저전력을 위한 버스-인버트 코딩 분할 기법
(Decomposed Bus-Invert Coding Technique for Low Power)

홍 성 백 † 김 태 환 ††
(Sungpack Hong) (Taewhan Kim)

요 약 이 논문에서 우리는 버스에서의 연속된 데이타 전송 시 발생하는 데이타 값의 천이를 줄이는 새로운 버스-인버트 코딩 기법을 다룬다. 기존의 방식에서는, 버스 라인 전부나 어느 한 일부분이 버스-인버트 코딩에 적용 된 것과는 달리, 우리의 기법은 다양한 버스 분할을 시도하여, 각 분할에 독립적으로 버스-인버트 코딩을 적용하여 전체의 데이타 값 천이를 최소화하고자 한다. 실제 회로를 통한 실험에서 기존의 버스-인버트 코딩과 비교하여 우리의 제안한 기법은 데이타 값의 천이를 전체적으로 10%-50% 수준으로 줄일 수 있음을 보여 준다.

Abstract In this paper, we study a new bus-invert coding scheme for reducing the number of bus transitions. Contrary to the existing bus-invert schemes in which the entire bus lines or only one subset of bus lines are attempted for bus-invert coding, our scheme exploits the possible partitioning of bus lines so that each partitioned group is considered independently for bus-invert coding to maximize the effectiveness of reducing the total number of transitions. Experiments using benchmark circuits show that our proposed scheme produces the number of transitions 10%-55% less compared to the conventional bus-invert coding schemes.

1. 서 론

노트북 컴퓨터나 핸드폰 같은 휴대용 전자기기의 발달로 인해 저 전력 CMOS 회로 설계는 무척이나 중요한 연구 분야가 되었다. 이런 장비들에서는 제한된 배터리 용량을 가지고 있으므로, 회로에서는 실행 속도를 떨어뜨리지 않는 한 가능하면 적은 전력만을 소모하도록 설계되어야 한다. CMOS 회로에서의 전력 소모의 대부분은 게이트에서의 동적 전력 소모에서 기인하며, 그 값은 다음의 공식으로 나타내어 질 수 있다[1].

$$P_{dynamic} = \sum_{i=0}^{N} C_i \cdot V_{dd}^2 \cdot f_i$$

이 식에서 C_i는 게이트 i 에서의 출력 캐퍼시탄스, V_{dd} 는 전압, f_i 는 노드 i 에서의 천이 빈도, N 은 칩에서의 게이트의 총 개수를 나타낸다. 기존의 많은 연구들에서는 단순화를 위해 모든 C_i 가 같은 값을 가진다고 가정하였었다. 그러나 [2, 9]에서 보여지었듯 I/O 연결에 대한 C_i 값들은 칩 내부의 게이트의 C_i 값보다 몇 자리 수 높은 정도의 큰 값을 가지게 된다.

그러므로, 버스 기반 시스템에서는 버스에서의 천이(transition)가 전력 소비의 주된 원인이 된다. 칩 바깥으로(off-chip)의 신호를 전달하는 버스에서의 신호변환에 의한 전력 소모는 전체 전력 소모의 70%에 이르기도 한다[3]. 이러한 사실은 버스의 큰 커패시턴스 값과 큰 오프칩 드라이버에 기인한다. 결국 버스에서의 스위칭 정도를 낮추게 되면 전체 시스템 전력 소비를 획기적으로 줄일 수 있게 된다.

이런 목적을 위해 버스를 통하는 일련의 신호들을 부호화 하는 기법들이 연구되어져 왔다. Gray 코드[4]와 T0 코드[5], Beach 코드[6]는 명령어 주소(instruction address) 버스를 대상으로 고안되었는데, 명령어 주소 값들은 연속적인 값을 많이 가지게 된다는 성질에 착안한 것이다. 버스-인버트(Bus-invert) 코드[2]는 임의의 값을 가질 수 있는 데이타 버스를 대상으로 고안된 것으로 버스에서의 한 번의 최대 천이의 수를 버스의 너비(width)의 절반 이하로 제한한 것이다. 버스-인버트

· 본 연구는 첨단정보기술 연구센터(AITrc)를 통하여 과학재단의 지원을 받았음.
† 학생회원 : 한국과학기술원 전산학과
hongsup@jupiter.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전산학과 교수
tkim@cs.kaist.ac.kr
논문접수 : 1999년 12월 22일
심사완료 : 2000년 10월 21일

코드를 적용하기 위해서는 현재 버스에서의 데이타가 반전(invert)된 것인지 여부를 나타내기 위해, 추가적으로 한 회선이 더 필요하다. 부분적 버스-인버트(Partial bus-invert) 코드 [7]는 데이타 주소 버스를 대상으로 한 것으로, 데이타 주소는 명령어 주소만큼 연속적인 값을 가지지는 않지만, 데이타만큼 임의 값을 가지지는 않는다고 가정되었다. 부분적 버스-인버트 코드는 버스의 데이타 중에 상호 연관성이 높은 한 그룹의 신호에 대해서만 버스-인버트 코드를 적용하도록 하였다.

이 논문에서 우리가 제안하는 부호화 기법은 분할 버스-인버트 (Decomposed bus-invert) 코드로, 이러한 데이타 주소 버스를 대상으로 한 것이다. 이 코드에서는 일단 회로의 수행 내용이 주어지면, 정적으로 통계적 방법을 사용하여 상호 연관성 높은 버스 회선들을 몇 개의 서브 버스들로 분할한다. 이러한 서브 버스들 각각에 대해 서로 독립적으로 버스-인버트 코드를 적용한다.

2. 개 괄

부분적 버스-인버트(PBI) 코드 [7]에서는 버스 회선들은 두 개의 그룹 R, S 로 나뉘어 진다. 그룹 S는 서로간의 천이의 연관성이 높고, 천이의 정도가 큰 버스 회선들로 이루어지고, R은 그 나머지 버스 회선들이다. 실제 회로 동작 시에는 그룹 S에 대해서만 버스-인버트 코드를 적용한다. 부분적 버스-인버트 코드에서의 추가적인 버스 회선은 그룹 S에 해당하는 버스 회선들의 데이타가 반전되었는지 여부를 나타내게 된다. 이 부호화 기법이 가질 수 있는 제한성 중의 하나는 하나의 그룹에 대한 버스-인버트 코드만을 고려대상에 넣었다는 점이다. 이를테면 버스 회선이 B_1에서 B_N의 N개라고 할 때, k개의 버스 회선들이 천이의 상호 연관성이 높아서 하나의 그룹을 이루었다고 하자. 이때 N-k의 나머지 버스 회선들 중에는 앞의 k개의 회선들과는 상호 연관성이 적으나, 남은 회선들 중에서 다시금 서로간에 상호 연관성이 있는 회선들이 있을 수 있다.

여기서 두 버스 회선 사이에 상호 연관성이 있다는 의미는, 두 개의 회선이 동시에 1의 값을 가지게 될 경우가 많다는 뜻이다. 버스-인버트 코드에서는 버스 라인을 통해 전송되는 데이타에서 1의 개수가 전체 버스 회선의 너비의 1/2을 넘을 경우, 데이타 값을 0은 1로, 1은 0으로 반전하여 전송하기 때문에, 상호 연관성이 높은 회선들이 많을 경우 버스-인버트 코드를 통해서 얻는 이익이 높음을 쉽게 알 수 있다.

위에서 설명한 것과 같은 경우, PBI 코드를 연속적으로 적용하는 방법을 생각할 수 있다. 즉 남아 있는 R의 버스 회선에 대해 반복적으로 PBI 코드를 적용하는 것이다. 그러나 이러한 방법의 경우, 한 단계에서의 PBI 코드는 현 상태에서의 최대로 효율을 높일 수 있는 그룹을 찾으므로 버스 회선 전체에 대한 고려가 부족할 수 있다. 게다가 PBI 코드의 경우 가장 좋은 그룹을 찾기 위해 시뮬레이션을 확인 과정을 거치므로, 매 단계마다 이러한 확인 과정을 거치는 것은 수행 시간에 커다란 부담이 될 수 있다.

우리가 제시하는 *분할 버스-인버트(DBI)* 코드에서는 전체 버스 회선을 몇 개의 상호 배제 부분집합들로 분할하되, 각각에 그룹에 대한 버스-인버트 코드의 효과를 전체적으로 고려하면서 한다. 이렇게 함으로서 버스에서의 천이의 개수를 더 줄일 수 있게 된다.

N개의 버스 회선 $B_1, B_2, \ldots B_N$에서 각 버스 회선 $\boldsymbol{b_i}$는 $(b_i^0, b_i^2, \ldots b_i^n)$의 데이타를 전송한다고 해보자. 여기서 n은 데이타가 전성되는 시간 인덱스를 말한다. DBI 코드는 이 회선들을 $S_1, S_2, \ldots S_m$의 m개의 그룹으로 분할하는데, 각각의 그룹에 속한 회선들끼리는 상호 연관성이 높으며 다른 그룹의 회선들과는 상호 연관성이 적게 되도록 한다. 일단 이러한 분할이 결정된 다음, 회로가 동작할 때는 각각의 그룹에 대해서 버스-인버트 코드를 적용한다. 즉, $\boldsymbol{S_i}$의 j번째 시간 인덱스에 전송되는 데이타들은 $\boldsymbol{S_i}$의 j-1번째 시간 인덱스에 전송된 데이타와의 Hamming 거리를 계산하여 그 값이 $|S_i|/2$보다 크게 되면, 0의 값은 1의 값으로, 1의 값은 0의 값으로 반전하여 전송한다. 이때 데이타가 반전이 될 경우 추가적인 *반전 회선*은 1의 값을 전송하도록 한다.

3. 분할 알고리즘

모든 가능한 분할들 중에서, 전체 천이의 개수가 최소가 되도록 하는 최적의 분할을 찾는 일은 계산학적으로 어려운 일이다. 여기에서는 계산 가능한 하나의 휴리스틱을 제안한다. 우리의 알고리즘은 그래프에서 노드를 한 쌍씩 합쳐 나간다는 개념을 가지고 있다. 좀 더 자세히 말하자면, 우선 주어진 버스 회선들과 버스 회선을 통하는 데이타들에 대한 사전 정보를 통해 어떤 (완전) 그래프를 구성한다. 이 그래프에서 노드는 버스 회선을 나타내며 노드 사이의 에지는 두 버스 회선을 한데 묶어서 버스-인버트 코드를 적용할 경우 그 효과가 얼마나 좋을 것인가 하는 예상치의 측도를 나타낸다. 이러한 에지들 중에서 가장 높은 값을 가지는 에지를 선택하여, 그 노드들을 하나로 합친다. 합쳐진 이후의 그래프는 원래의 그래프와는 다른, 하나의 후보 구성 상태가 된다.

Algorithm DBI
· $M=\{S_1, S_2, \ldots S_N\}, S_i=\{B^{i-1}\}$ /* *Initialize* */
· *현재의 구성 상태를 기억한다*
While ($|M| \geq 2$) /* *Decompose* */
· *Gain* 값이 가장 큰 에지에 해당하는 노드쌍 (S_j,S_k)를 찾는다.
· S_j와 S_k를 하나로 합치고, 그 구성상태를 저장 한다.
· *Gain* 값을 새로 계산한다.
endWhile
· 각 구성 상태에서 각 Si에 대해 BI 코드를 각각 적용한다.
· 천이의 개수가 가장 적은 구성 상태를 선택한다.

그림 1 알고리즘의 흐름

이런 식으로 한번에 하나의 에지를 선택해 가면서 일련의 후보 구성 상태들을 만들어 내어, 모든 에지가 사라지고 하나의 노드만 남을 때까지 반복한다. 마지막으로 각각의 후보 구성 상태들 중에서 시뮬레이션을 통해 가장 좋은 구성 상태를 골라낸다. 우리의 분할 알고리즘은 다음과 같이 기술 될 수 있다.

위에서 $S_1, S_2, \ldots S_N$ 은 원래 그래프의 노드를 나타낸다. *Gain*값은 4절에서 정의되는 값으로 그래프의 각 에지에 대해 값이 주어지며 각 노드를 합쳤을 때 버스-인버트 코드를 적용하였을 경우 얻어지는 이득(천이의 개수로 따졌을 때)에 대한 값이다. While 문이 한번 반복될 때마다 *Gain*값이 가장 큰 두 노드를 선택하여 하나로 합쳐진다. 이 알고리즘에 있어서 대부분의 수행시간은 가장 적은 천이 개수를 가지게 되는 구성 상태를 고르기 위해 마지막 한번의 시뮬레이션을 하는데 소비되고, 분할 자체에 걸리는 시간은 매우 작다는데 주목할 필요가 있다.

4. Cost 계산 과정

먼저, 일련의 데이타 순열에 대해 천이 부호를 정의하겠다. j번째 버스 라인에 있어서 i번째 시간 인덱스에 천이 부호 t_i^j는 다음과 같이 정의한다.

$$t_i^j = 1, \text{ if } b_{i-1}^j \neq b_i^j$$

$$t_i^j = 0, \textit{ otherwise}$$

이 천이 부호의 벡터를 천이 벡터라고 하겠다. 그림 2는 8-비트 버스 $[b_0, b_1, \ldots, b_7]$에서 전송되는 데이타 순열을 천이 벡터의 순열로 바꾼 예를 보여주고 있다. 원래 이 천이 벡터만을 고려해도 원래 데이타 순열이 버스-인버트(BI) 코드에 의해 부호화 되어 될 때의 실제 버스에서의 천이의 개수를 세어 볼 수 있다는 사실을 쉽게 알 수 있다. 즉, 천이 벡터에 대해 BI 코드를 적용

b0	10100110	**t0**	11110101
b1	10100011	**t1**	11110010
b2	10100110	**t2**	11110100
b3	10100111	**t3**	11110101
b4	00001100	**t4**	00001010
b5	00001001	**t5**	00001101
b6	00001000	**t6**	00001100
b7	00001101	**t7**	00001011

그림 2 천이 벡터로의 변환

t0	11110101	**t0'**	00000001
t1	11110010	**t1'**	00000110
t2	11110100	**t2'**	00000000
t3	11110101	**t3'**	00000001

그림 3 버스-인버트 코드를 적용한 다음의 천이 벡터

하여 1의 개수를 세는 것은 원래 데이타 순열에 대해 BI 코드를 적용하여 천이의 개수를 세는 것과 같게 된다. 어떤 천이 벡터에 BI 코드가 적용되었을 때, 이 벡터의 1의 개수가 (벡터의 길이)/2 보다 컸다면, 1의 값은 0으로 바뀌고 0의 값은 1로 바뀌는 *반전*이 일어나게 된다.

그림 2의 예를 살펴보면, 하위 4개 회선 $[b_0, b_1, b_2, b_3]$은 천이 벡터에서의 1의 개수가 많으므로, 다른 회선들에 비해 전력 소모가 더 크다는 사실을 알 수 있다. 그림 3은 이 4개 회선에 대해 BI 코드가 적용된 다음의 천이 벡터의 순열을 나타낸다. 우리가 풀고자 하는 문제는 BI 코드를 각 집합에 적용했을 때, 전체 천이의 개수가 가장 작아지도록 하는 버스 회선들의 분할을 찾고자 하는 것이다.

한 쌍의 버스 회선 j_1, j_2에서의 천이 부호의 순열 ($t_0^{j_1}, t_1^{j_1}, \ldots t_{n-1}^{j_1}$), ($t_0^{j_2}, t_1^{j_2}, \ldots t_{n-1}^{j_2}$)이 주어졌을 때, 두 버스 회선의 연관성을 측정하기 위해 다음의 값을 정의한다.

$$P_{ab}(j_1, j_2) = |\{ (t_i^{j_1} = a, t_i^{j_1} = b) \text{ or } (t_i^{j_1} = b, t_i^{j_1} = a) \}|,$$

where $a, b \in \{0, 1\}$

즉, 이를테면 $P_{00}(j_1, j_2)$는 두 회선의 천이 부호가 동시에 0의 값을 가지는 쌍의 개수를 나타낸다. 이제 이 값을 s개의 버스 회선의 집합 $S_j = \{j_1, j_2, \ldots j_s\}$에 대하여 확장하여 다음과 같이 정의한다.

$$P_{ab}(S_j) = \sum_{x=1}^{s-1} \sum_{y=x+1}^{s} P_{ab}(j_x, j_y), \textit{ where } a, b \in \{0, 1\}$$

이 값은 집합 S_j에서 임의의 두 버스 회선이 동시에 (a,b) 값을 가지는 쌍의 개수를 나타낸다. 예를 들어 그림 2에서 $S_1 = \{ t_0, t_1, t_2, t_3 \}$ 이라고 한다면, $P_{11}(S_1) = 28$, $P_{01}(S_1) = 10$, $P_{00}(S_1) = 10$ 이 된다. 버스 회선들의 집합에서의 천이의 개수는 그 천이 벡터에서의 1의 개수와 같으므로 다음과 같은 식으로 표현될 수 있다.

$$F(Sj) = \frac{2 \cdot P_{11}(S_j) + P_{01}(S_j)}{|S_j| - 1}$$

BI 코드가 적용되고 난 다음에는 천이 벡터가 변하므로, 이 값들은 조금 바뀌게 된다. 즉 몇몇의 (1,1) 쌍은 (0,0) 쌍으로 바뀌게 되고, (0,0) 쌍은 (1,1) 쌍으로 바뀌게 된다. (0,1) 쌍의 개수는 변하지 않음에 주목할 필요가 있다. 그림 3에서 BI 코드를 적용하고 난 다음에 $P_{11}(S_1)$의 값은 1로 변하게 된다. BI 코드를 적용하였을 때 이 값들이 변한 결과를 예측하기 위해 다음의 값들을 도입하겠다.

$$\widetilde{P_{11}}(S_1) = P_{11}(S_1) \cdot (1 - prob_{11}) + P_{00}(S1) \cdot prob_{00}$$

$$\widetilde{P_{00}}(S_1) = P_{00}(S_1) \cdot (1 - prob_{00}) + P_{11}(S1) \cdot prob_{11}$$

여기서 $prob_{11}$과 $prob_{00}$은 BI 코드를 적용할 때, 결과적으로 전체 집합에서, 각각 (1,1)쌍 중의 얼마만큼의 양이 (0,0) 쌍으로 반전되는가의 정도와, 그 반대의 정도를 예측한 값을 나타낸다. 원래 천이 벡터에서 (1,1) 쌍이 많을수록 더 많은 반전이 일어날 것임을 알 수 있다. 마찬가지로 (0,0)쌍이 많을 경우에는 (1,1)쌍의 반전이 적을 것이라고 생각 할 수 있다. 이러한 관찰에서 비추어 보아 우리는 $prob_{11}$과 $prob_{00}$을 다음과 같은 식으로 예측하였다.

$$prob_{11} = \frac{P_{11}(S_j) + \frac{1}{2} P_{01}(S1)}{P_{11}(S_j) + P_{01}(S_j) + P_{00}(S_j)}$$

$$prob_{00} = \max \{ \frac{-P_{00}(S_j) + \frac{1}{2} P_{01}(S1)}{P_{11}(S_j) + P_{01}(S_j) + P_{00}(S_j)}, o \}$$

한편 P_{01}값은 BI 코드를 적용한 다음에도 그대로 유지됨을 알 수 있다. 이렇게 생각했을 때, BI 코드를 적용한 다음 전체 버스 회선에서 천이의 수에 대한 예측치는 다음과 같은 식으로 표현된다.

$$\widehat{F(Sj)} = \frac{2 \cdot \widetilde{P_{11}}(S_j) + \widetilde{P_{01}}(S_j)}{|S_j| - 1}$$

결국 어떤 임의의 버스 회선들의 집합 Sj에 대해서 BI 코드를 적용함으로서 감소시킬 수 있게 되는 천이의 수는 다음과 같게 된다.

$$saved(S_j) = F(S_j) - \widehat{F}(Sj)$$

$$= \frac{2 \cdot (P_{11}(S_j) \cdot prob_{11} - P_{00}(S_j) \cdot prob_{00})}{|S_j| - 1}$$

예를 들어서, 그림 3에서 $S_1 = \{ t_0, t_3 \}$, $S_2 = \{ t_1, t_2 \}$ 라고 할 때, $P_{11}(S_1 \cup S_2) = 28$, $P_{01}(S_1 \cup S_2) = 10$, $P_{00}(S_1 \cup S_2) = 10$이므로, $prob_{11}$과 $prob_{00}$은 각각 0.687, 0이 된다. 결국 $Saved(S_1 \cup S_2) = 12.83$ 이 되는데, 그 의미는 $S_1 \cup S_2$를 하나의 그룹으로 BI 코드를 적용할 경우 BI 코드를 적용하지 않은 것에 비해, 12.83개 정도의 천이를 줄일 수 있을 것을 기대한 다는 것이다.

마지막으로 두 집합 *S1, S2*에 독립적으로 BI 코드를 적용했을 때와 $S_1 \cup S_2$를 하나의 그룹으로 적용했을 때의 천이 개수 감소의 차이는 다음의 식으로 표현된다.

$$Gain(S_j, S_k) = Saved(S_j \cup S_k) - Saved(S_j) - Saved(S_k)$$

5. 구현시의 추가 부담

각각의 분할된 버스들에 대해 BI 코드를 적용하기 위해서는 부호화를 위한 추가적인 하드웨어가 필요하다. 이러한 하드웨어는 회로 면적과 수행 시간의 증가를 가져올 수 있다. 그렇지만, 추가적인 내부 회로로 인한 전력 소모량은 I/O 회선에 의한 전력 소모보다 상대적으로 훨씬 적다. 이런 추가적인 내부 회로로 인한 절대적인 면적, 시간 증가량을 재기보다는, 다른 버스 부호화 기법인 BI 와 PBI와 비교했을 때 DBI는 어느 정도의 면적과 시간의 추가 부담을 가지고 있는지를 알아보았다. 이를 위해 [2]에서 제안한 것과 같은 구현에 따라 BI, PBI, DBI 각각에 대한 추가적인 면적, 시간, 버스 회선의 증가량을 예측하여 아래의 표 1에 그 결과를 요약하였다.

n개의 버스 라인에 대한 부호화 회로를 구현하기 위해서는 2n개의 XOR 게이트(버스의 데이타 값을 조건에 따라 반전하기 위해 n개, 현재의 값과 이전 값을 비교하는 데 n개)와 n중의 n/2개 이상의 값에 따르는 다수결 회로가 있으면 된다. 다수결 회로는 전가산기(FA)의 균형 트리로 구현될 수 있다. 다수결 회로의 FA의 개수는 대략 n/2정도 된다. 왜냐하면 n개의 가수(addend)들은 마지막에 n/2개로 줄어들어야 하는데, 하나의 FA는 3개의 가수들을 2개로 줄이기 때문이다. 복호화 회로 쪽은 상대적으로 단순한데, 조건부로 버스의 데이타 값을 반전하기만 하면 되기 때문이다.

위의 표 1에 Area 행은 각 부호화 기법에 대한 XOR 게이트와 FA의 개수를 나타내고 있다. BI와 비교했을 때 DBI는 면적을 전혀 증가시키지 않는다. 표의 Delay 행은 다수결 회로를 구현하는 FA 트리의 높이를 나타

내고 있다. 추가적인 속도 지연 차이는 이 FA 트리의 높이에 비례하며, XOR게이트에 의한 지연은 모든 부호화 기법에 대해 동일하다. 이 행의 값들을 비교해 보면, DBI에 의한 지연은 BI나 PBI에 의한 지연보다 훨씬 작다는 것을 알 수 있다. DBI의 단점은 s-1개[1]의 추가적인 *반전 회선*이 필요하다는 것이다. 실험 결과에 따르면, 이 반전 회선 자체에 의한 전력 소모를 고려에 넣더라도, 작은 수에 s에 대한 DBI 코드의 결과라도 버스에서의 천이의 값을 BI나 PBI에 비해 상당히 줄이고 있다는 사실을 알 수 있다.

표 1 부호화 기법에 대한 추가 부담 비교

(단, 여기서 버스 회선의 수는 N, PBI에서 BI 코드가 적용되는 집합의 크기는 k, DBI에서는 s개의 그룹으로 분할되고 각 부분집합의 크기는 n_1, n_2, .. n_s 단 마지막 그룹은 BI 코드를 적용하지 않는다.)

		BI	PBI	DBI
Area	# XOR	$2N$	$2k$	$2 \cdot \sum_{i=1}^{s} n_i$
	# FA	$N/2$	$k/2$	$\sum_{i=1}^{s} \frac{n_i}{2}$
Delay		$\log_{1.5} N$	$\log_{1.5} k$	$\max\{\log_{1.5} n_1 \ldots \log_{1.5} n_{s-1}\}$
Bus Line		1	1	s-1

6. 실험 결과

우리는 [8]에서 제시된 일련의 벤치마크 설계에 대해 실험을 해 보았다. 이 설계들에 대해 어떤 특정 수행에 대한 데이타 패턴을 추출해서 제시된 알고리즘을 적용해 보았다. 그림 4는 [8]에의 *Laplace* 설계의 데이타 어드레스 패턴에 대한 우리 알고리즘의 적용 결과를 나타내고 있다. 천이의 개수가 가장 적은 구성상태는 3절의 알고리즘의 while문이 24번 반복된 다음에 얻어진 것을 알 수 있다. 우리의 알고리즘의 장점은 주어진 설계의 목표와 제약조건에 맞는 버스 회선의 분할을 설계자가 선택할 수 있도록 해 줄 수 있다는 점이다.

우리는 우리의 결과를 버스-인버트 (BI)와 부분적 버스-인버트 (PBI) 코드에 의한 결과와 비교해 보았다. 이 결과는 표 2에 요약되어 있다. 결과에 따르면 DBI는 BI와 PBI에 비해 천이의 개수를 각각 평균적으로 47.2%와 11.9% 정도 감소시켰다. 표의 맨 마지막 행의 *Wavelet* 설계에서는 버스 회선의 천이의 정도가 작기 때문에 DBI 역시 정확히 2개의 그룹으로 나뉘어지어, PBI와 같은 결과를 나타내었다. 하지만 다른 설계에 있어서는 DBI가 더 좋은 결과를 나타내었다.

표 3은 최대 분할의 개수를 2,3으로 제약했을 때, 수행된 DBI의 결과를 나타낸다. 특히 2개의 분할을 사용했을 때, DBI는 버스 천이의 개수를 BI나 PBI에 비해 평균적으로 42.6%, 4.5%정도 감소 시켰다.

5절에서 이야기 된 바와 같이, 수행 성능은 부호화 하드웨어 구성을 위한 FA 트리의 높이에 비례한다. 그리고 회로의 면적에 있어서는 부호화 대상에 포함되는 버스의 개수에 비례한다. 표 4는 이렇게 5절에서 설명된 수식을 가지고 각 설계에 BI, PBI와 DBI의 결과를 구현하는 데에 필요한 면적과 수행시간의 추가적인 증가를 계산한 것이다. 결과적으로 DBI는 어떤 부호화 기법에 비해서도 수행시간에 있어서 추가 부담이 작을 뿐 아니라, 면적 부담은 PBI보다는 크나 BI에 비해서는 항상 작게 됨을 알 수 있다. 이 표에서 면적부담의 계산에는 lcbg10pv (0.35u) 타켓 라이브러리[10]의 XOR와 FA의 셀면적 값을 Synopsys Design Compiler[11]를 통해 추출하여 사용하였다.

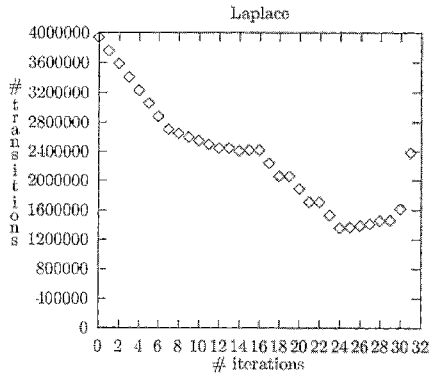

그림 4 *Laplace* 설계에 대한 DBI 알고리즘의 각 단계에서의 천이 개수

표 2 벤치마크 설계에 대한 부호화기법 적용결과 비교

Design	# of Transitions			Redeuction % with respect to	
	BI	PBI	DBI	BI	PBI
compress	1066266	722260	588150	44.8	18.6
Laplace	2377233	1603476	1349402	43.2	15.8
Linear	2402802	1227402	1109002	54.2	9.65
Lowpass	656133	399690	331462	49.5	17.1
SOR	1900735	1343694	1210218	36.3	9.93
Wavelet	1406	626	626	55.4	0
Average				47.2	11.9

1) s개의 그룹 중에 마지막 하나의 그룹에는 PBI에서의 R 그룹처럼, 버스-인버트 코드를 적용하지 않도록 한다. 이 그룹에는 천이의 개수가 적은 버스 회선들이 속하게 된다.

표 3 분할의 최대 개수를 제한하였을 때의 천이의 개수

Design	# of Transitions		Redeuction % with respect to BI	
	2 -Partition	3-Partition	2-Partition	3-Partition
compress	663466	611370	37.8	42.7
Laplace	1460284	1448032	38.6	39.1
Linear	1207402	1144692	50.1	52.7
Lowpass	364422	340757	44.5	48.0
SOR	1347235	1277983	29.1	32.8
Wavelet	626	626	55.4	55.4
Average			42.7	45.1

표 4 부호화 회로의 추가 부담

	Area Overhead			Delay Overhead		
	BI	PBI	DBI	BI	PBI	DBI
Compress	367.84	229.90	310.87	8.548	7.389	6.509
Laplace	367.84	235.90	310.87	8.548	7.509	6.509
Linear	367.84	258.89	281.88	8.548	7.733	6.834
Lowpass	367.84	206.91	304.70	8.548	7.129	6.834
SOR	367.84	206.91	321.81	8.548	7.129	5.679
Wavelet	367.84	275.88	275.88	8.548	7.839	7.839
BI와 비교시 평균 감소량		-35.91%	-18.17%		-12.79%	-21.62%

7. 결 론

이 논문에서 저전력 회로 설계의 한 세부 기법으로, 버스에서의 연속된 데이타 전송 시 발생하는 데이타 값의 천이를 줄이는 새로운 버스-인버트 코딩 기법을 소개하였다. 기존의 방식에서는, 버스 라인 전부나 어느 한 일부분이 버스-인버트 코딩에 적용 된 것과는 달리, 우리의 기법은 다양한 버스 분할을 시도하여, 각 분할에 독립적으로 버스-인버트 코딩을 적용하여 전체의 데이터 값 천이를 최소화를 시도하였다. 실제 실험에 우리의 기법을 적용하여 기존의 버스-인버트 코딩과 비교하여 47%, 부분 버스-인버트 코딩과 비교하여 12%의 평균적 데이타 값의 천이를 줄일 수 있었다.

참 고 문 헌

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

[2] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power I/O," *IEEE Transactions on VLSI Systems*, vol 3, no.1, pp.49-58, 1995

[3] D. Liu and C. Svensson, "Power consumption estimation in CMOS VLSI chips." *IEEE Journal of Solid State Circuits*, Vol. 29, No. 6, pp. 663-670, 1994.

[4] 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 .

[5] L. Benni, G. De Micheli, E. Macii, D. Scivto, and C. Silvano, "Asymptotic zero-transition activity encoding for address busses in low-power microprocessor-based systems," *Proceedings of Grate Lakes Symposium on VLSI*, pp.77-82, 1997

[6] L. Benni, G. De Micheli, E. Macii, M. Poncino, and S. Quer, "System-level power optimization of special purpose applications, the beach solution," *Proceedings of International Symposium on Low Power Electronics and Design*, pp.24-29, 1997

[7] Y. Shin, S. Chae and K. Choi, "Reduction of bus transitions with partial bus-invert coding," *IEE Electronics Letters*, vol 34, no. 7, pp.642-643, 1998

[8] P. R. Panda and N.D. Dutt, "1995 high level synthesis design repository," *Proceedings of International Symposium on System Synthesis*, 1995

[9] M. R. Stan and W .P. Burleson, "Limited Weight Codes for Low Power I/O," *International Workshop on Low Power Design*, pp.209-214, 1994

[10] LSI Logic Inc., *G10-p Cell-Based ASIC Products Databook*, 1996.

[11] Synopsys Inc., *DesignWare Components Databook*, 1996.

홍 성 백

1999년 한국과학기술원(KAIST) 전산학과 학사. 1999년 ~ 현재 한국과학기술원(KAIST) 전산학과 석사과정 재학.

김 태 환

1985년 서울대학교 전산통계학 학사. 1987년 서울대학교 전산학 석사. 1993년 미국 일리노이 주립대 전산학 박사. 현재 한국과학기술원 전기전산학과 전산학전공 부교수. 관심분야는 VLSI 설계 자동화, 계산이론.