논문 2009-46SD-11-4

보조선을 사용하지 않은 Sequence Switch Coding 회로의 설계

(Design of A Sequence Switch Coding Circuit Without Using Auxiliary Lines)

윤 명 철*

(Myungchul Yoon)

요 약

코딩정보의 전송을 위하여 보조선에서 발생하는 오버헤드전이는 Sequence Switching Code (SSC) 알고리즘의 확장성을 제한하는 가장 큰 걸림돌이 되어왔다. 본 논문에서는 보조선을 사용하지 않고 SSC 회로를 구현하는 방법과 함께 이 방법을 사용하여 보조선을 사용하였을 때 보다 오버헤드전이를 적게 발생시키는 방법을 제시하였다. 실험결과 새로운 방법을 적용함으로써 보조선을 사용하는 방식에 비하여 오버헤드전이의 발생을 50% 이하로 줄이고 알고리즘의 효율을 약 30% 향상시킬 수 있었다.

Abstract

The transition of auxiliary lines for transmitting coding information has been one of the major obstacles to restricting the scalability of Sequence Switch Coding (SSC) algorithms. A new design of SSC which does not use auxiliary lines is presented in this paper. The new design makes overhead transitions far less than the previous designs that use auxiliary lines. By applying the new technique, more than 50% of overhead transitions have been reduced, leading to the increase of 30% of the overall efficiency of SSC algorithm.

Keywords : Sequence-Switch Coding, Flip-Driver, Low-Power Bus, Transition-Reduction Coding, Low-Power Design

I. 서 론

반도체의 집적도가 증가할수록 저전력 설계기술의 중요도가 높아진다. 저전력 VLSI 칩을 설계하기 위해서는 단위모듈의 저전력 설계도 중요하지만, 그에 못지않게 버스(Bus)의 저전력 설계가 매우 중요하다. 그 이유는 반도체 기술의 발전으로 피쳐 크기(Feature size)가 작아져 모듈의 크기와 소비전력은 작아지지만, 집적도가 높아짐에 따라 버스에 연결된 모듈 수가 증가하여 버스의 부하는 오히려 증가하기 때문이다. 따라서 집적도가 높고 칩의 크기가 클수록 저전력 버스를 설계하는 것이 매우 중요한 요소가 된다.

버스의 소비전력을 줄이기 위한 방법은 크게 버스선들(Bus lines)의 동작 소비전력을 줄이는 방법과 버스선들의 구동 횟수를 줄이는 방법으로 나눌 수 있다. 전자는 칩구조의 변경 또는 모듈의 재배치 등을 통하여 버스의 부하를 감소시키거나, Reduced-Swing Bus[1~2]와 같은 특수한 저전력 버스회로를 사용하여 버스의 소비전력을 줄이는 방법이다. 반면, 후자는 코딩 알고리즘을 통하여 버스의 활성화를 억제함으로써 버스의 소비전력을 줄이는 방법이다.

* 정회원, 단국대학교(천안) 전자공학과
(Department of Electronics Engineering, Dankook University)
※ 이 연구는 2008학년도 단국대학교 대학연구비 지원으로 연구되었음.
접수일자: 2009년9월8일, 수정완료일: 2009년10월18일

버스의 소비전력을 줄이는데 있어 코딩방식은 버스 구조나 회로의 개발만으로는 개선할 수 없는 효과를 얻을 수 있으므로 그동안 많은 연구가 수행되어 왔다. 코딩방식에는 전송하려는 데이터의 패턴이나 분포 등을 사전에 알 수 있는 경우에 사용하는 특수대상의 코딩방식 (Special purpose coding scheme)과 데이터에 대한 아무런 사전정보가 필요 없이 일반적으로 사용할 수 있는 일반목적의 (General purpose) 범용 코딩방식으로 나눌 수 있다. 전자의 방식에는 Gray Coding,[3] T0 Coding,[4] Beach Solution[5] 등이 있으며, 범용방식에 비해 버스의 전이 (Transitions)를 감소시키는 효율은 높지만 전송데이터의 특성을 사전에 알아야 하므로 적용대상이 제한되는 단점이 있다. 후자에는 Bus-Invert (BI) Coding[6]과 Sequence Switch Coding (SSC)[7~8]이 있으며, 전송데이터에 대한 사전정보가 필요 없이 일반적인 데이터 전송에 사용할 수 있는 장점이 있다.

이 중 SSC는 다수의 데이터를 전송할 때 버스선들의 전이 횟수가 최소가 되도록 데이터의 순서를 바꾸어 전송함으로써 버스의 소비전력을 감소시키는 방법으로써, 인터넷 또는 멀티미디어용 칩과 같이 다수의 데이터를 연속적으로 전송하는 경우에 적용할 수 있는 저전력 전송방법이다. 다수의 데이터를 전송할 때 버스에서 발생하는 전이 횟수가 최소가 되는 순서를 찾는 문제는 잘 알려진 TSP (Traveling Salesman Problem)[10] 문제와 동일하다. 그러므로 SSC는 기존에 개발된 수많은 TSP 알고리즘들을 참조하여 최적의 알고리즘을 개발할 수 있을 것이다. 그러나 이러한 잠재적인 가능성에도 불구하고 그동안 SSC는 몇 가지 문제점에 의해 그 확장성이 제한되어 왔다. 이 중 가장 큰 문제점들의 하나는 코딩정보를 전송하기 위해 발생하는 소비전력이 버스선에서의 소비전력 감소분을 상쇄시켜 전체효율을 감소시키는 문제이다.

기존의 SSC 알고리즘들은 버스선에 보조선을 추가하여 이를 통하여 코딩정보를 전송하였다. 코딩정보를 전송하기 위해서는 보조선이 활성화되어야 하고, 이 보조선에서 발생하는 전이는 원래의 버스선에서는 존재하지 않던 추가적인 전이를 발생하게 된다. 일반적으로 보조선의 길이와 부하는 버스선의 그것과 거의 동일하므로 보조선의 전이는 버스선의 전이횟수에 포함시켜야 한다. 이와 같이 코딩정보를 전송하기위해 추가되는 모든 전이는 코딩에 따른 오버헤드전이(OT: Overhead Transition)가 된다. 물론 버스선에서의 전이횟수 감소량이 OT의 발생량보다 크므로 전체적인 전이횟수는 감소한다. 그러나 버스선에서의 감소량을 OT가 상쇄시킴으로써 결과적으로 SSC 알고리즘의 효율을 떨어뜨리게 된다. 이러한 문제점을 해결하기 위하여 본 논문에서는 기존의 SSC 알고리즘을 확장한 Extended Lagger Algorithm을 제시하고, 알고리즘의 효율성을 높이기 위하여 오버헤드전이의 발생을 감소시킬 수 있는 새로운 구현방법을 개발하였다.

Ⅱ장에서는 Lagger Algorithm을 확장한 Extended Lagger (EL) 알고리즘을 제시하고, 코딩정보를 전송하는 방법의 중요성을 살펴보았으며, 기존의 보조선을 사용하는 방법의 문제점에 유의하여, EL 알고리즘을 보조선을 사용하지 않으면서 오버헤드전이를 적게 발생시키는 새로운 전송방법을 기술하였다. Ⅲ장에서는 EL 알고리즘을 회로구현을 설명하였다. Ⅳ장에서는 새로운 방법의 검증을 위한 실험과정과 그 결과 및 분석내용을 기술하였으며, Ⅴ장에서는 전체적인 결론을 요약하였다.

Ⅱ. 확장된 Sequence Switch Coding 알고리즘 및 오버헤드전이를 감소시키는 방법

1. Extended Lagger Algorithm

현재까지 발표된 SSC 알고리즘들에는 Grouping Algorithm과 Lagger Algorithm이 있으며, 효율성과 하드웨어 구현의 용이성 면에서 Lagger Algorithm이 Grouping Algorithm에 비해 우수하다고 판명되었다.[8] Lagger Algorithm[7]은 매 전송 시기마다 두 개의 데이터 즉, 전송이 지연 (lagging) 되어 대기하고 있는 하나의 데이터 (lagger)와 하나의 새로운 데이터 중에 버스선의 전이를 더 적게 발생시키는 데이터를 선택하여 전송하는 방법이다. 이렇게 매 전송주기마다 2 개의 데이터만을 비교하였을 때에도 전체 버스전이회수의 약 10%를 줄일 수 있었다.[8] 만일 더 많은 수의 데이터 중에서 버스선의 전이를 가장 적게 발생시키는 데이터를 선택한다면 버스선의 전이를 감소시키는 효과는 더욱 증가할 것이다. 본 논문에서는 이와 같이 기존의 Lagger Algorithm을 확장하여 m (≥ 2) 개의 데이터 중에서 하나의 데이터를 선택하여 전송하는 Extended Lagger (EL) 알고리즘과 그 설계방법을 개발 하였다.

그림 1.에 EL 알고리즘을 Pseudo-code를 사용하여 표현하였다. EL은 일정한 크기의 데이터풀 (Data Pool)을 만들고, 데이터 전송 시마다 풀 내의 데이터 중에서

```
EXTENDED_LAGGER ( stream in, bus B)  {
  struct  Data_POOL  Pool :  //  data registers
  Data    D ;        //  A new data from the input stream
  int     ix ;       //  index (register #) of selected data
  int     np:        //  the number of valid data in the Pool

  np=0 ;
  for(i=0 ; i < POOL_SIZE -1 ; i++) {  //  fill the pool
     D = GetNewData (in) ;
     if (D == NULL )  invalidate (Pool(i)) ;
       else {
         Pool(i) = D ;
         np= np+1 ; }
     }
  ix = np ;
  while (np>0)  {  // begin date transmission
     D = GetNewWord (in)
     if( D != NULL)  Pool(ix) = D ;
       else {   // steps for the end of data stream
         invalidate (Pool(ix) ) ;
         np = np-1 ; }
     ix = Select(Pool, B) ;
     Send (Pool(ix), ix) ;        // send the chosen data
     }                            // and its index
}
```

그림 1. Extended Lagger 알고리즘의 Pseudo 코드
Fig. 1. Pseudo-code for the Extended Lagger Algorithm.

버스에 전이를 가장 적게 발생시키는 데이터를 찾아내어 전송한다. 데이터풀의 크기가 m인 EL을 EL-m으로 표시하기로 한다. Lagger Algorithm은 풀의 크기가 2인 경우에 해당하고, 이 표기법에 따르면 EL-2이 된다. EL-m의 처음 m-1 사이클 동안은 데이터 전송이 이루어지지 않으며, 입력 데이터는 풀을 채우는데 사용된다. m 사이클 이후에는 매 사이클마다 Select(Pool, B) 함수에 의해 풀 안에서 버스에 가장 적은 전이를 야기하는 데이터가 선택되어 전송된다. 전송된 데이터의 빈자리는 새로운 데이터로 채워짐으로써 풀의 크기는 계속 m으로 유지된다. 수신단에서의 순서복원을 위해 선택된 데이터의 레지스터 번호, ix가 데이터 (Pool(ix))와 같이 전달된다. 디코딩을 위해 필요한 정보는 디코딩 방법에 따라 달라질 수 있으나, 본 알고리즘에서는 단순히 선택된 데이터의 레지스터 번호를 전송한다.

EL은 새로운 알고리즘은 아니며 이와 같은 개념의 m-way Lagger Algorithm[8]이 이전에 제시되었다. EL-m은 m-way Lagger Algorithm과 개념적으로 같으며, 단지 코딩정보를 전송하는 방식에 차이가 있다. m-way Lagger Algorithm은 보조선을 사용하여 코딩정보를 전송하는 반면, EL-m은 보조선을 사용하지 않고 3절과 4절에서 설명하는 새로운 방식을 사용하여 코딩정보를 전송한다. SSC 알고리즘에서는 코딩정보를 전송하는 방법이 매우 중요하며 SSC의 효율에 큰 영향을 미친다.

2. 코딩정보 전송방법의 중요성

코딩을 사용하여 데이터를 전송하는 경우 버스선의 전이는 크게 데이터를 전송하기 위해 필요한 전이성분과 코딩정보를 전송하기 위해 필요한 전이성분으로 나눌 수 있다. 현재의 버스 값이 B일 때, 데이터 D와 코딩정보 ix를 전송할 때 발생하는 버스선의 전이횟수는 다음과 같이 표시할 수 있다.

$$T(B,D) = T_D(B,D) + T_{OT}(B,ix) \tag{1}$$

여기서 $T_D(B,D)$는 데이터 전송을 위한 버스의 전이횟수이며, $T_{OT}(B,ix)$는 코딩정보 전송을 위해 추가적으로 발생하는 오버헤드전이 (Overhead Transitions)의 수이다. 전체 데이터열을 전송할 때 발생하는 버스선의 전이횟수의 총합 T_t 는 각 각의 데이터 전송 시 발생하는 전이들을 모두 더하여 얻을 수 있다.

$$\begin{aligned} T_t &= \sum_k T(B_k, D_k) \\ &= \sum_k T_D(B_k, D_k) + \sum_k T_{OT}(B_k, ix_k) \\ &= T_{D,t} + T_{OT,t} \end{aligned} \tag{2}$$

윗 식에서 $T_{D,t}$ 는 데이터 전송을 위한 전이수의 총합이며, $T_{OT,t}$ 는 코딩정보 전송을 위한 OT의 총합을 나타낸다. 코딩을 사용하지 않으면 $T_{OT,t}$ 는 0 이 되고 $T_{D,t}$ 는 코딩을 사용하지 않고 전송할 때의 전체 버스전이수 (T_R 로 표시하기로 한다) 가 된다. SSC 알고리즘의 효율, e 는 코딩을 사용하였을 때 발생하는 전체 전이수 T_t 가 T_R 에 비하여 감소한 정도로 나타내며 다음의 식으로 표시할 수 있다.

$$\begin{aligned} \text{효율}(e) &= \frac{(T_R - T_t)}{T_R} = \left(1 - \frac{T_{D,t}}{T_R}\right) - \frac{T_{OT,t}}{T_R} \\ &= (1 - R_D) - R_{OT} \\ &= e_D - R_{OT} \end{aligned} \tag{3}$$

첫째 항 (e_D)은 데이터 전송만을 고려하였을 때의 효율이며 둘째 항 (R_{OT})은 코딩정보를 전송하기 위해 발생하는 OT의 발생률이다. 식 (3)에서 보는 바와 같이

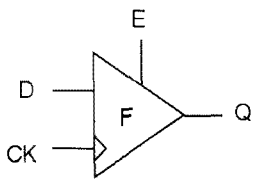
SSC 알고리즘의 효율은 데이터 전송만을 고려하였을 때의 효율에서 OT의 발생비율 만큼 감소하게 된다.

EL-m 알고리즘에서 데이터풀의 크기를 증가시키면, e_D가 증가할 것이라는 것은 쉽게 예상할 수 있다. 그러나 풀이 커질수록 R_{OT}도 증가한다. 풀의 크기가 크지 않을 때에는 e_D의 증가율이 R_{OT}의 증가율 보다 크므로 전체효율은 풀의 크기에 따라 증가하지만, 풀이 점점 커질수록 e_D의 증가율은 감소하고 R_{OT}는 계속 증가하여 어느 지점부터는 전체효율이 포화상태에 도달하거나 오히려 감소하게 된다. R_{OT}의 증가율은 코딩정보를 전송하는 방식에 따라 다르며, 풀의 크기에 따른 R_{OT}의 증가율이 클수록 효율이 감소할 뿐 아니라 포화에 도달하는 풀의 크기도 작아진다. SSC 알고리즘은 일차적으로 e_D를 결정하고 코딩정보의 전송방법은 R_{OT}를 결정하므로, SSC 알고리즘의 개발에는 코딩정보의 전송방법을 함께 고려해야 하며, 이것은 전체 효율을 향상시키는데 매우 중요한 요소가 된다.

3. 버스선을 이용한 코딩정보 전송방법

기존의 SSC 알고리즘들은 버스에 보조선을 추가하여 이를 통하여 코딩정보를 전송하였다.[7~8] 보조선의 사용은 결과적으로 버스폭을 증가시켜 칩의 면적을 증가시키는 한편, 보조선 수에 비례하여 OT의 발생도 많아져서 알고리즘의 효율을 크게 악화시키는 결과를 야기한다. 이러한 단점을 극복하기 위하여 본 논문에서는 보조선을 사용하지 않고 코딩정보를 전송하는 새로운 방법을 개발하였다. 이 방법은 보조선을 사용하는 것에 비해 OT를 적게 발생시켜 결과적으로 SSC 알고리즘의 효율이 증가되는 효과를 얻을 수 있다.

새로운 방법은 보조선을 사용하지 않으므로 버스선을 통



(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 2. Flip-Driver (a) 논리심볼 (b) 진리표
Fig. 2. Flip-Driver.
(a) Conventional logic symbol (b) Truth table

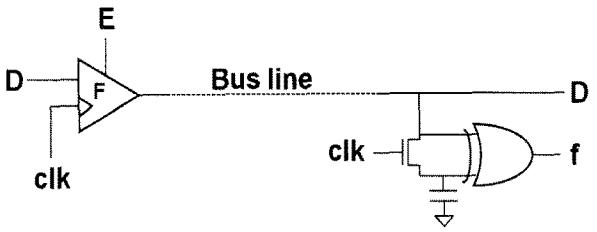


그림 3 Flip-driver를 사용한 버스선의 구조
Fig. 3. The structure of the FD-busline.

하여 데이터와 동시에 코딩정보를 전달하기 위해서는, 한 클럭에 2-비트의 정보를 전송할 수 있는 방법이 필요하며 이를 위하여 Flip-Driver[9] (FD)라는 특별한 버스드라이버를 사용하였다. 동기식 Flip-Driver의 심볼과 동작은 그림 2와 같다.[9] Enable 시의 FD의 동작을 알기 쉽게 설명하기 위해서 한 클럭의 주기를 반으로 나누어 앞의 반클럭을 $CK_{1/2}$로 뒤의 반클럭을 $CK_{2/2}$로 표시하기로 한다. FD는 Enable 되면 $CK_{1/2}$ 에서는 먼저 $\overline{D}$를 출력하고 반 사이클이 지난 $CK_{2/2}$ 에서 D를 출력한다. 이런 동작에 의해 Enable 시에는 클럭 중간에 출력의 반전 (Flip)이 발생하게 된다. 이와 같이 FD는 버스 드라이버에 플립 (Flipping) 기능을 활성화시키는 입력, E (Enable)를 추가하여 한 클럭에 2-비트의 값을 전송할 수 있도록 설계된 특별한 로직회로이다.

FD는 Disable (E=0) 상태에서는 일반적인 버스드라이버와 같이 동작하는 반면, Enable (E=1) 상태에서는 전송주기의 중간에서 출력에 반전 (Flip)이 일어나도록 설계되었다. FD는 두 개의 입력 D와 E 에 따라 4 가지의 다른 파형이 가능하므로 한 사이클에 2-비트의 데이터를 전송할 수 있다. 1-비트의 정보는 D의 값을 통하여 전송되고 다른 1-비트의 정보는 전송주기 (클럭)의 중간에 D의 반전(Flip) 여부를 통하여 전송된다.

그림 3에 FD를 사용한 버스선 (FD선)의 구조를 도시하였다. 기존의 버스드라이버를 FD로 대치하여 입력에 클럭과 Enable을 첨가하였다. 수신단에서는 두 개의 출력 (버스선의 최종 값, D와 플립신호, f) 값을 얻을 수 있다. 이와 같은 구조를 사용하여 1 개의 FD 선으로 1-비트의 데이터와 1-비트의 코딩정보를 동시에 전송할 수 있으므로 하나의 버스선과 하나의 보조선을 대치할 수 있다.

4. 오버헤드전이 발생을 감소시키는 방법

EL-m 에서, 데이터풀 안의 레지스터 번호 ix는 m 개의 서로 다른 번호를 가질 수 있으므로, 보조선을 사

용할 경우, ix 값을 전송하기 위해 필요한 보조선의 수는 다음과 같다.[8]

$$k_a = \lceil \log_2 m \rceil \quad (4)$$

만일 각 보조선의 활성화가 완전 불규칙적 (random) 이라면 매 전송주기마다 평균 1/2 개의 전이가 각 보조선에 발생하므로 매 전송 시마다 평균 $k_a/2$ 개의 OT가 발생하게 된다.

FD선을 사용할 때 발생하는 OT의 수는 버스와 데이터의 상대적인 값에 따라 달라진다. 전송전의 버스값과 전송데이터의 값이 다를 경우, FD선을 사용하여 코딩정보를 동시에 전달함으로써 버스선에 추가되는 전이(OT)는 없다. 그러나 버스값과 전송데이터의 값이 동일할 경우에는 FD선의 출력파형은 일반 버스드라이버로 데이터만 전달할 때에 비해 2 개의 전이가 추가적으로 발생한다. 만일 FD가 Enable 되었을 때 두 경우가 같은 비율로 발생한다면, Enable되는 FD선마다 평균 1 개의 OT가 발생한다. 따라서 k_a 개의 보조선들을 동수의 FD선으로 단순히 대치할 경우, 매 전송주기 마다 k_a 개의 OT가 발생하게 되어 보조선을 사용하는 것에 비해 약 2 배의 OT를 발생시키게 된다.

위와 같이, 보조선을 FD선으로 단순히 일대일로 대치할 경우 보조선을 사용할 때보다 오히려 OT의 발생이 증가한다. 이런 이유로 본 논문에서는 보조선을 사용할 때 보다 OT의 발생을 적게 만들기 위해서 다음과 같은 새로운 방법을 개발하였다.

【 FD를 이용한 코딩정보 전송방법 】

1. 모든 버스선들을 FD선으로 대치한다.
2. 버스폭을 w 라고하면, 각 선을 0부터 w-1 까지 번호를 붙인다.
3. $0 \le ix < w$ 인 값을 전송하고자 하면, ix번 버스선의 FD 만을 Enable시키고 다른 FD는 Disable 시킨다. 수신단에서는 플립신호가 발생한 FD선의 위치로써 ix의 값을 알 수 있다.
4. 데이터풀의 크기 m 이 버스폭 (w) 보다 클 경우, $w \le ix < m$ 의 값에 대해서는 복수의 FD선을 통하여 ix 값을 전송한다. $w \le ix < m$ 의 값에 따라 Enable되는 FD선들의 위치는 데이터 특성에 따라 OT가 최소가 되도록 미리 결정된다.

예를 들면, 데이터풀의 크기가 8 일 때 ix=5의 값을 전달하기 위해서는 보조선을 사용할 경우 3개의 보조선, $A_0A_1A_2$를 사용하여 $A_0A_1A_2$=101_2 을 전송해야 하지만 위 방법을 사용하면 5 번 FD 하나만을 Enable 시키면 된다. 일반적으로 데이터풀의 크기가 m 일 때 Enable 해야 되는 FD선의 최대 개수 k_F는 다음 식을 만족하는 최소의 k_F 값을 구하여 얻을 수 있다.

$$m \le \sum_{i=1}^{k_F} {}_wC_i \quad (5)$$

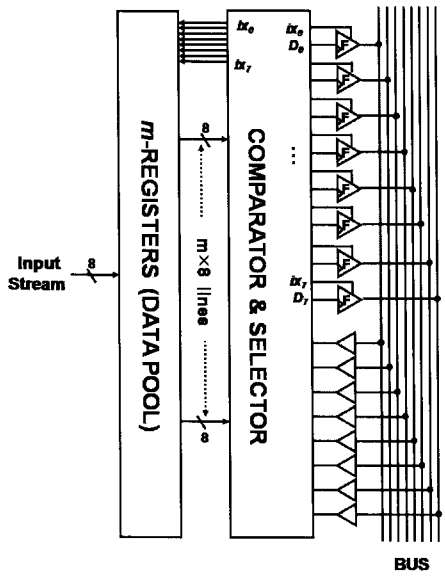
이 식을 적용하면 $m \le w$ 이면 하나의 FD선 만 Enable 하여 코딩정보의 전송이 가능하며 2 개의 FD로는 8-비트 버스에서는 m=36 까지, 16-비트 버스에서는 m=136 까지 가능하다.

식 (4)와 (5)를 비교하면 동일한 m (>2) 값에 대하여 활성화되는 FD선의 수는 보조선의 수에 비해 1/2 이하이다. FD선이 보조선에 비해 2 배의 OT를 발생하지만, 위의 방법을 사용하면 활성화되는 FD선의 수가 보조선 개수의 1/2 이하이므로 보조선 방식에 비해 OT을 감소시킬 수 있다.

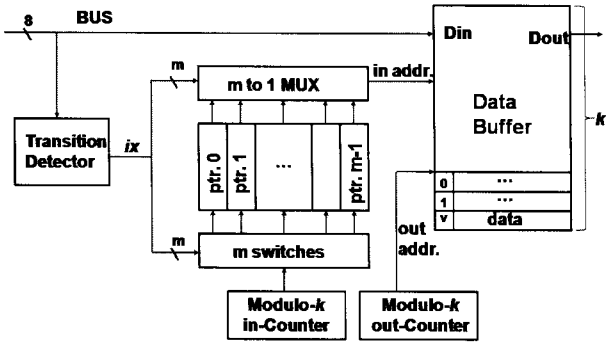
Ⅲ. Extended Lagger Algorithm 회로의 구현

EL-m 알고리즘을 그림 5와 같은 회로로 구현하였다. 그림 5-(a)는 8-bit 버스용 EL-m의 인코더 (Encoder)를 구현한 것이다. 데이터풀은 m 개의 8-비트 레지스터로 구성된다. 비교선택기(Comparator & selector)는 데이터풀 안의 모든 데이터 값과 현재의 버스값을 입력으로 받아, 식 (1)을 적용하여 각 데이터에 대해 전이수를 계산하고 이 값이 제일 작은 데이터를 선택한다. 그러면 선택된 데이터의 레지스터 번호 (k) 에 해당하는 ix_k는 1로 셋팅되고 그 이외의 ix_j 들은 0이 된다. ix_k는 선택회로 (멀티플렉서)의 입력으로도 사용되어 해당 레지스터의 데이터가 FD의 입력으로 전달된다. 전송이 끝난 레지스터는 입력데이터열 (Input stream)의 다음 데이터로 채워진다.

그림 5-(b)는 5-(a)의 인코더와 짝을 이루는 디코더 (Decoder)의 구조이다. 수신된 데이터의 순서를 복원하기 위해 데이터버퍼가 사용된다. 수신된 데이터를 포인터가 지시하는 주소에 따라 버퍼에 삽입한 후 버퍼에 저장된 데이터를 주소 0 번부터 차례로 출력함으로써



(a) Encoder



(b) Decoder

그림 5. Extended Lagger Algorithm의 회로 설계
Fig. 5. The circuit design for Extended Lagger Algorithm.

원래의 순서를 복원한다. 버퍼는 여러 개의 슬롯(Slot)으로 구성되어 있고, 각 슬롯은 valid-비트와 데이터 필드로 구성된다. 데이터 필드에는 수신된 데이터를 저장하며, valid-비트는 데이터필드에 저장된 값이 유효한 값인가를 나타낸다. 포인터 레지스터 (pointer register)는 0 번부터 m-1 번까지 m 개의 포인터로 구성되어 있으며, j 번째 포인터는 ix_j 값이 감지되었을 때 수신된 데이터를 저장할 주소가 저장된다.

데이터버퍼의 입력주소와 출력주소를 얻기 위해 두 개의 모듈로-k 카운터가 사용되었다. 입력카운터 (in-counter)는 초기 값이 m 으로 설정되고 포인터들은 순서대로 0 부터 m-1 까지의 값으로 각각 초기화 된다. 데이터 수신이 시작되어 ix_j 값이 감지되면 수신된 데이터는 ptr.-j 가 가리키는 주소에 저장된다. 저장이 완료되면 ptr.-j는 입력카운터의 값으로 업데이트 되고 입력카운터는 증가된다. 출력카운터 (out-counter)는 초기 값이 0으로 설정되며, 수신이 시작되면 매 클럭마다 출력카운터가 가리키는 주소의 valid-비트를 검사하여 v=1 이면 데이터를 출력하고 valid-비트를 0으로 리셋시킨다. 데이터가 출력되면 출력카운터의 값을 증가시킨다. 만일 v=0 상태이면, 출력은 일어나지 않으며 카운터의 값도 증가되지 않는다. 이 과정을 매 클럭마다 반복하면 데이터버퍼에서 출력된 데이터열은 원래의 순서로 복원된 데이터열이 된다.

표 1. 실험에 사용된 파일
Table 1. The benchmark files used for the experiments.

기호	형식	설 명
RN	BIN	랜덤하게 생성된 Binary 파일
IM	JPG	스위스 산의 풍경 이미지
NB	MP3	노래, Nobody (원더걸스)
IL	WMA	노래, 나 가거든(조수미)
TR	MP4	영화, 트랜스포머-2의 예고편(Trailer)

Ⅳ. 실 험

새로운 구현방법이 버스선의 전이감소에 얼마나 효율적인지를 알아보기 위하여 시뮬레이션을 통한 실험을 수행하였다. 기존방식과의 비교를 위하여 EL-m 알고리즘을 보조선을 사용하여 구현한 방식 (AUX)과 FD를 사용하여 구현한 새로운 방식 (FD)의 두 가지로 구현하고 동일한 데이터 스트림을 전송하면서 버스선의 전이횟수를 측정, 비교하여 그 효용성을 검토하였다.

실험을 위하여 인터넷 및 멀티미디어 분야에 많이 사용되는 파일형식들에 대해 여러 개의 파일들을 선택하여 벤치마크로 사용하였다. 본 논문에서는 간결성을 위하여 표 1과 같이 5 개의 파일형식에 대해 각 하나씩의 파일만을 선택하여 그 결과를 나타내었다. 실험은 8-bit 버스에 대하여 수행하였으며 다음과 같이 진행하였다.

1. 위의 파일들을 코딩을 사용하지 않고 원래의 순서대로 전송하고 (Raw Transmission) 버스선에서 발생하는 전이 수, T_R 을 측정한다.
2. 동일한 파일을 보조선 (AUX) 방식으로 구현한 회로를 통해 전송하면서 발생하는 전이수를 버스선에서 발생하는 전이수, $T_{D,t}$ 와 보조선에서 발

생하는 오버헤드전이수, $T_{OT,t}$ 를 측정한다.

3. 동일한 파일을 새로운 FD 방식으로 구현한 회로를 통해 전송하면서 버스선에서 발생하는 전이수 T_t 를 $T_{D,t}$ 와 $T_{OT,t}$로 분리하여 측정한다.
4. 데이터풀의 크기를 2 에서 8 까지 바꾸어 가며 각 크기마다 2, 3의 과정을 반복한다.

위의 과정을 통해 얻은 값들을 비교하여 EL-m 알고리즘의 효율을 측정하고 AUX 방식과 FD 방식을 비교하였다. 그림 6 에는 IM(JPG) 파일에 대한 결과를 도표를 사용하여 나타내었다. 다른 형식의 파일들도 모두 같은 경향을 나타내므로 대표적으로 하나만을 인용하였다. 표 2 에는 실험에서 얻은 결과들을 정리하였다. 각 파일형식 사이의 비교를 용이하게 하기 위하여 모든 측정 값들은 T_R 로 나눈 규격화된 값 (Normalized value), R_D (=$T_{D,t}$ /T_R), R_{OT} (=$T_{OT,t}$ /T_R), R_t (=T_t /T_R) 을 사용하였다. U_{OT} (=$T_{OT,t}$ /N)는 전체 OT 수를 데이터크기 (또는 전송횟수)로 나눈 값으로써 매 전송주기마다 발생하는 평균 OT 개수를 의미한다.

그림 6과 표-2의 실험 결과를 살펴보면 파일의 형식에 따라 다소 정도의 차이가 있지만 모두 다음과 같은 공통점을 갖는다.

1. 두 방식 모두 데이터풀의 크기, m이 커질수록 전체 전이횟수가 감소한다
2. m=2 이외의 모든 경우에 FD 방식이 AUX 방식보다 R_t 가 작다. 즉 FD 방식이 버스선의 전이를 감소시키는데 더욱 효율적이다.
3. 데이터풀의 크기에 상관없이 R_D는 AUX 방식이 FD 방식보다 작은 반면, R_{OT}는 FD 보다 크다.
4. FD 방식의 R_{OT}는 크기가 거의 비슷한 데 반하여 AUX 방식은 m 의 크기 즉, 보조선의 수에 따라 증가한다.

위의 결과를 분석해 보면 먼저, 예상한 바와 같이 데이터풀의 크기가 커질수록 EL-m 알고리즘의 효율은증가한다. 데이터풀의 크기가 2 에서 8 로 증가함에 따라 AUX 방식의 전이횟수 감소효율은 평균 9.5%에서 21.5%로 약 2.3 배 증가하였으며, FD 방식은 평균 9.0%에서 27.6%로 약 3 배로 증가하였다. FD 방식이 AUX 방식에 비해 전체효율 및 효율의 증가율도 높으므로 새로 개발된 방식이 버스의 전이회수 감소에 더욱 효율적

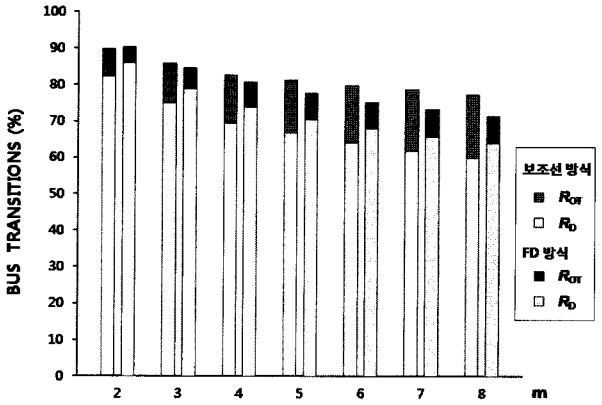


그림 6. 데이터풀의 크기에 따른 버스전이횟수의 변화
Fig. 6. Variation of the number of bus-transitions along the size of data-pool.

임이 입증되었다.

흥미로운 점은 표 2와 그림 6에서 볼 수 있는 바와 같이 전체 전이수는 FD 방식이 적은데 반하여, 데이터 전송을 위한 버스선의 전이(R_D)는 AUX 방식이 FD 방식에 비해 작다는 것이다. 이것은 FD 방식이 AUX 방식보다 OT 발생에 대한 부담이 더 크기 때문에 OT를 발생하지 않는 데이터가 경쟁에 이길 확률이 더 커지기 때문이다. 즉 AUX 방식에서는 OT를 발생하는 데이터가 그렇지 않은 데이터보다 전이수 한 개의 부담을 더 갖게 되지만, FD 방식에서는 전이수 두 개의 부담을 갖게 된다. 8-bit 버스에서 2 개의 전이 차는 매우 크므로 OT를 발생시키는 데이터가 그렇지 않은 데이터에 비해 선택될 확률이 매우 낮아진다. 이런 이유로 AUX 방식에서는 버스선의 전이를 감소시키는 데이터가 상대적으로 많이 선택되는 반면, OT의 부담이 큰 FD 방식은 버스선의 전이 감소가 적더라도 OT를 발생하지 않는 데이터가 선택되는 경우가 많기 때문이다. 결과적으로 AUX 방식의 경우 데이터전송을 위한 버스선의 전이수는 작지만, 보조선에서 발생하는 OT가 버스선에서의 감소분을 많이 상쇄시키기 때문에 전체 효율이 떨어지게 된다. 반면 FD 방식은 데이터 전송을 위한 버스선의 전이 감소는 AUX 방식에 비해 떨어지지만 OT의 발생이 작기 때문에 전체적인 알고리즘의 효율은 AUX 방식에 비해 높아진다.

표 2에서 보는 바와 같이 데이터 전송 시 평균 OT발생 개수, U_{OT}는 AUX의 경우 보조선이 증가함에 따라 점점 증가한다, 그러나 FD 방식은 $2 \le m \le 8$ 에서는 하나의 FD선만 활성화 되므로 U_{OT}가 약 0.3 개 정도로 거의 일정함을 볼 수 있다. 보조선당 U_{OT}는 AUX 방식

표 2. 실험 결과
Table 2. Result of the experiments.

파일기호 [형식]	원 전이회수 (T_R) [데이터크기 (Byte)]	구현 방식	측정값	데이터 풀의 크기 (m)						
				2	3	4	5	6	7	8
RN [BIN]	16,437 [4,096]	AUX	R_D (%)	83.56	75.85	70.45	67.87	64.85	62.51	61.19
			R_{OT} (%)	7.17	10.48	12.87	14.08	15.66	16.56	17.31
			R_t (%)	90.73	86.33	83.32	81.95	80.51	79.07	78.50
			U_{OT} (개)	0.29	0.42	0.52	0.56	0.63	0.66	0.69
		FD	R_D (%)	86.55	78.80	74.37	71.67	68.75	66.79	64.54
			R_{OT} (%)	4.17	5.48	6.75	6.58	6.86	6.95	7.34
			R_t (%)	90.72	84.28	81.12	78.25	75.61	73.74	71.88
			U_{OT} (개)	0.17	0.22	0.27	0.26	0.28	0.28	0.29
IM [JPG]	1,469,263 [367,712]	AUX	R_D (%)	82.39	75.09	69.57	66.80	64.09	61.84	59.99
			R_{OT} (%)	7.40	10.94	13.10	14.70	15.86	16.86	17.49
			R_t (%)	89.79	86.03	82.67	81.50	79.95	78.70	77.48
			U_{OT} (개)	0.30	0.44	0.52	0.59	0.63	0.67	0.70
		FD	R_D (%)	86.11	78.84	74.02	70.59	67.92	65.69	63.90
			R_{OT} (%)	4.23	5.88	6.79	7.20	7.46	7.59	7.62
			R_t (%)	90.34	84.72	80.81	77.79	75.38	73.28	71.52
			U_{OT} (개)	0.17	0.24	0.27	0.29	0.30	0.30	0.30
NB [MP3]	13,088,884 [3,417,571]	AUX	R_D (%)	83.78	76.74	71.37	68.57	65.91	63.78	61.85
			R_{OT} (%)	7.30	10.87	13.11	14.71	15.92	16.90	17.68
			R_t (%)	91.08	87.61	84.48	83.28	81.83	80.68	79.53
			U_{OT} (개)	0.28	0.42	0.50	0.56	0.61	0.65	0.68
		FD	R_D (%)	87.51	80.61	75.97	72.58	69.91	67.72	65.92
			R_{OT} (%)	4.18	5.91	6.78	7.28	7.54	7.72	7.81
			R_t (%)	91.69	86.52	82.75	79.86	77.45	75.44	73.73
			U_{OT} (개)	0.16	0.23	0.26	0.28	0.29	0.30	0.30
IL [WMA]	7,958,352 [2,028,608]	AUX	R_D (%)	83.03	75.84	70.42	67.57	64.90	62.72	60.81
			R_{OT} (%)	7.29	10.80	13.00	14.56	15.71	16.71	17.43
			R_t (%)	90.32	86.63	83.42	82.13	80.61	79.43	78.24
			U_{OT} (개)	0.29	0.42	0.51	0.57	0.62	0.66	0.68
		FD	R_D (%)	86.66	79.54	74.80	71.37	68.62	66.45	64.63
			R_{OT} (%)	4.22	5.90	6.76	7.19	7.48	7.63	7.72
			R_t (%)	90.88	85.44	81.56	78.56	76.10	74.08	72.35
			U_{OT} (개)	0.17	0.23	0.27	0.28	0.29	0.30	0.30
TR [MP4]	119,979,248 [30,627,248]	AUX	R_D (%)	83.26	76.13	70.72	67.91	65.24	63.09	61.13
			R_{OT} (%)	7.27	10.80	13.02	14.59	15.77	16.77	17.51
			R_t (%)	90.53	86.93	83.74	82.50	81.01	79.86	78.64
			U_{OT} (개)	0.29	0.42	0.51	0.57	0.62	0.66	0.69
		FD	R_D (%)	86.91	79.89	75.22	71.79	69.11	66.93	65.11
			R_{OT} (%)	4.13	5.81	6.65	7.12	7.40	7.54	7.62
			R_t (%)	91.05	85.70	81.87	78.91	76.50	74.48	72.73
			U_{OT} (개)	0.16	0.23	0.26	0.28	0.29	0.30	0.30

에서는 약 0.22~0.28 개로 FD 방식의 0.3 개 보다 작다. 그러나 전송선 (보조선 또는 FD선)이 활성화 되었을 때, FD 방식의 평균 OT 수가 AUX 방식의 2 배인 것을 감안하며 실험값의 차이는 매우 작은 편이다. 그 이유는 앞에서 설명한 바와 같이, FD 방식이 AUX 방식보다 OT 발생에 대한 부담이 더 크기 때문에 OT를 발생하는 데이터가 경쟁에 이길 확률이 AUX 방식보다 적어지기 때문이라고 판단된다.

하나의 보조선이 사용되는 m=2 인 경우는 두 방식의 효율이 비슷하거나 AUX 방식이 FD 방식에 비해 근소하게 효율이 높다. 그 이유는 코딩정보의 전송에 사용되는 전송선(보조선 또는 FD선)의 수가 같을 때에는 AUX 방식이 FD 방식에 비해 OT 발생량이 적기 때문이다.

위의 실험결과로 새로운 방식이 기존의 보조선을 사용한 방식에 비해 데이터전송 시 발생하는 버스선의 전이횟수를 감소시키는데 더욱 효과적인 것을 확인할 수 있었다. 그 주요원인은 오버헤드전이의 발생이 적기 때문으로, 오버헤드전이의 발생횟수가 전체 효율에 많은 영향을 미치는 것을 알 수 있다. FD선을 사용한 코딩정보 전송방식을 적용하여 오버헤드전이의 발생을 감소시키는 것만으로도 기존의 보조선방식에 비해 알고리즘의 효율이 30% 증가하였다. 이것은 SSC 알고리즘의 개발에는 오버헤드전이의 발생을 억제할 수 있는 코딩정보 전송방법의 개발이 매우 중요함을 보여준다.

V. 결 론

SSC 알고리즘의 설계에서 코딩정보의 전송을 위하여 전통적으로 사용되어왔던 보조선을 추가하는 방식은 오버헤드전이를 많이 발생시켜 전체효율을 떨어뜨리고 결과적으로 SSC 알고리즘의 확장성을 방해하는 중요한 요인이 되는 것을 실험을 통하여 확인하였다. 본 논문에서는 보조선을 사용하지 않고, Flip-Driver를 이용하여 버스선을 통하여 코딩정보를 전송하는 방법과 함께 오버헤드전이의 발생을 감소시키는 새로운 전송방법을 개발하였다. 이 방식을 적용하여 Extended Lagger Algorithm을 구현할 경우 오버헤드전이의 증가 없이 데이터풀의 크기를 버스폭과 같은 크기까지 증가시킬 수 있다. 새로운 방식을 적용한 Extended Lagger 알고리즘을 실험을 통하여 검증한 결과 기존의 보조선 추가 방식에 비하여 오버헤드전이의 발생이 50% 이하로 줄어들게 되어, 결과적으로 알고리즘의 효율이 약 30% 향상되는 효과를 얻을 수 있었다.

효율적인 Sequence Switch Coding 알고리즘의 개발을 위해서는 알고리즘 자체의 효율성뿐만 아니라 코딩정보를 전송하는 방법의 효율성도 같이 고려해야 하며, 오버헤드전이를 적게 발생시키는 코딩정보 전송방법의 개발이 알고리즘의 확장성을 위하여 매우 중요한 요소임을 확인하였다.

참 고 문 헌

[1] Y. Nakagome, K, Itoh, M. Isoda, K, Takeuchi, and M. Aoki, "Sub-1-V Swing Internal Bus Architecture for Future Low-Power ULSI's," IEEE Journal of Solid State Circuits, vol. 28, no.4, pp. 414-419, Apr. 1993.

[2] H. Zhang, V. George, and J. M. Rabaey, "Low swing on-chip signaling techniques: effectiveness and robustness," IEEE Trans. VLSI Syst. vol. 8, no. 3 pp. 264-272, June 2000.

[3] C. L. Su, C. Y. Tsui, and A. M. Despain, "Low power architecture design and compilation technique for high-performance processors," in Proc. IEEE COMPCON, Feb. 1994, pp. 209214.

[4] L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C Silvano, "Asymptotic zero-transition activity encoding for address busses in low -power microprocessor-based systems,"in Proc. Great Lakes Sysmp. VLSI, Mar. 1997, pp. 7782.

[5] L. Benini, G. De Micheli, E. Macii, M. Poncino, and S. Quer, "System-level power optimization of special purpose applications: the beach solution," in *Proc. Int. Symp. Low Power Electronics Design*, pp. 24 - 29, Aug. 1997.

[6] 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.

[7] M. Yoon, "Sequence-Switch Coding for Low-Power Data Transmission", IEEE Trans. VLSI Syst., vol. 12, pp. 1381-1385, Dec. 2004.

[8] 윤명철, "패킷형 데이터를 위한 저전력 전송방법," 전자공학회논문지, 제41권 SD편, 제7호, 71-79쪽, 2004년 7월

[9] M. Yoon and B. Roh, "A Novel Low-Power Bus Design for Bus-Invert Coding," IEICE Trans. Electronics, vol. E90-C, no. 4, pp. 731-734, 2007.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, New York, McGraw-Hill Book Company, 1992.

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 교수

<주관심분야: VLSI/SoC 설계, Embedded System, Digital System 설계>