저전력과 크로스톡 지연 제거를 위한 버스 인코딩
(Bus Encoding for Low Power and Crosstalk Delay Elimination)

여 준 기 † 김 태 환 ††
(Chun-Gi Lyuh) (Taewhan Kim)

요 약 딥 서브 마이크론 (DSM: deep-submicron) 설계에서는 버스에서 선들 간의 커플링 효과는 크로스톡 지연, 노이즈, 전력 소모와 같은 심각한 문제를 야기 시킨다. 버스 인코딩에 대한 대부분의 이전 연구들은 버스에서의 전력 소모를 최소화하거나 크로스톡 지연을 최소화하는데 초점을 맞추고 있지만 모두를 고려한 방법은 보이지 않는다. 이 논문에서, 우리는 버스에서의 전력 소모 최소화와 크로스톡 지연 방지를 동시에 고려한 새로운 버스 인코딩 알고리즘을 제안하였다. 우리는 이 문제를 공식화하여, 자체 천이와 상호 천이의 가중 합 문제를 풂으로써 해결하였다. 여러 벤치마크 설계를 이용한 실험으로부터 제안한 인코딩 방법을 이용할 경우, 크로스톡 지연을 완전히 제거할 뿐 아니라 이전의 방법들을 사용한 것 보다 최소 15% 이상 적은 전력을 소모하였음을 보았다.

키워드 : VLSI CAD, 버스 인코딩, 저전력, 크로스톡 지연 제거

Abstract In deep-submicron (DSM) design, coupling effects between wires on the bus cause serious problems such as crosstalk delay, noise, and power consumption. Most of the previous works on bus encoding are targeted either to minimize the power consumption on bus or to minimize the crosstalk delay, but not both. In this paper, we propose a new bus encoding algorithm that minimizes the power consumption on bus and eliminates the crosstalk delay simultaneously. We formulate and solve the problem by minimizing a weighted sum of the self transition and cross-coupled transition activities on bus From experiments using a set of benchmark designs, it is shown that the proposed encoding technique consumes at least 15% less power over the existing techniques, while completely eliminating the crosstalk delay.

Key words : VLSI CAD, Bus Encoding, Low Power, Crosstalk Delay Elimination

1. 서 론

전력 소모와 지연 시간은 칩 내부 버스 설계에서 가장 중요한 두 가지 설계 요소이다. 버스에서의 동적 전력 소모의 두 가지 주요 원인은 부하 전기용량(loading capacitance)과 상호 전기용량(coupling capacitance)이다. 딥 서브 마이크론 공정에서는 상호 전기용량은 부하 전기용량에 비해 크기 때문에, 전선의 천이 지연 시간은 안정적인 전선의 천이 지연 시간의 두 배까지 커지게 된다. (이에 의한 지연 시간을 *크로스톡 지연(crosstalk delay)*이라고 한다.)

버스에서 각 비트 라인의 천이 활동 수를 줄임으로써 버스에서의 동적 전력 소모를 크게 줄일 수 있다[1]. 이렇게 버스에서 천이 활동을 줄이는 연구 결과들이 많이 수행되었다. 주로 인접한 값으로 바뀌는 인스트럭션 주소 버스에 대해서는 그래이 코드[2], T0 코드[3], Beach 코드[4]들이 만들어졌다. 또한, 일반적으로 임의의 값으로 볼 수 있는 데이타 버스에 대해서 버스-인버트 코드[5]가 제안되고 있다. 그리고 [5]의 몇 몇 확장된 방법들이 제안되었다. [6]에서는 버스의 각 라인의 천이 확률을 분석함으로써 버스 라인들을 두 부분으로 나누고, 각각의 부분에 대해 버스-인버트 방법을 적용하였다. [7]에서는 [5]의 연구를 더욱 확장하여서 버스 라인을 여러 개의 부분으로 나누고, 이들 각각의 부분들에 대해 독립적으로 버스-인버트 코딩 방법을 적용하였다. 이 두 연구에서는 단지 버스에서의 자체 천이 활동만을

• 이 논문은 2001년도 한국학술진흥재단의 지원에 의하여 연구 되었음 (KRF-2001-041-E00222).
† 학생회원 : 한국과학기술원 전자전산학과
cglyuh@vlsisyn.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전자전산학과 교수
tkim@cs.kaist.ac.kr
논문접수 : 2002년 8월 27일
심사완료 : 2002년 10월 10일

고려하였다. 최근에 [8]의 연구에서는 전력 소모를 줄이기 위하여 버스의 각 비트 라인간의 상호 천이 활동을 최소화 하고자 하였다. 더 나아가 [9]의 연구에서는 상호 전기용량을 줄이기 위하여 버스의 각 비트 라인의 순서를 결정하는 방법을 제안하였다.

반면에, 지연 시간을 최소화해야 하는 설계에서는 크로스톡 지연이 클락 속도를 제한한다. 이러한 크로스톡 지연을 막기 위한 가장 간단한 방법은 버스의 모든 신호 선들 사이에 그라운드(GND) 선을 추가하는 것이다. 이 방법이 크로스톡 지연을 완전히 제거하지만, 버스의 크기 면에서 추가 비용이 너무 크게 된다. 크로스톡 지연을 제거함과 동시에 좀 더 적은 수의 추가 전선을 사용하기 위하여 최근에 Victor와 Keutzer[10]는 "자체-보호(self-shield) 인코딩" 개념을 제안하였다. 그러나 이 연구에서는 버스에서의 전력 소모를 최소화하는 것과 결합된 자체-보호 인코딩을 만들어내는 방법이나 어떤 지침을 주고 있지는 않다.

이러한 모든 인코딩 방법 [2, 3, 4, 5, 6, 7, 8, 9, 10]은 (1) 버스에서의 동적 전력 소모를 최소화 하거나 (2) 크로스톡 지연을 최소화 하고자 하였으나 이 둘을 동시에 고려하지는 않았다. 이 제약을 넘어서기 위해서, 우리는 버스에서의 전력 소모를 최소화할 뿐 아니라 크로스톡 지연을 완전히 제거하는 새로운 칩 내부 버스 인코딩 알고리즘을 제안하였다. 우리는 자체-보호 인코딩 개념을 바탕으로 하여 자체 천이 활동과 상호 천이 활동의 가중 합을 최소화 하는 문제를 풂으로써 (1)과 (2)를 동시에 해결하고자 하였다. 여러 벤치마트 설계를 통한 실험으로부터, 제안한 버스 인코딩 알고리즘을 이용함으로써 [5]의 버스-인버트 인코딩 방법(먼저 (1)을 최소화 한 후, 보호 전선을 추가)에 비해 매우 효율적인 전력 소모 결과를 얻을 수 있었다.

2. 크로스톡 지연이 없는 저전력 버스 설계

2.1 상호연결 전력 모델

상호연결선에서의 동적 전력 소모는 C_s와 C_l을 부하 전기용량, C_c을 상호 전기용량, V_{dd}를 인가전압, f_c을 클락 주파수라고 할 때 다음의 식 1으로 계산할 수 있다[8].

$$P_{dyn}=(X\cdot(C_s+C_l)+Y\cdot C_c)\cdot V_{dd}^2\cdot f_c \qquad (1)$$

X와 Y는 다음의 방법으로 구할 수 있다.

■ **자체 천이 활동** X는 부하 전기용량 C_s와 C_l에 해당하는 자체 천이 활동을 나타낸다. $p^i_{r,s}$를 버스의 신호선 i가 상태 $r\in\{0,1\}$에서 상태 $s\in\{0,1\}$으로 바뀌는 천이 확률을 나타낸다고 하자. 이 때, 신호선 i에 대한 X_i는 다음과 같이 계산할 수 있다.

$$X_i=p^i_{\{0,1\}}$$

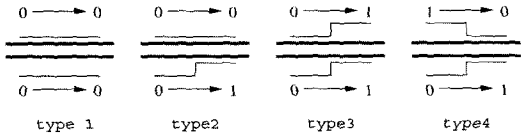

그림 1 두 비트 라인의 신호 천이 관계: 스위칭이 없는 경우 (*유형 1*), 하나의 선이 스위칭 되는 경우 (*유형 2*), 두 라인이 같은 상태로 스위칭 되는 경우 (*유형 3*), 두 라인이 다른 상태로 동시에 스위칭 되는 경우 (*유형 4*).

■ **상호 천이 활동** Y는 상호 전기용량 C_c에 대한 상호 천이 활동을 나타낸다. 이 상호 천이 활동은 두 개의 인접한 비트 라인 사이의 스위칭 활동 관계에 영향을 받는다. 두 비트 라인의 신호 천이 관계는 그림 1에서 보여 주는 것과 같이 네 가지 유형으로 분류된다. *유형 1*은 인접한 두 전선의 상태가 바뀌지 않는 경우이다. 이 경우, 동적 전력 소모는 상호 전기용량 C_c에 의해 영향을 받지 않는다. *유형 2*는 인접한 두 전선 중 하나만 그 상태를 바꾸어서 그 결과 서로 다른 상태가 되는 경우이다. 이 경우, 동적 전력 소모는 α가 상수 인자일 때, $\alpha\cdot C_c$ 만큼 영향을 받는다. *유형 3*은 인접한 두 전선 모두 그 상태를 바꾸어서, 그 결과 같은 상태가 되는 경우이다. 이 경우, 동적 전력 소모는 *유형 1*과 같이 C_c에 의해 영향을 받지 않는다. 마지막으로, *유형 4*는 인접한 두 전선이 그 상태를 바꾸어서 그 결과 서로 다른 상태가 되는 경우이다. 이 경우, 동적 전력 소모는 β가 상수 인자일 때, $\beta\cdot C_c$ 만큼 영향을 받는다. *유형 4*에 따라 받는 영향은 *유형 2*에 의해 받는 영향 보다 크다. 그리고 β 값은 일반적으로 α의 두 배이다. (이에 따라 상호 천이 활동을 계산하기 위하여 우리는 편의상 $\beta=2$와 $\alpha=1$ 값을 사용한다.)

$p^{i,j}_{pq,rs}$가 버스의 신호선 i가 상태 $p\in\{0,1\}$에서 상태 $r\in\{0,1\}$로 바뀌고, 인접한 신호선 j가 상태 $q\in\{0,1\}$에서 상태 $s\in\{0,1\}$로 바뀌는 상호 천이 확률을 나타낸다고 하자. 이 때, 신호선 i와 j 사이의 $Y_{i,j}$는 다음과 같이 계산할 수 있다.

$$Y_{i,j}=\alpha\cdot\sum_{s=0,1}(p^{i,j}_{ss,01}+p^{i,j}_{ss,10})+\beta\cdot(p^{i,j}_{01,10}+p^{i,j}_{10,01})$$

마지막으로, W를 버스의 비트 라인 수라고 할 때, 버스에서의 *식 1*의 동적 전력 소모를 계산하기 위한 자체 천이 활동 X와 상호 천이 활동 Y는 다음과 같이 계산할 수 있다.

$$X=\sum_{i=1}^{W} X_i, \quad Y=\sum_{i=1}^{W-1} Y_{i,i+1}$$

2.2 자체-보호 인코딩

크로스톡 지연은 인접한 전선이 동시에 서로 다른 상태로 천이가 일어날 때 (즉, 그림 1의 *유형 4*) 나타나게 된다. 이러한 *유형 4*에 해당하는 스위칭은 간단하게 버스의 모든 인접하는 비트 라인 간에 보호(GND) 선을 넣어줌으로써 제거할 수 있다. 이 단순한 방법이 크로스톡 지연을 제거하는데 효과적이라 할지라도, 라우팅 밀집과, 전선이 차지하는 넓이로 인한 비용이 두 배로 늘어나는 문제가 있다. 최근에, Victor와 Keutzer[10]는 *유형 4*에 해당하는 스위칭 활동을 제거함과 동시에 늘어나는 전선 비용을 줄이는 *자체-보호 인코딩*이라는 개념을 소개하였다. 기본 아이디어는 버스를 따라 전송되는 비트 폭이 n인 최초의 코드를 인코딩 된 코드 순서에서 *유형 4*에 해당하는 스위칭이 없는 비트 폭 m이 $2n$보다 작은 코드로 인코딩 하고자 하는 것이다. 이 때, 이러한 인코딩 된 코드를 *코드단어*라고 한다.

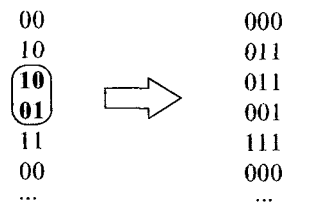

그림 2 자체-보호 인코딩의 개념을 나타내는 예

그림 2는 인코딩 아이디어를 명확히 보여주는 간단한 예이다. 비트 폭 2인 실제 데이타 단어들의 순서가 그림 2에서와 같이 00, 10, 10, 01, 11, 00, … 라고 할 때, 크로스톡 지연 (즉, *유형 4*에 해당하는 스위칭)은 10과 01사이의 천이에서 나타난다. 그러나 만약 데이타 단어 00, 01, 10, 11이 각각 000, 001, 011, 111로 인코딩 되어 전송된다면 그림 2의 오른쪽에서 볼 수 있는 바와 같이 *유형 4*에 해당하는 스위칭은 존재하지 않게 된다. 이 때, 코드단어의 비트 라인 수는 3(즉, $m=3$)이다. 사실, 만약 단순한 보호 방법을 사용했다 하더라도 비트 폭은 여전히 3이다. 그러나 [10]은 최초의 데이타 단어를 위한 버스의 크기가 커짐에 따라 자체-보호 인코딩 개념에 의한 방법에서 필요로 하는 비트 라인 수는 단순한 보호 방법에 비해 훨씬 작아지게 된다. 그러나 [10]에서 자체-보호 인코딩의 개념이 이론적으로 잘 연구되었다 할지라도, 그들은 전력 소모를 최소화하기 위해 자체-보호 인코딩을 만들어내는 지침이나 방법을 주고 있지는 않다.

2.3 문제 정의

전기용량 비 γ를 $\gamma=\frac{C_c}{C_s+C_l}$로 정의하자. 이 값 γ는 집적도가 커짐에 따라 커지게 된다. 그리고 *식 1*의 동적 전력 소모는 다음 식의 값 Z에 비례한다.

$$Z=X+\gamma \cdot Y \tag{2}$$

결국, 우리는 크로스톡 지연을 제거함과 동시에 Z 값을 가능한 줄일 수 있도록 데이타 단어를 인코딩 하고자 하는 것이다. 이 때, 어떤 자체-보호 인코딩 결과를 사용한다 하더라도 크로스톡 지연을 막을 수는 있으나, 어떤 자체-보호 인코딩 결과가 다른 결과 보다 전력 소모를 줄이는데 효과적인지에 대한 연구는 수행된 바 없다. 이런 관점에서, 본 연구에서는 전력 소모가 최소화되는 자체-보호 인코딩 결과를 찾고자 하였다. 우리는 이를 위해 모의실험 결과에 기반을 둔 프로그램의 데이타 순열의 천이 특징을 이용하였다. 즉, 본 논문에서 다루는 저전력 인코딩 문제는 입력으로 버스의 각 비트 라인의 데이타 값의 천이 확률 집합과 버스의 라인 수 (즉, m)를 받아서 *식 1*의 P_{dyn} 값을 최소화하는 자체-보호 인코딩 결과를 내는 것이다. 그리고 이 P_{dyn}을 최소화 한다는 것은 *식 2*의 자체 천이 활동과 상호 천이 활동의 가중 합 Z를 최소화 하는 것과 같다.

2.4 알고리즘

우리는 어떤 자체-보호 인코딩 결과가 최소의 전력 소모를 가지게 되는지에 대해 알아보기 위해 자체-보호 인코딩의 여러 예를 들고자 한다. 그림 3(a)는 데이타 값의 모든 쌍 간의 천이 확률을 보여준다. 예를 들어 그림 3(a)의 천이 확률 그래프에 따라 버스의 데이타 값이 01에서 00으로 바뀔 확률은 0.5이고, 00에서 01로 바뀔 확률은 0.04이다. 이제, 2비트 데이타 단어를 3비트 코드단어로 인코딩 하는 경우를 보도록 하자. 그림 3(b)는 각 노드는 가능한 코드단어를 나타내고, 각 노드 사이에 크로스톡 지연이 없는 경우 (즉, *유형 4*에 해당하는 스위칭이 없는 경우)일 때 에지가 존재하는 3비트 코드단어 그래프를 나타낸다. 결국, 저전력 자체-보호 인코딩은 그림 3(b)의 코드단어 그래프로부터 크기가 4인 클릭을 찾고, 코드단어를 초기 코드에 대응시키는 것

이다. 하지만, 크기 m인 클릭을 찾는 문제는 NP-complete 문제에 해당한다.

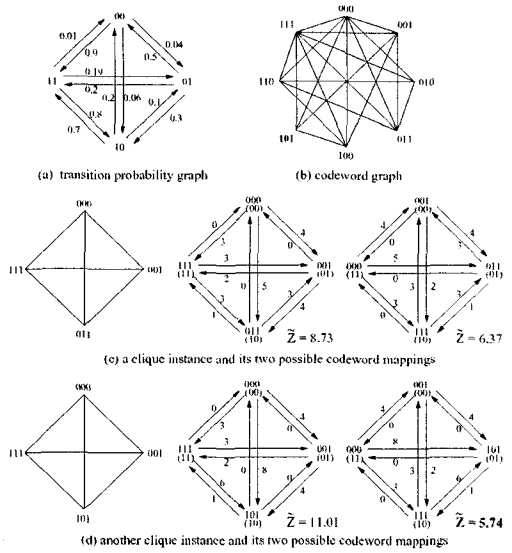

그림 3 코드단어 선택이 전력 소모에 끼치는 영향을 보여주는 예

그림 3(c)의 왼쪽 그림은 그림 3(b)의 코드단어 그래프에 크기 4인 하나의 클릭을 보여준다. 여기에서, 우리는 코드단어로 000, 001, 011, 111이 선택되었음을 볼 수 있다. 기호 A를 초기 코드 데이타의 집합, 기호 B를 코드단어의 집합이라고 하자. 이 예에서, A는 {00, 01, 10, 11}이고, B는 {000, 001, 001, 010, 011, 100, 101, 110, 111}이다. 그리고 함수 f는 A에서 B로의 onto-사상 함수이다. 그림 3(c)에서는 두 가지 가능한 사상을 나타낸다. 처음 것은 $f(00)=000$, $f(11)=111$, $f(10)=011$, $f(01)=001$이고, 두 번째 것은 $f(11)=000$, $f(10)=111$, $f(01)=011$, $f(00)=001$이다. 우리는 이 코드단어 사상 함수 f와 코드단어 클릭을 *인코딩 그래프*라고 한다. 인코딩 그래프에서 데이타 단어 c_i로부터 c_j로의 에지 가중치 $w(f(c_i), f(c_j))$는 c_i와 c_j가 연속으로 전송될 때의 자체 천이와 상호 천이의 가중 합을 나타낸다. 예를 들어, $\gamma=3$이라고 두었을 때, 000이 전송된 후 001이 전송되는 경우에 자체 천이가 1이고 상호 천이가 1이므로 $Z=X+\gamma\cdot Y=1+3\cdot 1=4$이므로 $w(000, 001)=4$이다. $p(c_i, c_j)$가 천이 확률 그래프에서 c_i로부터 c_j로의 천이 확률이고, Z가 다음과 같을 때, 인코딩 그래프 G_{en}와 사상 함수 f에 대해 전체 전력 소모를 측정할 수 있다.

$$Z=\sum_{\forall \text{arc}(c_i, c_j)\in G_{en}} p(c_i, c_j)\cdot w(f(c_i), f(c_j)) \quad (3)$$

예를 들어, 그림 3(c)에서 첫 번째 인코딩 그래프에 대해서는 $Z= p(00, 11)\cdot w(f(00), f(11)) + p(00, 01)\cdot w(f(00), f(01)) + \cdots + p(11, 10)\cdot w(f(11), f(10)) = 0.9\cdot 3 + 0.04\cdot 4 + \cdots + 0.8\cdot 3 = 8.73$이다. 반면에, 그림 3(d)의 오른 쪽 그림은 그림 3(d)의 클릭에 대한 또 다른 두 가지 가능한 인코딩 그래프를 보여준다. 그리하여, 네 가지 인코딩 그래프 중 전력 소모 면에서 가장 효율적인 것은 Z값이 가장 적은 마지막 인코딩 그래프에 해당하는 자체-보호 인코딩이다. 이 예로부터, 전체적으로 가장 좋은 자체-보호 인코딩 예를 찾기 위해 클릭 선택과, 코드단어 사상을 동시에 고려하여야 함을 알 수 있다.

우리가 제안한 알고리즘 EM_shield는 그림 4와 같다. 알고리즘의 입력으로는 천이 확률 그래프 $G(V, E)$, 전기용량 비 γ, 코드단어의 비트 폭 m이 주어진다. 이 알고리즘은 초기화 단계와 개선 단계의 두 단계로 이루어져 있다. 초기 단계에서 코드단어 크기가 m인 코드단어

```
저전력 자체-보호 인코딩 알고리즘( G(A, E), γ, m)
/* (1) 초기화 단계 */
■ m으로부터 코드단어 그래프 G_c(V_c, E_c) 얻기;
■ G_c의 에지 가중치 w(·) 계산;
■ 크기 |A|인 클릭 선택하여, 클릭으로부터 B 찾기;
■ 천이 확률 그래프 G(A, E)를 이용하여 B에서 A로의
    Greedy 사상을 이용하여 인코딩 그래프 G_en 생성;
/* (2) 개선 단계 */
repeat
    ■ G_en에 대한 Z 계산;
    ■ Z_min := Z;
    while ( G_en에 잠겨진 노드가 존재하지 않는 동안)
        ■ min_cost := ∞;
        foreach (잠겨지지 않은 데이타 단어 노드 v∈G_en)
            ■ 노드 v를 클릭을 유지하는 B의 다른 노드로 사상;
            ■ 해당하는 Z 계산;
            if ( Z < min_cost)
                ■ min_cost := Z;
            endif
        endfor
        ■ 노드 v를 min_cost를 가지도록 사상;
        ■ 노드 v 잠금;
        if (min_cost < Z_min)
            ■ Z_min := min_cost;
        endif
    endwhile
    ■ G_en을 Z_min 값을 가지는 인코딩 그래프로 두기;
    ■ G_en의 모든 노드의 잠금 풀기;
until ( Z_min이 더 이상 줄여지지 않은 경우);
■ return ( Z_min 값을 가지는 인코딩 그래프);
```

그림 4 제안된 알고리즘

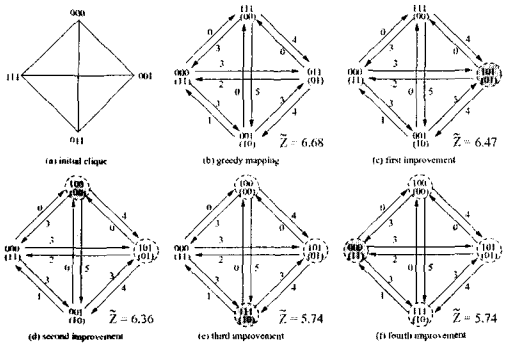

그림 5 그림 3(a)의 데이타 단어에 우리의 제안한 방법을 적용하여 만들어진 단계적 결과

그래프 $G_c(V_c, E_c)$를 생성하고, E_c의 모든 에지에 대한 $w(\cdot)$ 값을 계산한다. 그리고 A가 초기 데이타 단어의 집합일 때 (즉, n을 원래 데이타의 비트 폭이라 하면 $|A|=2^n$이 됨.) G_c로부터 크기 $|A|$의 클릭 하나를 선택한다. 그 다음, 가장 큰 $p(\cdot)\cdot w(\cdot)$ 값을 가지는 에지에 인접한 노드들을 먼저 사상하는 방법으로 A로부터 B로의 (greedy) onto-사상을 수행하고, 나머지 노드들에 대해서 이 작업을 반복한다. 개선 단계에서는 초기 인코딩 결과를 반복적으로 향상시킨다. *while-loop*의 각 반복 수행에서 코드단어의 모든 가능한 재 사상 후보 중에서 $\tilde{Z}$의 값이 가장 적은 것을 선택하여 재 사상한다. 그리고 A의 나머지 데이타 단어에 대해서 이 작업을 반복한다. 바깥쪽 *repeat-loop*은 지역적 극소점을 피하기 위해 두어졌다. *while-loop*의 시간 복잡도는 *foreach*의 각 루프에서 재 사상 시도가 최대 $O(|A|^2)$ 시간이 소요되고, 최대 $|A|$개의 잠겨지지 않은 노드가 있으므로 $O(|A|^3)$에 바운드 된다. 즉, $|A|=2^n$이므로 시간 복잡도는 $O(2^{3n})$이 된다.

그림 5는 그림 3(a)의 데이타 단어에 우리의 제안한 방법을 적용하여 만들어진 단계적인 결과를 보여준다. 그림 5(b)는 초기 단계에 의해 만들어진 인코딩 그래프를 보여준다. 그리고 네 개의 데이타 단어 중, $\tilde{Z}$ 값이 가장 많이 줄어들기 때문에 데이타 단어 01을 선택하여 코드단어 101로 재 사상(mapping) 한 것이 그림 5(c)에 나타내어져 있다. 그림 5(d), (e), (f)는 각각 두 번째, 세 번째, 네 번째 반복에서 00, 10, 11이 100, 111, 000으로 재 사상된 것을 나타낸다. 이러한 반복 수행 동안 만들어진 인코딩 그래프 중에서 그림 5(f)가 가장 적은 $\tilde{Z}$ 값을 가지므로 이것이 선택된다. 그리고 선택된 인코딩 그래프로부터 이러한 작업을 반복 수행한다.

3. 실험 결과

제안한 방법은 C++로 구현되어 Intel Pentium III 컴퓨터에서 수행되었다. 실험을 위해서 다양한 비트 크기를 가지는 무작위 데이타를 이용하여 이전의 인코딩 방법들과 우리의 제안한 방법 ours를 적용하였다. 표 1은 인코딩하지 않은 경우(No-encoding), 버스 인버트 인코딩(Bus-invert encoding)[5], 커플링을 고려한 인코딩(Coupling-driven encoding)[8], 단순한 보호 라인 추가 방법(Naive shielding), 버스 인버트 인코딩[5]과 보호 라인 추가 방법을 결합한 방법, 커플링을 고려한 인코딩[8]과 보호 라인 추가 방법을 결합한 방법, 우리의 제안한 방법에 의해 만들어진 결과를 비교한 것이다. 이 방법들 중에서 나중의 네 가지 인코딩 방법이 다른 것과는 달리 크로스톡 지연을 완전히 제거한 방법들이다. #선은 결과로 만들어지는 버스의 전체 비트 라인 수

표 1 여러 비트 폭을 가지는 무작위 데이타에 대한 이전 인코딩 방법과 우리의 제안한 방법의 결과 비교

방법	8-비트			16-비트			24-비트			32-비트		
	전력	차이(%)	#선	전력	차이(%)	#선	전력	차이(%)	#선	전력	차이(%)	#선
No-encoding (×)	249457	19.36	8	509905	17.04	16	784728	16.54	24	982563	15.93	32
Bus-invert[5] (×)	234579	14.24	9	468525	9.72	17	742151	11.75	25	945136	12.60	33
Coupling[8] (×)	196672	-2.29	9	410374	-3.07	17	671493	2.47	25	842194	1.91	33
Shield (○)	249673	19.43	15	509914	17.05	31	798051	17.94	47	997203	17.16	63
[5]+Shield (○)	228498	11.96	17	480153	11.90	33	746990	12.33	49	981700	15.85	65
[8]+Shield (○)	217897	7.68	17	468301	9.68	33	728163	10.06	49	984031	16.05	65
ours (○)	201170	-	13	422992	-	27	654918	-	31	826075	-	55

× : 크로스톡 지연을 완전히 제거하지 않은 인코딩 방법
○ : 크로스톡 지연을 완전히 제거한 인코딩 방법

를 나타낸다. ours의 시간 복잡도를 고려하여, 이 실험에서는 주어진 데이타를 4-비트 폭으로 자른 후에, 이들 각각에 대해서 우리의 방법을 적용하였다. 구체적으로, 4-비트 데이타를 6-비트 데이타로 각각 인코딩 하였으며, 6-비트 사이에는 추가의 shield 선을 삽입하였다. 또한, shield 선을 지표에 부착한 것으로 하여 논리값 0 이 항상 있는 것으로 간주하고 인접 선과의 커플링 양을 계산하여 결과표들의 전력 수치 란과 *#선* 란에 모두 반영하였다.

이 결과로부터 우리의 방법이 (커플링을 고려한 방법[8]을 제외하고) 이전의 방법들을 이용한 결과 보다 7.6%-19.3% 적은 전력을 소모함을 볼 수 있다. 특히, 우리의 방법은 버스 인버트 방법[5]과 보호 라인 추가 방법을 결합한 방법보다 15.3%-23.5% 적은 수의 라인을 사용하면서 평균 13% 적은 전력을 소모하는 결과를 얻었고, 커플링을 고려한 인코딩 방법[8]과 보호 라인 추가 방법을 결합한 방법보다 약 10% 적은 전력을 소모하는 결과를 얻었다. 물론, 커플링을 고려한 인코딩[8] 방법이 우리의 방법의 결과와 거의 같은 전력을 소모하지만, 커플링을 고려한 인코딩[8]의 방법은 크로스톡 지연을 완전히 제거하지 않은 방법이다.

표 2는 여러 벤치마크 프로그램[11]에 대해, 이전의 크로스톡 지연을 완전히 제거하는 방법과의 전력 소모를 비교하여 보여준다. *#인스트럭션*은 각 프로그램의 인스트럭션 단어의 수를 나타낸다. 이들 벤치마크 프로그램에 대해 인스트럭션 단어들이 버스를 통해 전달될 때의 전력 소모(식 2의 Z 값을 기준으로)를 측정하였다. 이 실험으로부터, 제안한 방법은 다른 이전의 방법들에 비해 47%까지 전력 소모를 줄였음을 볼 수 있다. 각각의 방법과 비교하였을 때, 우리의 방법은 단순한 보호 라인 추가 방법(Shield), 버스 인버트 인코딩 방법과 보호 라인 추가 방법을 결합한 방법([5]+Shield), 커플링을 고려한 인코딩 방법과 보호 라인 추가 방법을 결합한 방법([8]+Shield)에 비해 각각 평균 26%, 23%, 24% 적은 전력을 소모하는 결과를 얻음을 볼 수 있다.

4. 결 론

본 논문에서는 (1) 전력 소모를 최소화 하고, (2) 크로스톡 지연을 제거한 새로운 버스 인코딩 방법을 제안하였다. 이전의 연구들에서는 (1)과 (2)를 동시에 고려한 방법이 없으나, 본 논문에서는 이 둘을 동시에 고려하였다. 특히, 우리는 [10]에서 연구된 자체-보호 인코딩 개념에 바탕을 두고 자체 천이 활동과 상호 천이 활동을 최소화 하는 문제를 풂으로써 버스에서 크로스톡 지연을 완전히 제거하는 동시에, 동적 전력 소모를 최소화 하였다. 그리고 여러 벤치마크 설계를 이용한 실험을 통해 우리가 제안한 버스 인코딩 알고리즘이 이전의 저전력 인코딩 방법에 크로스톡 지연을 완전히 제거하는 방법을 결합한 방법들에 비해 최소 15% 적은 전력을 소모하는 결과를 얻을 수 있음을 보였다. 본 연구의 한 계성으로는 제안한 알고리즘의 시간 복잡도가 데이타 폭의 양에 비례하여 지수 승으로 증가한다는 것이다. 따라서 큰 데이타 폭을 전반적으로 인코딩 할 수 있는 효율적인 알고리즘에 대한 연구가 앞으로 고려되어야 할 것이다.

참 고 문 헌

[1] A. P. Chandrakasan and R. W. Broderson, *Low Power Digital CMOS Design*, Kluwer Academic Publishers, 1995.

[2] C. L. Su, C. Y. Tsui, and A. M. Despain, "Saving Power in the Control Path of Embedded Processors," *IEEE Design and Test of Computers*, Vol. 11, No. 4, pp. 24-30, 1994.

[3] L. Benni, G. De Micheli, E. Macii, D. Scivto, and

표 2 벤치마크 설계에 대한 이전의 보호 방법과 우리의 제안한 방법에 의한 결과의 전력 소모 비교

설계	비트 폭	#인스트럭션	Shield	[5]+Shield	[8]+Shield	ours	감소 율 (%)		
							Shield	[5]+Shield	[8]+Shield
Compress	16	176915	2119353	2107832	2071269	1701000	19.74	19.30	17.88
Laplace	16	323930	4994726	4956873	4856614	4108606	17.74	17.11	15.40
Linear	16	485503	5576707	5816507	5862607	4211227	24.49	27.60	28.17
Lowpass	16	101107	1506380	1522263	1501961	1102983	26.78	27.54	26.56
SOR	32	240592	9856623	6265728	7327673	5179417	47.45	17.34	29.32
Wavelet	16	245	2807	2954	3086	2085	25.72	29.42	32.44
평균							26.98	23.05	24.96

C. Silvano, "Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-Power Microprocessor-based Systems," *Proc. of Grate Lakes Symposium on VLSI*, pp. 77-82, 1997.

[4] L. Benni, G. De Micheli, E. Macii, M. Poncino, and S. Quer, "System-level Power Optimization of Special Purpose Applications, the Beach Solution," *Proc. of International Symposium on Low Power Electronics and Design*, pp. 24-29, 1997.

[5] M. R. Stan and W. P. Burleson, "Bus-Invert Coding for Low-Power I/O," *IEEE Trans. on Very Large Scale Integration(VLSI) Systems*, Vol. 3, No. 1, pp. 49-58, March 1995.

[6] Y. Shin, S. Chae, and K. Choi, "Partial Bus-Invert Coding for Power Optimization of Application-Specific Systems," *IEEE Trans. on Very Large Scale Integration(VLSI) Systems*, Vol. 9, No. 2, pp. 377-383, April 2001.

[7] S. Hong, K. Chung, U. Narayanan, and T. Kim, "Decomposition of Bus-Invert Coding for Low-Power I/O," *Journal of Circuits, Systems, and Computers*, Vol. 10, Nos. 1 & 2, pp. 101-111, 2000.

[8] K.-W. Kim, K.-H. Baek, N. Shanbhag, C. L. Liu, and S.-M. Kang, "Coupling-Driven Signal Encoding Scheme for Low-Power Interface Design," *Proc. of International Conference on Computer Aided Design (ICCAD)*, pp. 318-321, 2000.

[9] Y. Shin and T. Sakurai, "Coupling-Driven Bus Design for Low-Power Application-Specific Systems," *Proc. of Design Automation Conference (DAC)*, pp. 750-753, 2001.

[10] B. Victor and K. Keutzer, "Bus Encoding to Prevent Crosstalk Delay," *Proc. of International Conference on Computer Aided Design (ICCAD)*, pp. 57-63, 2001.

[11] P. R. Panda and N. Dutt, "1995 High-level Synthesis Design Repository," Proc. of International Symposium on System Synthesis, 1995.

김 태 환

서울대학교 계산통계학과 학사. 서울대학교 계산통계학과 석사. Ph.D. Dept. of Computer Science, Univ. of Illinois at Urbana-Champaing, R&D Lattice Semiconductor Corp., USA. R&D Synopsis Inc., USA. 현재 한국과학기술원 전자전산학과 전산학전공 재직중. 관심분야는 Synthesis for System-on-chip design

여 준 기

경북대학교 컴퓨터공학과 학사. 한국과학기술원 전산학과 석사. 한국과학기술원 전자전산학과 전산학전공 박사과정. 관심분야는 Behavioral and Logic Synthesis, High-level Interconnect Synthesis for low power