논문 2007-44SD-6-9

Flip-Driver를 이용한 효율적인 Bus-Invert Coding 회로의 설계

(A Design of an Effective Bus-Invert Coding Circuit Using Flip-Driver)

윤 명 철*

(Myungchul Yoon)

요 약

본 논문에서는 기존의 Bus-Invert Coding 계통의 여러 알고리즘들이 관습적으로 사용하던 Invert-line 을 제거하고 버스선을 통하여 코딩정보를 전송하는 새로운 설계방법을 제시하였다. Flip-Driver라고 하는 새로운 형태의 버스드라이버와 선택회로를 사용한 이 방식은 Invert-line을 사용하지 않을 뿐 아니라 코딩정보 전송 시 발생하는 추가적인 버스의 전이를 효과적으로 억제한다. 이러한 회로설계방식의 변환만으로도 여러 BI 알고리즘의 효율이 Invert-line을 사용할 때와 비교하여 40%-100% 증대시킬 수 있음을 실험을 통하여 검증하였다.

Abstract

A new circuit design for Bus-Invert Coding is presented in this paper. The new scheme sends the coding information through the bus-lines instead of the invert-line which has been used conventionally for many types of Bus-Invert algorithms. By employing a newly developed bus-driver called Flip-Driver and a selection circuit, it not only removes the invert-line but suppresses the additional bus-transitions in sending coding information. It is verified by simulations that the efficiency of various Bus-Invert algorithms is increased about 40% to 100% by employing the new design.

Keywords: NIL-BIC, Bus-Invert Coding, Low-Power Bus, Flip-Driver

I. 서 론

반도체기술의 발달과 함께 반도체소자 및 기능모듈(Functional Module)들의 크기와 개별회로들이 소비하는 전력은 점점 작아지고 있는 반면 집적도가 높아짐에 따라 하나의 칩에 내장되는 모듈의 수가 증가하여 칩 전체의 크기와 그 소비전력은 점점 증가하고 있다. 칩 안에 집적되는 기능모듈의 수가 증가할수록 이들을 연결하고 상호간의 데이터를 주고받기 위한 통신수단으로써 버스의 기능과 역할이 점점 더 중요해지고 있다. 버스들은 보통 그에 연결되어 있는 많은 수의 모듈들에 의해 높은 부하 커패시턴스 값을 갖게 되므로 동작 시 많은 전력을 소모한다. 또한 근래의 고집적회로 칩들은 전체 칩면적의 반 이상을 버스가 차지하고 있으므로[1] 저전력용 칩을 설계하기 위해서는 버스의 소비전력을 줄이는 것이 매우 중요한 요소가 된다.

버스의 소비전력을 줄이기 위하여 여러 가지 방법이 개발되었다. 이중 Bus-Invert (BI) Coding[2]은 코딩을 통하여 데이터 전송 시 버스선들이 전이하는 횟수를 감소시킴으로써 버스의 동적소비전력을 감소시키는 방법이다. BI는 현재의 버스상태를 고려하여 데이터를 그대로 전송하는 것 보다, 데이터를 반전한 상태 (Bit-wise invert)로 전송할 때 적은 수의 버스전이가 발생한다면, 그 데이터를 반전시켜 전송하는 방법이다. 데이터의 반전 여부를 알려주기 위해서 데이터와 함께 *inv*라는 상

* 정회원, 단국대학교(천안캠퍼스) 전자공학과
(Department of Electronics Engineering, Dankook University)

접수일자: 2007년4월2일, 수정완료일: 2007년5월14일

태신호를 전송하며, 보통 *inv*는 invert-line이라는 보조선을 기존의 버스에 추가하여 데이터와 함께 전송한다.

위와 같은 간단한 BI의 개념을 기본으로 하여 BI의 효율을 높이기 위한 여러 가지 알고리즘들이[3,4,5,6,7] 파생되었다. 넓은 버스폭을 갖는 버스에 대하여 하나의 버스를 여러 개의 sub-bus로 분할하여 각 sub-bus 마다 BI 알고리즘을 적용하는 Partitioned Bus-Invert Coding[2]이나, 데이터 어드레스버스와 같이 버스의 일부분만 변화하고 나머지는 항상 고정된 값을 갖는 경우에 적용하기위한 Partial Bus-Invert Coding[3]과 같이 전송되는 데이터의 특성을 고려하여 BI의 효율을 높이려는 여러 가지 방법이 개발되고 있다.

BI 및 BI에서 파생된 거의 모든 알고리즘들은 코딩정보를 전송하기위한 방법으로써 단·복수의 보조선을 사용하고 있다. 그러나 이러한 알고리즘들은 그동안 버스선의 전이회수를 최소화하는 데에만 초점을 맞추어 왔을 뿐 보조선자체에서 발생하는 전이횟수를 감소시키려는 노력이나 보조선에서 발생하는 전이가 BI의 효율 감소에 미치는 영향에 대해서는 간과해온 경향이 있다. 본 논문에서는 코딩정보를 전송하기위하여 사용해온 보조선의 단점 및 보조선에서 발생하는 전이가 BI의 효율을 상당히 감소시키는 점에 주의하여, 보조선을 사용하지 않고 버스선을 통하여 코딩정보를 전송하는 새로운 방법(NIL-BI, No Invert-Line Bus-Invert)을 개발하였다. 또한 데이터와 함께 코딩정보를 전송하면서도 추가적인 전이의 발생을 거의 완전히 억제함으로써, 결과적으로 보조선을 사용할 때와 비교하여 BI의 효율을 대폭 향상시키는 방법을 제시하였다.

Ⅱ장에서는 그동안 사용해왔던 보조선의 문제점을 살펴보았으며, Ⅲ장에서는 이 문제점들을 해결하기 위한 방안으로 보조선을 사용하지 않고 버스선을 통하여 코딩정보를 전송하는 방법을 제시하였다. 또한 제시된 회로의 성질을 이용하여 코딩정보 전송 시 버스선의 추가적인 전이를 억제하는 방법을 Ⅳ장에 설명하였다, Ⅴ장에서는 실험을 통하여 이 회로들의 효용성을 검증하였으며 Ⅵ장에서는 전반적인 결과들을 요약하였다.

Ⅱ. 보조선을 이용한 코딩정보 전송의 문제점

코딩정보의 전송을 위하여 BI 계통의 대부분의 알고리즘들에서 채택하고 있는 보조선 추가방법은 설계가 간편하고, 코딩정보를 데이터와 동시에 보내줌으로써 디코딩(Decoding)에 따른 지연을 최소화 할 수 있는 장점이 있으나 다음과 같은 몇 가지 단점을 수반한다.

첫째, 보조선의 추가로 버스폭이 증가하여 칩의 면적이 증가한다. 버스들은 보통 칩의 상하·좌우로 길게 뻗어 있으므로 버스폭이 증가하면 칩 전체에서 버스가 차지하는 면적이 증가한다. 오늘날의 집적회로는 전체 칩 면적의 50% 이상을 버스가 차지하고 있으므로 버스폭의 증가는 전체 칩 면적에 큰 영향을 준다.

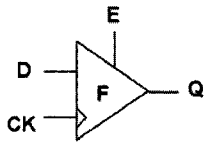
둘째, 외부 I/O 버스에 BI를 적용하는 경우 칩의 패키지에 영향을 주게 된다. 즉, 보조선(들)을 위한 I/O 핀을 마련해야 하므로 핀 수가 많아진다. 칩 설계 시 BI를 옵션으로 사용한다면 BI를 사용하는 칩과 사용하지 않는 칩 사이의 패키지호환성에 영향을 미치게 된다.

셋째, 코딩정보를 전송하기 위해 필연적으로 보조선이 활성화되어야 하므로 보조선에서 전이가 일어나 전력을 소모하게 된다. 일반적으로 보조선의 길이와 부하는 버스선의 그것과 거의 동일하므로 보조선의 전이는 버스선의 전이횟수에 포함시켜야 한다. 보조선의 전이는 코딩을 사용하지 않으면 발생하지 않는 것이므로, 코딩정보를 전송하기위해 추가되는 버스선 또는 보조선의 모든 전이는 코딩에 따른 오버헤드전이(Overhead Transition)가 된다. 물론 BI 코딩에 의해 버스선에서 발생하는 전이 횟수의 감소량이 오버헤드전이의 발생량보다 많으므로 전제적인 전이횟수는 감소한다. 그러나 버스선에서의 감소량을 오버헤드전이가 상쇄시킴으로써 결과적으로 BI의 효율을 떨어뜨리게 된다.

개념 및 설계상의 간편성 때문에 그동안 보조선은 대부분의 BI 류의 알고리즘들에서 관습적으로 사용되어 왔다. 본 논문에서는 위에 언급한 보조선의 문제점들에 유의하여 보조선을 사용하지 않고 BI를 설계하는 방법을 제시하였다.

Ⅲ. 버스선을 이용한 코딩정보 전송

보조선을 사용하지 않는다면 코딩정보를 버스를 통하여 전송하여야만 한다. 코딩정보, 즉 *inv* 비트를 데이터와 함께 버스선을 통하여 전송하기 위해서는, 한 전송주기에 하나의 버스선을 통하여 두 비트 이상의 데이터를 전송할 수 있는 방법이 필요하다 (앞으로 데이터의 전송주기는 한 클럭싸이클이라 가정하기로 한다). 본 논문에서는 이를 위하여 Flip-Driver(FD)라고 명명한 특별한 버스드라이버를 사용하였다. FD는 두 개의 입력단 D(Data), E(Enable)와 하나의 출력단으로 구성되어 있으며, D 와 E 의 조합에 따라 출력단에 4 가지



(a)

E	D	Q	
		$CK_{1/2}$	$CK_{2/2}$
L	L	L	
L	H	H	
H	L	H	L
H	H	L	H

(b)

그림 1. Flip-Driver.
(a) 논리 심볼 (b) 정 flip-driver의 진리표
Fig. 1. Flip-Driver.
(a) Conventional logic symbol
(b) Truth table of positive flip-driver

의 서로 다른 파형을 생성함으로써 한번에 2 비트의 정보 전송이 가능하다.

FD는 보통의 버스드라이버에 E 입력을 추가하여 E가 0 의 값을 가질 때 (Disable 상태)에는 일반적인 버스드라이버와 같이 작용하도록 하는 반면, E 가 1의 값을 가질 경우 (Enable 상태)에는 전송주기의 중간에서 출력에 반전 (Flip)이 일어나도록 설계한 로직회로이다. 전송주기의 중간에 출력이 반전되도록 하려면 FD의 출력 값은 반드시 전송주기의 전반부와 후반부에 서로 다른 값을 가져야만 하므로 FD는 전송주기의 전반, 또는 후반에 출력을 강제로 D 입력단의 반대값 ($\overline{D}$)을 갖도록 설계된다. FD는 enable 되면, 먼저 $\overline{D}$를 출력한 후 이후에 D를 출력하는 정-FD (Positive Flip-Driver, FD^+)와 먼저 D를 출력하고 중간에 $\overline{D}$로 반전되는 부-FD (Negative Flip-Driver, FD^-)의 두 가지 종류로 설계할 수 있으며, 동기식 또는 비동기식으로 동작할 수 있다. 본 논문에서는 동기식 데이터 전송을 가정하여 동기식 FD 만을 설명하기로 하며, 편의를 위하여 그림 1-(a)와 같은 심볼을 사용하여 나타내기로 한다. 한 클럭의 주기를 반으로 나누어 첫 번째 반클럭을 $CK_{1/2}$로 두 번째 반클럭을 $CK_{2/2}$ 로 나타내기로 하면 동기식 FD^+의 동작은 그림 1-(b)와 같다. 이 동기식 FD^+는 그림 2와 같이 몇 개의 논리회로를 조합하여 간단히 설계할 수 있다 (회로 중 D-FF는 입력신호의 안정성 또는 주변회로와의 관계를 고려하여 생략이 가능하다).

그림 3은 기존의 BI 회로[2]를 FD를 이용하여 수정·설계한 것이다. 보조선과 관련된 부분(점선부분)을 제거하고 하나의 버스선을 택하여 버스드라이버를 FD로 대치한 후, *inv*를 FD의 E 단자에 연결함으로써 *inv* 신호가 1 일 때 FD가 활성화 되어 전송주기의 중간에 버스선에 전이를 일으킨다. 수신단에서는 해당 버스선에 전이 감지회로를 이용하여 전송주기의 중간에서 버스선의

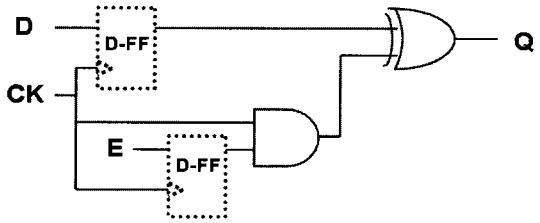


그림 2. 동기식 Flip-driver의 간단한 구현 예
Fig. 2. A sample implementation of synchronous flip-driver.

전이여부를 판단하여 *inv* 신호를 수신한다. 설계 시 FD^+ 또는 FD^- 중 어느 것이라도 사용이 가능하나, FD^+를 사용하면 버스의 최종 값이 보조선을 사용한 BI 설계회로의 값과 같아져 두 경우 똑같은 반전열(Invert sequence)을 얻게 되는 장점이 있으므로 본 논문에서는 FD^+를 기본으로 사용하였다. (이후로 모두 FD^+로 가정한다). 기존의 BI 설계와 달리 FD를 이용한 설계 (그림 3)에서는 데이터의 반전여부를 판단할 때 (Majority Voting) 오버헤드전이의 발생여부를 고려하지 않았다. 그 이유는 다음 장에 설명하는 선택회로에 의해 오버헤드전이의 발생이 억제되어 대부분의 경우 오버헤드전이가 거의 발생하지 않는다고 가정할 수 있기 때문에, 회로의 간단한 구현을 위하여 생략하였다.

Ⅳ. 오버헤드전이 발생의 억제

보조선을 이용한 일반적인 BI 설계 방식에서는 *inv* 비트를 전송하기위하여 보조선의 전이를 피할 수 없으므로 이에 따르는 전력소모도 고려하여야 한다. 보조선은 BI를 위해 필요한 것으로써 BI을 사용하지 않는다면 코딩정보를 전송할 필요가 없다. 따라서 코딩정보를 전달하기위하여 추가적으로 발생하는 모든 전이는 BI에 의해 생기는 오버헤드전이라 볼 수 있다.

전체 데이터열을 전송할 때 데이터를 반전시켜 전송하는 회수(즉 *inv*가 1 이 되는 횟수)를 I 라고 하고, 이들 중에 연속적으로 반전 (Successive Inverting, 연속반전)이 일어난 횟수를 S 라고 하면 보조선을 사용한 BI 설계에서 보조선에서 발생하는 오버헤드전이회수는

$$N_{OT} = 2I - 2S = 2I(1 - r_s) \quad (\therefore r_s = S/I) \qquad (1)$$

로 나타낼 수 있다. 예를 들어 데이터가 "DDDRRRRDRRRD" (D는 데이터 원형, R은 반전된 상태) 형태로 전송되었다고 하면 I=7, S=5 이므로 보조선에는 4번의 전이가 일어나게 된다. 일반적으로 r_s는 주어진 BI 알고리

즘에서는 데이터의 배열에 의해 결정되는 값이므로 보조선을 사용하는 경우 오버헤드전이를 줄일 수 있는 방법은 없다. 또한 역설적으로, 보조선을 사용할 경우에는 오버헤드전이가 많이 발생할수록 BI의 효율이 증가한다. 그 이유는, $H(A,B)$를 두 데이터 A, B간의 Hamming Distance라 할 때 $H(A,B)=H(\overline{A},\overline{B})$ 이므로 연속반전의 경우 버스선에 발생하는 전이 횟수, $H(\overline{A},\overline{B})$는 BI을 사용하지 않았을 때의 전이 횟수 $H(A,B)$와 동일하다. 그러므로 연속반전에서 뒤 데이터의 반전은 버스선의 전이횟수를 줄이는데 기여하지 못한다. 따라서 동일한 I 값에 대하여 r_s 가 커질수록, 오버헤드전이 수는 감소하지만 BI의 효율도 감소한다. 반면 r_s가 작을수록 오버헤드전이 수는 커지지만 버스선에서는 증가되는 오버헤드전이 보다 더 많은 수의 전이가 감소하게 되어 BI의 효율이 증가하게 된다. 이런 이유로 보조선을 사용한 BI에서는 오버헤드전이가 많은 것이 오히려 더 바람직하다.

한편, FD를 사용할 경우 오버헤드전이가 발생하는 경우와 그 수를 알아보면, 우선, inv=0 일 때 FD는 일반적인 버퍼와 동일하므로 오버헤드전이가 발생하지 않는다. inv=1일 때는 FD가 활성화되고 FD의 출력은 현재의 버스상태와 전송하려는 데이터 값에 따라 그림 4와 같은 파형을 갖는다. 전송하려는 데이터 값이 현재의 버스 값과 동일할 때(D=B), 보통의 버스드라이버를 사용할 때는 버스선에 전이가 발생하지 않지만, FD를 사용하면 버스선에 2번의 전이가 발생하게 되어 보통의 버스드라이버에 비해 2개의 오버헤드전이를 발생하게 된다. 한편, 데이터 값이 현재의 버스 값과 다를 때는 두 가지 드라이버 모두 버스선에는 한 번의 전이만 발생하게 되므로, 이 경우 오버헤드 전이는 발생하지 않는다. (FD는 단지 버스선의 전이시간을 1/2 클럭 지연시키는 작용을 한다). 이상과 같이 FD에 의해 오버헤드 전이가 발생하는 경우는 오직 inv가 1일 경우 전송하려는 데이터 값이 현재의 버스 값과 같을 때뿐이다.

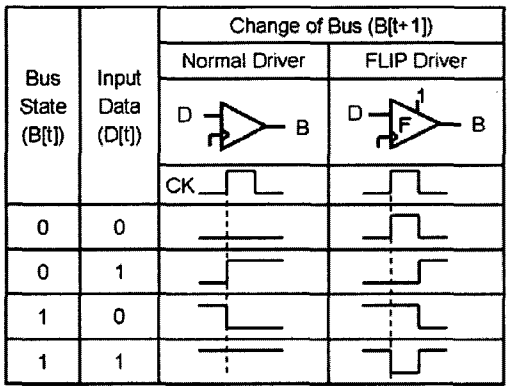


그림 4. 일반 버스드라이버와 Flip-Driver의 출력파형 비교

Fig. 4. Comparison of the output waveforms between normal bus-driver and Flip-driver.

FD를 사용하여 전체 데이터열을 전송할 때 데이터가 반전되는 회수를 I 라 하고, inv가 1일 때 버스선의 상태가 전송하려는 데이터와 다를($D\neq B$) 평균확률을 $P_{inv=1}(D\neq B)=p$ 라 하면 발생하는 전체 오버헤드전이 회수는 다음과 같다.

$$N_{OT}=I[p\cdot 0+(1-p)\cdot 2]=2I(1-p) \qquad (2)$$

식 (1)의 r_s와 같이 p는 데이터 배열에 의해 결정된다. 보조선을 사용한 경우와의 중요한 차이점은 r_s의 경우 BI의 효율과 직접적인 관계가 있는 변수지만 p의 경우는 그와 무관하다는 점이다. 그러므로 p를 증가시킬 수 있다면 버스선에서의 전이 횟수 감소와는 무관하게 오버헤드전이 만을 감소시킬 수 있다. 그림 3과 같이 오직 하나만의 버스선에 FD를 적용하여 항상 이 선을 통하여 inv 비트를 전송한다면 오버헤드전이를 피할 수 없다. 그러나 전송할 데이터는 1 비트인데 반해 이용할

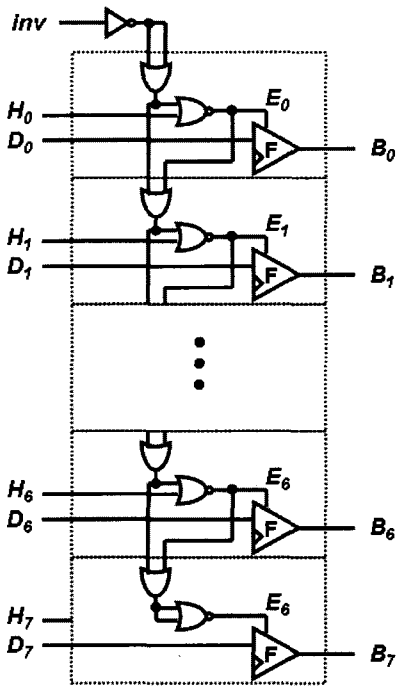


그림 5. 선택회로. B_i, H_i, D_i 는 그림 3의 해당 B, H, D에 연결된다

Fig. 5. An exmaple of the selection circuit. Each of B_i, H_i and D_i is connected to the corresponding B, H and D node in Fig. 3.

수 있는 버스선은 여러 개이므로, 다수의 버스선에 FD를 적용하고 *inv*=1 인 경우 FD를 적용한 버스선들 중에서 오버헤드전이가 발생하지 않는 조건을 만족하는 버스선을 찾아 전송한다면 p가 증가하여 오버헤드전이의 발생을 대폭 줄일 수 있다.

그림 5는 8-bit 버스에서 모든 버스선에 FD을 적용하여 8개의 버스선 중에서 $D \neq B$ 인 버스선을 찾는 "선택회로 (Selection circuit)"를 나타내었다. 선택회로를 그림 3의 BI 인코더(encoder)에 삽입하려면 그림 5의 점선으로 된 상자를 그림 3의 버스드라이버 대신 연결한다. (버스드라이버의 입력 D와 출력 B는 점선상자의 D와 B에 각 각 연결하고, 점선상자의 H 단자는 그림 3의 해당 H와 연결한다). 이 선택회로는 FD가 적용된 버스선들 중에서 $D_i \neq B_i$ (H_i=0)인 첫 번째 버스선을 찾는 작용을 한다. 위 조건을 만족하는 버스선이 존재 하면 그 선의 E_i 가 1로 되어 FD를 활성화 시킨다. 또한 이후의 모든 FD는 E_i에 의해 비활성화 (Disable) 된다. 위 조건을 만족하는 버스선이 존재하지 않는다면 마지막 버스선이 강제적으로 활성화 되어 *inv* 비트를 전송하게 되며, 2 개의 오버헤드전이가 발생하게 된다. 선택회로 사용 시 오버헤드전이가 발생하는 경우는 전송하려는 데이터가 현재의 버스선의 값과 동일할 때뿐이다. 일반적으로 상관관계가 없는 데이터 (Uncorrelated data)의 전송 시 이러한 확률은 매우 낮으므로 오버헤드전이는 거의 발생하지 않는다고 볼 수 있다.

V. 실 험

새로 제안된 방법이 보조선을 제거함과 동시에 버스선의 전이감소에 얼마나 효율적인지 검증하기 위하여 일련의 실험을 수행하였다. 실험을 위하여 그림, 동영상, 및 웹 소스파일들을 각 2 개씩 선정하였다. 인터넷 응용분야에서 사용되는 대표적인 데이터 형식들에 대하여 BI의 효율성을 알아보고자 이러한 파일들을 선택하

표 1. 실험 결과
Table 1. Result of the experiments.

				PJ	RP	BW	WM	RX	YT
Data Size (bytes)				602,912	535,784	7,208,088	5,544,024	99,016	53,368
8-bit Bus	Raw		N_R	2,372,576	2,104,362	28,220,122	20,520,634	275,629	154,222
	BI	Aux	$N_T(R)$	1,935,657(18.42)	1,733,594 (17.62)	23,288,988 (17.47)	16,974,471 (17.28)	264,097 (4.18)	146,857(4.78)
			$N_{OT}(R_{OT})$	215,409 (9.08)	185,860 (8.83)	2,459,682 (8.72)	1,789,957 (8.72)	8,574 (3.11)	5,989 (3.88)
		NIL	$N_T(R)$	1,730,878 (27.05)	1,552,278 (26.24)	20,892,812 (25.96)	15,616,100 (23.90)	257,771 (6.48)	141,578 (8.20)
			$N_{OTN}(r)$	10,630 (4.93)	4,544 (2.44)	63,506 (2.58)	431,586 (24.11)	2,248 (26.22)	710 (11.86)
	Partial BI	Aux	$N_T(R)$	-	-	-	-	246,809 (10.46)	129,914(15.76)
			$N_{OT}(R_{OT})$	-	-	-	-	20,230 (7.34)	14,363 (9.31)
		NIL	$N_T(R)$	-	-	-	-	227,677 (17.40)	106,796 (30.75)
			$N_{OTN}(r)$	-	-	-	-	1,098 (5.43)	5,468 (38.07)
16-bit Bus	Raw		N_R	2,362,337	2,109,947	28,177,947	20,726,872	275,370	155,981
	BI	Aux	$N_T(R)$	2,024,310 (14.31)	1,816,352 (13.91)	24,270,291 (13.87)	17,835,296 (13.95)	266,442 (3.24)	151,768 (2.70)
			$N_{OT}(R_{OT})$	117,273 (4.96)	101,729 (4.82)	1,341,348 (4.76)	1,002,410 (4.84)	4,264 (1.55)	2,225 (1.43)
		NIL	$N_T(R)$	1,915,623 (18.91)	1,714,635 (18.74)	22,929,313 (18.63)	17,012,470 (17.92)	263,504 (4.31)	149,777 (3.98)
			$N_{OTN}(r)$	8,586 (7.32)	12 (0.01)	370 (0.03)	179,584 (17.92)	1,326 (31.10)	234 (10.52)
	Partitioned BI	Aux	$N_T(R)$	1,938,087 (17.96)	1,737,792 (17.64)	23,142,369 (17.87)	17,039,155 (17.79)	259,286 (5.84)	147,129 (5.68)
			$N_{OT}(R_{OT})$	212,962 (9.01)	186,227 (8.83)	2,476,244 (8.79)	1,842,849 (8.89)	11,914 (4.33)	6,350 (4.07)
		NIL	$N_T(R)$	1,740,407 (26.33)	1,555,907 (26.26)	20,742,385 (26.39)	15,532,082 (25.06)	253,354 (8.00)	142,449 (8.68)
			$N_{OTN}(r)$	15,282 (7.18)	4,342 (2.33)	76,260 (3.08)	335,776 (18.22)	5,982 (50.21)	1,670 (26.30)
32-bit Bus	Raw		N_R	2,369,618	2,117,588	28,125,758	20,734,862	285,611	163,386
	BI	Aux	$N_T(R)$	2,103,213 (11.24)	1,891,676 (10.67)	25,315,311 (9.99)	18,538,310 (10.59)	280,773 (1.69)	160,174 (1.97)
			$N_{OT}(R_{OT})$	63,411 (2.68)	54,434 (2.57)	686,501 (2.44)	528,984 (2.55)	1,682 (0.59)	1,138 (0.70)
		NIL	$N_T(R)$	2,042,960 (13.79)	1,837,242 (13.24)	24,628,812 (12.43)	18,100,524 (12.70)	279,109 (2.28)	159,048 (2.66)
			$N_{OTN}(r)$	3,158 (4.98)	0 (0.00)	2 (0.00)	91,198 (17.24)	18 (1.07)	12 (1.05)
	Partitioned BI	Aux	$N_T(R)$	1,938,025 (18.21)	1,740,887 (17.79)	23,184,362 (17.57)	17,040,594 (17.82)	268,287 (6.07)	151,284 (7.41)
			$N_{OT}(R_{OT})$	214,645 (9.06)	188,615 (8.91)	2,461,094 (8.75)	1,845,010 (8.90)	11,672 (4.09)	7,652 (4.68)
		NIL	$N_T(R)$	1,737,540 (26.67)	1,556,604 (26.49)	20,790,512 (26.08)	15,528,454 (25.11)	259,637 (9.09)	144,714 (11.43)
			$N_{OTN}(r)$	14,160 (6.60)	4,332 (2.30)	67,244 (2.73)	332,870 (18.04)	3,022 (25.89)	1,082 (14.14)

였으며, 가능하면 영상처리분야에 사용되는 벤치마크 파일을 사용하였다. 그림파일로는 p9050408.jpg[8] (PJ)와 82_r340.png[9] (RP)를 사용하였으며, 동영상파일로는 WMV 형식의 영화 Beautiful Mind의 예고편 (BW)과 MPEG 형식의 Walk1.mpg[10] (WM)을 사용하였다. 웹 소스파일로는 XML 및 HTML 파일로서, 웹 페이지 http://www.w3.org/TR/REC-xml/의 XML 버전인 REC-xml-20060816.xml (RX)과 Youtube 홈페이지의 소스파일 (YT)을 사용하였다.

실험은 3가지의 BI 알고리즘 (BI, Partitioned BI, 및 Partial BI)에 대하여 보조선을 사용하여 구현한 경우와 FD 및 선택회로를 적용한 새로운 NIL(No Invert-Line) 방식으로 구현한 경우에 대하여 각각 전체 전이회수 및 오버헤드전이 횟수를 측정하여 서로 비교하는 방법으로 진행하였다. Partitioned BI 알고리즘에 NIL 방식을 적용한 효과를 알아보기 위해서는, 16-bit 버스 및 32-bit 버스를 8-bit 씩의 서브버스들로 나누어 각 서브버스마다 BI를 적용하였다. Partial BI 알고리즘에 NIL 방식을 적용한 효과를 측정하기 위해서는, HTML 파일 및 XML 파일에 대하여 먼저 버스전이를 가장 적게 발생하는 부분조합을 찾아낸 후 이를 적용하여 실험을 수행하였다. (두 파일 모두 8-bit 버스 중 상위 3개 버스를 배제하고 5개의 하위(LSB) 버스선만 BI를 적용하였을 때 가장 적은 버스전이 횟수를 얻을 수 있었다.)

표 1에 위 파일들에 대한 실험결과를 나타내었다. 8-bit, 16-bit, 32-bit 버스에 대하여, 먼저 파일들을 BI를 사용하지 않고 그냥 전송(Raw Transmission)할 때 버스선에서 발생하는 전이수(N_R)를 측정한 후, BI 및 Partial BI, Partitioned BI을 사용하여 동 파일들을 전송할 때 발생하는 전체 전이수 (N_T) 및 오버헤드전이수 (N_{OT})를 측정하여 각 알고리즘이 버스선의 전이회수를 감소시키는 효율(R, 전이감소율) 및 오버헤드전이를 발생시키는 비율(R_{OT}, 오버헤드전이 발생율)을 구하였다. 또한, 동일한 알고리즘에 대하여 보조선을 사용할 때와 NIL 방식을 사용하였을 때의 오버헤드전이수의 차이를 알아보기 위해, NIL방식의 오버헤드전이에 대해서는 R_{OT} 대신, 보조선사용 시 발생하는 오버헤드전이와의 상대적 오버헤드전이발생비율(r)을 표 1에 나타내었다.

$$\text{전이감소율}(R) = (1 - \frac{N_T}{N_R}) \times 100 \ (\%) \qquad (3)$$

$$\text{오버헤드전이 발생율}(R_{OT}) = \frac{N_{OT}}{N_R} \times 100(\%) \qquad (4)$$

$$r = \frac{N_{OT,N}}{N_{OT}} \times 100 \ (\%) \qquad (5)$$

표 1로 부터 8-bit 버스에 NIL 방식를 적용한 경우 보조선을 사용했을 때와 비교하여 BI의 효율(전이감소율, R)이 38%~86% 증가함을 볼 수 있다. 그림 및 동영상 파일의 경우 효율이 약 50% 증가하였으며, 효율의 증가가 상대적으로 적은 WM의 경우 상대적 오버헤드 전이발생율(r)이 24%로서 오버헤드전이가 효과적으로 제거되지 못하였음을 알 수 있다. 이것은 실험에 사용된 WM 동영상파일이 변화가 별로 없는, 매우 정적인 동영상이어서 상대적으로 동일한 데이터 값이 다른 파일에 비해 많았기 때문이다. WM을 제외한 다른 파일들은 NIL 방식의 선택회로에 의해 90%이상의 오버헤드전이가 제거되었음을 볼 수 있다. NIL 방식의 적용에 의한 효율증가는 순전히 오버헤드전이의 감소에 의한 현상으로써, 보조선에서 발생하는 전이가 알고리즘의 효율을 대폭 감소시키고 있음을 보여준다. XLM이나 HTML과 같은 텍스트파일의 경우 BI의 효율이 80% 이상 증가하였다. 텍스트파일의 경우에는 기본적으로 낮은 BI 효율에 비해 오버헤드전이의 상대적 비율이 높기 때문에, 오버헤드전이의 감소에 의한 BI 효율의 증가가 다른 파일들에 비해 크게 나타났다.

그림파일 및 동영상파일의 경우 16-bit 버스에 NIL 방식을 적용하였을 경우 BI의 효율이 29%~35% 증가하였으며, 32-bit 버스에서는 20%~24% 증가하였다. Partitioned BI를 사용하면 버스의 전이감소율이 8-bit 버스의 효율에 근접하고 NIL 방식의 적용에 의한 효과도 역시 그것에 근접하는 결과를 얻었다. 일반적으로 BI의 효율은 버스폭이 넓을수록 감소한다. BI의 효율이 낮으면 반전횟수도 적어지고 따라서 오버헤드전이도 적게 발생하므로, NIL 방식을 적용하였을 때 오버헤드전이를 줄여서 얻게 되는 효과도 감소하게 된다. 즉 버스폭이 넓어질수록 BI의 효율도 감소하고 NIL방식을 적용한 효과도 감소하는 경향을 보이고 있다.

일반적으로 BI는 텍스트파일에 대해 효율이 좋지 않다. 그 이유는 텍스트 파일은 ASCII 코드의 일부분만을 사용하므로 버스선의 일부가 고정되어있기 때문이다. 따라서 Partial BI를 적용하여 버스의 하위 5개 선에 만 BI을 적용하였을 경우 그 효율이 대폭 증가한다. 또한 Partial BI에 NIL 방식을 적용함으로써 대상파일에 대하여 Partial BI의 효율이 67% 및 95% 향상되었음을 볼 수 있다. 이렇게 상대적으로 큰 효율의 증가는 단순

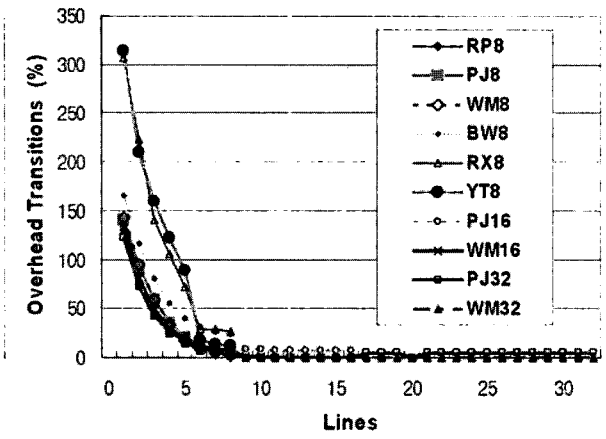


그림 6. 선택회로에 사용된 버스선의 갯수에 따른 오버헤드전이 발생 횟수의 변화 (보조선을 사용하였을 때 발생하는 오버헤드전이 수에 대한 백분율)

Fig. 6. The variation of the number of overhead transitions vs. the number of bus-lines used in the selection circuit (the percentage to the number of OTs when using the auxiliary lines).

히 오버헤드전이의 감소뿐 아니라 버스선의 전이가 더 많이 감소된 데에 기인한다. 그 이유는 Partial BI을 적용하는 버스선의 수와 관련이 있다. 일반적으로 버스폭은 8-bit, 16-bit 등 짝수 이다. 이 경우 보조선을 포함하면 선의 개수가 홀수가 되어 반전을 하기위한 조건이 명확해진다. 반면 버스선이 홀수 개이고 보조선을 포함하여 짝수 개가 되면 반전을 위한 조건이 애매한 경우가 생기게 된다. 예를 들어 Partial BI을 적용하는 버스선의 수가 5개 일 때, 원 데이터를 전송하면 버스선에 3개의 전이가 일어난다고 가정하자, 한편 데이터를 반전시켜 전송할 때 보조선에 전이가 발생한다면 이때 역시 3개의 전이가 발생하기 때문에 반전을 하나 마나 전이 횟수는 감소하지 않는다. 이 때 반전여부는 설계자의 선택문제이며, 어느 경우에도 알고리즘의 효율에 영향을 미치지 않는다. 그러나 NIL 방식에서는 이러한 경우 오버헤드전이는 선택회로에서 제거한다고 가정하고 무조건 반전함으로써 일단 버스선의 전이 수를 감소시킨다. 확률적으로 오버헤드전이가 선택회로에서 제거되는 경우가 대부분이므로, 결과적으로 오버헤드전이뿐 아니라 버스선에서도 더 많은 전이를 감소시켜 알고리즘의 효율이 더욱 증가하게 된다. 이러한 복합적이 이유로써 Partial BI 실험에서와 같이 홀수 개의 버스선에 NIL 방식이 적용되는 경우에 효율이 더욱 크게 향상된다. 또한, 표 1에서 Partial BI에 NIL 방식을 적용하였을 때 상대적 오버헤드전이발생율(r)이 큰 것은 실제 오버헤드전이가 적게 제거된 것이 아니라, 오버헤드전이를 일으키는 경우가 증가하였기 때문이다 (즉 제거하려는 모집단의 크기가 대폭 증가하였다).

선택회로에 사용되는 버스선의 수가 많을수록 오버헤드전이 수는 감소하지만 회로의 복잡도 및 회로구현을 위한 칩의 면적은 증가한다. 오버헤드전이의 발생을 효과적으로 억제하면서 회로에 대한 부담을 줄이기 위해서는 적당한 수의 버스선을 사용해야한다. 그림 6은 8-bit, 16-bit, 32-bit 버스에서 선택회로에 사용되는 버스선의 수를 변화시키면서 상대적 오버헤드전이발생율(r)을 측정한 것이다 (심볼 뒤의 숫자는 버스폭을 나타낸다). 선택회로를 사용하지 않고 FD 만을 사용하였을 때는 보조선을 사용하였을 때에 비해 오버헤드전이가 더 많이 발생하였다. 따라서 버스의 소비전력을 줄이기 위해서는 선택회로의 사용은 필수적이며 8-bit 버스에서 버스 전체에 선택회로를 사용하였을 때 90% 이상의 오버헤드전이를 제거하였다. 16-bit 버스 등 버스폭이 넓은 버스에 대해서도 상관관계(Correlation)가 없는 데이터열에 대해서는 버스 전체에 선택회로를 적용할 필요 없이 8 개 버스선만 적용해도 오버헤드전이를 90% 이상 제거할 수 있었다.

VI. 결 론

본 논문에서는 BI 알고리즘 및 BI에서 파생된 여러 알고리즘에 적용될 수 있는 NIL-BIC 라는 새로운 회로 설계 방식을 제시하였다. NIL 방식은 근본적으로, BI계의 알고리즘에서 습관적으로 사용되어온 보조선을 제거하고 버스선을 통하여 *inv* 신호를 전송하는 방식이다. 이를 위하여 Flip-Driver라는 새로운 버스드라이버를 사용하였으며 FD의 전송특성을 이용하여 오버헤드전이를 발생하지 않고도 *inv* 신호를 전송할 있는 선택회로를 개발하였다. 선택회로와 결합한 FD는 BI 및 Partitioned BI, Partial BI 알고리즘에 대하여 오버헤드전이를 효과적으로 제거함으로써, 보조선을 사용하였을 때에 비하여 알고리즘의 효율이 크게 개선시킬 수 있음을 실험으로서 검증하였다. NIL-BIC 방식은 새로운 알고리즘이 아니라 회로의 설계방법에 의한 알고리즘의 효율증대 방법이므로 BI에서 파생된 거의 모든 알고리즘에 적용할 수 있는 장점이 있다.

참 고 문 헌

[1] R. Wilson, "Low power and paradox," Electronic Engineering Times, pp. 38. Nov. 1, 1993.
[2] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power I/O," IEEE Trans. VLSI Syst., vol. 3, pp. 4958, Mar. 1995.
[3] Y. Shin, S. I. Chae, and K. Choi, "Partial bus-invert coding for power optimization of application-specific systems," IEEE Trans. VLSI Syst., vol. 9, pp. 377-383, Apr. 2001.
[4] R. B. Lin and C. M. Tsai, "Weight-based bus-invert coding for low-power applications" , in Proc. ASP-DAC/VLSI Design, pp. 121-125. Jan. 2002.
[5] J. Natesan and D. Radhakrishnan, "Shift invert coding (SINV) for low power VLSI", 2004. Euromicro Symp. on Digital system Design, pp. 190-194, Sept. 2004.
[6] Y. Zhang, J. Lach, K. Skadron, and M. R. Stan, "Odd/Even bus invert with two-phase transfer for buses with coupling" , Proc. 2002 Int. Symp. on Low Power Electronics and Design, pp. 80-83, 2002
[7] U. Narayanan, K. S. Chung, and T. Kim, "Enhanced bus invert encodings for low-power" , IEEE Int. Symp. on Circuits and Systems, Scottsdale, Vol. 5, pp. 25-28, May 2002.
[8] MIT-CSAIL Database of Objects and Scenes, http://web.mit.edu/torralba/www/database.html [Online]
[9] Amsterdam Library of Object Images, http://staff.science.uva.nl/~aloi/ [Online]
[10] CAVIAR Test Case Scenarios, http://homepages.inf.ed.ac.uk/rbf/CAVIAR/ [Online]

저 자 소 개

윤 명 철(정회원)
1986년 서울대학교 전자공학과 학사
1988년 서울대학교 전자공학과 석사
1998년 The Univ. of Texas at Austin. ECE, Ph.D.
1988년~2002년 현대전자 (현 하이닉스)
2005년 대구경북 과학기술연구원(DGIST) 책임연구원
2006년~현재 단국대학교(천안) 전자공학과.
<주관심분야: VLSI/SoC 설계, Embedded System, Digital System 설계>