Crosstalk 제거를 위한 체계적, 저비용의 버스 인코딩 기법

유예신, 김태환
서울대학교 시스템 합성 연구실
{ysryu, tkim}@ssl.snu.ac.kr

A Systematic, Low-cost Bus Encoding for Crosstalk Elimination

Yesin Ryu, Taewhan Kim
SSL Lab, School of Electrical Engineering and Computer Science, Seoul National University

요 약

연결선(interconnect) 사이의 간섭으로 발생하는 crosstalk 지연시간(delay)을 제거하기 위한 두 가지의 방법을 제안 한다. (1) 체계적인 코드를 생성해내는 방법으로 crosstalk 지연시간(delay) 유발 경우를 두 가지의 종류로 분류하여 각각에 대해 버스(bus) 비트 수의 증가에 따른 analytic 한 코드 생성 공식을 유도하였다; (2) 부-버스(sub-bus) 간에 발생하는 crosstalk 지연시간(delay)을 기존의 방법에 비해 보다 효율적으로 제거하는, 즉 추가적인 차단 라인 (또는 complement 비트 라인)를 감소시키는 방법을 제안 한다. 두 연구 결과는 연결선 상의 데이터 전송에 따른 신뢰성, 지연시간 및 전력 소모 증가를 유발하는 crosstalk를 차단하는 엔코딩 기법으로 유용하게 사용될 것으로 보인다.

1. 서론

Deep-submicron (DSM) 시대로 접어들면서 온-칩(on-chip) 설계에서의 연결선(interconnect)과 관련된 많은 문제 중의 하나는 crosstalk 지연시간 (delay)이다. 즉, 인접된 연결선 라인들 간의 간격이 좁아짐에 따라 연결선 간의 capacitance (C_I)가 증가하게 되고, 따라서 한 라인의 전송 데이터 비트(bit)가 변할 때 인접하는 두 라인에게도 영향이 미치게 되였다. 이는 자체 라인에서의 데이터 비트 변화로 생기는 (라인과 substrate 사이의) capacitance (C_L) 보다 더 큰 데이터 전송 지연시간 (delay)과 전력 소모를 필요로 하게 된다 [1]. ([2]의 연구에 따르면, 0.1μm 공정에서 C_I/C_L의 값은 5가 넘는 것으로 알려져 있다.) 라인 간의 capacitance로 인해 생기는 지연시간 중에서 최악지연시간 (worst-case delay)를 초래하는 데이터 전송 비트 패턴들이 있는데 그 핵심적인 것을 포괄적으로 crosstalk 지연시간 (delay)라 부른다 [1, 3, 4, 5]. (Crosstalk 지연시간이 무엇이며 어떤 경우에 발생하는지에 대해서는 앞으로 자세히 설명된다.)

Crosstalk를 완전히 없애거나 줄이기 위한 여러 가지 전송 데이터의 인코딩(encoding) 방법들이 연구되어 왔다. [2]의 연구에서는 두 인접라인 간에 반대방향의 비트변화{↓↑, ↑↓} (여기서, ↓↑ 이란 한 라인에서 0->1로 변할 때 다른 라인에서는 1->0으로 변함을 말함. ↑↓ 는 그 반대 의미를 표시함.) 가 일어나지 않는 코드를 제시하였다. 하지만 {-↑-, -↓-}와 {↓↓↑, ↑↓↓, ↑↑↓, ↓↑↑}는 같은 크기의 지연시간을 가지므로 [5] {-↑-, -↓-}를 제거하지 않고 {↓↑, ↑↓}만 제거한다고 해서 최악지연시간을 줄일 수 있는 것은 아니다. 반면, [5]의 연구에서는 인접된 세 라인 $\{b_{n-1}, b_n, b_{n+1}\}$의 비트 변화의 모든 조합에 대한 지연시간을 정형적인 식으로 구했다. [3]에서는 [5]의 연구 결과를 바탕으로 주어진 지연시간 제약 조건을 만족시키기 위해 금지되어야 하는 비트 변화 패턴을 제시함으로써 전송 지연시간과 데이터 encoder-decoder 면적의 trade-off (교환) 할 수 있음을 보여 주었다.

많은 수의 라인에 해당하는 데이터 변환을 한꺼번에 하는 것은 encoder-decoder 설계의 복잡도와 해당 설계 면적을 증가시키다. [6]의 연구에서는 엔코더의 입력 수가 증가할수록 해당 하드웨어의 복잡도가 지수적으로 증가한다고 밝혔다. 이는 데이터 코딩에 쓰이는 회로 면적과 지연시간을 증가시키므로 [7], L개의 비트(즉 L-bit bus)를 인접된 l-비트씩 m개의 그룹으로 나누어 각각을 k-비트 코드로 변환하여 m 개의 k-비트 부-버스(sub-bus)로 보내는 것이 일반적인 방법이다 [1,3]. 그러나, 이런 경우 부-버스 내에서 발생하는 crosstalk는 제거할 수 있지만 부-버스 간에 생기는 crosstalk는 없앨 수가 없다. 따라서 이를 막기 위한 방법으로 부-버스 사이에 차단(shielding) 라인을 추가하거나 [1], crosstalk가 예상될 시에는 부-버스의 출력을 전부 역 (즉, inverting) 비트 시켜서 보내거나 [3, 8, 9], 인접 부-버스의 경계를 겹치는 방법 [6] 등이 제시되었다. 이러한 연구들은 부-버스 간 crosstalk는 피할 수 있었으나 버스 분할에 따른 또 다른 면적 증가를 초래하게 된다.

본 논문에서는 기존의 crosstalk 제거를 위한 엔코딩 방법의 문제점을 극복하는 새로운 방법을 제시한다. 크게 두 가지의 새로운 기여로 요약된다.

1. 허용되는 데이터 전송 지연시간의 허용 상한 값이 주어졌을 때, [3]의 연구에서 제시된 crosstalk 지연시간 분류에 따라, 수용 가능한 비트 변화 관계를 분석하여 원하는 최소 비트 사용의 코드를 체계적으로 생성하는 방법을 제시한다 (2-3장).
2. 버스 분할에서 기존 방법들에 따른 라인 추가 할당으로 인한 면적 손실을 줄이는 새로운 코딩 기법을 제시한다. 즉, 최소 크기의 코드로 변환하고 동시에 부-버스 사이의

차단 라인을 줄임으로써, 지연시간을 늘리지 않고도 평균 34.7%의 라인 절감 효과를 얻을 수 있음을 보인다 (4장).

이어지는 내용은 다음과 같다. 2장에서는 데이터 전송 지연시간 모델을 제시하여 허용되는 지연시간 상한 값이 주어졌을 때 이를 만족하는 비트 변화 패턴을 제시한다. 3장에서는 비트 패턴을 확장하여 체계적으로 코드를 만드는 방법을 제시한다. 4장에서는 부-버스 사이에도 비트 패턴 규칙을 적용하여 부-버스로의 분할에 따른 부담을 줄이는 방법을 다룬다. 5장에서는 실험 결과를 보이며, 6장에서는 본 연구에 대한 결론을 다룬다.

2. 서론 및 배경

2.1 지연시간(Delay) Model

본 절에서는 먼저 버스의 데이터 전송에 따른 지연시간 모델을 살펴본다. 버스에 있는 현재 전송의 k-비트 데이터를 $d_t = d_t^1 .. d_t^k$라 하고 다음 전송의 데이터를 $d_{t+1} = d_{t+1}^1 .. d_{t+1}^k$ 라 할 때, 이때 나타나는 전송 지연시간은 다음과 같이 구할 수 있다 [5].

$$T(d_t, d_{t+1}) = \max\{T_n(d_t, d_{t+1}) \mid 1 \le n \le k\} \quad (1)$$

$$T_n(d_t, d_{t+1}) = ((C_L + 2C_I)\Delta_n - C_I(\Delta_{n-1} + \Delta_{n+1}))\Delta_n \cdot R_T \quad (2)$$

여기서, R_T 는 라인 저항이고 $\Delta_n = d_{t+1}^n - d_t^n$ 이다. 이를 바탕으로 지연시간을 표 1에 보여주는 6가지로 분류할 수 있다. (- 표기는 천이(0->1, 1->0)가 없음을 말한다.)

표1. Crosstalk 분류 표

Crosstalk class	천이 패턴 (transition pattern)	중간라인에서의 상대적인 지연시간
1	---	0
2	↑↑↑, ↓↓↓	$C_L R_T$
3	-↑↑, -↓↓, ↑↑-, ↓↓-	$(C_L + C_I)R_T$
4	-↑-, -↓-, ↓↓↑, ↑↓↓, ↑↑↓, ↓↑↑	$(C_L + 2C_I)R_T$
5	-↓↑, ↑↓-, -↑↓, ↓↑-	$(C_L + 3C_I)R_T$
6	↓↑↓, ↑↓↑	$(C_L + 4C_I)R_T$

2.2 기본 비트 변화 세트

Crosstalk class가 1, 2, 3에 해당되는 패턴으로만 코드를 만든다면 최대 지연시간은 $(C_L + C_I)R_T$이고 1, 2, 3, 4인 패턴으로 코드를 만든다면 최대 지연시간은 $(C_L + 2C_I)R_T$가 된다. 또한 1, 2, 3, 4, 5인 패턴으로 만든다면 최대 지연시간은 $(C_L + 3C_I)R_T$가 된다. 즉 패턴 가지 수와 지연시간 사이의 trade-off 가 생기게 된다. 이를 바탕으로 만들어지는 코드는 표 2와 같다.

3. 코드 생성 기법

3.1. $2C_I$ 지연시간 이하 조건의 코드 생성 기법

표2에서 제시한 $(C_L + 2C_I)R_T$ 열의 코드 set는 $2C_I$ 이하의 crosstalk 지연시간을 갖는 비트 변화 패턴 코드이다. 이를 2C_basic_set 라고 부르자. 즉,

$$2C\_basic\_set = \{000, 001, 011, 100, 110, 111\} \quad (3)$$

여기서 주목하고자 하는 것은, 코드의 길이가 아무리 늘어나도 인접하는 3개의 라인에서의 비트 천이가 2C_basic_set 안에 포함되어 있다면 $2C_I$를 초과하는 지연시간 천이는 일어나지 않는다. 다시 말해, 만일 $\{b_{n-1}, b_n, b_{n+1}\} \in$ 2C_basic_set $(1 \le n \le k-1)$ 이면, 최대 지연시간은 $2C_I$이다.

표2. 상한 지연시간을 만족시키는 기본 set

Crosstalk class	1~3	1~4	1~5	1~6
상한 delay	$(C_L + C_I)R_T$	$(C_L + 2C_I)R_T$	$(C_L + 3C_I)R_T$	$(C_L + 4C_I)R_T$
코드 set	000	000	000	000
	001	001	001	001
	100	011	010	010
	111	100	011	011
		110	100	100
		111	110	101
			111	110
				111

2C_basic_set 을 살펴보면 {00*, 011, 100, 11*}이다. $\{b_{n-1}, b_n\}$이 {00}이나 {11}이면 b_{n+1}은 0과 1 둘 다 가능하고 $\{b_{n-1}, b_n\}$이 {01}이나 {10}이면 $b_{n+1}=b_n$이여야 함을 알 수 있다. 따라서 4-bit 이상의 code는 다음과 같은 방법으로 만든다. $2C_code_set_k$는 최대 지연시간이 $2C_I$이고 길이가 k-bit의 코드의 집합을 의미한다. 집합 $2C_code_set_k$의 원소를 $2C_code_k$라 하면 이는 k개의 비트를 가진 코드 벡터이고 $\{a_0, a_1, \cdots\cdots a_{k-1}\}$로 구성되어 있다.

$$2C\_code\_set_3 = 2C\_basic\_set = \{000, 001, 011, 100, 110, 111\} \quad (4)$$

$$2C\_code\_set_{(k>3)} = \{\{\bar{x}, 0\}, \{\bar{x}, 1\}, \{\bar{y}, y_{(k-2)}\} \mid \bar{a} \in 2C\_code\_set_{k-1} \text{ and if } a_{k-3} = a_{k-2} \text{ then } \bar{a} = \bar{x}, \text{else } \bar{a} = \bar{y}\} \quad (5)$$

여기서 k은 엔코딩된 코드의 비트 수를 뜻한다. 예를 들어 $2C_code_set_4$를 만들려면 $2C_code_set_3$의 원소 중 {000, 011, 100, 111}은 x에 해당되므로 {0000, 0001, 0110, 0111, 1000, 1001, 1110, 1111}을 만들어 낼 수 있고, {001, 110}은 y에 해당되므로 {0011, 1100}을 만들어 낸다. 따라서 $2C_code_set_4$는 10개의 원소를 갖는 집합 {0000, 0001, 0110, 0111, 1000, 1001, 1110, 1111, 0011, 1100}가 된다.

표3은 코드의 길이에 따른 코드 set를 보여 준다. 코드 set의 원소 개수는 입력으로 받을 수 있는 데이터 길이를 결정하는 중요한 요인이다. 예를 들어 3-bit의 코드로 만들 수 있는 코드 세트가 6개 이므로($[\log_2 6]=2$) 2-bit의 데이터(00,

01, 10, 11)를 입력으로 받아드릴 수 있다.

표3. 2C_code set의 확장

#of code bit	3	4	5
code set	000	0000	00000
	001	0001	00001
	011	0011	00011
	100	0110	00110
	110	0111	00111
	111	1000	01100
		1001	01110
		1100	01111
		1110	10000
		1111	10001
			10011
			11000
			11001
			11100
			11110
			11111
#of code set	6	10	16
#of data bit	2	3	4

이때 코드 set의 개수는 피보나치 수열이라는 일정 규칙을 따른다. 길이가 k-bit이고 최대 crosstalk 지연시간이 $2C_I$인 코드 set의 개수를 Num_2Cset_k라 하면 식 (6)과 같은 규칙을 따른다.

$$Num\_2Cset_k = Num\_2Cset_{k-1} + Num\_2Cset_{k-2} \quad (6)$$

F(k)를 피보나치 수열이라고 했을 때 $Num\_2Cset_k = 2*F(k+1)$ 가 된다. [3]

3.2. $3C_I$ 지연시간 이하 조건의 코드 생성 기법

표2의 $(C_L+3C_I)R_T$ 열의 코드 set는 $3C_I$ 이하의 crosstalk 지연시간을 갖는 비트 변화 패턴이 코드이다. 이를 3C_basic_set이라 하면

$$3C\_basic\_set = \{000, 001, 010, 011, 100, 110, 111\} \quad (7)$$

이는 3.1 절의 $2C_I$ 이하 코드 생성 기법과 마찬가지로 코드의 길이가 아무리 늘어나도 인접하는 3개 라인의 신호가 3C_basic_set 안에 포함되어 있으면 $3C_I$를 초과하는 지연시간을 갖는 천이는 일어나지 않는다. 즉, 만일 $\{b_{n-1}, b_n, b_{n+1}\} \in$ 3C_basic_set $(1 \le n \le k-1)$ 이면 최대 지연시간은 $3C_I$ 이다.

$3C\_code\_set_k$ 은 다음과 같이 재귀 반복적으로 생성할 수 있다. 여기서도 집합 $3C_code_set_k$의 원소를 $3C_code_k$라 하고 이는 k개의 비트를 가진 코드 벡터이고 $\{a_0, a_1, \cdots\cdots a_{k-1}\}$로 구성되어 있다고 하자.

$$3C\_code\_set_3 = 3C\_basic\_set = \{000, 001, 010, 011, 100, 110, 111\} \quad (8)$$

$$3C\_code\_set_{(k>3)} = \{\{\bar{x}, 0\}, \{\bar{x}, 1\}, \{\bar{y}, y_{k-2}\} \quad (9)$$

$$|\ \bar{a} \in 3C\_code\_set_{k-1} \text{ and if } \{a_{k-3}, a_{k-2}\} = \{c_1, c_2\} \text{ then } \bar{a} = \bar{y}, \text{ else } \bar{a} = \bar{x}\}$$

$$\text{If k is even then } c_1 = 0, c_2 = 1 \text{ else } c_1 = 1, c_2 = 0$$

예를 들어 $3C_code_set_4$를 만들려면 $3C_code_set_3$의 원소 중 {001}은 (a_1=0이고 a_2=1이므로) y에 해당되고, 나머지는 x에 해당된다. 따라서 {001}로는 {0011}만 만들 수 있고 나머지 코드로는 각각 두 개씩 만들 수 있다. 즉, {000}으로는 {0000, 0001}을 {010}으로는 {0100, 0101}을 만들 수 있다. 표 4는 코드의 길이에 따른 코드 set를 보여 준다.

표4. 3C_code set의 확장

#of code bit	3	4	5
code set	000	0000	00000
	001	0001	00001
	010	0011	00010
	011	0100	00011
	100	0101	00110
	110	0110	00111
	111	0111	01000
		1000	01001
		1001	01010
		1100	01011
		1101	01100
		1110	01110
		1111	01111
			10000
			10001
			10010
			10011
			11000
			11001
			11010
			11011
			11100
			11110
			11111
#of code set	7	13	24
#of data bit	2	3	4

여기서도 코드 set의 개수는 입력으로 받을 수 있는 데이터의 길이를 결정하는 중요한 요소이다. 길이가 k-bit이고 최대 crosstalk 지연시간이 $3C_I$인 코드 set의 개수를 Num_3Cset_k라 하자. Num_3Cset_k 도 피보나치 수열을 따랐던 Num_2Cset_k처럼 식 (10)과 같은 일정 규칙을 따르게 된다.

$$Num\_3Cset_k = Num\_3Cset_{k-1} + Num\_3Cset_{k-2} + Num\_3Cset_{k-3} \quad (10)$$

4. 부-버스 사이의 crosstalk 지연시간 제거 코딩 기법

4.1 부-버스 사이의 $2C_I$ 초과 지연시간 제거 기법

2C_code_set 에 있는 코드만 사용하여 하나의 부-버스에

해당하는 엔코더에서 나오는 코드는 C_L+2C_I 이상의 지연시간을 갖는 비트 천이 패턴을 제거할 수 있지만, 두 부-버스 경계에서 일어나는 천이에서는 보장할 수 없다. 따라서 부-버스 사이에 crosstalk가 발생할 시에는 두 번째 부-버스는 모든 비트를 역비트 (inverting) 시키고 역비트 되었다는 정보를 담은 complement bit를 추가해서 보내는 방법이 필요하다. 하지만 앞에서 언급했듯이 이는 추가적인 면적의 낭비가 생기게 된다. 그러나 본 연구에서는 제안된 2C_code_set 규칙을 수정함으로써 이 문제를 해결하는 기법을 제안한다. 2C_basic_set은 {00*, 011, 100, 11*}로 구성 되어 있다. 즉 $\{b_{n-1}, b_n\}$이 {00, 11}이라면 b_{n+1}에 어떠한 시그널이 나오든 C_L+2C_I 이상의 지연시간을 갖는 천이 패턴은 생기지 않는다. 따라서 2C_code_set에서 01과 10로 끝나는 코드를 엔코딩의 고려 대상에서 제외하면 부-버스간의 crosstalk는 생기지 않는다. 예를 들어 첫 번째 부-버스는 $2C\_code\_set_3$ 에서 01과 10으로 끝나는 {001, 110}을 제거하여 {000, 011, 100, 111}만 남기고, 두 번째 서브버스는 $2C_code_set_3$에서 01과 10으로 시작하는 {011, 100}을 제거하여 {000, 001, 110, 111}만 남긴다.

그림1. 부-버스 간 $2C_I$ 를 제거하는 코드

부-버스 1

p1(0)	p1(1)	p1(2)
0	0	0
0	1	1
1	0	0
1	1	1

부-버스 2

p2(0)	p2(1)	p2(2)
0	0	0
0	0	1
1	1	0
1	1	1

그림 1에서 보면, 부-버스1의 마지막 두 비트 (p1(1), p1(2))는 00, 11로 만 끝나고 부-버스2의 시작하는 두 비트 (p2(0), p2(1))은 00, 11로만 시작되므로 {p1(1), p1(2), p2(0)}으로는 절대 2C_basic_set 이외의 패턴 (위의 예에서는 010, 011, 100, 101)을 만들어 낼 수 없고, {p1(2), p2(0), p2(1)}으로도 절대로 2C_basic_set 이외의 패턴(위의 예에서는 001, 010, 101, 110)을 만들어 낼 수 없다. 따라서 두 부-버스 사이에는 complement bit 라인이 필요 없게 된다. 게다가 코드 set의 크기가 6개에서 4개로 줄어 들어도 여전히 $[Log_26] = [Log_24]$이므로 2-bit 부-버스의 데이터를 받아들일 수 있기에 전혀 손실이 없다. 즉, 부-버스1과 부-버스2 사이에 complement bit 라인을 없앨 수 있고, 같은 방법으로 부-버스3와 부-버스4 사이에도 역시 complement bit 라인을 없앨 수 있다. 따라서 기존 방법에서 요구되는 complement bit 라인 수 보다 1/2 정도 감소된 양만 필요하다.

4.2. 부-버스 사이의 $3C_I$ 초과 지연시간 제거 기법

3C_code_set도 2C_code_set처럼 코드 set에서 몇 개의 코드를 제거함으로써 버스 분할로 인한 추가 라인 수를 줄일 수 있는 code를 만들 수 있다. 다행히 3C_code_set는 코드 set의 크기가 크기 때문에 더 많은 종류의 엔코더를 만들 수 있다. 또한 [3]에서는 3C_code_set도 2C_code_set 와 같이 역비트 (inverting) 시키는 방법을 사용하였으나 사실 $3C_I$는 굳이 역비트 시키는 방법을 사용하지 않고 단순히 차단 라인을 추가함으로써 제거될 수 있다. (역비트를 이용하는 방법은 엔코더를 복잡하게 만들고 complement bit 라인 역시 데이터처럼 코드로 변환되어야 하기 때문에 동적 전력 소모를 추가적으로 초래한다.)

3.2절에서 언급했듯이 3C_code_set은 인접한 3개의 비트가 {101} 또는 {010}이 아니면 된다. 예를 들어, 버스의 한쪽 끝을 00, 10, 11 로만 끝내고 다음 부-버스의 앞 두 비트를 00, 01, 11로만 사용하면, 두 부-버스간에 {010}이 나오는 경우는 없다. 즉 첫 번째 부-버스는 $4C_code_set_3$에서 01로 끝나는 코드 {001}을 제거하고 두 번째 부-버스는 10으로 시작하는 코드 {100}를 제거해보자.

그림2. 부-버스 간 $3C_I$ 를 제거하는 코드

부-버스 1

p1(0)	p1(1)	p1(2)
0	0	0
0	0	1
0	1	0
1	0	0
1	1	1
1	1	0
1	1	1

부-버스 2

p2(0)	p2(1)	p2(2)
0	0	0
0	0	1
0	1	0
0	1	1
1	1	0
1	1	1

그림 2에서 보이듯이, 부-버스1의 마지막 두 비트 (p1(1), p1(2))는 00, 10, 11로 만 끝나고 부-버스2의 시작하는 두 비트 (p2(0), p2(1))은 00, 01, 11로만 시작되므로 {p1(1), p1(2), p2(0)}으로는 절대 3C_basic_set 이외의 패턴 (위의 예에서는 010)을 만들어 낼 수 없고, {p1(2), p2(0), p2(1)}으로도 절대 3C_basic_set 이외의 패턴 (위의 예에서는 010)을 만들어 낼 수 없다. 따라서 두 서브버스 사이에는 차단 라인이 필요 없게 된다. 게다가 코드 세트 크기가 7개에서 6개로 줄어들어도 여전히 $[Log_27] = [Log_26]$이므로 2-bit의 data를 받아드릴 수 있기에 전혀 손실이 없다.

5. 기존 인코딩 방법과의 비교

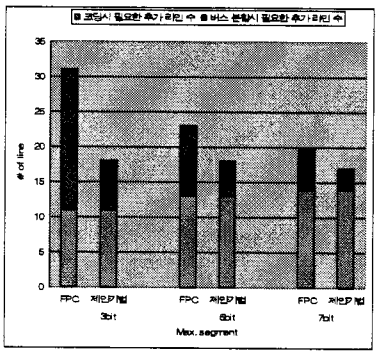
$2C_I$ 와 $3C_I$ 초과 지연시간 제거를 위해 제안한 코딩 기법을 기존의 코딩 기법들과의 비교를 보고자 한다. 4장에서 설명된 코드로 변환시킬 경우 부-버스 간에 crosstalk가 발생하지 않으므로 기존 방법처럼 인위적인 라인을 추가하거나 역비트를 시키지 않아도 된다. 따라서 버스 분할 시 발생하는 상당한 추가 라인 수를 줄일 수 있기에 지연시간을 늘리지 않고도 면적을 줄일 수 있는 가능성을 가져다 준다.

5.1. $2C_I$ 초과 지연시간 제거 인코딩 방법에 대한 비교

그림 3은 본 논문에서 제안한 방법과 $2C_I$ 초과 지연시간 제거를 위한 기존의 특정 코드 패턴 제거 방법의 하나인 FPC(forbidden pattern coding)[3]와의 사용되는 라인 수의 비교를 보여 주는 그래프이다. FPC에서는 코드로 변환하는 과정에서 라인 수의 증가가 발생하고 또한 분할에 따라 부-

버스의 끝 라인의 데이트를 역비트 시키는 과정에서 complement bit에 해당하는 라인 수 증가가 생긴다. 본 논문에서 제안한 방법에서도 코드 변환과정에서 같은 양의 라인 수 증가가 생기나 버스 분할 시 필요한 추가 라인 수는 훨씬 적게 생긴다. 그림3에서 보듯이 32-비트의 데이터를 여러 크기의 부-버스로 분할했을 경우 발생하는 추가 라인 수를 포함한 사용 라인 수를 보여 준다.

그림3. 제안한 $2C_I$ 제거 인코딩 방법과 기존 방법과의 버스 추가 라인 수의 비교

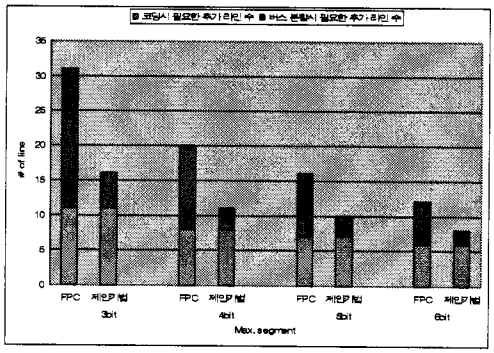


요약하면 평균 26%의 추가 라인 수 감소를 보인다. 여기서 단점은 코드 set의 크기가 크지 않은 경우 효과를 보지 못할 수도 있다는 것이다. 예를 들어 4->5 엔코더는 5-비트의 코드로 16개의 코드를 만들 수 있어 입력으로 4-비트의 데이터까지 받아드릴 수 있었으나 위의 방법으로 코드를 제거하면 13개의 코드가 남게 된다. 그러면 4-비트의 데이터는 5-비트가 아닌 6-비트 코드로 변환해야 하기 때문에 오히려 라인 수의 증가를 초래한다.

5.2. $3C_I$ 초과 지연시간 제거 인코딩 방법에 대한 비교

$3C_I$ 초과 지연시간 제거를 위한 엔코딩 방법인 FPC [3]와 본 논문에서 제안한 방법을 비교한 결과가 그림 4에 보여진다. $2C_I$ 초과 지연시간 제거 방법에서의 비교와 거의 유사하다.

그림4. 제안한 $3C_I$ 제거 인코딩 방법과 기존 방법과의 버스 추가 라인 수의 비교



요약하면 추가 버스 라인 수는 41%가 줄어 들게 되는데 이는 5.1의 경우 보다 많다. 역비트 시키는 방법이 아닌 차단 라인 추가로 부-버스 간의 crosstalk를 해결하였기 때문에 라인 수의 절감이 큰 것이다. 또한 $2C_I$ 초과 제거 경우에서의 단점인 코드 set의 작은 크기가 $3C_I$ 제거 경우에는 대부분 작지 않기 때문에 더 다양한 종류의 엔코더를 만들 수 있다.

6. 결론

본 논문에서는 연결선 사이에서 발생하는 crosstalk를 제거하기 위한 두 가지의 연구 결과를 보이고 있다. 하나는 체계적인 코드를 생성해내는 방법을 고안하였다. 또한, 버스 간에 발생하는 crosstalk를 기존의 방법에 비해 보다 효율적으로 제거하는 방법을 보이고 있다. 기존 방법처럼 부-버스 사이에 추가라인을 인위적으로 삽입하거나 역비트 시키는 방법이 아닌 처음부터 crosstalk가 발생하지 않는 코드로 변환시킴으로써 기존 방법에서 문제시 되었던 추가 라인 수를 상당히 줄일 수 있었다.

두 연구 결과는 심각한 연결선에 대한 신뢰성 및 지연시간, 전력 소모 증가를 유발하는 crosstalk를 차단하는 엔코딩 기법으로 유용하게 사용될 것으로 기대한다.

7. Acknowledgement

본 논문은 2007년도 두뇌한국21 사업에 의하여 지원되었음.

8. 참고

[1] B. Victor and K. Keutzer, "Bus encoding to prevent crosstalk delay," in *Proc. ICCAD*, 2001, pp. 57-63.

[2] S. P. Khatri, "Cross-talk noise immune VLSI design using regular layout fabrics," PhD thesis, UC Berkeley, Dec 1999.

[3] C. Duan, A. Tirumala, and S. P. Khatri, "Analysis and avoidance of cross-talk in on-chip buses," in *Proc. Hot Interconnects*, 2001, pp. 133-138.

[4] H. Kawaguchi and T. Sakurai, "Delay and noise formulas for capacitively coupled distributed RC lines," in *Proc. ASP-DAC*, 1998, pp. 35-43.

[5] P. P. Sotiriadis and A. Chandrakasan, "Reducing bus delay in submicron technology using coding," in *Proc. ASP-DAC*, 2001, pp. 109-114.

[6] S. R. Sridhara, A. Ahmed, and N. R. Shanbhag, "Area and energy-efficient crosstalk avoidance code for on-chip buses," in *Proc. ICCD*, 2004, pp12-17

[7] C. G. Lyuh and T. kim "Low-power bus encoding with crosstalk delay elimination," *IEE Proc. Computers & Digital Techniques*, Vol. 153, No. 2, pp. 93-100, March 2006.

[8] M.R. Stan and W. P. Burleson, "Bus-invert coding for low-power I/O," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, March 1995, (1), pp. 49-58

[9] K. Kim, K. Baek, N. Shanbhag, C. Liu, and S.-M. Kang, "Couplingdriven signal encoding scheme for low-power interface design," in *Proc. ICCAD*, 2000, pp318-321