과, 최대 약 44%의 성능 향상을 가져왔다.

References

[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.