Bus-Invert 로직변환을 이용한 새로운 저전력 버스 인코딩 기법

준회원 이 윤 진*, Qu Shidi**, 정회원 김 영 철*°

A New Low-Power Bus Encoding Scheme Using Bus-Invert Logic Conversion

Youn-Jin Lee*, Qu Shidi** *Associate Members,* Young-Chul Kim*° *Regular Member*

요 약

UDSM(Ultra Deep SubMicron)기술을 이용한 시스템 온-칩 설계 시 가장 중요한 설계 요소는 버스 상에서의 전력소모와 지연시간을 최소화 하는 것이다. 인접한 선에서 발생되는 crosstalk는 전파 지연을 발생시키는데 지대한 영향을 미치며, 이를 제거하거나 최소화 시키는 일은 SoC(System on a Chip) 설계에서 시스템의 신뢰성 및 성능 향상과 직결된다. 기존의 방법들은 주로 crosstalk 지연이나 버스 스위칭 횟수 중 하나 만을 최소화하는 방법이 제안되었다. 본 논문에서는 인코딩 적용 4 비트 클러스터 상의 버스 스위칭 횟수에 따라 crosstalk과 스위칭 횟수를 동시에 최소화 할 수 있도록 "invert" 기능과 "logic-convert" 기능을 적응적으로 선택하는 새로운 인코딩 기법을 제안한다. 실험결과 제안한 버스 인코딩 기법은 완벽하게 crosstalk 지연를 제거한 반면, 기존의 다른 방법들에 비해 25% 이상 전력을 절약하였음을 보여준다.

Key Words : Crosstalk, Low Power, Switching activity, On-chip, Bus Encoding

ABSTRACT

In ultra-deep submicron technology, minimization of propagation delay and power consumption on buses is one of the most important design objectives in system-on-chip (SOC) design. Crosstalk between adjacent wires on the bus may create a significant portion of propagation delay. Elimination or minimization of such faults is crucial to the performance and reliability of SOC designs. Most of the previous works on bus encoding are targeted either to minimize the bus switching or minimize the crosstalk delay, but not both. This paper proposes a new bus encoding scheme which can adaptively select one of functions "invert" and "logic-convert" according the number of bus switching on an encoded 4-bit cluster. This scheme leads to minimization of both crosstalk and bus switching. In experiment result, our proposed encoding technique consumes about 25% less power over the previous, while completely eliminating the crosstalk delay.

Ⅰ. 서 론

최근 UDSM과 휴대형 가전 시대로 접어들면서 시스템을 하나의 칩에 구현하는 SoC 시스템의 다양한 저전력 기법 적용은 점차 보편화 되고 있다. 시스템 구조 측면에서의 버스사이의 인접한 연결선상의

※ 본 연구는 교육과학기술부 및 한국연구재단의 지역혁신인력양성사업과 부분적으로 지식경제부 및 정보통신산업진흥원의 대학IT연구센터의 지원사업으로 수행된 연구결과임. (NIPA-2011-C1090-1111-0008)

* 전남대학교 정보통신시스템 온칩 연구실 (younjin1204@naver.com, yckim@jnu.ac.kr), (° : 교신저자)

** LGInnotek Electronics(YANTAI)Company, Shan Dong, China (bluewhite1990@yahoo.cn)

논문번호 : KICS2011-09-395, 접수일자 : 2011년 9월 14일, 최종논문접수일자 : 2011년 11월 28일

crosstalk은 데이터 전송지연을 일으킬 뿐 만 아니라 전력소모를 증가시키는 주요 요인이다. 인접된 연결선 간의 간격이 좁아짐에 따라 연결선 간의 정전용량이 증가하게 되고, 이에 따라 한 선의 전송 데이터 비트가 변할 때 인접한 선에도 영향을 미치게 되는데 이로 인해 생기는 지연시간을 crosstalk 지연시간이라고 부른다. 이는 SoC 설계에서 불필요한 시간지연과 전력소모를 유발 시킨다[1,2].

이를 해결하기 위해 crosstalk 지연을 최소화 시키는 연구가 진행되어 왔으나 대부분의 연구는 전력소비에 영향을 미치는 버스 스위칭을 고려하지 않고 crosstalk 지연 최소화에만 주력하였다[1~3,5,8].

Harmander[3]의 연구에서 제안한 버스 인코딩 기법은 최악의 crosstalk("01"에서 "10"으로 변화)을 감소시켰으나 그 방법이 전력을 감소시킬 수 있는지에 대한 여부는 검증되지 않았다. 이와 대조적으로 Stanislaw[4]의 연구에서 제안한 버스 인코딩 기법은 전력을 감소시켰지만 최악의 crosstalk은 제거하지 못했다. Z.Khan[5]와 S.K.Verma[6]의 연구에서 제안한 기법에서는 crosstalk 지연과 전력소비를 동시에 감소시켰으나 최악의 crosstalk을 완벽히 제거하지 못한다. Hsieh와 Chen의 연구[7]에서 제안된 기법은 확률적으로 데이터의 정보를 찾아 인코딩을 수행하는 방법으로 이 기법은 버스 길이가 길어짐에 따라 연산의 복잡성이 기하급수적으로 증대되어 구현상의 어려움이 있다.

본 논문에서는 버스 스위칭 횟수를 최소화 시키는 동시에 crosstalk 문제를 해결하는 새로운 버스 인코딩 기법을 제안한다. 기존의 "bus-invert"기법은 버스 스위칭 횟수를 줄이는데는 최상의 방법이지만, 동시에 crosstalk을 줄이지는 못한다[9]. 본 논문에서는 기존의 "bus-invert" 기법이 majority voter를 통해 "invert" 수행 여부를 결정하는 대신 버스 스위칭 횟수에 따라 crosstalk과 스위칭 횟수를 동시에 최소화 할 수 있도록 "invert" 기능과 "logic- convert" 기능을 적응적으로 선택하는 새로운 인코딩 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 제안한 버스 인코딩 기법이 적용되는 두 가지 crosstalk 유형을 제시하고 에너지 소비 식에 관하여 설명한다. 3장에서는 본 논문에서 제안한 버스 인코딩 기법을 설명하고 4장에서는 제안한 기법의 구현 및 적용에 대해 설명한다. 5장에서는 다른 기법과 비교, 분석 후 6장에서는 결론을 맺는다.

Ⅱ. 관련연구

본 논문에서는 crosstalk 유형을 아래 표 1과 같이 2가지 유형으로 나누어 제안하는 버스 인코딩 기법을 적용한다.

Chunjie Duan은 crosstalk을 type 1에서 4까지 네 가지로 분류하였으며, 이 중 type 2-4는 최악의 crosstalk에 속한다[10]. 여기에서 type 2와 3은 본 논문에서의 type A에 해당하며, type 4는 type B에 해당한다. 에너지 소비 식

$$E=E_{0\to 1}\cdot N_x+\lambda\cdot(4\cdot N_4+3\cdot N_3+2\cdot N_2+1\cdot N_1) \quad (1)$$

에 따르면 에너지 소비에 영향을 미치는 것은 자체 스위칭 횟수와 crosstalk 발생 횟수이다[8]. 여기에서 N_x와 N_i는 각각 자체 스위칭 횟수와 type i의 crosstalk 발생 횟수이고, λ는 정전용량계수를 의미한다. 에너지 소비에 가장 큰 영향을 미치는 것은 type 4의 crosstalk 발생 횟수이며 type 2와 3 또한 영향을 미친다. 이에 따라 기존 알고리즘에서는 type B crosstalk에 비해 상대적으로 type A crosstalk가 무시되는 경향이 있지만, type A 또한 에너지 소비량을 증가시키므로 최소화해야 한다.

또한 Sotiriadis[8]의 연구에서는 버스에서 자체 라인에서의 데이터 비트 스위칭과 인접한 라인에서의 데이터 비트 스위칭을 고려한 대략적인 에너지함수를 제시한다. 그림 1은 세 개의 인접한 연결선을 고려한 집중형 모델이다. 이 모델은 각 유형의 crosstalk을 고

표 1. Crosstalk 유형

유형	비트 변환 정보
type A	0 1 → 1 0
	1 0 → 0 1
type B	1 0 1 → 0 1 0
	0 1 0 → 1 0 1

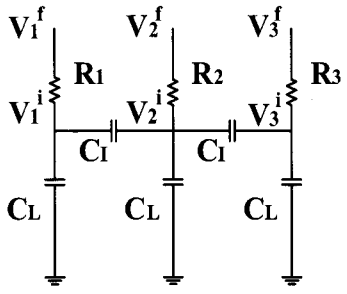


그림 1. 온-칩 버스의 집중형 모델

려하여 에너지 소비량을 계산하기 위해 사용되며 3 비트 버스에 대한 에너지 소비량이 다음과 같이 표현된다.

$$E_1 = C_L\cdot\{(1+\lambda)\cdot(V_1^f-V_1^i)-\lambda\cdot(V_2^f-V_2^i)\}\cdot V_1^f \quad (2a)$$

$$E_2 = C_L\cdot\{-\lambda\cdot(V_1^f-V_1^i)+(1+2\cdot\lambda)\cdot(V_2^f-V_2^i) -\lambda\cdot(V_3^f-V_3^i)\}\cdot V_2^f \quad (2b)$$

$$E_3 = C_L\cdot\{-\lambda\cdot(V_2^f-V_2^i)+(1+\lambda)\cdot(V_3^f-V_3^i)\}\cdot V_3^f \quad (2c)$$

$$E = E_1 + E_2 + E_3 \quad (2d)$$

여기에서 V_1^f, V_2^f, V_3^f는 각각 세 연결선의 최종 전압이고 V_1^i, V_2^i, V_3^i는 초기 전압을 나타낸다. V_1^f, V_2^f, V_3^f, V_1^i, V_2^i, V_3^i 의 값은 공급전압 또는 0이 된다. E_1, E_2, E_3은 세 개의 연결선에 대한 각각의 에너지를 나타낸다. 위 식에서 λ는 데이터 스위칭으로 생기는(연결선과 기판 사이의) 커패시턴스에 대한 연결선 간의 커패시턴스(C_L)의 비율을 의미하며 Sotiriadis 와 Chandrakasan[8]의 논문 내용에 의하면 0.18-μm *CMOS* 공정에서 $\lambda = C_I/C_L = 3.2$ 이다.

식 (2d)는 3 비트 버스에 대한 전체적인 에너지 소비량을 나타낸다. type B crosstalk이 발생했을 경우 에너지 소비는 식 (2d)에 의해 다음 식 (3)과 같이 나타낼 수 있다.

$$E = E_{0\to1}\cdot(1 + 4\cdot\lambda) \quad (3)$$

$E_{0\to1}$ 는 자체 라인에서의 데이터 비트 스위칭 때문에 발생되는 에너지 소비량이며 이는 $C_LV_{dd}^2$ 값과 동일하다. 식 (3)에서 type B crosstalk이 발생했을 경우 에너지 소비는 4λ에 의해 증가된다는 것을 알 수 있다. 또한 식 (3)로부터 우리는 $E/E_{0\to1} = 1 + 4\cdot\lambda$ 식을 유도할 수 있으며, 이것은 1개의 자체 라인에서의 데이터 비트 스위칭과 1개의 type B crosstalk이 발생했을 경우임을 알 수 있다. type B crosstalk이 에너지 소비량에 영향을 미치는 부분은 4λ이다. 따라서 에너지 소비량은 자체 라인에서의 데이터 비트 스위칭과 type B crosstalk 발생에 의존한다는 것을 알 수 있다. 식 (4)는 에너지 절감 양을 표현하는 식이다.

$$Energy\ saving = (1 - N_C/N_U)\cdot 100 \quad (4)$$

위 식에서, N_U는 인코딩 되기 전 발생하는 스위칭 횟수를 나타내고, N_C는 인코딩 된 후 발생하는 스위칭 횟수를 나타낸다.

Ⅲ. 제안한 새로운 버스 인코딩 기법

Sotiriadis[8] 연구의 에너지 소비 식에서 볼 수 있듯이, crosstalk은 온-칩 통신에서 에너지 소비의 중요한 요소이다. 최악의 crosstalk은 신호 천이에서 예측 불가능한 지연을 일으킬 수 있으며 또한 시스템의 문제를 일으킬 수 있는 논리적 오류의 원인이 될 수 있다. 따라서 이를 해결하기 위해서 최악의 crosstalk 커플링을 없애거나 최소화 시키는 것이 필요하다.

"bus-invert" 알고리즘은 모든 버스 데이터의 패턴에 대해 최대 $n/2$번의 변환만을 수행하며 버스 스위칭을 최소화 시키는 다른 연구에서 동일 조건하에 평균 $n/2$번의 변환이 있거나 $n/2$ 이상의 변환을 수행한다는 점을 미루어 보았을 때 "bus-invert" 알고리즘은 최적의 기법임을 알 수 있다. 그러나 전통적인 "bus-invert" 알고리즘은 $n/2$번 이상의 스위칭이 일어날 때만을 고려하기 때문에 4 비트 데이터에서 type B의 crosstalk은 제거 할 수 있는 반면 type A crosstalk은 효과적으로 제거 할 수 없다.

본 논문에서는 기존의 "bus-invert" 알고리즘이 majority voter를 통해 "invert" 수행 여부를 결정하는 대신 crosstalk 발생 여부에 따라 "invert" 수행 여부를 결정하는 방식으로 버스 스위칭을 최소화 시켰으며, crosstalk이 발생되는 경우 "logic- convert"를 통하여 crosstalk을 최소화 하였다.

본 논문에서 제안한 버스 인코딩 기법은 4 비트 이진 시퀀스를 기반으로 하였으며, 제안한 버스 인코딩 기법의 전체 블록 다이어그램은 그림 2와 같다. 여기에서 $x(n)$은 4 비트 크기의 현재 입력 데이터이고 $y(n-1)$은 이전 버스의 데이터 이다. 그리고 $de_info(n)$는 "invert"와 "logic-convert" 수행 여부를 판단하는 정보의 1 비트 데이터이다. 처음 단계로 이전 버스의 데이터 $y(n-1)$과 현재 버스 데이터 $x(n)$의 데이터 변환 정보를 추출하기 위해 XOR 시킨다. 이렇게 출력된 4

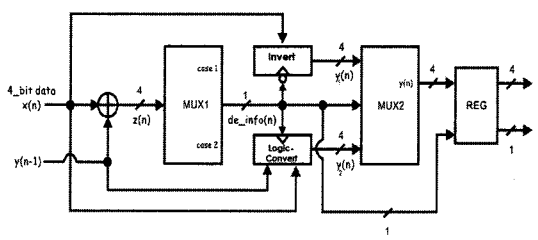


그림 2. 제안한 버스 인코딩 기법의 블록 다이어그램

비트 길이의 출력 $z(n)$이 case 1인 경우는 *de_info*의 값을 '0'으로 할당하고 case 2인 경우는 *de_info*의 값을 '1'로 할당한다. 이는 MUX1을 통해 수행되며 본 논문에서는 case 1과 case 2 두 부분으로 나누는 기준을 제시하였다.

- case 1 : 각 자리의 비트에 모두 "invert"를 수행시켰을 때 버스 스위칭 횟수 최소화가 가능하고, type B crosstalk 최소화가 가능한 데이터 변화를 가지고 있는 경우로 버스 데이터 변환 정보인 z(n)이 "1111", "1110", "1101", "1011", "0111", "1010", "0101", "0110"의 값을 가지고 있을 때 이다.
- case 2 : 버스 데이터 변환이 발생한 비트 수가 2개 이하이면서 첫 번째 비트와 두 번째 비트 또는 세 번째 비트와 네 번째 비트가 연속적으로 데이터 변환이 발생한 경우로 버스 데이터 변환 정보인 $z(n)$이 "1100", "0011", "1001", "0001", "0010", "0100", "1000", "0000"의 값을 가지고 있을 때 이다.

다음 단계로 출력된 *de_info* 정보에 따라서 *de_info*의 값이 '0'이면 "invert"를 수행시키고, *de_info*의 값이 '1'이면 "logic-convert"를 수행시킨다. "invert"의 경우 현재 데이터 각 자리의 비트를 모두 역변환 하여 4 비트 데이터 $y_1(n)$을 출력시킨다. "logic-convert"는 그림 3과 같이 수행된다.

"logic-convert"는 4 비트 데이터 중에서 양 끝자리의 비트 $x_1(n)$과 $x_4(n)$를 제외한 중앙의 2 비트 데이터의 자리를 서로 바꿔 주는 역할을 수행한다. 이는 연속된 자리의 데이터의 스위칭을 막아 type A crosstalk을 제거하기 위한 것으로서 XOR 게이트로 이루어져 있고 "logic-convert"의 출력은 4 비트 크기의 $y_2(n)$가 된다. 최종 버스 인코딩의 출력 값을 위해서 "invert" 또는 "logic-convert"를 통해서 출력된 $y_1(n)$과 $y_2(n)$ 중 최종 $y(n)$ 데이터는 *de_info* 값에 따라 결정 되어야 하며 이는 MUX2를 통해 수행된다.

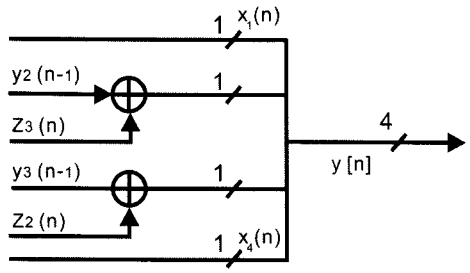


그림 3. Logic-Convert

이로써 제안한 버스 인코딩 기법의 최종 출력은 4 비트의 $y(n)$과 1 비트의 *de_info* 이며 *de_info*는 디코딩 정보로 사용된다. 표 2와 표 3은 제안한 방법이 효과적으로 crosstalk를 제거함을 보여주는 표이다. 표에서 b_i는 i 번째 자리의 비트 값을 의미하며 "-" 는 이전 데이터와 현재 데이터를 비교 했을 때 스위칭이 일어난 경우를 나타내는 기호로 사용되었다. 표 2는 기존의 스위칭 최소화 기법인 "bus-invert" 알고리즘의 수행 전과 수행 후의 crosstalk 발생 여부를 판단하기 위한 표로 여기에서 *de_info*의 값이 '0'인 경우 "invert"가 수행되고 '1' 인 경우는 수행되지 않은 경우를 뜻한다. "bus-invert" 알고리즘의 경우 No. 1, 2, 5 에서 볼 수 있듯이 type B의 crosstalk이 제거됨을 알 수 있다. 그러나 No. 6, 7, 11에서와 같이 이전 데이터와 현재 데이터의 스위칭이 연속적으로 일어날 경우(type A crosstalk이 발생한 경우) "bus-invert"는 3개 이상의 스위칭이 있어야 적용 되므로 여전히 type A crosstalk이 발생됨을 확인 할 수 있다. 표 3은 제안한 버스 인코딩 기법의 수행 전과 수행 후의 crosstalk 발생 여부를 판단하기 위해 제시된 표로 여기에서 *de_info*의 값이 '0'인 경우 "invert"가 수행되고 '1'인 경우는 "logic-convert"가 수행되는 경우를 뜻한다. 표 2에서와 같이 No. 1, 2, 5 에서 type B의 crosstalk이 제거됨을 알 수 있고, "bus-invert" 알고리즘과는 달리

표 2. Bus Invert 알고리즘의 수행 전과 수행 후

No.	이전 데이터	de_info	현재 데이터	
			인코딩 수행 전	인코딩 수행 후
1	$b_3\ b_2\ b_1\ b_0$	0	$\overline{b_3}\ \overline{b_2}\ \overline{b_1}\ \overline{b_0}$	$b_3\ b_2\ b_1\ b_0$
2	$b_3\ b_2\ b_1\ b_0$	0	$\overline{b_3}\ \overline{b_2}\ \overline{b_1}\ b_0$	$b_3\ b_2\ b_1\ \overline{b_0}$
3	$b_3\ b_2\ b_1\ b_0$	0	$\overline{b_3}\ \overline{b_2}\ b_1\ \overline{b_0}$	$b_3\ b_2\ \overline{b_1}\ b_0$
4	$b_3\ b_2\ b_1\ b_0$	0	$\overline{b_3}\ b_2\ \overline{b_1}\ \overline{b_0}$	$b_3\ \overline{b_2}\ b_1\ b_0$
5	$b_3\ b_2\ b_1\ b_0$	0	$b_3\ \overline{b_2}\ \overline{b_1}\ \overline{b_0}$	$\overline{b_3}\ b_2\ b_1\ b_0$
6	$b_3\ b_2\ b_1\ b_0$	1	$\overline{b_3}\ \overline{b_2}\ b_1\ b_0$	$\overline{b_3}\ \overline{b_2}\ b_1\ b_0$
7	$b_3\ b_2\ b_1\ b_0$	1	$b_3\ b_2\ \overline{b_1}\ \overline{b_0}$	$b_3\ b_2\ \overline{b_1}\ \overline{b_0}$
8	$b_3\ b_2\ b_1\ b_0$	1	$\overline{b_3}\ b_2\ b_1\ \overline{b_0}$	$\overline{b_3}\ b_2\ b_1\ \overline{b_0}$
9	$b_3\ b_2\ b_1\ b_0$	1	$\overline{b_3}\ b_2\ \overline{b_1}\ b_0$	$b_3\ b_2\ b_1\ b_0$
10	$b_3\ b_2\ b_1\ b_0$	1	$b_3\ \overline{b_2}\ b_1\ \overline{b_0}$	$b_3\ \overline{b_2}\ b_1\ \overline{b_0}$
11	$b_3\ b_2\ b_1\ b_0$	1	$b_3\ \overline{b_2}\ \overline{b_1}\ b_0$	$b_3\ \overline{b_2}\ \overline{b_1}\ b_0$
12	$b_3\ b_2\ b_1\ b_0$	1	$b_3\ b_2\ b_1\ \overline{b_0}$	$b_3\ b_2\ b_1\ \overline{b_0}$
13	$b_3\ b_2\ b_1\ b_0$	1	$b_3\ b_2\ \overline{b_1}\ b_0$	$b_3\ b_2\ \overline{b_1}\ b_0$
14	$b_3\ b_2\ b_1\ b_0$	1	$b_3\ \overline{b_2}\ b_1\ b_0$	$b_3\ \overline{b_2}\ b_1\ b_0$
15	$b_3\ b_2\ b_1\ b_0$	1	$\overline{b_3}\ b_2\ b_1\ b_0$	$\overline{b_3}\ b_2\ b_1\ b_0$
16	$b_3\ b_2\ b_1\ b_0$	1	$b_3\ b_2\ b_1\ b_0$	$b_3\ b_2\ b_1\ b_0$

표 3. 제안한 버스 인코딩 기법의 수행 전과 수행 후

No.	이전 데이터	de_info	현재 데이터	
			인코딩 수행 전	인코딩 수행 후
1	$b_3\ b_2\ b_1\ b_0$	0	$\overline{b_3}\ \overline{b_2}\ \overline{b_1}\ \overline{b_0}$	$b_3\ b_2\ b_1\ b_0$
2	$b_3\ b_2\ b_1\ b_0$	0	$\overline{b_3}\ \overline{b_2}\ \overline{b_1}\ b_0$	$b_3\ b_2\ b_1\ \overline{b_0}$
3	$b_3\ b_2\ b_1\ b_0$	0	$\overline{b_3}\ \overline{b_2}\ b_1\ \overline{b_0}$	$b_3\ b_2\ \overline{b_1}\ b_0$
4	$b_3\ b_2\ b_1\ b_0$	0	$\overline{b_3}\ b_2\ \overline{b_1}\ \overline{b_0}$	$b_3\ \overline{b_2}\ b_1\ b_0$
5	$b_3\ b_2\ b_1\ b_0$	0	$b_3\ \overline{b_2}\ \overline{b_1}\ \overline{b_0}$	$\overline{b_3}\ b_2\ b_1\ b_0$
6	$b_3\ b_2\ b_1\ b_0$	1	$\overline{b_3}\ \overline{b_2}\ b_1\ b_0$	$\overline{b_3}\ b_2\ \overline{b_1}\ b_0$
7	$b_3\ b_2\ b_1\ b_0$	1	$b_3\ b_2\ \overline{b_1}\ \overline{b_0}$	$b_3\ \overline{b_2}\ b_1\ \overline{b_0}$
8	$b_3\ b_2\ b_1\ b_0$	1	$\overline{b_3}\ b_2\ b_1\ \overline{b_0}$	$\overline{b_3}\ b_2\ b_1\ \overline{b_0}$
9	$b_3\ b_2\ b_1\ b_0$	0	$\overline{b_3}\ b_2\ \overline{b_1}\ b_0$	$\overline{b_3}\ b_2\ \overline{b_1}\ b_0$
10	$b_3\ b_2\ b_1\ b_0$	0	$b_3\ \overline{b_2}\ b_1\ \overline{b_0}$	$b_3\ \overline{b_2}\ b_1\ \overline{b_0}$
11	$b_3\ b_2\ b_1\ b_0$	0	$b_3\ \overline{b_2}\ \overline{b_1}\ b_0$	$\overline{b_3}\ b_2\ b_1\ \overline{b_0}$
12	$b_3\ b_2\ b_1\ b_0$	1	$b_3\ b_2\ b_1\ \overline{b_0}$	$b_3\ b_2\ b_1\ \overline{b_0}$
13	$b_3\ b_2\ b_1\ b_0$	1	$b_3\ b_2\ \overline{b_1}\ b_0$	$b_3\ b_2\ \overline{b_1}\ b_0$
14	$b_3\ b_2\ b_1\ b_0$	1	$b_3\ \overline{b_2}\ b_1\ b_0$	$b_3\ \overline{b_2}\ b_1\ b_0$
15	$b_3\ b_2\ b_1\ b_0$	1	$\overline{b_3}\ b_2\ b_1\ b_0$	$\overline{b_3}\ b_2\ b_1\ b_0$
16	$b_3\ b_2\ b_1\ b_0$	1	$b_3\ b_2\ b_1\ b_0$	$b_3\ b_2\ b_1\ b_0$

No. 6, 7, 11에서 인접한 데이터 변환을 제거함으로써 type A crosstalk이 제거됨을 알 수 있다.

또한 본 논문에서 제안한 버스 인코딩 기법과 이전 논문에서 제안한 버스 인코딩 기법의 로직 overhead를 비교하기 위해 각 기법을 기본적인 AND, OR, NOT 게이트로 구현하여 그 수를 비교하였다. 제안한 버스 인코딩 기법의 "invert" 블록과 "logic-convert" 블록을 게이트로 나타낸 것은 그림 4와 같다. "invert" 블록은 총 4 비트의 변환이 필요하기 때문에 4개의 NOT 게이트로 구성된다. "logic-convert" 블록은 4개의 AND 게이트와 4개의 NOT 게이트, 2개의 OR 게이트로 구성된다. 표 4는 이전 버스 인코딩 기법들과 제안한 버스 인코딩 기법의 게이트 수를 비교한 것으로 표 4에서와 같이 Khan의 기법과 "OE bus-invert"[11]는 제안한 기법에 비해 게이트 수가 월등하게 높은 수치이고, "bus-invert"는 제안한 기법에 비해 게이트 수가 적지만 제안한 기법이 버스 스위칭 최소

표 4. 로직 overhead (게이트 수) 비교

버스 인코딩 기법	AND	OR	NOT
Bus-Invert	24	5	16
Khan의 기법	69	34	58
OE Bus-Invert	80	28	65
제안한 기법	36	11	21

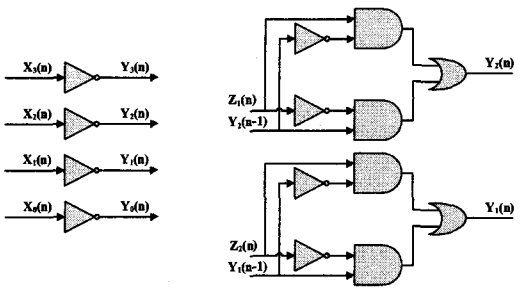


그림 4. 제안한 버스 인코딩에서 게이트 레벨의 Invert와 Logic-Convert

화와 최악의 crosstalk을 제거 한다는 점에서 충분히 상쇄될 수 있는 수치임을 알 수 있다.

또한 로직 overhead뿐만 아니라 로직에서 지연시간을 고려해야 한다. 로직에서 지연을 발생시키는 요소는 게이트 고유 지연, 로딩지연, 입력 회전율 지연, 연결선에 의한 지연과 같이 네 가지로 나누어지며 게이트의 고유 지연은 고정된 값이고, 이를 제외한 다른 지연은 변수와 디자인에 따라 값이 변한다[12]. 따라서 게이트 레벨에서의 "bus-invert" 알고리즘과 제안한 기법의 제안시간 비교 시 게이트 고유 지연시간만을 고려하였다. 제안한 기법은 "bus-invert" 알고리즘과 비교하였을 때 MUX 블록과 "logic-convert" 블록이 추가되지만 실제로 여기에서 지연시간에 영향을 미치는 critical path가 "bus-invert" 알고리즘의 majority voter 블록의 critical path보다 짧아 지연시간은 상대적으로 더 적다. 실제로 CSM 0.18-μm 공정 라이브러리를 사용할 경우 1.8V의 입력 전압과 25°C 환경에서 "bus-invert" 구현 시 지연시간은 약 916ps이며, 제안한 버스 인코딩 기법의 지연시간은 약 695ps 이다. 결과적으로 제안한 기법은 "bus-invert" 알고리즘에 비해 로직 overhead가 약간 증가하지만, crosstalk 최소화 및 로직에 의한 지연시간에 대해서는 훨씬 더 효과적임을 알 수 있다.

Ⅳ. SoC 플랫폼에서의 구현

본 논문에서 제안한 버스 인코딩 기법을 실제 SoC에 적용하는 방법은 두 가지가 가능하다. 첫 번째 방법은 4 비트로 분할된 클러스터 사이에 1개의 비트를 추가하는 방법이고, 두 번째 방법은 4 비트 클러스터 사이뿐만 아니라 디코딩 정보 비트 사이에도 1개의 비트를 추가하는 shied wire 삽입 방법이다. 실제 SoC에 적용시 디코딩 정보 비트와 4 비트 클러스터 사이에서 crosstalk이 발생하므로 첫 번째 방법에 비해 두

번째 방법이 crosstalk 최소화에 더욱 효과적이다[5].

따라서 본 논문에서는 두 번째 방법을 사용하여 기존 버스 인코딩 기법과 제안한 버스 인코딩 기법의 추가적인 선의 수와 전력 절감 효과를 비교하였다.

V. 실 험

제안한 기법은 Intel Core 2 dual 1.8GHz PC기반의 매틀랩 프로그램을 통해 구현되었다. 본 실험에서 우리는 4 비트 크기 10,000개의 랜덤 데이터를 이용하였다. 표 5와 표 6은 제안한 버스 인코딩 방법과 Bus-Invert[9], Khan의 기법[5], Odd/Even Bus Invert[11]의 4 비트 랜덤 데이터와 이미지 데이터에 대한 스위칭 활동을 비교해 놓은 결과이다. 표에서와 같이 제안한 버스 인코딩 기법은 type A crosstalk과 type B crosstalk을 완벽하게 제거시킴을 알 수 있다.

버스 인코딩이 실제 버스에 적용되었을 때의 overhead는 추가적으로 필요한 선의 수로 비교할 수 있다. 제안한 기법과 기존 기법 간의 전송 버스 비트의 증가에 따른 overhead와 성능을 비교한 결과는 아래 표 7, 8과 같다. 본 실험에서는 전송 버스 비트 데이터를 4 비트씩 나눈 후에 버스 인코딩 기법을 적용하였다. 또한 각 4 비트 폭의 데이터와 인코딩 정보 데이터 사이에는 4장에서 제시한 추가 shield wire를 삽입하였다. 표 7과 같이 제안한 기법의 경우 "bus-invert"나 Khan의 기법과 동일하게 "OE bus-invert" 알고리즘에 비해 요구되는 선의수가 적고, 전송 버스 폭이 증가할수록 그 효율이 높아진다는 것을 알 수 있다. 반면 표 8에서 보는 바와 같이 제안한 기법은 "bus-invert" 알고리즘 보다 약 40% 더 추가적인 전력 절감 효과가 있음을 알 수 있다.

또한 제안한 버스 인코딩 기법을 다양한 전송 폭의 버스에 적용할 경우, 버스 데이터를 각각 4 비트 크기로 분리하여 적용할 수 있다. 이로 인해 발생되는 추가적인 crosstalk에 대해서는 4 비트 데이터 사이와 디코딩 정보 비트 사이에 shield wire를 삽입하여 해결할 수 있고 표 8에서의 실험 결과와 같이 버스 폭이 커질수록 제안한 기법의 전력 절감 효과는 커짐을 알 수 있다.

표 5. 4 비트 랜덤 데이터에 대한 스위칭 활동

버스 인코딩 기법	자체 스위칭	type A	type B
원래 데이터	19940	6834	608
Bus-Invert	15631	4478	0
Khan의 기법	14592	3033	0
OE Bus-Invert	12022	4262	0
제안한 기법	12496	0	0

표 6. 4 비트 이미지 데이터에 대한 스위칭 활동

버스 인코딩 기법	자체 스위칭	type A	type B
원래 데이터	18862	6432	608
Bus-Invert	15221	4622	0
Khan의 기법	14060	3025	0
OE Bus-Invert	11846	4277	0
제안한 기법	12096	0	0

표 7. 전송 버스 폭에 따라 추가적으로 필요한 선의 수

버스 인코딩 기법	추가 선의 수				
	4비트	8비트	16비트	32비트	64비트
원래 데이터	0	0	0	0	0
Bus-Invert	2	5	9	23	47
Khan의 기법	2	5	9	23	47
OE Bus-Invert	4	9	19	39	74
제안한 기법	2	5	9	23	47

표 8. 전송 버스 폭에 따른 버스 인코딩 기법의 전력 절감 효과

버스 인코딩 기법	전력 절감 효과 (%)									
	4 비트		8 비트		16 비트		32 비트		64 비트	
	랜덤 데이터	이미지 데이터	랜덤 데이터	이미지 데이터	랜덤 데이터	이미지 데이터	랜덤 데이터	이미지 데이터	랜덤 데이터	이미지 데이터
원래 데이터	0	0	0	0	0	0	0	0	0	0
Bus-Invert	37.4	30.1	37.9	31.2	38.1	37.3	38.8	38.2	38.8	39.9
Khan의 기법	49.8	48.0	50.2	51.3	51.5	53.3	52.2	54.5	54.8	55.7
OE Bus-Invert	46.2	43.1	48.2	49.2	50.4	48.8	50.2	49.7	53.7	50.3
제안한 기법	77.0	76.8	77.2	75.3	78.9	75.8	78.9	77.4	79.2	77.9

Ⅵ. 결 론

본 논문에서는 UDSM 공정의 온-칩 시스템에서 직면하는 crosstalk 지연이 발생시키는 에너지 손실과 지연문제를 해소시킬 수 있는 새로운 버스 인코딩 기법을 제안하였다. 이전의 인코딩 기법과 달리 제안한 기법은 버스 스위칭을 최소화하여 소비전력을 최소화 시키는 동시에 crosstalk 지연을 최소화 시킨다. 또한 실험을 통하여 본 연구에서 제안한 버스 인코딩 기법이 이전의 인코딩 기법들에 비해 crosstalk 발생 횟수가 확연하게 줄었으며 적은 전력을 소모하는 결과를 보였다.

참 고 문 헌

[1] N. Hanchate, N. Ranganathan, "A line- ar time algorithm for wire sizing with simultaneous optimization of interconnect delay and crosstalk noise," *VLSI Design, 5th International Conference on Embedded Systems and Design., 19th International Conference on,* Jan, 2006.

[2] T. Zhang, S. S. Sapatnekar, "Simultaneous shield and buffer insertion for crosstalk noise reduction in global routing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst,* 15(6), pp.624-636, Jan, 2007.

[3] Harmander Singh, Richard Brown, "Dynamically Pulsed MTCMOS With Bus Encoding for Reduction of Total Power and Crosstalk Noise," *IEEE Trans. On Very Large Scale Integration (VLSI) System,* 18(1), pp.166-170, Jan, 2010.

[4] J. Piestrak Stanislaw, Olivier Sentieys, "Designing Efficient Codecs for Bus-Invert Berger Code for Fully Asymmetric Communication," *IEEE Trans. On Circuits and System-II, Express Briefs,* 57(10), pp. 777-781, Oct, 2010.

[5] Z. Khan, T. Arslan, A. T. Erdogan, "Low power system on chip bus encoding scheme with crosstalk noise reduction capability," *Computer and Digital Techniques, IEEE Proceedings,* 153(2), pp.101-108, Mar, 2006.

[6] S. K. Verma, B. K. Kaushik, "Crosstalk and Power Reduction Using Bus Encoding in RC Coupled VLSI Interconnects," *IEEE Third International Conference on Emerging Trends in Engineering and Technology,* pp. 735-740, Nov, 2010.

[7] W. W. Hsieh, P. - Y. Chen, C. Y. Wang and T. T. Hwang, "A Bus-Encoding Scheme for Crosstalk Elimination in High-Performance Processor Design," *IEEE Trans. Comput- Aided Des. Integr. Circuits Syst.,* pp. 2222-2227, Dec, 2007.

[8] P. P. Sotiriadis, A. Chandrakasan, "Low power bus coding techniques considering inter-wire capacitance," *Proc. IEEE Custom Integrated Circuits Conf., CICC 2000,* pp. 507-510, 2000.

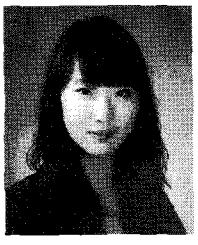
[9] M. R. Stan, W. P. Burleson, "Bus-Invert Coding for Low Power I/O," *IEEE Trans on VLSI System,* 3(1), pp.49-58, Mar, 1995.

[10] Chunjie Duan, Anup Tirumala, S. P. Khatri, "Analysis and avoidance of crosstalk in on-chip buses," Hot Interconnects, pp. 133-138, Sep, 2001.

[11] Yan Ahang, J. Lach, K. Skadon, M. R. Stan, "Odd/Even bus invert with two-phase transfer for buses with coupling," *Proc. IEEE Low Power Electronics and Design ISLPED,* pp.80-83, 2002.

[12] K. Siomalas, "Standardizing Delay Calculation in Verilog," *Verilog HDL Conference, Proceedings., 1995 IEEE International,* pp. 49-55, Mar, 1955.

이 윤 진 (Youn-jin Lee) 준회원

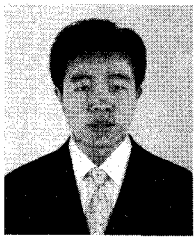


2010년 2월 전남대학교 전자컴퓨터공학부 졸업

2010년 2월~현재 전남대학교 컴퓨터공학과 석사과정

<관심분야> SoC, 저전력 설계

Qu Shidi 준회원



2010년 2월 Jilin University 전기공학과 졸업
2011년 2월 전남대학교 컴퓨터공학과 석사
2011년 2월~현재 LG 이노텍 근무
<관심분야> SoC, 저전력 설계

김 영 철 (Young-Chul Kim) 정회원

1981년 2월 한양대학교 전자공학과 졸업
1987년 2월 Univ. of Detroit 전자공학과 석사
1993년 2월 Michigan state Univ. 전자공학과 박사
1993년~현재 전남대학교 전자컴퓨터공학부 교수
<관심분야> 임베디드 SoC 설계, 저전력 설계, 영상 관련 SoC 및 VDP 설계